

The impact of inheritance depth on maintenance tasks – Detailed description and evaluation of two experiment replications

Barbara Unger, Lutz Prechelt, Michael Philippsen
(unger, prechelt, phlipp@ira.uka.de)

Fakultät für Informatik
Universität Karlsruhe
D-76128 Karlsruhe, Germany

Technical Report 18/1998

September 2, 1998

Abstract

Inheritance is one of the main concepts of object-oriented technology. It is claimed that the use of inheritance improves productivity and decreases development time.

John Daly et al. reported on two experiments evaluating the effects of inheritance depth on program maintenance. They found that maintenance was performed significantly quicker for software using three levels of inheritance, compared to equivalent ‘flattened’ software without inheritance. A second experiment found that maintenance for software using five levels of inheritance tended to be slightly slower than for equivalent software without inheritance.

We report on similar experiments on the same question. Our results contradict those mentioned above. Several crucial changes were made to the setup. In particular longer and more complex programs were used, an inheritance diagram was available to the subjects, and we used more and different kinds of maintenance tasks. Furthermore, our experiment design compares zero level, three level and five level inheritance directly in one experiment.

The results suggest that there is a tendency that deeper inheritance may complicate program understanding. But the effect depends rather on other factors such as complexity of the program and type of maintenance task than on inheritance depth. We found a high correlation between maintenance time and the number of methods to trace to gain program understanding. Further work should be done to identify other influence factors.

Contents

1	Introduction	4
1.1	Experiment introduction and overview	4
1.2	Related work	6
2	Description of the experiment	7
2.1	Experiment purpose and hypotheses	7
2.1.1	Our hypotheses	7
2.2	Experiment design	8
2.2.1	Subject groups	8
2.2.2	Preparation: The JAKK course	9
2.2.3	Preparation: The Informatik II course	10
2.3	Experiment format and conduct	10
2.4	Experimental subjects JAKK	11
2.4.1	Education	11
2.4.2	Programming experience	11
2.5	Experimental subjects Informatik II	14
2.5.1	Education	14
2.5.2	Programming experience	14
2.6	Knowledge of Java, inheritance, and polymorphism	17
2.7	Constraints of the experiment	17
2.8	Programs and Tasks	19
2.8.1	Programs	19
2.8.2	Tasks	20
2.8.3	Solving the tasks	23
2.9	Internal validity	26
2.10	External validity	26
3	Experiment results and discussion	28
3.1	Statistical methods	28
3.2	Performance on the tasks	28
3.2.1	Metrics employed	29
3.2.2	Notation	29
3.2.3	Performance on the tasks – JAKK	30
3.2.4	Performance on the tasks - Informatik II	35
3.3	Subjects' experience	43
3.3.1	Inheritance use	43
3.3.2	Program structure	43

3.3.3	Simplicity of the tasks	44
3.3.4	Concentration of the subjects	44
3.3.5	Quality assessment	47
3.3.6	Usefulness of OO knowledge	47
4	Analysis of the different results of the previous and our new experiment	50
5	Conclusion	53
A	Tasks and Solutions	55
A.1	Handling descriptions	55
A.2	Original questionnaire (translated into English)	56
A.3	Solutions	68
A.3.1	Solution for Task 1 – "Y2K"	68
A.3.2	Solution for Task 2 - "Time Interval Price Display and Gain/Loss Display"	68
B	Programs	71
B.1	Programs – JAKK	71
B.1.1	Boerse.java (no inheritance)	71
B.1.2	Boerse.java (3 levels of inheritance)	111
B.1.3	Boerse.java (5 levels of inheritance)	133
B.2	OMT diagrams – JAKK	152
B.3	Program differences between JAKK and Informatik II experiments	156
B.3.1	Boerse.java (no inheritance)	156
B.3.2	Boerse.java (3 levels of inheritance)	167
B.3.3	Boerse.java (5 levels of inheritance)	171
C	Translations of German terms	175
	Bibliography	177

Chapter 1

Introduction

The present report is the detailed description and evaluation of two instances of a controlled experiment on the influence of inheritance depth on the maintainability of object-oriented programs.

In this first chapter we will discuss the general topic of the experiment, give a broad overview of the purpose and setup of the experiment, and refer to related work.

The second chapter describes the hypotheses, preparation, setup, and conduct of the experiment. It also discusses possible threats to the internal validity and external generalizability of the experiment. Chapter 3 presents and interprets in detail the results obtained in the experiment and Chapter 4 explains the differences between the related work and our results. Chapter 5 presents the conclusions.

The appendix contains the handouts used in the experiment: questionnaires, task descriptions, and program source code. Furthermore it describes correct solutions for the tasks. The program source is written in German, but an additional appendix provides German to English translations for the program's main terms.

1.1 Experiment introduction and overview

Object-orientation was expected to be a silver bullet for a long time. Vendors made promises that object-orientation enhances productivity, reliability, reuse, life cycle time, etc. In [9] the claims about object orientation are summarized:

- Methodological advantages of object-orientation:
 - natural imitation of the application domain by object-oriented analysis
 - smooth transition from object-oriented analysis to object-oriented design
 - consistent terminology from analysis to implementation
- General advantages of object-orientation:
 - higher productivity and shorter development time through inheritance and reuse
 - higher quality of the products through encapsulation and modularization
 - less effort for maintenance

Much of the general advantages described above is attributed to inheritance: Inheritance is expected to reduce overall code size because functionality that is common to more than one class needs only be defined once. Classes can be viewed as implementations of abstract data types, providing an interface. The functionality can be understood in terms of the interface. Understanding derived classes is eased because they are built on the abstractions implemented by their respective superclasses. Inheritance allows reuse of the classes' implementation. If the superclass is trusted to be correct, procedures that are inherited completely from the superclass need not be tested again. Therefore derived classes are less prone to errors.

These effects are expected to multiply if a class is inherited from more than once. The described advantages lead to the assumption that using more abstraction levels the maintenance of a program will be easier and that the quality of maintenance tasks will be higher.

John Daly et al. [4] conducted an experiment to evaluate the influence of inheritance on maintainability. They tested the hypothesis that “the use of a hierarchy of 3 levels of inheritance depth does positively affect the maintainability of object-oriented programs”¹. Subjects had to implement identical maintenance tasks on object-oriented programs with a hierarchy of 3 levels of inheritance depth and on functionally equivalent programs with no inheritance. The results indicated that inheritance depth does affect the maintainability such that maintaining a program with no inheritance takes significantly more time than maintaining a program with inheritance depth of 3. A replication of the same experiment had the same results.

Daly et al. also performed a similar experiment with a modified hypothesis derived from the results of the questionnaire survey on the object-oriented paradigm mentioned in Section 1.2. They tested whether “the use of a hierarchy of 5 levels of inheritance depth does affect the maintainability of object-oriented programs such that subjects maintaining the inheritance program version will take longer than those subjects maintaining the flat program version” [4]. The same (!) 29 subjects (except for two) as in the first experiment were used for this experiment. The results obtained did not show significant differences between the program versions.

Our criticism of these experiments is that they used rather small and simple programs. Details on the programs size are listed in the table below. Lines of code (*LOC*) are here defined as all lines of all header files and source files except for leading or trailing blank lines in a file. “{”- or ”}”-lines are all lines containing only one brace, *comment lines* are lines of code containing only a comment, and *include lines* are lines of code containing an include statement. Programs *university-1* and *literature* were used in the two parts of the first experiment (1a and 1b, *university* also in the replication 1r) and *university-2*, an extended version of *university-1* was used in the second experiment. All classes of the programs were quite simple. For example in the *university* programs, each class represented a person group (such as professor, student, secretary) with almost the same class variables (such as last name, first name, age), and similar methods (for assigning values to the variables and for printing the object).

When we first saw the experiment of Daly et al., we hypothesized that more abstraction levels would universally make maintenance easier, if only the subjects were given a class hierarchy diagram. We decided to perform a similar experiment to test this assumption.

We implemented three Java program versions with identical functionality that differed only in the number of abstraction levels (5, 3, and 0 abstraction levels) and created appropriate class hierarchy diagrams.

Subjects of two university courses participated in the experiment, 57 computer science students from a graduate course and 58 computer science students from an undergraduate course. Each course was randomly divided into three groups. The sampling was stratified for balancing experience among the groups. All subjects of one experiment group received the same program version, the appropriate class hierarchy diagram, and two

¹For a detailed definition of inheritance depth, see 2.8.1

program	levels of inheritance	Number of classes	Number of methods	LOC	”{- or }” lines	Number of comment lines	Number of include lines
university-1	3	5	21	252	52	29	22
university-1	0	3	26	273	58	16	14
literature	3	6	27	323	60	30	26
literature	0	4	35	370	78	21	19
university-2	5	11	56	694	136	78	56
university-2	0	8	96	1007	208	59	41

Table 1.1: Characterization of the programs used by Daly et al.

maintenance tasks. For each subject we measured the time required for completing each task and we rated the delivered solution. We tested whether and how the performance of the three groups differs.

1.2 Related work

A good summary of the related work can be found in the article of John Daly et al. [4] so that we do not reiterate all the references they cite. But one reference we emphasize: a structured interview study on the object-oriented paradigm [3]. These structured interviews, each based on a template of 23 questions, were carried out with 13 experienced object-orientation users to explore problems discussed in the object-oriented literature. For example “Subjects were asked to define how many levels of inheritance it took before their understanding began to become constrained”. Half of the subject indicated a depth of inheritance of 3-4, a quarter of the subjects indicated an inheritance depth of 5-6 and another quarter more than 6 or “as not relevant”. Another topic with respect to inheritance is the delocalization of functionality. “On the effect of spreading method functionality over the inheritance hierarchy on system understanding almost every subject commented that if the inheritance hierarchy is designed properly than it would not be detrimental to understanding” [3]. These structured interviews were an initial study that was followed by a larger questionnaire survey of experience with the object-oriented paradigm [2]. This study also contained the question “ How deep would your inheritance hierarchy be before you became uncomfortable with it?” The answer scale for the subjects was a little different from the structured interviews but the result indicates that the subjects dare more: 12% answered an inheritance depth of 2-3, 31% a depth of 4-6 and 57% a depth more than 6 or even did not see problems with inheritance at all.

Chapter 2

Description of the experiment

2.1 Experiment purpose and hypotheses

Genericity, overloading, inheritance, polymorphism, and encapsulation are the main components of the object-oriented concept. Until now it is an open question how and to which extent these features have to be employed to get better results: better in the sense of shorter development time, higher product quality, cheaper maintenance, etc.

The purpose of this experiment is to answer a small part of the research question: When is the use of inheritance beneficial? Does it matter how deep the inheritance hierarchy is? Does it matter whether programmers have an inheritance diagram where they can trace method and class definitions?

In this experiment we want to test whether the inheritance depth significantly affects the maintainability of programs. The maintainability will be measured by the time required to complete specific maintenance tasks and by the quality of the solutions delivered.

2.1.1 Our hypotheses

We pose the following hypotheses:

1. **Hypothesis 1:** The more abstraction levels are used, the easier the program is to comprehend and the less time is required for maintenance tasks.
2. **Hypothesis 2:** The more abstraction levels are used, the better is the quality of the delivered solutions.

Explanation of the hypotheses:

Hypothesis 1: Using inheritance provides the maintainer with the information that classes inheriting from a superclass have the same functionality as the superclass except for the methods that are overwritten. So it is not necessary to comprehend lines of code twice or detect that there are duplicates. On the other hand, this effect might be reduced by the dispersal of methods declared in more than one superclass. So the maintainer has to switch between different classes to understand what the functionality of a class is. To reduce this impact, we offer class hierarchy diagrams in the experiment.

Another aspect of this hypothesis is the required time for *implementing* changes when more than one class is affected. Adding common functionality to more than one class can be done by adding and testing the functionality once in a superclass, hence maintenance time can be reduced.

Hypothesis 2: First, when using more code from a superclass, less places to insert defects will be given to a maintainer maintaining the appropriate subclass. A smaller part of the functionality is displayed to the maintainer where s/he can make changes and introduce defects when some functionality is provided in a superclass that needs not be touched. This is important in particular if the maintenance tasks only affect concrete classes from which no other classes inherit. Similarly, if a maintainer needs to make changes that affect more than one class, s/he can often implement the changes just once in a common superclass.

For these hypotheses we assume that the design is sound, i.e., each abstraction level in the inheritance hierarchy represents a sensible and useful concept of the program.

2.2 Experiment design

We designed a basic program version with a deep inheritance hierarchy and derived two functionally equivalent versions from it, differing in the inheritance depth. Functionality, programming style, and comments are essentially the same in all three versions. This is explained in detail in Section 2.8.1.

The subjects are assigned into three groups called G0, G3 and G5, each subject works on one of these program versions. We measure the time required by each individual subject and judge their solutions by both a black box test and an examination of the source program.

The independent variable in this experiment is the inheritance depth. The dependent variables are the time required for each task, the quality of the solutions, the approach the subjects chose and additional subjective information from a postmortem questionnaire.

We also administer a short questionnaire (immediately *before* the actual experiment) for gathering statistical information about our subjects, and for assessing their understanding of object-orientation in general and Java in particular.

2.2.1 Subject groups

We conducted the experiment two times. The first run was with students from our JAKK course (see below). The next run (done two parts) was with students from the Informatik II course.

To make sure to have three comparable groups for each run, we used block-randomization. For the JAKK course subjects, we blocked the subjects based on their pre-performance. The pre-performance was determined as the sum of points the students obtained in the five exercises of the JAKK course. They could take part in the experiment if they achieved more than 75 percent of all points. The motivation to take part in the experiment was to receive lab course credit. In the Informatik II course, we asked the subjects to tell us about their programming experience instead, measured in terms of the size of the biggest program they had ever implemented. All students of the Informatik II course were allowed to take part in the experiment, one third of them did. Their motivation was to obtain a small bonus on their exam grade.

The three treatment groups are called G0 for the group with the flat program, G3 for the group with the program of inheritance depth 3 and G5 for the group with the program of inheritance depth 5. The suffixes I and

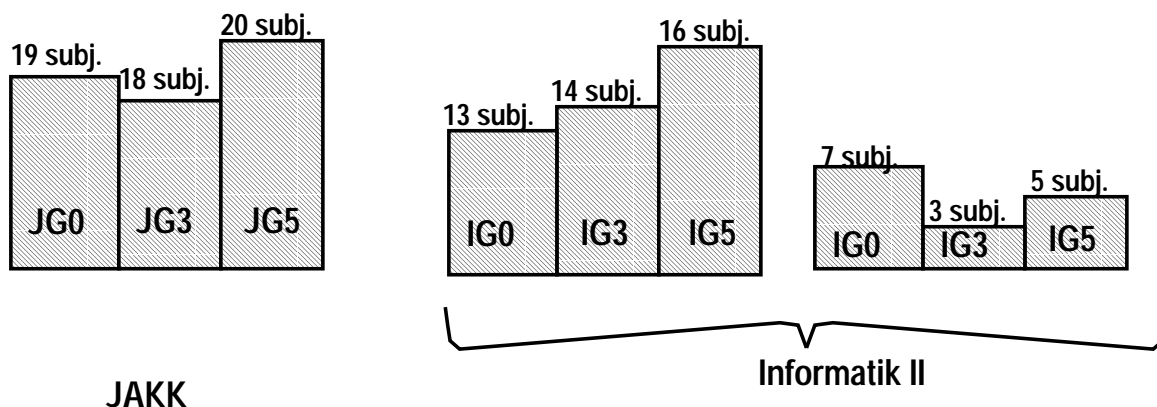


Figure 2.1: Group size and group name for the JAKK and Informatik II subjects. The slight imbalance is due to no-show participants, who registered, but did not appear.

J describe the course: J for the JAKK course and I for the Informatik II course. For the group sizes, see Figure 2.1.

2.2.2 Preparation: The JAKK course

In order to have enough subjects with sufficient knowledge and comparable background, we taught a course, called JAKK (“Java/AWT-Kompaktkurs”), to prepare our subjects. The topic of the course were Java and AWT (Abstract Windowing Toolkit), which attracted over 70 highly motivated graduate students. Participants were required to have at least basic knowledge of OO techniques.¹ It was a five week intensive course with a 90 minute lecture and one exercise per week. The results of the exercises were submitted on disk and were graded by the instructors.

We gave two lectures on Java itself, one on AWT, one on thread programming, and one for discussing frequent errors etc. The course exercises were chosen as follows:

Exercise 1 was to implement a clone of the Unix *wc* program. This exercise was meant for learning how to install and operate the compiler and learning basic Java syntax, file I/O etc.

Exercise 2 introduced reuse of functionality. The exercise was to implement a clone of the Unix *ls* program. Students had to reuse and enhance a given class, and they had to add new classes on their own.

Exercise 3 was on programming with AWT. The task was to display a graphical directory viewer somewhat like the Windows file manager. The students had to reuse their classes from Exercise 2.

Exercise 4 was on programming with threads. The task was to concurrently search files for lines whose acronyms matched the search string or words that were anagrams of the search string. The algorithms were to be implemented using the Template Method design pattern. The number of concurrent threads was to be limited in either of two user-selective ways, using the Strategy design pattern to make the selection transparent. We required all concurrent activity to be properly synchronized and free of race conditions.

Exercise 5 was a simple graphical game. It introduced mouse handling and animation.

¹We use OO as the abbreviation for object-oriented or object-orientation.

In a questionnaire about the course the course participants indicated that they invested an average of 11.6 hours per week (median: 10 hours) over the 5 weeks of the course. A minimum of 75 percent of all points of the exercises was required for participating in the experiment; 59 of the 73 course participants qualified.

2.2.3 Preparation: The Informatik II course

The subjects were taught as a part of their undergraduate studies. They learned basic computer science knowledge and used the Java programming language for their practical exercises. If they solved enough exercises, they obtained a bonus for their exam.

Among others, practical lab exercises contained a character counter, concatenated lists, different output representations (reuse of the concatenated list), cross sum calculation with reading from standard input, exceptions, read in data with AWT, binary trees, data processing with reading from files, a simple AWT game, and hash tables.

2.3 Experiment format and conduct

We carried out the JAKK run of the experiment in June 1997 on a Tuesday morning, 8 00 to 13 00 hours. The run² for the Informatik II course was in June 1997, too, but later in the afternoon, from 16 00 to 22 00 hours. The experiment was conducted in a computer lab equipped with 80 IBM RS6000 workstations running the AIX operating system. The subjects were allowed to use any installed editor. Most of them used either Joe or Emacs. The Java installation was JDK1.1. Although the subjects knew before the experiment that they had to work in an Unix environment and that they should be familiar with it, we could not be sure that they all knew the working environment, so we gave them a short additional handout describing useful commands and tools.

The first handout contained two parts. The first part, a questionnaire, covered personal data such as name, semester, programming experience, etc., see Appendix A.2. The second part was a small test of Java and OO knowledge to see whether the groups had been well balanced, see again Appendix A.2. The subjects had to solve the exercise without using the computer. After they completed this, they were handed out the first work task consisting of a task description (see Appendix A.2), a pretty-printed program listing (see Appendix B.1), a class diagram with the inheritance structure (see Appendix B.2), and the short Unix information mentioned above. At this point, the subjects had to log into their computers. Subjects of the JAKK course found the program in their account, subjects of the Informatik II course had to invoke a script that copied the program into their account.³ As a subject finished with the task, we collected the exercise, checked the time stamp and whether s/he had invoked the required script for delivering the source code, and then handed out the next work task. We performed the same procedure after the second task and handed out the postmortem questionnaire. The order of the handouts is shown in Figure 2.2

For further details see the actual documents as used in the experiment; they should be self-explanatory and are printed in the appendices starting on page 55.

²Actually, this was conducted in two parts on different days, because not all students were available on the same day.

³This is because the JAKK subjects had new accounts that we had configured before, whereas the Informatik II subjects used their own accounts.

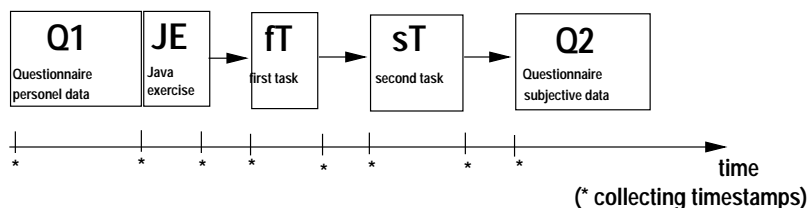


Figure 2.2: Order of the handouts, each arrow between a block means collecting the last handout, controlling of the timestamp and distributing the next handout. At each mark (*) on the time axis a timestamp is collected.

2.4 Experimental subjects JAKK

2.4.1 Education

57 subjects participated in the first run of our experiment. All of them were male Informatics students, and all but 8 of them held a Vordiplom (similar to B.Sc.) degree, one subject did not answer the question. On average, these students were in their eighth semester at the university; see Figure 2.3.

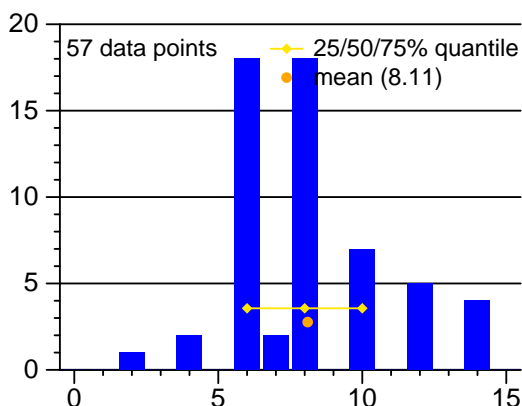


Figure 2.3: [JAKK] Distribution of semester numbers of experimental subjects.

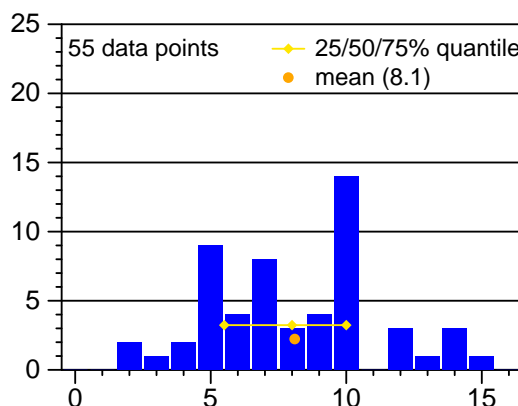


Figure 2.4: [JAKK] Distribution of years of programming experience of subjects.

2.4.2 Programming experience

The students had an average of 8.1 (median: 8) years of programming experience (Figure 2.4), 79% of them had written more or much more than 3000 Lines of Code (LOC) in their life (Figure 2.5). 91% of the subjects claimed they had significant practical experience with object-oriented programming (Figure 2.6) and 47% claimed that they had significant practical experience in programming graphical user interfaces (Figure 2.7). A subject is classified as having significant practical experience if s/he has written more than 300 LOC on a topic. Our subjects had practice with a median of 4 different programming languages (Figure 2.8). The median of the largest program ever written by each of our subjects had 2750 LOC and took 2 person months; see Figures 2.9 and 2.10. 39% of the subjects had also previously participated in a team software project and their median contribution was 1300 LOC and 2 person months to the total median project size of 20 KLOC and 6 person months; see Figures 2.11 to 2.14.

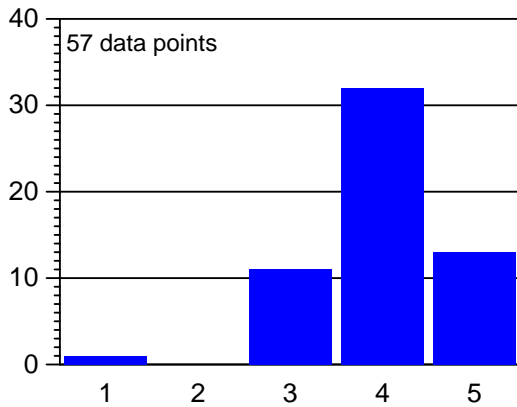


Figure 2.5: [JAKK] Distribution of previous programming knowledge and experience: 1=only theoretical knowledge, 2=less than 300 LOC written, 3=less than 3000, 4=less than 30000, 5=more than 30000.

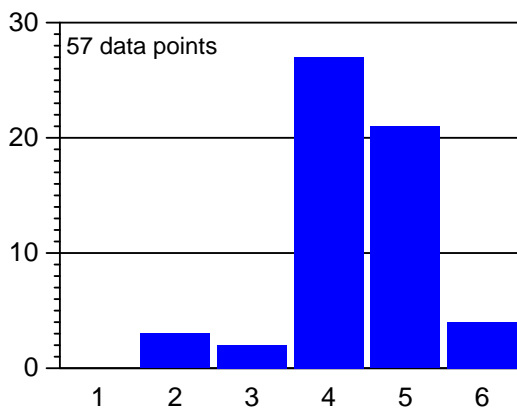


Figure 2.6: [JAKK] Distribution of previous experience in object-oriented programming: 1=no knowledge, 2=only theoretical knowledge, 3=less than 300 LOC written, 4=less than 3000, 5=less than 30000, 6=more than 30000.

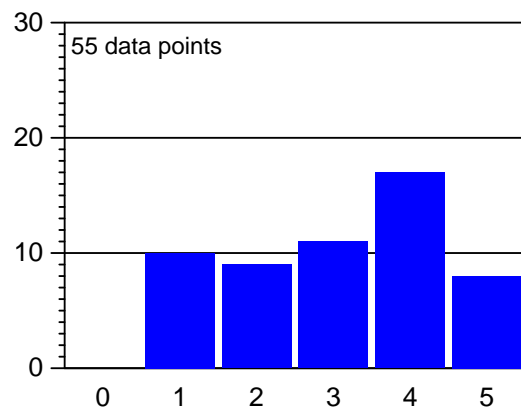


Figure 2.7: [JAKK] Distribution of previous experience programming in graphical user interfaces (GUI). The same encoding is used as in Figure 2.6.

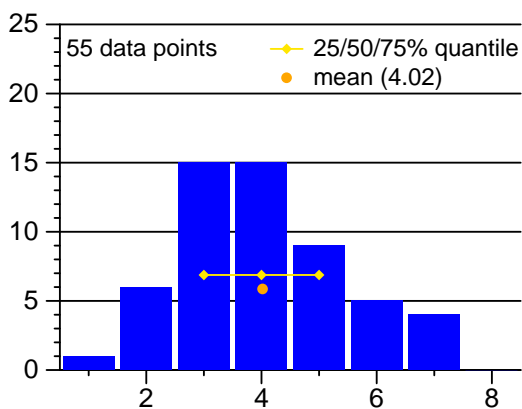


Figure 2.8: [JAKK] Distribution of number of programming languages previously used.

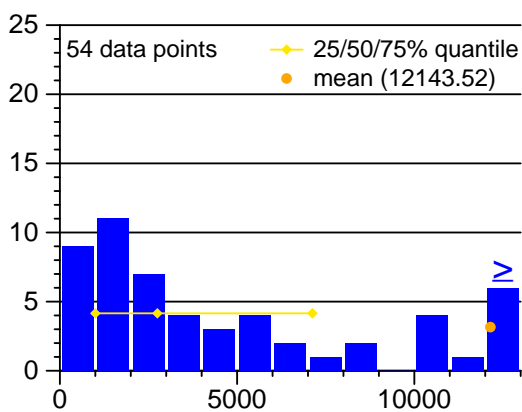


Figure 2.9: [JAKK] Distribution of size (in LOC) of the subject's largest program ever written alone.

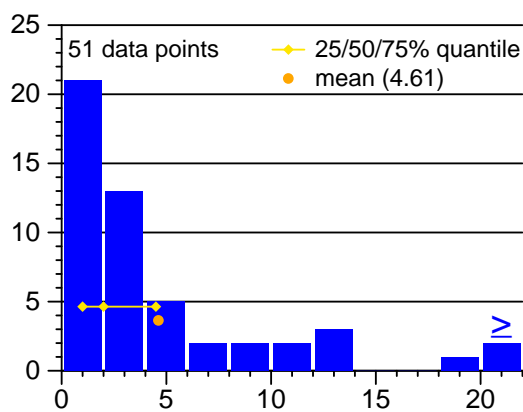


Figure 2.10: [JAKK] Distribution of time (in person months) of largest program ever written alone.

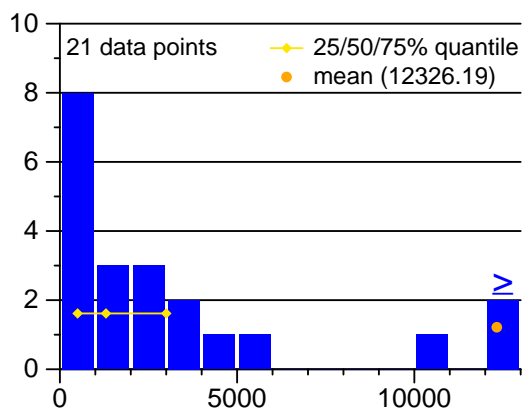


Figure 2.11: [JAKK] Distribution of size (in LOC) of subject's contribution to his/her largest team software project.

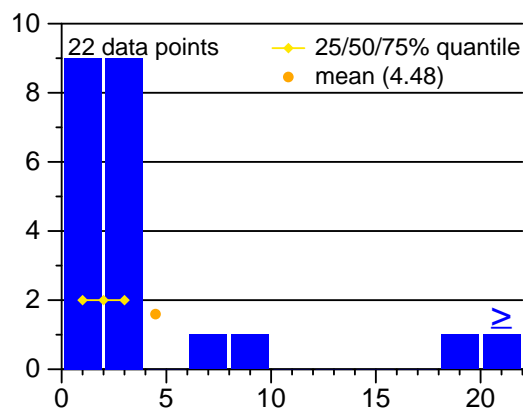


Figure 2.12: [JAKK] Distribution of time (in person months) of subject's contribution to his/her largest team software project.

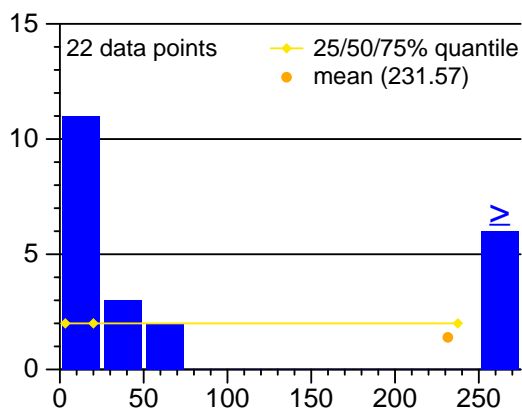


Figure 2.13: [JAKK] Distribution of size (in KLOC) of the subject's largest team software project.

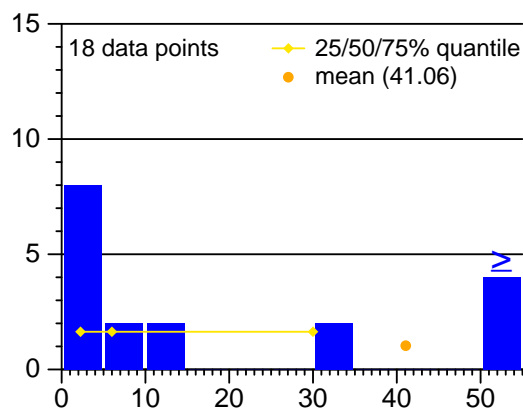


Figure 2.14: [JAKK] Distribution of time (in person months) of subject's largest team software project.

2.5 Experimental subjects Informatik II

2.5.1 Education

58 subjects participated in this run of our experiment. All of them were Informatics undergraduate students (54 male, 4 female) in their Vordiplom (similar to B.Sc.) studies, except two Physics undergraduate students and two Math undergraduate students. All students were in their second semester at the university except one first semester Math student.

2.5.2 Programming experience

The students had a median of 6 years(!) of programming experience (Figure 2.15), 53% of them had written more or much more than 3000 LOC (Figure 2.16). 62% of the subjects claimed that they had significant practical experience with object-oriented programming (Figure 2.17) and 45% claimed that they had significant practical experience in programming graphical user interfaces (Figure 2.18). A subject is again classified as having significant practical experience if s/he has written more than 300 LOC on a topic.

Our subjects had practice with a median of 4 different programming languages (Figure 2.19). The largest program ever written by our subjects had a median size of 2000 LOC and 1.5 person months; see Figures 2.20 and 2.21. 14% of the subjects had also previously participated in a team software project and contributed a median of 350 LOC and 2.0 person months to the total project size of 5.5 KLOC (median) and 2.0 person months (median); see Figures 2.22 to 2.25.

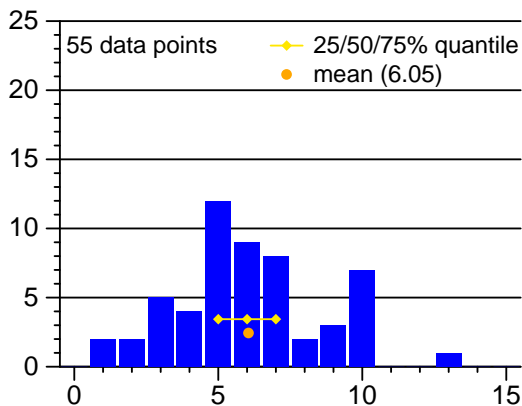


Figure 2.15: [Informatik II] Distribution of years of programming experience of subjects.

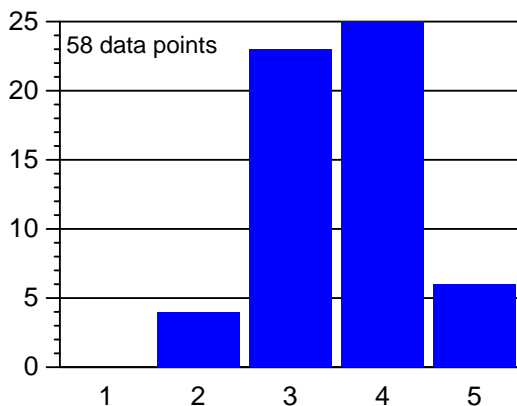


Figure 2.16: [Informatik II] Distribution of previous programming knowledge and experience: 1=only theoretical knowledge, 2=less than 300 LOC written, 3=less than 3000, 4=less than 30000, 5=more than 30000.

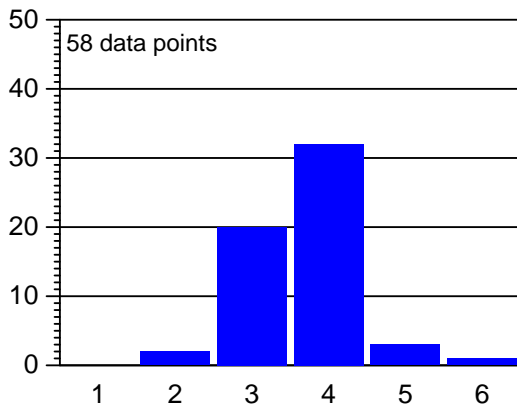


Figure 2.17: [Informatik II] Distribution of previous experience in object-oriented programming: 1=no knowledge, 2=only theoretical knowledge, 3=less than 300 LOC written, 4=less than 3000, 5=less than 30000, 6=more than 30000.

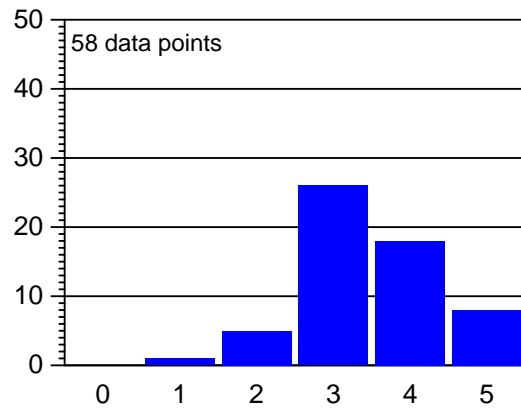


Figure 2.18: [Informatik II] Distribution of previous experience in programming graphical user interfaces (GUI). The same encoding is used as in Figure 2.17.

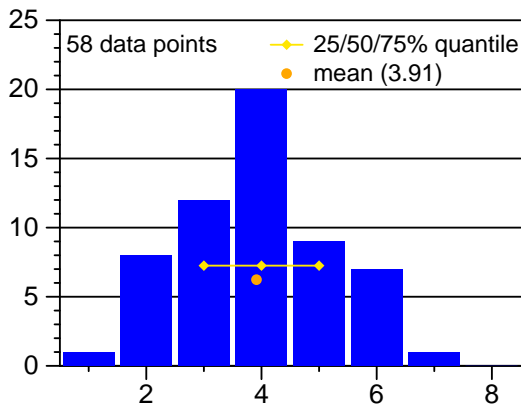


Figure 2.19: [Informatik II] Distribution of number of programming languages previously used.

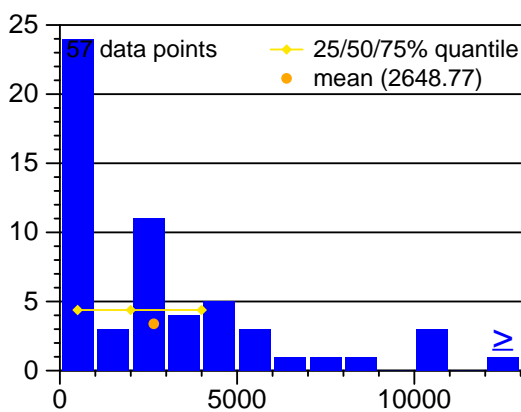


Figure 2.20: [Informatik II] Distribution of size (in LOC) of largest program ever written alone.

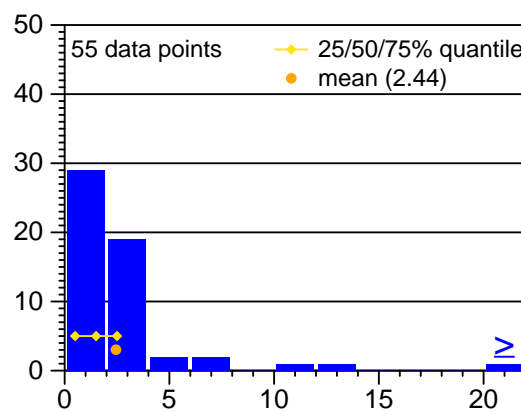


Figure 2.21: [Informatik II] Distribution of time (in person months) of largest program ever written alone.

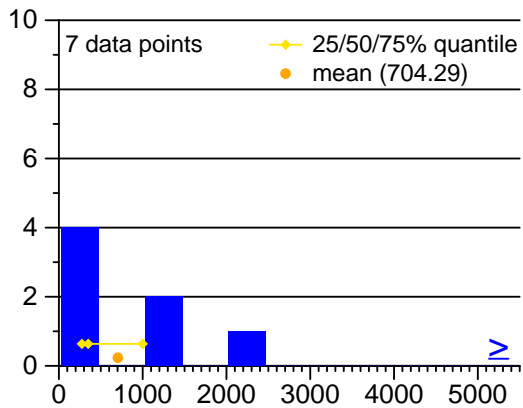


Figure 2.22: [Informatik II] Distribution of size (in LOC) of subject's contribution to his/her largest team software project.

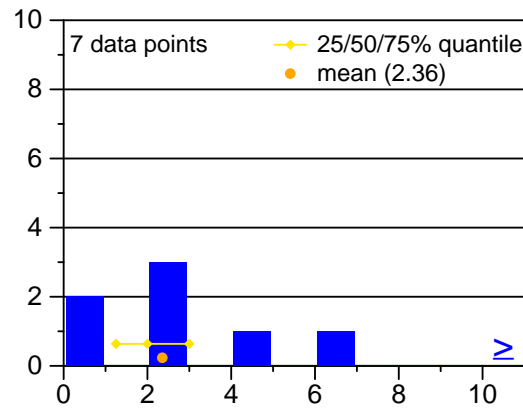


Figure 2.23: [Informatik II] Distribution of time (in person months) of subject's contribution to his/her largest team software project.

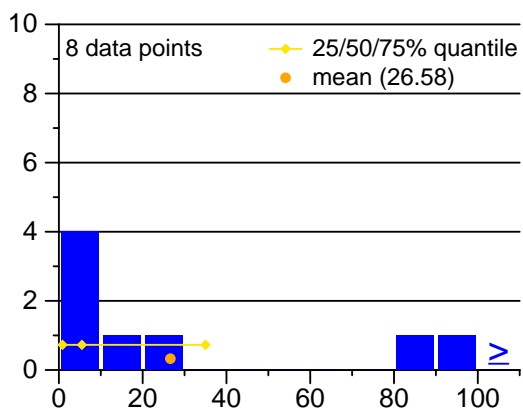


Figure 2.24: [Informatik II] Distribution of size (in KLOC) of subject's largest team software project.

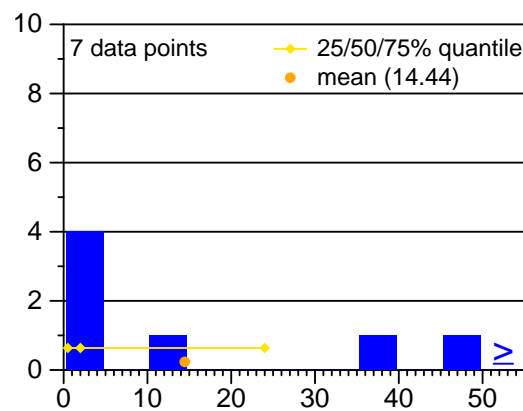


Figure 2.25: [Informatik II] Distribution of time (in person months) of subject's largest team software project.

2.6 Knowledge of Java, inheritance, and polymorphism

Before the actual experiment started, we tried to learn about our subject’s knowledge of Java, inheritance, and polymorphism. The introductory questionnaire contained a small test. They were given a tricky Java program consisting of three classes (see Appendix A.2). First the subjects had to mark those lines that would result in a compilation or runtime error. Second they had to determine the output of the program (all without using the computer.)

The answers for the first part, “mark wrong lines” we classified in four different groups: 1 = correct, 2 = subject marked lines wrong because of insufficient understanding of the keyword *static*, 3 = subject marked lines wrong because of insufficient understanding of the keyword *static* **and** marked **one** other line wrong for a different reason, 4 = subject marked **more** than one line wrong not caused by a misunderstanding of the keyword *static*. The *static* (i.e. class variables/methods) feature of Java is not relevant for our experiment. Figures 2.26 and 2.27 show the distribution for the JAKK and the Informatik II subjects.

There are some differences between the three different treatment groups of the JAKK and the Informatik II course but they are not significant, see Figure 2.28.

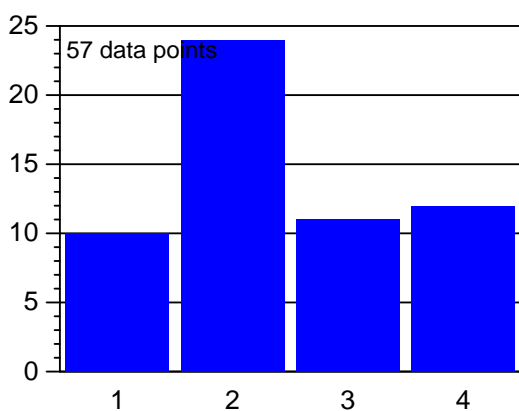


Figure 2.26: Distribution of the answer classes for the JAKK subjects. Answer classification: 1 = correct; 2 = subject marked lines wrong because of insufficient understanding of the keyword *static*, 3 = subject marked lines wrong because of insufficient understanding of the keyword *static* **and** marked **one** other line wrong for a different reason, 4 = subject marked **more** than one line wrong not caused by a misunderstanding of the keyword *static*

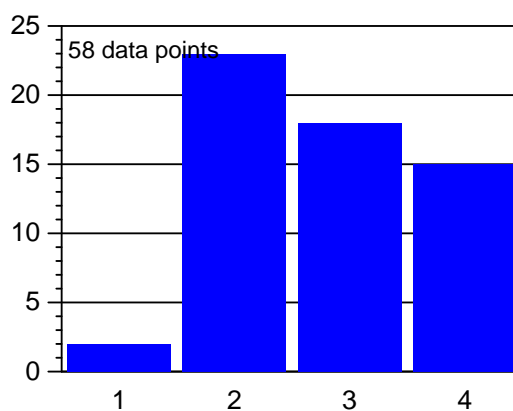


Figure 2.27: Distribution of the answer classes for the Informatik II subjects. The same encoding is used as in Figure 2.26

The second part of the test was evaluated with respect to the first part. We counted the wrong output that was not a consequence of wrongly marked lines. Figures 2.29 shows the distribution for the JAKK and the Informatik II subjects.

2.7 Constraints of the experiment

The tasks and programs used in our experiment had to obey the following constraints:

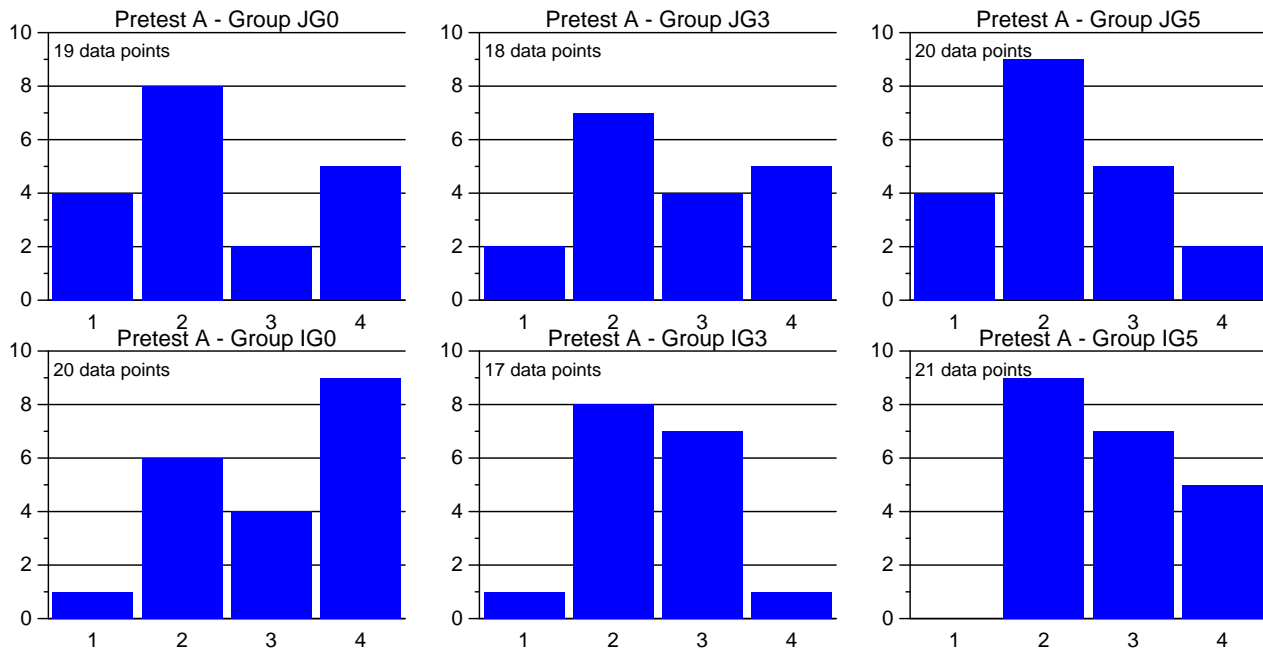


Figure 2.28: Distribution of the answer classes for the JAKK (suffix J) and the Informatik II (suffix I) subjects for the first part of the Java-Test separated into the different treatment groups, where G0 is the group with the flat program version, G3 with the inheritance depth 3, and G5 with the inheritance depth of 5.

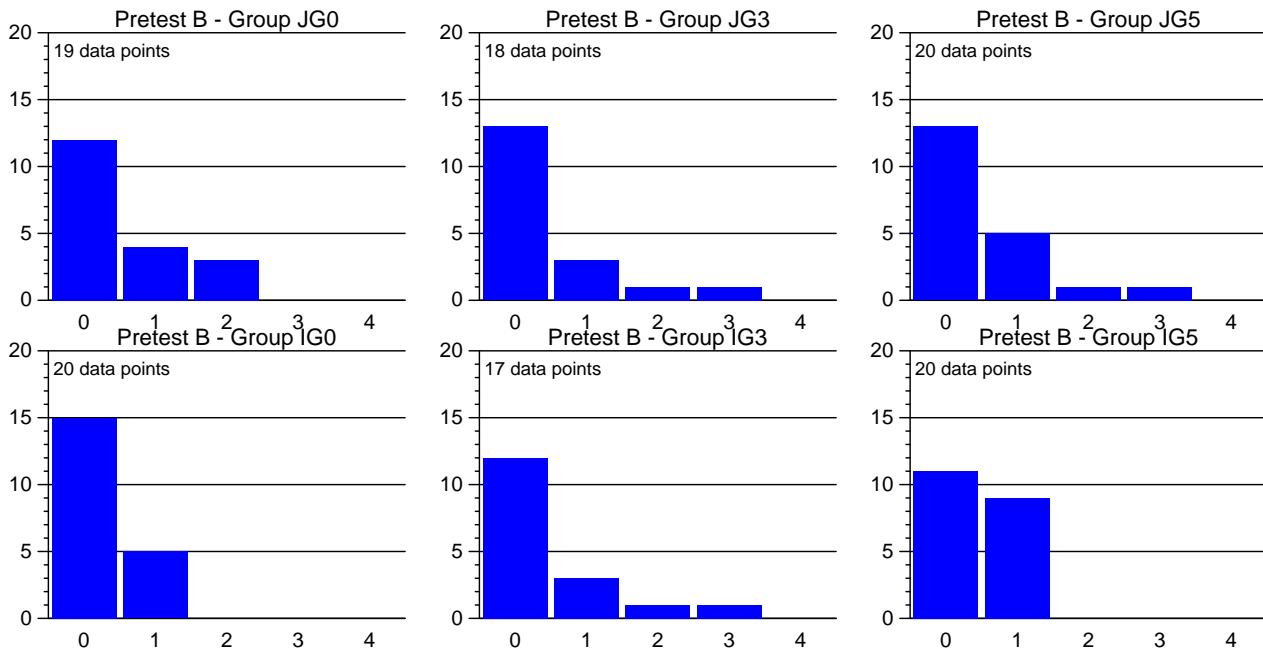


Figure 2.29: Distribution of the number of wrong answers for the JAKK and the Informatik II subjects for the second part of the Java-Test separated into the different treatment groups.

1. The experiment has to be carried out in a single time interval. We assume that four hours is a reasonable maximum, due to the subjects' concentration ability. Therefore, the tasks should not consume more than about two hours for an average subject.
2. The application domains of the programs had to be well understandable by all subjects. Therefore, we could only use domains that were either known from the course or were so simple that they could be explained in a few sentences.

We handled these constraints as follows. Constraint 1 rules out large programs that are much larger than the largest program they have ever written alone. (The median of 2750 LOC for the JAKK subjects and 2000 LOC for the Informatik II subjects suggests a program of a size not much larger than 2000 LOC.) Complex and difficult tasks are also ruled out. Therefore, we used modest tasks (described below) and a program of modest size.

For satisfying constraint 2, we selected a domain that is mostly intuitive and easy to explain in a few words. Viewing data points in different manners is common and not difficult to understand even if the subjects do not know the correct technical terms. Assuming that some subjects were not familiar with the technical terms we provided descriptions of the domain and the meaning of the technical terms used in the program, see Appendix A.2.

2.8 Programs and Tasks

This section will shortly describe the programs and tasks and explain why we chose them. The tasks for the JAKK subjects differ from the tasks for the Informatik II subjects but the programs are almost the same, see below. One can find the task descriptions translated into English in Appendix A.2 and the corresponding program listings in Appendix B.1 (not translated, but see the translations of technical terms given in Appendix C).

2.8.1 Programs

For testing the hypotheses we need programs differ only with respect to inheritance depth. Therefore we developed a base version of an object-oriented program with a deep inheritance hierarchy and derived two new program versions from it by eliminating inheritance levels.

The base version is a Java application using the Abstract Window Toolkit. The application, called "Boerse", is for presenting stock market data that are stored in two files. The stock market data can be displayed in different kinds of tables (data of one day, data of the last week, data of the last month, the percentage of changes between two prices) or charts (for selected stock data prices of a day or the last month). For more details about the domain and functionality, see Chapter B.1 or A.2.

The base version consists of 28 classes (7 abstract classes, 1 interface and 20 concrete classes) with an inheritance depth of 5, where inheritance depth is defined as the number of extension of the longest chain of extensions (B extends A)⁴. For a visualization of that definition, see Figure 2.30. Some of the abstract classes use the design pattern "Template Method", (as in a quicksort algorithm that is implemented in the abstract superclass except for the compare method for the different data types. This base version with an inheritance depth of 5 will be called Boerse5 in the following.

⁴A maintenance task on a program with an inheritance depth of n that requires adding a new abstraction level is defined to have inheritance depth $n+1$, because maintainers have to understand n inheritance levels plus the one they have to add.

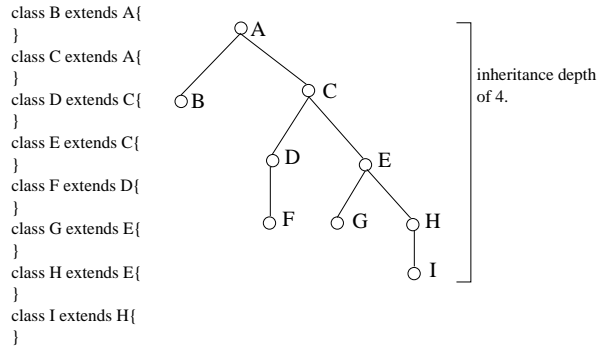


Figure 2.30: Inheritance depth is defined as the number of pairs in the longest chain of extensions (“A extends B”) of a class. In this example: C extends A; E extends C; H extends E; I extends H. This is an inheritance depth of 4.

program version	JAKK			Informatik II		
	LOC	# classes	#methods	LOC	# classes	#methods
Boerse5	1200	28	80	1187	28	79
Boerse3	1344	27	100	1317	27	96
Boerse0	2470	20	158	2465	20	160

Table 2.1: Size characterization of the programs used for the experiment

The version with an inheritance depth of 3 was derived from the base version by making all concrete classes inherit from only abstract classes following the suggestions of Gamma et al. [7] to prefer inheriting interfaces over inheriting implementations. The functionality (methods) and the comments defined in inherited concrete superclasses were moved into the derived class by “cut and paste”. Functionality and comments that are not used by the derived class were omitted. In the following, we call this version Boerse3.

The flat version, called Boerse0, was built from the inheritance version Boerse3. All abstract classes were removed and the functionality of the superclasses were added into the concrete classes. Useless functionality that was defined in the superclass but not used in derived classes was removed. Comments were moved along with the code as appropriate.

There are only small differences between the programs for the JAKK course and the Informatik II course. The programs for JAKK subjects are written in the programming style of JDK 1.0.2 (using the now deprecated event handling) because most subjects learned this. The Informatik II subjects learned JDK1.1 with the new event handling mechanism. So we modified the programs accordingly and added some comments concerning the event handling. For details, see Appendix B.3.

2.8.2 Tasks

In total, we had three different tasks, called “Task 1”, “Task 2A”, and “Task 2B”. Subjects from JAKK had to work on all three tasks. For them, we did not measure Task 2A and Task 2B separately; the combination is called “Task 2”.

Trying to satisfy constraint 1 (time) we omitted Task 1 for the Informatik II subjects because we assumed that the Informatik II subjects were less experienced programmers than the JAKK course subjects.

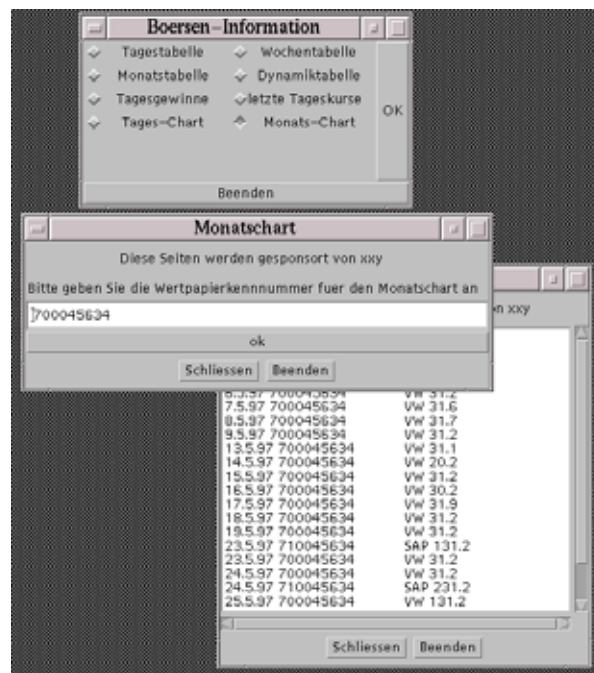


Figure 2.31: Screenshot: The upper window is the main menu window. The middle window is for selecting the stock number. A user can select a stock number and press the OK button. The lower left window displays a month table of all stock data of the last 30 days.

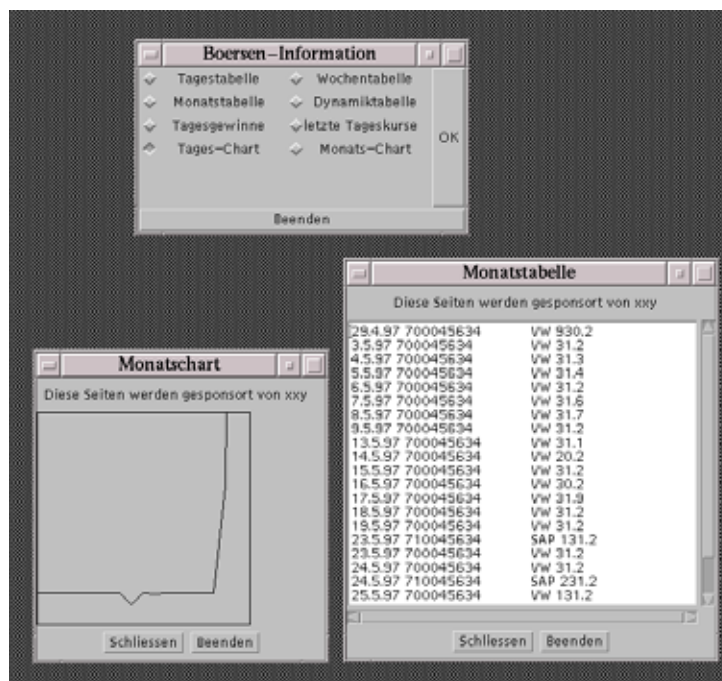


Figure 2.32: The scene from Figure 2.31 after pressing the OK button.

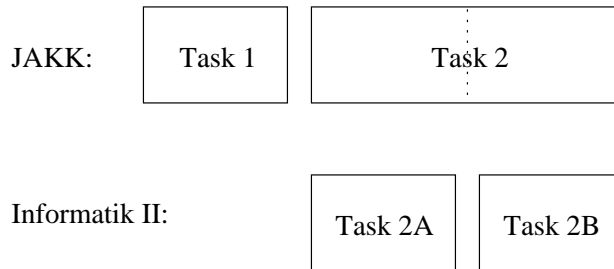


Figure 2.33: Task order for the two groups. Because of the time constraints we omitted Task 1 for the Informatik II group and split Task 2 into two subtasks 2A and 2B instead.

Task 1 – Y2K-Problem

The first task addresses the Y2K-problem (Year 2000). The format of the date in the original input files is “dd.mm.yy”. We provided a second set of input files that differed only in the date format: “dd.mm.yyyy”. The subjects had to change the program so that instead of the original input files with the old date format the new files with the extended date format are handled by the program.

Extract from the file KURSE (PRICES), where the first line is the translation of the row names. (This line was not in the original files.)

```

#date          time      stocknumber ID      name          price
#DATUM         UHRZEIT  WERTPAPIERKENNUMMER NAME           KURS
01.03.97       11:51    700045634           Daimler Benz  130.70
01.03.97       12:34    700045634           Daimler Benz  134.50
01.03.97       14:26    700045634           Daimler Benz   34.20
  
```

Extract of the file KURSE_2000:

```

#date          time      stocknumber ID      name          price
#DATUM         UHRZEIT  WERTPAPIERKENNUMMER NAME           KURS
01.03.1997     11:51    700045634           Daimler Benz  130.70
01.03.1997     12:34    700045634           Daimler Benz  134.50
01.03.1997     14:26    700045634           Daimler Benz   34.20
  
```

Extract from the file KASSAKURSE (AVG_PRICES):

```

#date          stocknumber ID      name      day-average price
#DATUM         WERTPAPIERKENNUMMER NAME       KASSAKURS
02.04.97       700045634           VW        131.20
03.04.97       700045634           VW        231.20
03.05.97       700045634           VW         31.20
  
```

Extract from the file KASSAKURSE_2000:

```

#date          stocknumber ID      name      day-average price
#DATUM         WERTPAPIERKENNUMMER NAME       KASSAKURS
02.04.1997     700045634           VW        131.20
03.04.1997     700045634           VW        231.20
03.05.1997     700045634           VW         31.20
  
```

Task 2A – Time Interval Price Display

The given program has two ways of displaying stock charts and six sorts of table-based stock data displays. All table displays present data for fixed time intervals (day, week, or month) and there is no way to focus on a particular stock number. In contrast, the chart output asks the user to enter a stock number. Then the chart for that particular stock in a fixed time interval will be displayed.

For Task 2A the subjects had to add a new table-based output. But instead of a fixed time interval, the user should be prompted (similar to the stock number selection for chart displays) to specify the time interval for which the data was to be displayed.

Task 2B – Time Interval Gain/Loss Display

Task 2B is very similar to Task 2A. Again, a given table display, this time a gain/loss table, must be extended to accept user input for selecting a specific time interval.

2.8.3 Solving the tasks

Solving Task 1

In this section it will be discussed that Task 1 does not require a full understanding of the programs and the inheritance. Most JAKK subjects will complete Task 1 without even trying to fully understand the code. Others will gain a deeper understanding of the code before tackling Task 1.

A straightforward way to solve this task is to trace the data flow through the program. The lines where the input files are read are easily found in all versions of the code. In Boerse0 there are 8 identical code fragments that handle the file input. In Boerse3 and Boerse5 there is only a single position in the code. With the inheritance diagram at hand, the spot where input files are read is easy to find, for example Program Listing B.1.3, lines 184-200.

From the file input method called `leseDaten [readData]` individual lines of input are read. These lines are then processed in two different methods. One method, called `selektiere [select]` (Program Listing B.1.3, lines 378, 466, 556, and 794) analyses the line and determines whether it will be included into the output. This method is invoked by another method, called `erzeugeDatenvektor [generateDataVector]` (Program Listing B.1.3, lines 368, 459, 563, and 800) that constructs a special data type from the input line. In both methods the input line is segmented with `substring` and explicit line positions. With that understood, the subjects could conclude that they had to change the explicit positions in the `substring` operations in both methods.

In the flat version Boerse0 and in Boerse3, each of the two methods `selektiere [select]` and `erzeugeDatenvektor [generateDataVector]` appears 8 times. In Boerse5 the methods appear only 4 times. However, if the concept is understood, a single pass through the file with the “search next” mode of the editor is sufficient to complete the task since all appearances look much alike.

In addition, the special data type that is used to store the date had to be adopted to the four digit year format. This change was identical in all three versions.

The difference with respect to inheritance between the flat version and the other versions was locality. In the flat version, the name of the data file is stored into an instance variable. In immediate proximity in the code,

this variable is used to open the file. This is different in the other two versions. The method that handles the files (`leseDaten` [`readData`], class `KostenloseInfos` [`FreeInfos`]) uses the same instance variable to access the file name, however, that instance variable is initialized in the sub-sub-classes (class `ZeitraumTabelle` [`IntervalTable`] in `Boerse5`), see Figure 2.34.

A solution for this task is given in Appendix A.3.1

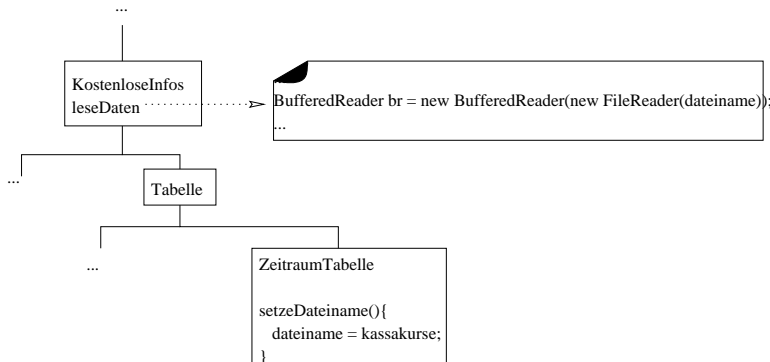


Figure 2.34: Initialization and access of the input file name for Boerse5

Solving Task 2A

From the task description and from executing the original program, the subjects can conclude that almost all of the required functionality is already present in the program. The new table-based output must be quite similar to existing table-based output. In addition, for implementing the required user input the subjects can find an example from the chart implementations where the user is prompted to enter a stock number. Moreover, the main menu must be extended to offer the new option in close analogy to its existing entries.

Gaining comprehension for Boerse0: For the flat version `Boerse0`, the subjects can gain an understanding of the charts and tables by selecting an example class of each; one table-based class and one chart class. Due to the flat code, a single class contains all code needed to implement the table output or the chart output. The class implementations are organized homogeneously. Thus, only one class has to be understood fully, e. g. `WochenTabelle` [`WeekTable`]. If the maintainer realized that the functionality he needed is already there, a line-by-line comparison reveals differing aspects between chart and table implementations: some new instance variables are needed for the user input and three methods are different, namely, the event handling method (`myaction`⁵), result preparation method (`selektiere` [`select`]) and result display method (`stelleDar` [`present`]). Given an understanding of one table class and the differences of the two class types (table classes and chart classes) the solution of the task becomes quite simple: A new class is created as a copy of an existing table class; then the three methods mentioned above are altered according to the differences in comparison with the chart class.

Gaining comprehension for Boerse3 and Boerse5: Doing the same maintenance task for `Boerse3` or `Boerse5` this is more complicated. First, one of the table classes has to be understood and the differences between the table-based classes and the chart classes have to be found. However, for gaining an understanding of the class `WochenTabelle` [`WeekTable`] or `MonatsTabelle` [`MonthTable`], one must seek and find the functionality in the inheritance hierarchy. This is shown in Figures 2.35 and 2.36.

⁵Only for the JAKK subjects. Informatik II subjects had event handling using inner classes, see Section B.3

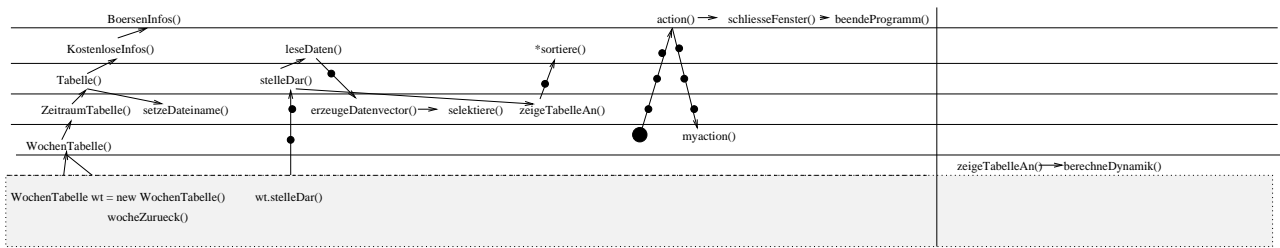


Figure 2.35: For Boerse5 program comprehension must be gained by searching through the inheritance hierarchy, starting in the main program. The arrows mean an invocation of a method. An arrow crossing a horizontal line means that the method invoked is located in a super- or subclass. The shaded region in the lower part are method calls from classes outside the main hierarchy. The black dot in the figure is a starting point for searching the method “action” that is invoked implicitly by the AWT. Methods are marked with a star if they invoke other methods but due to meaningful names method invocations do not need to be traced. The left side of this figure is for Task 2A and the right side for Task 2B.

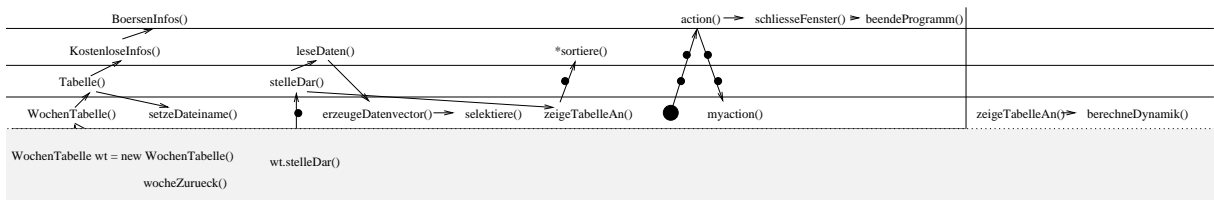


Figure 2.36: Same as Figure 2.35 for Boerse3.

The differences between the charts and tables can be found by comparing the two types of classes including the functionality that is realized in superclasses. So the maintainer has not only to search two classes for differences, like the Boerse0 case; instead, s/he has to search the whole inheritance path to the root.

As discussed for Boerse0, three methods differ. In Boerse3, result display is implemented in the abstract table super-classes of the concrete table classes. Result preparation is implemented on the concrete class level. Therefore, for all three methods different classes must be compared to understand the differences between table output and chart output.

In Boerse5 this becomes even more complicated since result preparation occurs at different levels of the class hierarchy.

In Table 2.2 the methods that have to be compared, copied and/or changed are listed with the class levels in which the classes are located that contain such a method.

	myaction()		selektiere() [select()]		stelleDar() [present()]	
	Chart	Table	Chart	Table	Chart	Table
Boerse0	0	0	0	0	0	0
Boerse3	3	2	3	3	2	2
Boerse5	4	2	3	3	3	3

Table 2.2: Search levels where the differing methods can be found.

Solving: Whereas the structure of the given code suggests to copy and modify an existing table class for Boerse0, there are different solution types for Boerse3 and Boerse5. The cut-and-paste-and-modify approach does not work, since the code to be modified is spread across super-classes as well. Therefore, some methods must be redefined to hide unwanted implementations in super-classes. For Boerse3 the new table can be added at the same level as existing tables, or it can be added as a subclass of an existing table class. For Boerse5 the same is true; however, in this case different types of tables are at different levels of the inheritance graph. Subjects must be more careful to pick the appropriate ancestor. The most useful ancestor class is the interval table display class `ZeitraumTabelle [IntervalTable]` that implements large parts of the new functionality already.

Solving Task 2B

Since the task is similar, the cut-and-paste-and-modify approach can be used again for Boerse0. In comparison with Task 2A, analogous changes must be repeated (three methods and some instance variables) and analogous code lines must be kept unchanged. For Boerse3 and Boerse5 a new class must be introduced, but the body of that class is completely identical to the class introduced during Task 2A.

A solution for Tasks 2A and 2B is given in Appendix A.3.2

2.9 Internal validity

There are two sources of threats to the interval validity of an experiment⁶: Insufficient control of relevant variables, and inaccurate data gathering or processing.

As far as we can see, all relevant external variables have been appropriately controlled in this experiment. In particular, there is no significant bias in the random group sampling, the subjects seemed willing to perform as best as they could in all three experimental conditions, there was acceptable mortality in the JAKK group (7 subjects did not start the second part of Task 2, 2 subjects of group JG0, 3 subjects of JG3, and 2 subjects of JG5). The mortality in the Informatik II group was biased (11 subjects did not start with Task 2B, 3 subjects of group IG0, 2 subjects of IG3, and 6 subjects of IG5) so the results presented have to be interpreted carefully. The environmental conditions were essentially the same for all subjects.

We minimized data gathering errors by exercising utmost care. Data processing was almost completely automated. Manual and automated consistency checks were applied for detecting various kinds of possible mistakes in data gathering or processing.

2.10 External validity

There are four causes for differences between the experimental situation and real software maintenance situations that limit the generalization (external validity) of the experiment: (1) subjects with more experience, (2) programs of different size, structure, or programming language, (3) familiarity with the program, and (4) tasks of different kind or complexity.

⁶Definition from [1]: “*Internal validity* refers to the extent to which we can accurately state that the independent variable produced the observed effect.”

1. The most frequent concern with controlled experiments using student subjects is that the results cannot be generalized to professional software engineers because the latter are so much more experienced. Because of the replication with different groups (JAKK and Informatik II), we can estimate the effect of having programmers with different experience. If the data have the same trend for more experienced subjects as for less experienced subjects (which ours do), the results may be also transferable to professional programmers. Furthermore, our JAKK subjects are quite experienced, anyway; see Section 2.4.1.
2. Another obvious difference between our experiment and reality is program size and structure. Compared to typical industrial programs, the experiment programs are rather small and are not as complex as (large size) industrial programs. The programming language is Java, so we can, for instance, not determine the effects of multiple inheritance. The influence of both factors is unknown.
3. Usually a programmer maintains a program more than once. So after a learning phase the programmer works on a familiar program. Our subjects work on a unfamiliar program and they are perhaps not acquainted with the programming style. It is unclear whether or how program familiarity interacts with inheritance depth.
4. Finally, the kind of task and its complexity may be different. In the experiment, the tasks were relatively simple program changes or program extensions that can be done mainly by adding some completely new classes, and subjectively the complexity was rather low. The experiment does not tell us much about other kinds of tasks.

Chapter 3

Experiment results and discussion

This chapter presents and interprets the results of the experiment. Section 3.1 explains the methods of statistical analysis and result presentation that we used and explains why they were chosen. Section 3.2 presents the main results (subjects' measured performance) and Section 3.3 adds data from the postmortem questionnaire for understanding some underlying effects.

3.1 Statistical methods

Most of our formal statistical reasoning is for comparing the means of pairs of distributions. Hardly any of these distributions are “normal distributions”: Some of them are discrete and coarse-grained, several of them have two or more peaks, and many are heavily skewed or even monotone. Therefore, statistical analysis must not use a parametric test such as the t-test that assumes a normal distribution. On the other hand, classical non-parametric tests, such as the Wilcoxon rank sum test, cannot perform inference for the mean but only for the median, which is less relevant for our purpose. Moreover, rank sum tests cannot provide confidence intervals.

In this report, we thus use resampling statistics (bootstrap) to compare the means of arbitrary distributions non-parametrically. The basic idea of resampling is described in [8]. For all resampling statistics we used 10000 trials.

The other statistical test used is the χ^2 test on a four field table for testing the significance of frequency differences of a binary attribute. The application is comparing the incidence of a certain event in two experimental groups. We report the Pearson's chi-square test with Yates' continuity correction and the Fisher exact p statistic in addition to the p -value of the Pearson's χ^2 test; the exact p is more reliable in some cases (Fisher's exact test does not require the cell counts to be large). These tests were performed using Statistica 5.0 (Windows NT 4.0) and S-Plus 3.4 (Sun OS 5.5).

We consider a test result significant if p is less or equal to 0.1.

3.2 Performance on the tasks

In this section, we compare the performance of the groups with different inheritance depths. For each task, we first consider the individual types of errors that occurred and then investigate global quantitative effects with respect to time required and solution quality obtained.

3.2.1 Metrics employed

In the evaluation below, the following measurements and criteria will be used. Each class of them is described by the following terms [6]: A measurement can be either objective (and therefore in principle completely reproducible) or subjective (and therefore subject to debate); it can be either direct or be derived from other measurements; it can be on a nominal, ordinal, interval, cardinal, or absolute scale; it can have limited precision and limited accuracy even if it is objective.

Groups (objective, direct, nominal scale, completely accurate): The groups were used for comparing performance with maintenance tasks on programs of different inheritance depth. The groups are named by the course (JAKK = J, Informatik II = I) and the program version they worked on (flat version = G0, inheritance version with depth 3 = G3, inheritance version with depth 5 = G5).

Time measurements (objective, direct, cardinal scale, precision 1 minute, inaccuracy should be about 1 minute): The subjects noted start and end times on each page of the experiment materials. We computed the difference between the end of the last page of a task and the start of the first page of the task as the work time measurement; the subjects did not make major breaks that had to be subtracted. We also gathered time data by scripts run on the subject's computer during the experiment so that we could do consistency checks in case that the start time of a new task was not close to the end time of the previous task. We used the time data only on the task level (as opposed to the subtask level as for the JAKK subjects between the first and the second part of Task 2) as it is the one with the clearest interpretation and the one that was validated upon collection of each part of the experiment materials.

Points achieved (subjective/objective, direct, cardinal scale, precision 1 point): We graded the solutions by assigning points, using a penalty system where possible (subtracting a fixed number of points for each kind of error). The individual penalties are explained in the actual results sections below. We consider the differences of numbers of points between the groups for each subtask individually and for the whole task. Points are meant to characterize the quality of a solution, but this interpretation must be applied only with care.

Incidence counts (subjective, direct, absolute scale): Incidence counts reflect how often a particular event occurs in a group. We considered incidence counts for various classes of errors in the solutions delivered by the subjects. In a few of the cases it is debatable whether a certain solution is an instance of the event or not, so there is some amount of subjectivity in the data. Except for subjectivity, the incidence data is considered accurate.

Productivity (objective, derived, cardinal scale, precision $\frac{1}{60}$ pt/hour): A measure that is meant to characterize productivity was derived by computing points per time unit. Due to the restrictions of the point measure, points per time also have to be interpreted with care.

3.2.2 Notation

For the discussion of the results we introduce a short and compact notation to summarize the group comparisons. This will help the reader to gain a quick overview.

Definitions:

$G_i, G_j \in \{IG0, IG3, IG5, JG0, JG3, JG5\}$

$R \in \{t, pt, pr\}$ where t is mean time; pt is mean points; pr is mean productivity

$R(G_i) < R(G_j)$, if the result R of G_i is **significantly smaller** than the result R of G_j , $p < 0.1$.

$R(G_i) \leq R(G_j)$, if the result R of G_i is **smaller** than the result R of G_j , but $0.1 \leq p \leq 0.2$

$R(G_i) \approx R(G_j)$, if the p -value is larger than 0.2.

We also use the symbols $>$ and \geq accordingly.

3.2.3 Performance on the tasks – JAKK

Task 1 – “Y2K”

This task can be divided into two dependent subtasks, program comprehension and program changing. The program comprehension does not have to be extensive to get the task done because the way how the date is read and manipulated is easy to discover. See 2.8 for a detailed discussion of Task 1.

Time: It is remarkable that JG5 and JG0 are significantly faster than JG3 ($t(JG5) \approx t(JG0) < t(JG3)$), see Table 3.2, lines 4 - 6). Figures 3.1 and 3.2 show the time distribution for all subjects and by treatment groups. You can see from these figures that there are some “outliers” that may distort the results. We can eliminate them by cutting off 10 or 20% of the right side of the distributions but the results remain similar. The obtained results seem to contradict Hypothesis 1. This will be discussed in the Paragraph *Conclusion* below.

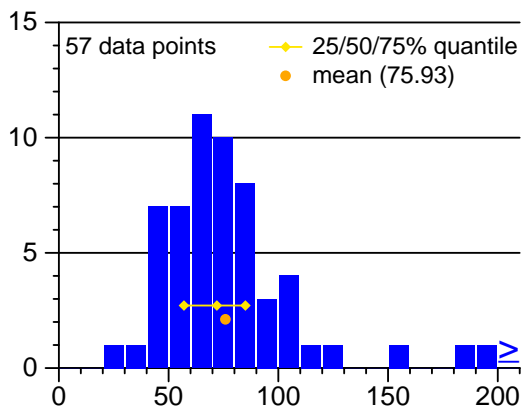


Figure 3.1: [JAKK] Distribution of the time required for the task “Y2K”

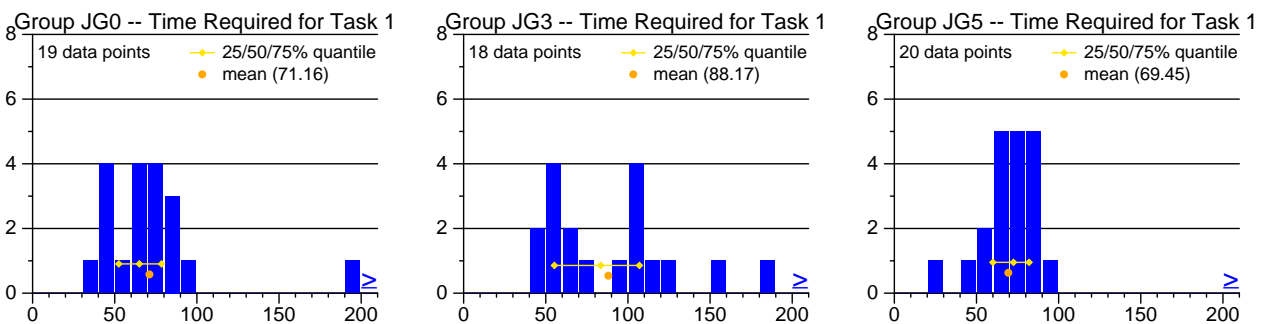


Figure 3.2: Distribution of the time required for the task “Y2K” separately for the three treatment groups.

Points: The quality of the solutions differs between the groups. The maximum of 4 points was given for correct solutions as determined by a black box test. Errors cost between 1 and 4 points depending on the error

group	no. of subjects	no. of correct solutions	groups	χ^2	p	Fisher exact p
JG0	19	100%	JG0 - JG3	1.57	0.21	0.105
JG3	18	83%	JG0 - JG5	4.63	0.031	0.020
JG5	20	70%	JG3 - JG5	0.34	0.56	0.454

Table 3.1: [JAKK] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 1.

class. For errors of class B (one/two/more tables or charts are empty) we reduced the points by 1, 2, or 4; errors of class C (at least one table or chart wrong; subsequent substring indices not corrected) costed 1 point, and errors of class D (forgot to change a hard-coded date variable) costed 2 points. Error class X for miscellaneous errors was judged individually for each case. As you can see from Figure 3.3, all subjects from JG0 produced correct solutions while some subjects from the groups JG3 and JG5 made some errors. A significance test for differences in correctness between JG0 and JG3 or between JG0 and JG5 is not useful because in group JG0 there is no variance at all. An incidence count on the number of correct solutions shows a significant difference between the groups (see Table 3.1; for this table the Fisher exact p -value is more reliable, because some of the counts in the four field table are very small).

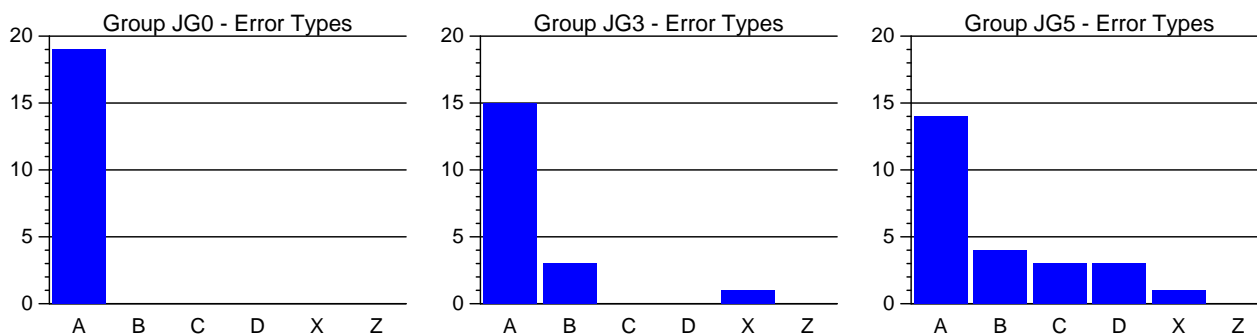


Figure 3.3: Frequency of different error types for task “Y2K”. Codes: A = everything OK; B = one/two/more tables or charts are empty; C = at least one table or chart wrong (subsequent indices not corrected); D = forgot to correct ‘heute’ (today), a hard coded date; X = other; Z = nothing done but delivered. (A solution can have more than one error class.)

Productivity: When comparing time and points results, there is only one obvious conclusion: JG0 was more productive than the other two groups. Comparing JG3 and JG5 there is no significant difference in the productivity: ($pr(JG0) > pr(JG3) \approx pr(JG5)$).

Conclusion: From the time results $t(JG0) < t(JG3)$ and knowing from Section 2.8.2 that there are the same number of changes for both program versions we can conclude that the program Boerse3 is harder to understand than Boerse0. So Hypothesis 1 has to be rejected. Comparing the time results from Boerse5 with the other two groups it may seem strange that they are almost as fast as Boerse0. On a closer look, this can be explained: In Boerse5 there was only half as many of changes than in Boerse3 or Boerse0. So longer program comprehension time was compensated by less work to be done. We can conclude from this that Boerse5 was harder to understand than Boerse0 but we can not make a decision between Boerse3 and Boerse5. Considering the points obtained and the defects made, we see a trend that with deeper inheritance the number of defects is growing.

So for this work task we have to reject both hypotheses.

	Variable	Groups	Range min ... max	mean1	mean2	means difference (90% confid.)	significance <i>p</i>
1	JT time	G0, G3	5...76	15.9	17.5	-51%...24%	0.381
2	-	G0, G5	7...32	15.9	16.8	-23%...13%	0.313
3	-	G3, G5	5...76	17.5	16.8	-28%...46%	0.470
4	T1 time	G0, G3	32...194	71.1	88.1	-41%...3.0%	0.075
5	-	G0, G5	25...194	71.1	69.4	-16%...24%	0.447
6	-	G3, G5	25...185	88.1	69.4	4.4%...51%	0.023
7	-(10%)	G0, G3	32...125	62.7	78.2	-37%...-3.4%	0.022
8	-	G0, G5	25...85	62.7	67.0	-19%...5.9%	0.200
9	-	G3, G5	25...125	78.2	67.0	-2.2%...36%	0.073
10	-(20%)	G0, G3	32...107	60.0	72.2	-34%...-0.2%	0.046
11	-	G0, G5	25...82	60.0	64.7	-20%...5.9%	0.172
12	-	G3, G5	25...107	72.2	64.7	-6.5%...31%	0.153
13	T1 points	G0, G3	0...4	4.0	3.6	(1.5%...23%)	(0.000)
14	-	G0, G5	0...4	4.0	3.2	(9.3%...42%)	(0.000)
15	-	G3, G5	0...4	3.6	3.2	-7.1%...33%	0.142
16	T1 productivity	G0, G3	0...7	3.8	3.0	2.7%...54%	0.030
17	-	G0, G5	0...7	3.8	2.8	14%...64%	0.003
18	-	G3, G5	0...6	3.0	2.8	-20%...35%	0.339
19	T2 time	G0, G3	64...200	115.7	132.2	-24%...-0.8%	0.038
20	-	G0, G5	78...175	115.7	134.8	-23%...-5.1%	0.005
21	-	G3, G5	64...200	132.2	134.8	-14%...10%	0.393
22	-(10%)	G0, G3	64...173	112.1	124.1	-21%...1.3%	0.071
23	-	G0, G5	78...162	112.1	130.4	-23%...-4.7%	0.006
24	-	G3, G5	64...173	124.1	130.4	-16%...6.2%	0.239
25	-(20%)	G0, G3	64...152	108.4	118.6	-19%...2.4%	0.097
26	-	G0, G5	78...150	108.4	126.6	-23%...-5.2%	0.005
27	-	G3, G5	64...152	118.6	126.6	-18%...5.2%	0.185
28	T2 points	G0, G3	0...8	6.3	6.0	-15%...28%	0.313
29	-	G0, G5	0...8	6.3	6.4	-18%...17%	0.495
30	-	G3, G5	0...8	6.0	6.4	-26%...13%	0.299
31	T2 productivity	G0, G3	0...6	3.4	2.9	-11%...45%	0.156
32	-	G0, G5	0...6	3.4	3.0	-11%...36%	0.178
33	-	G3, G5	0...5	2.9	3.0	-29%...22%	0.412

Table 3.2: Results of the JAKK subjects; (left to right:) line number, name of variable, groups that are compared (group1, group2), smallest and biggest value in the combined sample, arithmetic average of group1, arithmetic average of group2, 90% confidence interval I for the difference group1 – group2 (measured in percent of group2), significance p of this difference (one sided). I and p were computed using resampling with 10000 trials. Parentheses around intervals and p values indicate dubious results due to zero variance in at least one of the samples. JT time means the time for answering the Java-test on the first questionnaire. T1 and T2 indicate tasks 1 and 2, respectively. (10%) or (20%) means the result that is obtained if 10 or 20 percent are cut off of the right side of the distribution to eliminate possible outliers.

Task 2 – “Time Interval Price Display and Gain/Loss Display”

Task 2 combines Tasks 2A and 2B as discussed in Section 2.8. We gathered timestamps only at the start and the end of the task but not between the two parts.

Time: For this task the time of group JG0 is significantly shorter than for the other two groups, on average 15 percent less: $t(JG0) < t(JG3) \approx t(JG5)$. Figures 3.4 and 3.5 show the time distribution for all subjects and by treatment group. In the group of Boerse3 some data points may be considered outliers so we recalculated the statistics with 10 and 20% cut off of the right side of the distribution. But the results did not change much. For details, see Table 3.2, lines 19 - 27.

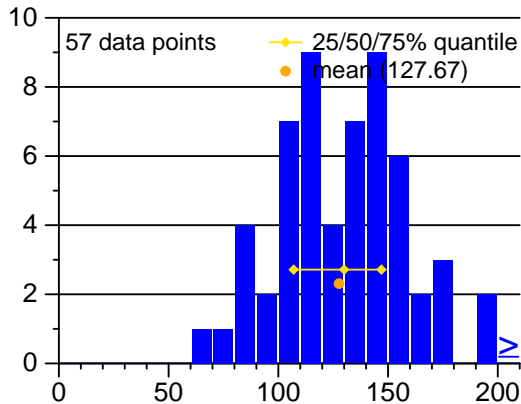


Figure 3.4: [JAKK] Distribution of the time required for all subjects for task “Time Interval Price Display and Gain/Loss Display”.

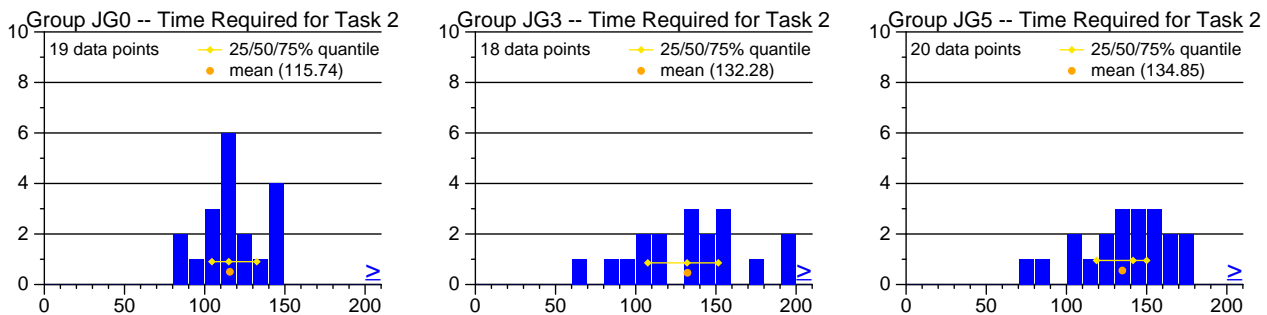


Figure 3.5: Distribution of the time required by group.

Points: No significant differences were observed between the groups, see Figures 3.6, 3.7, 3.8, and Table 3.2, lines 28 - 30.

Incidence counts: The number of correct solutions does not differ significantly between the groups, see Tables 3.3.

Types of solutions: For this task it is interesting to know which type of solution the subjects chose — whether they preserved the design or “destroyed” its structure.

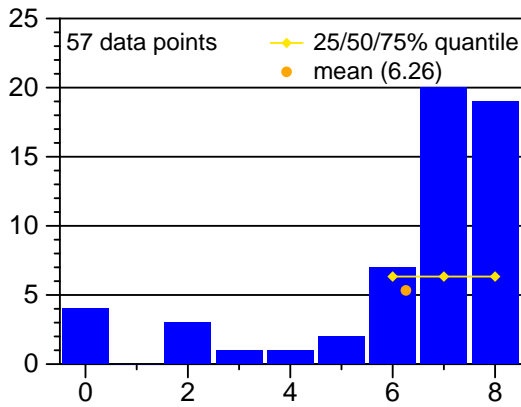


Figure 3.6: [JAKK] Distribution of the points for all subjects; the maximum achievable is 8.

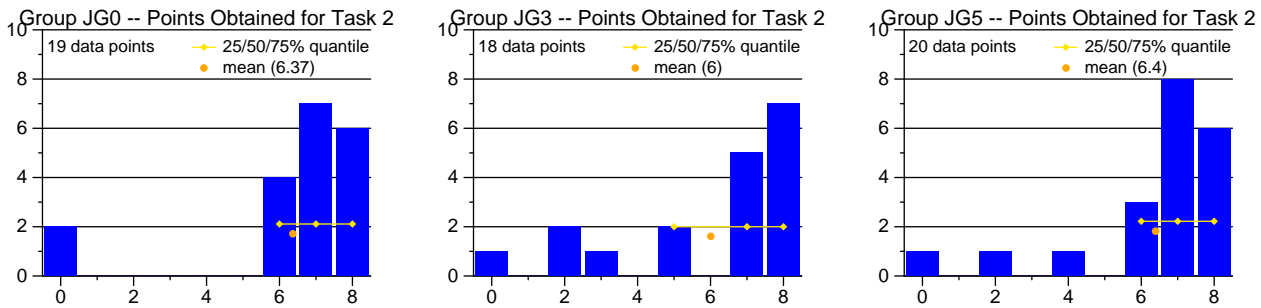


Figure 3.7: Distribution of points obtained by group.

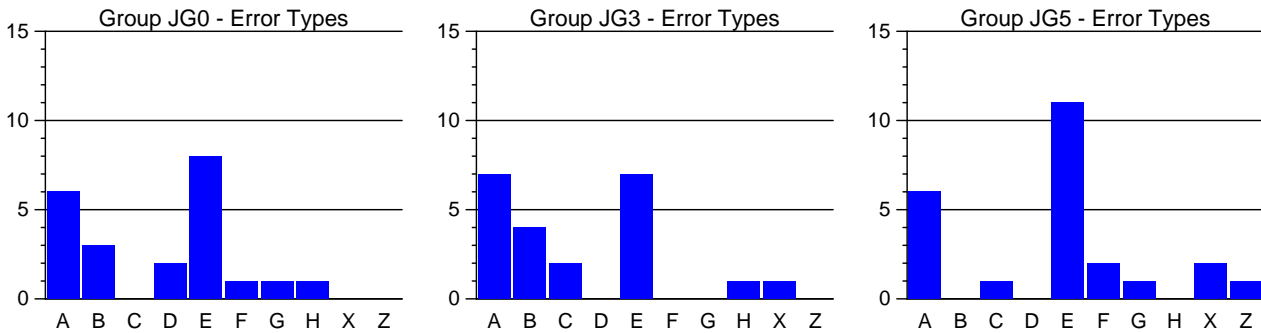


Figure 3.8: Frequency of different errors for task “Time Interval Price Display and Gain/Loss Display”. Codes: A = everything OK; B = one/both of the tables unimplemented; C = only dialog implemented; D = dialog not implemented; E = start day missing; F = end day missing; G = records appear twice; H = uses KURSE instead of KASSAKURSE; X = other; Z = nothing done but delivered.(A solution can have more than one error class.)

group	no. of subjects	fraction of correct solutions	groups	χ^2	p	Fisher exact p
JG0	19	37%	JG0 - JG3	0.044	0.83	1
JG3	18	39%	JG0 - JG5	0.013	0.91	0.74
JG5	20	30%	JG3 - JG5	0.055	0.81	0.73

Table 3.3: [JAKK] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 2.

Task 2A, JG0: 17 out of 19 subjects from group JG0 chose the solution type as described in Section 2.8.3 and thus preserved the structure: A new class is created as a copy of an existing table class. Only 2 subjects inherited from an existing class and overwrote the functionality that had to be changed.

Task 2B, JG0: 15 out of 17 created the solution by duplicating and modifying an existing class. Only 2 subjects inherited from the class implemented in Task 2A.

Task 2A, JG3: 11 out of 18 subjects retained the structure and inherited from the abstract table class while 5 of the subjects inherited from a concrete class that is derived from the abstract table class and 2 subjects inherited from the abstract common superclass of the charts and the tables.

Task 2B, JG3: 7 subjects out of 15 inherited from the abstract table class, 2 subjects inherited from the class implemented in Task 2A, 5 subjects inherited from a concrete table class that has similar functionality but this destroys the design structure (the design decision was: only inheriting from abstract classes), and one subject inherited from the abstract class that is the superclass of the charts and the tables.

Task 2A, JG5: 18 out of 20 subjects from group JG5 extended the abstract class for time interval table displays and added the selection dialog functionality for user defined time intervals, 1 subject extended a concrete table class, and 1 subject extended the Java library class Dialog and tried to implement the class from scratch.

Task 2B, JG5: 9 subjects out of 18 inherited from the class implemented in the first part and the other 9 subjects inherited from a concrete table class that has similar functionality.

	Task 2A		Task2B	
	group size	fraction of subjects retaining the structure	group size	fraction of subjects retaining the structure
JG0	19	89%	17	88%
JG3	18	61%	15	47%
JG5	20	90%	18	100%

Table 3.4: [JAKK] Fraction of subjects that retained the design structure. The group sizes differ between Task 2A and Task 2B because of mortality in the groups.

Conclusion: Differences in time required are quite small between groups JG3 and JG5; JG5 required on average 2 percent more time. This is plausible: For this task it was only necessary to find the differences between the classes “MonatsTabelle [MonthTable]” and “Chart”. In Boerse3 as well as in Boerse5 the code had to be found across several hierarchy levels; in Boerse5 a few more superclasses had to be taken into consideration. Once the differences between the two classes were detected, the exercise could be solved easily by inheritance and/or copy-and-paste in all three groups. The actual changes were similar for all for all three groups, so we can assume that the time differences result from program comprehension. As in Task 1 we can conclude that with increasing inheritance depth comprehension tends to become more difficult and the time required is increasing correspondingly.

3.2.4 Performance on the tasks - Informatik II

Keep in mind that the tasks are not the same as for the JAKK course. Task 1 from the JAKK course is skipped and Task 2 is split into two tasks, called Task 2A and 2B. We did this because of the time constraints described in Section 2.7.

Task 2A – “Time Interval Price Data Display”

	Variable	Groups	min ... max	mean1	mean2	means difference (90% confid.)	significance p
1	JT time	G0, G3	5 ... 20	12.7	13.1	-19%... 11%	0.349
2	-	G0, G5	5 ... 25	12.7	13.2	-19%... 11%	0.337
3	-	G3, G5	7 ... 25	13.1	13.2	-16%... 15%	0.487
4	T2A time	G0, G3	82 ... 305	152.3	154.0	-19%... 16%	0.456
5	-	G0, G5	93 ... 330	152.3	180.8	-31%... -0.9%	0.038
6	-	G3, G5	82 ... 330	154.0	180.8	-30%... 0.5%	0.055
7	-(10%)	G0, G3	82 ... 187	138.7	139.8	-13%... 11%	0.460
8	-	G0, G5	93 ... 238	138.7	168.3	-28%... -7.0%	0.003
9	-	G3, G5	82 ... 238	139.8	168.3	-29%... -5.6%	0.007
10	-(20%)	G0, G3	82 ... 179	133.6	136.4	-14%... 9.9%	0.386
11	-	G0, G5	93 ... 219	133.6	161.1	-28%... -6.7%	0.003
12	-	G3, G5	82 ... 219	136.4	161.1	-27%... -4.0%	0.012
13	T2A points	G0, G3	0 ... 8	5.9	5.4	-20%... 39%	0.295
14	-	G0, G5	0 ... 8	5.9	5.0	-14%... 47%	0.185
15	-	G3, G5	0 ... 8	5.4	5.0	-27%... 38%	0.374
16	T2A productivity	G0, G3	0 ... 5	2.6	2.4	-26%... 47%	0.319
17	-	G0, G5	0 ... 5	2.6	1.9	-0.3%... 81%	0.051
18	-	G3, G5	0 ... 5	2.4	1.9	-16%... 72%	0.144
19	T2B time	G0, G3	10 ... 84	27.0	30.9	-47%... 18%	0.263
20	-	G0, G5	11 ... 64	27.0	19.1	12%... 72%	0.006
21	-	G3, G5	10 ... 84	30.9	19.1	19%... 111%	0.004
22	-(10%)	G0, G3	10 ... 41	22.9	24.0	-28%... 18%	0.385
23	-	G0, G5	11 ... 38	22.9	18.2	4.2%... 47%	0.021
24	-	G3, G5	10 ... 41	24.0	18.2	5.0%... 58%	0.021
25	-(20%)	G0, G3	10 ... 41	20.9	22.4	-30%... 15%	0.311
26	-	G0, G5	11 ... 31	20.9	17.3	0.9%... 41%	0.042
27	-	G3, G5	10 ... 41	22.4	17.3	3.8%... 56%	0.027
28	T2B points	G0, G3	0 ... 3	2.3	2.2	-24%... 30%	0.422
29	-	G0, G5	0 ... 3	2.3	2.5	-30%... 19%	0.349
30	-	G3, G5	0 ... 3	2.2	2.5	-34%... 17%	0.298
31	T2B productivity	G0, G3	0 ... 60	14.5	16.0	-79%... 59%	0.404
32	-	G0, G5	0 ... 60	14.5	20.9	-81%... 21%	0.167
33	-	G3, G5	0 ... 60	16.0	20.9	-77%... 32%	0.244

Table 3.5: Results of the Informatik II subjects; (left to right:) line number, name of variable, groups that are compared (group1, group2), smallest and biggest value in the combined sample, arithmetic average of group1, arithmetic average of group2, 90% confidence interval I for difference group1 – group2 (measured in percent of group2), significance p of the difference (one sided). I and p were computed using resampling with 10000 trials. JT time means the time for answering the Java-test on the first questionnaire. T2A means Task 2A, “Time Interval Price Data Display”, T2B means Task 2B, “Time Interval Gain/Loss Display”, (10%) or (20%) means the result that is obtained if 10 or 20 percent are cut off of the right side of the distribution to eliminate possible outliers.

Time: Regarding the average time of the groups there is one obvious conclusion: IG5 needed more time (≈ 18 percent more) than IG3 and IG0: $t(IG0) \approx t(IG3) < t(IG5)$. The Histograms 3.9 and 3.10 suggest some data points to be outliers. However, by cutting the right end (10% and 20%) off of the distribution of each group, we still get similar results. Subjects of IG5 needed significantly much more time than the other two groups.

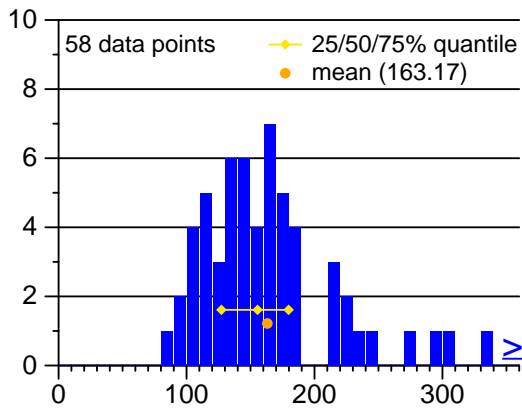


Figure 3.9: [Informatik II] Distribution of the time required for the task “Time Interval Price Display”.

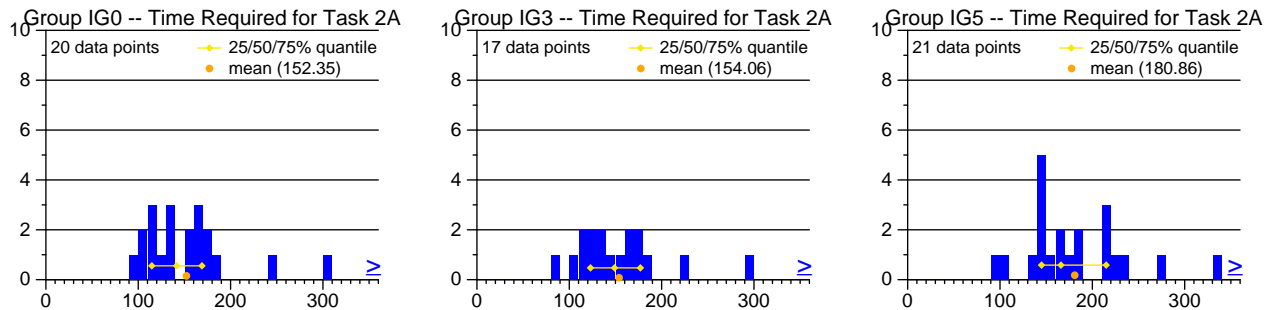


Figure 3.10: [Informatik II] Distribution of the time required for the task “Time Interval Price Display” for each of the three treatment groups.

Points: No significant quality differences can be observed between the three groups. The number of correct solutions in each group is not significantly different, either; see Figures 3.11, 3.12, 3.13 and Table 3.6.

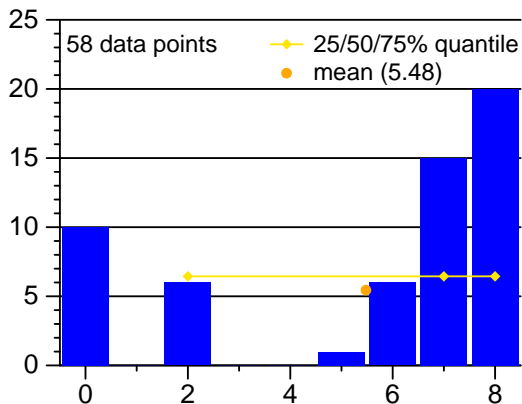


Figure 3.11: [Informatik II] Distribution of the points for all subjects; the maximum achievable is 8.

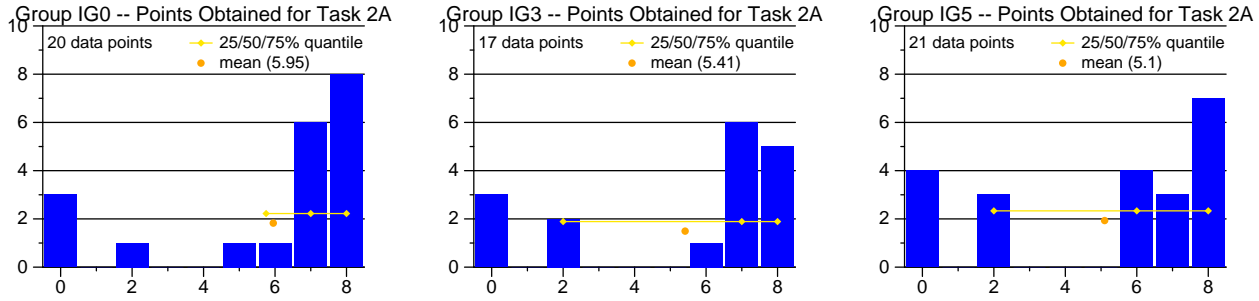


Figure 3.12: [Informatik II] Distribution of points obtained by group.

group	no. of subjects	fraction of correct solutions	groups	χ^2	p	Fisher exact p
IG0	20	40%	IG0 - IG3	0.107	0.74	0.73
IG3	17	29%	IG0 - IG5	0.014	0.91	0.75
IG5	21	33%	IG3 - IG5	0.009	0.93	1

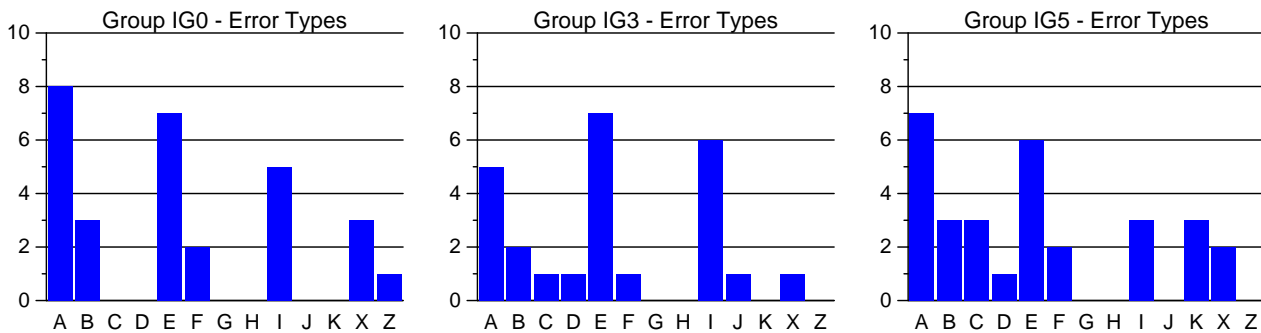
Table 3.6: [Informatik II] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 2A.

Figure 3.13: [Informatik II] Frequency of different errors for task "Time Interval Price Display". Codes: A = everything OK; B = table unimplemented, only checkbox appears; C = only dialog implemented, no table appears; D = dialog unimplemented, some other table with new time range appears; E = start day missing; F = end day missing; G = records appear twice; H = uses KURSE instead of KASSAKURSE; I = requires two-digit years; J = start day wrong (e.g., wrong year), but not just E; K = end day wrong (e.g., wrong year), but not just F; X = other; Z = nothing done but delivered. (A solution can have more than one error class.)

Type of solutions: For this task, it is interesting to know which type of solution the subjects chose – whether they preserved the design or “destroyed” its structure.

Task 2A, IG0: 17 out of 20 chose the solution type as described in Section 2.8. A new class is created as a copy of an existing table class. Only 2 subjects inherited from an existing class and overwrote the functionality that had to be changed. (1 subject did not deliver a solution.)

Task 2A, IG3: 14 out of 17 subjects retained the structure and inherited from the abstract table class, 2 subjects retained the structure, but inherited from the abstract class that is the superclass of the charts and the tables. Only 1 subject inherited from a concrete table display class.

Task 2A, IG5: 17 out of 21 subjects extended the abstract class for fixed time interval table displays and added the selection dialog functionality for user defined time intervals, 3 subjects also extended an abstract class but only for table displays which is the superclass for the fixed time interval table display. Only 1 subject inherited from the abstract class that is the superclass of the charts and the tables.

	group size	fraction of subjects retaining the structure
JG0	20	85%
JG3	17	82%
JG5	21	81%

Table 3.7: [Informatik II] Fraction of subjects that retained the design structure.

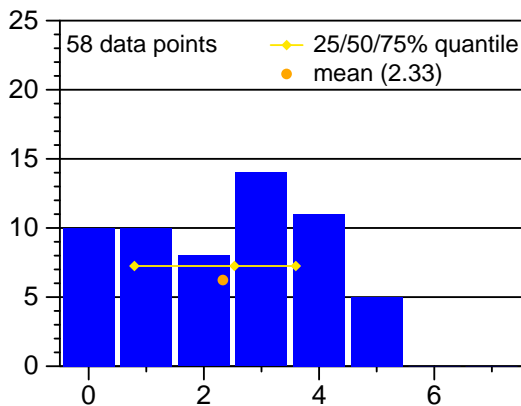


Figure 3.14: [Informatik II] Distribution of the productivity for all subjects.

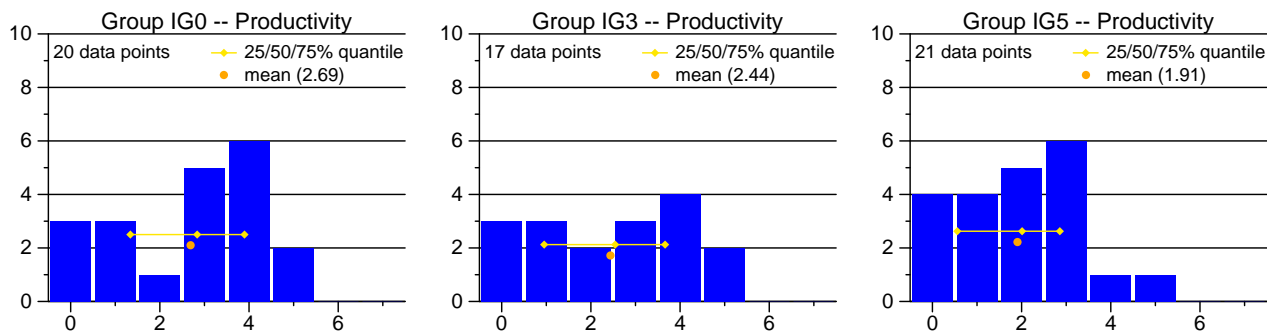


Figure 3.15: [Informatik II] Distribution of the productivity by group.

Conclusion: In contrast to the work task of the JAKK experiment with its “warm-up” task 1, all program comprehension must be gained in this task. But we still observe the same trend as in the JAKK run: with an increasing inheritance hierarchy the maintenance time is increasing, too. The quality of the solutions does not differ much between the groups so we have to reject both of our hypotheses.

Task 2B – “Time Interval Gain/Loss Display”

Task 2B differs from all other tasks in this experiment – the subjects just worked on a quite similar task and gained almost the complete program understanding required for this task beforehand in Task 2A. So the time results might give information about the working time required for the different program versions.

Caveat: The time results presented for this task are quite unreliable. When looking on the mortality it is salient that the mortality in group IG5 is higher than in the two other groups (IG5 = 29%; IG3 = 12%; IG0 = 10%). The table below lists for each subject the time for the Task 2A so that it is obvious that more slow subjects quit in IG5 than in IG3 and IG0. From these considerations we expect that more faster subjects remained in the group IG5 and that the groups are not balanced anymore.

Conclusion: We may conclude for this task that more subjects working on a program with a deep inheritance structure found it too difficult than in the other two groups.

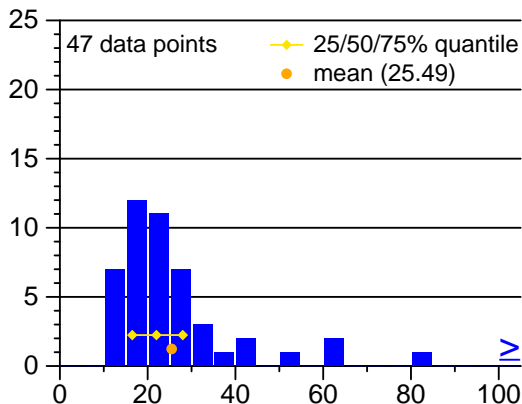


Figure 3.16: [Informatik II] Distribution of the time required for Task 2B for all subjects.

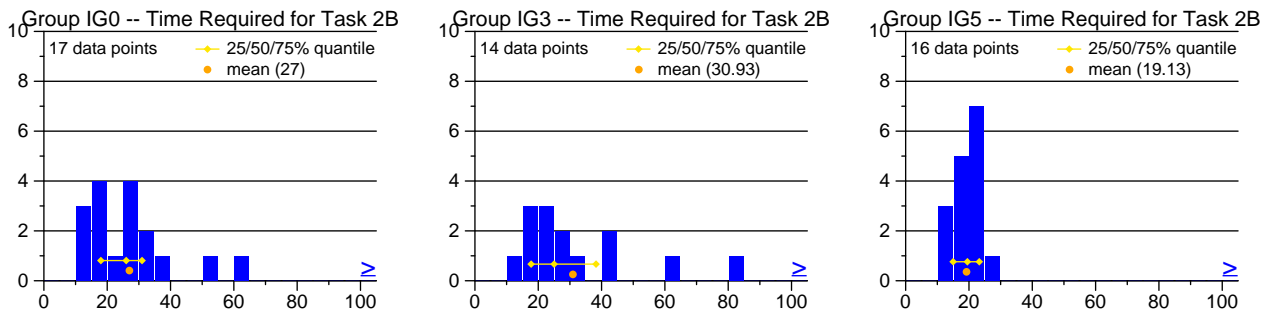


Figure 3.17: [Informatik II] Distribution of the time required by group.

	IG0	IG3	IG5
1	96	82	93
2	102	101	109
3	108	110	138
4	110	112	140
5	111	123	143
6	116	126	145
7	123	131	147
8	131	135	149
9	133	149	157
10	133	***157***	166
11	***150***	163	166
12	154	165	179
13	162	177	***180***
14	167	179	182
15	167	187	212
16	175	223	215
17	179	299	219
18	180		220
19	245		238
20	305		270
21			330

Table 3.8: [Informatik II] Times required for each subject for Task 2A. Boldfaced and underlined entries are for subjects which gave up in the second task. Times marked with *** are the time closest to the mean in the group. From this table we can see that the mortality is unbalanced.

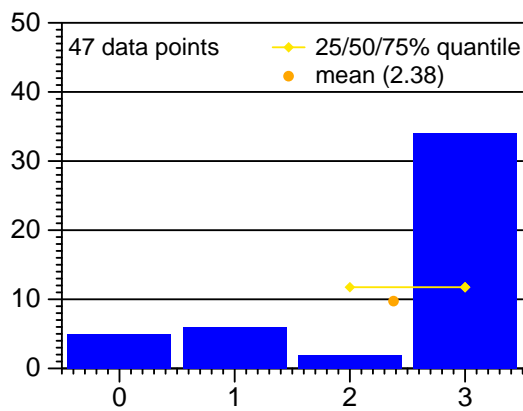


Figure 3.18: [Informatik II] Distribution of the obtained points for all subjects; the maximum is 3.

group	no. of subjects	fraction of correct solutions	groups	χ^2	p	Fisher exact p
IG0	20	35%	IG0 - IG3	0.16	0.69	0.5
IG3	17	24%	IG0 - IG5	0.011	0.92	0.74
IG5	21	29%	IG3 - IG5	0.0	0.98	1

Table 3.9: [Informatik II] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 2B.

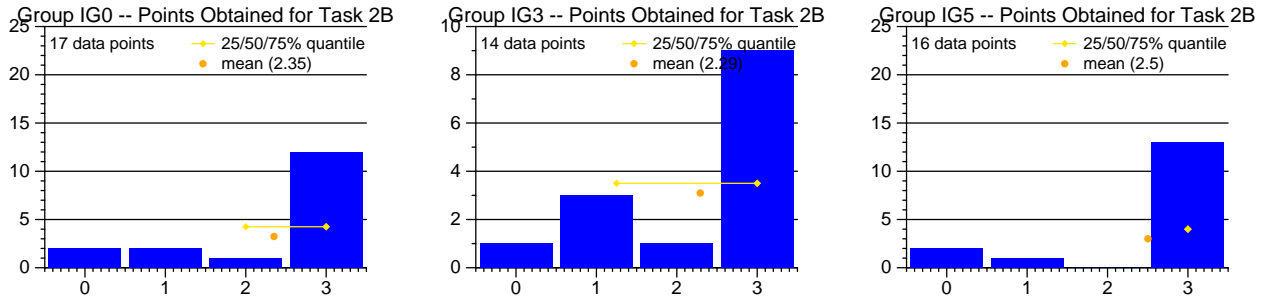


Figure 3.19: [Informatik II] Distribution of the points obtained by group.

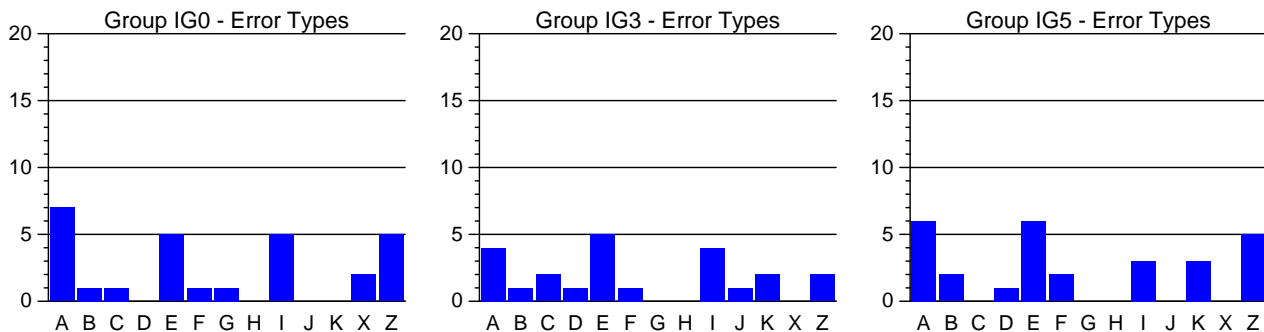


Figure 3.20: [Informatik II] Frequency of different errors for task "Time Interval Gain/Loss Display". Codes: A = everything OK; B = table unimplemented, only checkbox appears; C = only dialog implemented, no table appears; D = dialog unimplemented, some other table with new time range appears; E = start day missing (but was correct in 2A, otherwise no deduction); F = end day missing (but was correct in 2A, otherwise no deduction); G = records appear twice (but was correct in 2A, otherwise no deduction); H = uses KURSE instead of KASSAKURSE (but was correct in 2A, otherwise no deduction); I = requires two-digit years; J = start day wrong (e.g., wrong year), but not only E; K = end day wrong (e.g., wrong year), but not only F; X = other; Z = nothing done but delivered. (A solution can have more than one error class.)

3.3 Subjects' experience

In the final questionnaire we wanted to learn about our subjects' impressions: how they assess, for example, the program structure, the use of inheritance, the difficulty of the tasks, their concentration, etc.

3.3.1 Inheritance use

With the first question we wanted to know how the subjects assess the use of inheritance. Our subjects had five choices from "inheritance is used much too much" to "inheritance is used much too little". The median of the answers from the JAKK subjects were "inheritance is used too little" for group JG0 and "inheritance is used in the right degree" for the groups JG3 and JG5. The distribution is shown in Figure 3.21.

There is an obvious difference between the performance of the subjects and the subjective impression they have. Regarding that JG0 was on average the best and fastest group, it is curious that most of these subjects graded the inheritance use as "too little" or "much too little". The discrepancy between subjective impression and objective results demonstrates how necessary controlled experiments are.

We have similar answers from the Informatik II subjects, except for IG0. These subject graded the inheritance use as "used in the right degree" (median).

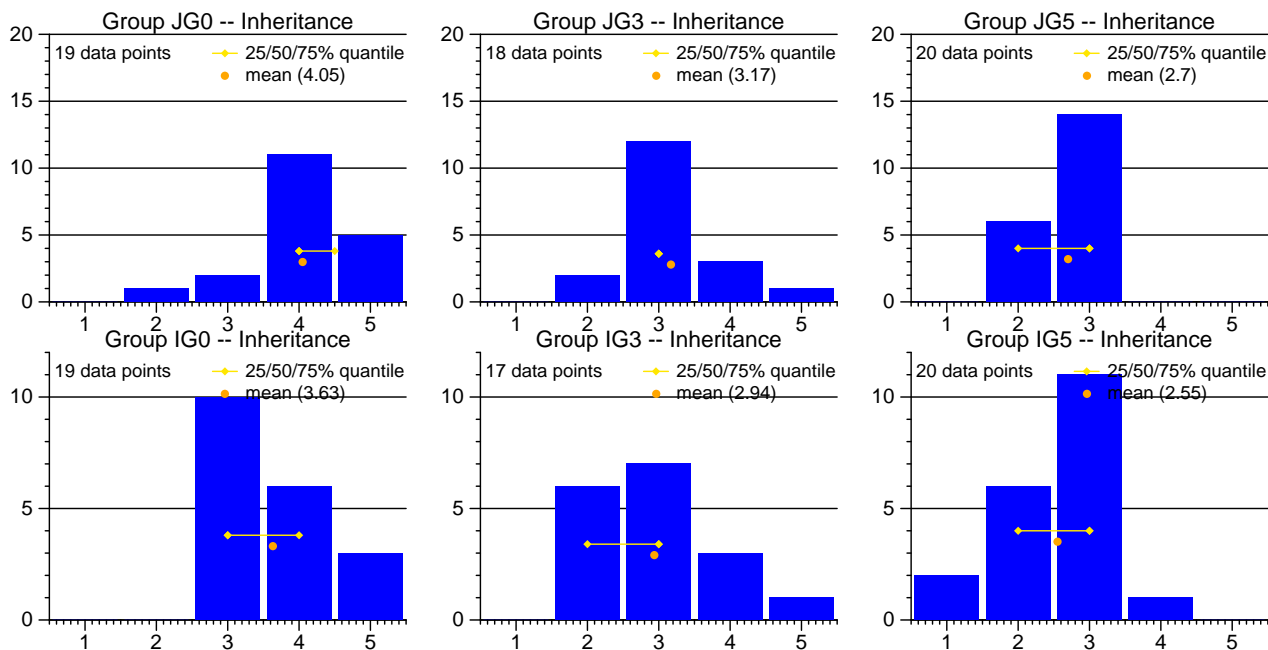


Figure 3.21: Answers from the JAKK subjects (first row) and for the Informatik II subjects (second row) on how they assess the use of inheritance, split into treatment groups. The subjects had the following choices: Inheritance is used much too much (1), is used too much (2), is used in the right degree (3), is used too little (4), or is used much too little (5).

3.3.2 Program structure

In the next question the subjects were asked how they rated the quality of the program structure for a program of this size. They had five choices from "very clear structure" to "unclear structure". The median of the answers of

the JAKK and Informatik II subjects were between “clear structure” and “medium structure”. The distribution is shown in Figure 3.22. We expected that the group JG0 would grade the structure as less clear than the other groups because of the code replication and the resulting code extension. However, comparing the actual subjective impressions reported with the objective results, it is odd that more than half of the subjects of JG5 graded the structure as a “very clear structure” or “clear structure”, while of JG3 and JG0 less than half graded it as “clear or very clear structure”, even though they received better results. Probably the judgement was mostly based on the OMT diagrams, where JG5 clearly appears the most structured. Overall, the Informatik II subjects judged the structure more realistically than the JAKK subjects.

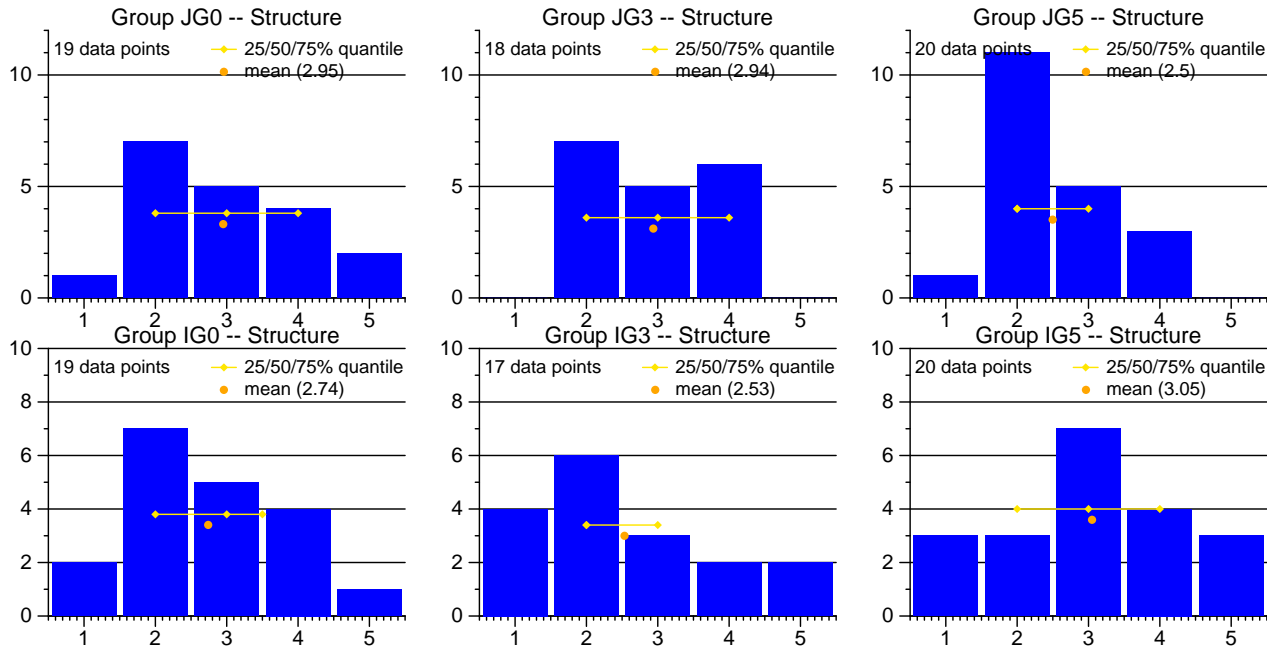


Figure 3.22: Answers from the JAKK subjects (first row) and for the Informatik II subjects (second row) on how they assess the structure of the program, split into treatment groups. The subjects had the following choices: very clear structure (1), clear structure (2), medium structure (3), less clear structure(4), unclear structure (5).

3.3.3 Simplicity of the tasks

This question investigated how difficult the subjects found the tasks. We expected from the time results obtained and the comprehension steps counted that the group with the flat program version found the tasks easier than the other groups but no large differences between the groups were observed. The distributions are shown in Figures 3.23 to 3.24.

3.3.4 Concentration of the subjects

How well the subjects were subjectively able to concentrate on the tasks is shown in Figures 3.25 to 3.26. The median of all groups was that their concentration was “high”, except for the inheritance groups of the JAKK experiment in Task 2. They probably realized that they had problems with tracing the methods to gain program understanding and they put it down to their concentration ability.

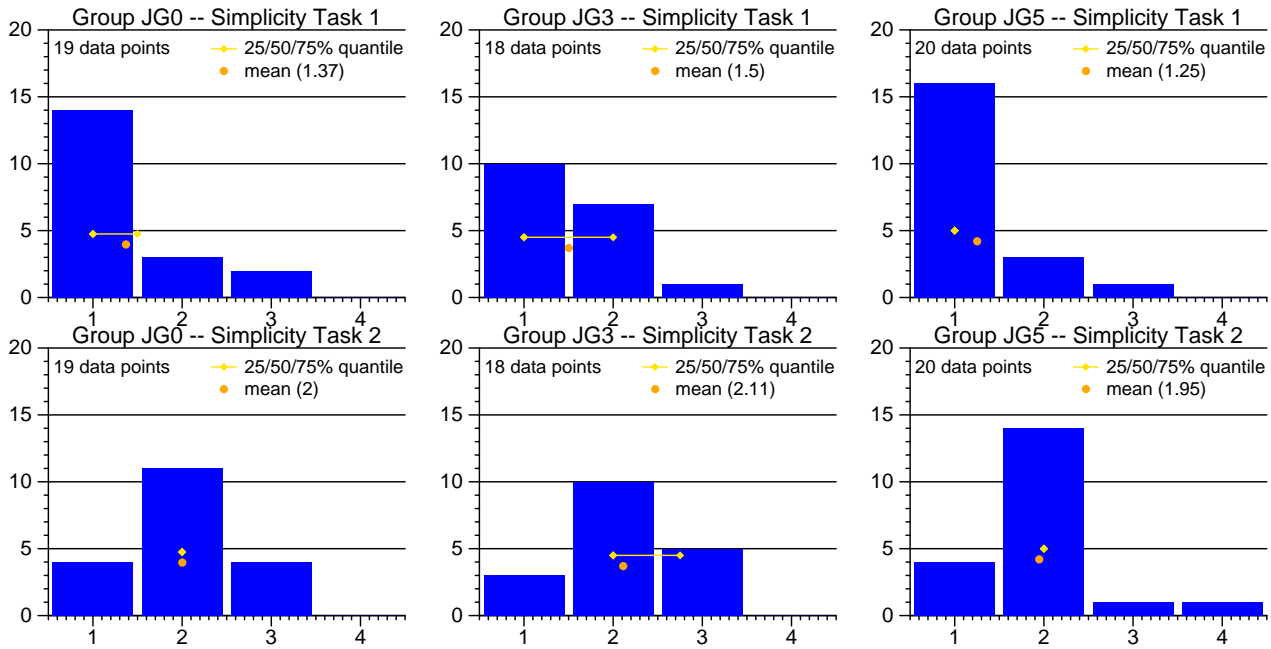


Figure 3.23: Answers from the JAKK subjects on how they assess the simplicity of Task 1 (first row) and Task 2 (second row) of the experiment, split into treatment groups. The subjects had the following choices: pretty simple (1), not quite so simple (2), pretty difficult (3), difficult (4).

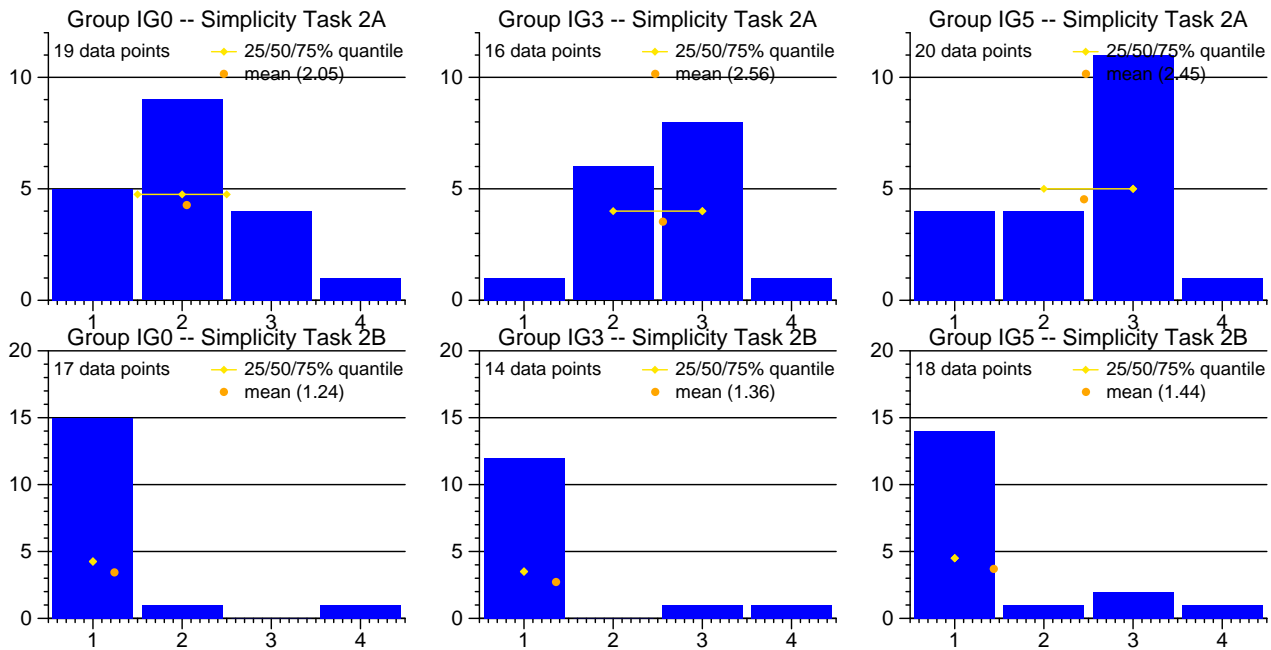


Figure 3.24: Answers from the Informatik II subjects on how they assess the simplicity of Task 2A (first row) and Task 2B (second row) of the experiment, split into treatment groups. For a legend see Figure 3.23.

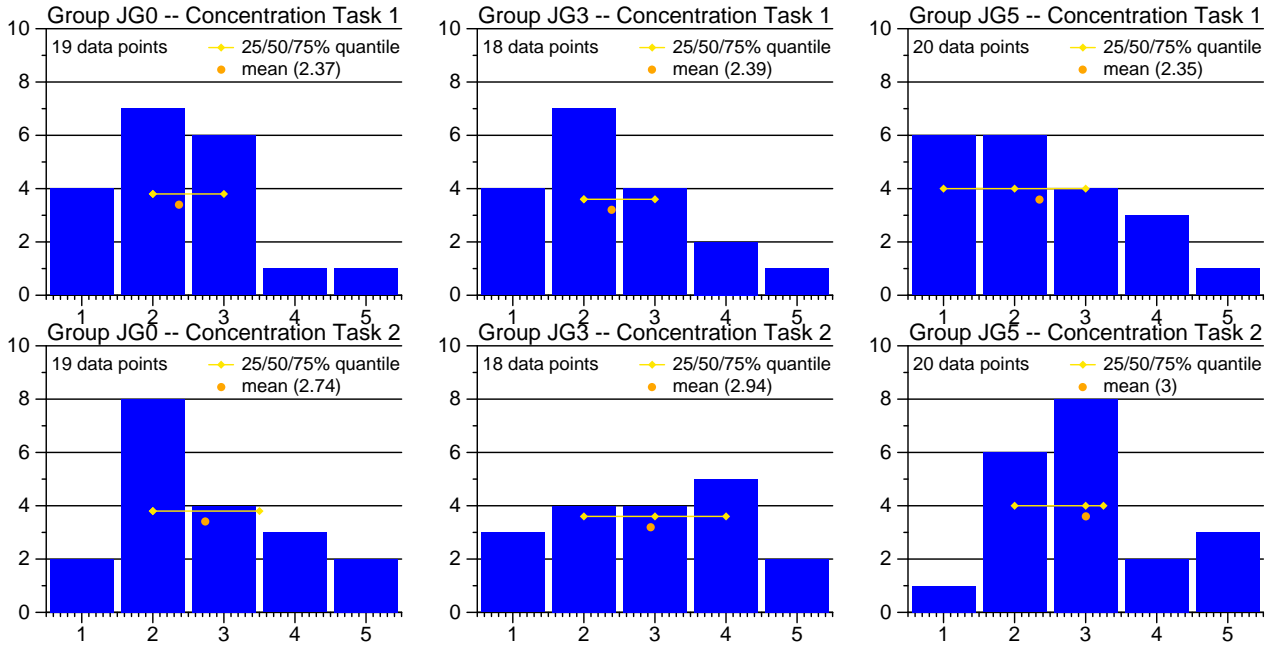


Figure 3.25: Answers from the JAKK subjects on how they assess their ability of concentration during Task 1 (first row) and Task 2 (second row), split into treatment groups. The subjects had the following choices: very high (1), high (2), OK (3), not so high (4), low (5).

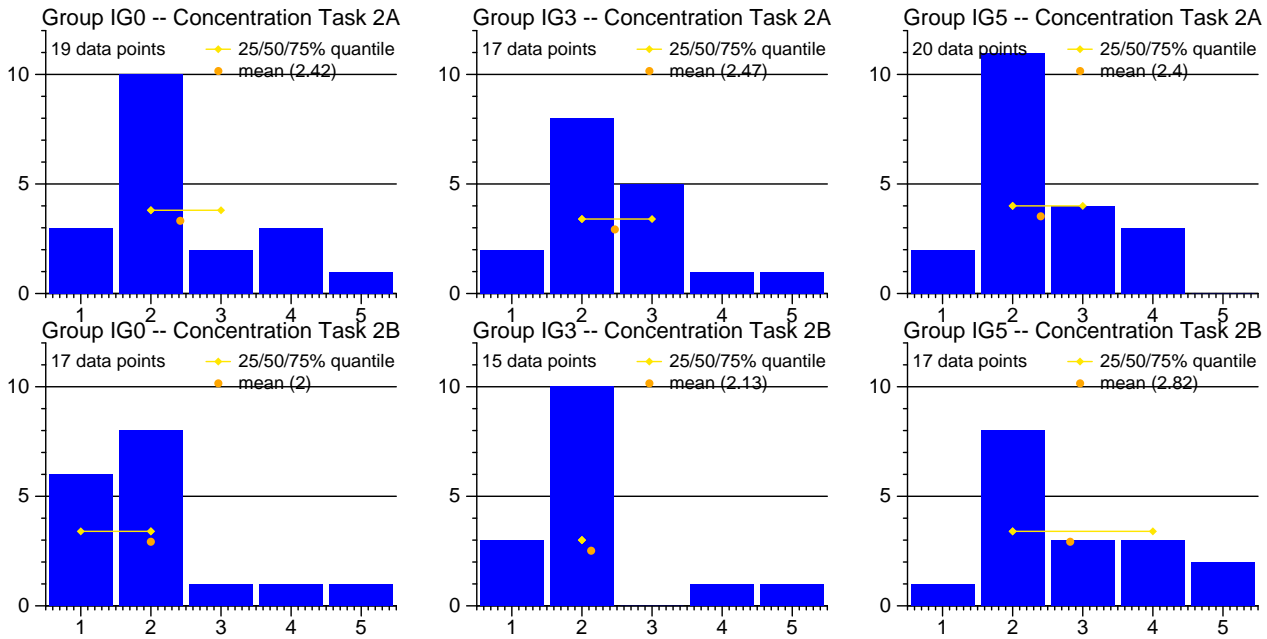


Figure 3.26: Ditto for the Informatik II subjects for Task 2A (first row) and Task 2B (second row).

3.3.5 Quality assessment

The subjects were asked how they subjectively assess the quality of their solutions to the two tasks. For each task they had the following choices of answers: no more errors or omissions (1), at most one error or omission (2), probably several errors or omissions (3), or don't know (4). G0 had most confidence in correctness(!) except for IG5 in Task 2B (but in Task 2B mortality distorts these results, anyway). This is unexpected from the hypotheses but expected from examining the tasks: G0 had all code local in a class and that resulted in a better overview of what they were doing. The distributions are shown in the Figures 3.27 and 3.28.

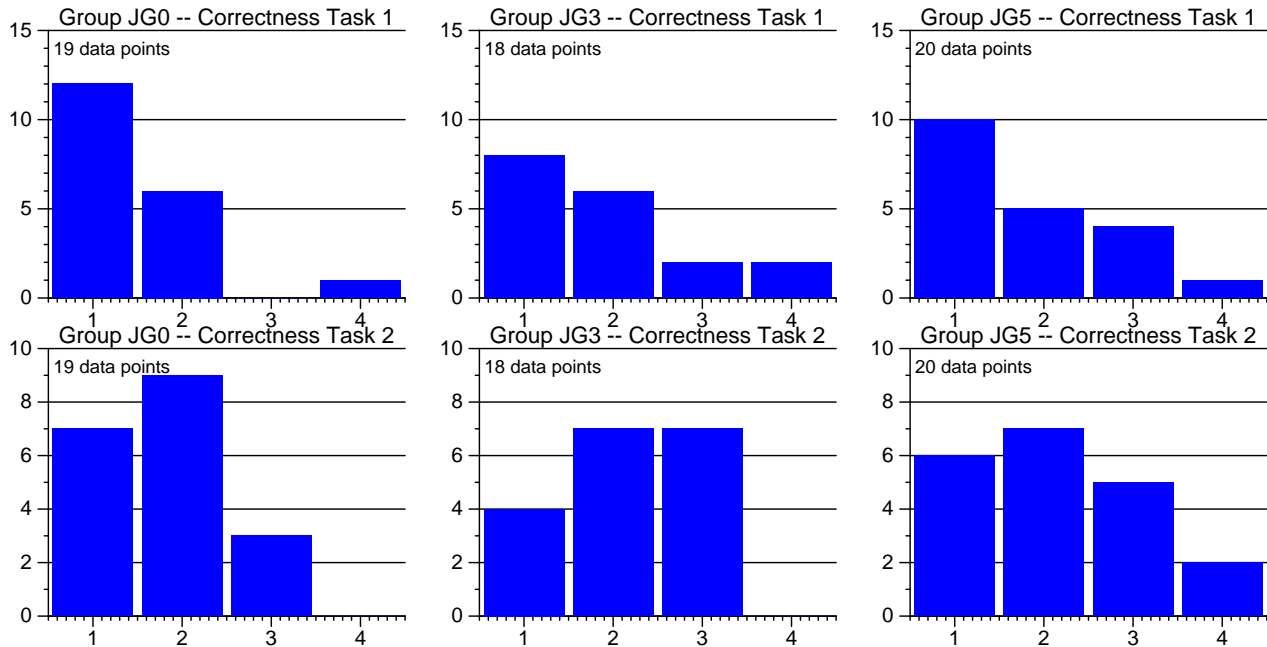


Figure 3.27: Answers from the JAKK subjects on how they assess their solutions for Task 1 (first row) and Task 2 (second row), split into treatment groups. The subjects had the following choices: no more errors or omissions (1), at most one error or omission (2), probably several errors or omissions (3), don't know (4).

3.3.6 Usefulness of OO knowledge

We asked the subjects whether they think that their OO knowledge was helpful. We expected that the subjects found it the more helpful the more inheritance is used in the program. They had the following choices of answers: no, not at all (1), only a little (2), can't decide (3), yes, somewhat (4), or yes, very much (5). We expected to observe a huge difference between the flat program version and the deep inheritance version because no inheritance knowledge is needed in the flat program version but for understanding Boerse3 and Boerse5 this knowledge is essential. We were surprised that no large differences were observed between the groups, see Figures 3.29 and 3.30. We speculate that to a large degree the concept of "OO knowledge" was equated with general knowledge of Java and its API, which is roughly as relevant in all three groups.

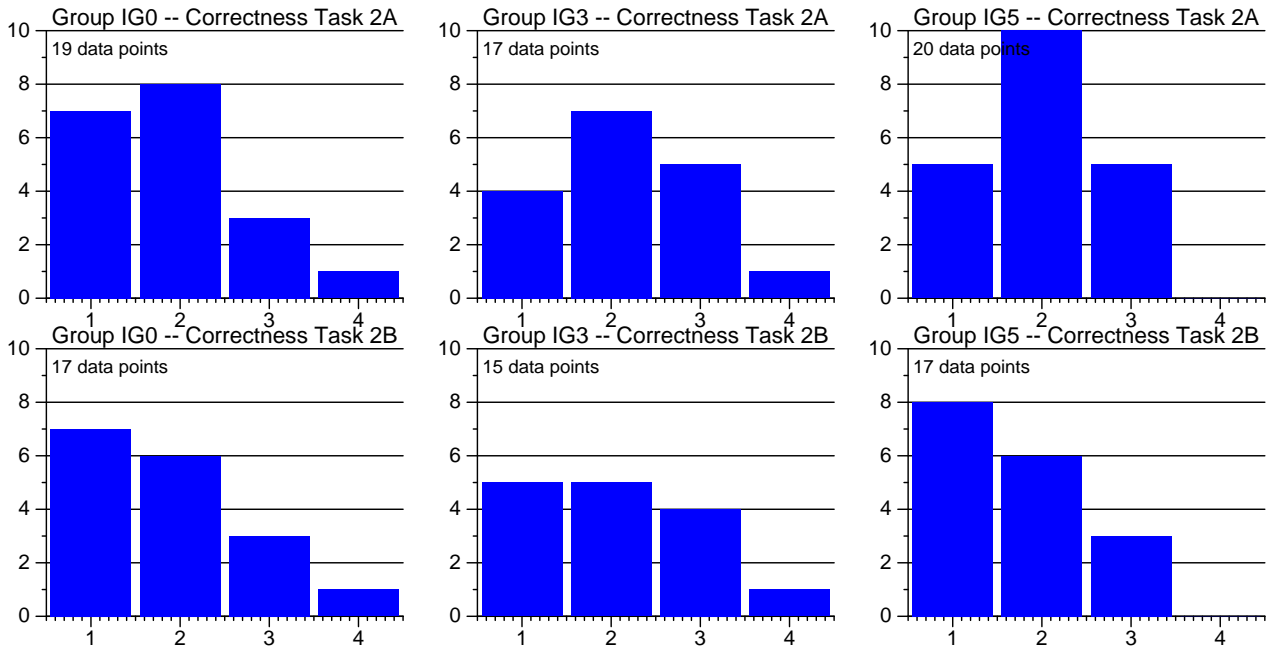


Figure 3.28: Ditto for the Informatik II subjects for Task 2A (first row) and Task 2B (second row).

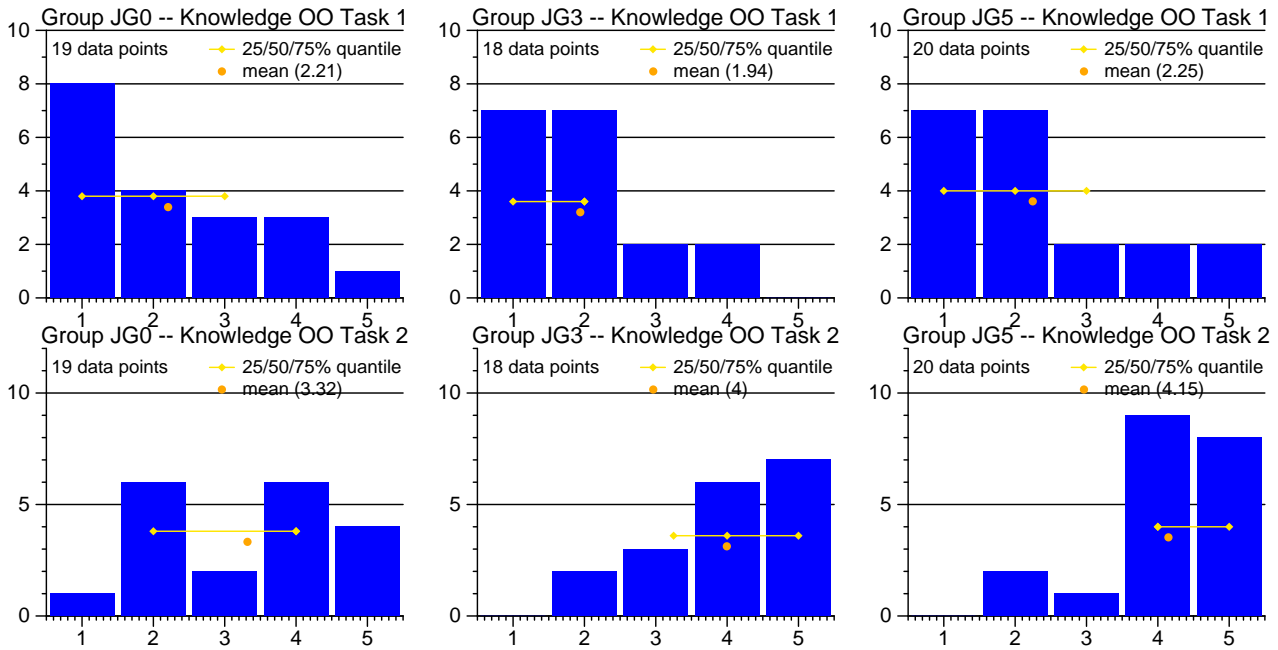


Figure 3.29: Answers from the JAKK subjects on how they assess the usefulness of OO knowledge for Task 1 (first row) and Task 2 (second row), split into treatment groups. The subjects had the following choices: no, not at all (1), only a little (2), can't decide (3), yes, somewhat (4), yes, very much (5).

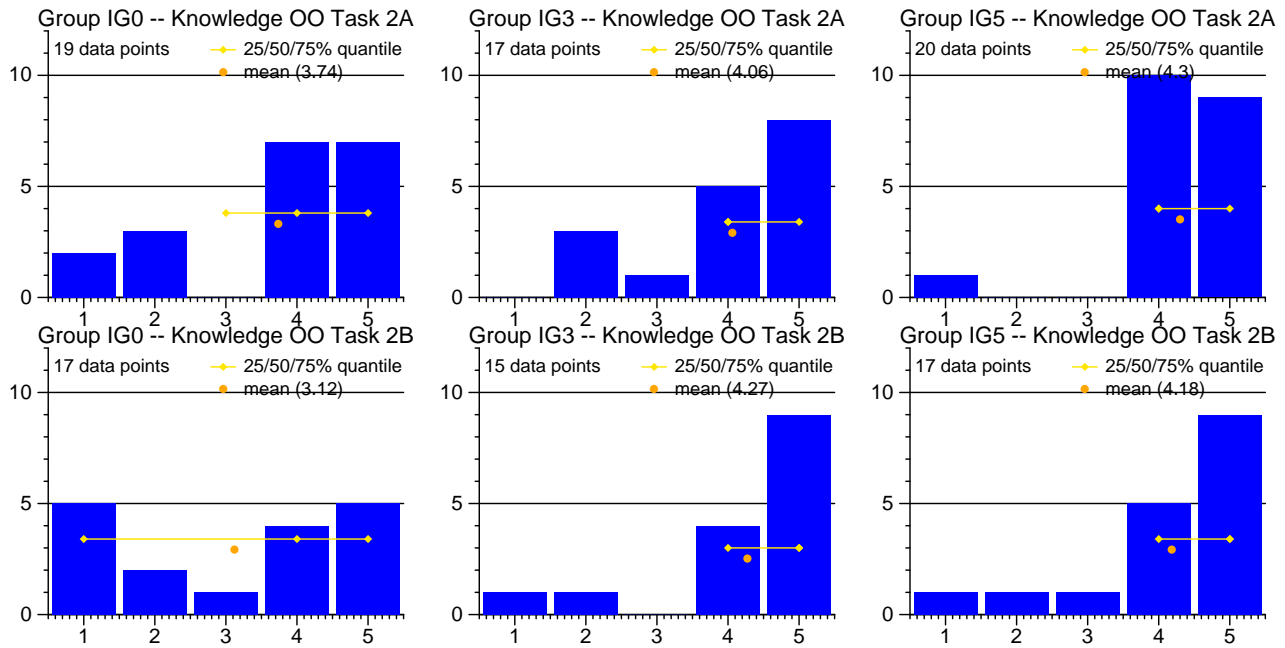


Figure 3.30: Ditto for the Informatik II subjects for Task 2A (first row) and Task 2B (second row).

Chapter 4

Analysis of the different results of the previous and our new experiment

The results found in this experiment and the results found by Daly et al. contradict. Daly et al. found that the program with inheritance depth of 3 is faster to maintain than the flat version and that the time for maintaining the program with inheritance depth of 5 is not significantly different from the equivalent flat program. In contrast, we found that for Task 2A (Informatik II) and Task 2 (JAKK) subjects with an inheritance program are slower than subjects maintaining the flat program. Task 2B could not be assessed because of the high mortality rate in the group with the deep inheritance hierarchy. For Task 1 the fastest group is the group with the deepest inheritance hierarchy (and hence the smallest number of places needing change).

What causes the differences?

Examining the programs of Daly et al. and our programs, we find several differences. Program length, number of classes, and number of methods differ, see Tables 1.1 and 2.1. The programs and maintenance tasks do not only differ in these respects, they also differ in complexity. Trying to find a measure for complexity we counted the number of methods that have to be understood to perform the task and the distance of the methods to their invocation, e. g. if a method M is invoked in the main program on an object of type C but this inherits the method from class A, that is the super-superclass of C, we count one external method invocation (call from main), two hierarchy changes (switch to super-superclass) and one method to trace (namely M). This is shown in Table 4.1. The process of gaining program understanding is sketched in Figures 2.35 and 2.36 on page 25 for Task 2 of our experiment and in Figure 4.1 for the original experiment. Each arrow across horizontal lines means that the method that is invoked is located in one of the superclasses or one of the subclasses.

	Boerse0	Boerse3	Boerse5	Univ0(3)	Univ3	Univ0(5)	Univ5
methods to trace	15	18	19	3	5	4	7
hierarchy changes	0	17	21	0	2	0	5
external method invocations	3	3	3	2	2	2	2

Table 4.1: Comparison of task complexity between our experiment and the ones of Daly et al. *Methods to trace* is the number of methods that must be investigated for gaining enough program understanding for solving the tasks. *hierarchy changes* is the number of inheritance steps that have to be followed to find the required methods (see Figure 4.1, and *external method invocations* are method invocations from outside the main hierarchy (e.g. from the main program).

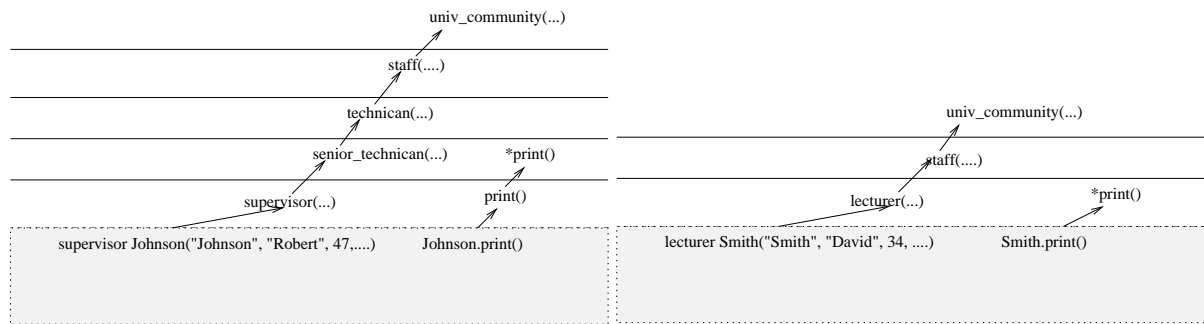


Figure 4.1: For University5 program with inheritance (left) and University3 program with inheritance (right) comprehension must be gained by searching through the inheritance hierarchy, starting in the main program. The arrows mean an invocation of a method. An arrow crossing a horizontal line means that the method invoked is located in a super- or subclass. The shaded region in the lower part are method calls from classes outside the main-hierarchy tree. Methods are marked with a star if they invoke other methods but (due to meaningful names) the method invocations do not need to be traced.

Analyzing the complexity metrics (number of classes in program, number of methods in program, LOC, number of methods to trace for the maintenance task, number of hierarchy changes to gain program understanding) we found two metrics having a high correlation with the required time. The average maintenance time correlates highly with the number of trace methods ($r = 0.98$) and the number of classes in the program ($r = 0.97$)¹ This is visualized in Figure 4.2.

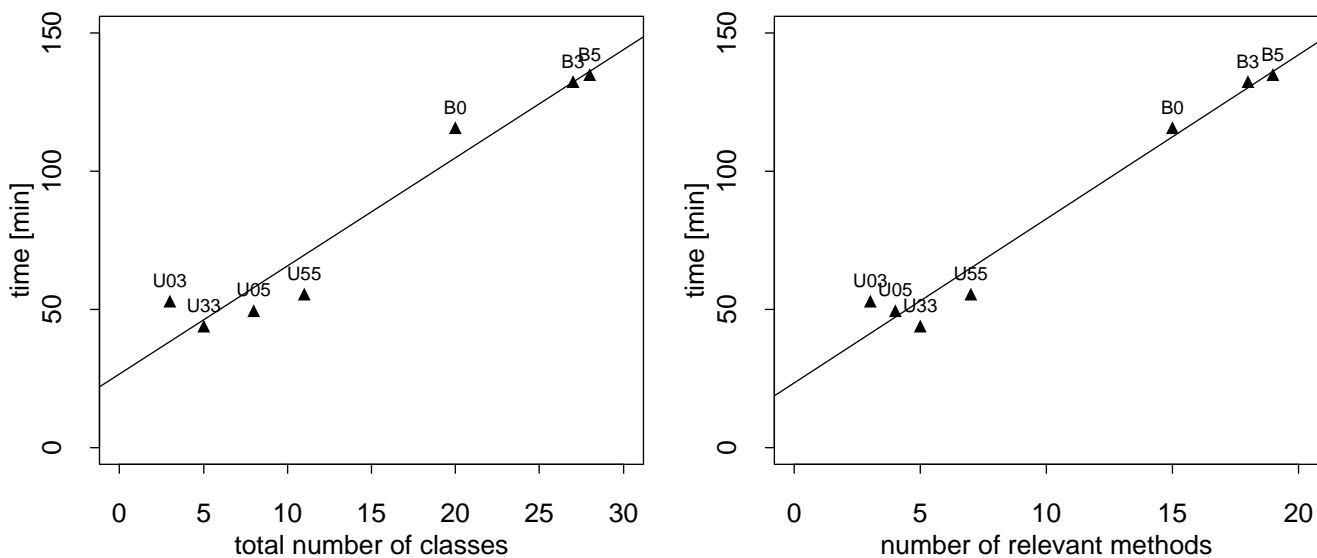


Figure 4.2: Correlation between the maintenance time and the number of relevant classes (left) and the number of relevant methods that have to be traced (right).

As we see, the university-3 program behaves against the trend of all the others: despite an increasing number of classes and trace methods the maintenance time decreases for the university-3 program. In the light of the

¹This correlation comprise only the University03-programs, the University05-programs and our JAKK-programs. The correlations change to 0.96 (time vs. classes) and 0.96 (time vs. trace methods) if we add the data points of the Literature-programs of Daly et al. We leave these data points out because the Literature programs are quite similar to the University03-program.

other data points and the strength of the relation, this is unexpected and unexplainable for us. So these results suggest that the main data point in Dalys conclusion may be some sort of outlier. It would be best to analyze their data subject-by-subject, but unfortunately we do not have access to it; according to John Daly, the raw data was lost.

Our conclusion: Not the inheritance depth itself is a good predictor of maintenance effort; crucial attributes in our experiment are related ones such as the number of methods to be understood.

In this experiment a lot of (probably) relevant factors could not be investigated: (1) We considered predominantly comprehension-bound tasks. What is the effect of more implementation-bound tasks? (2) Much of the code was relevant for the task. But in reality only a small fraction of a program will typically be investigated during any single maintenance task. What is the effect of this? (3) Polymorphism was not really essential for the program structure, inheritance was employed for reuse only. Are our results valid for designs using polymorphism as well?

So future work should address these questions, try to determine the relevant factors related to inheritance, and quantify their impact on maintenance time.

Chapter 5

Conclusion

In this experiment we tested the impact of inheritance hierarchy on maintainability. The experiment was inspired by a similar experiment conducted by Daly et al.

For testing our hypotheses, (H1) the more abstraction levels are used, the easier the program is to comprehend and the less time is required for maintenance tasks and (H2) the more abstraction levels are used, the better is the quality of the delivered solutions, we divided the groups into three balanced groups working on equivalent programs with zero, three, or five levels of inheritance.

In both groups, JAKK and Informatik II, we recognized similar effects. The “flat” program version was faster to comprehend than the other two program versions with inheritance depth 3 and 5. In Task 1 (Y2K-Problem), the subjects with the flat program version had more modifications to apply than the group with the 5-level program, but they were almost as fast (2.4 percent slower, not significant). The 3-level group had the same number of changes as the 0-level group but was much slower (23.9 percent, significant).

In Task 2 and Task 2A (adding new classes), the trend differed because here overall program comprehension was required to be able to do the program changes in contrast to Task 1 where it was sufficient to comprehend only a small part of the program. The time required by the subjects increased with the inheritance depth. Subjects with the flat version were significantly faster than subjects of the 5-level group. Although they were faster, their solutions were on average more correct than in the other groups.

In Task 2B, mortality was high and non-uniform over the groups so we can not properly evaluate this task.

Hypothesis 1 and hypothesis 2 have to be rejected. Contrary effects have been observed in this controlled experiment. The flat version of the program was easier to comprehend and to maintain.

Regarding the subjective experience of the subjects collected in the final questionnaire, it is obvious that the subject’s experience do not fit to the objective results. This indicates that experiments have to be conducted instead of relying on intuitive judgements. The software community should not rely on subjective experiences as was suggested by Al Davis in [5].

Our results contradict the results of the previous experiments by John Daly et al. It is most likely that the source of the differences is somewhere in the size and complexity of the programs and maintenance tasks; however, we can not point out exactly where.

We found that inheritance depth is not a good predictor for maintenance time. Instead we found two attributes that are good predictors – at least for the programs and “add a class” tasks in the given experiments: these

are the total number of classes in the program and the number of methods relevant for understanding. Further work is required for finding other relevant attributes that allow to predict maintenance time and for creating a quantitative model of maintenance that is applicable to a broader set of contexts.

Appendix A

Tasks and Solutions

A.1 Handling descriptions

This appendix contains the original materials given to the subjects. They were handed out in four parts:

1. a questionnaire collecting personal information and a small test of the subjects' knowledge of Java, inheritance, and polymorphism
2. the explanation of the first task together with the program listing and a some Unix help information
3. the explanation of the second task
4. a postmortem questionnaire

The subjects had to give each part back to the experimenters (except for the listing and the Unix info) when they received the next part. Each subject could promptly do this at any time.

A.2 Original questionnaire (translated into English)

Instruction and Questionnaire for the JAKK experiment

Barbara Unger, Lutz Prechelt, Michael Philippsen

Fakultät für Informatik, Universität Karlsruhe

June 10, 1997

Please read and follow these instructions **carefully**. Fill in your student ID and time correctly when required. Invoke the command `delivering` for delivering the source code only when you are finished with one of the tasks.

Take as much time as you need. We planned sufficient working time. For us, it is more important that you work well rather than fast.

Student ID:

Questionnaire Part 1: Personal information

Please fill in this part of the questionnaire before reading further and before the actual experiment starts. *All information will be considered confidential.*

Completeness and correctness of your information (please print!) are important for the accuracy of the scientific results of the experiment. Therefore, please answer all questions.

_____ lastname _____ firstname

_____ sex: m/f

_____ time

I am a _____major in my _____-th semester,
subject

with Vordiplom, without Vordiplom.

Before the Java course started I had the following programming experience (overall):

- only theoretical knowledge.
- wrote less than 300 lines of code myself.
- wrote less than 3,000 lines of code myself.
- wrote less than 30,000 lines of code myself.
- wrote more than 30,000 lines of code myself.

I have been programming for about _____ years now and used predominantly the following programming languages (ordered by decreasing experience):

_____ language 1, language 2, ...

Before the Java course started I had the following experience in object oriented programming:

- no knowledge
- only theoretical knowledge.
- wrote less than 300 lines of code myself.
- wrote less than 3,000 lines of code myself.
- wrote less than 30,000 lines of code myself.
- wrote more than 30,000 lines of code myself.

Before the Java course started I had the following experience in programming graphical user interfaces (GUIs):

- no knowledge
- only theoretical knowledge.
- wrote less than 300 lines of code myself.
- wrote less than 3,000 lines of code myself.
- wrote less than 30,000 lines of code myself.
- wrote more than 30,000 lines of code myself.

The longest program I have written *alone* had about _____ lines of code. It was written in _____ and consumed an effort of _____ person month.

LOC
person month

Answer the following question only if you have already worked in a team software project or are doing this currently. The largest program in whose construction I have *participated* had about _____ lines of code altogether and consumed an effort of _____ person months. My own contribution was about _____ lines of code or _____ person months, respectively.

LOC
person months
LOC
person months

My understanding of object-oriented concepts in Java is as follows:

(Enter a number between 1 and 5 for each concept. The number indicates how well you subjectively believe to understand the concept.

1: I understand and apply it very well,

2: I understand it well,

3: I understand it roughly,

4: I am beginning to understand it,

5: I do not understand it)

_____ Class

_____ Object (instance)

_____ Polymorphism

_____ Inheritance (*extends*)

_____ Overloading and overwriting methods

_____ abstract class

_____ abstract method

_____ Interface (*implements*)

_____ Difference value type (e. g. *int*) and object type

Now please enter your student id and time.

Student ID:

Time:

Questionnaire Part 2: A small Java test

Consider the following Java program:

```
class Superclass {
    int i = 7;
    void m() {
        System.out.println("Superclass i = "+i);
    }
}
class DerivedClass extends Superclass {
    static int j = 9;
    void m() {
        System.out.println("DerivedClass j = "+j);
    }
    static void mi() {
        System.out.println("DerivedClass i = "+i);
    }
}
class Test {
    public static void main(String args[]) {
        Superclass o1 = new Superclass();
        Superclass o2;
        DerivedClass u1 = new DerivedClass();
        DerivedClass u2;
        o2.m();
        o2 = o1;
        Superclass.m();
        DerivedClass.mi();
        o1.i++;
        o2.m();
        u2 = u1;
        u1.j++;
        u2.m();
        u1.j++;
        o2 = u1;
        o1.i = 0;
        o2.i = 1;
        o2.m();
        o2.mi();
    }
}
```

a.) Some of the instructions in the above program are illegal, that is, they result in compile time errors or run time errors. Mark all of these instructions in the listing above. Please do not use the computer for solving this task.

b.) Some of the instructions in the program produce output lines. Note in the listing behind these lines which output will be produced. Assume that the lines marked as wrong in a.) are removed from the program. Please do *not* use the computer for solving this task.

Tip: The simplest method is to write down step by step all changes of the objects and variables.

When you have finished this task **invoke the command** `deliver`, enter the actual time here, and request new materials. Time:

Now please enter the time. Time:

Student id: 000000

group: BoerseX

The exercise: Program “Boerse”

You received a listing of the program “Boerse” and a corresponding class diagram indicating the inheritance structure.

The source code is located on your computer:

Your Login: Test

Your Password: pwd

Directory: ~/Boerse

File: Boerse.java (and in case of need a copy is stored in Boerse.java.orig)

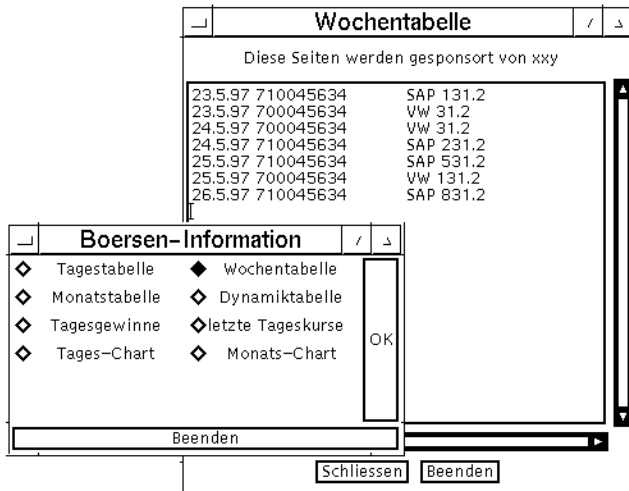
The purpose of the program is to show a selection of display types of various stock shares. For simplification the data are read from a file and the displays are simple, too.

There are two files that provide the data. One file called AVG.PRICES contains one entry per day and stock. Each entry consists of a date, the stock ID number (that identifies the stock share unambiguously), the appropriate (maybe ambiguous) name of the company and the average price of the share for that day. The second file called PRICES contains possibly several price values per day for the same stock, described by date, time, stock ID number, company name and share value (price).

The user can chose between various displays of the prices in tables or graphs with different selections and contents. For example, the DayTable displays all share values for all shares of one day, while a DayChart displays these data graphically for just one share.

A WeekTable displays all share values for all shares of the last seven days, a MonthTable accordingly for the last 30 days, etc. A MonthDynamicsTable shows for any two successive days of one month the change of the average share value in percent, etc.

As an example the following screen shot displays a view of the program while creating a week table:



Read and comprehend the program `Boerse.java` so that you can process the following two work tasks. The first task is described below, the second task will be handed out on a different sheet after completing the first one.

If you think that comprehending some parts of the program is not necessary, you do not need to investigate them.

Please do not write the solutions into the program listing. Implement your solution directly in the source code of `Boerse.java`, compile it and test it.

However, if you want to, you can write marks and notes into the listing. After the second work task deliver the listing together with the other handouts.

Work task 1 for program ”Boerse”

Starting with the year 2000, it will be insufficient to code the year number with two digits.

Enhance `Boerse.java` in that manner that the program can process the new files `PRICES_2000` and `AVG_PRICES_2000` correctly. These files contain four digit year numbers instead of two digit year numbers (the rest of the format and content is identical). Test your program with these files.

When you have finished this task or if you think, you can't do better to complete this task, **invoke the command `deliver`**, enter the actual time here, and request new materials. Time:

Now please enter the time. Time:

Student id: 000000

group: BoerseX

Work task 2 for program ”Boerse”

The tables implemented yet all cover fixed time intervals, for example one week back or one month back starting from today. This is too restrictive for the users; they want to choose the time interval themselves.

Enhance the program so that a new menu item “AnyIntervalTable” is offered. “AnyIntervalTable” should display a new table covering a user-specified time interval. The user is asked (via a dialog box) for a time interval (start date and end date, format dd.mm.yyyy-dd.mm.yyyy, for example 17.02.1997-26.03.1997).

Analogous to (for example) a WeekTable, all average prices (date, stock ID number, company name and share price) should be displayed for the specified time interval.

For solving the work task you can introduce new classes or you can change existing classes – as you prefer. For the dialog box use the same Frame as for the table created subsequently (only change its contents), just as it is done for the chart displays.

In much the same manner, introduce another new menu item ”AnyDynamicIntervalTable”, that displays a dynamics table (analogous to month dynamics table) of a time interval specified in a dialog box.

When you have finished this task or if you think, you can't do better to complete this task, **invoke the command deliver**, enter the actual time here, and request new materials. Time:

Now please enter your student id and time.

Student ID:

Time:

Questionnaire Part 3: Experience in the program

For proper evaluation of the experiment, we need your cooperation.

This part of the questionnaire is meant to complement the answers you gave above by subjective background information in order to allow for a better analysis of the experiment results.

Most questions have some additional space below for arbitrary comments; such information would be very useful for us.

I think that in the program “Boerse.java” inheritance is

- used much too much
 is used too much
 is used in the right degree
 is used too little
 is used much too little

comment:

Measured on programs of this size, I think that “Boerse.java” has a

- very clear structure
 clear structure
 medium clear structure
 not so clear structure
 unclear structure

comment:

The questions below have two checkboxes for each answer: for Task 1 (Year 2000) and for Task 2 (Time interval).

Please check exactly one box per column for each question.

Overall and in the given situation I found task 1 and 2

- | | | |
|---|---|---------------------|
| A | B | pretty simple |
| A | B | not quite so simple |
| A | B | pretty difficult |
| A | B | difficult |

comment:

During task 1 and 2 my ability to concentrate on the work was

- | | | |
|---|---|-------------|
| A | B | very high |
| A | B | high |
| A | B | OK |
| A | B | not so high |
| A | B | low |

comment:

I believe that my solutions for task 1 and 2 have

- | | | |
|---|---|--------------------------------------|
| A | B | no errors or omissions |
| A | B | at most one error or omission |
| A | B | probably several errors or omissions |
| A | B | (don't know) |

comment:

I think that for solving task 1 and 2 my previous knowledge of object oriented programming (as opposed to general programming knowledge) was helpful:

- | | | |
|---|---|----------------|
| A | B | no, not at all |
| A | B | only a little |
| A | B | can't decide |
| A | B | yes, somewhat |
| A | B | yes, very much |

comment:

This is how much effort I invested in various aspects while solving task 1 and task 2 (Please fill in each line a number between 1 and 5:

1: very much, 2: much, 3: medium, 4: somewhat, 5: only a little)

- | | | |
|---|---|------------------------------------------------|
| A | B | Implementing a well understandable program |
| A | B | Implementing a program that is well extensible |
| A | B | Implementing a well testable program |
| A | B | Implementing a defect-free program |
| A | B | Implementing an efficient program |
| A | B | Solving the task with little effort |

comment:

I think the purpose of this experiment was:

(Note: If you don't have an idea, just skip this question.)

Something else I would like to say (e. g. what I found particularly difficult, unclear in the experiment setup, interesting, etc.):

Thank you!

Many thanks for participating in our experiment.

A.3 Solutions

The solution described below is for the JAKK program version with inheritance depth 5. The solutions for the other program versions are analogous.

A.3.1 Solution for Task 1 – "Y2K"

line	original	new
49	static Datum heute = new Datum(27,5,97);	static Datum heute = new Datum(27,5,1997);
50	static String kurse = "KURSE";	static String kurse = "KURSE_2000";
51	static String kassakurse = "KASSAKURSE";	static String kassakurse = "KASSAKURSE_2000";
380	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
382	return (wkn.compareTo(s.substring(15,24))==0);	return (wkn.compareTo(s.substring(17,26))==0);
416	uhrzeit = new Uhrzeit(s.substring(9,15));	uhrzeit = new Uhrzeit(s.substring(11,17));
468	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
470	return (wkn.compareTo(s.substring(9,18))==0);	return (wkn.compareTo(s.substring(11,20))==0);
499	datum = new Datum(s.substring(0,9));	datum = new Datum(s.substring(0,10));
558	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
565	Datum datum = new Datum(s.substring(0,8));	Datum datum = new Datum(s.substring(0,10));
566	Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));	Uhrzeit uhrzeit = new Uhrzeit(s.substring(11,16));
568	String wkn = s.substring(14,24);	String wkn = s.substring(16,26);
569	String name = s.substring(25,s.length()-7);	String name = s.substring(27,s.length()-7);
665	jahr = new Integer(s.substring(6,8)).intValue();	jahr = new Integer(s.substring(6,10)).intValue();
677	jahr = new Integer(s.substring(6,8)).intValue();	jahr = new Integer(s.substring(6,10)).intValue();
796	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
802	Datum datum = new Datum(s.substring(0,8));	Datum datum = new Datum(s.substring(0,10));
803	String wkn = s.substring(9,18);	String wkn = s.substring(11,20);
804	String name = s.substring(19,s.length()-7);	String name = s.substring(21,s.length()-7);

A.3.2 Solution for Task 2 - "Time Interval Price Display and Gain/Loss Display"

```

class BelTabelle extends ZeitraumTabelle {
    Datum bisDatum;
    Label l = new Label();
    TextField t = new TextField("29.11.1996-29.11.1997",21);
    Button ok = new Button("ok");

    BelTabelle(String titel) {
        super(titel);
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,10));
        return (vonDatum.kleinergleich(d) && d.kleinergleich(bisDatum));
    }

    void stelleDar(){

```

//ueberdeckt stelleDar aus Tabelle.

```

}

boolean myaction(Event ev, Object target){
    if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
        String zeitraum = t.getText();
        vonDatum = new Datum(zeitraum.substring(0,10));
        bisDatum = new Datum(zeitraum.substring(11,21));
        p2.removeAll();
        leseDaten(dateiname);
        zeigeTabelleAn();
        return true;
    }
    return false;
}
}
}

```

```

class BelDynamikTabelle extends MonatsDynamikTabelle {
    Datum bisDatum;
    Label l = new Label();
    TextField t = new TextField("29.11.1996-29.11.1997",21);
    Button ok = new Button("ok");

    BelDynamikTabelle(String titel) {
        super(titel);
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,10));
        return (vonDatum.kleinergleich(d) && d.kleinergleich(bisDatum));
    }

    void stelleDar(){
        p2.add("North",l);
        p2.add("Center",t);
        p2.add("South",ok);
        pack();

        show();
    }
}

```

//ueberdeckt stelleDar aus Tabelle.

```

boolean myaction(Event ev, Object target){
    if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
        String zeitraum = t.getText();
        vonDatum = new Datum(zeitraum.substring(0,10));
        bisDatum = new Datum(zeitraum.substring(11,21));
        p2.removeAll();
        leseDaten(dateiname);
        zeigeTabelleAn();
        return true;
    }
    return false;
}
}
}

```

//////////Add in class Auswahl:

```
Checkbox cbbt = new Checkbox("Bel. Tabelle", cbg, false);
Checkbox cbbdt = new Checkbox("Bel. Dynamiktabelle", cbg, false);

p.add(cbbt);
p.add(cbbdt);

} else if(cbg.getSelectedCheckbox() == cbbt) {
    BelTabelle bt = new BelTabelle("Bel. Tabelle");
    bt.stelleDar();
    return true;
} else if(cbg.getSelectedCheckbox() == cbbdt) {
    BelDynamikTabelle bdt = new BelDynamikTabelle("Bel. Dynamiktabelle");
    bdt.stelleDar();
    return true;
```

Appendix B

Programs

B.1 Programs – JAKK

B.1.1 Boerse.java (no inheritance)

```
1  /*****/
2  /*Programmautor: Barbara Unger */
3  /*Datum: 1997-06-03 */
4  /*Gruppe: BoerseI */
5  /*****/
6
7  import java.awt.*;
8  import java.util.Vector;
9  import java.util.Enumeration;
10 import java.util.Date;
11 import java.io.*;
12
13 /* Programm zum Betrachten von Boersenkursen.
14    Abfolge der Klassen in dieser Datei:
15
16    interface Konstanten
17    class TagesChart extends Frame implements Konstanten
18    class TagesChartWert
19    class MonatsChart extends Frame implements Konstanten
20    class MonatsChartWert
21    class TagesTabelle extends Frame implements Konstanten
22    class Tagesdaten
23    class Zeitraumdaten
24    class Datum
25    class Uhrzeit
26    class Kurs
27    class Tagesgewinne extends Frame implements Konstanten
28    class LetzteTageskurse extends Frame implements Konstanten
29    class WochenTabelle extends Frame implements Konstanten
30    class MonatsTabelle extends Frame implements Konstanten
31    class MonatsDynamikTabelle extends Frame implements Konstanten
32    class TagesChartAnzeiger extends Canvas
33    class MonatsChartAnzeiger extends Canvas
34    class Auswahl extends Frame
```

```

35     class Boerse
36     */
37
38
39     interface Konstanten {
40     // static Date date = new Date();
41     // static Datum heute = new Datum(date.getDay(), date.getMonth(), date.getYear());
42     static Datum heute = new Datum(27, 5, 97);
43     static String kurse = "KURSE";
44     static String kassakurse = "KASSAKURSE";
45     }
46
47
48     /* Klasse TagesChart
49
50     Implementiert ein Fenster, in dem eine Boersen-Information angezeigt wird.
51     Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
52     des Programmes, berechnet fuer das Abrufen von Informationen keine
53     Kosten (Kosten werden von einem Sponsor getragen).
54     Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
55     Daten jedoch daraus erzeugt werden, wird durch die Methode
56     erzeugeDatenvector() entschieden. Diese entscheidet durch eine
57     selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
58     Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
59     Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
60     werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
61
62     Methoden:
63     TagesChart(String) Konstruktor, der den Titel setzt, einen Sponsor
64     einfaegt, die Kosten auf 0.00 setzt und die Datei
65     mit den Daten angibt.
66     leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
67     der Methode erzeugeDatenvector() in den Vector vec
68     erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
69     entscheidet mit Hilfe von selektiere(), ob der
70     Datensatz in den Datenvektor eingefuegt werden soll.
71     selektiere(String s) entscheidet, ob der String ausgewaehlt werden
72     soll, konkret: ob er von dem angegebenen Datum ist
73     und die entsprechende Wertpapierkennnummer hat
74     qsort(int, int), partition(int, int), exchange(int, int), sortiere()
75     sind Methoden, die einen Sortieralgorithmus zur
76     Verfuegung stellen.
77     schliesseFenster() kehrt zum aufrufenden Fenster zurueck
78     beendeProgramm() beendet das Programm komplett
79     action(Event, Object) verarbeitet Interaktionsereignisse
80     myaction() erweitert die action-Methode.
81     void stelleDar() stellt eine Dialogbox dar.
82     zeigeCanvasAn() erzeugt ein Objekt der Canvas-Klasse, in der der
83     Chart aus den selektierten Daten, die im Datenvektor
84     stehen, angezeigt wird.
85     */
86
87
88     class TagesChart extends Frame implements Konstanten {
89     Panel p1 = new Panel();
90     Panel p2 = new Panel();
91     Panel p3 = new Panel();
92     Button schliessen = new Button("Schliessen");
93     Button beenden = new Button("Beenden");
94     float kosten;
95     String dateiname;
96     Label sponsor;
97     Vector vec = new Vector();
98     int xChartSize = 200;

```



```

99     int yChartSize = 200;
100    String wkn;
101    Label l = new Label();
102    TextField t = new TextField("700045634",9);
103    Button ok = new Button("ok");
104    TagesChartWert tcw;
105
106    /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
107       bestimmt das Layout.
108       Panel p1 enthaelt die Sponsorinformation
109       Panel p2 enthaelt eine Auswahlmoeglichkeit des gewuenschten Wertpapiers
110       Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
111           des Programms.
112       Die Kosten werden auf 0.00 gesetzt.
113    */
114
115    TagesChart(String titel) {
116        super(titel);
117        this.setLayout(new BorderLayout());
118        p1.setLayout(new FlowLayout());
119        add("North",p1);
120        p2.setLayout(new BorderLayout());
121        add("Center",p2);
122        p3.setLayout(new FlowLayout());
123        add("South",p3);
124        p3.add(schliessen);
125        p3.add(beenden);
126        sponsor = new Label("Diese Seiten werden gesponsort von xxy");
127        p1.add(sponsor);
128        kosten = 0.0f;
129        dateiname = kurse;
130        l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Tageschart an");
131    }
132
133    /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
134       Methode erzeugeDatenvector eine Datenstruktur, in der die
135       relevanten Daten gespeichert werden.
136    */
137    void leseDaten(String dateiname) {
138        try{
139            BufferedReader br = new BufferedReader(new FileReader(dateiname));
140            String s;
141            try{
142                while((s = br.readLine())!=null) {
143                    if(!s.startsWith("#")) {
144                        erzeugeDatenvector(s);
145                    }
146                }
147            } catch (IOException e) {
148                System.err.println("Lesefehler bei Datei " + dateiname);
149            }
150        } catch (FileNotFoundException e) {
151            System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
152        }
153    }
154
155    /* bekommt eine Datensatz in Stringform und entscheidet mit Hilfe von selektiere(),
156       ob der Datensatz als TagesChartWert-Objekt in den Datenvektor vec
157       eingefuegt werden soll.
158    */
159    void erzeugeDatenvector(String s){
160        if(selektiere(s) == true) {
161            tcw = new TagesChartWert(s);
162            vec.addElement(tcw);

```

```

163     }
164   }
165
166   /* entscheidet, ob der String ausgewaehlt werden soll, konkret: ob er von
167     dem angegebenen Datum ist und die entsprechende Wertpapierkennnummer hat
168   */
169   boolean selektiere(String s) {
170     Datum d = new Datum();
171     d.setzeDatum(s.substring(0,8));
172     if((heute.tag == d.tag) && (heute.monat ==d.monat) && (heute.jahr == d.jahr)) {
173       return (wkn.compareTo(s.substring(15,24))==0);
174     }
175     return false;
176   }
177
178   /* Beschreibung des Quicksort siehe oben
179   */
180   void qsort(int left, int right) {
181     int middle;
182     if(left < right){
183       middle = partition(left, right);
184       qsort(left,middle-1);
185       qsort(middle+1, right);
186     }
187   }
188
189   /* Hilfsprozedur fuer den Quicksort
190   */
191   int partition(int left, int right) {
192     Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
193     int l = left;
194     int r = right;
195     int middle;
196     while(l < r) {
197       while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
198         l = l+1;
199       }
200       while((r ≥ left) && (!(Tagesdaten)vec.elementAt(r)).vergleiche(pivot)){
201         r = r-1;
202       }
203       if(l < r) {
204         exchange(l, r);
205       }
206     }
207     middle = r;
208     exchange (left, middle);
209     return middle;
210   }
211
212   /* Hilfsprozedur fuer den Quicksort
213   */
214   void exchange(int i, int j) {
215     Tagesdaten t = (Tagesdaten)vec.elementAt(i);
216     vec.setElementAt(vec.elementAt(j),i);
217     vec.setElementAt(t,j);
218   }
219
220   /* Ruft Quicksort auf
221   */
222   void sortiere() {
223     qsort(0, vec.size()-1);
224   }
225
226   /* Schliesst das Fenster und gibt seine Ressourcen frei.

```

```

227  */
228  void schliesseFenster() {
229      this.dispose();
230  }
231
232  /* Beendet das Programm komplett: alle Fenster werden geschlossen.
233  */
234  void beendeProgramm() {
235      System.exit(0);
236  }
237
238  /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
239  Knopfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
240  anderen Ereignisse wird "myaction" aufgerufen.
241  */
242  public boolean action(Event event, Object target) {
243      if(event.id == Event.ACTION_EVENT) {
244          if(event.target == schliessen) {
245              schliesseFenster();
246              return true;
247          }
248          else if(event.target == beenden) {
249              beendeProgramm();
250              return true;
251          } else {
252              return myaction(event, target);
253          }
254      } else {
255          return false;
256      }
257  }
258
259  /* myaction erweitert die action-Methode, so dass nun beim Druck des
260  Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
261  aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
262  und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
263  */
264  boolean myaction(Event ev, Object target){
265      if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
266          wkn = t.getText();
267          wkn.trim();
268          p2.removeAll();
269          leseDaten(dateiname);
270          p2.add("Center",zeigeCanvasAn());
271          pack();
272          show();
273          return true;
274      }
275      return false;
276  }
277
278  void stelleDar(){
279      p2.add("North",l);
280      p2.add("Center",t);
281      p2.add("South",ok);
282      pack();
283      show();
284  }
285
286  /* erzeugt ein Objekt der Canvas-Klasse, in der der Chart aus den
287  selektierten Daten, die im Datenvektor stehen, angezeigt wird.
288  */
289  Canvas zeigeCanvasAn() {
290      return new TagesChartAnzeiger(vec, xChartSize, yChartSize);

```

```

291 }
292 }
293
294 /*Klasse TagesChartWert
295 ist ein Objekttyp, der aus einer Uhrzeit und einem Kurswert besteht.
296 Das sind genau die Daten, die man zum Anzeigen eines Tagescharts braucht.
297 Die Klasse hat eine vergleiche-Methode, so dass man einen Vector mit
298 Objekten dieses Typs mit der sortier()-Methode sortieren kann.
299
300 Methoden:
301 TagesChartWert(String) bekommt einen String, in dem die Uhrzeit
302 und der Kurs enthalten sind.
303 vergleiche(TagesChartWert t) vergleicht die Uhrzeiten
304 a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
305 */
306
307
308 class TagesChartWert {
309     Uhrzeit uhrzeit;
310     Kurs kurs;
311
312 /* bekommt einen String, in dem die Uhrzeit und der Kurs enthalten sind.
313 */
314     TagesChartWert(String s) {
315         uhrzeit = new Uhrzeit(s.substring(9,15));
316         kurs = new Kurs(s.substring(s.length()-7, s.length()));
317     }
318
319 /* vergleicht die Uhrzeiten zweier TagesChartWert-Objekte
320 */
321     public boolean vergleiche(TagesChartWert t) {
322         return ((this.uhrzeit).kleinergleich(t.uhrzeit));
323     }
324 }
325
326 /* Klasse MonatsChart
327
328 Implementiert ein Fenster, in dem eine Boersen-Information angezeigt wird.
329 Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
330 des Programmes, berechnet fuer das Abrufen von Informationen keine
331 Kosten (Kosten werden von einem Sponsor getragen).
332 Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
333 Daten jedoch daraus erzeugt werden, wird durch die Methode
334 erzeugeDatenvector() entschieden. Diese entscheidet durch eine
335 selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
336 Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
337 Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
338 werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
339
340 Methoden:
341 MonatsChart(String) Konstruktor, der den Titel setzt, einen Sponsor
342 einfaegt, die Kosten auf 0.00 setzt und die Datei
343 mit den Daten angibt.
344 leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
345 der Methode erzeugeDatenvector() in den Vector vec
346 erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
347 entscheidet mit Hilfe von selektiere(), ob der
348 Datensatz in den Datenvektor eingefuegt werden soll.
349 selektiere(String s) entscheidet, ob der String ausgewaehlt werden
350 soll, konkret: ob er von dem angegebenen Datum ist
351 und die entsprechende Wertpapierkennnummer hat
352 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
353 sind Methoden, die einen Sortieralgorithmus zur
354 Verfuegung stellen.

```

```

355  schliesseFenster()  kehrt zum aufrufenden Fenster zurueck
356  beendeProgramm()   beendet das Programm komplett
357  action(Event, Object) verarbeitet Interaktionsereignisse
358  myaction()          erweitert die action-Methode.
359  void stelleDar()    stellt eine Dialogbox dar.
360  zeigeCanvasAn()     erzeugt ein Objekt der Canvas-Klasse, in der der
361                      Chart aus den selektierten Daten, die im Datenvektor
362                      stehen, angezeigt wird.
363  */
364
365
366  class MonatsChart extends Frame implements Konstanten {
367      Panel p1 = new Panel();
368      Panel p2 = new Panel();
369      Panel p3 = new Panel();
370      Button schliessen = new Button("Schliessen");
371      Button beenden = new Button("Beenden");
372      float kosten;
373      String dateiname;
374      Label sponsor;
375      Vector vec = new Vector();
376      int xChartSize = 200;
377      int yChartSize = 200;
378      String wkn;
379      Label l = new Label();
380      TextField t = new TextField("700045634",9);
381      Button ok = new Button("ok");
382      MonatsChartWert mcw;
383      Datum letzterMonat = new Datum();
384
385      /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
386         bestimmt das Layout.
387         Panel p1 enthaelt die Sponsorinformation
388         Panel p2 enthaelt eine Auswahlmoeglichkeit des gewuenschten Wertpapiers
389         Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
390         des Programms.
391         Die Kosten werden auf 0.00 gesetzt.
392     */
393
394     MonatsChart(String titel) {
395         super(titel);
396         this.setLayout(new BorderLayout());
397         p1.setLayout(new FlowLayout());
398         add("North",p1);
399         p2.setLayout(new BorderLayout());
400         add("Center",p2);
401         p3.setLayout(new FlowLayout());
402         add("South",p3);
403         p3.add(schliessen);
404         p3.add(beenden);
405         sponsor = new Label("Diese Seiten werden gesponsort von xxy");
406         p1.add(sponsor);
407         kosten = 0.0f;
408         l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Monatschart an");
409         dateiname = "kassakurse";
410         letzterMonat = heute.monatzurueck();
411     }
412
413     /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
414        Methode erzeugeDatenvektor eine Datenstruktur, in der die
415        relevanten Daten gespeichert werden.
416     */
417     void leseDaten(String dateiname) {
418         try{

```

```

419     BufferedReader br = new BufferedReader(new FileReader(dateiname));
420     String s;
421     try{
422         while((s = br.readLine())!=null) {
423             if(!s.startsWith("#")) {
424                 erzeugeDatenvector(s);
425             }
426         }
427     } catch (IOException e) {
428         System.err.println("Lesefehler bei Datei " + dateiname);
429     }
430 } catch (FileNotFoundException e) {
431     System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
432 }
433 }
434
435 /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
436 gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
437 */
438 void erzeugeDatenvector(String s){
439     if(selektiere(s) == true) {
440         mcw = new MonatsChartWert(s);
441         vec.addElement(mcw);
442     }
443 }
444
445 boolean selektiere(String s) {
446     Datum d = new Datum();
447     d.setzeDatum(s.substring(0,8));
448     if(!d.kleinergleich(letzterMonat)) {
449         return (wkn.compareTo(s.substring(9,18))==0);
450     }
451     return false;
452 }
453
454 /* Beschreibung des Quicksort siehe oben
455 */
456 void qsort(int left, int right) {
457     int middle;
458     if(left < right){
459         middle = partition(left, right);
460         qsort(left,middle-1);
461         qsort(middle+1, right);
462     }
463 }
464
465 /* Hilfsprozedur fuer den Quicksort
466 */
467 int partition(int left, int right) {
468     MonatsChartWert pivot = (MonatsChartWert)vec.elementAt(left);
469     int l = left;
470     int r = right;
471     int middle;
472     while(l < r) {
473         while((l ≤ right) && (((MonatsChartWert)vec.elementAt(l)).vergleiche(pivot))){
474             l = l+1;
475         }
476         while((r ≥ left) && (!(MonatsChartWert)vec.elementAt(r)).vergleiche(pivot)){
477             r = r-1;
478         }
479         if(l < r) {
480             exchange(l, r);
481         }
482     }

```

```

483     middle = r;
484     exchange (left, middle);
485     return middle;
486 }
487
488 /* Hilfsprozedur fuer den Quicksort
489 */
490 void exchange(int i, int j) {
491     MonatsChartWert m = (MonatsChartWert)vec.elementAt(i);
492     vec.setElementAt(vec.elementAt(j),i);
493     vec.setElementAt(m,j);
494 }
495
496 /* Ruft Quicksort auf
497 */
498 void sortiere() {
499     qsort(0, vec.size()-1);
500 }
501
502 /* Schliesst das Fenster und gibt seine Ressourcen frei.
503 */
504 void schliesseFenster() {
505     this.dispose();
506 }
507
508 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
509 */
510 void beendeProgramm() {
511     System.exit(0);
512 }
513
514 /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
515 Knopfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
516 anderen Ereignisse wird "myaction" aufgerufen.
517 */
518 public boolean action(Event event, Object target) {
519     if(event.id == Event.ACTION_EVENT) {
520         if(event.target == schliessen) {
521             schliesseFenster();
522             return true;
523         }
524         else if(event.target == beenden) {
525             beendeProgramm();
526             return true;
527         } else {
528             return myaction(event, target);
529         }
530     } else {
531         return false;
532     }
533 }
534
535 /* myaction erweitert die action-Methode, so dass nun beim Druck des
536 Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
537 aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
538 und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
539 */
540 boolean myaction(Event ev, Object target){
541     if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
542         wkn = t.getText();
543         wkn.trim();
544         p2.removeAll();
545         leseDaten(dateiname);
546         p2.add("Center",zeigeCanvasAn());

```

```

547     pack();
548     show();
549     return true;
550 }
551 return false;
552 }
553
554 void stelleDar(){
555     p2.add("North",l);
556     p2.add("Center",t);
557     p2.add("South",ok);
558     pack();
559     show();
560 }
561
562 Canvas zeigeCanvasAn() {
563     return new MonatsChartAnzeiger(vec, xChartSize, yChartSize);
564 }
565 }
566
567 /*Klasse MonatsChartWert
568 ist ein Objekttyp, der aus einem Datum und einem Kassakurswert besteht,
569 Daten, die man zum Anzeigen eines Monatschart braucht. Die Klasse hat
570 eine vergleiche-Methode, so dass man einen Vector mit diesem Typ mit
571 der sortier()-Methode sortieren kann.
572
573 Methoden:
574 MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
575 und der Kurs enthalten sind.
576 vergleiche(MonatsChartWert m) vergleicht das Datum
577 a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
578 */
579
580 class MonatsChartWert {
581     Kurs kassakurs;
582     Datum datum;
583
584     MonatsChartWert(String s){
585         kassakurs = new Kurs(s.substring(s.length()-6, s.length()));
586         datum = new Datum(s.substring(0,9));
587     }
588
589     public boolean vergleiche(MonatsChartWert m){
590         return ((this.datum).kleinergleich(m.datum));
591     }
592 }
593
594 /* Klasse TagesTabelle
595
596 Implementiert ein Fenster, in dem eine tabellarische
597 Darstellung aller Kurse eines Tages angezeigt wird.
598 Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
599 des Programmes, berechnet fuer das Abrufen von Informationen keine
600 Kosten (Kosten werden von einem Sponsor getragen).
601 Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
602 Daten jedoch daraus erzeugt werden, wird durch die Methode
603 erzeugeDatenvector() entschieden. Diese entscheidet durch eine
604 selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
605 Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
606 Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
607 werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
608
609 Methoden:
610 TagesTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor

```



```

611         einfuegt, die Kosten auf 0.00 setzt und die Datei
612         mit den Daten angibt.
613     leseDaten(String)   liest Daten aus einer Datei und schreibt sie mit Hilfe
614         der Methode erzeugeDatenvector() in den Vector vec
615     selektiere(String s) entscheidet, ob der String ausgewählt werden
616         soll, konkret: ob er von dem angegebenen Datum ist
617         und die entsprechende Wertpapierkennnummer hat
618     qsort(int, int), partition(int, int), exchange(int, int), sortiere()
619         sind Methoden, die einen Sortieralgorithmus zur
620         Verfügung stellen.
621     schliesseFenster() kehrt zum aufrufenden Fenster zurueck
622     beendeProgramm()   beendet das Programm komplett
623     action(Event, Object) verarbeitet Interaktionsereignisse
624     myaction(Event, Object) wird in "action" aufgerufen
625 */
626
627 class TagesTabelle extends Frame implements Konstanten {
628     Panel p1 = new Panel();
629     Panel p2 = new Panel();
630     Panel p3 = new Panel();
631     Button schliessen = new Button("Schliessen");
632     Button beenden = new Button("Beenden");
633     float kosten;
634     String dateiname;
635     Label sponsor;
636     Vector vec = new Vector();
637     TextArea textarea;
638
639     /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
640         bestimmt das Layout.
641         Panel p1 enthaelt die Sponsorinformation
642         Panel p2 dient zum Anzeigen der angeforderten Informationen,
643         Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
644         des Programms.
645         Die Kosten werden auf 0.00 gesetzt.
646     */
647
648     TagesTabelle(String titel) {
649         super(titel);
650         this.setLayout(new BorderLayout());
651         p1.setLayout(new FlowLayout());
652         add("North",p1);
653         p2.setLayout(new BorderLayout());
654         add("Center",p2);
655         p3.setLayout(new FlowLayout());
656         add("South",p3);
657         p3.add(schliessen);
658         p3.add(beenden);
659         sponsor = new Label("Diese Seiten werden gesponsort von xxy");
660         p1.add(sponsor);
661         kosten = 0.0f;
662         setzeDateiname();
663     }
664
665     void setzeDateiname(String s) {
666         dateiname = s;
667     }
668
669     void setzeDateiname() {
670         dateiname = kurse;
671     }
672
673     /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
674         Methode erzeugeDatenvector eine Datenstruktur, in der die

```

```

675     relevanten Daten gespeichert werden.
676     */
677     void leseDaten(String dateiname) {
678         try{
679             BufferedReader br = new BufferedReader(new FileReader(dateiname));
680             String s;
681             try{
682                 while((s = br.readLine())!=null) {
683                     if(!s.startsWith("#")) {
684                         erzeugeDatenvector(s);
685                     }
686                 }
687             } catch (IOException e) {
688                 System.err.println("Lesefehler bei Datei " + dateiname);
689             }
690         } catch (FileNotFoundException e) {
691             System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
692         }
693     }
694
695     /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
696     gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
697     */
698     void erzeugeDatenvector(String s) {
699         if(selektiere(s) == true) {
700             Datum datum = new Datum(s.substring(0,8));
701             Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
702             Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
703             String wkn = s.substring(14,24);
704             String name = s.substring(25, s.length()-7);
705             Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
706             vec.addElement(tagesdaten);
707         }
708     }
709
710     boolean selektiere(String s) {
711         Datum d = new Datum();
712         d.setzeDatum(s.substring(0,8));
713         return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
714     }
715
716     /* Beschreibung des Quicksort siehe oben
717     */
718     void qsort(int left, int right) {
719         int middle;
720         if(left < right){
721             middle = partition(left, right);
722             qsort(left,middle-1);
723             qsort(middle+1, right);
724         }
725     }
726
727     /* Hilfsprozedur fuer den Quicksort
728     */
729     int partition(int left, int right) {
730         Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
731         int l = left;
732         int r = right;
733         int middle;
734         while(l < r) {
735             while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
736                 l = l+1;
737             }
738             while((r ≥ left) && (!(Tagesdaten)vec.elementAt(r)).vergleiche(pivot)){

```

```

739     r = r-1;
740     }
741     if(l < r) {
742         exchange(l, r);
743     }
744     }
745     middle = r;
746     exchange (left, middle);
747     return middle;
748 }
749
750 /* Hilfsprozedur fuer den Quicksort
751 */
752 void exchange(int i, int j) {
753     Tagesdaten t = (Tagesdaten)vec.elementAt(i);
754     vec.setElementAt(vec.elementAt(j),i);
755     vec.setElementAt(t,j);
756 }
757
758 /* Ruft Quicksort auf
759 */
760 void sortiere() {
761     qsort(0, vec.size()-1);
762 }
763
764 /* Schliesst das Fenster und gibt seine Ressourcen frei.
765 */
766 void schliesseFenster() {
767     this.dispose();
768 }
769
770 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
771 */
772 void beendeProgramm() {
773     System.exit(0);
774 }
775
776 /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
777 Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
778 anderen Ereignisse wird "myaction" aufgerufen.
779 */
780 public boolean action(Event event, Object target) {
781     if(event.id == Event.ACTION_EVENT) {
782         if(event.target == schliessen) {
783             schliesseFenster();
784             return true;
785         }
786         else if(event.target == beenden) {
787             beendeProgramm();
788             return true;
789         } else {
790             return myaction(event, target);
791         }
792     } else {
793         return false;
794     }
795 }
796
797 /* Wird von "action" aufgerufen.
798 */
799 boolean myaction(Event ev, Object target){
800     return true;
801 }
802

```

```

803  /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
804  zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
805  noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
806  "stelleDar" aufgerufen.
807  */
808  void stelleDar(){
809      leseDaten(dateiname);
810      zeigeTabelleAn();
811  }
812
813  void zeigeTabelleAn(){
814      textarea = new TextArea(20,40);
815      p2.add("Center",textarea);
816      sortiere();
817      for(int i = 0; i < vec.size(); i++) {
818          textarea.append((((Tagesdaten)vec.elementAt(i)).uhrzeit.stunde + " : " +
819                          (((Tagesdaten)vec.elementAt(i)).uhrzeit.minute + " " +
820                          ((Tagesdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
821                          ((Tagesdaten)vec.elementAt(i)).name + " " +
822                          ((Tagesdaten)vec.elementAt(i)).kurs.kurs + "\n"));
823      }
824      if(vec.size() == 0) {
825          textarea.append("Fuer das heutige Datum, " + heute.tag + ". " +
826                          heute.monat + " . " + heute.jahr +
827                          ", existieren keine Daten\n");
828      }
829      this.pack();
830      this.show();
831  }
832 }
833
834
835 /* Klasse zur Repraesentation eines einzelnen Kursinformations-Datensatzes
836 a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
837 */
838 class Tagesdaten {
839     Datum datum;
840     Uhrzeit uhrzeit;
841     String wertpapierkennnummer;
842     String name;
843     Kurs kurs;
844
845     Tagesdaten(Datum d, Uhrzeit u, String wkn, String n, Kurs k) {
846         datum = d;
847         uhrzeit = u;
848         wertpapierkennnummer = wkn;
849         name = n;
850         kurs = k;
851     }
852
853     public boolean vergleiche(Tagesdaten t) {
854         return (this.uhrzeit).kleinergleich(t.uhrzeit);
855     }
856 }
857
858 /* Klasse zur Repraesentation eines einzelnen Kassakursinformations-Datensatzes
859 a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
860 */
861
862 class Zeitraumdaten {
863     Datum datum;
864     String wertpapierkennnummer;
865     String name;
866     Kurs kassakurs;

```

```

867
868 Zeitraumdaten(Datum d, String wkn, String n, Kurs k) {
869     datum = d;
870     wertpapierkennnummer = wkn;
871     name = n;
872     kassakurs = k;
873 }
874
875 public boolean vergleiche(Zeitraumdaten z) {
876     return (this.datum).kleinergleich(z.datum);
877 }
878 }
879
880
881 /* Klasse zur Repraesentation eines Datums, incl. Berechnung des
882 Datums 'vor einer Woche' und 'vor einem Monat'.
883 (Anmerkung zum Experiment:
884 Diese Routinen sind vereinfacht, um den Code zu verkuerzen.
885 Sie funktionieren nicht wirklich korrekt. Das soll uns hier aber mal
886 gerade nicht stoeren...)
887 */
888
889 class Datum {
890     int tag;
891     int monat;
892     int jahr;
893
894     Datum() {
895         tag = 0;
896         monat = 0;
897         jahr = 0;
898     }
899
900     Datum(String s){
901         tag = new Integer(s.substring(0,2)).intValue();
902         monat = new Integer(s.substring(3,5)).intValue();
903         jahr = new Integer(s.substring(6,8)).intValue();
904     }
905
906     Datum(int t, int m, int j){
907         tag = t;
908         monat = m;
909         jahr = j;
910     }
911
912     void setzeDatum(String s) {
913         tag = new Integer(s.substring(0,2)).intValue();
914         monat = new Integer(s.substring(3,5)).intValue();
915         jahr = new Integer(s.substring(6,8)).intValue();
916     }
917
918     void setzeDatum(int t, int m, int j) {
919         tag = t;
920         monat = m;
921         jahr = j;
922     }
923
924     boolean kleinergleich(Datum d) {
925         if(this.jahr < d.jahr) {
926             return true;
927         } else if(this.jahr == d.jahr){
928             if(this.monat < d.monat) {
929                 return true;
930             } else if(this.monat == d.monat){

```

```

931     if(this.tag ≤ d.tag) {
932         return true;
933     }
934 }
935 }
936 return false;
937 }
938
939 Datum monatzurueck() {
940     Datum d = new Datum();
941     d.tag = tag;
942     d.jahr = jahr;
943     if(monat == 1) {
944         d.monat = 12;
945         d.jahr = jahr - 1;
946     } else {
947         d.monat = monat - 1;
948     }
949     return d;
950 }
951
952 /* vereinfachte Berechnung: Alle Monate haben 30 Tage
953 */
954 Datum wochezurueck() {
955     Datum d = new Datum();
956     d.tag = tag - 7;
957     d.monat = monat;
958     d.jahr = jahr;
959     if(d.tag < 1) {
960         d.tag = d.tag + 30;
961         d.monat = monat - 1;
962         if(d.monat < 1) {
963             d.monat = d.monat + 12;
964             d.jahr = jahr - 1;
965         }
966     }
967     return d;
968 }
969
970 void print() {
971     System.out.println(" Datum = " + tag + " . " + monat + " . " + jahr);
972 }
973 }
974
975 /* Klasse zur Repraesentation einer Uhrzeit
976 */
977
978 class Uhrzeit {
979     int stunde;
980     int minute;
981
982     Uhrzeit(String s) {
983         stunde = (new Integer(s.substring(0,2))).intValue();
984         minute = (new Integer(s.substring(3,5))).intValue();
985     }
986
987     boolean kleinergleich(Uhrzeit u) {
988         if(this.stunde < u.stunde) {
989             return true;
990         } else if(this.stunde == u.stunde) {
991             if(this.minute ≤ u.minute) {
992                 return true;
993             }
994         }

```

```

995     return false;
996 }
997 }
998
999
1000 /* Klasse zur Repraesentation eines Kurses oder Kassakurses
1001 */
1002
1003 class Kurs {
1004     float kurs;
1005
1006     Kurs(String s){
1007         kurs = (new Float(s)).floatValue();
1008     }
1009 }
1010
1011
1012 /* Klasse Tagesgewinne
1013
1014     Implementiert ein Fenster, in dem eine tabellarische Darstellung der
1015     prozentualen Unterschiede zwischen dem ersten und letzten Kurs
1016     jeder Aktie am heutigen Datum angezeigt wird.
1017     Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1018     des Programmes, berechnet fuer das Abrufen von Informationen keine
1019     Kosten (Kosten werden von einem Sponsor getragen).
1020     Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
1021     Daten jedoch daraus erzeugt werden, wird durch die Methode
1022     erzeugeDatenvector() entschieden. Diese entscheidet durch eine
1023     selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
1024     Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
1025     Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
1026     werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
1027
1028     Methoden:
1029     Tagesgewinne(String) Konstruktor, der den Titel setzt, einen Sponsor
1030     einfaegt, die Kosten auf 0.00 setzt und die Datei
1031     mit den Daten angibt.
1032     leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
1033     der Methode erzeugeDatenvector() in den Vector vec
1034     selektiere(String s) entscheidet, ob der String ausgewaehlt werden
1035     soll, konkret: ob er von dem angegebenen Datum ist
1036     und die entsprechende Wertpapierkennnummer hat
1037     qsort(int, int), partition(int, int), exchange(int, int), sortiere()
1038     sind Methoden, die einen Sortieralgorithmus zur
1039     Verfuegung stellen.
1040     schliesseFenster() kehrt zum aufrufenden Fenster zurueck
1041     beendeProgramm() beendet das Programm komplett
1042     action(Event, Object) verarbeitet Interaktionsereignisse
1043     myaction(Event, Object) wird in "action" aufgerufen
1044 */
1045
1046 class Tagesgewinne extends Frame implements Konstanten {
1047     Panel p1 = new Panel();
1048     Panel p2 = new Panel();
1049     Panel p3 = new Panel();
1050     Button schliessen = new Button("Schliessen");
1051     Button beenden = new Button("Beenden");
1052     float kosten;
1053     String dateiname;
1054     Label sponsor;
1055     Vector vec = new Vector();
1056     TextArea textarea;
1057     Vector v = new Vector();
1058

```

```

1059  /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
1060  bestimmt das Layout.
1061  Panel p1 enthaelt die Sponsorinformation
1062  Panel p2 dient zum Anzeigen der angeforderten Informationen,
1063  Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1064  des Programms.
1065  Die Kosten werden auf 0.00 gesetzt.
1066  */
1067
1068  Tagesgewinne(String titel) {
1069      super(titel);
1070      this.setLayout(new BorderLayout());
1071      p1.setLayout(new FlowLayout());
1072      add("North",p1);
1073      p2.setLayout(new BorderLayout());
1074      add("Center",p2);
1075      p3.setLayout(new FlowLayout());
1076      add("South",p3);
1077      p3.add(schliessen);
1078      p3.add(beenden);
1079      sponsor = new Label("Diese Seiten werden gesponsort von xxy");
1080      p1.add(sponsor);
1081      kosten = 0.0f;
1082      setzeDateiname();
1083  }
1084
1085  void setzeDateiname(String s) {
1086      dateiname = s;
1087  }
1088
1089  void setzeDateiname() {
1090      dateiname = kurse;
1091  }
1092
1093  /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
1094  Methode erzeugeDatenvector eine Datenstruktur, in der die
1095  relevanten Daten gespeichert werden.
1096  */
1097  void leseDaten(String dateiname) {
1098      try{
1099          BufferedReader br = new BufferedReader(new FileReader(dateiname));
1100          String s;
1101          try{
1102              while((s = br.readLine())!=null) {
1103                  if(!s.startsWith("#")) {
1104                      erzeugeDatenvector(s);
1105                  }
1106              }
1107          } catch (IOException e) {
1108              System.err.println("Lesefehler bei Datei " + dateiname);
1109          }
1110      } catch (FileNotFoundException e) {
1111          System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
1112      }
1113  }
1114
1115  /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
1116  gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
1117  */
1118  void erzeugeDatenvector(String s) {
1119      if(selektiere(s) == true) {
1120          Datum datum = new Datum(s.substring(0,8));
1121          Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
1122          Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));

```



```

1123     String wkn = s.substring(14,24);
1124     String name = s.substring(25, s.length()-7);
1125     Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
1126     vec.addElement(tagesdaten);
1127 }
1128 }
1129
1130 boolean selektiere(String s) {
1131     Datum d = new Datum();
1132     d.setzeDatum(s.substring(0,8));
1133     return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
1134 }
1135
1136 /* Beschreibung des Quicksort siehe oben
1137 */
1138 void qsort(int left, int right) {
1139     int middle;
1140     if(left < right){
1141         middle = partition(left, right);
1142         qsort(left,middle-1);
1143         qsort(middle+1, right);
1144     }
1145 }
1146
1147 /* Hilfsprozedur fuer den Quicksort
1148 */
1149 int partition(int left, int right) {
1150     Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
1151     int l = left;
1152     int r = right;
1153     int middle;
1154     while(l < r) {
1155         while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
1156             l = l+1;
1157         }
1158         while((r ≥ left) && (!(Tagesdaten)vec.elementAt(r)).vergleiche(pivot)){
1159             r = r-1;
1160         }
1161         if(l < r) {
1162             exchange(l, r);
1163         }
1164     }
1165     middle = r;
1166     exchange (left, middle);
1167     return middle;
1168 }
1169
1170 /* Hilfsprozedur fuer den Quicksort
1171 */
1172 void exchange(int i, int j) {
1173     Tagesdaten t = (Tagesdaten)vec.elementAt(i);
1174     vec.setElementAt(vec.elementAt(j),i);
1175     vec.setElementAt(t,j);
1176 }
1177
1178 /* Ruft Quicksort auf
1179 */
1180 void sortiere() {
1181     qsort(0, vec.size()-1);
1182 }
1183
1184 /* Schliesst das Fenster und gibt seine Ressourcen frei.
1185 */
1186 void schliesseFenster() {

```

```

1187     this.dispose();
1188 }
1189
1190 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
1191 */
1192 void beendeProgramm() {
1193     System.exit(0);
1194 }
1195
1196 /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
1197   Knopfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
1198   anderen Ereignisse wird "myaction" aufgerufen.
1199 */
1200 public boolean action(Event event, Object target) {
1201     if(event.id == Event.ACTION_EVENT) {
1202         if(event.target == schliessen) {
1203             schliesseFenster();
1204             return true;
1205         }
1206         else if(event.target == beenden) {
1207             beendeProgramm();
1208             return true;
1209         } else {
1210             return myaction(event, target);
1211         }
1212     } else {
1213         return false;
1214     }
1215 }
1216
1217 /* Wird von "action" aufgerufen.
1218 */
1219 boolean myaction(Event ev, Object target){
1220     return true;
1221 }
1222
1223 /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
1224   zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
1225   noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
1226   "stelleDar" aufgerufen.
1227 */
1228 void stelleDar(){
1229     leseDaten(dateiname);
1230     zeigeTabelleAn();
1231 }
1232
1233 boolean schonAbgearbeitet(String s) {
1234     int i = 0;
1235     while(i < v.size()) {
1236         if((((String)v.elementAt(i)).compareTo(s))==0) {
1237             i = v.size();
1238             return true;
1239         } else {
1240             i++;
1241         }
1242     }
1243     v.addElement(s);
1244     return false;
1245 }
1246
1247 float tagesgewinn(String wertpapierkennnummer, int start) {
1248     float min = 0.0f;
1249     float max = 0.0f;
1250     int i = start;

```

```

1251   while((wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) != 0) && (i <
vec.size())) {
1252       i = i+1;
1253   }
1254   min = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
1255   while(i < vec.size()) {
1256       if(wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
1257           max = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
1258       }
1259       i = i+1;
1260   }
1261   //   System.out.println("Min = " + min + "      Max = " + max);
1262   if(min == 0.0f) {
1263       return 0;
1264   } else {
1265       return ((max - min)/min);
1266   }
1267 }
1268
1269 void zeigeTabelleAn() {
1270     textarea = new TextArea(20,40);
1271     p2.add("Center",textarea);
1272     boolean keineDaten = true;
1273     sortiere();
1274     for(int i = 0; i < vec.size(); i++) {
1275         if(! schonAbgearbeitet(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
1276             textarea.append(((Tagesdaten)(vec.elementAt(i))).name + " " +
1277                 ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + " "
1278                 + tagesgewinn(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
1279                 "\n");
1280             keineDaten = false;
1281         }
1282     }
1283     if(keineDaten == true) {
1284         textarea.append("Fuer das heutige Datum, " + heute.tag + " ." +
1285             heute.monat + " ." + heute.jahr +
1286             ", existieren keine Daten\n");
1287     }
1288     this.pack();
1289     this.show();
1290 }
1291 }
1292
1293
1294 /* Klasse LetzteTageskurse
1295
1296   Implementiert ein Fenster, in dem eine tabellarische Darstellung der
1297   letzten vorhandenen heutigen Kurse fuer alle Aktientitel angezeigt wird.
1298   Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1299   des Programmes, berechnet fuer das Abrufen von Informationen keine
1300   Kosten (Kosten werden von einem Sponsor getragen).
1301   Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
1302   Daten jedoch daraus erzeugt werden, wird durch die Methode
1303   erzeugeDatenvector() entschieden. Diese entscheidet durch eine
1304   selektiere()-Methode, welche Datensatze ueberhaupt ausgewaehlt werden sollen.
1305   Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
1306   Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
1307   werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
1308
1309   Methoden:
1310   LetzteTageskurse(String) Konstruktor, der den Titel setzt, einen Sponsor
1311                       einfuegt, die Kosten auf 0.00 setzt und die Datei
1312                       mit den Daten angibt.
1313   leseDaten(String)   liest Daten aus einer Datei und schreibt sie mit Hilfe

```

```

1314         der Methode erzeugeDatenvector() in den Vector vec
1315 selektiere(String s) entscheidet, ob der String ausgewaehlt werden
1316 soll, konkret: ob er von dem angegebenen Datum ist
1317 und die entsprechende Wertpapierkennnummer hat
1318 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
1319 sind Methoden, die einen Sortieralgorithmus zur
1320 Verfuegung stellen.
1321 schliesseFenster() kehrt zum aufrufenden Fenster zurueck
1322 beendeProgramm() beendet das Programm komplett
1323 action(Event, Object) verarbeitet Interaktionsereignisse
1324 myaction(Event, Object) wird in "action" aufgerufen
1325 */
1326
1327
1328 class LetzteTageskurse extends Frame implements Konstanten {
1329     Panel p1 = new Panel();
1330     Panel p2 = new Panel();
1331     Panel p3 = new Panel();
1332     Button schliessen = new Button("Schliessen");
1333     Button beenden = new Button("Beenden");
1334     float kosten;
1335     String dateiname;
1336     Label sponsor;
1337     Vector vec = new Vector();
1338     TextArea textarea;
1339     Vector v = new Vector();
1340
1341     /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
1342 bestimmt das Layout.
1343 Panel p1 enthaelt die Sponsorinformation
1344 Panel p2 dient zum Anzeigen der angeforderten Informationen,
1345 Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1346 des Programms.
1347 Die Kosten werden auf 0.00 gesetzt.
1348 */
1349
1350     LetzteTageskurse(String titel) {
1351         super(titel);
1352         this.setLayout(new BorderLayout());
1353         p1.setLayout(new FlowLayout());
1354         add("North",p1);
1355         p2.setLayout(new BorderLayout());
1356         add("Center",p2);
1357         p3.setLayout(new FlowLayout());
1358         add("South",p3);
1359         p3.add(schliessen);
1360         p3.add(beenden);
1361         sponsor = new Label("Diese Seiten werden gesponsort von xxy");
1362         p1.add(sponsor);
1363         kosten = 0.0f;
1364         setzeDateiname();
1365     }
1366
1367     void setzeDateiname(String s) {
1368         dateiname = s;
1369     }
1370
1371     void setzeDateiname() {
1372         dateiname = kurse;
1373     }
1374
1375     /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
1376 Methode erzeugeDatenvector eine Datenstruktur, in der die
1377 relevanten Daten gespeichert werden.

```

```

1378  */
1379  void leseDaten(String dateiname) {
1380      try{
1381          BufferedReader br = new BufferedReader(new FileReader(dateiname));
1382          String s;
1383          try{
1384              while((s = br.readLine())!=null) {
1385                  if(!s.startsWith("#")) {
1386                      erzeugeDatenvector(s);
1387                  }
1388              }
1389          } catch (IOException e) {
1390              System.err.println("Lesefehler bei Datei " + dateiname);
1391          }
1392          } catch (FileNotFoundException e) {
1393              System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
1394          }
1395      }
1396
1397      /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
1398         gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
1399         */
1400      void erzeugeDatenvector(String s) {
1401          if(selektiere(s) == true) {
1402              Datum datum = new Datum(s.substring(0,8));
1403              Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
1404              Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
1405              String wkn = s.substring(14,24);
1406              String name = s.substring(25, s.length()-7);
1407              Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
1408              vec.addElement(tagesdaten);
1409          }
1410      }
1411
1412      boolean selektiere(String s) {
1413          Datum d = new Datum();
1414          d.setzeDatum(s.substring(0,8));
1415          return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
1416      }
1417
1418      /* Beschreibung des Quicksort siehe oben
1419         */
1420      void qsort(int left, int right) {
1421          int middle;
1422          if(left < right){
1423              middle = partition(left, right);
1424              qsort(left,middle-1);
1425              qsort(middle+1, right);
1426          }
1427      }
1428
1429      /* Hilfsprozedur fuer den Quicksort
1430         */
1431      int partition(int left, int right) {
1432          Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
1433          int l = left;
1434          int r = right;
1435          int middle;
1436          while(l < r) {
1437              while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))) {
1438                  l = l+1;
1439              }
1440              while((r ≥ left) && (!(Tagesdaten)vec.elementAt(r)).vergleiche(pivot)){
1441                  r = r-1;

```

```

1442     }
1443     if(l < r) {
1444         exchange(l, r);
1445     }
1446 }
1447 middle = r;
1448 exchange (left, middle);
1449 return middle;
1450 }
1451
1452 /* Hilfsprozedur fuer den Quicksort
1453 */
1454 void exchange(int i, int j) {
1455     Tagesdaten t = (Tagesdaten)vec.elementAt(i);
1456     vec.setElementAt(vec.elementAt(j),i);
1457     vec.setElementAt(t,j);
1458 }
1459
1460 /* Ruft Quicksort auf
1461 */
1462 void sortiere() {
1463     qsort(0, vec.size()-1);
1464 }
1465
1466 /* Schliesst das Fenster und gibt seine Ressourcen frei.
1467 */
1468 void schliesseFenster() {
1469     this.dispose();
1470 }
1471
1472 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
1473 */
1474 void beendeProgramm() {
1475     System.exit(0);
1476 }
1477
1478 /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
1479 Knöpfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
1480 anderen Ereignisse wird "myaction" aufgerufen.
1481 */
1482 public boolean action(Event event, Object target) {
1483     if(event.id == Event.ACTION_EVENT) {
1484         if(event.target == schliessen) {
1485             schliesseFenster();
1486             return true;
1487         }
1488         else if(event.target == beenden) {
1489             beendeProgramm();
1490             return true;
1491         } else {
1492             return myaction(event, target);
1493         }
1494     } else {
1495         return false;
1496     }
1497 }
1498
1499 /* Wird von "action" aufgerufen.
1500 */
1501 boolean myaction(Event ev, Object target){
1502     return true;
1503 }
1504
1505 /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da

```

```

1506     zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
1507     noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
1508     "stelleDar" aufgerufen.
1509     */
1510 void stelleDar(){
1511     leseDaten(dateiname);
1512     zeigeTabelleAn();
1513 }
1514
1515 boolean schonAbgearbeitet(String s) {
1516     int i = 0;
1517     while(i < v.size()) {
1518         if(((String)v.elementAt(i)).compareTo(s)==0) {
1519             i = v.size();
1520             return true;
1521         } else {
1522             i++;
1523         }
1524     }
1525     v.addElement(s);
1526     return false;
1527 }
1528
1529 float letztetageskurse(String wertpapierkennnummer, int start) {
1530     float letzter = 0.0f;
1531     int i = start;
1532     while(i < vec.size()) {
1533         if(wertpapierkennnummer.compareTo(
1534             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
1535             letzter = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
1536         }
1537         i = i+1;
1538     }
1539     return letzter;
1540 }
1541
1542 void zeigeTabelleAn() {
1543     textarea = new TextArea(20,40);
1544     p2.add("Center",textarea);
1545     sortiere();
1546     for(int i = 0; i < vec.size(); i++) {
1547         if(!schonAbgearbeitet(
1548             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
1549             textarea.append(((Tagesdaten)(vec.elementAt(i))).name + " " +
1550                 ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + " " +
1551                 letztetageskurse(
1552                     ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
1553                 "\n");
1554         }
1555     }
1556     if(vec.size() == 0) {
1557         textarea.append("Fuer das heutige Datum, " +
1558             heute.tag + "." + heute.monat + "." +
1559             heute.jahr + ", existieren keine Daten\n");
1560     }
1561     this.pack();
1562     this.show();
1563 }
1564 }
1565
1566
1567 /* Klasse WochenTabelle
1568
1569     Implementiert ein Fenster, in dem eine tabellarische Darstellung der

```

```

1570 Kassakurse aller Aktientitel fuer die letzten 7 Tage angezeigt wird.
1571 Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1572 des Programmes, berechnet fuer das Abrufen von Informationen keine
1573 Kosten (Kosten werden von einem Sponsor getragen).
1574 Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
1575 Daten jedoch daraus erzeugt werden, wird durch die Methode
1576 erzeugeDatenvector() entschieden. Diese entscheidet durch eine
1577 selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
1578 Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
1579 Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
1580 werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
1581
1582 Methoden:
1583 WochenTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor
1584 einfaegt, die Kosten auf 0.00 setzt und die Datei
1585 mit den Daten angibt.
1586 leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
1587 der Methode erzeugeDatenvector() in den Vector vec
1588 selektiere(String s) entscheidet, ob der String ausgewaehlt werden
1589 soll, konkret: ob er von dem angegebenen Datum ist
1590 und die entsprechende Wertpapierkennnummer hat
1591 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
1592 sind Methoden, die einen Sortieralgorithmus zur
1593 Verfuegung stellen.
1594 schliesseFenster() kehrt zum aufrufenden Fenster zurueck
1595 beendeProgramm() beendet das Programm komplett
1596 action(Event, Object) verarbeitet Interaktionsereignisse
1597 myaction(Event, Object) wird in "action" aufgerufen
1598 */
1599
1600 class WochenTabelle extends Frame implements Konstanten {
1601     Panel p1 = new Panel();
1602     Panel p2 = new Panel();
1603     Panel p3 = new Panel();
1604     Button schliessen = new Button("Schliessen");
1605     Button beenden = new Button("Beenden");
1606     float kosten;
1607     String dateiname;
1608     Label sponsor;
1609     Vector vec = new Vector();
1610     Datum vonDatum;
1611     TextArea textarea;
1612
1613     /* Konstruktor. Setzt den Titel des Fensters, faegt drei Panels ein und
1614 bestimmt das Layout.
1615 Panel p1 enthaelt die Sponsorinformation
1616 Panel p2 dient zum Anzeigen der angeforderten Informationen,
1617 Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1618 des Programms.
1619 Die Kosten werden auf 0.00 gesetzt.
1620 */
1621
1622     WochenTabelle(String titel) {
1623         super(titel);
1624         vonDatum = heute.wochezurueck();
1625         this.setLayout(new BorderLayout());
1626         p1.setLayout(new FlowLayout());
1627         add("North",p1);
1628         p2.setLayout(new BorderLayout());
1629         add("Center",p2);
1630         p3.setLayout(new FlowLayout());
1631         add("South",p3);
1632         p3.add(schliessen);
1633         p3.add(beenden);

```



```

1634     sponsor = new Label("Diese Seiten werden gesponsort von xxy");
1635     pl.add(sponsor);
1636     kosten = 0.0f;
1637     setzeDateiname();
1638 }
1639
1640 void setzeDateiname(String s) {
1641     dateiname = s;
1642 }
1643
1644 void setzeDateiname() {
1645     dateiname = kassakurse;
1646 }
1647
1648 /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
1649    Methode erzeugeDatenvector eine Datenstruktur, in der die
1650    relevanten Daten gespeichert werden.
1651 */
1652 void leseDaten(String dateiname) {
1653     try{
1654         BufferedReader br = new BufferedReader(new FileReader(dateiname));
1655         String s;
1656         try{
1657             while((s = br.readLine())!=null) {
1658                 if(!s.startsWith("#")) {
1659                     erzeugeDatenvector(s);
1660                 }
1661             }
1662         } catch (IOException e) {
1663             System.err.println("Lesefehler bei Datei " + dateiname);
1664         }
1665     } catch (FileNotFoundException e) {
1666         System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden"
1667 );
1668     }
1669 }
1670
1671 /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
1672    gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
1673 */
1674 void erzeugeDatenvector(String s){
1675     if(selektiere(s) == true) {
1676         Datum datum = new Datum(s.substring(0,8));
1677         String wkn = s.substring(9,18);
1678         String name = s.substring(19, s.length()-7);
1679         Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
1680         Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
1681         vec.addElement(zrd);
1682     }
1683 }
1684
1685 boolean selektiere(String s) {
1686     Datum d = new Datum();
1687     d.setzeDatum(s.substring(0,8));
1688     return (!d.kleinergleich(vonDatum));
1689 }
1690
1691 /* Beschreibung des Quicksort siehe oben
1692 */
1693 void qsort(int left, int right) {
1694     int middle;
1695     if(left < right){
1696         middle = partition(left, right);
1697         qsort(left,middle-1);

```

```

1698     qsort(middle+1, right);
1699   }
1700 }
1701
1702 /* Hilfsprozedur fuer den Quicksort
1703 */
1704 int partition(int left, int right) {
1705     Zeitraumdaten pivot = (Zeitraumdaten)vec.elementAt(left);
1706     int l = left;
1707     int r = right;
1708     int middle;
1709     while(l < r) {
1710         while(l ≤ right) && (((Zeitraumdaten)vec.elementAt(l)).vergleiche(pivot)){
1711             l = l+1;
1712         }
1713         while((r ≥ left) && (!(Zeitraumdaten)vec.elementAt(r)).vergleiche(pivot)){
1714             r = r-1;
1715         }
1716         if(l < r) {
1717             exchange(l, r);
1718         }
1719     }
1720     middle = r;
1721     exchange (left, middle);
1722     return middle;
1723 }
1724
1725 /* Hilfsprozedur fuer den Quicksort
1726 */
1727 void exchange(int i, int j) {
1728     Zeitraumdaten z = (Zeitraumdaten)vec.elementAt(i);
1729     vec.setElementAt(vec.elementAt(j),i);
1730     vec.setElementAt(z,j);
1731 }
1732
1733 /* Ruft Quicksort auf
1734 */
1735 void sortiere() {
1736     qsort(0, vec.size()-1);
1737 }
1738
1739 /* Schliesst das Fenster und gibt seine Ressourcen frei.
1740 */
1741 void schliesseFenster() {
1742     this.dispose();
1743 }
1744
1745 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
1746 */
1747 void beendeProgramm() {
1748     System.exit(0);
1749 }
1750
1751 /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
1752 Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
1753 anderen Ereignisse wird "myaction" aufgerufen.
1754 */
1755 public boolean action(Event event, Object target) {
1756     if(event.id == Event.ACTION_EVENT) {
1757         if(event.target == schliessen) {
1758             schliesseFenster();
1759             return true;
1760         }
1761         else if(event.target == beenden) {

```

```

1762     beendeProgramm();
1763     return true;
1764 } else {
1765     return myaction(event, target);
1766 }
1767 } else {
1768     return false;
1769 }
1770 }
1771
1772 /* Wird von "action" aufgerufen.
1773 */
1774 boolean myaction(Event ev, Object target) {
1775     return true;
1776 }
1777
1778 /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
1779 zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
1780 noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
1781 "stelleDar" aufgerufen.
1782 */
1783 void stelleDar(){
1784     leseDaten(dateiname);
1785     zeigeTabelleAn();
1786 }
1787
1788 void zeigeTabelleAn(){
1789     textarea = new TextArea(20,40);
1790     p2.add("Center",textarea);
1791     sortiere();
1792     for(int i = 0; i < vec.size(); i++) {
1793         textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
1794                         (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
1795                         (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
1796                         ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
1797                         ((Zeitraumdaten)vec.elementAt(i)).name + " " +
1798                         ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
1799     }
1800     if(vec.size() == 0) {
1801         textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
1802     }
1803     this.pack();
1804     this.show();
1805 }
1806 }
1807
1808
1809 /* Klasse MonatsTabelle
1810
1811 Implementiert ein Fenster, in dem eine tabellarische Darstellung der
1812 Kassakurse aller Aktientitel fuer die letzten 30 Tage (genauer gesagt:
1813 seit dem gleichen Tag im Vormonat) angezeigt wird.
1814 Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1815 des Programmes, berechnet fuer das Abrufen von Informationen keine
1816 Kosten (Kosten werden von einem Sponsor getragen).
1817 Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
1818 Daten jedoch daraus erzeugt werden, wird durch die Methode
1819 erzeugeDatenvector() entschieden. Diese entscheidet durch eine
1820 selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
1821 Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
1822 Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
1823 werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
1824
1825 Methoden:

```

```

1826 MonatsTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor
1827      einfuegt, die Kosten auf 0.00 setzt und die Datei
1828      mit den Daten angibt.
1829 leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
1830      der Methode erzeugeDatenvector() in den Vector vec
1831 selektiere(String s) entscheidet, ob der String ausgewaehlt werden
1832      soll, konkret: ob er von dem angegebenen Datum ist
1833      und die entsprechende Wertpapierkennnummer hat
1834 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
1835      sind Methoden, die einen Sortieralgorithmus zur
1836      Verfuegung stellen.
1837 schliesseFenster() kehrt zum aufrufenden Fenster zurueck
1838 beendeProgramm() beendet das Programm komplett
1839 action(Event, Object) verarbeitet Interaktionsereignisse
1840 myaction(Event, Object) wird in "action" aufgerufen
1841 */
1842
1843 class MonatsTabelle extends Frame implements Konstanten {
1844     Panel p1 = new Panel();
1845     Panel p2 = new Panel();
1846     Panel p3 = new Panel();
1847     Button schliessen = new Button("Schliessen");
1848     Button beenden = new Button("Beenden");
1849     float kosten;
1850     String dateiname;
1851     Label sponsor;
1852     Vector vec = new Vector();
1853     Datum vonDatum;
1854     TextArea textarea;
1855
1856     /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
1857      bestimmt das Layout.
1858      Panel p1 enthaelt die Sponsorinformation
1859      Panel p2 dient zum Anzeigen der angeforderten Informationen,
1860      Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
1861      des Programms.
1862      Die Kosten werden auf 0.00 gesetzt.
1863     */
1864
1865     MonatsTabelle(String titel) {
1866         super(titel);
1867         vonDatum = heute.monatzurueck();
1868         this.setLayout(new BorderLayout());
1869         p1.setLayout(new FlowLayout());
1870         add("North",p1);
1871         p2.setLayout(new BorderLayout());
1872         add("Center",p2);
1873         p3.setLayout(new FlowLayout());
1874         add("South",p3);
1875         p3.add(schliessen);
1876         p3.add(beenden);
1877         sponsor = new Label("Diese Seiten werden gesponsort von xxy");
1878         p1.add(sponsor);
1879         kosten = 0.0f;
1880         setzeDateiname();
1881     }
1882
1883     void setzeDateiname(String s) {
1884         dateiname = s;
1885     }
1886
1887     void setzeDateiname() {
1888         dateiname = kassakurse;
1889     }

```

```

1890
1891 /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
1892   Methode erzeugeDatenvector eine Datenstruktur, in der die
1893   relevanten Daten gespeichert werden.
1894 */
1895 void leseDaten(String dateiname) {
1896     try{
1897         BufferedReader br = new BufferedReader(new FileReader(dateiname));
1898         String s;
1899         try{
1900             while((s = br.readLine())!=null) {
1901                 if(!s.startsWith("#")) {
1902                     erzeugeDatenvector(s);
1903                 }
1904             }
1905         } catch (IOException e) {
1906             System.err.println("Lesefehler bei Datei " + dateiname);
1907         }
1908     } catch (FileNotFoundException e) {
1909         System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
1910     }
1911 }
1912
1913 /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
1914   gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
1915 */
1916 void erzeugeDatenvector(String s){
1917     if(selektiere(s) == true) {
1918         Datum datum = new Datum(s.substring(0,8));
1919         String wkn = s.substring(9,18);
1920         String name = s.substring(19, s.length()-7);
1921         Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
1922         Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
1923         vec.addElement(zrd);
1924     }
1925 }
1926
1927 boolean selektiere(String s) {
1928     Datum d = new Datum();
1929     d.setzeDatum(s.substring(0,8));
1930     return (!d.kleingleich(vonDatum));
1931 }
1932
1933 /* Beschreibung des Quicksort siehe oben
1934 */
1935 void qsort(int left, int right) {
1936     int middle;
1937     if(left < right){
1938         middle = partition(left, right);
1939         qsort(left,middle-1);
1940         qsort(middle+1, right);
1941     }
1942 }
1943
1944 /* Hilfsprozedur fuer den Quicksort
1945 */
1946 int partition(int left, int right) {
1947     Zeitraumdaten pivot = (Zeitraumdaten)vec.elementAt(left);
1948     int l = left;
1949     int r = right;
1950     int middle;
1951     while(l < r) {
1952         while((l ≤ right) && (((Zeitraumdaten)vec.elementAt(l)).vergleiche(pivot))){
1953             l = l+1;

```

```

1954     }
1955     while((r ≥ left) && (!(Zeitraumdaten).vec.elementAt(r)).vergleiche(pivot)){
1956         r = r-1;
1957     }
1958     if(l < r) {
1959         exchange(l, r);
1960     }
1961 }
1962 middle = r;
1963 exchange (left, middle);
1964 return middle;
1965 }
1966
1967 /* Hilfsprozedur fuer den Quicksort
1968 */
1969 void exchange(int i, int j) {
1970     Zeitraumdaten z = (Zeitraumdaten).vec.elementAt(i);
1971     vec.setElementAt(vec.elementAt(j),i);
1972     vec.setElementAt(z,j);
1973 }
1974
1975 /* Ruft Quicksort auf
1976 */
1977 void sortiere() {
1978     qsort(0, vec.size()-1);
1979 }
1980
1981 /* Schliesst das Fenster und gibt seine Ressourcen frei.
1982 */
1983 void schliesseFenster() {
1984     this.dispose();
1985 }
1986
1987 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
1988 */
1989 void beendeProgramm() {
1990     System.exit(0);
1991 }
1992
1993 /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
1994 Knöpfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
1995 anderen Ereignisse wird "myaction" aufgerufen.
1996 */
1997 public boolean action(Event event, Object target) {
1998     if(event.id == Event.ACTION_EVENT) {
1999         if(event.target == schliessen) {
2000             schliesseFenster();
2001             return true;
2002         }
2003         else if(event.target == beenden) {
2004             beendeProgramm();
2005             return true;
2006         } else {
2007             return myaction(event, target);
2008         }
2009     } else {
2010         return false;
2011     }
2012 }
2013
2014 /* Wird von "action" aufgerufen.
2015 */
2016 boolean myaction(Event ev, Object target) {
2017     return true;

```

```

2018 }
2019
2020 /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
2021    zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
2022    noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
2023    "stelleDar" aufgerufen.
2024    */
2025 void stelleDar(){
2026     leseDaten(dateiname);
2027     zeigeTabelleAn();
2028 }
2029
2030 void zeigeTabelleAn(){
2031     textarea = new TextArea(20,40);
2032     p2.add("Center",textarea);
2033     sortiere();
2034     for(int i = 0; i < vec.size(); i++) {
2035         textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
2036                        (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
2037                        (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
2038                        ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
2039                        ((Zeitraumdaten)vec.elementAt(i)).name + " " +
2040                        ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
2041     }
2042     if(vec.size() == 0) {
2043         textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
2044     }
2045     this.pack();
2046     this.show();
2047 }
2048 }
2049
2050
2051 /* Klasse MonatsDynamikTabelle
2052
2053 Implementiert ein Fenster, in dem eine tabellarische Darstellung
2054 der prozentualen Unterschiede zwischen aufeinanderfolgenden
2055 Kassakursen aller Aktientitel fuer den letzten Monat angezeigt wird.
2056 Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
2057 des Programmes, berechnet fuer das Abrufen von Informationen keine
2058 Kosten (Kosten werden von einem Sponsor getragen).
2059 Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
2060 Daten jedoch daraus erzeugt werden, wird durch die Methode
2061 erzeugeDatenvector() entschieden. Diese entscheidet durch eine
2062 selektiere()-Methode, welche Datensaeetze ueberhaupt ausgewaehlt werden sollen.
2063 Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
2064 Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
2065 werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
2066
2067 Methoden:
2068 MonatsDynamikTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor
2069 einfaegt, die Kosten auf 0.00 setzt und die Datei
2070 mit den Daten angibt.
2071 leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
2072 der Methode erzeugeDatenvector() in den Vector vec
2073 selektiere(String s) entscheidet, ob der String ausgewaehlt werden
2074 soll, konkret: ob er von dem angegebenen Datum ist
2075 und die entsprechende Wertpapierkennnummer hat
2076 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
2077 sind Methoden, die einen Sortieralgorithmus zur
2078 Verfuegung stellen.
2079 schliesseFenster() kehrt zum aufrufenden Fenster zurueck
2080 beendeProgramm() beendet das Programm komplett
2081 action(Event, Object) verarbeitet Interaktionseignisse

```

```

2082  myaction(Event, Object) wird in "action" aufgerufen
2083 */
2084
2085 class MonatsDynamikTabelle extends Frame implements Konstanten {
2086     Panel p1 = new Panel();
2087     Panel p2 = new Panel();
2088     Panel p3 = new Panel();
2089     Button schliessen = new Button("Schliessen");
2090     Button beenden = new Button("Beenden");
2091     float kosten;
2092     String dateiname;
2093     Label sponsor;
2094     Vector vec = new Vector();
2095     Vector dyn = new Vector();
2096     Datum vonDatum;
2097     TextArea textarea;
2098
2099     /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
2100        bestimmt das Layout.
2101        Panel p1 enthaelt die Sponsorinformation
2102        Panel p2 enthaelt eine Auswahlmoeglichkeit des gewuenschten Wertpapiers
2103        Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
2104        des Programms.
2105        Die Kosten werden auf 0.00 gesetzt.
2106     */
2107
2108     MonatsDynamikTabelle(String titel) {
2109         super(titel);
2110         vonDatum = heute.monatzurueck();
2111         this.setLayout(new BorderLayout());
2112         p1.setLayout(new FlowLayout());
2113         add("North",p1);
2114         p2.setLayout(new BorderLayout());
2115         add("Center",p2);
2116         p3.setLayout(new FlowLayout());
2117         add("South",p3);
2118         p3.add(schliessen);
2119         p3.add(beenden);
2120         sponsor = new Label("Diese Seiten werden gesponsort von xxy");
2121         p1.add(sponsor);
2122         kosten = 0.0f;
2123         setzeDateiname();
2124     }
2125
2126     void setzeDateiname(String s) {
2127         dateiname = s;
2128     }
2129
2130     void setzeDateiname() {
2131         dateiname = kassakurse;
2132     }
2133
2134     /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
2135        Methode erzeugeDatenvector eine Datenstruktur, in der die
2136        relevanten Daten gespeichert werden.
2137     */
2138     void leseDaten(String dateiname) {
2139         try{
2140             BufferedReader br = new BufferedReader(new FileReader(dateiname));
2141             String s;
2142             try{
2143                 while((s = br.readLine())!=null) {
2144                     if(!s.startsWith("#")) {
2145                         erzeugeDatenvector(s);

```



```

2146     }
2147     }
2148     } catch (IOException e) {
2149         System.err.println("Lesefehler bei Datei " + dateiname);
2150     }
2151     } catch (FileNotFoundException e) {
2152         System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
2153     }
2154 }
2155
2156 /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
2157    gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
2158    */
2159 void erzeugeDatenvector(String s){
2160     if(selektiere(s) == true) {
2161         Datum datum = new Datum(s.substring(0,8));
2162         String wkn = s.substring(9,18);
2163         String name = s.substring(19, s.length()-7);
2164         Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
2165         Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
2166         vec.addElement(zrd);
2167     }
2168 }
2169
2170 boolean selektiere(String s) {
2171     Datum d = new Datum();
2172     d.setzeDatum(s.substring(0,8));
2173     return (!d.kleinergleich(vonDatum));
2174 }
2175
2176 /* Beschreibung des Quicksort siehe oben
2177    */
2178 void qsort(int left, int right) {
2179     int middle;
2180     if(left < right){
2181         middle = partition(left, right);
2182         qsort(left,middle-1);
2183         qsort(middle+1, right);
2184     }
2185 }
2186
2187 /* Hilfsprozedur fuer den Quicksort
2188    */
2189 int partition(int left, int right) {
2190     Zeitraumdaten pivot = (Zeitraumdaten)vec.elementAt(left);
2191     int l = left;
2192     int r = right;
2193     int middle;
2194     while(l < r) {
2195         while((l ≤ right) && (((Zeitraumdaten)vec.elementAt(l)).vergleiche(pivot))){
2196             l = l+1;
2197         }
2198         while((r ≥ left) && (!(Zeitraumdaten)vec.elementAt(r)).vergleiche(pivot)){
2199             r = r-1;
2200         }
2201         if(l < r) {
2202             exchange(l, r);
2203         }
2204     }
2205     middle = r;
2206     exchange (left, middle);
2207     return middle;
2208 }
2209

```

```

2210  /* Hilfsprozedur fuer den Quicksort
2211  */
2212  void exchange(int i, int j) {
2213      Zeitraumdaten z = (Zeitraumdaten)vec.elementAt(i);
2214      vec.setElementAt(vec.elementAt(j),i);
2215      vec.setElementAt(z,j);
2216  }
2217
2218  /* Ruft Quicksort auf
2219  */
2220  void sortiere() {
2221      qsort(0, vec.size()-1);
2222  }
2223
2224  /* Schliesst das Fenster und gibt seine Ressourcen frei.
2225  */
2226  void schliesseFenster() {
2227      this.dispose();
2228  }
2229
2230  /* Beendet das Programm komplett: alle Fenster werden geschlossen.
2231  */
2232  void beendeProgramm() {
2233      System.exit(0);
2234  }
2235
2236  /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
2237  Knöpfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
2238  anderen Ereignisse wird "myaction" aufgerufen.
2239  */
2240  public boolean action(Event event, Object target) {
2241      if(event.id == Event.ACTION_EVENT) {
2242          if(event.target == schliessen) {
2243              schliesseFenster();
2244              return true;
2245          }
2246          else if(event.target == beenden) {
2247              beendeProgramm();
2248              return true;
2249          } else {
2250              return myaction(event, target);
2251          }
2252      } else {
2253          return false;
2254      }
2255  }
2256
2257  /* Wird von "action" aufgerufen.
2258  */
2259  boolean myaction(Event ev, Object target) {
2260      return true;
2261  }
2262
2263  /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
2264  zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
2265  noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
2266  "stelleDar" aufgerufen.
2267  */
2268  void stelleDar(){
2269      leseDaten(dateiname);
2270      zeigeTabelleAn();
2271  }
2272
2273  float berechneDynamik(int i) {

```

```

2274 //suche den letzten Wert des gleichen Wertpapiers, um die prozentuale
2275 //Abweichung des neuen Wertes vom alten zu bestimmen.
2276 int j = i-1;
2277 boolean gefunden = false;
2278 while ((j ≥ 0) && !gefunden) {
2279     if (((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer).compareTo(
2280         ((Zeitraumdaten)vec.elementAt(j)).wertpapierkennnummer) == 0) {
2281         gefunden = true;
2282         return (((((Zeitraumdaten)vec.elementAt(i)).kassakurs).kurs -
2283             (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)/
2284             (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)*100);
2285     }
2286     j--;
2287 }
2288 return 0.0f;
2289 }
2290
2291 void zeigeTabelleAn(){
2292     textarea = new TextArea(20,40);
2293     p2.add("Center",textarea);
2294     sortiere();
2295     for(int i = 0; i < vec.size(); i++) {
2296         textarea.append(((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
2297             (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
2298             (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
2299             ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
2300             ((Zeitraumdaten)vec.elementAt(i)).name + " " +
2301             berechneDynamik(i) + "%\n");
2302     }
2303     if(vec.size() == 0) {
2304         textarea.append("Fuer diesen Zeitraum existieren keine Daten\n");
2305     }
2306     this.pack();
2307     this.show();
2308 }
2309
2310 }
2311
2312
2313 /* graphische Darstellung aller Kurse eines Aktientitels fuer den heutigen
2314 Tag.
2315 */
2316 class TagesChartAnzeiger extends Canvas {
2317     Vector v;
2318     int xSize;
2319     int ySize;
2320     int abstand;
2321
2322     TagesChartAnzeiger(Vector vec, int xSize, int ySize) {
2323         v = vec;
2324         this.xSize = xSize;
2325         this.ySize = ySize;
2326         if(vec.size() ≠ 0) {
2327             abstand = xSize/vec.size();
2328         }
2329         setSize(xSize, ySize);
2330     }
2331
2332     public void paint(Graphics g) {
2333         g.drawRect(0, 0, xSize-1, ySize-1);
2334         for(int i = 0; i < v.size()-1; i++) {
2335             drawLine(g, i);
2336         }
2337     }

```

```

2338
2339 void drawLine(Graphics g, int i){
2340     g.drawLine(i*abstand,
2341               ySize - (int)(((TagesChartWert)(v.elementAt(i))).kurs).kurs,
2342               (i+1)*abstand,
2343               ySize - (int)(((TagesChartWert)(v.elementAt(i+1))).kurs).kurs));
2344 }
2345 }
2346
2347
2348 /* graphische Darstellung aller Kassakurse eines Aktientitels fuer die
2349 letzten 30 Tage.
2350 */
2351 class MonatsChartAnzeiger extends Canvas {
2352     Vector v;
2353     int xSize;
2354     int ySize;
2355     int abstand;
2356
2357     MonatsChartAnzeiger(Vector vec, int xSize, int ySize) {
2358         v = vec;
2359         this.xSize = xSize;
2360         this.ySize = ySize;
2361         if(vec.size() ≠ 0) {
2362             abstand = xSize/vec.size();
2363         }
2364         setSize(xSize, ySize);
2365     }
2366
2367     public void paint(Graphics g) {
2368         g.drawRect(0, 0, xSize-1, ySize-1);
2369         for(int i = 0; i < v.size()-1; i++) {
2370             drawLine(g, i);
2371         }
2372     }
2373
2374     void drawLine(Graphics g, int i){
2375         g.drawLine(i*abstand,
2376                 ySize - (int)(((MonatsChartWert)(v.elementAt(i))).kassakurs).kurs,
2377                 (i+1)*abstand,
2378                 ySize - (int)(((MonatsChartWert)(v.elementAt(i+1))).kassakurs).kurs));
2379     }
2380 }
2381
2382
2383 /* Kern des Hauptprogramms: Auswahlnenue der einzelnen Darstellungsformen.
2384 */
2385 class Auswahl extends Frame {
2386     Button beenden = new Button("Beenden");
2387     Button ok = new Button("OK");
2388     Panel p = new Panel();
2389     CheckboxGroup cbg = new CheckboxGroup();
2390     Checkbox cbtt = new Checkbox("Tagestabelle", cbg, true);
2391     Checkbox cbwt = new Checkbox("Wochentabelle", cbg, false);
2392     Checkbox cbmt = new Checkbox("Monatstabelle", cbg, false);
2393     Checkbox cbdt = new Checkbox("Dynamiktabelle", cbg, false);
2394     Checkbox cbtg = new Checkbox("Tagesgewinne", cbg, false);
2395     Checkbox cbltk = new Checkbox("letzte Tageskurse", cbg, false);
2396     Checkbox cbtc = new Checkbox("Tages-Chart", cbg, false);
2397     Checkbox cbmc = new Checkbox("Monats-Chart", cbg, false);
2398
2399     Auswahl(String titel) {
2400         super(titel);
2401         setLayout(new BorderLayout());

```

```

2402     p.setLayout(new GridLayout(6,1));
2403     p.add(cbtt);
2404     p.add(cbwt);
2405     p.add(cbmt);
2406     p.add(cbdt);
2407     p.add(cbtg);
2408     p.add(cbltk);
2409     p.add(cbtc);
2410     p.add(cbmc);
2411     add("Center",p);
2412     add("East",ok);
2413     add("South",beenden);
2414     pack();
2415     show();
2416 }
2417
2418 public boolean action(Event ev, Object target){
2419     if(ev.id == Event.ACTION_EVENT) {
2420         if(ev.target == ok) {
2421             if(cbg.getSelectedCheckbox() == cbtt) {
2422                 TagesTabelle tt = new TagesTabelle("Tagestabelle");
2423                 tt.stelleDar();
2424                 return true;
2425             } else if(cbg.getSelectedCheckbox() == cbwt) {
2426                 WochenTabelle wt = new WochenTabelle("Wochentabelle");
2427                 wt.stelleDar();
2428                 return true;
2429             } else if(cbg.getSelectedCheckbox() == cbmt) {
2430                 MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
2431                 mt.stelleDar();
2432                 return true;
2433             } else if(cbg.getSelectedCheckbox() == cbdt) {
2434                 MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
2435                 dt.stelleDar();
2436                 return true;
2437             } else if(cbg.getSelectedCheckbox() == cbtg) {
2438                 Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
2439                 tg.stelleDar();
2440                 return true;
2441             } else if(cbg.getSelectedCheckbox() == cbltk) {
2442                 LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
2443                 ltg.stelleDar();
2444                 return true;
2445             } else if(cbg.getSelectedCheckbox() == cbtc) {
2446                 TagesChart tc = new TagesChart("Tageschart");
2447                 tc.stelleDar();
2448                 return true;
2449             } else if(cbg.getSelectedCheckbox() == cbmc) {
2450                 MonatsChart mc = new MonatsChart("Monatschart");
2451                 mc.stelleDar();
2452                 return true;
2453             }
2454         } else if(ev.target == beenden) {
2455             System.exit(0);
2456             return true;
2457         }
2458     }
2459     return false;
2460 }
2461 }
2462
2463
2464 /* Programmeinstiegspunkt
2465 */

```

```
2466 class Boerse {
2467     public static void main(String args[]) {
2468         Auswahl a = new Auswahl("Boersen-Information");
2469     }
2470 }
2471
2472
2473
```

B.1.2 Boerse.java (3 levels of inheritance)

```

1  /*****/
2  /*Programmautor: Barbara Unger */
3  /*Datum:      1997-06-03 */
4  /*Gruppe:     Boerse3 */
5  /*****/
6
7  import java.awt.*;
8  import java.util.Vector;
9  import java.util.Enumeration;
10 import java.util.Date;
11 import java.io.*;
12
13 /* Programm zum Betrachten von Boersenkursen.
14    Abfolge der Klassen in dieser Datei:
15
16    interface Konstanten
17    abstract class BoersenInfo extends Frame implements Konstanten
18    abstract class KostenloseInfos extends BoersenInfo
19    abstract class KostenpflichtigeInfos extends BoersenInfo
20    abstract class Chart extends KostenloseInfos
21    class TagesChart extends Chart
22    class TagesChartWert implements Comparable
23    class MonatsChart extends Chart
24    class MonatsChartWert implements Comparable
25    abstract class Tabelle extends KostenloseInfos
26    class TagesTabelle extends Tabelle
27    class Tagesdaten implements Comparable
28    class Zeitraumdaten implements Comparable
29    class Datum
30    class Uhrzeit
31    class Kurs
32    class Tagesgewinne extends Tabelle
33    class LetzteTageskurse extends Tabelle
34    class WochenTabelle extends Tabelle
35    class MonatsTabelle extends Tabelle
36    class MonatsDynamikTabelle extends Tabelle
37    interface Comparable
38    abstract class ChartAnzeiger extends Canvas
39    class TagesChartAnzeiger extends ChartAnzeiger
40    class MonatsChartAnzeiger extends ChartAnzeiger
41    class Auswahl extends Frame
42    class Boerse
43 */
44
45 interface Konstanten {
46 // static Date date = new Date();
47 // static Datum heute = new Datum(date.getDay(), date.getMonth(), date.getYear());
48 static Datum heute = new Datum(27, 5, 97);
49 static String kurse = "KURSE";
50 static String kassakurse = "KASSAKURSE";
51 }
52
53
54 /* abstrakte Klasse BoersenInfo Implementiert Fenster, in denen
55    Boersen-Informationen angezeigt werden. Stellt sicher, dass in
56    jeder der daraus abgeleiteten Klassen Knoepfe zum Schliessen des

```

```

57     Fensters und zum Beenden des Programmes existieren.
58
59     Methoden:
60     BoersenInfo(String) erzeugt das Fenster und setzt den Titel
61     schliesseFenster() kehrt zum aufrufenden Fenster zurueck
62     beendeProgramm() beendet das Programm komplett
63     action(Event, Object) verarbeitet Interaktionsereignisse
64     abstract myaction(Event, Object)
65     wird in "action" aufgerufen; Unterklassen
66     koennen hier durch ueberschreiben
67     spezielle Ereignisbehandlung
68     implementieren
69 */
70
71 abstract class BoersenInfo extends Frame implements Konstanten{
72     Panel p1 = new Panel();
73     Panel p2 = new Panel();
74     Panel p3 = new Panel();
75     Button schliessen = new Button("Schliessen");
76     Button beenden = new Button("Beenden");
77     float kosten;
78
79     /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und bestimmt das Layout.
80     Panel p1 dient zum Darstellen von Sponsor- oder Kosteninformationen,
81     Panel p2 dient zum Anzeigen der angeforderten Informationen,
82     Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden des Programms.
83     */
84     BoersenInfo(String titel) {
85         super(titel);
86         this.setLayout(new BorderLayout());
87         p1.setLayout(new FlowLayout());
88         add("North",p1);
89         p2.setLayout(new BorderLayout());
90         add("Center",p2);
91         p3.setLayout(new FlowLayout());
92         add("South",p3);
93         p3.add(schliessen);
94         p3.add(beenden);
95     }
96
97     /* Schliesst das Fenster und gibt seine Ressourcen frei.
98     */
99     void schliesseFenster() {
100         this.dispose();
101     }
102
103     /* Beendet das Programm komplett: alle Fenster werden geschlossen.
104     */
105     void beendeProgramm() {
106         System.exit(0);
107     }
108
109     /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
110     Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
111     anderen Ereignisse wird "myaction" aufgerufen.
112     */
113     public boolean action(Event event, Object target) {
114         if(event.id == Event.ACTION_EVENT) {
115             if(event.target == schliessen) {
116                 schliesseFenster();
117                 return true;
118             }
119             else if(event.target == beenden) {
120                 beendeProgramm();

```



```

121     return true;
122   } else {
123     return myaction(event, target);
124   }
125   } else {
126     return false;
127   }
128 }
129
130 /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
131 werden, um eine spezielle Ereignisbehandlung zu
132 implementieren. (Diese Konstruktion verhindert, dass die Behandlung
133 der Knoepfe "Schliessen" und "Beenden" jedesmal neu implementiert
134 werden muss.
135 */
136 abstract boolean myaction(Event ev, Object target);
137 }
138
139
140 /* abstrakte Klasse KostenloseInfos
141 Stellt sicher, dass fuer das Abrufen von Informationen durch alle
142 aus ihr abgeleiteten Klassen keine Kosten anfallen. (Kosten werden
143 von einem Sponsor getragen).
144 Des weiteren stellt sie eine Methode zum Dateneinlesen
145 bereit. Welche Daten jedoch daraus erzeugt werden ist zu diesem
146 Zeitpunkt unklar und wird erst in einer Unterklasse entschieden
147 durch die Methode erzeugeDatenvector(). Ausserdem wird nun schon
148 ein Sponsorlabel fuer alle kostenlosen Infos gesetzt.
149
150 Methoden:
151 KostenloseInfos(String) Konstruktor, setzt Fenstertitel, fuegt einen Sponsor ein
152 und setzt die Kosten auf 0.00.
153 leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
154 der Methode erzeugeDatenvector() in den Vector vec
155 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
156 sind Methoden, die einen Sortieralgorithmus fuer alle
157 Unterklassen zur Verfuegung stellen. Da in jeder
158 Unterklasse unterschiedliche Datentypen zu sortieren
159 sind, wurde Schablonenmethode angewandt, die
160 vergleiche()-Methode ist nur abstrakt realisiert
161 und wird vom jeweiligen Datentyp erst implementiert.
162 */
163
164 abstract class KostenloseInfos extends BoersenInfo {
165   String dateiname;
166   Label sponsor;
167   Vector vec = new Vector();
168
169   /* Konstruktor, setzt den Titel des Fensters, fuegt einen Sponsor in das
170   Fenster ein (in p1) und setzt die Kosten auf 0.00
171   */
172   KostenloseInfos(String titel) {
173     super(titel);
174     sponsor = new Label("Diese Seiten werden gesponsort von xxy");
175     p1.add(sponsor);
176     kosten = 0.0f;
177   }
178
179   /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
180   abstrakte Methode erzeugeDatenvector eine Datenstruktur, in der die
181   relevanten Daten gespeichert werden.
182   */
183   void leseDaten(String dateiname) {
184     try{

```

```

185     BufferedReader br = new BufferedReader(new FileReader(dateiname));
186     String s;
187     try{
188         while((s = br.readLine()) != null) {
189             if(!s.startsWith("#")) {
190                 erzeugeDatenvector(s);
191             }
192         }
193     } catch (IOException e) {
194         System.err.println("Lesefehler bei Datei " + dateiname);
195     }
196 } catch (FileNotFoundException e) {
197     System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
198 }
199 }
200
201 /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
202 gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
203 */
204 abstract void erzeugeDatenvector(String s);
205
206 /* Beschreibung des Quicksort siehe oben
207 */
208 void qsort(int left, int right) {
209     int middle;
210     if(left < right){
211         middle = partition(left, right);
212         qsort(left,middle-1);
213         qsort(middle+1, right);
214     }
215 }
216
217 /* Hilfsprozedur fuer den Quicksort
218 */
219 int partition(int left, int right) {
220     Comparable pivot = (Comparable)vec.elementAt(left);
221     int l = left;
222     int r = right;
223     int middle;
224     while(l < r) {
225         while((l ≤ right) && (((Comparable)vec.elementAt(l)).vergleiche(pivot))){
226             l = l+1;
227         }
228         while((r ≥ left) && (!((Comparable)vec.elementAt(r)).vergleiche(pivot))){
229             r = r-1;
230         }
231         if(l < r) {
232             exchange(l, r);
233         }
234     }
235     middle = r;
236     exchange (left, middle);
237     return middle;
238 }
239
240 /* Hilfsprozedur fuer den Quicksort
241 */
242 void exchange(int i, int j) {
243     Comparable c = (Comparable)vec.elementAt(i);
244     vec.setElementAt(vec.elementAt(j),i);
245     vec.setElementAt(c,j);
246 }
247
248 /* Ruft Quicksort auf

```

```

249  */
250  void sortiere() {
251      qsort(0, vec.size()-1);
252  }
253 }
254
255
256 /* Klasse KostenpflichtigeInfos
257    enthaelt einen Rahmen fuer Informationen, die vom Kunden bezahlt werden
258    muessen, wie beispielsweise Prognosen, etc.
259 */
260 /*Wird nicht ausprogrammiert.*/
261 abstract class KostenpflichtigeInfos extends BoersenInfo {
262
263     KostenpflichtigeInfos(String titel) {
264         super(titel);
265         kosten = 1.0f;
266     }
267 }
268
269
270 /* abstrakte Klasse Chart
271    Diese Klasse enthaelt allgemeine Methoden zur Darstellung eines Charts.
272    Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
273    werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
274
275    Methoden:
276    Chart(String)      Konstruktor, setzt den Fenstertitel
277    void stelleDar()   stellt eine Dialogbox dar.
278    myaction()        erweert die action-Methode der Oberklasse.
279    abstract zeigeCanvasAn() liefert ein Canvas zurueck.
280 */
281
282 abstract class Chart extends KostenloseInfos {
283     int xChartSize = 200;
284     int yChartSize = 200;
285     String wkn;
286     Label l = new Label();
287     TextField t = new TextField("700045634",9);
288     Button ok = new Button("ok");
289
290     /* Konstruktor, ruft den Konstruktor von KostenloseInfos auf, und fuegt in den
291        freien Rahmen p2 eine Auswahlmoeglichkeit des gewuenschten Wertpapiers ein
292     */
293     Chart(String titel) {
294         super(titel);
295     }
296
297     void stelleDar(){
298         p2.add("North",l);
299         p2.add("Center",t);
300         p2.add("South",ok);
301         pack();
302         show();
303     }
304
305
306     /* myaction erweert die action-Methode, so dass nun beim Druck des
307        Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
308        aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
309        und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
310     */
311     boolean myaction(Event ev, Object target){
312         if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {

```

```

313     wkn = t.getText();
314     wkn.trim();
315     p2.removeAll();
316     leseDaten(dateiname);
317     p2.add("Center", zeigeCanvasAn());
318     pack();
319     show();
320     return true;
321 }
322 return false;
323 }
324
325 /* liefert ein Canvas zurueck. Was das Canvas enthaelt, wird erst in der Unterklasse
326    bestimmt.
327 */
328 abstract Canvas zeigeCanvasAn();
329 }
330
331
332 /* Klasse TagesChart
333    In dieser Klasse wird das Erzeugen des Datenvektors implementiert.
334    Diese entscheidet durch eine selektiere()-Methode, welche
335    Datensatze ueberhaupt ausgewaehlt werden sollen.
336
337    Methoden:
338    TagesChart(String)      Konstruktor, der den Titel setzt und die Datei
339                           mit den Daten angibt.
340    erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
341                           entscheidet mit Hilfe von selektiere(), ob der
342                           Datensatz in den Datenvektor eingefuegt
343                           werden soll.
344    selektiere(String s)   entscheidet, ob der String ausgewaehlt werden
345                           soll, konkret: ob er von dem angegebenen
346                           Datum ist und die entsprechende
347                           Wertpapierkennnummer hat
348    zeigeCanvasAn()       erzeugt ein Objekt der Canvas-Klasse, in der der
349                           Chart aus den selektierten Daten, die im Datenvektor stehen,
350                           angezeigt wird.
351 */
352
353 class TagesChart extends Chart {
354     TagesChartWert tcw;
355
356     /* Konstruktor, der den Titel setzt und die Datei mit den Daten angibt
357     */
358     TagesChart(String titel) {
359         super(titel);
360         dateiname = kurse;
361         l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Tageschart an");
362     }
363
364     /* bekommt eine Datensatz in Stringform und entscheidet mit Hilfe von selektiere(),
365        ob der Datensatz in den Datenvektor eingefuegt werden soll.
366     */
367     void erzeugeDatenvector(String s){
368         if(selektiere(s) == true) {
369             tcw = new TagesChartWert(s);
370             vec.addElement(tcw);
371         }
372     }
373
374     /* entscheidet, ob der String ausgewaehlt werden soll, konkret: ob er von dem angegebenen
375        Datum ist und die entsprechende Wertpapierkennnummer hat
376     */

```

```

377 boolean selektiere(String s) {
378     Datum d = new Datum();
379     d.setzeDatum(s.substring(0,8));
380     if((heute.tag == d.tag) && (heute.monat ==d.monat) && (heute.jahr == d.jahr)) {
381         return (wkn.compareTo(s.substring(15,24))==0);
382     }
383     return false;
384 }
385
386 /* erzeugt ein Objekt der Canvas-Klasse, in der der Chart aus den selektierten Daten, die im
387    Datenvektor stehen, angezeigt wird.
388 */
389 Canvas zeigeCanvasAn() {
390     return new TagesChartAnzeiger(vec, xChartSize, yChartSize);
391 }
392 }
393
394
395
396 /*Klasse TagesChartWert
397    ist ein Objekttyp, der aus einer Uhrzeit und einem Kurswert besteht.
398    Das sind genau die Daten, die man zum Anzeigen eines Tagescharts braucht. Die Klasse hat
399    eine vergleiche-Methode, so dass man einen Vector mit Objekten dieses Typs mit
400    der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.
401
402    Methoden:
403    TagesChartWert(String) bekommt einen String, in dem die Uhrzeit
404    und der Kurs enthalten sind.
405    vergleiche(Comparable c) vergleicht die Uhrzeiten
406 */
407
408 class TagesChartWert implements Comparable {
409     Uhrzeit uhrzeit;
410     Kurs kurs;
411
412     /* bekommt einen String, in dem die Uhrzeit und der Kurs enthalten sind.
413     */
414     TagesChartWert(String s) {
415         uhrzeit = new Uhrzeit(s.substring(9,15));
416         kurs = new Kurs(s.substring(s.length()-7, s.length()));
417     }
418
419     /* vergleicht die Uhrzeiten zweier TagesChartWert-Objekte
420     */
421     public boolean vergleiche(Comparable c) {
422         return ((this.uhrzeit).kleinergleich(((TagesChartWert)c).uhrzeit));
423     }
424 }
425
426
427 /* Klasse MonatsChart
428    In dieser Klasse wird das Erzeugen des Datenvektors implementiert.
429    Diese entscheidet durch eine selektiere()-Methode, welche
430    Datensätze ueberhaupt ausgewaehlt werden sollen.
431
432    Methoden:
433    MonatsChart(String)        Konstruktor, der den Titel setzt und die Datei
434                                mit den Daten angibt.
435    erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
436                                entscheidet mit Hilfe von slektiere(), ob der
437                                Datensatz in den Datenvector eingefuegt
438                                werden soll.
439    selektiere(String s)       entscheidet, ob der String ausgewaehlt werden
440                                soll, konkret: ob er von dem angegebenen

```

```

441                                     Datum ist und die entsprechende
442                                     Wertpapierkennnummer hat
443         zeigeCanvasAn()                erzeugt ein Objekt der Canvas-Klasse, in der der
444                                     Chart angezeigt wird.
445 */
446
447 class MonatsChart extends Chart {
448     MonatsChartWert mcw;
449     Datum letzterMonat = new Datum();
450
451     MonatsChart(String titel) {
452         super(titel);
453         l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Monatschart an");
454         dateiname = kassakurse;
455         letzterMonat = heute.monatzurueck();
456     }
457
458     void erzeugeDatenvector(String s){
459         if(selektiere(s) == true) {
460             mcw = new MonatsChartWert(s);
461             vec.addElement(mcw);
462         }
463     }
464
465     boolean selektiere(String s) {
466         Datum d = new Datum();
467         d.setzeDatum(s.substring(0,8));
468         if(!d.kleinerleich(letzterMonat)) {
469             return (wkn.compareTo(s.substring(9,18))==0);
470         }
471         return false;
472     }
473
474     Canvas zeigeCanvasAn() {
475         return new MonatsChartAnzeiger(vec, xChartSize, yChartSize);
476     }
477 }
478
479
480 /*Klasse MonatsChartWert
481 ist ein Objekttyp, der aus einem Datum und einem Kassakurswert besteht,
482 Daten, die man zum Anzeigen eines Monatschart braucht. Die Klasse hat
483 eine vergleiche-Methode, so dass man einen Vector mit diesem Typ mit
484 der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.
485
486 Methoden:
487 MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
488 und der Kurs enthalten sind.
489 vergleiche(Comparable c) vergleicht das Datum
490 */
491
492 class MonatsChartWert implements Comparable {
493     Kurs kassakurs;
494     Datum datum;
495
496     MonatsChartWert(String s){
497         kassakurs = new Kurs(s.substring(s.length()-6, s.length()));
498         datum = new Datum(s.substring(0,9));
499     }
500
501     public boolean vergleiche(Comparable c){
502         return ((this.datum).kleinerleich(((MonatsChartWert)c).datum));
503     }
504 }

```

```

505
506
507 /* abstrakte Klasse Tabelle.
508   Gemeinsame Oberklasse fuer alle tabellarischen Darstellungen von
509   Kursen oder Kassakursen.
510 */
511
512 abstract class Tabelle extends KostenloseInfos {
513     String dateiname;
514
515     Tabelle(String titel) {
516         super(titel);
517         setzeDateiname();
518     }
519
520 /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
521   zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
522   noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
523   "stelleDar" aufgerufen.
524 */
525     void stelleDar(){
526         leseDaten(dateiname);
527         zeigeTabelleAn();
528     }
529
530     abstract void setzeDateiname();
531     abstract void zeigeTabelleAn();
532 }
533
534 /* Klasse zur tabellarischen Darstellung aller Kurse eines Tages
535 */
536
537 class TagesTabelle extends Tabelle {
538     TextArea textarea;
539
540     TagesTabelle(String titel) {
541         super(titel);
542     }
543
544     void setzeDateiname(String s) {
545         dateiname = s;
546     }
547
548     void setzeDateiname() {
549         dateiname = kurse;
550     }
551
552     boolean selektiere(String s) {
553         Datum d = new Datum();
554         d.setzeDatum(s.substring(0,8));
555         return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
556     }
557
558
559     void erzeugeDatenvector(String s) {
560         if(selektiere(s) == true) {
561             Datum datum = new Datum(s.substring(0,8));
562             Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
563             Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
564             String wkn = s.substring(14,24);
565             String name = s.substring(25, s.length()-7);
566             Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
567             vec.addElement(tagesdaten);
568         }

```

```

569 }
570
571 boolean myaction(Event ev, Object target){
572     return true;
573 }
574
575 void zeigeTabelleAn(){
576     textarea = new TextArea(20,40);
577     p2.add("Center",textarea);
578     sortiere();
579     for(int i = 0; i < vec.size(); i++) {
580         textarea.append((((Tagesdaten)vec.elementAt(i)).uhrzeit.stunde + " : " +
581             (((Tagesdaten)vec.elementAt(i)).uhrzeit.minute + " " +
582             ((Tagesdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
583             ((Tagesdaten)vec.elementAt(i)).name + " " +
584             ((Tagesdaten)vec.elementAt(i)).kurs.kurs + "\n"));
585     }
586     if(vec.size() == 0) {
587         textarea.append("Fuer das heutige Datum, " + heute.tag + ". " +
588             heute.monat + ". " + heute.jahr +
589             ", existieren keine Daten\n");
590     }
591     this.pack();
592     this.show();
593 }
594 }
595
596
597 /* Klasse zur Repraesentation eines einzelnen Kursinformations-Datensatzes
598 */
599
600 class Tagesdaten implements Comparable {
601     Datum datum;
602     Uhrzeit uhrzeit;
603     String wertpapierkennnummer;
604     String name;
605     Kurs kurs;
606
607     Tagesdaten(Datum d, Uhrzeit u, String wkn, String n, Kurs k) {
608         datum = d;
609         uhrzeit = u;
610         wertpapierkennnummer = wkn;
611         name = n;
612         kurs = k;
613     }
614
615     public boolean vergleiche(Comparable c) {
616         return (this.uhrzeit).kleinergleich(((Tagesdaten)c).uhrzeit);
617     }
618 }
619
620
621 /* Klasse zur Repraesentation eines einzelnen Kassakursinformations-Datensatzes
622 */
623
624 class Zeitraumdaten implements Comparable {
625     Datum datum;
626     String wertpapierkennnummer;
627     String name;
628     Kurs kassakurs;
629
630     Zeitraumdaten(Datum d, String wkn, String n, Kurs k) {
631         datum = d;
632         wertpapierkennnummer = wkn;

```



```

633     name = n;
634     kassakurs = k;
635 }
636
637 public boolean vergleiche(Comparable c) {
638     return (this.datum).kleinergleich(((Zeitraumdaten)c).datum);
639 }
640 }
641
642
643 /* Klasse zur Repraesentation eines Datums, incl. Berechnung des
644    Datums 'vor einer Woche' und 'vor einem Monat'.
645    (Anmerkung zum Experiment:
646     Diese Routinen sind vereinfacht, um den Code zu verkuerzen.
647     Sie funktionieren nicht wirklich korrekt. Das soll uns hier aber mal
648     gerade nicht stoeren...)
649 */
650
651 class Datum {
652     int tag;
653     int monat;
654     int jahr;
655
656     Datum() {
657         tag = 0;
658         monat = 0;
659         jahr = 0;
660     }
661
662     Datum(String s){
663         tag = new Integer(s.substring(0,2)).intValue();
664         monat = new Integer(s.substring(3,5)).intValue();
665         jahr = new Integer(s.substring(6,8)).intValue();
666     }
667
668     Datum(int t, int m, int j){
669         tag = t;
670         monat = m;
671         jahr = j;
672     }
673
674     void setzeDatum(String s) {
675         tag = new Integer(s.substring(0,2)).intValue();
676         monat = new Integer(s.substring(3,5)).intValue();
677         jahr = new Integer(s.substring(6,8)).intValue();
678     }
679
680     void setzeDatum(int t, int m, int j) {
681         tag = t;
682         monat = m;
683         jahr = j;
684     }
685
686     boolean kleinergleich(Datum d) {
687         if(this.jahr < d.jahr) {
688             return true;
689         } else if(this.jahr == d.jahr){
690             if(this.monat < d.monat) {
691                 return true;
692             } else if(this.monat == d.monat){
693                 if(this.tag <= d.tag) {
694                     return true;
695                 }
696             }

```

```

697     }
698     return false;
699 }
700
701 Datum monatzurueck() {
702     Datum d = new Datum();
703     d.tag = tag;
704     d.jahr = jahr;
705     if(monat == 1) {
706         d.monat = 12;
707         d.jahr = jahr - 1;
708     } else {
709         d.monat = monat - 1;
710     }
711     return d;
712 }
713
714 /* vereinfachte Berechnung: Alle Monate haben 30 Tage
715 */
716 Datum wochezurueck() {
717     Datum d = new Datum();
718     d.tag = tag - 7;
719     d.monat = monat;
720     d.jahr = jahr;
721     if(d.tag < 1) {
722         d.tag = d.tag + 30;
723         d.monat = monat - 1;
724         if(d.monat < 1) {
725             d.monat = d.monat + 12;
726             d.jahr = jahr - 1;
727         }
728     }
729     return d;
730 }
731
732 void print() {
733     System.out.println(" Datum = " + tag + " . " + monat + " . " + jahr);
734 }
735 }
736
737 /* Klasse zur Repraesentation einer Uhrzeit
738 */
739
740 class Uhrzeit {
741     int stunde;
742     int minute;
743
744     Uhrzeit(String s) {
745         stunde = (new Integer(s.substring(0,2))).intValue();
746         minute = (new Integer(s.substring(3,5))).intValue();
747     }
748
749     boolean kleinergleich(Uhrzeit u) {
750         if(this.stunde < u.stunde) {
751             return true;
752         } else if(this.stunde == u.stunde) {
753             if(this.minute ≤ u.minute) {
754                 return true;
755             }
756         }
757         return false;
758     }
759 }
760

```

```

761  /* Klasse zur Repraesentation eines Kurses oder Kassakurses
762  */
763
764  class Kurs {
765      float kurs;
766
767      Kurs(String s){
768          kurs = (new Float(s)).floatValue();
769      }
770 }
771
772
773  /* Tabellarische Darstellung der prozentualen Unterschiede zwischen dem
774  ersten und letzten Kurs jeder Aktie am heutigen Datum.
775  */
776
777  class Tagesgewinne extends Tabelle {
778      TextArea textarea;
779      Vector v = new Vector();
780
781      Tagesgewinne(String titel) {
782          super(titel);
783      }
784
785      void setzeDateiname(String s) {
786          dateiname = s;
787      }
788
789      void setzeDateiname() {
790          dateiname = kurse;
791      }
792
793      boolean selektiere(String s) {
794          Datum d = new Datum();
795          d.setzeDatum(s.substring(0,8));
796          return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
797      }
798
799      void erzeugeDatenvector(String s) {
800          if(selektiere(s) == true) {
801              Datum datum = new Datum(s.substring(0,8));
802              Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
803              Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
804              String wkn = s.substring(14,24);
805              String name = s.substring(25, s.length()-7);
806              Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
807              vec.addElement(tagesdaten);
808          }
809      }
810
811      boolean myaction(Event ev, Object target){
812          return true;
813      }
814
815      boolean schonAbgearbeitet(String s) {
816          int i = 0;
817          while(i < v.size()) {
818              if((String)v.elementAt(i)).compareTo(s)==0) {
819                  i = v.size();
820                  return true;
821              } else {
822                  i++;
823              }
824          }

```

```

825     v.addElement(s);
826     return false;
827 }
828
829 float tagesgewinn(String wertpapierkennnummer, int start) {
830     float min = 0.0f;
831     float max = 0.0f;
832     int i = start;
833     while((wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)  $\neq$  0) && (i <
vec.size())) {
834         i = i+1;
835     }
836     min = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
837     while(i < vec.size()) {
838         if(wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
839             max = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
840         }
841         i = i+1;
842     }
843     // System.out.println("Min = " + min + "      Max = " + max);
844     if(min == 0.0f) {
845         return 0;
846     } else {
847         return ((max - min)/min);
848     }
849 }
850
851 void zeigeTabelleAn() {
852     textarea = new TextArea(20,40);
853     p2.add("Center",textarea);
854     boolean keineDaten = true;
855     sortiere();
856     for(int i = 0; i < vec.size(); i++) {
857         if(! schonAbgearbeitet(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
858             textarea.append(((Tagesdaten)(vec.elementAt(i))).name + " " +
859                 ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + " "
860                 + tagesgewinn(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
861                 "\n");
862             keineDaten = false;
863         }
864     }
865     if(keineDaten == true) {
866         textarea.append("Fuer das heutige Datum, " + heute.tag + ". " +
867             heute.monat + ". " + heute.jahr +
868             ", existieren keine Daten\n");
869     }
870     this.pack();
871     this.show();
872 }
873 }
874
875 /* tabellarische Darstellung der letzten vorhandenen heutigen Kurse fuer alle
876 Aktientitel.
877 */
878
879 class LetzteTageskurse extends Tabelle {
880     TextArea textarea;
881     Vector v = new Vector();
882
883     LetzteTageskurse(String titel) {
884         super(titel);
885     }
886
887     void setzeDateiname(String s) {

```

```

888     dateiname = s;
889 }
890
891 void setzeDateiname() {
892     dateiname = kurse;
893 }
894
895 boolean selektiere(String s) {
896     Datum d = new Datum();
897     d.setzeDatum(s.substring(0,8));
898     return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
899 }
900
901 void erzeugeDatenvector(String s) {
902     if(selektiere(s) == true) {
903         Datum datum = new Datum(s.substring(0,8));
904         Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
905         Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
906         String wkn = s.substring(14,24);
907         String name = s.substring(25, s.length()-7);
908         Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
909         vec.addElement(tagesdaten);
910     }
911 }
912
913 boolean myaction(Event ev, Object target){
914     return true;
915 }
916
917 boolean schonAbgearbeitet(String s) {
918     int i = 0;
919     while(i < v.size()) {
920         if(((String)v.elementAt(i)).compareTo(s)==0) {
921             i = v.size();
922             return true;
923         } else {
924             i++;
925         }
926     }
927     v.addElement(s);
928     return false;
929 }
930
931 float letztetageskurse(String wertpapierkennnummer, int start) {
932     float letzter = 0.0f;
933     int i = start;
934     while(i < vec.size()) {
935         if(wertpapierkennnummer.compareTo(
936             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
937             letzter = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
938         }
939         i = i+1;
940     }
941     return letzter;
942 }
943
944 void zeigeTabelleAn() {
945     textarea = new TextArea(20,40);
946     p2.add("Center",textarea);
947     sortiere();
948     for(int i = 0; i < vec.size(); i++) {
949         if(!schonAbgearbeitet(
950             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
951             textarea.append(((Tagesdaten)(vec.elementAt(i))).name + " " +

```

```

952             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + " " +
953             letztetageskurse(
954             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
955             "\n");
956     }
957 }
958 if(vec.size() == 0) {
959     textarea.append("Fuer das heutige Datum, " +
960     heute.tag + "." + heute.monat + "." +
961     heute.jahr + ", existieren keine Daten\n");
962 }
963 this.pack();
964 this.show();
965 }
966 }
967
968
969 /* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
970 letzten 7 Tage. Benutzt die Kassakurse.
971 */
972
973 class WochenTabelle extends Tabelle {
974     Datum vonDatum;
975     TextArea textarea;
976
977     WochenTabelle(String titel) {
978         super(titel);
979         vonDatum = heute.wochezurueck();
980     }
981
982     void setzeDateiname(String s) {
983         dateiname = s;
984     }
985
986     void setzeDateiname() {
987         dateiname = kassakurse;
988     }
989
990     boolean selektiere(String s) {
991         Datum d = new Datum();
992         d.setzeDatum(s.substring(0,8));
993         return (!d.kleinergleich(vonDatum));
994     }
995
996     void erzeugeDatenvector(String s){
997         if(selektiere(s) == true) {
998             Datum datum = new Datum(s.substring(0,8));
999             String wkn = s.substring(9,18);
1000             String name = s.substring(19, s.length()-7);
1001             Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
1002             Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
1003             vec.addElement(zrd);
1004         }
1005     }
1006
1007     boolean myaction(Event ev, Object target) {
1008         return true;
1009     }
1010
1011     void zeigeTabelleAn(){
1012         textarea = new TextArea(20,40);
1013         p2.add("Center",textarea);
1014         sortiere();
1015         for(int i = 0; i < vec.size(); i++) {

```

```

1016     textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
1017                     (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
1018                     (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
1019                     ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
1020                     ((Zeitraumdaten)vec.elementAt(i)).name + " " +
1021                     ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
1022     }
1023     if(vec.size() == 0) {
1024         textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
1025     }
1026     this.pack();
1027     this.show();
1028 }
1029 }
1030
1031
1032 /* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
1033 letzten 30 Tage (genauer gesagt: seit dem gleichen Tag im Vormonat).
1034 Benutzt die Kassakurse.
1035 */
1036
1037 class MonatsTabelle extends Tabelle {
1038     Datum vonDatum;
1039     TextArea textarea;
1040
1041     MonatsTabelle(String titel) {
1042         super(titel);
1043         vonDatum = heute.monatzurueck();
1044     }
1045
1046     void setzeDateiname(String s) {
1047         dateiname = s;
1048     }
1049
1050     void setzeDateiname() {
1051         dateiname = kassakurse;
1052     }
1053
1054     boolean selektiere(String s) {
1055         Datum d = new Datum();
1056         d.setzeDatum(s.substring(0,8));
1057         return (!d.kleingleich(vonDatum));
1058     }
1059
1060     void erzeugeDatenvector(String s){
1061         if(selektiere(s) == true) {
1062             Datum datum = new Datum(s.substring(0,8));
1063             String wkn = s.substring(9,18);
1064             String name = s.substring(19, s.length()-7);
1065             Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
1066             Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
1067             vec.addElement(zrd);
1068         }
1069     }
1070
1071     boolean myaction(Event ev, Object target) {
1072         return true;
1073     }
1074
1075     void zeigeTabelleAn(){
1076         textarea = new TextArea(20,40);
1077         p2.add("Center",textarea);
1078         sortiere();
1079         for(int i = 0; i < vec.size(); i++) {

```

```

1080     textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
1081                     (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
1082                     (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
1083                     ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
1084                     ((Zeitraumdaten)vec.elementAt(i)).name + " " +
1085                     ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
1086     }
1087     if(vec.size() == 0) {
1088         textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
1089     }
1090     this.pack();
1091     this.show();
1092 }
1093 }
1094
1095
1096 /* tabellarische Darstellung der prozentualen Unterschiede zwischen
1097 aufeinanderfolgenden Kassakursen aller Aktientitel fuer den
1098 letzten Monat. Benutzt die Kassakurse.
1099 */
1100
1101 class MonatsDynamikTabelle extends Tabelle {
1102     Vector dyn = new Vector();
1103     Datum vonDatum;
1104     TextArea textarea;
1105
1106
1107     MonatsDynamikTabelle(String titel) {
1108         super(titel);
1109         vonDatum = heute.monatzurueck();
1110     }
1111
1112     void setzeDateiname(String s) {
1113         dateiname = s;
1114     }
1115
1116     void setzeDateiname() {
1117         dateiname = kassakurse;
1118     }
1119
1120     boolean selektiere(String s) {
1121         Datum d = new Datum();
1122         d.setzeDatum(s.substring(0,8));
1123         return (!d.kleinergleich(vonDatum));
1124     }
1125
1126     void erzeugeDatenvector(String s){
1127         if(selektiere(s) == true) {
1128             Datum datum = new Datum(s.substring(0,8));
1129             String wkn = s.substring(9,18);
1130             String name = s.substring(19, s.length()-7);
1131             Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
1132             Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
1133             vec.addElement(zrd);
1134         }
1135     }
1136
1137     boolean myaction(Event ev, Object target) {
1138         return true;
1139     }
1140
1141     float berechneDynamik(int i) {
1142         //suche den letzten Wert des gleichen Wertpapiers, um die prozentuale
1143         //Abweichung des neuen Wertes vom alten zu bestimmen.

```



```

1144     int j = i-1;
1145     boolean gefunden = false;
1146     while ((j ≥ 0) && !gefunden) {
1147         if (((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer).compareTo(
1148             ((Zeitraumdaten)vec.elementAt(j)).wertpapierkennnummer) == 0) {
1149             gefunden = true;
1150             return (((((Zeitraumdaten)vec.elementAt(i)).kassakurs).kurs -
1151                 ((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)/
1152                 (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)*100);
1153         }
1154         j--;
1155     }
1156     return 0.0f;
1157 }
1158
1159 void zeigeTabelleAn(){
1160     textarea = new TextArea(20,40);
1161     p2.add("Center",textarea);
1162     sortiere();
1163     for(int i = 0; i < vec.size(); i++) {
1164         textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
1165             (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
1166             (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
1167             ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
1168             ((Zeitraumdaten)vec.elementAt(i)).name + " " +
1169             berechneDynamik(i) + "%\n");
1170     }
1171     if(vec.size() == 0) {
1172         textarea.append("Fuer diesen Zeitraum existieren keine Daten\n");
1173     }
1174     this.pack();
1175     this.show();
1176 }
1177 }
1178
1179
1180 /* Schnittstelle fuer 'vergleichbare' Datentypen (zum Sortieren).
1181    a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
1182 */
1183
1184 interface Comparable {
1185     public boolean vergleiche(Comparable c);
1186 }
1187
1188 /* Oberklasse aller graphischen Anzeigeklassen
1189 */
1190
1191 abstract class ChartAnzeiger extends Canvas {
1192     Vector v;
1193     int xSize;
1194     int ySize;
1195     int abstand;
1196
1197     ChartAnzeiger(Vector vec, int xSize, int ySize) {
1198         v = vec;
1199         this.xSize = xSize;
1200         this.ySize = ySize;
1201         if(vec.size() ≠ 0) {
1202             abstand = xSize/vec.size();
1203         }
1204         setSize(xSize, ySize);
1205     }
1206
1207     public void paint(Graphics g) {

```

```

1208     g.drawRect(0, 0, xSize-1, ySize-1);
1209     for(int i = 0; i < v.size()-1; i++) {
1210         drawLine(g, i);
1211     }
1212 }
1213
1214 abstract void drawLine(Graphics g, int i);
1215 }
1216
1217
1218 /* graphische Darstellung aller Kurse eines Aktientitels fuer den heutigen
1219 Tag.
1220 */
1221
1222 class TagesChartAnzeiger extends ChartAnzeiger {
1223
1224     TagesChartAnzeiger(Vector vec, int xSize, int ySize) {
1225         super(vec, xSize, ySize);
1226     }
1227
1228     void drawLine(Graphics g, int i){
1229         g.drawLine(i*abstand,
1230                 ySize - (int)((((TagesChartWert)(v.elementAt(i))).kurs).kurs),
1231                 (i+1)*abstand,
1232                 ySize - (int)((((TagesChartWert)(v.elementAt(i+1))).kurs).kurs));
1233     }
1234 }
1235
1236
1237 /* graphische Darstellung aller Kassakurse eines Aktientitels fuer die
1238 letzten 30 Tage.
1239 */
1240
1241 class MonatsChartAnzeiger extends ChartAnzeiger {
1242
1243     MonatsChartAnzeiger(Vector vec, int xSize, int ySize) {
1244         super(vec, xSize, ySize);
1245     }
1246
1247     void drawLine(Graphics g, int i){
1248         g.drawLine(i*abstand,
1249                 ySize - (int)((((MonatsChartWert)(v.elementAt(i))).kassakurs).kurs),
1250                 (i+1)*abstand,
1251                 ySize - (int)((((MonatsChartWert)(v.elementAt(i+1))).kassakurs).kurs));
1252     }
1253 }
1254
1255 /* Kern des Hauptprogramms: Auswahlmenue der einzelnen Darstellungsformen.
1256 */
1257
1258 class Auswahl extends Frame {
1259     Button beenden = new Button("Beenden");
1260     Button ok = new Button("OK");
1261     Panel p = new Panel();
1262     CheckboxGroup cbg = new CheckboxGroup();
1263     Checkbox cbtt = new Checkbox("Tagestabelle", cbg, true);
1264     Checkbox cbwt = new Checkbox("Wochentabelle", cbg, false);
1265     Checkbox cbmt = new Checkbox("Monatstabelle", cbg, false);
1266     Checkbox cbdt = new Checkbox("Dynamiktabelle", cbg, false);
1267     Checkbox cbtg = new Checkbox("Tagesgewinne", cbg, false);
1268     Checkbox cbltk = new Checkbox("letzte Tageskurse", cbg, false);
1269     Checkbox cbtc = new Checkbox("Tages-Chart", cbg, false);
1270     Checkbox cbmc = new Checkbox("Monats-Chart", cbg, false);
1271

```

```

1272 Auswahl(String titel) {
1273     super(titel);
1274     setLayout(new BorderLayout());
1275     p.setLayout(new GridLayout(6,1));
1276     p.add(cbtt);
1277     p.add(cbwt);
1278     p.add(cbmt);
1279     p.add(cbdt);
1280     p.add(cbtg);
1281     p.add(cbltk);
1282     p.add(cbtc);
1283     p.add(cbmc);
1284     add("Center",p);
1285     add("East",ok);
1286     add("South",beenden);
1287     pack();
1288     show();
1289 }
1290
1291 public boolean action(Event ev, Object target){
1292     if(ev.id == Event.ACTION_EVENT) {
1293         if(ev.target == ok) {
1294             if(cbg.getSelectedCheckbox() == cbtt) {
1295                 TagesTabelle tt = new TagesTabelle("Tagestabelle");
1296                 tt.stelleDar();
1297                 return true;
1298             } else if(cbg.getSelectedCheckbox() == cbwt) {
1299                 WochenTabelle wt = new WochenTabelle("Wochentabelle");
1300                 wt.stelleDar();
1301                 return true;
1302             } else if(cbg.getSelectedCheckbox() == cbmt) {
1303                 MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
1304                 mt.stelleDar();
1305                 return true;
1306             } else if(cbg.getSelectedCheckbox() == cbdt) {
1307                 MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
1308                 dt.stelleDar();
1309                 return true;
1310             } else if(cbg.getSelectedCheckbox() == cbtg) {
1311                 Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
1312                 tg.stelleDar();
1313                 return true;
1314             } else if(cbg.getSelectedCheckbox() == cbltk) {
1315                 LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
1316                 ltg.stelleDar();
1317                 return true;
1318             } else if(cbg.getSelectedCheckbox() == cbtc) {
1319                 TagesChart tc = new TagesChart("Tageschart");
1320                 tc.stelleDar();
1321                 return true;
1322             } else if(cbg.getSelectedCheckbox() == cbmc) {
1323                 MonatsChart mc = new MonatsChart("Monatschart");
1324                 mc.stelleDar();
1325                 return true;
1326             }
1327         } else if(ev.target == beenden) {
1328             System.exit(0);
1329             return true;
1330         }
1331     }
1332     return false;
1333 }
1334 }
1335

```

```
1336
1337 /* Programmeinstiegspunkt
1338 */
1339
1340 class Boerse {
1341     public static void main(String args[]) {
1342         Auswahl a = new Auswahl("Boersen-Information");
1343     }
1344 }
1345
1346
```

B.1.3 Boerse.java (5 levels of inheritance)

```

1  /*****/
2  /*Programmautor: Barbara Unger */
3  /*Datum: 1997-06-03 */
4  /*Gruppe: Boerse5 */
5  /*****/
6
7  import java.awt.*;
8  import java.util.Vector;
9  import java.util.Enumeration;
10 import java.util.Date;
11 import java.io.*;
12
13 /* Programm zum Betrachten von Boersenkursen.
14 Abfolge der Klassen in dieser Datei:
15
16 interface Konstanten
17 abstract class BoersenInfo extends Frame implements Konstanten
18 abstract class KostenloseInfos extends BoersenInfo
19 abstract class KostenpflichtigeInfos extends BoersenInfo
20 abstract class Chart extends KostenloseInfos
21 class TagesChart extends Chart
22 class TagesChartWert implements Comparable
23 class MonatsChart extends Chart
24 class MonatsChartWert implements Comparable
25 abstract class Tabelle extends KostenloseInfos
26 class TagesTabelle extends Tabelle
27 class Tagesdaten implements Comparable
28 class Zeitraumdaten implements Comparable
29 class Datum
30 class Uhrzeit
31 class Kurs
32 abstract class ZeitraumTabelle extends Tabelle
33 class Tagesgewinne extends LetzteTageskurse
34 class LetzteTageskurse extends TagesTabelle
35 class WochenTabelle extends ZeitraumTabelle
36 class MonatsTabelle extends ZeitraumTabelle
37 class MonatsDynamikTabelle extends MonatsTabelle
38 interface Comparable
39 abstract class ChartAnzeiger extends Canvas
40 class TagesChartAnzeiger extends ChartAnzeiger
41 class MonatsChartAnzeiger extends ChartAnzeiger
42 class Auswahl extends Frame
43 class Boerse
44 */
45
46 interface Konstanten {
47 // static Date date = new Date();
48 // static Datum heute = new Datum(date.getDay(), date.getMonth(), date.getYear());
49 static Datum heute = new Datum(27, 5, 97);
50 static String kurse = "KURSE";
51 static String kassakurse = "KASSAKURSE";
52 }
53
54
55 /* abstrakte Klasse BoersenInfo Implementiert Fenster, in denen
56 Boersen-Informationen angezeigt werden. Stellt sicher, dass in

```

```

57  jeder der daraus abgeleiteten Klassen Knöpfe zum Schliessen des
58  Fensters und zum Beenden des Programmes existieren.
59
60  Methoden:
61  BoersenInfo(String) erzeugt das Fenster und setzt den Titel
62  schliesseFenster() kehrt zum aufrufenden Fenster zurueck
63  beendeProgramm() beendet das Programm komplett
64  action(Event, Object) verarbeitet Interaktionsereignisse
65  abstract myaction(Event, Object)
66  wird in "action" aufgerufen; Unterklassen
67  koennen hier durch ueberschreiben
68  spezielle Ereignisbehandlung
69  implementieren
70  */
71
72  abstract class BoersenInfo extends Frame implements Konstanten{
73      Panel p1 = new Panel();
74      Panel p2 = new Panel();
75      Panel p3 = new Panel();
76      Button schliessen = new Button("Schliessen");
77      Button beenden = new Button("Beenden");
78      float kosten;
79
80  /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und bestimmt das Layout.
81  Panel p1 dient zum Darstellen von Sponsor- oder Kosteninformationen,
82  Panel p2 dient zum Anzeigen der angeforderten Informationen,
83  Panel p3 enthaelt Knöpfe zum Schliessen des Fensters und zum Beenden des Programms.
84  */
85  BoersenInfo(String titel) {
86      super(titel);
87      this.setLayout(new BorderLayout());
88      p1.setLayout(new FlowLayout());
89      add("North",p1);
90      p2.setLayout(new BorderLayout());
91      add("Center",p2);
92      p3.setLayout(new FlowLayout());
93      add("South",p3);
94      p3.add(schliessen);
95      p3.add(beenden);
96  }
97
98  /* Schliesst das Fenster und gibt seine Ressourcen frei.
99  */
100 void schliesseFenster() {
101     this.dispose();
102 }
103
104 /* Beendet das Programm komplett: alle Fenster werden geschlossen.
105 */
106 void beendeProgramm() {
107     System.exit(0);
108 }
109
110 /* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
111 Knöpfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
112 anderen Ereignisse wird "myaction" aufgerufen.
113 */
114 public boolean action(Event event, Object target) {
115     if(event.id == Event.ACTION_EVENT) {
116         if(event.target == schliessen) {
117             schliesseFenster();
118             return true;
119         }
120     else if(event.target == beenden) {

```

```

121     beendeProgramm();
122     return true;
123 } else {
124     return myaction(event, target);
125 }
126 } else {
127     return false;
128 }
129 }
130
131 /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
132 werden, um eine spezielle Ereignisbehandlung zu
133 implementieren. (Diese Konstruktion verhindert, dass die Behandlung
134 der Knoepfe "Schliessen" und "Beenden" jedesmal neu implementiert
135 werden muss.
136 */
137 abstract boolean myaction(Event ev, Object target);
138 }
139
140
141 /* abstrakte Klasse KostenloseInfos
142 Stellt sicher, dass fuer das Abrufen von Informationen durch alle
143 aus ihr abgeleiteten Klassen keine Kosten anfallen. (Kosten werden
144 von einem Sponsor getragen).
145 Des weiteren stellt sie eine Methode zum Dateneinlesen
146 bereit. Welche Daten jedoch daraus erzeugt werden ist zu diesem
147 Zeitpunkt unklar und wird erst in einer Unterklasse entschieden
148 durch die Methode erzeugeDatenvector(). Ausserdem wird nun schon
149 ein Sponsorlabel fuer alle kostenlosen Infos gesetzt.
150
151 Methoden:
152 KostenloseInfos(String) Konstruktor, setzt Fenstertitel, fuegt einen Sponsor ein
153 und setzt die Kosten auf 0.00.
154 leseDaten(String) liest Daten aus einer Datei und schreibt sie mit Hilfe
155 der Methode erzeugeDatenvector() in den Vector vec
156 qsort(int, int), partition(int, int), exchange(int, int), sortiere()
157 sind Methoden, die einen Sortieralgorithmus fuer alle
158 Unterklassen zur Verfuegung stellen. Da in jeder
159 Unterklasse unterschiedliche Datentypen zu sortieren
160 sind, wurde Schablonenmethode angewandt, die
161 vergleiche()-Methode ist nur abstrakt realisiert
162 und wird vom jeweiligen Datentyp erst implementiert.
163 */
164
165 abstract class KostenloseInfos extends BoersenInfo {
166     String dateiname;
167     Label sponsor;
168     Vector vec = new Vector();
169
170     /* Konstruktor, setzt den Titel des Fensters, fuegt einen Sponsor in das
171 Fenster ein (in p1) und setzt die Kosten auf 0.00
172 */
173     KostenloseInfos(String titel) {
174         super(titel);
175         sponsor = new Label("Diese Seiten werden gesponsort von xxy");
176         p1.add(sponsor);
177         kosten = 0.0f;
178     }
179
180     /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
181 abstrakte Methode erzeugeDatenvector eine Datenstruktur, in der die
182 relevanten Daten gespeichert werden.
183 */
184     void leseDaten(String dateiname) {

```

```

185     try{
186         BufferedReader br = new BufferedReader(new FileReader(dateiname));
187         String s;
188         try{
189             while((s = br.readLine())!=null) {
190                 if(!s.startsWith("#")) {
191                     erzeugeDatenvector(s);
192                 }
193             }
194         } catch (IOException e) {
195             System.err.println("Lesefehler bei Datei " + dateiname);
196         }
197     } catch (FileNotFoundException e) {
198         System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
199     }
200 }
201
202 /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
203 gespeichert sind. Dieses Objekt wird im Vektor vec gespeichert
204 */
205 abstract void erzeugeDatenvector(String s);
206
207 /* Beschreibung des Quicksort siehe oben
208 */
209 void qsort(int left, int right) {
210     int middle;
211     if(left < right){
212         middle = partition(left, right);
213         qsort(left,middle-1);
214         qsort(middle+1, right);
215     }
216 }
217
218 /* Hilfsprozedur fuer den Quicksort
219 */
220 int partition(int left, int right) {
221     Comparable pivot = (Comparable)vec.elementAt(left);
222     int l = left;
223     int r = right;
224     int middle;
225     while(l < r) {
226         while((l ≤ right) && (((Comparable)vec.elementAt(l)).vergleiche(pivot))){
227             l = l+1;
228         }
229         while((r ≥ left) && (!((Comparable)vec.elementAt(r)).vergleiche(pivot))){
230             r = r-1;
231         }
232         if(l < r) {
233             exchange(l, r);
234         }
235     }
236     middle = r;
237     exchange (left, middle);
238     return middle;
239 }
240
241 /* Hilfsprozedur fuer den Quicksort
242 */
243 void exchange(int i, int j) {
244     Comparable c = (Comparable)vec.elementAt(i);
245     vec.setElementAt(vec.elementAt(j),i);
246     vec.setElementAt(c,j);
247 }
248

```



```

249  /* Ruft Quicksort auf
250  */
251  void sortiere() {
252      qsort(0, vec.size()-1);
253  }
254 }
255
256
257 /* Klasse KostenpflichtigeInfos
258   enthaelt einen Rahmen fuer Informationen, die vom Kunden bezahlt werden
259   muessen, wie beispielsweise Prognosen, etc.
260 */
261 /*Wird nicht ausprogrammiert.*/
262 abstract class KostenpflichtigeInfos extends BoersenInfo {
263
264     KostenpflichtigeInfos(String titel) {
265         super(titel);
266         kosten = 1.0f;
267     }
268 }
269
270
271 /* abstrakte Klasse Chart
272   Diese Klasse enthaelt allgemeine Methoden zur Darstellung eines Charts.
273   Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
274   werden soll, und nach der Auswahl sollte der Chart angezeigt werden.
275
276   Methoden:
277   Chart(String)      Konstruktor, setzt den Fenstertitel
278   void stelleDar()   stellt eine Dialogbox dar.
279   myaction()        erweert die action-Methode der Oberklasse.
280   abstract zeigeCanvasAn() liefert ein Canvas zurueck.
281 */
282
283 abstract class Chart extends KostenloseInfos {
284     int xChartSize = 200;
285     int yChartSize = 200;
286     String wkn;
287     Label l = new Label();
288     TextField t = new TextField("700045634",9);
289     Button ok = new Button("ok");
290
291     /* Konstruktor, ruft den Konstruktor von KostenloseInfos auf, und fuegt in den
292     freien Rahmen p2 eine Auswahlmoeglichkeit des gewuenschten Wertpapiers ein
293     */
294     Chart(String titel) {
295         super(titel);
296     }
297
298     void stelleDar(){
299         p2.add("North",l);
300         p2.add("Center",t);
301         p2.add("South",ok);
302         pack();
303         show();
304     }
305
306
307     /* myaction erweert die action-Methode, so dass nun beim Druck des
308     Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
309     aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
310     und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
311     */
312     boolean myaction(Event ev, Object target){

```

```

313     if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
314         wkn = t.getText();
315         wkn.trim();
316         p2.removeAll();
317         leseDaten(dateiname);
318         p2.add("Center", zeigeCanvasAn());
319         pack();
320         show();
321         return true;
322     }
323     return false;
324 }
325
326 /* liefert ein Canvas zurueck. Was das Canvas enthaelt, wird erst in der Unterklasse
327    bestimmt.
328 */
329 abstract Canvas zeigeCanvasAn();
330 }
331
332
333 /* Klasse TagesChart
334    In dieser Klasse wird das Erzeugen des Datenvektors implementiert.
335    Diese entscheidet durch eine selektiere()-Methode, welche
336    Datensatze ueberhaupt ausgewaehlt werden sollen.
337
338    Methoden:
339    TagesChart(String)      Konstruktor, der den Titel setzt und die Datei
340                           mit den Daten angibt.
341    erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
342                           entscheidet mit Hilfe von selektiere(), ob der
343                           Datensatz in den Datenvektor eingefuegt
344                           werden soll.
345    selektiere(String s)    entscheidet, ob der String ausgewaehlt werden
346                           soll, konkret: ob er von dem angegebenen
347                           Datum ist und die entsprechende
348                           Wertpapierkennnummer hat
349    zeigeCanvasAn()        erzeugt ein Objekt der Canvas-Klasse, in der der
350                           Chart aus den selektierten Daten, die im Datenvektor stehen,
351                           angezeigt wird.
352 */
353
354 class TagesChart extends Chart {
355     TagesChartWert tcw;
356
357     /* Konstruktor, der den Titel setzt und die Datei mit den Daten angibt
358     */
359     TagesChart(String titel) {
360         super(titel);
361         dateiname = kurse;
362         l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Tageschart an");
363     }
364
365     /* bekommt eine Datensatz in Stringform und entscheidet mit Hilfe von selektiere(),
366        ob der Datensatz in den Datenvektor eingefuegt werden soll.
367     */
368     void erzeugeDatenvector(String s){
369         if(selektiere(s) == true) {
370             tcw = new TagesChartWert(s);
371             vec.addElement(tcw);
372         }
373     }
374
375     /* entscheidet, ob der String ausgewaehlt werden soll, konkret: ob er von dem angegebenen
376        Datum ist und die entsprechende Wertpapierkennnummer hat

```

```

377  */
378  boolean selektiere(String s) {
379      Datum d = new Datum();
380      d.setzeDatum(s.substring(0,8));
381      if((heute.tag == d.tag) && (heute.monat ==d.monat) && (heute.jahr == d.jahr)) {
382          return (wkn.compareTo(s.substring(15,24))==0);
383      }
384      return false;
385  }
386
387  /* erzeugt ein Objekt der Canvas-Klasse, in der der Chart aus den selektierten Daten, die im
388     Datenvektor stehen, angezeigt wird.
389  */
390  Canvas zeigeCanvasAn() {
391      return new TagesChartAnzeiger(vec, xChartSize, yChartSize);
392  }
393 }
394
395
396
397  /*Klasse TagesChartWert
398     ist ein Objekttyp, der aus einer Uhrzeit und einem Kurswert besteht.
399     Das sind genau die Daten, die man zum Anzeigen eines Tagescharts braucht. Die Klasse hat
400     eine vergleiche-Methode, so dass man einen Vector mit Objekten dieses Typs mit
401     der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.
402
403     Methoden:
404     TagesChartWert(String) bekommt einen String, in dem die Uhrzeit
405                             und der Kurs enthalten sind.
406     vergleiche(Comparable c) vergleicht die Uhrzeiten
407  */
408
409  class TagesChartWert implements Comparable {
410      Uhrzeit uhrzeit;
411      Kurs kurs;
412
413      /* bekommt einen String, in dem die Uhrzeit und der Kurs enthalten sind.
414      */
415      TagesChartWert(String s) {
416          uhrzeit = new Uhrzeit(s.substring(9,15));
417          kurs = new Kurs(s.substring(s.length()-7, s.length()));
418      }
419
420      /* vergleicht die Uhrzeiten zweier TagesChartWert-Objekte
421      */
422      public boolean vergleiche(Comparable c) {
423          return ((this.uhrzeit).kleinergleich(((TagesChartWert)c).uhrzeit));
424      }
425  }
426
427
428  /* Klasse MonatsChart
429     In dieser Klasse wird das Erzeugen des Datenvektors implementiert.
430     Diese entscheidet durch eine selektiere()-Methode, welche
431     Datensätze ueberhaupt ausgewaehlt werden sollen.
432
433     Methoden:
434     MonatsChart(String)      Konstruktor, der den Titel setzt und die Datei
435                             mit den Daten angibt.
436     erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
437                             entscheidet mit Hilfe von slektiere(), ob der
438                             Datensatz in den Datenvector eingefuegt
439                             werden soll.
440     selektiere(String s)     entscheidet, ob der String ausgewaehlt werden

```

```

441                                     soll, konkret: ob er von dem angegebenen
442                                     Datum ist und die entsprechende
443                                     Wertpapierkennnummer hat
444         zeigeCanvasAn()                erzeugt ein Objekt der Canvas-Klasse, in der der
445                                     Chart angezeigt wird.
446     */
447
448     class MonatsChart extends Chart {
449         MonatsChartWert mcw;
450         Datum letzterMonat = new Datum();
451
452         MonatsChart(String titel) {
453             super(titel);
454             l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Monatschart an");
455             dateiname = kassakurse;
456             letzterMonat = heute.monatzurueck();
457         }
458
459         void erzeugeDatenvector(String s){
460             if(selektiere(s) == true) {
461                 mcw = new MonatsChartWert(s);
462                 vec.addElement(mcw);
463             }
464         }
465
466         boolean selektiere(String s) {
467             Datum d = new Datum();
468             d.setzeDatum(s.substring(0,8));
469             if(!d.kleinergleich(letzterMonat)) {
470                 return (wkn.compareTo(s.substring(9,18))==0);
471             }
472             return false;
473         }
474
475         Canvas zeigeCanvasAn() {
476             return new MonatsChartAnzeiger(vec, xChartSize, yChartSize);
477         }
478     }
479
480
481     /*Klasse MonatsChartWert
482     ist ein Objekttyp, der aus einem Datum und einem Kassakurswert besteht,
483     Daten, die man zum Anzeigen eines Monatschart braucht. Die Klasse hat
484     eine vergleiche-Methode, so dass man einen Vector mit diesem Typ mit
485     der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.
486
487     Methoden:
488     MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
489     und der Kurs enthalten sind.
490     vergleiche(Comparable c) vergleicht das Datum
491     */
492
493     class MonatsChartWert implements Comparable {
494         Kurs kassakurs;
495         Datum datum;
496
497         MonatsChartWert(String s){
498             kassakurs = new Kurs(s.substring(s.length()-6, s.length()));
499             datum = new Datum(s.substring(0,9));
500         }
501
502         public boolean vergleiche(Comparable c){
503             return (((this.datum).kleinergleich(((MonatsChartWert)c).datum));
504         }

```

```

505 }
506
507
508 /* abstrakte Klasse Tabelle.
509 Gemeinsame Oberklasse fuer alle tabellarischen Darstellungen von
510 Kursen oder Kassakursen.
511 */
512
513 abstract class Tabelle extends KostenloseInfos {
514
515     Tabelle(String titel) {
516         super(titel);
517         setzeDateiname();
518     }
519
520 /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
521 zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
522 noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
523 "stelleDar" aufgerufen.
524 */
525     void stelleDar(){
526         leseDaten(dateiname);
527         zeigeTabelleAn();
528     }
529
530     abstract void setzeDateiname();
531     abstract void zeigeTabelleAn();
532 }
533
534 /* Klasse zur tabellarischen Darstellung aller Kurse eines Tages
535 */
536
537 class TagesTabelle extends Tabelle {
538     TextArea textarea;
539
540     TagesTabelle(String titel) {
541         super(titel);
542     }
543
544     void setzeDateiname(String s) {
545         dateiname = s;
546     }
547
548     void setzeDateiname() {
549         dateiname = kurse;
550     }
551
552     boolean myaction(Event ev, Object target){
553         return true;
554     }
555
556     boolean selektiere(String s) {
557         Datum d = new Datum();
558         d.setzeDatum(s.substring(0,8));
559         return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
560     }
561
562
563     void erzeugeDatenvector(String s) {
564         if(selektiere(s) == true) {
565             Datum datum = new Datum(s.substring(0,8));
566             Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
567             Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
568             String wkn = s.substring(14,24);

```

```

569     String name = s.substring(25, s.length()-7);
570     Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
571     vec.addElement(tagesdaten);
572 }
573 }
574
575 void zeigeTabelleAn(){
576     textarea = new TextArea(20,40);
577     p2.add("Center",textarea);
578     sortiere();
579     for(int i = 0; i < vec.size(); i++) {
580         textarea.append((((Tagesdaten)vec.elementAt(i)).uhrzeit.stunde + " : " +
581             (((Tagesdaten)vec.elementAt(i)).uhrzeit.minute + " " +
582             ((Tagesdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
583             ((Tagesdaten)vec.elementAt(i)).name + " " +
584             ((Tagesdaten)vec.elementAt(i)).kurs.kurs + "\n"));
585     }
586     if(vec.size() == 0) {
587         textarea.append("Fuer das heutige Datum, " + heute.tag + ". " +
588             heute.monat + ". " + heute.jahr +
589             ", existieren keine Daten\n");
590     }
591     this.pack();
592     this.show();
593 }
594 }
595
596
597 /* Klasse zur Repraesentation eines einzelnen Kursinformations-Datensatzes
598 */
599
600 class Tagesdaten implements Comparable {
601     Datum datum;
602     Uhrzeit uhrzeit;
603     String wertpapierkennnummer;
604     String name;
605     Kurs kurs;
606
607     Tagesdaten(Datum d, Uhrzeit u, String wkn, String n, Kurs k) {
608         datum = d;
609         uhrzeit = u;
610         wertpapierkennnummer = wkn;
611         name = n;
612         kurs = k;
613     }
614
615     public boolean vergleiche(Comparable c) {
616         return (this.uhrzeit).kleinergleich(((Tagesdaten)c).uhrzeit);
617     }
618 }
619
620
621 /* Klasse zur Repraesentation eines einzelnen Kassakursinformations-Datensatzes
622 */
623
624 class Zeitraumdaten implements Comparable {
625     Datum datum;
626     String wertpapierkennnummer;
627     String name;
628     Kurs kassakurs;
629
630     Zeitraumdaten(Datum d, String wkn, String n, Kurs k) {
631         datum = d;
632         wertpapierkennnummer = wkn;

```

```

633     name = n;
634     kassakurs = k;
635 }
636
637 public boolean vergleiche(Comparable c) {
638     return (this.datum).kleinergleich(((Zeitraumdaten)c).datum);
639 }
640 }
641
642
643 /* Klasse zur Repraesentation eines Datums, incl. Berechnung des
644    Datums 'vor einer Woche' und 'vor einem Monat'.
645    (Anmerkung zum Experiment:
646     Diese Routinen sind vereinfacht, um den Code zu verkuerzen.
647     Sie funktionieren nicht wirklich korrekt. Das soll uns hier aber mal
648     gerade nicht stoeren...)
649 */
650
651 class Datum {
652     int tag;
653     int monat;
654     int jahr;
655
656     Datum() {
657         tag = 0;
658         monat = 0;
659         jahr = 0;
660     }
661
662     Datum(String s){
663         tag = new Integer(s.substring(0,2)).intValue();
664         monat = new Integer(s.substring(3,5)).intValue();
665         jahr = new Integer(s.substring(6,8)).intValue();
666     }
667
668     Datum(int t, int m, int j){
669         tag = t;
670         monat = m;
671         jahr = j;
672     }
673
674     void setzeDatum(String s) {
675         tag = new Integer(s.substring(0,2)).intValue();
676         monat = new Integer(s.substring(3,5)).intValue();
677         jahr = new Integer(s.substring(6,8)).intValue();
678     }
679
680     void setzeDatum(int t, int m, int j) {
681         tag = t;
682         monat = m;
683         jahr = j;
684     }
685
686     boolean kleinergleich(Datum d) {
687         if(this.jahr < d.jahr) {
688             return true;
689         } else if(this.jahr == d.jahr){
690             if(this.monat < d.monat) {
691                 return true;
692             } else if(this.monat == d.monat){
693                 if(this.tag <= d.tag) {
694                     return true;
695                 }
696             }

```

```

697     }
698     return false;
699 }
700
701 Datum monatzurueck() {
702     Datum d = new Datum();
703     d.tag = tag;
704     d.jahr = jahr;
705     if(monat == 1) {
706         d.monat = 12;
707         d.jahr = jahr - 1;
708     } else {
709         d.monat = monat - 1;
710     }
711     return d;
712 }
713
714 /* vereinfachte Berechnung: Alle Monate haben 30 Tage
715 */
716 Datum wochezurueck() {
717     Datum d = new Datum();
718     d.tag = tag - 7;
719     d.monat = monat;
720     d.jahr = jahr;
721     if(d.tag < 1) {
722         d.tag = d.tag + 30;
723         d.monat = monat - 1;
724         if(d.monat < 1) {
725             d.monat = d.monat + 12;
726             d.jahr = jahr - 1;
727         }
728     }
729     return d;
730 }
731
732 void print() {
733     System.out.println(" Datum = " + tag + " . " + monat + " . " + jahr);
734 }
735 }
736
737 /* Klasse zur Repraesentation einer Uhrzeit
738 */
739
740 class Uhrzeit {
741     int stunde;
742     int minute;
743
744     Uhrzeit(String s) {
745         stunde = (new Integer(s.substring(0,2))).intValue();
746         minute = (new Integer(s.substring(3,5))).intValue();
747     }
748
749     boolean kleinergleich(Uhrzeit u) {
750         if(this.stunde < u.stunde) {
751             return true;
752         } else if(this.stunde == u.stunde) {
753             if(this.minute ≤ u.minute) {
754                 return true;
755             }
756         }
757         return false;
758     }
759 }
760

```



```

761  /* Klasse zur Repraesentation eines Kurses oder Kassakurses
762  */
763
764  class Kurs {
765      float kurs;
766
767      Kurs(String s){
768          kurs = (new Float(s)).floatValue();
769      }
770 }
771
772
773 /* abstrakte Klasse ZeitraumTabelle.
774 Gemeinsame Oberklasse fuer alle tabellarischen Darstellungen, die
775 ueber mehrere Tage gehen. Benutzt die Kassakurse.
776 */
777
778 abstract class ZeitraumTabelle extends Tabelle {
779     Datum vonDatum;
780     TextArea textarea;
781
782     ZeitraumTabelle(String titel) {
783         super(titel);
784     }
785
786     void setzeDateiname(String s) {
787         dateiname = s;
788     }
789
790     void setzeDateiname() {
791         dateiname = kassakurse;
792     }
793
794     boolean selektiere(String s) {
795         Datum d = new Datum();
796         d.setzeDatum(s.substring(0,8));
797         return (!d.kleinergleich(vonDatum));
798     }
799
800     void erzeugeDatenvector(String s){
801         if(selektiere(s) == true) {
802             Datum datum = new Datum(s.substring(0,8));
803             String wkn = s.substring(9,18);
804             String name = s.substring(19, s.length()-7);
805             Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
806             Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
807             vec.addElement(zrd);
808         }
809     }
810
811     void zeigeTabelleAn(){
812         textarea = new TextArea(20,40);
813         p2.add("Center",textarea);
814         sortiere();
815         for(int i = 0; i < vec.size(); i++) {
816             textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
817                 (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
818                 (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
819                 ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
820                 ((Zeitraumdaten)vec.elementAt(i)).name + " " +
821                 ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
822         }
823         if(vec.size() == 0) {
824             textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");

```

```

825     }
826     this.pack();
827     this.show();
828     }
829 }
830
831
832 /* Tabellarische Darstellung der prozentualen Unterschiede zwischen dem
833 ersten und letzten Kurs jeder Aktie am heutigen Datum.
834 */
835
836 class Tagesgewinne extends LetzteTageskurse {
837
838     Tagesgewinne(String titel) {
839         super(titel);
840     }
841
842     float tagesgewinn(String wertpapierkennnummer, int start) {
843         float min = 0.0f;
844         float max = 0.0f;
845         int i = start;
846         while((wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)  $\neq$  0) && (i <
vec.size())) {
847             i = i+1;
848         }
849         min = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
850         while(i < vec.size()) {
851             if(wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
852                 max = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
853             }
854             i = i+1;
855         }
856         // System.out.println("Min = " + min + "      Max = " + max);
857         if(min == 0.0f) {
858             return 0;
859         } else {
860             return ((max - min)/min);
861         }
862     }
863
864     void zeigeTabelleAn() {
865         textarea = new TextArea(20,40);
866         p2.add("Center",textarea);
867         boolean keineDaten = true;
868         sortiere();
869         for(int i = 0; i < vec.size(); i++) {
870             if(! schonAbgearbeitet(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
871                 textarea.append(((Tagesdaten)(vec.elementAt(i))).name + " " +
872                                 ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + " "
873                                 + tagesgewinn(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
874                                 "\n");
875                 keineDaten = false;
876             }
877         }
878         if(keineDaten == true) {
879             textarea.append("Fuer das heutige Datum, " + heute.tag + ". " +
880                             heute.monat + ". " + heute.jahr +
881                             ", existieren keine Daten\n");
882         }
883         this.pack();
884         this.show();
885     }
886 }
887

```

```

888  /* tabellarische Darstellung der letzten vorhandenen heutigen Kurse fuer alle
889     Aktientitel.
890  */
891
892  class LetzteTageskurse extends TagesTabelle {
893      Vector v = new Vector();
894
895      LetzteTageskurse(String titel) {
896          super(titel);
897      }
898
899      boolean schonAbgearbeitet(String s) {
900          int i = 0;
901          while(i < v.size()) {
902              if(((String)v.elementAt(i)).compareTo(s)==0) {
903                  i = v.size();
904                  return true;
905              } else {
906                  i++;
907              }
908          }
909          v.addElement(s);
910          return false;
911      }
912
913      float letztetageskurse(String wertpapierkennnummer, int start) {
914          float letzter = 0.0f;
915          int i = start;
916          while(i < vec.size()) {
917              if(wertpapierkennnummer.compareTo(
918                  ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
919                  letzter = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
920              }
921              i = i+1;
922          }
923          return letzter;
924      }
925
926      void zeigeTabelleAn() {
927          textarea = new TextArea(20,40);
928          p2.add("Center",textarea);
929          sortiere();
930          for(int i = 0; i < vec.size(); i++) {
931              if(!schonAbgearbeitet(
932                  ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
933                  textarea.append(((Tagesdaten)(vec.elementAt(i))).name + " " +
934                      ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + " " +
935                      letztetageskurse(
936                          ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
937                      "\n");
938              }
939          }
940          if(vec.size() == 0) {
941              textarea.append("Fuer das heutige Datum, " +
942                  heute.tag + " ." + heute.monat + " ." +
943                  heute.jahr + ", existieren keine Daten\n");
944          }
945          this.pack();
946          this.show();
947      }
948  }
949
950
951  /* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die

```

```

952     letzten 7 Tage.
953 */
954
955 class WochenTabelle extends ZeitraumTabelle {
956
957     WochenTabelle(String titel) {
958         super(titel);
959         vonDatum = heute.wochezurueck();
960     }
961
962     boolean myaction(Event ev, Object target) {
963         return true;
964     }
965 }
966
967
968 /* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
969     letzten 30 Tage (genauer gesagt: seit dem gleichen Tag im Vormonat).
970 */
971
972 class MonatsTabelle extends ZeitraumTabelle {
973
974     MonatsTabelle(String titel) {
975         super(titel);
976         vonDatum = heute.monatzurueck();
977     }
978
979     boolean myaction(Event ev, Object target) {
980         return true;
981     }
982 }
983
984
985 /* tabellarische Darstellung der prozentualen Unterschiede zwischen
986     aufeinanderfolgenden Kassakursen aller Aktientitel fuer den
987     letzten Monat.
988 */
989
990 class MonatsDynamikTabelle extends MonatsTabelle {
991     Vector dyn = new Vector();
992
993     MonatsDynamikTabelle(String titel) {
994         super(titel);
995     }
996
997     float berechneDynamik(int i) {
998         //suche den letzten Wert des gleichen Wertpapiers, um die prozentuale
999         //Abweichung des neuen Wertes vom alten zu bestimmen.
1000        int j = i-1;
1001        boolean gefunden = false;
1002        while ((j ≥ 0) && !gefunden) {
1003            if (((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer).compareTo(
1004                ((Zeitraumdaten)vec.elementAt(j)).wertpapierkennnummer) == 0) {
1005                gefunden = true;
1006                return (((Zeitraumdaten)vec.elementAt(i)).kassakurs).kurs -
1007                    (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)/
1008                    (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs*100);
1009            }
1010            j--;
1011        }
1012        return 0.0f;
1013    }
1014
1015 void zeigeTabelleAn(){

```

```

1016     textarea = new TextArea(20,40);
1017     p2.add("Center",textarea);
1018     sortiere();
1019     for(int i = 0; i < vec.size(); i++) {
1020         textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + " . " +
1021             (((Zeitraumdaten)vec.elementAt(i)).datum).monat + " . " +
1022             (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + " " +
1023             ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + " " +
1024             ((Zeitraumdaten)vec.elementAt(i)).name + " " +
1025             berechneDynamik(i) + "%\n");
1026     }
1027     if(vec.size() == 0) {
1028         textarea.append("Fuer diesen Zeitraum existieren keine Daten\n");
1029     }
1030     this.pack();
1031     this.show();
1032 }
1033 }
1034
1035
1036 /* Schnittstelle fuer 'vergleichbare' Datentypen (zum Sortieren).
1037    a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
1038 */
1039
1040 interface Comparable {
1041     public boolean vergleiche(Comparable c);
1042 }
1043
1044 /* Oberklasse aller graphischen Anzeigeklassen
1045 */
1046
1047 abstract class ChartAnzeiger extends Canvas {
1048     Vector v;
1049     int xSize;
1050     int ySize;
1051     int abstand;
1052
1053     ChartAnzeiger(Vector vec, int xSize, int ySize) {
1054         v = vec;
1055         this.xSize = xSize;
1056         this.ySize = ySize;
1057         if(vec.size() != 0) {
1058             abstand = xSize/vec.size();
1059         }
1060         setSize(xSize, ySize);
1061     }
1062
1063     public void paint(Graphics g) {
1064         g.drawRect(0, 0, xSize-1, ySize-1);
1065         for(int i = 0; i < v.size()-1; i++) {
1066             drawLine(g, i);
1067         }
1068     }
1069
1070     abstract void drawLine(Graphics g, int i);
1071 }
1072
1073
1074 /* graphische Darstellung aller Kurse eines Aktientitels fuer den heutigen
1075    Tag.
1076 */
1077
1078 class TagesChartAnzeiger extends ChartAnzeiger {
1079

```

```

1080 TagesChartAnzeiger(Vector vec, int xSize, int ySize) {
1081     super(vec, xSize, ySize);
1082 }
1083
1084 void drawLine(Graphics g, int i){
1085     g.drawLine(i*abstand,
1086                ySize - (int)((((TagesChartWert)(v.elementAt(i))).kurs).kurs),
1087                (i+1)*abstand,
1088                ySize - (int)((((TagesChartWert)(v.elementAt(i+1))).kurs).kurs));
1089 }
1090 }
1091
1092
1093 /* graphische Darstellung aller Kassakurse eines Aktientitels fuer die
1094 letzten 30 Tage.
1095 */
1096
1097 class MonatsChartAnzeiger extends ChartAnzeiger {
1098
1099     MonatsChartAnzeiger(Vector vec, int xSize, int ySize) {
1100         super(vec, xSize, ySize);
1101     }
1102
1103     void drawLine(Graphics g, int i){
1104         g.drawLine(i*abstand,
1105                    ySize - (int)((((MonatsChartWert)(v.elementAt(i))).kassakurs).kurs),
1106                    (i+1)*abstand,
1107                    ySize - (int)((((MonatsChartWert)(v.elementAt(i+1))).kassakurs).kurs));
1108     }
1109 }
1110
1111 /* Kern des Hauptprogramms: Auswahlmenue der einzelnen Darstellungsformen.
1112 */
1113
1114 class Auswahl extends Frame {
1115     Button beenden = new Button("Beenden");
1116     Button ok = new Button("OK");
1117     Panel p = new Panel();
1118     CheckboxGroup cbg = new CheckboxGroup();
1119     Checkbox cbtt = new Checkbox("Tagestabelle", cbg, true);
1120     Checkbox cbwt = new Checkbox("Wochentabelle", cbg, false);
1121     Checkbox cbmt = new Checkbox("Monatstabelle", cbg, false);
1122     Checkbox cbdt = new Checkbox("Dynamiktabelle", cbg, false);
1123     Checkbox cbtg = new Checkbox("Tagesgewinne", cbg, false);
1124     Checkbox cbltk = new Checkbox("letzte Tageskurse", cbg, false);
1125     Checkbox cbtc = new Checkbox("Tages-Chart", cbg, false);
1126     Checkbox cbmc = new Checkbox("Monats-Chart", cbg, false);
1127
1128     Auswahl(String titel) {
1129         super(titel);
1130         setLayout(new BorderLayout());
1131         p.setLayout(new GridLayout(6,1));
1132         p.add(cbtt);
1133         p.add(cbwt);
1134         p.add(cbmt);
1135         p.add(cbdt);
1136         p.add(cbtg);
1137         p.add(cbltk);
1138         p.add(cbtc);
1139         p.add(cbmc);
1140         add("Center",p);
1141         add("East",ok);
1142         add("South",beenden);
1143         pack();

```

```

1144     show();
1145 }
1146
1147 public boolean action(Event ev, Object target){
1148     if(ev.id == Event.ACTION_EVENT) {
1149         if(ev.target == ok) {
1150             if(cbg.getSelectedCheckbox() == cbtt) {
1151                 TagesTabelle tt = new TagesTabelle("Tagestabelle");
1152                 tt.stelleDar();
1153                 return true;
1154             } else if(cbg.getSelectedCheckbox() == cbwt) {
1155                 WochenTabelle wt = new WochenTabelle("Wochentabelle");
1156                 wt.stelleDar();
1157                 return true;
1158             } else if(cbg.getSelectedCheckbox() == cbmt) {
1159                 MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
1160                 mt.stelleDar();
1161                 return true;
1162             } else if(cbg.getSelectedCheckbox() == cbdt) {
1163                 MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
1164                 dt.stelleDar();
1165                 return true;
1166             } else if(cbg.getSelectedCheckbox() == cbtg) {
1167                 Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
1168                 tg.stelleDar();
1169                 return true;
1170             } else if(cbg.getSelectedCheckbox() == cbltk) {
1171                 LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
1172                 ltg.stelleDar();
1173                 return true;
1174             } else if(cbg.getSelectedCheckbox() == cbtc) {
1175                 TagesChart tc = new TagesChart("Tageschart");
1176                 tc.stelleDar();
1177                 return true;
1178             } else if(cbg.getSelectedCheckbox() == cbmc) {
1179                 MonatsChart mc = new MonatsChart("Monatschart");
1180                 mc.stelleDar();
1181                 return true;
1182             }
1183         } else if(ev.target == beenden) {
1184             System.exit(0);
1185             return true;
1186         }
1187     }
1188     return false;
1189 }
1190 }
1191
1192
1193 /* Programmeinstiegspunkt
1194 */
1195
1196 class Boerse {
1197     public static void main(String args[]) {
1198         Auswahl a = new Auswahl("Boersen-Information");
1199     }
1200 }
1201
1202

```

B.2 OMT diagrams – JAKK

On the following three pages you find the depictions of the inheritance hierarchy that were given to the subjects working the 0-level, 3-level, or 5-level program version, respectively.

Anwahl	Button beenden Button ok Panel p CheckboxGroup cbg Checkbox cbt Checkbox cbwt Checkbox cbwt Checkbox cbdt Checkbox cbg Checkbox cbhk Checkbox cbhc Checkbox cbmc Auswahl(String) public boolean action(Event, Object)
---------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Boerse	public static void main(String)
---------------	---------------------------------

Konstanten	static Datum heute static String kurse static String kassakurse
-------------------	-----------------------------------------------------------------------

Zeitraumdaten	Datum datum String wertpapierkennnummer String name Kurs kassakurs Zeitraumdatum(Datum, String, String, Kurs) public boolean vergleiche(Zeitraumdaten)
----------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tagesdaten	Datum datum Uhrzeit uhrzeit String wertpapierkennnummer String name Kurs kurs Tagesdatum(Datum, Uhrzeit, String, String, Kurs) public boolean vergleiche(Tagesdaten)
-------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

MonatsChartWert	Kurs kassakurs Datum datum MonatsChartWert(String) public boolean vergleiche(MonatsChartWert)
------------------------	--------------------------------------------------------------------------------------------------------

TagesChartWert	Uhrzeit uhrzeit Kurs kurs TagesChartWert(String) public boolean vergleiche(TagesChartWert)
-----------------------	-----------------------------------------------------------------------------------------------------

MonatsDynamikTabelle	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec Datum vonDatum TextArea textarea TextArea textarea MonatsDynamikTabelle(String) void setzeDatename(String) void setzeDatename(String) void leseDaten(String) void leseDaten(String) void erzeugeDaten(vector(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() float berechneDynamik(int) void zeigeTabelleAn()
-----------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

MonatsTabelle	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec Datum vonDatum TextArea textarea MonatsTabelle(String) void setzeDatename(String) void setzeDatename(String) void leseDaten(String) void leseDaten(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() void zeigeTabelleAn()
----------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

WocherTabelle	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec Datum vonDatum TextArea textarea WocherTabelle(String) void setzeDatename(String) void setzeDatename(String) void leseDaten(String) void leseDaten(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() void zeigeTabelleAn()
----------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

LetzteTageskurse	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec TextArea textarea Vector v LetzteTageskurse(String) void setzeDatename(String) void setzeDatename(String) void leseDaten(String) void leseDaten(String) void erzeugeDaten(vector(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() boolean schonAbgegeben(String) float letzteTageskurse(String, int) void zeigeTabelleAn()
-------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tagesgewinne	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec TextArea textarea Vector v Tagesgewinne(String) void setzeDatename(String) void setzeDatename(String) void leseDaten(String) void leseDaten(String) void erzeugeDaten(vector(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() boolean schonAbgegeben(String) float tagesgewinn(String, int) void zeigeTabelleAn()
---------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TagesTabelle	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec TextArea textarea TagesTabelle(String) void setzeDatename(String) void setzeDatename(String) void leseDaten(String) void leseDaten(String) void erzeugeDaten(vector(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() void zeigeTabelleAn()
---------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

MonatsChart	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec int xChartSize int yChartSize String wkn Label l Textfield t Button ok MonatsChartWert new Datum letzterMonat MonatsChart(String) void leseDaten(String) void erzeugeDaten(vector(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() Canvas zeigeCanvasAn()
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

TagesChart	Panel p1, p2, p3; Button schliessen Button beenden float kosten String datename Label sponsor Vector vec int xChartSize int yChartSize String wkn Label l Textfield t Button ok TagesChartWert new void leseDaten(String) void erzeugeDaten(vector(String) boolean selektiere(String) void quot(int, int) in partition(int, int) void exchange(int, int) void sortiere(int, int) void schliesseFenster() void beendeProgramm() public boolean action(Event, Object) boolean myaction(Event, Object) void stelleDarf() Canvas zeigeCanvasAn()
-------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

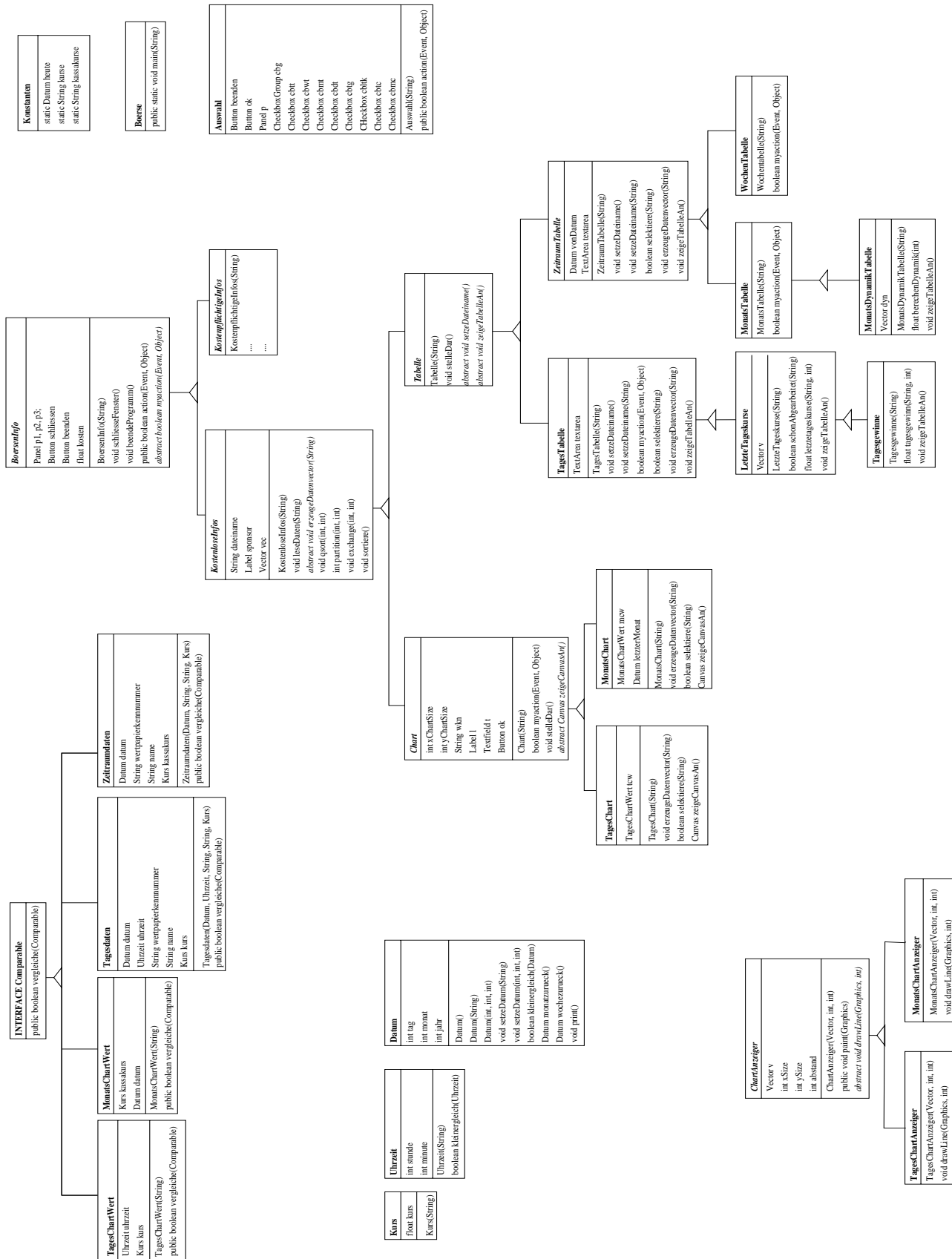
MonatsChartAnzeiger	Vector v int xSize int ySize int abstand MonatsChartAnzeiger(Vector, int, int) public void paint(Graphics) void drawLine(Graphics, int)
----------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------

TagesChartAnzeiger	Vector v int xSize int ySize int abstand TagesChartAnzeiger(Vector, int, int) public void paint(Graphics) void drawLine(Graphics, int)
---------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------

Datum	int tag int monat int jahr Datum() Datum(String) Datum(int, int, int) void setzeDatum(int, int, int) boolean kleinergleich(Datum) Datum monatzumeckl() Datum wochezumeckl() void print()
--------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Uhrzeit	int stunde int minute Uhrzeit(String) boolean kleinergleich(Uhrzeit)
----------------	-------------------------------------------------------------------------------

Kurs	float kurs Kurs(String)
-------------	----------------------------



B.3 Program differences between JAKK and Informatik II experiments

The Informatik II versions of the Boerse.java program are given here in the form of deltas to the corresponding JAKK versions. These deltas were produced by the Unix utility *diff*.

As one can see, the differences are rather uniform and are restricted to the `action()` and `myaction()` methods only.

B.3.1 Boerse.java (no inheritance)

```

7a8
> import java.awt.event.*;
79,80d79
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction() erweitert die action-Methode.
124a124,136
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
>
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
125a138,150
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
>
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
238,277d262
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
<   Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<   anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {

```

```

< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<
<                                     /* myaction erweitert die action-Methode, so dass nun beim Druck des
<       Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
<       aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
<       und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
<       */
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
281a267,285
> ok.addActionListener(
>
>                                     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                     //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
>                                     //in diesem Fall der ok-Knopf) spezialisierte
>                                     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
357,358d360
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction() erweitert die action-Methode.
403a406,418
> schliessen.addActionListener(
>
>                                     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                     //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
>                                     //in diesem Fall der schliessen-Knopf) spezialisierte
>                                     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }

```

```

> }
> );
404a420,432
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
514,553d541
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
< Knopfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
< anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
/* myaction erweitert die action-Methode, so dass nun beim Druck des
< Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
< aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
< und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
< */
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
< }
557a546,564
> ok.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der ok-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt

```

```

>
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> };
574c581
< MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
-
> MonatsChartWert(String) bekommt einen String, in dem das Datum
623,624d629
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
657a663,675
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> };
658a677,689
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> };
776,802d806
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
< Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
< anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);

```

```

< }
< } else {
< return false;
< }
< }
<
<
< /* Wird von "action" aufgerufen.
< */
< boolean myaction(Event ev, Object target){
< return true;
< }
<
<
1042,1043d1045
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1077a1080,1092
> schliessen.addActionListener(
>
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> // (in diesem Fall der schliessen-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1078a1094,1106
> beenden.addActionListener(
>
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> // (in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1196,1222d1223
<
< //Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
< //Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
< //anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }

```



```

<
<
<      /* Wird von "action" aufgerufen.
< */
< boolean myaction(Event ev, Object target){
< return true;
< }
<
1323,1324d1323
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1359a1359,1371
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1360a1373,1385
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1478,1504d1502
<
// Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
< Knopfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
< anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<
<
<
<      /* Wird von "action" aufgerufen.
< */
< boolean myaction(Event ev, Object target){
< return true;

```

```

< }
<
1596,1597d1593
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1632a1629,1641
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> };
1633a1643,1655
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> };
1639c1661
<
--
>
1751,1777d1772
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
< Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
< anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<
<
/* Wird von "action" aufgerufen.
< */
< boolean myaction(Event ev, Object target) {
< return true;
< }

```

```

<
1839,1840d1833
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1875a1869,1881
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1876a1883,1895
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1993,2019d2011
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
/* Wird von "action" aufgerufen.
< */
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
2081,2082d2072
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
2118a2109,2121

```

```

> schliessen.addActionListener(
>
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
> //((in diesem Fall der schliessen-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
2119a2123,2135
> beenden.addActionListener(
>
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
> //((in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
2236,2262d2251
< // Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
< Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
< anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< // Wird von "action" aufgerufen.
< */
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
2412a2402,2439
> ok.addActionListener(
>
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
> //((in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){

```

```

> public void actionPerformed(ActionEvent e){
> if(cbg.getSelectedCheckbox() == cbtt) {
> TagesTabelle tt = new TagesTabelle("Tagestabelle");
> tt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbwt) {
> WochenTabelle wt = new WochenTabelle("Wochentabelle");
> wt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmt) {
> MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
> mt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbdt) {
> MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
> dt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtg) {
> Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
> tg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtk) {
> LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
> ltg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtc) {
> TagesChart tc = new TagesChart("Tageschart");
> tc.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmc) {
> MonatsChart mc = new MonatsChart("Monatschart");
> mc.stelleDar();
> }
>
> }
> };
2414c2441,2454
< pack();
—
> beenden.addActionListener(
>
>
>
>
>
>
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> System.exit(0);
> }
> }
> };
> pack();
2417,2460d2456
<
< public boolean action(Event ev, Object target){
< if(ev.id == Event.ACTION_EVENT) {
< if(ev.target == ok) {
< if(cbg.getSelectedCheckbox() == cbtt) {
< TagesTabelle tt = new TagesTabelle("Tagestabelle");
< tt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbwt) {
< WochenTabelle wt = new WochenTabelle("Wochentabelle");
< wt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmt) {
< MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
< mt.stelleDar();
< return true;

```

```
< } else if(cbg.getSelectedCheckbox() == cbd) {
< MonatsDynamikTabelle dt = new MonatsDynamikTabelle("DynamikTabelle");
< dt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtg) {
< Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
< tg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbltk) {
< LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
< ltg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtc) {
< TagesChart tc = new TagesChart("Tageschart");
< tc.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmc) {
< MonatsChart mc = new MonatsChart("Monatschart");
< mc.stelleDar();
< return true;
< }
< } else if(ev.target == beenden) {
< System.exit(0);
< return true;
< }
< }
< return false;
< }
2462d2457
<
```

B.3.2 Boerse.java (3 levels of inheritance)

```

7a8
> import java.awt.event.*;
63,68d63
< action(Event, Object) verarbeitet Interaktionsereignisse
< abstract myaction(Event, Object)
< wird in "action" aufgerufen; Unterklassen
< koennen hier durch ueberschreiben
< spezielle Ereignisbehandlung
< implementieren
71c66
< abstract class BoersenInfo extends Frame implements Konstanten{
—
> abstract class BoersenInfo extends Frame implements Konstanten {
93a89,102
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeusserere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
>
94a104,116
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeusserere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
108,136d129
<
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();

```

```

< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<
<                                     /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
< werden, um eine spezielle Ereignisbehandlung zu
< implementieren. (Diese Konstruktion verhindert, dass die Behandlung
< der Knöpfe "Schliessen" und "Beenden" jedesmal neu implementiert
< werden muss.
< */
< abstract boolean myaction(Event ev, Object target);
278,279d270
< myaction() erweitert die action-Methode der Oberklasse.
< abstract zeigeCanvasAn() liefert ein Canvas zurueck.
300a292,310
> ok.addActionListener(
>                                     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                     //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
>                                     //(in diesem Fall der ok-Knopf) spezialisierte
>                                     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
306,324d315
<                                     /* myaction erweitert die action-Methode, so dass nun beim Druck des
< Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
< aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
< und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
< */
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
487c478
< MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
—
> MonatsChartWert(String) bekommt einen String, in dem das Datum
571,574d561

```



```

< boolean myaction(Event ev, Object target){
< return true;
< }
<
811,814d797
< boolean myaction(Event ev, Object target){
< return true;
< }
<
913,916d895
< boolean myaction(Event ev, Object target){
< return true;
< }
<
1007,1010d985
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
1071,1074d1045
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
1137,1140d1107
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
1285a1253,1289
> ok.addActionListener(
>
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
> //in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> if(cbg.getSelectedCheckbox() == cbbt) {
> TagesTabelle tt = new TagesTabelle("Tagestabelle");
> tt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbwt) {
> WochenTabelle wt = new WochenTabelle("Wochentabelle");
> wt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmt) {
> MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
> mt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbdt) {
> MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
> dt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtg) {
> Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
> tg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbltk) {
> LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
> ltg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtc) {
> TagesChart tc = new TagesChart("Tageschart");
> tc.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmc) {
> MonatsChart mc = new MonatsChart("Monatschart");
> mc.stelleDar();
> }

```

```

> }
> }
> );
1286a1291,1303
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> System.exit(0);
> }
> }
> );
1289,1332d1305
< }
<
< public boolean action(Event ev, Object target){
< if(ev.id == Event.ACTION_EVENT) {
< if(ev.target == ok) {
< if(cbg.getSelectedCheckbox() == cbtt) {
< TagesTabelle tt = new TagesTabelle("Tagestabelle");
< tt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbwt) {
< WochenTabelle wt = new WochenTabelle("Wochentabelle");
< wt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmt) {
< MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
< mt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbdt) {
< MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
< dt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtg) {
< Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
< tg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbltk) {
< LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
< ltg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtc) {
< TagesChart tc = new TagesChart("Tageschart");
< tc.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmc) {
< MonatsChart mc = new MonatsChart("Monatschart");
< mc.stelleDar();
< return true;
< }
< } else if(ev.target == beenden) {
< System.exit(0);
< return true;
< }
< }
< return false;

```

B.3.3 Boerse.java (5 levels of inheritance)

```

7a8
> import java.awt.event.*;
64,69d64
< action(Event, Object) verarbeitet Interaktionsereignisse
< abstract myaction(Event, Object)
< wird in "action" aufgerufen; Unterklassen
< koennen hier durch ueberschreiben
< spezielle Ereignisbehandlung
< implementieren
72c67
< abstract class BoersenInfo extends Frame implements Konstanten{
—
> abstract class BoersenInfo extends Frame implements Konstanten {
94a90,102
> schliessen.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeusserere Objekt
//((in diesem Fall der schliessen-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> };
95a104,116
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeusserere Objekt
//((in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> };
109,137d129
<
<
/* Wird bei Interaktions-Ereignissen aufgerufen. Druucken der
Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
anderen Ereignisse wird "myaction" aufgerufen.
< */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;

```

```

< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<
<                                     /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
< werden, um eine spezielle Ereignisbehandlung zu
< implementieren. (Diese Konstruktion verhindert, dass die Behandlung
< der Knoepfe "Schliessen" und "Beenden" jedesmal neu implementiert
< werden muss.
< */
< abstract boolean myaction(Event ev, Object target);
279d270
< myaction() erweitert die action-Methode der Oberklasse.
301a293,311
> ok.addActionListener(
>
>                                     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                     //Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
>                                     //in diesem Fall der ok-Knopf) spezialisierte
>                                     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                     //dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt
>                                     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
306,325d315
<
<                                     /* myaction erweitert die action-Methode, so dass nun beim Druck des
< Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
< aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
< und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
< */
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
488c478
< MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
—
> MonatsChartWert(String) bekommt einen String, in dem das Datum
552,555d541
< boolean myaction(Event ev, Object target){

```

```

< return true;
< }
<
962,964d947
< boolean myaction(Event ev, Object target) {
< return true;
< }
979,981d961
< boolean myaction(Event ev, Object target) {
< return true;
< }
1141a1122,1158
> ok.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//in diesem Fall der ok-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> if(cbg.getSelectedCheckbox() == cbbt) {
> TagesTabelle tt = new TagesTabelle("Tagestabelle");
> tt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbwt) {
> WochenTabelle wt = new WochenTabelle("Wochentabelle");
> wt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmt) {
> MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
> mt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbdt) {
> MonatsDynamikTabelle dt = new MonatsDynamikTabelle("DynamikTabelle");
> dt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtg) {
> Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
> tg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbltk) {
> LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
> ltg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtc) {
> TagesChart tc = new TagesChart("Tageschart");
> tc.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmc) {
> MonatsChart mc = new MonatsChart("Monatschart");
> mc.stelleDar();
> }
> }
> }
> }
> }
1142a1160,1172
> beenden.addActionListener(
>
//Die folgende Konstruktion bedeutet, dass eine unbenannte innere
//Klasse erzeugt wird. Diese stellt eine fuer das aeuessere Objekt
//in diesem Fall der beenden-Knopf) spezialisierte
//actionPerformed()-Methode zur Verfuegung. Sie wird immer
//dann ausgefuehrt, sobald eine Aktion auf dem aeuesseren Objekt
//ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> System.exit(0);
> }
> }
> }
> }
1147,1189d1176

```

```
< public boolean action(Event ev, Object target){
< if(ev.id == Event.ACTION_EVENT) {
< if(ev.target == ok) {
< if(cbg.getSelectedCheckbox() == cbtt) {
< TagesTabelle tt = new TagesTabelle("Tagestabelle");
< tt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbwt) {
< WochenTabelle wt = new WochenTabelle("Wochentabelle");
< wt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmt) {
< MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
< mt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbdt) {
< MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
< dt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtg) {
< Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
< tg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbltk) {
< LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
< ltg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtc) {
< TagesChart tc = new TagesChart("Tageschart");
< tc.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmc) {
< MonatsChart mc = new MonatsChart("Monatschart");
< mc.stelleDar();
< return true;
< }
< } else if(ev.target == beenden) {
< System.exit(0);
< return true;
< }
< }
< return false;
< }
```

Appendix C

Translations of German terms

The following translations are given in order to help the reader understand the German Java source code.

abstand	distance
Auswahl	selection
beendeProgramm	terminateProgram
beenden, Beenden	terminate
berechneDynamik	calculateDynamics
Bitte	please
Boerse	stock exchange
Boersen-Information	stock information
BoersenInfo	stock info
ChartAnzeiger	chartDisplay
Datei	file
dateiname	file name
Daten	data
Datum, datum	date
Dynamiktabelle	DynamicsTable
erzeugeDatenvektor	generateDataVector
gefunden	found
geoeffnet	opened
gesponsort	sponsored
gewuenschten	desired
heute, heutige	today, today's
jahr	year
KASSAKURSE, kassakurse, kassakurs	spot rates
keine	no, none
keineDaten	noData
kleinergleich	lessThan
kosten	costs
KURSE, Kurs, kurs, kurse	price, prices
Konstanten	constants

KostenloseInfos	FreeInfos
KostenpflichtigeInfos	InfoWithCosts
leseDaten	readData
Lesefehler	ReadError
letzte, letzter	last
letzterMonat	lastMonth
letztetageskurse, LetzteTageskurse	lastDayPrice
monat	month
MonatsChart, Monatschart	MonthChart
MonatsChartAnzeiger	MonthChartDisplay
MonatsChartWert	MonthChartValue
Monatsdaten	data for month
MonatsDynamikTabelle	MonthDynamicTable
MonatsTabelle, Monatstabelle	MonthTable
monatzurueck	month back
nicht	not
schliesseFenster	closeWindow
schliessen, Schliessen	close
schonAbgearbeitet	alreadyDone
Seiten	pages
selektiere	select
setzeDateiname	setFileName
setzeDatum	setDate
sortiere	sort
stelleDar	display, present
stunde	hour
Tabelle	table
tag	day
Tages-Chart, TagesChart, Tageschart	DayChart
TagesChartAnzeiger	DayChartDisplay
TagesChartWert	DayChartValue
Tagesdaten, tagesdaten	data for day
TagesTabelle	DayTable
tagesgewinn, Tagesgewinne	DayGains
Tageskurse	DayPrices
Tagestabelle	DayTable
titel	title
Uhrzeit, uhrzeit	time of day
vergleiche	compare
vonDatum	fromDate
Wertpapierkennnummer	stock ID number
WochenTabelle	WeekTable
wochezurueck	week back
zeigeCanvasAn	presentCanvas
zeigeTabelleAn	presentTable
Zeitraum	time interval
ZeitraumTabelle	IntervalTable
Zeitraumdaten	IntervalData

Bibliography

- [1] Larry B. Christensen. *Experimental Methodology*. Allyn and Bacon, Needham Heights, MA, 6th edition, 1994.
- [2] J. Daly, J. Miller, J. Brooks, M. Roper, and M. Wood. Issues on the object-oriented paradigm: A questionnaire survey. Technical report, University of Strathclyde, EFoCS-8-95, 1995.
- [3] J. Daly, M. Wood, J. Brooks, J. Miller, and M. Roper. Structured interviews object-oriented paradigm. Technical report, University of Strathclyde, EFoCS-7-95, 1995.
- [4] John W. Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. An empirical study evaluating depth of inheritance on the maintainability of object-oriented software. *Empirical Software Engineering, An international Journal*, 1(2):109–132, 1996.
- [5] Al Davis. Eras of software technology transfer. *IEEE Software*, 13(2):4–7, March 1996.
- [6] Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, 1991.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [8] Lutz Prechelt, Barbara Unger, and Douglas Schmidt. Replication of the first controlled experiment on the usefulness of design patterns: Detailed description and evaluation. Technical report, Washington University, St. Louis, Dept. of CS, 1997.
- [9] Helmut Roderus and Dieter Krißgau. Erfahrungen mit dem Einsatz objektorientierter Methoden. *OBJEK-Tspektrum*, 1(1):43–47, 94.