Seminar 4, Exceptions and Design Patterns

Object-Oriented Design, IV1350

Intended Learning Outcomes

This seminar concerns the learning outcomes *develop an object-oriented program by applying established guidelines for object-oriented architecture, design and programming* and *discuss the quality of a program referring to established guidelines for object-oriented architecture, design and programming.*

Goal

- Practice designing and coding exception handling.
- Practice designing and coding polymorphism and design patterns

Literature

- Chapters eight and nine in *A First Course in Object Oriented Development.*

Grading

There are three possible grades:

Fail (0 points) To pass the course you must pass all four seminars. If you fail this seminar you have to report it again at the end of the course, at the fifth seminar.

1 point Active participation in seminar discussions. Written solution submitted in Canvas proving that you can, with minor defects, apply a design pattern. It should also prove that you can use an exception to handle an error.

2 points Active participation in seminar discussions. Written solution submitted in Canvas proving that you can, without defects, apply design patterns and explain and motivate their use. It should also prove that you understand how to write and use exceptions for error handling.

Tasks

Task 1

a) Use exception(s) to handle alternative flow 3-4a in *Process Sale*, see document with tasks for seminar one. An exception shall be thrown to indicate that a search has been made for an identifier that did not exist in the inventory catalog.

b) Also use exception(s) to indicate that the database can not be called, it might be for example that the database server is simply not running. Since there is no real database, you must simulate this situation. That can be done by always throwing a database failure exception when a search is made for a particular, hardcoded, item identifier.

The bullets below from chapter eight about exception handling must be implemented.

- Choose between checked and unchecked exceptions.
- Use the correct abstraction level for exceptions.
- Name the exception after the error condition.
- Include information about the error condition.
- Use functionality provided in `java.lang.Exception`
- Write javadoc comments for all exceptions.
- An object shall not change state if an exception is thrown.
- Notify users.
- Notify developers.
- Write unit tests for the exception handling.

The program shall produce the following output.

- The user interface shall show an informative message when an exception is caught in the view. Apart from this, the grade is not affected no matter how simple or advanced the view is.
- An error report shall be written to a log when an exception is caught, and that exception indicates the program is not functioning as intended. This logging of errors shall be written to a file. How to print to a file is illustrated in `se.leiflindback.oodbook.polymorphism.logapi.FileLogger`, which can be found in listing 9.1 in the textbook and in the book's git repository.

There must also be unit tests for exception handling, as described in the section *Write unit tests for the exception handling* in the course literature. In the report, you have to include the following.

- In the `Method` chapter of your report, explain how you worked and how you reasoned when implementing exception handling.
- In the `Result` chapter of your report, briefly explain the program. Include links to your git repository, and make sure the repository is public. Also include a printout of a sample run.
- In the `Discussion` chapter of your report, evaluate your program using applicable assessment criteria from the document `assessment-criteria-seminar4.pdf`, which is available on the *Seminar Tasks* page in Canvas.

Task 2

To get two points for this lab you must give good solutions to both parts a and b below. To get one point it is enough to solve only part a and *not* part b.

Part a

In your *Process Sale* program, use the Observer pattern to implement a new functionality, namely to show the sum of the costs for all sales made since the program started. This total income shall be handled by two new classes. The first, `TotalRevenueView`, shall be placed in the view and show the total income on the user interface, for example by printing to `System.out`. The second, `TotalRevenueFileOutput`, shall print the total income to a file. How to print to a file is illustrated in `se.leiflindback.oodbook.polymorphism.logapi.FileLogger`, which can be found in listing 9.1 in the textbook and in the book's git repository. These two classes handling the total income shall never call the controller or any other class, but instead be updated using the Observer pattern. Both shall implement the same observer interface. The grade is not affected no matter how simple or advanced the view is.

- In the `Method` chapter of your report, explain how you worked and how you reasoned when implementing the observer pattern.
- In the `Result` chapter of your report, briefly explain source code for all classes you changed when implementing the Observer pattern. Include links to your git repository, and make sure the repository is public. Also include a printout of a sample run.
- In the `Discussion` chapter of your report, evaluate your program using applicable assessment criteria from the document `assessment-criteria-seminar4.pdf`, which is available on the *Seminar Tasks* page in Canvas.

Part b, only for two points

Use two more GoF patterns in your *Process Sale* program. You are free to choose any GoF patterns, apart from Observer and Template Method, since the former is used here in task 2a and the latter is used in *Additional Higher Grade Tasks*. You are allowed to choose GoF patterns not covered at the lectures. A suggestion for one of the patterns is to turn some registry/database into a singleton. Another suggestion is to use Strategy (maybe also Composite) for discount calculation.

You are *not* allowed to copy entire files or classes from code samples written at the lectures, even if you understand it and/or change it. *You are for example not allowed to use the logging example from the lecture on polymorphism*, to implement the Strategy pattern. You are, however, allowed to write code very similar to code examples from lectures.

- In the `Method` chapter of your report, explain how you worked and how you reasoned when implementing the chosen patterns.
- In the `Result` chapter of your report, briefly explain source code for all classes you changed when implementing the chosen patterns. Include links to your git repository, and make sure the repository is public. Also include a printout of a sample run.
- In the `Discussion` chapter of your report, evaluate your program using applicable assessment criteria from the document `assessment-criteria-seminar4.pdf`, which is available on the *Seminar Tasks* page in Canvas.