Object Oriented Software Engineering

Practical 05

Paolo Ballarini

Learning outcomes:

- Sorting of colletions
- implementation of interfaces, extensions of abstract classes
- Definition of equivalence relationship for objects of a class (overloading of `equals`)
- Application of JUnit testing
- Modelling with UML class diagrams

Exercise 1. Sorting of books

Define a `Book` data type for representing book objects knowing that each book is characterised by: a title, an author, a price, a number of pages.

Q1) Define a *natural ordering* for the `Book` data type according to which books are sorted w.r.t. their title. Test your implementation through a short client program where a collection of `Book` is created and populated with the following book objects.

- `book1`: "The Picture of Dorian Gray", "Oscar Wilde", 256 pages, 6.5 Euro
- `book2`: ''Perfume", "Peter Suskind", 272 pages, 10 Euro
- `book3`: ''Cien anos de Soledad", "Gabriel Garcia Marquex", 496 pages, 8 Euro

and the books in the collection are displayed according to the above described natural ordering.

Q2) Extend the design of the `Book` data type by allowing the client program to sort a collection of `Book` objects w.r.t any of the following criteria:

- *sort by author*: books are sorted in lexicographical order w.r.t. the author's name
- *sort by price*: books are sorted w.r.t. their price
- *sort by num. pages*: books are sorted w.r.t. the number of pages

Test your implementation through a short client program where the previously created collection of `Book` objects is displayed in ascending w.r.t. any of the above mentioned sorting criteria.

Exercise 2. A class hierarchy for Complex numbers

Bearing in mind that a complex number can be expressed as `a + bi`, where `a` and `b` are real numbers and `i` is the imaginary unit you are required to develop (using the Eclipse IDE) a class hierarchy for representing complex numbers. In particular

Q1) Develop a class called `Complex` that extends class `java.lang.Number`. In so doing make sure that `Complex` implements all required methods (**Hint**: use the java.lang API to find documentation about `Number` and `Object`).

Q2) Equip the `Complex` class with an implementation of methods: `equals()` (for comparing if two `Complex` objects represent the same complex number), `hashCode()` and `toString()` to display `Complex` objects.

Q3) In addition, implement the following interface for the `Complex` class:

```
public interface BasicOps {

        // Add to this number and return it as the result
        public Number addNum(Number a);

        // Subtract from this number and return it as the result
        public Number subNum(Number b);

        // Multiply to this number and return it as the result
        public Number multNum(Number a);

        // Divide this number by and return it as the result
        public Number divNum(Number b);

    }
```

Q4) Add to class `Complex` a simple `main` method where you create a couple of `Complex` objects and display the result of their sum.

Q5) Implement a class `Fraction` for representing fraction numbers (e.g. 1/3) where enumerators of denominator are integers. Class `Fraction` should be derived from `Number` and implement the interface `BasicOps`. Add to class `Fraction` a simple `main` method where you create a couple of `Fraction` objects and display the result of their sum.

Q6) Using JUnit define appropriate test units for testing the implementation of the four basic operations for both classes Complex and Fraction. Run the tests and check the results.

Q7) Draw a UML class diagram of the class hierarchy for this exercise.

Q8) You used two Java mechanisms: subclassing and interfaces. When did you use each? Should Fraction be implemented as a subclass of Complex? Why?

Practical OOSE - Sequence diagrams

Exercise 1. Sequence diagrams for modelling an ATM system

Build a model of a scenario of the *withdraw money* use case of a Bank ATM system. In such scenario a bank customer (the User) is able to make withdrawal of money operations on an ATM by using her bank card. The system employs a standard procedure of validating the card and account holders password. You are asked to develop first a model of a simplified, 2-components version of the ATM withdrawal scenario, then a second one, which consider a more complex 5-components version of the system.

Q1) Develop a UML sequence diagram for a two actors ATM system described as follows
Actors: the ATM systems consists of the following two kind of objects

- Cardholder (or Customer)
- ATM System

Events: Describe the main flow of events in this scenario by completing the list of events listed below:

- Customer arrives at the ATM machine and inserts a bank card.
- The system requests for user authentication (password).
- Customer insert PIN number
- ...

Once you have identified all relevant events translate them into corresponding system events (input and response) by completing the input/response table given below:

Actor	System response
User inserts card	
	System prompts user to enter PIN
User types PIN	
...	...
...	...
...	...

Q2) Develop a UML sequence diagram for a six actors version of the ATM system described as follows. Now model the scenario of the Withdraw Money use case in more detail. The sequence diagram is elaborated in more detail by including new internal objects of the withdraw money use case such as Card Controller (to control card management) and Bank (the issuing bank).
Actors: identify the objects of such system based on the few given below (do you need more?)

- User / Customer / Cardholder
- ATM System
- Card Controller
- Cash Dispenser
- Bank
- Account

Events: Extends/Modify the main flow of events you identified for the first scenario by including interactions with the 4 added actors Card Controller and Bank. The first 2 events are listed below:

- Customer arrives at the ATM machine and inserts a bank card.
- The card is verified by the Card Controller.

Once you have identified all relevant events translate them into corresponding system events (input and response) by filling in the corresponding input/response table given below.

Actor	System response
...	...
...	...
...	...

Exercise 2. Implementation of ATM system

Based on the use case and sequence diagrams scenarios identified in the previous exercise build a Java implementation of the ATM system starting from the Scenario 1 and then extending it to the Scenario 2. To this aim you should:

Q1) Develop a UML class diagram identifying the classes involved in your implementation, their relationships (generic adirectional association? generic directional association?) as well as the most relevant attributes and methods of each class.

Q2) Based on the class diagram you have developed in the previous question give a Java implementation of the ATM systems. Test the implementation through a `main()` method which simulates the second scenario (that corresponding to the 6 components version of the system (Q2 of Exercise 1).