Lab exercise 4: Implementing a complex design

1 Introduction

The aim of this lab session is to implement the design of a bookstore application from Seminar 4. The session is mandatory and you have to deliver the source code of the Java project and a document describing the implementation decisions and details.

As usual, you are free to implement your own design or use the sample design that appears in the Aula Global. In either case, your program has to read an XML file that contains information about books in the store. How to read XML files in Java is explained below.

2 Code

In the Aula Global you will find a compressed file called `P4Code.zip` that contains various other files. If you decide to implement your own design, it is mandatory to incorporate the Java class `Payment` and the XML file `books.xml` into your program. In addition, the class `BookCollection` contains a method `readCatalog` that you can copy and use to read the XML file `books.xml`.

If you decide to implement the sample design, you can also incorporate the remaining Java classes and interfaces from `P4Code.zip` into your program. In this case, the lab session consists in implementing the four classes `Stock`, `Book`, `Catalog` and `ShoppingCart`, as well as a class `TestStore` with a main method that launches an application of type `BookStore`.

3 Reading XML files

The XML format is often used to save information to files. Java contains a library for reading XML files. The class `BookCollection` contains a method `readCatalog` that reads information about the books in the store from the XML file `books.xml`. This method returns a list of arrays of strings. Each array of strings contains information about a single book. Each piece of information is stored in a particular index of the array:

0. Title.
1. Author.
2. Date of publication.
3. Place of publication.
4. ISBN.
5. Price.
6. Currency.
7. Number of copies in the store.

This information should be used to create instances of the classes `Book` y `Stock` (or similar classes in your own design). It is necessary to iterate over the list returned by the method `readCatalog` to access each array of information about a book. The following example illustrates how to iterate over a linked list (in the example each element is a string):

```
LinkedList<String> list = new LinkedList<String>();
// add elements to the list
for ( String element : list ) {
    System.out.println( element ); // or another operation on elements
}
```

Some of the information has to be *converted* to use it. For example, the publication date should be represented as an instance of the `Date` class, the ISBN number as a long integer (`long`), etc. The following example illustrates how to convert the information stored in a string `myString` to the appropriate type:

```
Date date = new Date();
try { date = new SimpleDateFormat().parse( myString ); } // Date instance
catch( Exception e ) {}
long isbn = Long.parseLong( myString );                   // convert to long
double price = Double.parseDouble( myString );            // convert to double
Currency currency = Currency.getInstance( myString );     // Currency instance
int copies = Integer.parseInt( myString );                // convert to int
```

4 Sample design

In this section we briefly explain the logic behind the sample design. The class `BookStore` represent the main application which includes a graphical interface that allows a user to select books from the catalog and pay for the books in the shopping cart.

The class `BookCollection` represents a collection of books, and is inherited by the classes `Catalog` and `ShoppingCart`. Instead of representing the number of copies as an attribute of the class `Book`, there is a separate class `Stock` that stores information about a book which is specific to the store (such as the number of copies, the price and the currency). The information about a book itself has to be accessed via the class `Stock`.

The class `BookCollection` includes existing methods for adding or removing copies of a book from the collection. These methods assume that there exists an instance of `Stock` in the collection whose associated book has the given title. To correctly initialize its collection, the class `Catalog` should process the information parsed using the method `readCatalog` of `BookCollection`. The class `ShoppingCart` has to add an instance of `Stock` to its collection for each book in the catalog, initially with a number of copies equal to 0 (representing the fact that the user has not selected any copy). When the user makes an order or cancels an order, an appropriate number of copies of a book should be moved from the catalog to the shopping cart or vice versa. The class `ShoppingCart` also needs a method `checkout` that causes the user to pay for the books in the shopping cart. There is no explicit user class in the design; instead, you can invent user information in the method `checkout` in order to call the method `doPayment` of the class `Payment`.

5 Document

Do not forget to include a document that describes the development of your program. The guidelines for the document appear in the handout for Lab Session 1.