Ayesha Quinto Gammuac 241789

Minh Nguyen Phuong 241841

24292 - Object-Oriented Programming

LAB 5: Implementing a complex design

REPORT

THE PROGRAM

1. MAIN PURPOSE AND DESCRIPTION

The aim of this report is to describe how we have done lab 5, including all the problems we encountered and our solutions. The objective of the program is to implement the design of a bookstore application from Seminar 5. In this lab, we were given various of files: *BookCollection.java, BookCollectionInterface.java, books.xml, BookStore.java, Payment.java, ShoppingCartInterface.java* and *StockInterface.java.* We then need to create four more classes which are *Stock.java, Book.java, Catalog.java* and *ShoppingCart.java,* as well as the equivalent testing classes following the given sample design. We also created the class *Date.java*.

1.1. Book

This class represents a physical book. Its attributes are the title, the author, the publication date and place and the isbn code. As methods, we have the constructor with the corresponding parameters and the getters for each of the attributes mentioned: *getTitle(), getAuthor(), getPublicationDate(), getPublicationPlace()* and *getISBN()*.

1.2. Stock

The class Stock was to implement the class StockInterface. We created four attributes: book, copies, price and currency. Beside the constructor, we had implemented the five methods in StockInterface. The concept of this class is that there is an instance in the collection whose associated book has the given title and later on, we can use it to add to the ShoppingCart or BookCollection.

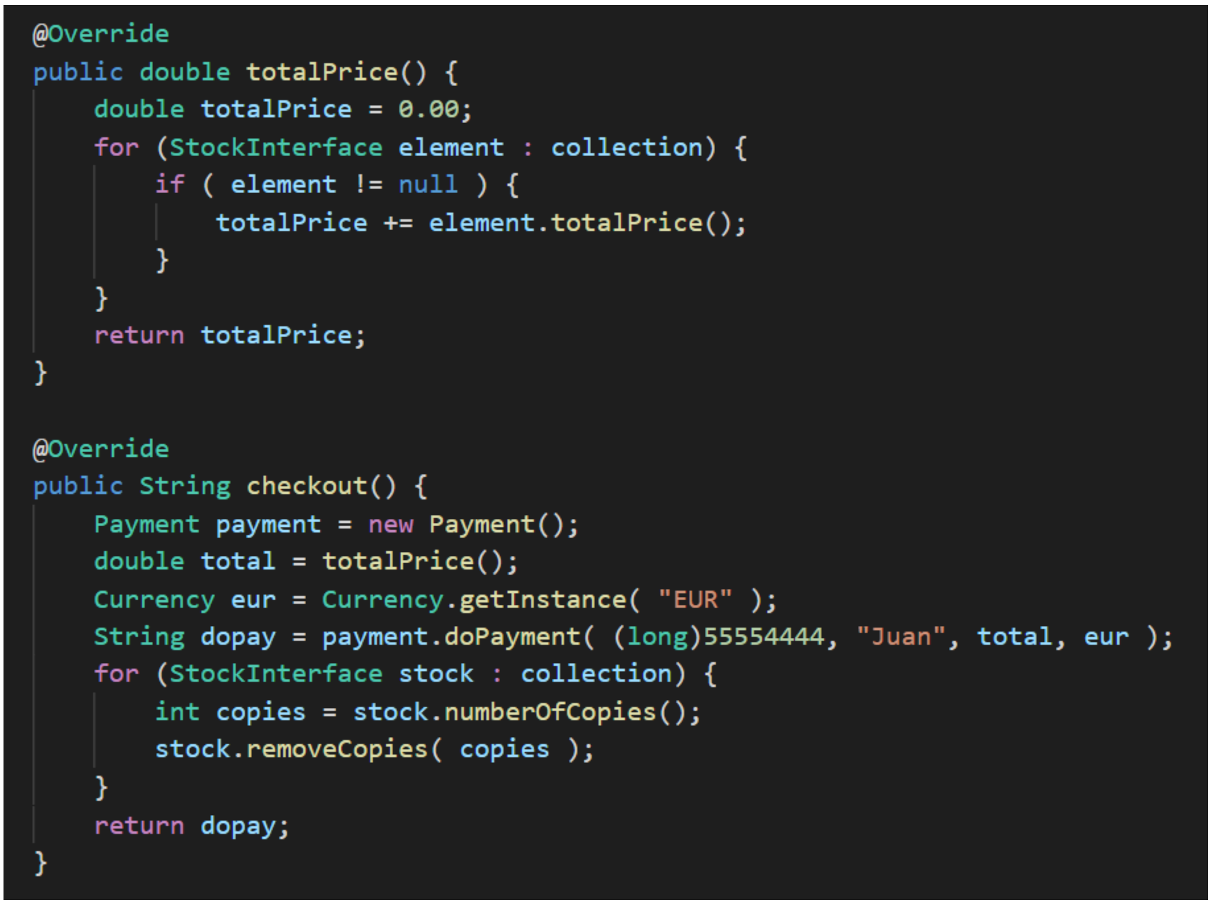
1.3. Catalog

Representing the catalog of the bookstore, we have this class that inherits from *BookCollection.java* and has as attribute a LinkedList of Stock. The constructor receives a String which will be the file name, and it will read the xml file with the method *readcatalog()*

from BookCollection, creating Stocks for each of the books in the file and populating the collection of the superclass.

1.4. Shopping Cart

This class inherited from BookCollection and implemented the class StockInterface. We are dealing with two BookCollection: the catalog of the shop and the collection of our shopping cart. Receiving the catalog from the xml file, we create our Shopping cart initializing the protected attribute *catalog* with it and duplicating this same catalog but with each stock initialized with 0 copies (since the user hasn't put anything yet in the cart). For the user to add copies of books or remove them from the cart, we overridden the two methods *addCopies()* and *removeCopies().* When the customer adds a stock to the shopping cart, we are removing one from the shop catalog. On the other hand, when the user removes the selected book from the cart, we are removing it from the collection of our cart and adding it (returning it) to the shop catalog. We also implemented the two methods from the StockInterface that are *totalPrice()* which will return the sum of all the prices for each copy of book in the cart and *checkout()* where the user pays and we erase the copies in the catalog the number of copies that the user bought. When doing it, we had some problems that are going to be explained below.



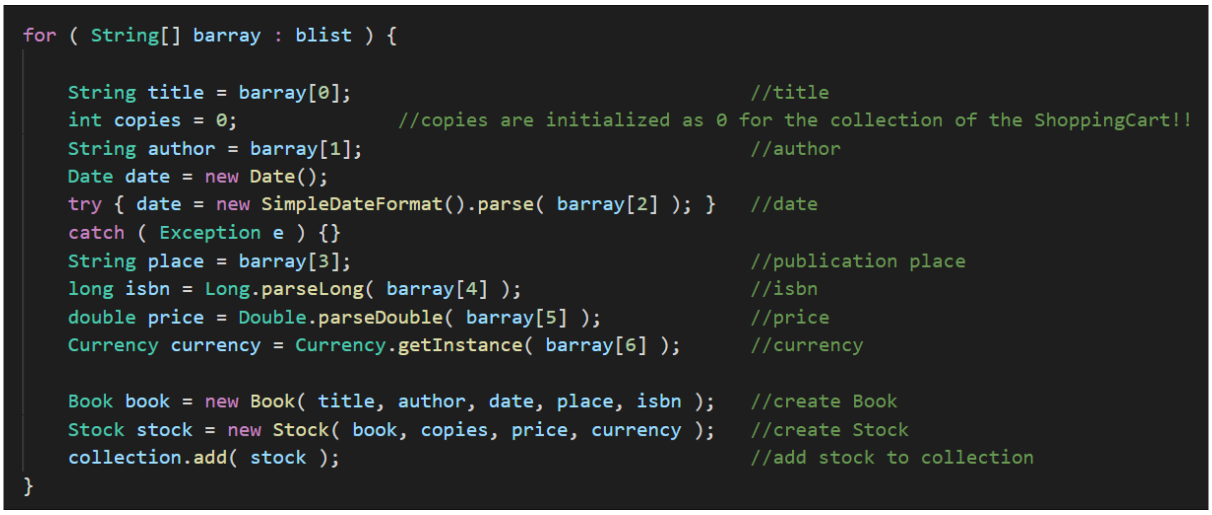
```java
@Override
public double totalPrice() {
    double totalPrice = 0.00;
    for (StockInterface element : collection) {
        if ( element != null ) {
            totalPrice += element.totalPrice();
        }
    }
    return totalPrice;
}

@Override
public String checkout() {
    Payment payment = new Payment();
    double total = totalPrice();
    Currency eur = Currency.getInstance( "EUR" );
    String dopay = payment.doPayment( (long)55554444, "Juan", total, eur );
    for (StockInterface stock : collection) {
        int copies = stock.numberOfCopies();
        stock.removeCopies( copies );
    }
    return dopay;
}
```

2. STRUGGLES AND PROBLEMS ENCOUNTERED

When we started implementing the class ShoppingCart, we had trouble understanding the mechanism, we didn’t grasp when the methods addCopies and removeCopies were called and which attributes they affected.

Another problem we encountered was in class Catalog. We needed to implement the code to read the file and fill the collection. However, since the method *readcatalog()* returns a LinkedList of String[] we had to convert all the information into the corresponding classes and create the Stock instances. For that, we had the instructions from the pdf but we still had to search for the conversion from string to a class representing Date to fully understand the conversion. In that matter, we thought about creating our own Date class since the java.util.Date constructor was deprecated and only accepted the date as milliseconds, but hopefully we found the right class to use which was in java.text.SimpleDateFormat.

The next problem we had was when we cancel or check out an order, the price goes back to 0, but when we make a new order, the program also sums the total price of the old order. To fix this problem, we had to create a book collection inside the shopping cart (before we only had a catalog because we thought the book collection would automatically change if we remove or add books). We also were struggling a bit to understand the *addCopies()* and *removeCopies()* methods. In the end, we decided to create an array of books, so that we can add and remove copies easily.



```
for ( String[] barray : blist ) {

    String title = barray[0];                                   //title
    int copies = 0;              //copies are initialized as 0 for the collection of the ShoppingCart!!
    String author = barray[1];                                  //author
    Date date = new Date();
    try { date = new SimpleDateFormat().parse( barray[2] ); }   //date
    catch ( Exception e ) {}
    String place = barray[3];                                   //publication place
    long isbn = Long.parseLong( barray[4] );                    //isbn
    double price = Double.parseDouble( barray[5] );             //price
    Currency currency = Currency.getInstance( barray[6] );      //currency

    Book book = new Book( title, author, date, place, isbn );   //create Book
    Stock stock = new Stock( book, copies, price, currency );   //create Stock
    collection.add( stock );                                    //add stock to collection
}
```

3. CONCLUSION

In general, the lab was quite challenging since it has many classes to be implemented, and we did not have much time since we also were preparing for the finals. On the one hand, we did not have many problems implementing the two classes *Book.java* and *Stock.java*. On the other hand, we did find some challenges as we explained above about the other classes. Throughout the lab, we were able to understand a lot more about the concept of abstract classes and interfaces and how a Shop of books in this case would work.