Pablo Fernández
Clara López
S101, P101

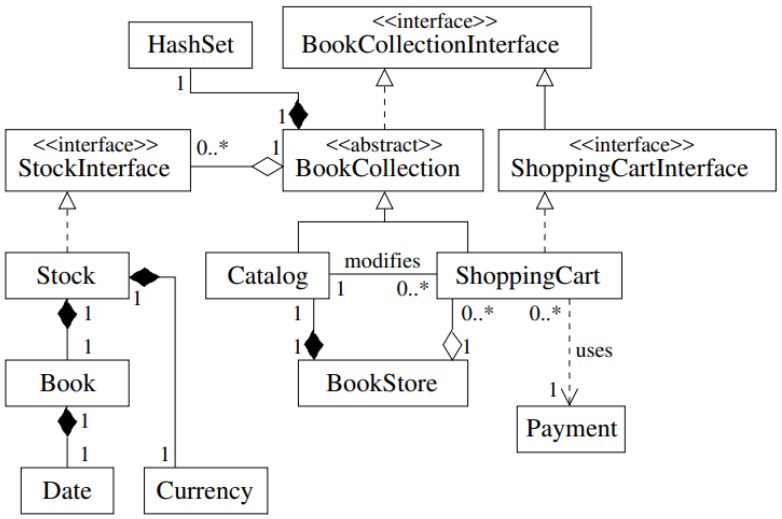
# Lab 5 Report

## Introduction

This report presents the work done in the fifth laboratory project of the Object-Oriented Programming course.

The topic of this project was completely different from the previous ones: it consisted in creating a bookstore; and as it is the last project of the subject, the tools implemented and used and the design itself was somehow more complex than the others. More specifically, it allowed us to get more familiar with the concept of *interface* and its implementation, as well as abstract classes and inheritance. Concepts such as overriding or protected visibility had to be well understood in order to elaborate this project.

The following diagram shows the given design for the project.




The goal of the program is to represent the way a bookstore works, that is, to buy copies of some books that belong to its particular catalog, using a shopping cart.Some of the classes are really similar, therefore we may reuse the instance members more than once in order to simplify the code.

Pablo Fernández
Clara López
S101, P101

# Description

1. Book

First we defined the class *Book* class.

```java
public class Book {
    private String title;
    private String author;
    private Date publicationDate;
    private String publicationPlace;
    private long ISBN;

    public Book(String tlinit, String auinit, Date dtinit, String plinit, long isbn){
        title = tlinit;
        author = auinit;
        publicationDate = dtinit;
        publicationPlace = plinit;
        ISBN = isbn;
    }
```

As we can see in the photo, we have defined as attributes the title, the author, the publication date and place and the ISBN of the book.

Then we have defined the Constructor method.

```java
public String getTitle(){
    return title;
}

public String getAuthor(){
    return author;
}

public Date getPublicationDate(){
    return publicationDate;
}

public String getPublicationPlace(){
    return publicationPlace;
}

public long getISBN(){
    return ISBN;
}
```

Later on, we defined the Getters of each one of the attributes that the book has.

2. Stock

Secondly, we defined the *Stock* class, which stores the information of a book that stores information about a book which is specific to the store. The information of a book has to be accessed via this class.

Pablo Fernández
Clara López
S101, P101

```java
public class Stock implements StockInterface{
    private Book book;
    private int copies;
    private double price;
    private Currency currency;

    public Stock(Book bookinit, int copinit, double priceinit, Currency curinit){
        book = bookinit;
        copies = copinit;
        price = priceinit;
        currency = curinit;
    }
```

As we can see here, the class *Stock* implements methods defined in the class *StockInterface*, that's why in some of the methods we have defined, we use *@Override*.
As attributes we have defined book, copies (which represents the number of copies this book has in the store), price (the price this book has) and currency.

```java
@Override
public Book getBook(){
    return book;
}

@Override
public String getBooktitle(){
    return book.getTitle();
}

@Override
public int numberOfCopies(){
    return copies;
}
```

Here we can see two Getters in order to obtain the book, the title of the book we are referring to and the number of copies this book has.

```java
@Override
public void addCopies(int numberOfCopies){
    copies += numberOfCopies;
}

public void removeCopies(int numberOfCopies){
    copies -= numberOfCopies;
}
```

Then we have defined the methods addCopies and removeCopies. These two methods are used to add/remove the number of copies of a certain book.

```java
@Override
public double totalPrice(){
    return price;
}

@Override
public Currency getCurrency() {
    return currency;
}
```

The method totalPrice is used to know the price of the book we are referring to.

Then the method getCurrency is a Getter too.

3. Catalog

```java
public Catalog(){
    super();
    collection = new HashSet<StockInterface>();
    LinkedList<String[]> difBooks = readCatalog("books.xml");
    Date date = new Date();

    for (int i = 0; i < difBooks.size(); i++){
        String[] n = difBooks.get(i);

        try {date = new SimpleDateFormat().parse(n[2]);}
        catch(Exception e){}

        String title = n[0];
        String author = n[1];
        String publicationPlace = n[3];
        long isbn = Long.parseLong(n[4]);
        double price = Double.parseDouble(n[5]);
        Currency currency = Currency.getInstance(n[6]);
        int copies = Integer.valueOf(n[7]);

        Book book = new Book(n[0], n[1], date, n[3], isbn);
        Stock stock = new Stock(book, copies, price, currency);

        collection.add(stock);
    }
}
```

Here we create the Catalog class. The constructor does not take in any parameters as this class has no attributes. Nevertheless, when initializing a catalog, we shall follow the procedure in order to store the information read from the XML file. We also convert the different characteristics that need to be converted into the proper type. We store the information in each book, then add some properties and store them in a stock, and finally we add it to the collection.

4. Shopping cart

```
public class ShoppingCart extends BookCollection implements ShoppingCartInterface{
    protected Catalog catalog;

    public ShoppingCart(Catalog initCatalog){
        catalog = initCatalog;
    }
```

The ShoppingCart class inherits from the abstract class BookCollection and implements the ShoppingCartInterface. It has a Catalog as an attribute, which is initialized in the constructor method.

```
@Override
public String[] booktitles(){
    return catalog.booktitles();
}

@Override
public int numberOfCopies(String booktitle){
    return catalog.numberOfCopies(booktitle);
}
```

Here, we override the methods that belong to the upper classes, because now we can implement them and they return the necessary values.

```
@Override
public void removeCopies(int numberOfCopies, String booktitle){
    for (StockInterface e : collection){
        if(e.getBooktitle().equals(booktitle)){
            if (e.numberOfCopies()- numberOfCopies >= 0){
                e.removeCopies(numberOfCopies);
                catalog.addCopies(numberOfCopies, booktitle);
            }
        }
    }
}
```

In this function, we make it possible to remove copies in the ShoppingCart class. We compare the booktitle with every book in the collection, and when it is found we subtract the wanted quantity when possible of copies and return them to the catalog.

Pablo Fernández
Clara López
S101, P101

```java
@Override
public double totalPrice(){
    double p = 0;

    for(StockInterface e: collection){
        if (e != null){
            int s = e.numberOfCopies();

            for(int i = 0; i<s; i++){
                p += e.totalPrice();
            }
        }
    }
    return p;
}
```

In this function, we loop through the elements in the collection in order to add every price and get the total price.

```java
@Override
public String checkout(){
    Payment payment = Payment.getTheInstance();
    StockInterface stock = getStock(catalog.booktitles()[0]);

    long visaNum = 666;
    String cardHolder = "Roddy";
    Currency c = stock.getCurrency();

    String p = payment.doPayment(visaNum,cardHolder, totalPrice(), c);

    for (StockInterface e: collection){
        if(e.numberOfCopies() > 0){
            e.removeCopies(e.numberOfCopies());
        }
    }
    return p;
}
```

The checkout function makes it possible to effectuate the payment and simulate the order. This is possible through the payment class. We initialize some random data to apply the doPayment method, and finally remove the copies from the stock, as once a user buys them they are no longer available.

Pablo Fernández
Clara López
S101, P101

## Conclusion

In conclusion, this practice by far was the hardest of them all. We did our best to make it work, and we are very proud of what we achieve, even though some weird things keep happening.