Lab exercise 5: Implementing a complex design
24292-Object Oriented Programming

Simón Gasión Medina: 240126
Jordi Polo Tormo : 239980

1.Introduction

Este lab se basa en implementar un sistema de gestión de una página web de comprar libros, el usuario puede interactuar con el programa mediante una interfaz grafica que se proporciona en el enunciado y te permite mover una cantidad de libros de un catálogo a un carrito de compra y pagar por los libros que estén en el carro en un momento determinado, eliminandolos de éste y mostrando un mensaje con el precio conjunto de todos los libros.
Los libros en el catálogo se introducen al programa mediante una función que lee un fichero xml que proporciona datos sobre cada uno de éstos.

Para realizar la práctica hemos implementado las clases 'Book', 'Stock', 'Catalog' y 'ShoppingCart'; las primeras dos clases sirven en conjunto para crear la ficha técnica de un libro con todos sus datos incluyendo el número de copias disponible en una dada lista que puede ser el catálogo general o el carrito de compra, que están definidos por las siguientes dos clases y usan tablas de hash para ordenar instancias de la clase Stock.

También hemos realizado cambios en algunas clases proporcionadas para poder conseguir que el programa funcione, notablemente en las clases BookCollection y BookStore para poder acceder a ellas con instancias, ya que por defecto tomaban por defecto interfaces.

2.Description

Book y Stock
La clase book contiene toda la información del libro ajena al servicio de la tienda en sus atributos: el título, el autor, la fecha y lugar de publicación y su número ISBN. Como en el fichero original toda esta información se da en forma de strings, en la clase 'Catalog' transformamos cada uno de esos strings en el tipo que necesitamos para cada atributo diferente.

Stock por otra parte contiene toda la información del libro referente a la tienda como el precio y el número de copias disponible, además de el libro en si. En todo el lab accederemos a los diferentes libros a través de la clase 'Stock'.

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

    public Book(String t, String auth, Date date, String place, long ISBN) {
        title = t;
        author = auth;
        publicationDate = date;
        publicationPlace = place;
        this.ISBN = ISBN;
    }
    public String getTitle(){
        return this.title;
    }
    public String getAuthor() {
        return this.author;
    }
    public Date getPublicationDate(){
        return this.publicationDate;
    }
    public String getPublicationPlace(){
        return this.publicationPlace;
    }
    public long getISBN(){
        return this.ISBN;
    }
}
```

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

    public Stock(Book b, int c, double p, Currency curr) {
        book = b;
        copies = c;
        price = p;
        currency = curr;
    }
    public Book getBook() {
        return this.book;
    }
    public String getBooktitle() {
        return this.book.getTitle();
    }
    public int numberOfCopies() { //i assume this is a getter since the parameters are empty
        return this.copies;
    }
    public void addCopies(int n) {
        copies += n;
    }
    public void removeCopies(int n) {
        copies -= n;
    }
    public double totalPrice() {
        return (price * copies);
    }
}
```

Catalog

Catalog solo cuenta con un constructor en el que se llama al método ‘readCatalog()’ de su superclase ‘BookCollection’, que sirve para extraer los datos de un fichero, y los transforma para añadirlos a la colección de libros en forma de Stock.

```java
public class Catalog extends BookCollection{

    public Catalog() {
        super();
        LinkedList< String[] > booksRaw = readCatalog("books.xml");
        for ( String[] element : booksRaw ) {
            String title = element[0];
            String author = element[1];
            Date date = new Date();
            try { date = new SimpleDateFormat().parse( element[2] ); }
            catch( Exception e ) {}
            String place = element[3];
            long isbn = Long.parseLong( element[4] );
            double price = Double.parseDouble( element[5] );
            Currency currency = Currency.getInstance( element[6] );
            int copies = Integer.parseInt( element[7] );
            Book currentBook = new Book(title, author, date, place, isbn);
            super.collection.add(new Stock(currentBook, copies,  price, currency));
        }
    }
}
```

ShoppingCart

ShoppingCart también inherita 'BookCollection' porque el carro de compra se trata como un catálogo igual que el de la tienda, el cual en esta clase lo tenemos como atributo porqué necesitamos modificarlo cada vez que queremos añadir una cierta cantidad de un libro concreto a la cesta. Debemos sobreescribir los métodos 'addCopies' y 'removeCopies' de la clase BookCollection debido a que añadir un libro a la cesta supone eliminar uno del catálogo.

ShoppingCart usa el método doPayment de la clase estática Payment para imprimir un mensaje con la suma del precio de cada libro, junto con algunos datos representando una tarjeta de crédito inventada.

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

    public ShoppingCart(Catalog cat) {
        super();
        this.catalog = cat;
        LinkedList< String[] > booksRaw = readCatalog("books.xml");
        for ( String[] element : booksRaw ) {
            String title = element[0];
            String author = element[1];
            Date date = new Date();
            try { date = new SimpleDateFormat().parse( element[2] ); }
            catch( Exception e ) {}
            String place = element[3];
            long isbn = Long.parseLong( element[4] );
            double price = Double.parseDouble( element[5] );
            Currency currency = Currency.getInstance( element[6] );
            int copies = 0; //0 copias de cada libro al inicializarse
            Book currentBook = new Book(title, author, date, place, isbn);
            super.collection.add(new Stock(currentBook, copies,  price, currency));
        }
    }
    @Override
    public void addCopies(int n, String btitle) {
        this.catalog.removeCopies(n, btitle);
        super.addCopies(n, btitle);
    }
    @Override
    public void removeCopies(int n, String btitle) { //como el método removeCopies de BookCollection no comprueba que se su
        this.catalog.addCopies(n, btitle);
        super.addCopies(n, btitle);
    }
```

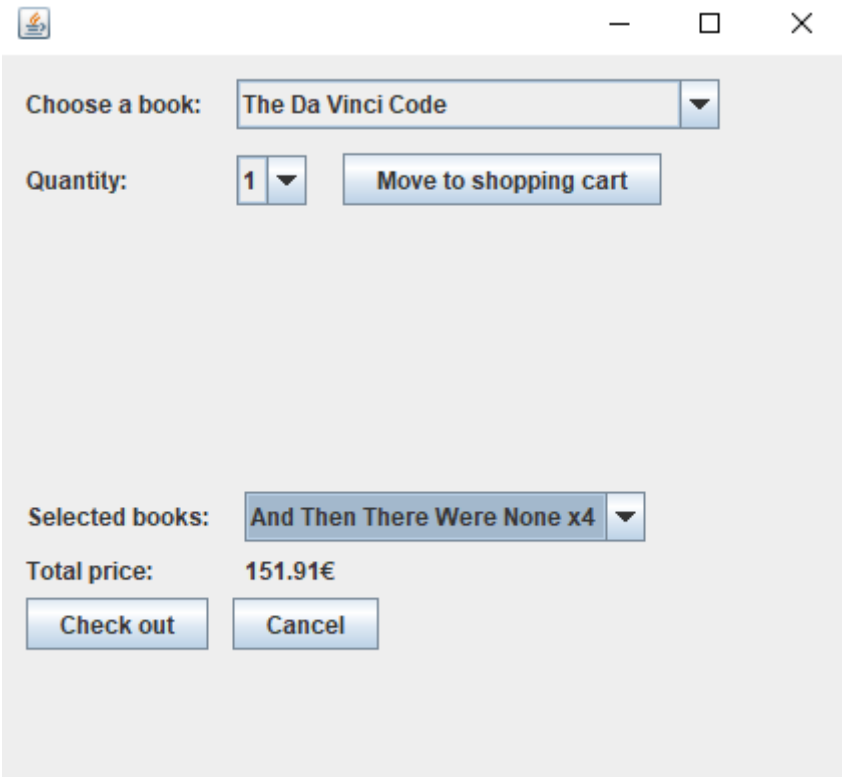
```java
    @Override
    public double totalPrice() {
        double sum = 0;
        Iterator<Stock> it = super.collection.iterator();
        while(it.hasNext()){
            sum += it.next().totalPrice();
        }
        return sum;
    }
    @Override
    public String checkout() {
        long VISA = 1111222233334444l;
        String cardHolder = "GUILLERMO DIAZ IBAÑEZ";
        Currency currency = Currency.getInstance( "EUR" ); //hardcoded since all books in books.xml use the currency EUR
        return Payment.doPayment(VISA, cardHolder, totalPrice(), currency);
    }


}
```

BookCollection, BookStore y TestStore

En las clases ‘BookCollection’ y ‘BookStore’ hemos cambiado todos los casos en los que se usa ‘StockInterface’ por ‘Stock’, además en el constructor de BookStore hemos cambiado los argumentos ‘BookCollectionInterface’ y ‘ShoppingCartInterface’ por ‘BookCollection’ y ‘ShoppingCart’ por tal de poder llamar este constructor en el main method presente en la clase ‘TestStore’, debido a que es imposible instanciar una clase interfaz para pasarla como argumento.

El código en ‘TestStore’ és muy sencillo; solo tenemos que inicializar una instancia de ‘Catalog’ a partir de la cual inicializamos una instancia de ‘ShoppingCart’, y con ambos creamos una instancia de ‘BookStore’ momento a partir del que la interfaz gráfica del lab se encarga de el resto.

```java
package code.bookstore;
public class TestStore {
    Run | Debug
    public static void main(String[] args) {
        Catalog ctlg = new Catalog();
        ShoppingCart spncrt = new ShoppingCart(ctlg);
        BookStore bkstr = new BookStore(ctlg, spncrt);
    }
}
```



3.Conclusión

La práctica ha estado realizada de la manera correcta, el programa se ejecuta y es completamente funcional sin ningún error.

La mayor dificultad que hemos encontrado ha sido el entender cómo interpretar qué es lo que teníamos que hacer. En el enunciado del lab se presentan las interacciones con clases genéricas, las interfaces, pero a la hora de escribir el main nos damos cuenta de que si és imposible instanciar una interficie es imposible también hacer cosas como pasarle los argumentos al constructor de BookStore o almacenar instancias de tipo ‘StockInterface’

debido a que la clase en sí es imposible de instanciar, por lo que tuvimos que hacer unos cambios en las clases proporcionadas.
Aún no estamos seguros de si éste es el método que se esperaba que usáramos o existe alguna manera de hacer funcionar el código con las clases sin alterar, pero compila y funciona correctamente por lo que creemos que la nuestra es una solución correcta.