Aitana González Cárdenas - Álvaro Jiménez Barreno
U166651 - U186356

REPORT PRÀCTICA 5

1. ***An introduction where the problem is described. For example, what should the program do? Which classes do you have to define? Which methods do you have to define? Which methods do you have to implement for these classes?***

En aquesta última pràctica, hem canviat el tema de programació a comparació de les altres pràctiques. Deixem el tema de la construcció i distribució de regions geogràfiques i, passem a crear una aplicació online de compra-venta de llibres. Amb el codi inicial proporcionat i el codi que hem dissenyat i afegit, hagut de ser capaços de:

- Poder emmagatzemar un stock de llibres disponibles (llibres que té la botiga).

- Poder demanar, tants llibres com stock de llibres n'hi hagi, en les compres que volguem. És a dir que, per exemple, hem de poder comprar 2 unitats d'un llibre en una mateixa selecció o en diverses.

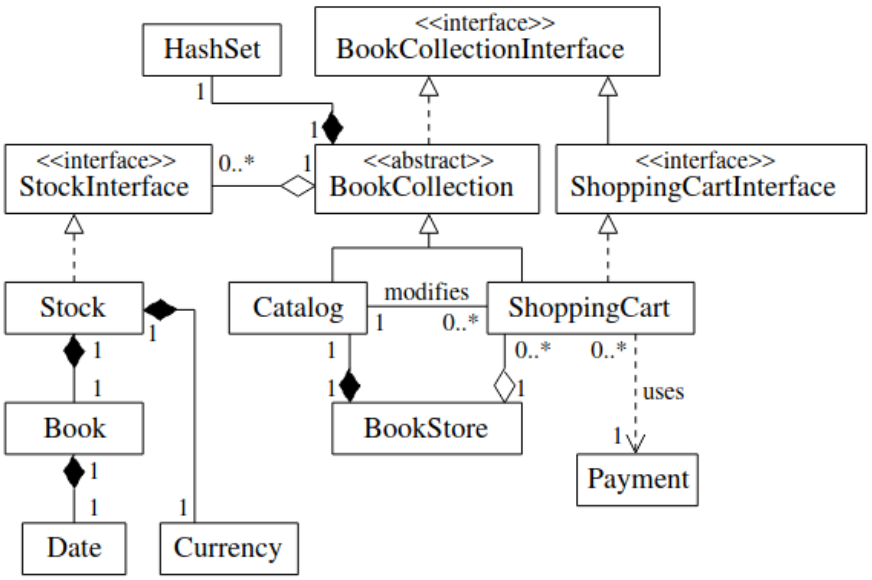
- Que cada llibre es contabilitzi amb el preu establert, sense passar-se del nombre de llibres disponibles.

- Un cop comprat els llibres desitjats, que no torni a aparèixer el nombre de llibres inicials al stock.

Per fer tot el necessari, analitzem el codi i l'enunciat proporcionats.

1. INTRODUCCIÓ

L'esquema inicial del nostre programa és el següent:



Les classes que inicialment ja estaven implementades eren BookCollection, BookCollectionInterface, BookStore, Payment, ShoppingCardInterface i StockInterface.

De forma addicional, les classes Date, Currency i Payment no les hem hagut d'implementar. D'una banda, les classes Date i Currency són classes implementades per Java. D'altra banda, la classe Payment la rebíem feta al codi suplementat inicialment. D'aquesta forma, hem de modificar o crear les classes Book, Stock, StockInterface, Catalog i ShoppingCart. A més, també vam crear la classe TestStore com a main per provar el nostre codi.

1. CREACIÓ DE BOOK

De primera forma, creem les característiques de la classe Book (definició dels atributs). Sabem que una instància d'aquesta classe tindrà un títol, autor, data de publicació, lloc i un ISBN.

```
private String title;
private String author;
private Date publicationDate;
private String publicationPlace;
private long ISBN;
```

La classe Book no tindrà mètodes específics, només els especials: constructor i getters.

- Constructor: Actualitzem tots els valors dels atributs amb el this per inicialitzar la instància de classe Book.

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

- Getters: Retornem el valor dels atributs. Fem un getter per cada atribut.

```
// Obtenim el títol d'un llibre.
public String getTitle() {
    return this.title;
}

// Obtenim l'autor d'un llibre.
public String getAuthor() {
    return this.author;
}
```

```
// Obtenim la data de publicació d'un llibre.
public Date getPublicationDate() {
    return this.publicationDate;
}

// Obtenim el lloc de publicació d'un llibre.
public String getPublicationPlace() {
    return this.publicationPlace;
}

// Obtenim el ISBN d'un llibre.
public long getISBN() {
    return this.ISBN;
}
```

2. CREACIÓ DE STOCK

Un cop feta la classe Book, realitzem la classe Stock. Veiem al disseny que Stock implementa de la interfície StockInterface. Per tant, afegim el implements:

```
public class Stock implements StockInterface {...}
```

Els atribut de la classe seran una instància de classe Book, el nombre de còpies, el preu del llibre i la moneda amb la que es pagarà (Currency).

```
    private Book book;
    protected int copies;
    private double price;
    private Currency currency;
```

Creem aquestes característiques (definició d'atributs) per cada llibre. Els mètodes d'altra banda, seran tots seguint l'override, ja que estan definits a StockInterface però no implementats.

Definim el constructor. Passem per paràmetre els atributs i els actualitzem:

```
public Stock( Book book, int copies, double price, Currency currency ) {
    this.book = book;
    this.copies = copies;
    this.price = price;
    this.currency = currency;
}
```

Necessitem cinc getters, el d'obtenir el llibre, el títol del llibre, el número de còpies, el preu i la currency. Per tant, els quatre getters necessaris retornaran els atributs que volem (copies i book).

```
// Obtenim la instància llibre.
public Book getBook() {
    return this.book;
}

// Obtenim el títol del llibre.
@Override
public String getBooktitle() {
    return this.book.getTitle();
}

// Obtenim el nombre de còpies del llibre.
@Override
public int numberOfCopies() {
    return this.copies;
}
```

```
// Obtenim el preu d'un llibre.
@Override
public double totalPrice() {
    return this.price;
}

// Obtenim la moneda que farem servir per fer el checkout.
@Override
public Currency getCurrency() {
    return this.currency;
}
```

Seguidament, definim tres mètodes addicionals necessaris per realitzar bé la pràctica:

a. **void addCopies (int numberOfCopies):** Creem una funció afegint el nombre de còpies d'un llibre, sumant a l'atribut copies la variable int que li passem per paràmetre.

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

b. **void removeCopies (int numberOfCopies):** Fem el mateix que amb addCopies() però restant-li la variable int que li passem per paràmetre. Creem però, una condició per definir que si el nombre de còpies que volem restar és major a les còpies actuals, el nombre de còpies és igual a 0.

```
@Override
public void removeCopies( int numberOfCopies ) {
    if (numberOfCopies > this.copies){
        this.copies = 0;
    }
    else {
        this.copies -= numberOfCopies;
    }
}
```

3. CODI COMPLEMENTARI DE STOCKINTERFACE

Un cop creades les dues primeres dues classes Book i Stock, definim els dos mètodes getCurrency() i getBook() a la interface perquè els hem creat nous de la classe Stock i necessiten estar definits per realitzar l'override.

```
public Currency getCurrency();
public Book getBook();
```

4. CREACIÓ DE CATALOG

Per poder realitzar la pràctica, és necessari tenir una classe per convertir la informació proporcionada pel fitxer .xml a la informació que necessita el nostre programa per poder utilitzar tots els mètodes. La classe Catalog fa aquest procés. Passem, gràcies al mètode readCatalog (pertanyent a la classe BookCollection) la informació processada pel fitxer en instàncies de classe Book.

Veiem que la classe Catalog hereta de BookCollection, per tant afegim l'extends:

```
public class Catalog extends BookCollection {...}
```

Pels atributs, només definim un catàleg de classe Catalog i visibilitat protected, perquè pugui ser visible tant a la classe com a les seves sublcasses:

```
protected Catalog catalog;
```

L'únic mètode que necessitem, llavors, serà el del constructor, que convertirà la informació del fitxer en instàncies de Book i les acabarà agrupant en una instància de tipus collection.

Cridem al super() per cridar al constructor de la classe que hereta. Creem una Linkedlist de books, que serà on afegirem totes les instàncies que llegirem amb la funció readCatalog("books.xml").

Hi ha informació que s'ha de convertir abans de fer-la servir. Per això, La "publication date" ha de ser representada com una instància de la classe Data, ISBN com un Long Int, preu com a Double, currency com a Currency i còpies com a Int. Un cop feta la conversió, guardem les característiques en una variable book (de classe Book) que serà inclosa amb la resta d'atributs a un stock. Aquest stock s'inclourà a una variable collection (una HashSet de tipus StockInterface). Com que volem que ho faci per cada llibre pertanyent a l'arxiu .xml, l'incloem en un for in que iterarà des de 0 fins a la mida de la Linked List de books.

```
public Catalog() {

    // Crida al constructor de la classe que hereda
    super();
    collection = new HashSet< StockInterface >();
    LinkedList<String[]> books = new LinkedList<String[]>();
    Date date = new Date();
    books = readCatalog("books.xml");

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

        // Date instance
        try { date = new SimpleDateFormat().parse( book_i[2] ); }
        catch( Exception e ) {}
        // Convert to long
        long isbn = Long.parseLong( book_i[4] );
        // Convert to double
        double price = Double.parseDouble( book_i[5] );
        // Currency instance
        Currency currency = Currency.getInstance( book_i[6] );
        // Convert to int
        int copies = Integer.parseInt( book_i[7] );
        // Es crea el llibre amb els seus paràmetres.
        Book book = new Book(book_i[0], book_i[1], date, book_i[3], isbn );
        // Es crea un stock amb les característiques del llibre que no hem pogut incloure en el llibre.
        Stock stock = new Stock(book, copies, price, currency);
        collection.add(stock);
    }
}
```

5. CREACIÓ DE SHOPPINGCART

Com a definició de classes, hem definit una classe més, la classe ShoppingCart, que hereda de BookCollection i implementa ShoppingCartInterface:

```
public class ShoppingCart extends BookCollection implements ShoppingCartInterface
{...}
```

En aquesta classe també hem declarat l'atribut de catàleg com un protected. El constructor, d'altra banda, només actualitza el catàleg.
No implementem el super() de l'atribut de la seva superclasse ja que, com es protected, el podem manipular directament.

```
public ShoppingCart( Catalog catalog ) {
    this.catalog = catalog;
}
```

Hem realitzat un Override dels getters de booktitles() i numberOfCopies(), així com dels mètodes addCopies() i removeCopies(), tots quatre declarats a la classe abstracta BookCollection. D'altra banda, també hem fet Override dels mètodes totalPrice() i checkout(), definits a la interface ShoppingCartInterface.

- **Getters:**

Pel booktitles, retornem el valor dels booktitles() del catàleg.

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

Pel numberOfCopies, retornem el valor del nombre de còpies d'un booktitle del catàleg.

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

- **Addició/eliminació de còpies**

Pel mètode addCopies(), comprovem que sigui possible afegir nombre de còpies, comprovant que la resta del nombre de còpies al catàleg menys la del nombre de còpies que volem afegir sigui major o igual a 0. En cas afirmatiu. restem el nombre de còpies al catàleg (que implementarem al nostre stock de compra). Comprovem que element es troba en la col·lecció, iterant-la amb un for.

Per cada element, obtenim el stock del llibre [getStock(booktitle)] Si està, l'afegim amb la funció addCopies() directament, sinó, fem el procés manual, creant instàncies Book, currency i price; després les afegim a una variable de classe Stock que s'afegirà a la col·lecció.

```java
@Override
public void addCopies(int numberOfCopies, String booktitle) {
    if (catalog.numberOfCopies(booktitle)-numberOfCopies >= 0 ){
        catalog.removeCopies(numberOfCopies, booktitle);
        for(StockInterface element: catalog.collection){
            if(element.getBooktitle().equals(booktitle)){
                StockInterface stock_true_false = getStock(booktitle);
                if (stock_true_false == null){
                    Book book = element.getBook();
                    Currency currency = element.getCurrency();
                    double price = element.totalPrice();
                    Stock stock = new Stock(book,numberOfCopies, price, currency);
                    collection.add(stock);
                }
                else {
                    stock_true_false.addCopies(numberOfCopies);
                }
            }
        }
    }
}
```

Pel mètode removeCopies() hem passat per paràmetres el nombre de còpies que volem eliminar i el nom del llibre. Creem un bucle for in per iterar per tots els elements de la HashSet. La primera condició la incloem per determinar que l'element que estem analitzant (en aquest cas un llibre) equival al nom del llibre (booktitle). Si és afirmatiu, fem un altre condició per comprovar si el nombre de còpies que tenim menys el nombre de còpies que volem restar és major a 0. Si els les dues condicions es compleixen, eliminem el nombre de còpies de la col·lecció (la teva llista de compra) i afegim les còpies al catàleg.

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

- **Preu total / Checkout**

Pel mètode del preu total creem una variable que acumularà el preu total i iterem amb un bucle dins la nostra col·lecció de llibres. Per cada llibre, tornem a fer un bucle pel nombre de còpies d'un llibre. La variable anirà sumant el preu anterior més el totalPrice() de l'element en qüestió fins iterar per totes les còpies disponibles de tots els llibres de la col·lecció.

Pel mètode del checkout(), creem una llista i stock amb els noms dels llibres. Això ens servirà per poder utilitzar la funció getCurrency(). Seguidament, creem una instància currency donant-li un valor, perquè sempre tingui una moneda de pagament. Aquest valor per defecte canviarà sempre que el stock no sigui null amb la funció stock.getCurrency(). Definim les variables price amb el preu total, user amb la funció Payment.getTheInstance(), un número de VISA i el nom del propietari de la targeta. Finalment creem la instància payment seguint la funció user.doPayment amb el nombre de VISA, el propietari de la targeta, el preu total i la currency com a atributs i retornem la variable payment.

```java
@Override
public String checkout(){

    String[] bookTitles = catalog.booktitles();
    StockInterface stock = getStock( bookTitles[0] );

    // Creem una instància inicial per fixar una Currency.
    // D'aquesta forma, currency sempre tindrà un valor.
    Currency currency = Currency.getInstance(Locale.GERMANY);

    /* La instància currency canviarà sempre i quan stock no sigui NULL,
    agafant el valor de l'atribut currency de la classe. */
    if(stock != null) {
        currency = stock.getCurrency();
    }

    double price = totalPrice();

    Payment user = Payment.getTheInstance();
    long VISANumber = 340779810;
    String cardHolder = "Patricia ;)";
    String payment = user.doPayment( VISANumber, cardHolder, price, currency );
    return payment;
}
```

6. CREACIÓ DE TESTSTORE

Un cop definides i programades totes les classes, creem el main per comprovar la funcionalitat del nostre programa. Com la llista de llibres està emmagatzemada en el fitxer books.xml i els mètodes de conversió del fitxer al codi ja estan automatitzats, creem una instància de classe Catalog, seguidament creem una instància ShoppingCart, passant com a paràmetre catalog i, finalment, creem una instància de classe BookStore, amb catalog i cart com a paràmetres.

```java
public class TestStore {
    Run | Debug
    public static void main(String[] args){

        // Creem instàncies de Catalog, ShoppingCard i BookStore pel correcte funcionament de  la nostra botiga.
        Catalog catalog = new Catalog();
        ShoppingCart cart = new ShoppingCart( catalog );
        BookStore store = new BookStore( catalog, cart );


    }
}
```

2. ***A description of possible alternative solutions that were discussed, and a description of the chosen solution and the reason for choosing this solution rather than others. It is also a good idea to mention the related theoretical concepts of object-oriented programming that were applied as part of the solution.***

Inicialment el programa ens funcionava bé amb els mètodes implementats. Tot i així, en el mètode de Checkout volíem millorar-ho, ja que quan no compràvem cap llibre i provàvem de fer el Check Out (o quan compràvem totes còpies disponibles d'un o més llibres), no ens acabava de respondre el programa. Com a solució alternativa, vam decidir definir una variable general pel currency. Vam establir la currency d'Alemanya, ja que és la mateixa que la d'Espanya (EURO). D'aquesta forma, si no escollíem cap llibre, o els escollíem tots, podíem pagar igualment (o 0€ o el màxim de llibres disponibles). Aquesta variable però, canviaria sempre que stock no fos null amb el valor desitjat (getCurrency()).

D'altra banda, també volíem fer un generador de nombres aleatoris per al número de la credit card, per fer un codi més professional i una mica més automatizat. Vam estar investigant i vam veure que perquè el codi per generar de forma aleatòria un número de targeta fos correcte, s'havia de seguir el Luhn Algorithm (conegut com a algoritme "mòdul 10" o "mod 10"). Tot i així, vam veure que per falta de temps i necessitat d'enfocar-nos en altres assignatures de la universitat, vam pensar que no calia complicar tant el codi, també perquè no era l'objectiu principal de la pràctica.

Finalment, com a conceptes teòrics, veiem que en aquest laboratori utilitzem el concepte d'interface, utilitzant les característiques i implementacions necessàries. Amb interface les classes no s'hereten, sinó que s'implementen. D'altra banda, veiem que, com una interface és semblant a una classe abstracta pura, però no és una classe, els mètodes i atributs que definim (però no implementem) no necessiten de la paraula **abstract**, ja que per defecte ho són. Tot i així, però, les classes que implementen una interface necessiten fer override de tots els mètodes definits a la interface per poder ser visibles i no classes abstractes.

3. ***A conclusion that describes how well the solution worked in practice, i.e. did the tests show that the classes vere correctly implemented? You can also mention any difficulties during the implementation as well as any doubts you might have had.***

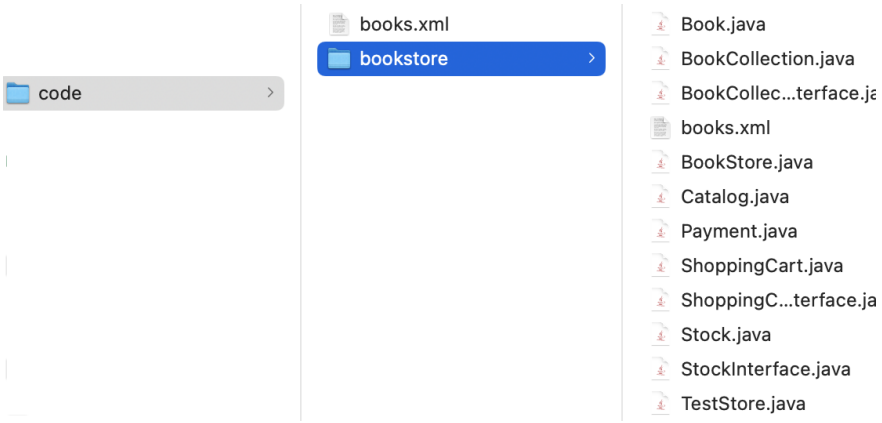
Com a conclusions generals de la pràctica, tot i que en un principi ens ha costat més enfrontar-nos a aquesta pel fet no teníem gaire clar com implementar els diferents mètodes de la classe ShoppingCard, ja que, no sabíem ben bé com treballar amb els diferents catalogs, creiem que ens n'hem sortit força bé, tot i haver realitzat la pràctica amb poc temps i en època d'exàmens.

D'altra banda, aquesta pràctica ens ha ajudat a ser més solvents a l'hora de treballar amb el llenguatge de programació de java. A més, la redacció d'un informe després de cada pràctica ens ajuda a assimilar tant els continguts teòrics com els continguts pràctics realitzats a les diferents classes de l'assignatura.

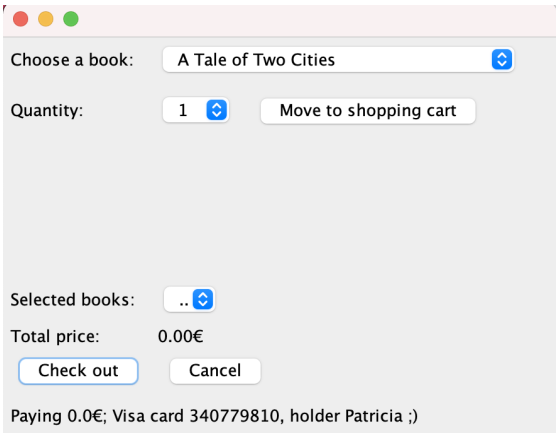
No obstant, volem destacar un parell d'inconvenients que s'han generat al llarg de la realització de la pràctica que podem dir que hem resolt.

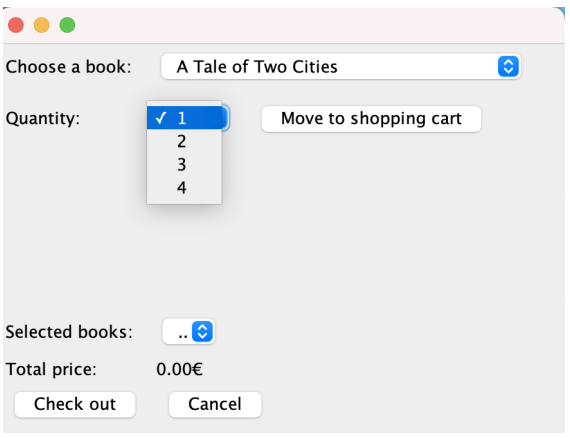
Principalment, l'error que ens va donar més problemes va ser la lectura del fitxer books.xml en la classe de Catalog. De de l'inici ens posava: *No such file or directory* (tal i com mostra la primera foto a continuació). Això era degut a que, el teníem dins de la carpeta del codi (del package bookstore) però no fora d'aquesta, i per tant, no trobava l'arxiu. Per solucionar-ho, vam fer una còpia de l'arxiu i la vam posar fora del package, tal i com mostrem a continuació (a la segona foto).

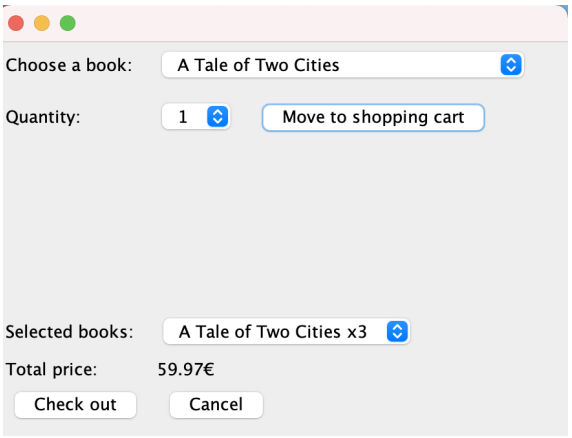
```
java.io.FileNotFoundException: /Users/aitanagonzalezcardenas/Downloads/code/books.xml (No such file or directory)
```

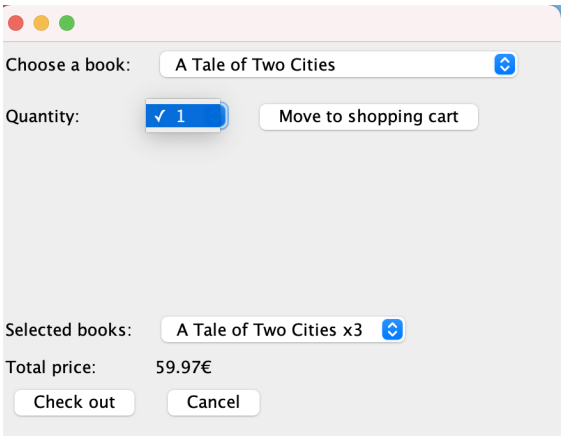


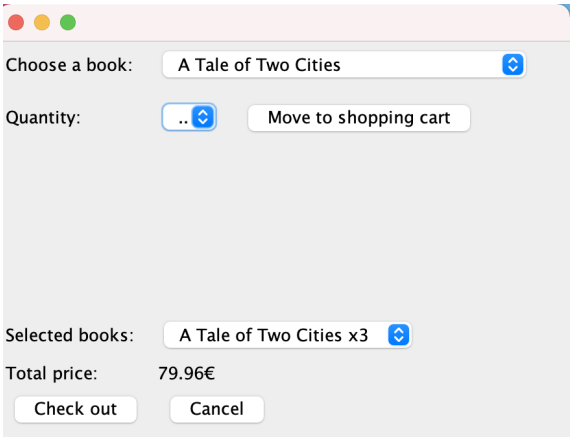
Altrament, vam tenir dificultats a l'hora de fer funcionar correctament la funció de quantity de la interfície gràfica que ens apareix a l'hora de fer run al programa. Inicialment, en les funcions de addCopies() i removeCopies() vam sumar i restar 1, respectivament, per afegir o eliminar còpies d'un llibre. Per aquest motiu, ens donava problemes, ja que quan afegies en la teva cistella de la compra una quantitat major que 1 de còpies d'un llibre, només eliminava 1 del catàleg. Aquest inconvenient el vam poder solucionar mitjançant el canvi de 1 i -1 per numberOfCopies. D'aquesta manera, sí que eliminava o afegia el nombre de còpies que sol·licitaves / rebutjaves.

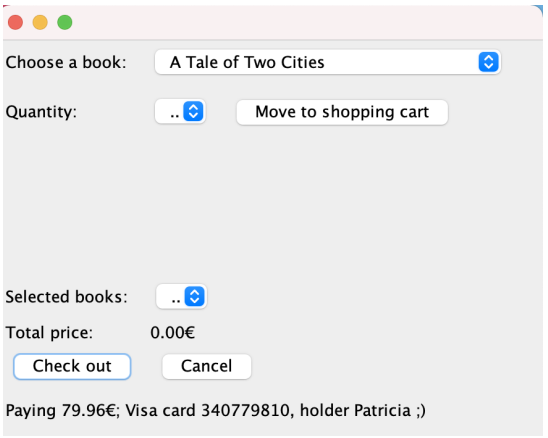





Choose a book:
A Tale of Two Cities
Quantity:
1
Move to shopping cart
Selected books:
A Tale of Two Cities x3
Total price:
59.97€
Check out
Cancel


Choose a book:
A Tale of Two Cities
Quantity:
✓ 1
Move to shopping cart
Selected books:
A Tale of Two Cities x3
Total price:
59.97€
Check out
Cancel


Choose a book:
A Tale of Two Cities
Quantity:
..
Move to shopping cart
Selected books:
A Tale of Two Cities x3
Total price:
79.96€
Check out
Cancel


Choose a book:
A Tale of Two Cities
Quantity:
..
Move to shopping cart
Selected books:
..
Total price:
0.00€
Check out
Cancel
Paying 79.96€; Visa card 340779810, holder Patricia ;)