Eric Navarro
Armand Sandiu

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

The aim of this lab session was to implement the design of a bookstore application from seminar 5. Our program had to read an xml file that contained information about books in the store.
The class book collection contained a method readCatalog that read information about the books in the store from the xml file books. xml. This method returned a listed of arrays of strings. Each array of strings containing information about a single book. Each piece of information was stored in a particular index of the array:

0. Title.
1. Author.
2. Date of publication.
3. Placed 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 by stock.
It was necessary to iterate over the lists returned by the method readCatalog to access each array of information about a book.

Some of the information had to be converted to use it (For example, the publication date should have been represented as an instance of the date class, the isbn number as a long integer (long), etc…
The class bookstore represents 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. To simplify the code, bookstore only contained an attribute representing a single shopping cart, whereas in the real application bookstore would interact with multiple users at once.

The class bookcollection represented a collection of books, and was inherited by the classes catalog and shoppingcart. Instead of representing the number of copies as an attribute of the class book, there was a separate class stock that stored information about a book which was specific to the store (such as the number of copies, the price and the currency).

The information about a book itself was accessed via the class stock.

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

    public Stock(Book bookinit, int copinit, real priceinit, Currency curinit){
        this.book = bookinit;
        this.copies = copinit;
        this.price = priceinit;
        this.currency= curinit;
    }
    public Book getBook(){
        return this.book;
    }
    public String getBooktitle(){
        return this.book.title;
    }
    public Int numberOfCopies(){
        return this.copies;
    }
    public void addCopies(int numberOfCopies){
        StockInterface.addCopies(numberOfCopies, this.book.title);
    }
    public void removeCopies(int numberOfCopies){
        StockInterface.removeCopies(numberOfCopies, this.book.title);
    }
    public real totalPrice(){
        return this.price;
    }

}
```

The class bookcollection included existing methods for adding or removing copies of a book from the collection. These methods assumed that there exists an instance of stock in the collection whose associated book had the given title.To correctly initialize its collection, the class catalog processed the information parsed using the method readcatalog of bookcollection.

```
public static LinkedList< String[] > readCatalog( String filename ) {
    LinkedList< String[] > books = new LinkedList< String[] >();
    String[] tokens = { "title", "author", "date", "place",
                        "isbn", "price", "currency", "copies" };
    try {
        File input = new File( filename );
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse( input );
        doc.getDocumentElement().normalize();
        NodeList list = doc.getElementsByTagName( "book" );
        for ( int i = 0; i < list.getLength(); ++i ) {
            Element element = (Element)list.item( i );
            String[] book = new String[tokens.length];
            for ( int j = 0; j < tokens.length; ++j ) {
                NodeList nl = element.getElementsByTagName( tokens[j] );
                Node node = nl.item( 0 ).getChildNodes().item( 0 );
                book[j] = node.getNodeValue();
            }
            books.add( book );
        }
    } catch ( Exception e ) {
        e.printStackTrace();
    }
    return books;
```

Eric Navarro
Armand Sandiu

The class shoppingcart had to it an instance added 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 had not selected any copy). When the user made an order or canceled an order, an appropriate number of copies of a book should be moved from the catalog to the shopping cart or vice versa.

```
public class ShoppingCart extends BookCollection implements ShoppingCartInterface{
    private Catalog catalog;
    public ShoppingCart(Catalog cinit){
        super();
        catalog = cinit;
        LinkedList<String[]> list = new LinkedList<String[]>();
        list = readCatalog("Lab-5/books.xml");
        Date date = new Date();
        for (String[] element : list){
            String[] book = element;
            String title = book[0];
            String author = book[1];
            String dateString = book[2];
            try { date = new SimpleDateFormat().parse( dateString ); }
            catch( Exception e ) {}
            String place = book[3];
            String isbnString = book[4];
            long isbn = Long.parseLong( isbnString );
            String priceString = book[5];
            double price = Double.parseDouble( priceString );
            String currencyString = book[6];
            Currency currency = Currency.getInstance( currencyString );
            Book book1 = new Book(title, author, date, place, isbn);
            Stock s1 = new Stock(book1, 0, price, currency);
            collection.add(s1);
        }
    }
```

The class shoppingcart also needed a method checkout that caused the user to pay for the books in the shopping cart. We could can invent user information in the method checkout in order to call the method doPayment of the class Payment.

```
70      public String checkout(){
71          double price = totalPrice();
72          Currency curr = Currency.getInstance("€");
73          Payment p = new Payment();
74          return p.doPayment(69420, "Ernesto", price, curr);
```