Assignment 5
Bookstore Application 3
CS190: Java
out: 25 February 2008
due: 8 March 2008

1 Description

We will be designing, implementing and testing a bookstore application in the next several homework assignments. Each assignment will build on the previous. It will be to your advantage to complete each assignment and do so on-time.

2 Implementation Instructions

a. Immutability

If something is immutable, it means cannot be changed. Java can create immutable data with the `final` keyword, but that is only useful if you know what the data is ahead of time. Consider books and our `Book` class. Once a book is created, most of its properties aren’t really allowed to change. Books in the real world don’t (usually) change their titles or ISBN numbers. However there are some properties, such as price that can and will change frequently.

To create a safer and more consistent `Inventory` system, we are going to make some properties of the `Merchandise` immutable. To accomplish this:

- ensure the appropriate data members (SKU, name) are private
- change their respective setter methods to be protected
- create a new constructor for `Merchandise` that sets the SKU and name when the object is created
- make the default constructor inaccessible

This will certainly affect classes that inherit from `Merchandise`. You will need to edit these classes also in order to conform to the new access specifications of `Merchandise`. Also, you should change the appropriate properties of classes further down the inheritance tree to be immutable also. When you are done, your constructors should look like this:

```java
Merchandise(int SKU, String name)
ReadableMerchandise(int SKU, String title, String editor)
Book(int SKU, String title, String editor)
Magazine(int SKU, String title, String editor, Date issueDate)
```
b. Exceptions

We are going to modify our `buy` and `returnMerchandise` methods once again, this time to throw exceptions for certain exceptional cases.

The new `buy` method should look like this:

```java
public void buy( int sku ) throws MerchandiseNotFoundException
```

If the supplied `sku` does not match any of the current Merchandise in the Inventory, then the `buy` method should throw a `MerchandiseNotFoundException`.

The new `returnMerchandise` method should look like this:

```java
public void returnMerchandise( int sku ) throws 
MerchandiseNotFoundException, 
MerchandiseNotReturnableException { ... }
```

Note the `returnMerchandise` method no longer returns a boolean. Instead we will throw a `MerchandiseNotFoundException` if the `sku` of the Merchandise being returned is not in our Inventory. Also we are adding a `MerchandiseNotReturnableException`. This will be used later.

c. Downloadable interface

As mentioned in the previous assignment, the owners of the Bookstore would like to sell digital merchandise over the Internet. For now they will just sell books, but perhaps they will see music or videos in the future. In order to be flexible, we will create a `Downloadable` interface that has the following methods:

```java
unsigned int getBytes()
```

Create a `ElectronicBook` class that extends `Book` and implements `Downloadable`. `ElectronicBook` should add the following properties:

```java
unsigned int fileSize
```

The `fileSize` property should also be immutable, and set via the constructor.

`ElectronicBook` should only implement the following methods:

```java
unsigned int getBytes()
```

Objects that are `Downloadable` must be handled differently in your `Inventory`. Basically, the number of available items is irrelevant since we can make infinite identical copies. Therefore if the Merchandise is `Downloadable` (hint: `instanceof`) your `buy` method should ignore the available count.

Also `Downloadable` objects are not returnable. If a `Downloadable` object is returned, the `returnMerchandise` should throw a `MerchandiseNotReturnableException`. 
3 Extra Credit

a. Additional Merchandise

Create at least two new classes that represent other merchandise that can be added to the Inventory. These can be anything - coffee, calendars, sailboats - or anything else you would like to sell at your bookstore.

Also since the kinds of Merchandise is growing, it would be helpful if each Merchandise knew how to print itself. Java's base Object class has a toString() method. If we want to print out our own information, we can override this method. For example, to toString() method of Merchandise could something look like this:

```java
public String toString() {
    return "SKU: " + this.SKU + " name:" + name + " price:" + price;
}
```

Add toString() methods for all existing Merchandise and also the new ones you are adding. Classes should always include the information from their parent class (hint: super). For example, the toString() method of Magazine should call the toString() method of its parent, ReadableMerchandise, and then append its issue date and return the whole string.

Next, modify the printInventory() method of your Inventory class print the Merchandise objects directly, using their toString() methods. The printInventory() method should still print the available count of each item also.

Finally if you choose to do the extra credit, please note it in your write-up when you submit your assignment.

4 Suggestions & Submission Details

You may wish to implement a testing procedure to evaluate the correctness of your implementation. Your code will be tested using a test harness to ensure its correctness. In addition to submitting the .java files for all of the object implementations, please submit a brief write-up (pdf or plain-text files only) describing any difficulties you have encountered or any concerns you may have.