COMP 132: Advanced Programming
Spring 2016

Quiz 2, February 16, 2016

This quiz problem builds on Homework 2, which is included below for your convenience. The solutions to the homework problems are also given to you. Remember the class hierarchy you built there. **In this quiz, you are asked to write**

- **a subclass of ShoppingCart called Purchase, and**
- **a class called Customer, and**
- **a class Test with a main method that constructs a Customer and calls its methods.**

The **Purchase** class is explained in detail below. (35 pts)

- **Purchase** object represents a purchased **ShoppingCart**
- It stores data regarding the **purchase date**, **expected shipping date** and **length of the expected delivery**.
- Total cost of a **Purchase** can be calculated using costs of each book in the **ShoppingCart**.
- The details of the purchase (i.e. the total number, price and cost of the books, purchase date, expected shipping date and length of delivery) can be printed out using **toString()** method

A **Purchase** object stores the following additional fields:

- **purchaseDate**: a string (you can assume format to be dd/mm/yyyy eg. 16/02/2016)
  Note: You don't have to check the format!
- **shippingDate**: a string (you can assume format to be dd/mm/yyyy eg. 16/02/2016)
  Note: You don't have to check the format!
- **deliveryLength**: an int (estimated delivery length in days)
  Note: The value should be non-negative

and also the following additional methods:

- You should have a proper **constructor** that takes in three arguments corresponding to the fields.
- **getter/setter** methods for each field with appropriate checks
- **public double getTotalPayment()**: a method that calculates the total payment of the Purchase calculated by summing cost of each book
- **toString()**: a method that uses the toString() method of ShoppingCart (i.e. the total number, price and cost of the books). The returned String should also contain the purchase date, shipping date and estimated delivery length.

The **Customer** class is explained in detail below. Create the **Customer** class in the **shopping** package. (45 pts)

- **Customer** object represents a user of an online store
- It stores data regarding the user name, user surname, an array of purchases and the number of purchases made.
- Customers will be able to place a purchase which will be added to the purchase history
- **Customer** object will store up to 2 purchase entries. After a customer reached the limit, store will not accept further purchases. (assume store owners don't know how to do a sale ☺)

A **Customer** object represents a user of an online store and stores the following fields:

- **userName**: a String
- **userSurname**: a String
- **purchaseHistory**: a Purchase array (representing all purchases placed by the user)
- **purchaseCount**: an int

with following methods:

- A **constructor** with two arguments, corresponding to first two fields.
- **getter/setter** methods for all non-array fields
- **public boolean placePurchase(Purchase current)** : a method that places a ShoppingCart object as a new Purchase to users purchaseHistory and increments the purchaseCount
  Note: Customer object can only have **2 Purchase**s in **purchaseHistory**, so your method should take this limit into account and return **false** if placing purchase is failed.
- **public double getTotalAmountSpent()**: a method that calculates total money amount spent by the user using getTotalPayment for each Purchase in purchaseHistory.
  Note: You should use **getContents** method in **ShoppingCart** for accessing the book array. And you should use only the actual price for each book.
- **toString()**: a method for the cart that uses the toString() method of the entries in the contents array. The returned String should also contain the total number, total amount spent and cost of each books in the previous purchases.

The **Test** class is explained in detail below. To test your code modify the existing Test class provided to you. (20 pts)

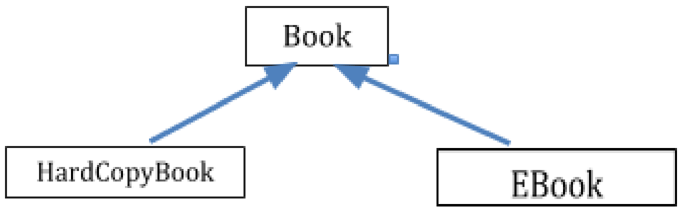
- Create a Customer object for yourself with your name and surname.
- Create 4 hard copy books with following names and prices
  - “Java How to Program”, 25 TRY
  - “C How to Program”, 20 TRY
  - “C++ How to Program”, 25 TRY
  - “The Java Programming Language”, 30 TRY
- Place a purchase with the 4 items above with your customer object with appropriate days (i.e. 01/02/2016 etc.)
- Print your customer object’s purchase history
- Create 3 e-books with following values for name, price, file size, encoding format and number of devices allowed
  - “Learn you some Erlang”, 75 TRY, 200, pdf, 1
  - “Principles of Programming Languages” , 125 TRY, 150, chm, 3
  - “Advanced Operating Systems”, 50 TRY, 250, chm, 2
- Place a purchase with the 3 e-books above with your customer object with appropriate days
- Print your customer object’s purchase history
- Create 1 book with following name and price
  - “Cloudy With a Chance of Meatballs”, 20 TRY
- Try to place a purchase with the book above if it fails print a warning like below
  - “We cannot process your purchase for now! Please try again later.”

COMP 132: Advanced Programming
Spring 2016

Homework 2

Imagine an online book store where you can buy hard copy books, or e-books. The store also sells books from other suppliers.

Represent the books that can be bought in this store using the following class hierarchy:



Books sold from other suppliers are represented by the Book class.

Book

➔ Each Book object stores the following information:
   - name: A String
   - price: A double
   - bookId: that is an ISBN number, and a String consisting of exactly 10 digits

➔ Each Book object should have the following methods:
   - It has getter and setter methods for each field. The setter methods make sure that the fields have reasonable values, e.g., the price is non-negative or the itemID is of the required format. Otherwise, the setters set the fields to some default value.
   - A constructor with three arguments, corresponding to the three fields.
   - A method public void applyDiscount(double discountPercentage) that reduces the book price by discountPercentage percent.
   - A toString() method

HardCopyBook

➔ Each HardCopyBook is a special kind of Book object that has the following additional fields:
   - weight: a double
   - shippingCost: a double

- ➔ HardCopyBook objects should also have proper getters and setters, and they should override Item's toString method so that the return value includes weight and shippingCost information as well as other information in the Book class. The HardCopyBook constructor should take five arguments corresponding to the five fields of this class and its superclass.

- ➔ HardCopyBook objects have an additional method
  - public double getTotalCost() which returns the cost of buying and shipping the item.

EBook

- ➔ EBook objects represents e-books, and have the following additional fields:
  - fileSize: a long, the number of bytes of the file representing this book
  - encodingFormat: A String such as "pdf", "chm", "djvu", etc. representing what format the file is encoded in.
  - numDevicesAllowed: an int, the maximum number of electronic devices that the customer can have copies of this e-book on.
  - numDevicesBeingUsed: an int representing the number of devices that the customer already has copies of this e-book on. The default is 0.

- ➔ EBook objects should also have a proper constructor that takes in six arguments corresponding to the fields. The EBook class should have getters and setters for all fields. Setters should do reasonable validity checking. EBook objects should have methods
  - public boolean addDevice()
  - epublic boolean removeDevice()

  that, when they return "true", add or remove a device from the set of devices on which copies of the item reside. When addition or removal fails, these methods return "false."

- ➔ Write a class called ShoppingCart that represents a shopping cart with at most 10 entries.
  - Use a Book array of 10 entries as one of ShoppingCart's fields. Call it contents. contents can store Book, HardCopyBook, or EBook objects.
  - Use an integer field, numEntriesInCart (<=10) that represents the number of books already in the cart.
  - Write getter and setter methods for all non-array fields.
  - Write a method public boolean addBook(Book book) that, if the cart has fewer than 10 entries, adds book to the shopping cart, and increases numEntriesInCart by 1. This method should return "true" if the addition is successful, "false" if the cart was full.
  - Write a method public boolean removeLastBook() that removes the last book added to the cart and returns "true" if there is such an item. This method returns "false" if the cart is empty.
  - Write a "toString()" method for the cart that uses the toString() method of the entries in the contents array. The returned String should also contain the total number, price and cost of the books in the shopping cart.

➔ Write a Test class that creates a ShoppingCart object, fills it with 7 objects (a mix of Book, HardCopyBook and EBook objects) and then calls the toString method of the ShoppingCart object.