Metropolitan State University
ICS 240 - 50: Introduction to Data Structures
Summer 2022

Homework 05: Inheritance & Polymorphism
Total points: 40
Out: Tuesday July 12, 2022
Due: 11:59 PM on Tuesday, July 26, 2022
Last day to submit with penalty: 11:59 PM on Friday July 29, 2022

Problem Description

In this assignment, you will use inheritance and polymorphism to implement a Java application, called `BookStoreApplication`, to be used by a book store that sells books and CDs. Each item that is sold by the store has the following attributes: `code`, `price`, and `quantity` and items are categorized into three categories where each category has another set of identifying attributes as explained below.

Implementation

You will implement 7 classes as explained below.

`class SalesItem`

`SalesItem` is an **abstract** class that is defined by the following **private** instance variables:

- `code` (type `int`)
- `price` (type `double`)
- `quantity` (type `int`).

And the following methods:

- A constructor that takes three input parameters to initialize the three instance variables. The inputs to the constructor must be in the same order as listed above.
- Getters and setters for all the instance variables.
- `itemTotalCost`: a method that returns the total cost of that item where the total cost is equal to price*quantity.
- `monthlyPromo`: an abstract method that is used to change the price of an item by applying a monthly discount to the item where the discount is dependent on the type of the item.
- `toString`: a method that returns a String representation of the sales item.

class Book

`Book` is a **subclass** of `SalesItem` such that, in addition to the `SalesItem`'s instance variables, a `Book` is also defined by the following instance variables:

- `author` (type `String`)
- `numPages` (type `int`)

And the following methods:

- A constructor that takes five input parameters to initialize all the instance variables. The inputs to the constructor must be in the same order as listed above and `SalesItem`'s parameters comes before `Book` parameters.
- Setter and getter methods for the book's additional instance variables.
- An implementation of the `monthlyPromo` method where, for a book, a price reduction of 3% is applied each month.
- Override the `toString` method from the `SalesItem` class to add the book's specific instance variables to the output string.

class Dictionary

`Dictionary` is a **subclass** of `Book` and it extends the `Book` class by adding the following instance variables:

- `language` (type `String`)
- `numDefinitions` (type `int`)

And the following methods:

- A constructor to initialize all the dictionary's instance variables. The inputs to the constructor must be in the same order as listed above and `SalesItem` parameters comes before `Dictionary` parameters.
- Setter and getter methods for the additional instance variables.
- Override the `toString` method from the `Book` class to include the additional instance variables in the output.
- A method called `getRatio` that returns the number of definitions (on average) for each page in the dictionary (i.e., number of definitions divided by the number of pages).

class AudioCD

The `AudioCD` is a **subclass** of `SalesItem` by adding the following additional Instance variables:

- `label` (type `String`)
- `playingTime` (type `int`)

And the following methods:

- A constructor to initialize all the instance variables. The inputs to the constructor must be in the same order as listed above and `SalesItem` parameters comes before `Book` parameters.

- Setter and getter methods for label and playing time.
- An implementation of the `monthlyPromo` method where, for a CD, a price reduction of 5% is applied each month.
- Override the `toString` methods to include the audio cd's specific instance variables.

`class Cart`

`Cart` is basically a collection class of `SalesItem`s that is defined by the following two instance variables:

- `itemsList`: an array of `SalesItem`
- `numItems`: the number of items on the cart.

And the following methods:

- `Cart`'s constructor that takes as input an integer parameter that determines the maximum number of items that can be placed in the cart. The constructor then uses this number to create the `itemsList` array.
- `addItem`: a method that takes one input parameter of type `SalesItem` as input and adds it to the `itemsList` array. If the cart is full, double the size of the cart. Hint: check ensureCapacity method from `IntArrayBag` in `BagCollectionClasses` that is posted on D2L.
- `capacity`: a method to return the capacity of the cart where the capacity is the maximum number of items that can be put in the cart.
- `size`: a method to return the number of items in the cart.
- `cartTotalCost`: a method that returns the total cost of all items in the cart.
- `getAtIndex`: a method that takes an input parameter of type integer that represents an array index and returns the `SalesItem` object that is located at this index in the `itemsList` array. The method returns `null` if the input index is not a valid index.
- `countType`: a method that takes as input one integer that represents the type of items to be counted where 1 means `Book`, 2 means `Dictionary`, and 3 means `AudioCD`. The method then counts how many items of this type are there in the cart. For example, if the input is 1 and the cart has an item of type `Book` and quantity 2 then this adds 2 the output count. The method returns 0 for any input value other than 1,2, or 3.
- `toString`: a method that prints a nicely formatted output of the cart where each item is printed in one line. The method also prints the cart's total cost.

`class ItemDiscount`

The `ItemDiscount` class is used to calculate the discount amount on a given sales item. The class has two instance variables:

- `discount`: an **integer** value that represent the discount percent. For example, discount of 10 means 10% discount.
- `qualifiedQuantity`: an integer value that represents the required quantity in order for the item to be qualified for discount.

And the following two methods:

- A constructor to initialize the two instance variables. The order of inputs to the constructor must be `discount` then `qualifiedQuantity`.
- `calculateDiscount`: a method that takes one input of type `SalesItem`. The method checks whether the item's quantity to decide whether the item is qualified for discount or no. If the item is qualified, then the method returns a double value that represent amount of money to be deducted from the item. For example, assume the store would like to apply 10% discount if 5 or more entities of the same item are sold together. To implement that, the following `ItemDiscount` object created.

```
ItemDiscount d = new ItemDiscount(10,5);
```

  Then, for each sales item, `itm`, the discount amount can be calculated as follows:

```
d.calcualteDiscount(itm);
```

`class ShoppingDriver`

The driver class includes a main method that does the following. Make sure to include appropriate test messages in your driver.

- Instantiate a `Cart` Object that can hold up to 5 items.
- Add five items of your choice and make sure to include at least one item of each type.
- Create an `ItemDiscount` class to apply 15% discount if 10 entities or more of any item are soled together.
- Write a loop that loops over all items in the cart and does the following:
  - Apply the `monthlyPromo` method.
  - calculate and display the discount that will applied to each item, if any.
- Display the total discount to be applied to the cart.
- Display the total cart cost before and after the discount.

What to submit?

Upload to D2L one zip file that includes the following:

- **A word document** that includes UML diagram of `BookStoreApplication`.
- A zip file that includes the all the seven files as explained above.