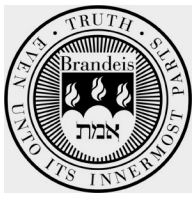

COMPUTER SCIENCE 12B-1 (SPRING TERM, 2017)
PROGRAMMING IN JAVA

PROGRAMMING ASSIGNMENT 4

Due Date: March 16, 2017, 11:55pm

Overview:

This assignment will give you practice with `ArrayList`.

Part 1

Your class will support a sample program that demonstrates some of the operations a windowing system must perform. Instead of windows, you will be manipulating "tiles". A tile consists of a position (specified by the upper-left corner `x` and `y`), a `width`, a `height` and a `color`. Positions and distances are specified in terms of pixels, with the upper-left corner being (0, 0) and the `x` and `y` coordinates increasing as you move left and down, respectively.

You won't have to understand a lot about tiles and coordinates. That code has been written for you. You are writing a small part of the overall system that keeps track of the list of tiles. You are required to implement this using an `ArrayList`. The only method you need to use from the `Tile` class (it is available in LATTE) is the following:

```
// returns true iff the given (x, y) point falls inside this tile
public boolean inside(int x, int y)
```

Your class is to be called `TileList` and must implement the following public methods:

```
// constructs an empty tile list
public TileList()

// searches through the list of tiles and returns a reference
//(i.e, an object) to the last tile for which (x, y) is inside
// the tile; returns null if (x, y) is not inside any tile of the
// list; moves the found tile to the back of the list

public Tile moveToBack(int x, int y)

// post: inserts t at the back of the list of tiles
public void insertBack(Tile t)

// post: returns the number of tiles in this list
public int size()

// post: returns the Tile at the given index
public Tile get(int index)
```

The file `TileMain.java` contains the main client program. Once you have written your solution to `TileList` and it compiles, try compiling and running `TileMain`. You should be

able to create rectangles by clicking and dragging to indicate the endpoints. You should also be able to click on a tile to bring it to the top and you should be able to click and drag to move a tile to a new location.

You should download from LATTE the classes `Tile.java`, `TileFrame.java`, `TileListener.java`, `TilePanel.java` and `TileMain.java`. Place them in the same directory as your `TileList.java` in order to run the client program and test your implementation.

Part 2

This assignment will allow you to review defining Java classes and the use of an `ArrayList`. You are to write a set of supporting classes for a simple shopping program. We are providing GUI code (Graphical User Interface) that will provide the "front end" to the program. You are to write the back end (what is often referred to as the "domain specific code").

Below is a screen shot of what the program could look like when the user has selected various items to order.

Prices are expressed using doubles and quantities are expressed as simple integers (e.g., you can't buy 2.345 of something). Notice that some of the items have a discount when you buy more. For example, silly putty normally costs $3.95 each, but you can buy 10 for $19.99. These items have, in effect, two prices: a single item price and a bulk item price for a bulk quantity. When computing the price for such an item, apply as many of the bulk quantity as you can and then use the single item price for any leftovers. For example, the user is ordering 12 buttons that cost $0.99 each but can be bought in bulk 10 for $5.00. The first 10 are sold at that bulk price ($5.00) and the two extras are charged at the single item price ($0.99 each) for a total of $6.98.

At the bottom of the frame you will find a checkbox for an overall discount. If this box is checked, the user is given a 10% discount off the total price. This is computed using simple double arithmetic, computing a price that is 90% of what it would be otherwise. For example, if we turn on that checkbox, the frame looks like this:

You are to implement four classes that are used to make this code work. You should implement a class called **`Item`** that will store information about the individual items. It should have the following public methods.

Method	Description
`Item(name, price)`	Constructor that takes a name and a price as arguments. The name will be a String and the price will be a double. Should throw an IllegalArgumentException if price is negative.
`Item(name, price, bulk quantity, bulk price)`	Constructor that takes a name and a single-item price and a bulk quantity and a bulk price as arguments. The name will be a String and the quantity will be an integer and the prices will be doubles. Should throw an IllegalArgumentException if any number is negative.
`priceFor(quantity)`	Returns the price for a given quantity of the item (taking into account bulk price, if applicable). Quantity will be an integer. Should throw an IllegalArgumentException if quantity is negative.
`toString()`	Returns a String representation of this item: name followed by a comma and space followed by price. If this has a bulk price, then you should append an extra space and a parenthesized description of the bulk pricing that has the bulk quantity, the word "for" and the bulk price.
`getName()`	Returns the name of the item

You should implement a class called **`Catalog`** that stores information about a collection of these items. It should have the following public methods.

Method	Description
`Catalog(name)`	Constructor that takes the name of this catalog as a parameter. The name will be a String.
`add(item)`	Adds an Item at the end of this list.
`size()`	Returns the number of items in this list.
`get(index)`	Returns the Item with the given index (0-based).
`getName()`	Returns the name of this catalog.

You should implement a class called **`ItemOrder`** that stores information about a particular item and the quantity ordered for that item. It should have the following public methods.

Method	Description
`ItemOrder(item, quantity)`	Constructor that creates an item order for the given Item and given quantity. The quantity will be an integer.
`getPrice()`	Returns the cost for this item order.
`getItem()`	Returns a reference to the item in this order.
`toString()`	Returns a string presentation of the item order. The String should have the format "(<item name>, <quantity>)"

You should implement a class called **`ShoppingCart`** that stores information about the overall order. It should have the following public methods.

Method	Description
`ShoppingCart()`	Constructor that creates an empty list of item orders.
`add(item order)`	Adds an item order to the list, replacing any previous order for this item with the new order. The parameter will be of type ItemOrder.
`setDiscount(value)`	Sets a boolean value that represents whether or not this order gets a discount (true means there is a discount, false means no discount).
`getTotal()`	Returns the total cost of the shopping cart.
`sortCart()`	It calls the sort static method in the Collections class and sorts the list of items. The cart should be sorted by the item order quantity, item orders with lower quantity should come first followed by item orders with higher quantities. Items with same quantity can appear in any order.
`toString()`	Returns a String representation of the item order list.

In order to sort the `ShoppingCart` you have to think about which class should implement the `Comparable` interface we introduced in class.

You are not to introduce any other public methods to these classes (except a `compareTo` method where needed), although you can add your own private methods. You are allowed to redefine `toString` in any of these classes (you might find that helpful in testing and debugging your code). You should use an `ArrayList` to implement the `ShoppingCart` and `Catalog` classes, but they should not extend `ArrayList`.

You will probably want to write your own testing code so that you can develop these classes in stages rather than all at once. When you have confidence that your classes are working, you should combine them with the GUI classes to make sure that they are working properly. You will need to include the files `ShoppingFrame.java` and `ShoppingMain.java` from LATTE

in the same folder as your program to run the GUI. You should open and compile `ShoppingMain.java` to run the program.

You will notice that as you scroll down in the GUI and pass through the items in the catalog adding quantities, your **sorted** shopping card is printed on your console. Also, note that if your cursor is placed in one item in your catalog and you don't put a quantity for that item, the shopping card will include this item with zero quantity. That is not a bug from your part, is just the way the application is implemented, so do not worry about it! You should make sure that the printed shopping card is sorted by the quantity of its items.

Your classes should be stored in files called `Item.java`, `Catalog.java`, `ItemOrder.java` and `ShoppingCart.java`.

General Stylistic Guidelines

PART 2: Most of these methods are fairly simple to write, but notice that when you add an `ItemOrder` to a `ShoppingCart`, you have to deal with replacing any old order for the item. A user at one time might request 3 of some item and later change the request to 5 of that item. The order for 5 replaces the order for 3. The user isn't requesting 8 of the item in making such a change. The add method might be passed an item order with a quantity of 0. This should behave just like the others, replacing any current order for this item or being added to the order list.

In the `Item` class you need to construct a `String` representation of the price. This isn't easy to do for a number of reasons, but Java provides a convenient built-in object that will do it for you. It's called a `NumberFormat` object and it appears in the `java.text` package (so you need to import `java.text.*`). You obtain a formatter by calling the static method called `getCurrencyInstance()`, as in:

```
NumberFormat nf = NumberFormat.getCurrencyInstance();
```

You can then call the "`format`" method of this object passing it the price as a double and it will return a `String` with a dollar sign and the price in dollars and cents. For example, you might say:

```
double price = 38.5;
String text = nf.format(price);
```

This would set the variable text to "`$38.50`".

There are several potential errors that you are required to handle, as outlined above. If the client requests an illegal index for the catalog, the `ArrayList` you are using will throw an `IndexOutOfBoundsException`. This is the right thing to have happen, so you don't have to introduce your own check for this case.

PART 1 & PART 2: You will be graded on program style including the use of good variable names, comments on each class and each method, using local variables when possible, correct use of generics and the other standard style guidelines.

Submission:

Your Java source code should be submitted via. For late policy check the syllabus.

Below is the list of files you should submit.

1. For the first part of the assignment : `Tile.java, TileFrame.java, TileList.java, TileListener.java, TileMain.java, TilePanel.java and readmepart1.txt.`

   The file readmepart1.txt should include your name, the date and the homework number and describe what the program of part1 is designed to do, and what it actually does.

2. For the second part of the assignment: `Item.java,Catalog.java, ItemOrder.java, ShoppingCart.java, ShoppingFrame.java, ShoppingMain.java and readmepart2.txt`

   The file `readmepart2.txt` should include your name, the date and the homework number and describe what the program of part2 is designed to do, and what it actually does.

Grading:

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.
- **One-on-one interactive grading:** By the end of the day that the assignment is due, please make an appointment with your TA for an interactive 10-15 minute grading session. You will receive an email notifying you of the name of the TA who has been assigned to you for this assignment with further instructions on setting up the appointment. (You will be meeting with a different TA for each assignment). One-on-one interactive grading will help you improve your programming skills and avoid repeating mistakes from one assignment to the next.