**Shopping Cart**

1. At this point, it's probably a good idea to move to a next-level IDE (if you haven't already). The ["How to use the Eclipse IDE" powerpoint from AP CS](https://www.dropbox.com/s/e7q97ql7h847kd5/02a%20How%20to%20use%20the%20Eclipse%20IDE.pptx?dl=0) will get you up and running; check the ["More Eclipse tricks and tips"](https://www.dropbox.com/s/3tqpw6juhpvydfl/20a%20More%20Eclipse%20tips%20and%20tricks.pptx?dl=0) at some point for more info on everything Eclipse can do.
2. If you would prefer to use another IDE like IntelliJ, VS Code, etc., that's fine! However, you're on your own with it – there is no way I can support every possible IDE option.
3. I will be showing you Eclipse tips and tricks throughout the year – even if you don't use Eclipse, it's likely that your IDE can do something similar.
4. (Riddle) How can the number four be half of five?

/\* I am a big fan of riddles. You will find them in most labs, they're not required and purely for fun! \*/

In this lab, you will create a graphical "shopping cart" style application. Import the **ShoppingMain.java** and **ShoppingFrame.java** files from the starter code folder into your project. ShoppingMain is a client class, for testing, with a main method; the ShoppingFrame class provides the graphical user interface (GUI).

Begin by adding the following classes:

**Item.java** – basic description of a single item in the shopping cart

|  |  |
| --- | --- |
| Item(String name, double price) | Constructor with only name and price parameters. Should utilize the four-parameter constructor below (with a call to this(), i.e. "constructor chaining", see **FAQ** for more info). |
| Item(String name, double price,  int bulkQty, double bulkPrice) | Overloaded constructor, also takes a bulk quantity and a bulk price as arguments, representing the discounted price (each) if the user buys bulkQty or more items. Throws an exception if any number is negative, as shown below:  if (...) throw new IllegalArgumentException("error"); |
| double priceFor(int quantity) | Returns the price for a given quantity of Item (considering bulk price, if applicable). Should throw an IllegalArgumentException if quantity is negative. |
| boolean equals(Object obj) | *<overridden>* Returns true if this Item has the same name as the supplied Item. See [here](https://www.dropbox.com/s/f2m3ma6lzqrruuf/10a%20The%20Object%20class%20%28toString%28%29%2C%20equals%28%29%29.pptx?dl=0) if you need a refresher on the Object class and its commonly overridden methods. |
| String toString() | *<overridden>* Returns a String representation of this item: name followed by a comma and space, followed by $price.  If this Item 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. |

**Catalog.java** - stores information about a collection (list) of Items for sale

|  |  |
| --- | --- |
| Catalog(String name) | Constructor that takes the name of this catalog as a parameter. |
| void add(Item item) | Adds an Item to the catalog (list). |
| int size() | Returns the number of items in this list. |
| Item get(int index) | Returns the Item at the supplied index. |
| String getName() | Returns the name of this catalog. |

**ItemOrder.java** - bundles together an item and the quantity ordered for that item

|  |  |
| --- | --- |
| ItemOrder(Item item, int qty) | Constructor that creates an item order for the given item and given quantity. |
| double getPrice() | Returns the cost for this item order. |
| Item getItem() | Returns a reference to the Item in this order. |
| boolean equals(Object obj) | *<overridden>* Returns true if this ItemOrder contains the same Item as the supplied ItemOrder. |

**ShoppingCart.java** - stores information about the user's orders

|  |  |
| --- | --- |
| ShoppingCart() | Constructor that creates an empty list of ItemOrders. |
| void add(ItemOrder newOrder) | Adds an ItemOrder to the list, **replacing any previous order for this item with the new order.**  Used when the user updates the quantity of an order for an Item. Should use calls to corresponding overridden equals methods.  /\* ArrayList's *contains* method uses a call to the *equals* method of the type it's storing. The *equals* method is the way Java tests objects for equivalence – if you haven't overridden a class' *equals* method, it will use the version inherited from Object! \*/ |
| double getTotal() | Returns the total cost of the shopping cart. |

You should not introduce any other public methods to these classes, although you can add your own private methods if needed. You can override toString in any of these classes (you might find that helpful in testing and debugging your code).

Test that your code works, for regular and bulk quantities, adding/removing, etc.

*Based on the* ***Shopping Cart*** *project*

*garfieldcs.com/wordpress/wordpress/wp-content/uploads/2013/03/Project-Shopping-Cart.pdf*

**(Advanced)** **Buy / sell advanced problems**

*Problems in red with an* ***(Advanced)*** *tag are generally more challenging and completely optional (not required to get 100). Give them a shot if you have the time!*

* Complete the method int buySell2(int[] stock), where the element at index i in stock represents the price of a share of a particular stock on day i. The method should return the maximum potential profit; however, you can make as many transactions as you like. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

[1, 2, 7, 4, 11] >>> 13

[2, 6, 8, 7, 8, 7, 9, 4, 1, 2, 4, 5, 8] >>> 16

* Complete the method int buySell3(int[] stock), where the element at index i in stock represents the price of a share of a particular stock on day i. The method should return the maximum potential profit, given ***you can make at most two transactions***. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again). This problem is harder than it appears.

[1, 4, 7, 2, 11] >>> 15

[1, 2, 4, 2, 5, 7, 2, 4, 9, 0, 9] >>> 17

* Complete the method int buySell4(int k, int[] stock), where the element at index i in stock represents the price of a share of a particular stock on day i. The method should return the maximum potential profit; ***however, you can make at most*** *k* ***transactions***. You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

4, [1, 2, 4, 2, 5, 7, 2, 4, 9, 0] >>> 15

2, [1, 2, 4, 2, 5, 7, 2, 4, 9, 0, 9] >>> 17