Shopping Cart

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").
`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. `/* add the @Override annotation to assert to the compiler that you're overriding a super-class method */`
`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.

If you haven't yet, check the "Eclipse tips and tricks" PowerPoint for some ways Eclipse can make your life easier (the code generation slides will be especially useful as you're writing a lot of simple classes).

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 <u>Item</u> at the supplied index.
`String getName()`	Returns the name of this catalog.

<u>ItemOrder.java</u> - 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 <u>Item</u> in this order.
`boolean equals(Object obj)`	*<overridden>* Returns true if `this` <u>ItemOrder</u> contains the same <u>Item</u> as the supplied <u>ItemOrder</u>.

<u>ShoppingCart.java</u> - stores information about the user's orders

`ShoppingCart()`	Constructor that creates an empty list of <u>ItemOrders</u>.
`void add(ItemOrder newOrder)`	Adds an <u>ItemOrder</u> 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 <u>Item</u>. 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.

When finished, create a runnable JAR file (an archive (bundle) of all the files necessary to make your project run) along with your source code (inside the same project folder is fine). Here is how:

- **BlueJ**: *Project -> Create jar file…*, then choose the class with the `main` method and the destination.
- **Eclipse**: *File -> Export*, then *Java -> Runnable JAR file*. Choose the "launch configuration" (the class with the `main` method in your project), then the destination.
  - If you don't see your class with the `main` method as an option, make sure you've run the program at least once; Eclipse will make a launch configuration for you when it runs.