Due: Activity (in-lab): Monday, November 7, 2016 by the end of lab

Goals:

By the end of this project you should:

- Have an understanding of polymorphism via inheritance
- Be able to create methods that use polymorphism to deal with multiple types of objects

Directions:

For this assignment, you will need your classes from Activity 9 except for the driver program.

ItemsList: class (5%)

- Create a class called ItemsList. This class will hold an array of InventoryItem objects (which includes objects of subclasses of InventoryItem; recall an instance of a subclass of InventoryItem *is an* InventoryItem).

ItemsList: instance variables, constructor, and methods (20%)

- Declare an instance variable in your class of type InventoryItem array which is called inventory and an int called count to track the number items in inventory.

```
private __________[] inventory;
private _____ count;
```

- Create a constructor for ItemsList that has no parameters. Add the following code to instantiate the InventoryItem array.

```
_____ = new InventoryItem[20];
count = ____;
```

- Create the following method stubs for your class:
  - `addItem`: with parameter InventoryItem itemIn; no return.
  - `calculateTotal`: with parameter double electronicsSurcharge; double return.

- The addItem method takes an InventoryItem as a parameter, assigns it to the element at positon count in the inventory array, and then increments count. No need to worry about exceeding the capacity of the array in this activity.

- Create* a toString method in ItemsList. The toString method should iterate through inventory from 0 up to count (but not including count) to include the toString for each item:

```
public String toString() {
    String output = "All inventory:\n\n";

    for (int i = 0; i < ______; i++) {
        output += __________ + "\n";
    }

    return _______;
}
```

* When you define a toString method in your class, you are overriding an inherited toString method. Recall, the Object class has a toString method, which is inherited by its subclasses.

Activity10 class and main method (20%):

- Create a class called Activity10 and add a main method. In the main method, instantiate a ItemsList object called myItems.

- In the main method, invoke setTaxRate in InventoryItem and set the tax rate to 0.05.

- Instantiate the following 4 items in the main method and add them to myItems:
  - ElectronicsItem: name = "laptop", price = $1234.56, weight = 10 lbs
    ```
    myItems._____(new ElectronicsItem(_____, 1234.56, 10));
    ```
  - InventoryItem: price = $9.8, name = "motor oil"
  - OnlineBook: price = $12.3, name = "All Things Java"
  - OnlineArticle: price = $3.4, name = "Useful Acronyms"

- Print the toString return value of the ItemsList object myItems to standard output:

```
----jGRASP exec: java -ea Activity10

All inventory:

laptop: $1311.288
motor oil: $10.290000000000001
All Things Java - Author Not Listed: $12.3
Useful Acronyms: $3.4

----jGRASP: operation complete.
```

ItemsList: calculateTotal (20%)

- The calculateTotal method in ItemsList accepts surcharge for electronics items as a parameter and returns the price of all of the items. You'll have to iterate through each of the items in inventory and add the cost for each item to a running sum (price below). If the item is an ElectronicItem, invoke calculateCost method and add the electronics surcharge that was passed as a parameter to calculateTotal.

```
double total = 0;
for (int i = 0; i < count; i++) {

}
return total;
```

- **Use the *instanceof* operator to determine whether an object is an instance of particular class** (place this *if-else* inside the for loop). If the object is of type ElectronicsItem, the following code shows how the *instanceof* operator is used to add the small surcharge to an ElectronicsItem:

```
if (inventory[i] instanceof ElectronicsItem) {
   total += _____________.calculateCost() + electronicsSurcharge;
}
```

Add an else clause to calculate the cost without adding the surcharge:

```
else {
   total += inventory[i].______________;
}
```

- Be sure to <u>return total</u> after the loop.

Main method (15%)

- In the main method, add code to print the total of all items and include an electronics surcharge of 2.0:

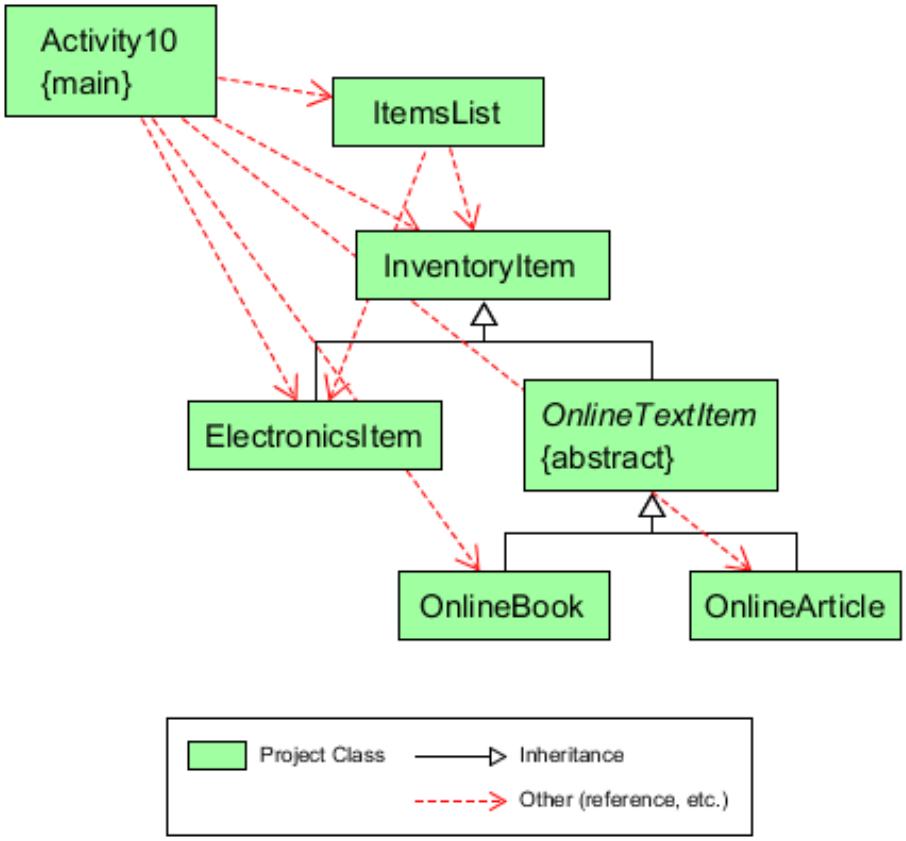
```
System.out.println("Total: " + myItems.calculateTotal(2.0));
```

- Your output should appear as follows.

```
 ----jGRASP exec: java -ea Activity10

All inventory:

laptop: $1311.288
motor oil: $10.290000000000001
All Things Java - Author Not Listed: $12.3
Useful Acronyms: $3.4

Total: 1339.278

 ----jGRASP: operation complete.
```

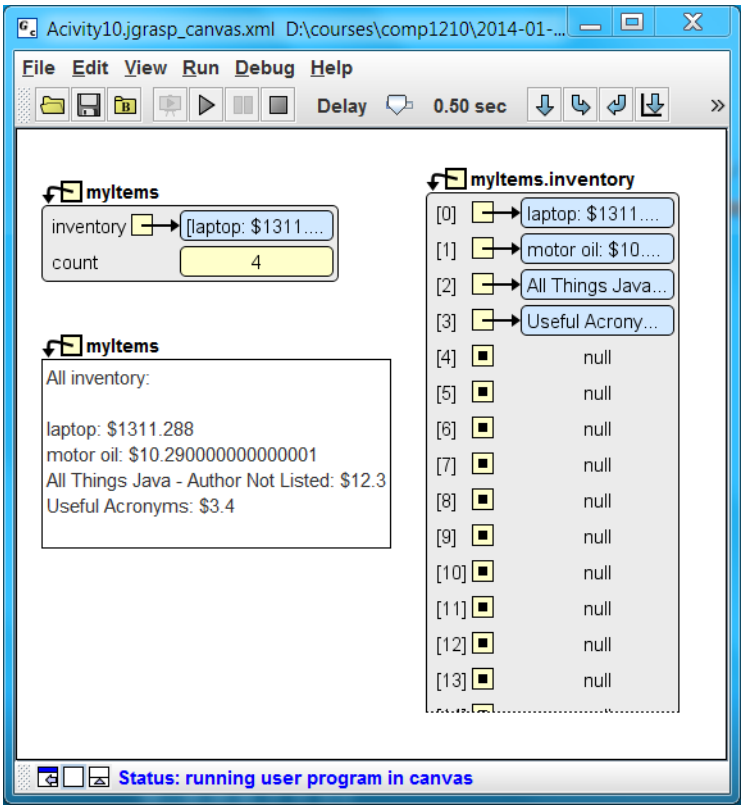
UML Class Diagram (10%)

- If you have not already done so, create a jGRASP project called Activity10 and add all of your classed for this activity.
- Generate theUML class diagram and arrange the class as shown below.



Viewer Canvas (10%)

- Create a canvas for your program as follows:
  (1) Run your program in the canvas. After the canvas opens, single step your program until you see myItems in the Debug tab.
  (2) Drag myItems onto the canvas. Then select the inventory field in myItems and drag it out onto the canvas.
  (3) Drag myItems onto the canvas again and change the viewer to the "toString" viewer. To do this, select the viewer window, click the menu button on the top-right of the viewer frame, then select "Viewer" then "toString".



  (4) Save the canvas then end the program. Run your program in the canvas. Now single step your program and watch the objects appear on the canvas.
  (5) Run your program in the canvas. Click the play button and watch the canvas and stepping in the program as the objects appear on the canvas. Notice that the debugger is stepping into each of your constructors and method as they are called. You should be sure that your understand "how" your program is running to accomplish its task. Note that you can pause the canvas and then hit play to continue auto stepping-in. You can also set one or more breakpoints and then resume to breakpoint.

- You should become comfortable using the canvas and debugger so that when needed, you will be able to use them to help find and correct errors in your projects.