Goals:

By the end of this project you should be able to do the following:

- Derive subclasses (or child classes) from a base (or parent) class
- Use the protected modifier for inheritance
- Use the super reference to refer to a parent class
- Create an abstract class
- Override methods in child classes

Directions:

Don't forget to add your Javadoc comments for your classes, constructor, and methods in this activity.

For this assignment, you will need to create a class called InventoryItem that represents an inventory item in a store. InventoryItem should have the following characteristics:

InventoryItem

- Variables:
  - A **String** called name using the protected visibility modifier
  - A **double** called price using the protected visibility modifier
  - A private static **double** called taxRate (initialized to 0)
- Constructor:
  - Sets the values for the instance variables name and price:
    ```
    public InventoryItem(String nameIn, double priceIn)
    ```
- Methods:
  - `public String getName() {`
    - Returns the customer name
  - `public double calculateCost() {`
    - Returns the price including tax: price * (1 + taxRate)
  - `public static void setTaxRate(double taxRateIn){`
    - Sets the tax rate
  - `public String toString() {`
    - Return a String representation with name and price with tax
      Example: "Computer: $789.02"
      ```
      return name + ": $" + ___________();
      ```

```
InventoryItem.setTaxRate(0.08);
InventoryItem item1 = new InventoryItem ("Birdseed", 7.99);
item1
Birdseed: $8.6292
InventoryItem item2 = new InventoryItem ("Picture", 10.99);
item2
Picture: $11.869200000000001
```

Be sure to unfold these objects on the workbench to verify the field values.

ElectronicsItem

- In a new file, create a class called ElectronicsItem that inherits from InventoryItem. Electronics items will have all of the characteristics of Inventory items and will take into account shipping costs.

```
public class ElectronicsItem extends InventoryItem {
```

- Add a **protected double** instance variable for the **weight** of the item. Then add a public constant to represent the shipping cost per pound:

```
public static final ________ SHIPPING_COST = 1.5;
```

- Add a constructor to ElectronicsItem that takes a **String for name (nameIn)**, a **double for price (priceIn)**, and a **double for weight (weightIn)**. Invoke the constructor for InventoryItem using the reserved word super:

```
super(nameIn, priceIn);
```

  Add code to set the weight of the item. Now compile this class.

- **Override** the calculateCost method to take into account shipping.

```
public double calculateCost() {
    return super.calculateCost() + (SHIPPING_COST * weight);
}
```

```
InventoryItem.setTaxRate(0.08);
ElectronicsItem eItem = new ElectronicsItem("Monitor", 100, 10.0);
eItem
Monitor: $123.0
```

OnlineTextItem

- In a new file, create a class to represent an online text item that users can buy (such as an electronic book or journal article). This class does not need to be instantiated as it is just a concept that represents a number of items, so it is an *abstract* class (more on abstract classes on Wednesday in class):

```
public abstract class OnlineTextItem extends InventoryItem {
```

- Write a constructor that will only call the constructor of the parent class (InventoryItem) using the super reserved word:

```
public _____________(String nameIn, double priceIn) {
    super(_________, _____________);
}
```

- These items are not taxed, so you'll need to override the calculateCost method to only return the price. **The price variable has been inherited from InventoryItem**:

```
public _____________calculateCost() {
    return price;
}
```

OnlineArticle

- Because we cannot instantiate OnlineTextItem, we will need to create subclasses that inherit from OnlineTextItem. In a new file, create the class OnlineArticle which is a electronic text that keeps track of word count in addition to everything that is done by OnlineTextItem and InventoryItem:

```
public class OnlineArticle extends OnlineTextItem {
    private int wordCount;
```

- As with your other classes, you will need a **constructor** with parameters for nameIn and priceIn that calls the constructor of the parent class and also **initializes wordCount to 0**.
- Add a **setWordCount** method to set the wordCount variable.

OnlineBook

- In a new file, create the class OnlineBook which will need to inherit from OnlineTextItem and will need to include a variable for the author's name:

```
public class OnlineBook extends OnlineTextItem {
   protected String author;
```

- As with your other classes, you will need a **constructor** with parameters for nameIn and priceIn that calls the constructor of the parent class and also initializes the author String to "Author Not Listed".

- **Override** the toString method so that the author's name is listed after the name of the book in the format "book name - author's name : $price (see interactions output below).

- Add a setAuthor method to set the author's name and then try the following in the interactions pane:

```
▶ OnlineBook book = new OnlineBook("A Novel Novel", 9.99);
▶ book
  A Novel Novel - Author Not Listed: $9.99
▶ book.setAuthor("Jane Lane");
▶ book
  A Novel Novel - Jane Lane: $9.99
```

InventoryApp – driver program with the main method

- In a new file, create the class InventoryApp as your driver program and add a main method that does the following:
  - Sets the tax rate to 0.05 by calling the static method InventoryItem.setTaxRate(0.05).

  Then instantiates and prints each of the following four objects using variable names *item1*, *item2*, *item3*, and *item4* respectively:
  - **InventoryItem** – name: Oil change kit; price: $39.00
  - **ElectronicsItem** – name: Cordless phone; price: $80.00; weight: 1.8 lbs
  - **OnlineArticle** – name: Java News; price: $8.50; wordcount: 700 words
  - **OnlineBook** – name: Java for Noobs; price: $13.37; author: L. G. Jones

Compile and run the program in canvas mode. After the canvas window opens, click **View** on the canvas toolbar and then select the check box for "Show Types in Viewer Labels". Now *single step* the program, then drag the objects onto the canvas shown in Figure 1 below. Unfold each of the objects in the debug tab to see the details of each field. Figure 2 shows *item4* unfolded in the debug tab. Notice that the symbol in front of each field is color-coded: *green* indicates the field was declared in the OnlineBook class (i.e., the type of item4); *orange* indicates the field was inheritited from the parent or super class which shown at the end of the field description. Also, note that *taxRate* is underlined to indicate that the variable is *static*.

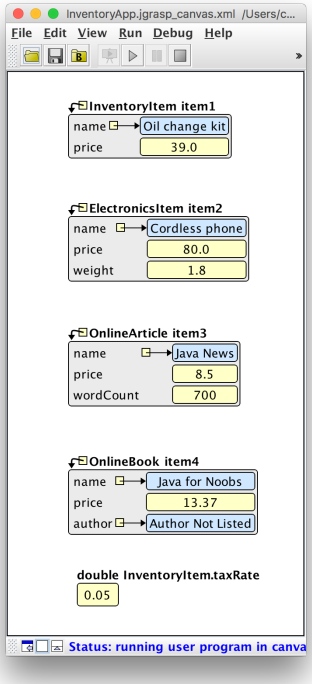

Figure 1. Canvas with the four objects created in the main method. The variables item1, item2, item3, and item4 are shown using the default *Basic* viewer.

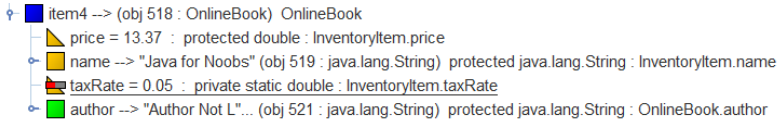

Figure 2. The variable item4 (unfolded) in the debug tab. On the canvas, the Detailed viewer can be used to display the same information as shown in the debug tab.

Project File (5%)

- Create a project file named InventoryApp and then add all of your java files above.
- Click on the UML diagram to generate the class hierarchy.
- Right-click in the UML diagram and select Layout > Tree Down then click All. Your UML class diagram should be similar to the one in Figure 3.

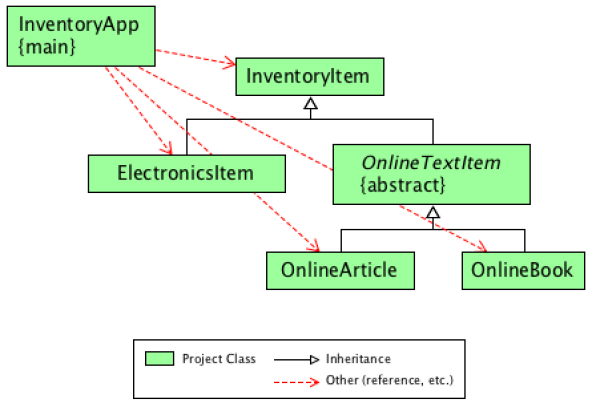

Figure 3. UML class diagram for InventoryApp.

- Finally, submit your files to the grading system.