CSC232: Object-Oriented Software Development
Homework 03 – Order Up!
Due: Day 13 – Monday, September 27th @ beginning of class

In this homework, you will (1) learn why we do not use float or double to represent currency amounts, (2) gain practice navigating and using the Java API, and (3) create a program that involves more than 1 Java class. Follow the directions below to complete this homework – be sure to read what I am asking for carefully as each component of your solution will be worth points and graded accordingly. If you wish, you may work with a single partner, however, you must email me your partner's name (CC them on the email) and only one member should submit a solution to Moodle.

Startup Steps

- On your computer, create a new folder (i.e., project) named **CSC232_HW3** – this will be the folder that you will save your Java classes in for this homework assignment
- Open Visual Studio Code, and select **File >> Open …** option, then navigate to and select the CSC232_HW3 folder in the dialog window that appears
- At this point, within Visual Studio Code, you should now see a collapsible/expandable section with the label **CSC232_HW3** in the Explorer pane on the left side. If the Explorer pane is not visible, select **View >> Explorer** or click on the icon that looks like "two sheets of paper" in the top-left corner.
- As we discussed in class, every Java program needs a class that contains the main( ) method. Add a **Driver.java** file to your project by hovering your mouse over the collapsible/expandable section with the label CSC232_HW3. You should see an icon of "a sheet of paper with a + sign" that reads **New File** when you hover on top of it. Click on this icon and a new file will appear waiting for you to provide its name – name this new file **Driver.java**
    - **Important**: You must put the **.java** extension on the end of the name, otherwise your computer will not know that Driver.java contains Java code
- Next, add the necessary code to define your Driver class – then, add the necessary code to define the main( ) method.
    - **Important**: In Java, the name of the class must always match the name of the file that it is stored in (excluding the .java extension). Therefore, your class's name must be Driver (lowercase vs uppercase matters)
- When you are finished with the steps above, a **Run** link should appear over the main( ) method. (If a Run link does not appear, be sure to rewatch the videos on Moodle in the Course Resources section for how to download and install the JDK, Visual Studio Code, and the Java Extensions Pack). Once the Run link is visible, click on it to run/execute your program. **Recall**: All Java programs begin executing at the main( ) method that you selected, therefore you can always trace through exactly what your code will do.
    - A window should appear at the bottom with the **Terminal** tab selected – this is the tab where your Java program will print its output. You may wish to add a System.out.println("Hello World") statement to your main( ) method to ensure everything is working correctly before continuing to the next steps
    - **Tip**: For this homework, I recommend using your main( ) method to create objects that test the creation of objects and calling their methods to ensure they are working correctly

Pre-Homework Steps

- First, I want you to observe the issue that we face if we use a floating-point type (e.g., **float** or **double**) to represent currency amounts. In your main( ) method, create a double variable whose name is **apples** and assign it the value 3.25, then, create another double variable whose name is **bananas** and assign it the value 1.06. Finally, let's create a double variable whose name is **result** and have it store the sum of apples and bananas. Now, print result's value to the screen … what do you notice? You should see that result does not store 4.31 as it should but instead stores 4.310000000000005. This is due to the fact that computers represent all of their data in **binary** which is a number system is based on **powers of 2** (this is similar if you try to represent the value 1/6 in **decimal** which is a number system based on **powers of 10** – the result is 0.16666667 – because your computer/calculator cannot store an infinite number of digits for a repeating value). Try exploring other combinations of currency values that produce a result with more digits than we would expect.
    - **Note**: You can remove these variables when you are finished
- As a result, Java provides the **BigDecimal** class for creating objects that can store values with a high-degree of accuracy and precision without loss of data. Take a moment to practice navigating to a class that Java provides (i.e., BigDecimal) using the Java API as we demonstrated in class.
- Once you arrive at the API webpage for the BigDecimal class, you will notice that this class is located in the **java.math** package. Click on the "CONSTR" link at the top of the page to jump down to the BigDecimal constructors. As we have discussed in class, a class can have multiple constructors – as this provides others who want to use this class (i.e., like us) the ability to construct a BigDecimal object in a variety of ways. **Important**: You should only construct a BigDecimal object for this assignment using a String (i.e., the **BigDecimal(String val)** constructor). The **BigDecimal(float val)** and **BigDecimal(double val)** constructors will still result in inaccurate values if you were to use them instead – therefore, do not use these two constructors.
- Let's observe that BigDecimal objects resolve the previous problem by first, creating a BigDecimal variable/object whose name is **apples** and that stores the value 3.25 (i.e., **BigDecimal apples = new BigDecimal("3.25")**). Next, create another BigDecimal variable/object whose name is **bananas** and that stores the value 1.06 (i.e., **BigDecimal bananas = new BigDecimal("1.06")**).
- On the BigDecimal API webpage, click on the "METHOD" link at the top to jump down to the BigDecimal methods section. You will notice that all of the major math operators (+ - * /) have a corresponding method (**add**( ) , **subtract**( ) , **multiply**( ) , **divide**( )). This is because, in Java, the math operators (+ - * /) are only defined for the Primitive types and BigDecimal is not a Primitive type (it is a class). Therefore, where we would normally use a math operator, we instead have to call the corresponding method and the method will return a BigDecimal object that represents the result. For example, try typing in the following code in your main() method:

```
BigDecimal apples = new BigDecimal("3.25");
BigDecimal bananas = new BigDecimal("1.06");
BigDecimal result = apples.add(bananas);

System.out.println(result.toString( ));
```

  The toString( ) method converts the BigDecimal object's value to a String so that it can be printed to the console. **Notice**: The BigDecimal object's value correctly represents the accurate sum (4.31) and avoids the previous problem when using float and double.
- Up next, you will be using the BigDecimal class to implement two Java classes: a **Product** class and an **Order** class

Homework Steps

- Add a **Product.java** file to your project's folder within Visual Studio Code and define a Product class
- Define the **member variables** listed below in your Product class that all Product objects that you construct will contain:
    - A member variable named **name** that refers to the product's name (example only: "Apples")
    - A member variable named **price** that stores the price of the item as a **BigDecimal** (do not use float or double or you will receive a 0% on this homework assignment)
    - **Important**: You should not add any additional member variables besides those that were requested above
- Next, write a **constructor** for the Product class that takes two inputs: (1) the name of the product as a String and (2) the price of the product **as a String** (do not use float or double)
- Then, write the following **methods** listed below that a Product should be able to perform/calculate if we were to call them:
    - A method named **getName**( ) that returns the product's name
    - A method named **getPrice**( ) that returns the product's price as a **BigDecimal** (do not use float or double)
- In your Driver's main( ) method, practice constructing different Product objects and calling their methods to ensure your code is working correctly at this point before proceeding

---

- Add a **Order.java** file to your project's folder within Visual Studio Code and define an Order class
- Define the **member variables** listed below in your Order class that all Order objects that you construct will contain:
    - A member variable named **products** that stores Products in an ArrayList
        - **Important**: You should take some time to visit the Java API's ArrayList page to remind yourself what constructors and methods an ArrayList type offers. One of the goals of this homework is to give you practice navigating and using the Java API. Please review the following simple tutorial to familiarize yourself with ArrayLists
    - A member variable named **total** that stores the sum of the individual Product prices as a **BigDecimal** (do not use float or double or you will receive a 0% on this homework assignment)
- Next, write a **constructor** for the Order class that takes no inputs and initializes the ArrayList and BigDecimal appropriately
- Then, write the following **methods** listed below that an Order should be able to perform/calculate if we were to call them:
    - A method named **addItem** that should take a Product as an input and update the ArrayList and total appropriately
    - A method named **getItem** that should take a String as an input for the Product's name that a user wants to search for in the Order. The method should return the Product in the Order whose name matches, otherwise it should return **null** (i.e., "no object"). If multiple Products have the same name, then it should return the first one that it finds.
        - **Important**: As we will soon learn, the == operator should **not** be used with Strings to determine if they are the same. Therefore, to give you more practice navigating and using the Java API, I want you to visit the String's page and find the method that we should use instead of the == operator to compare if two Strings contain the same text
    - A method named **removeItem** that should take a String as an input for the Product's name that a user wants to remove from the Order. This method should remove the Product whose name matches and

update the total accordingly. If a match is found, this method should return true when it is finished, otherwise, if no match is found, this method should return false. If multiple Products have the same name, then this method should remove the first one that it finds.

- A method named **calculateFinalPrice** that takes a **BigDecimal** that stores the sales tax (example only: 0.06) as an input (do not use float or double). This method should calculate the final price of the Order based upon the sales tax being applied to the total – and this method should return a **BigDecimal** when finished (do not use float or double)
  - **Important**: By applying the sales tax to the Order's total, the result might include fractions of cents (example only: if the total is $4.22 and the sales tax is 6%, then the final price would be $4.4732). Therefore, we need to **round down** to the nearest cent -- $4.4732 should be rounded to $4.47. Please review the following simple tutorial for how to round BigDecimal values to the nearest cent.

Testing Steps

- Before submitting your solution, you should use your Driver's main( ) method to create an Order object and add various Product objects with different values to test if all of the Order class's methods are working correctly. Below is a simple example of how I would expect your classes to work:

```
Order myOrder = new Order( );
myOrder.add( new Product("Apples", "3.16") );
myOrder.add( new Product("Bananas", "1.06") );
BigDecimal finalPrice = myOrder.calculateFinalPrice( new BigDecimal("0.06") );
System.out.println( finalPrice.toString( ) );  // This should print 4.47
```

- **Important**: Be sure to think about different scenarios that your method's code might need to handle (i.e., are there any scenarios that might "break" your code and cause the program to behave incorrectly)

Submission Steps

- Open up the Moodle in a web browser and navigate to our CSC232 course's page
- Upload the **.java** files OR a ZIP file containing your folder to the assignment's submission item on Moodle
  - **Important**: If you worked with a partner, only **one** member should submit a solution – be sure that you have emailed me who you were working with