CSCE Lab Exercise 9
More on the ArrayList Class + Primitive Wrapper Classes

Goals:

- To become familiar with the ArrayList class and some of its basic methods.
- To understand what wrapper classes are and how they can be used.
- To get more practice with the Scanner class, this time using it to parse String objects instead of reading from System.in.
- To learn how to write overloadded method, i.e., methods with the same name but different parameter types.
- To get a little exposure to generics in Java.

Orientation:

You are going to write a simple class, name of your choice, that allows a user of the class to store int, double, boolean, and String values in separate ArrayLists and retrieve information about what it contains. Your class you will allow values to be added via overloaded methods that accept the primitive data types or a String and will store them in the appropriate private ArrayList.

To be clear, you should have several ArrayList instance as members of your class, and you should use the methods provided by ArrayList when necessary.

Specification:

1. ***Review, as necessary, the Java Standard Library documentation*** for the *ArrayList*, *Scanner*, *Integer*, *Double*, *Boolean*, and *String* classes.

2. Pick a name for your class, something that captures its intended use (see Orientation above).

3. ***Define four private instance variables*** of type ArrayList. Remember that the ArrayList class is a generic class - this means you must specify the type of objects you are going to store in it. The Java generic feature does not support primitive data types but recall there is a wrapper class for each primitive type. You must use the wrapper classes Integer, Double, Boolean, for primitive types int, double, and boolean, respectively, and String as the generic types. Here is an example of how to declare and construct a private ArrayList that holds String objects:

   ```
   private ArrayList<String> myList = new ArrayList<String>();
   ```

4. Write any constructors you deem necessary to your design.

5. Now write *overloaded* ***add*** methods (methods with the same name but different parameters), one that has an ***int*** parameter, one that has a ***double*** parameter, one that has a ***boolean*** parameter, and one that has a ***String*** parameter x. All of the add methods should have a void return type. Each should add the value passed in to its associated generic ArrayList. Because of Java's automatic *boxing* and *unboxing* of primitive data types this should be relatively easy but remember that the primitive types can't actually be stored directly in the ArrayList and the the wrapper classes are involved even though the code does not

use them explicitly (though it may - give it a try later). I would suggest writing one at a time and test it before writing the next. To test you will need to write the associated accessor method (a method that lets you access private information contained by a class) specified below.

6. Write a method called ***minimumInt*** that has no parameters and returns the minimum integer value that has been added to your class, or zero if none has been added. You will have to write logic to search through the ArrayList of int values and find the minimum value.

7. Write a method called ***averageDouble*** that has no parameters and returns the average of all the double values that have been added to your class, or zero if none have been added. You will have to write logic to compute the average of all double values in the ArrayList of doubles.

8. Write a method called ***numberOfTrues*** that has no parameters and returns the number of true values that have been added to your class, or zero if none have been added. You will have to write logic to count the number of true values in the ArrayList of booleans.

9. Write a method called ***numberOfStrings*** that has no parameters and returns the number of String objects that have been added to your class, or zero if none have been added. You will have to write logic to count the number of String objects in the ArrayList of String objects.

10. When you believe your class is correct and complete, add this main method top your class. It has a loop that prompts a user to enter a value, and then uses a Scanner class instance to determine the data type of the user input. It then calls the appropriate overloadded add(Type type) method for that particular primitive type. The loop iterates until the user types "quit", then it breaks without adding the "quit" string to the list.
    Note: When comparing references for ***value equality*** (not *reference equality,*) the ***.equals(Object o)*** method ***is used*** (instead of the == operator, which only compares the addresses contained by the references)!
    Before exiting, the program should print out the minimum int, average double, number of trues, and number of Strings using the methods you wrote above.