ELEC 279 - Winter 2020
Introduction to Object-Oriented Programming
Lab 5

Generics in JAVA

1 Introduction

In this lab, you will practice Generics. As we have discussed in the class, Generic programming allows you to write parameterized classes. For example, you can define a class with a list of instance variables of type T, where T is a type parameter. You can then use this class to create objects by plugging in *String* for T. Similarly, you can plug in *Double*. The class *ArrayList* in the standard Java libraries is, in fact, just such a class for an array list of items of type T.

In this lab, you will first practice how to use a predefined class (the *ArrayList* class) with a type parameter. You will then practice how to define your own classes with type parameters. For more information about Generics, please refer to our lectures and the textbook.

2 Getting Started in Eclipse

There are a couple of steps that must be completed first to set up the lab in eclipse. Like last lab **let one member be the driver and the other the navigator.**

1. Open eclipse.
2. Start by creating a project called **"Lab5"**.
3. Select **File→New→Java Project** to create a new project.
4. Name the project **"Lab5"**.
5. Click finish.

Now we will add some classes to our project.

1. First we must create the main class for this project
2. Select **File→New→Class** to add a new class
3. Under **Name** enter *Lab5*.
4. Click finish
5. In *Lab5*, create the **main** method with proper modifiers and parameter.
6. Create two classes; "*Employee*"and "*HourlyEmployee*".
7. Copy the text for **"Employee.java "**, and **"HourlyEmployee.java "** on onQ to your created classes.
8. Create two new classes "*Pair*"and "*Triple*".

3 The ArrayList Class

The *ArrayList* class is an example of a generic class with a type parameter. The *ArrayList* class comes with a selection of powerful methods to manipulate arrays. You will practice only some of these methods here for different array types. Please refer to our lecture and the textbook for more information.

1. In the main method of *Lab5* class, create an *ArrayList* of type *Integer* with an initial capacity of 5 and name it *intArray* (You must import "java.util.ArrayList").
2. Use the appropriate method to add 10 integers to *intArray* with random values in the range of [0, 99]. (You may need to import "java.util.Random" class to create random numbers).
3. Now, reverse the sequence of *intArray* using the appropriate methods (e.g., if the originally intArray = [66, 2, 16, 35, 48, 88,9, 71, 90, 0], you should get revIntArray = [0, 90, 71, 9, 88, 48, 35, 16, 2, 66]). You must use a **For-Each loop** we have discussed in the lecture. You can follow the procedure below, but feel free to use any other approach:
   (a) Create an object *revIntArray* of type *ArrayList*.
   (b) Create a **For-Each** loop that iterates through all the elements of *intArray*.
   (c) Add each element to *revIntArray* at index=0.
4. Print the output.

Then you need to finish the following steps:

1. Now, create an *ArrayList* that contains objects of type *HourlyEmployee* with initial capacity of 100.
2. Add four *HourlyEmployee* objects into this ArrayList object.
3. Print the content of the ArrayList.
4. Print the size of the array list (i.e., the number of *HourlyEmployee* objects it has). Think about what's the difference between capacity and size. If you are not sure, check the textbook or ask TAs.
5. Create an new *HourlyEmployee* object and change the first element of the above ArrayList with this newly created object. Do not change the other elements of the *ArrayList*.
6. Print the content of the ArrayList.
7. Remove the third element in the array list and use the *trimToSize* method predefined in *ArrayList* to change the array list's actual capacity to be same as the number of its elements.

4 Creating a Generic Class

Now you will practice how to define your own generic classes. First, you will create a generic class "*Pair*" that has one type parameter for saving information for pairs of objects of the same type (e.g., a lady and her husband's names as a pair of *String*). Then, you will create a generic class *Triple* that accepts three type parameters for storing information for a triple of objects, e.g. departure city (*String*), distance (*Double*), arrival city (*String*).

1. As discussed in our lecture, inside the *Pair* generic class, define two instance variables of the same type. (Do not forget the angular bracket notation for the type parameter in the class heading).

2. Add the class constructors.

3. Add a public method **equals(Object otherPair)** that returns true if and only if both *pair*s have the same content.

4. Add a public **toString()** method that returns the values of the two instance variables.

5. Inside the main method of *Lab5* class, create three objects of type *Pair*, each storing the names of a couple (a wife and her husband's names). Make two of the three pair objects have the same content; i.e., for two couples, the wives' names are same with each other and husbands' names are same with each other.

6. Test your **equals** method and show us the **equals** works properly to compare the contents of the three *Pair* objects by printing the return values of equals.

Now, go ahead to finish the following steps:

1. Inside the *Triple* class, define three instance variables. The first and third have the same type, and the second has a different type. (Think about the example discussed above: a triple of departure city (*String*), distance (*Double*), arrival city (*String*).

2. Repeat steps 2 to 4 above and make necessary changes here for *Triple*.

3. Inside the main method of *Lab5* class define three triple objects, each storing a trip information: the departure city (String), distance (Double), and arrival city (String) such as: $Triple < String, Double > trip1 = newTriple <> (\text{"}Toronto\text{"}, 547.0, \text{"}Montreal\text{"});$
$Triple < String, Double > trip2 = newTriple <> (\text{"}Toronto\text{"}, 264.5, \text{"}Kingston\text{"});$
Make two of the three trips have same content.

4. Test your **equals** method and show us **equals** works properly in comparing the content of the three trip objects. Print the return values of **equals**.

5. Note that you can of course define the *Triple* class to have three different type parameters instead of two, so the three instance variables are defined with these three different types. Your lab partner can practice it here or off the lab and this is required by the lab and does not affect your mark.

Submit your solution before 9pm April 1 on onQ.