Lab 13: Generics

Objective

In this lab, we will look at three different ways that generics are commonly used in Java.

Overview

In this lab you will:

- Create a generic class, a generic method, and a use of generics to provide an upper bound for the classes that can be used with a method.

Step by Step Instructions

Exercise 1: Creating a generic class

There may be many times when we want to use a common class that may be used with several different types at runtime. These classes look like a regular Java class except the class name is followed with a parameter section. This is why these types of classes in Java are sometimes called parameterized classes or parameterized types. The parameter section may contain a list of one of more generics that may then be referenced in the body of the class.

1. Create a new project named **Generics** in your workspace as described previously. In the src folder create a class named **GenericBox** with a package name of **com.lq.generics**.

2. We want an instance of the **GenericBox** class to be used with different types. Indicate that the **GenericBox** class is parameterized by adding a generic parameter after the class name. You may use any name that you like for the generic parameter, but it is by convention formatted as a single cap – usually **"T"** or **"E"**.

   Generic declarations take the form *public class ClassName<T>* where *T* represents the parameter that is provided when the class is used i.e.

   ClassName<T> cn = new ClassName<>();

3. The T in your declaration is provided in code and to be used must be "bound" to an attribute of the class. In your class, add an attribute of type **T** (or whatever name you used for the generic) named **t**. Code or generate the getters and setters for t. Your code should be like the example below.

```java
public class GenericBox<T> {
    private T t;

    /**
     * @return the t
     */
    public T getT() {
        return t;
    }

    /**
     * @param t the t to set
     */
    public void setT(T t) {
        this.t = t;
    }
}
```

4. Create a driver named **GenericBoxExerciser** to see how the **GenericBox<T>** works in practice.

5. In the main method add code to create two instances of **GenericBox<T>**. The 1st instance should be of type **Integer** and the 2nd of type **String**. Name the instances as desired.

6. On the **GenericBox<String>** instance invoke **setT()** to pass the value “Hello World” to the instance. Call **setT()** on the Integer instance and pass 10.

7. Use **System.out.printf** to print the values of the parameter type to the console from both the **Integer** and **String GenericBox** instances. If you are not familiar with the printf method, look it up in the Standard Edition Javadoc. (If you would rather use println, that will work fine as well).

8. Run GenericBox as a Java Application to exercise. You should see output to the console that reflects the values that were passed using **setT().** Your completed class should be like the following:

```java
    public static void main( String... args) {
        GenericBox<Integer> integerBox = new GenericBox<>();
        GenericBox<String> stringBox = new GenericBox<>();
        integerBox.setT(10);
        stringBox.setT("Hello World");
        out.println("IntegerBox value: " + integerBox.getT());
        out.println("StringBox value: " + stringBox.getT());
    }
```

9. Generic methods allow us to write a single generic method declaration that can be called with arguments of different types. Based on the types of the

arguments passed to the generic method, the compiler handles each method call appropriately.

.

10. Create a class named **GenericMethod** in the **com.lq.generics** package Add this method to the class:

```
public static <E> void printArray(E[] inputArray)
```

Here you have declared a method named **printArray** that is bound to the parameterized type **E**. The instance of the parameterized type is named **inputArray**.

11. In the body of the method iterate over the inputArray and print each of the elements of the array. Use `System.out.printf("%s",element)` for the output. [Note: the code shown here assumes that `System.out` was statically imported.]

```
public class GenericMethod {

    public static <E> void printArray( E[] inputArray) {
        for( E element: inputArray) {
            out.printf("%s ", element);
        }
        out.println();
    }
}
```

12. Create a new class named **GenericMethodExerciser** In the body of the main method create three arrays. The 1st array is of type Integer, the 2nd is of type Double and the 3rd is of type Character. In the declaration of the arrays use {v1,v2,v3…} to populate the arrays.

13. Call the printArray<E>() method passing each of the arrays.

14. Exercise the application to verify that the values of the associated array print as expected. Your class should look similar to the following:

```
public static void main(String[] args) {

    Integer[] integerArray = { 1, 2, 3, 4, 5 };
    Double[] doubleArray = {1.1,2.2,3.3,4.4,5.5};
    Character[] charArray = {'H','E','L','L','O'};

    out.println("Integer array contains:");
    printArray(integerArray);
    out.println("Double array contains:");
    printArray(doubleArray);
    out.println("Character array contains:");
    printArray(charArray);
}
```

15. There are times in code when you want to restrict what types may be passed as a type parameter. This is what “bounded type parameters” provide in a Java application. To declare a bounded type parameter, list the type parameter's name followed by the **extends** keyword, followed by its upper bound.

    Add a class named **Maximum** to the **com.lq.generics** package.

16. Add a method named **maximum** as follows:

```
public static <T extends Comparable<T>> T maximum(T x, T y, T z) {
    T max = x;

    if (y.compareTo(max) > 0) {
        max = y;
    }

    if (z.compareTo(max) > 0) {
        max = z;
    }

    return max;
}
```

    In this method we have declared that this class is parameterized by a type that must be a type or subtype of Comparable. Since Comparable is an interface, this means that any class that implements Comparable or is a subtype of a class that implemented Comparable may be the type bound to this method.

    All the method does is use the **compareTo** method (as defined by the Comparable Interface) to compare the three method arguments to see which is the largest and to then return that value.

.

17. Add a class named **MaximumExerciser** to the **com.lq.generics** package
    Add a main method to the class to test the parameterized generic method as follows:

```
public static void main(String[] args) {
    out.printf("Maximum of %d, %d, and %d is %d%n", 5, 4, 3,
            maximum(5, 4, 3));
    out.printf("Maximum of %.1f, %.1f, and %.1f is %.1f%n", 6.6, 8.8, 7.7,
            maximum(6.6, 8.8, 7.7));
    out.printf("Maximum of %s ,%s, and %s is %s%n","pear","apple","orange",
            maximum("pear","apple","orange"));
}
```

    The exercise should display the arguments in the correct order.