GIFT UNIVERSITY
Realize Your Career Dreams

GIFT School of Engineering and Applied Sciences

Spring 2019

CS-204: Data Structures and Algorithms

Lab-1 Manual

Writing Generic Methods and Classes

v1.0
3/23/2019

Task #1: Writing Methods in Java

In this task, you are being asked to write a method **printArray** which takes an **Integer** array as an argument, and then prints the values of array on the console. You may use the following method header:

```
public void printArray(Integer[] inputArray)
```

Now, create an overloaded method with the same name to print values from a **Double** array argument. You may use the following method header:

```
public void printArray(Double[] inputArray)
```

Finally, create another overloaded method with the same name to print values from a **String** array. You may use the following method header:

```
public void printArray(String[] inputArray)
```

1. Create a class with the name **GenericsLab1.java.**
2. Create appropriate arrays: **intArray, doubleArray, stringArray** of types **Integer**, **Double** and **String.**
3. Insert values in all arrays. Do not use a **Scanner** for any inputs.
4. Make appropriate method calls for above created arrays and print the values.
5. Give appropriate message while printing values.

What are Generic Methods?

You can 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. Following are the rules to define Generic Methods:

1. All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type (< T > in the coming example).
2. Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
3. The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
4. A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char).
5. **Below is an example of a generic method**, which takes a generic argument of type **T** and prints its value regardless of its type:

```
public <T> void printGenericValue(T var) {
     System.out.println(var);
}//printGenericValue
```

Task #2: Writing a Generic Method in Java

In this task, you are being asked to write a generic method **printArray** which takes a generic array as an argument, and then prints the values of array on the console without specifying the type for the array. You may use the following method header:

```
public <T> void printArray(T[] inputArray )
```

1. Create a program called **GenericMethodsLab1.java**
2. Create the method **printArray** with the generic data type.
3. Create appropriate arrays: **intArray, doubleArray, stringArray** of types **Integer**, **Double** and **String** having appropriate values.
4. Create an appropriate call for the generic method you created above for all arrays **intArray, doubleArray,** and **stringArray** and print the values**.**
5. Give appropriate messages while printing values.

What are Bounded Type Parameters?

There may be times when you'll want to restrict the kinds of types that can be passed to a type parameter. For example, a method that operates on numbers might only want to accept instances of **Number** or its subclasses. This is what bounded type parameters are for.

To declare a bounded type parameter, list the type parameter's name, followed by the **extends** keyword, followed by its *upper bound*.

Example

Following example illustrates how **extends** is used in a general sense to mean either "**extends**" (as in classes) or "**implements**" (as in interfaces). This is an example which takes two generic arguments and return **true** if the numbers are equal, otherwise returns **false**.

```
public <T extends Number> boolean isEqual(T number1, T number2)
{
     if (number1.doubleValue() != number2.doubleValue()) {
            return false;
     } else {
            return true;
     }//if
}//isEqual
```

Task #3: Writing a Generic Method with Bounded Type Parameters

In this task, you are being asked to write a generic method **largestValue** that takes three generic arguments and prints the largest value from the given arguments. The arguments must be bounded to **Numbers** only, that means that the method will not accept any other type, such as **String** values except number values. The numbers could be of types **Float**, **Double** or **Integer**. You may use the following method header:

```
     public <T extends Number> void largestValue(T number1,
                                              T number2, T number3)
```

1. Create a program called **BoundedTypeLab1.java**
2. Create the method **largestValue** with three generic data types.
3. Create 3 variables of type **Double** having appropriate values and call the **largestValue** method.
4. Create 3 variables of type **Integer** having appropriate values and call the **largestValue** method.

What are Generic Classes?

A generic class declaration looks like a non-generic class declaration, except that the class name is followed by a type parameter section. As with generic methods, the type parameter section of a generic class can have one or more type parameters separated by commas. These classes are known as parameterized classes or parameterized types because they accept one or more parameters.

Example

```
public class Box<T> {
   private T value;

   public void setValue(T value) {
      this.value = value;
   }//Setter

   public T getValue() {
      return this.value;
   }//Getter

   public static void main(String[] args) {
     //Creating Objects with the Generic Class
      Box<Integer> integerBox = new Box<Integer>();
      Box<String> stringBox = new Box<String>();

      //Setter Calling for setting values in the Generic
      //variable value
      integerBox.setValue(new Integer(10));
      stringBox.setValue(new String("Hello World"));

      //Printing Values
      System.out.println("Integer Value: " +
integerBox.getValue());
      System.out.println("String Value: " +
stringBox.getValue());
   }//main
}//class
```

NOTE: Open the **Box.java** code from the **Code** folder. Compile and run the code.

Task #4: Writing a Generic Class

In this task you are being asked to write a simple generic class **GenericClass<T>** which has two generic private data members, and you are asked to implement its constructor, setter and getter methods, and the **print** method which will print the state of the object with appropriate messages.

You may use the following skeleton code that has been placed in the **Code** folder:

```
 public class GenericClassLab1<T> {

    private T obj1;
    private T obj2;

    //Constructor
    public GenericClassLab1(T obj1, T obj2) {
    }//GenericClass

    //Setters
    public void setObj1(T obj1) {
    }//Set Obj1

    public void setObj2(T obj2) {
    }//Set Obj2

    //Getters
    public T getObj1() {
    }//get Obj1

    public T getObj2() {
    }//get Obj2

    //print the state
    public void print() {
    }//print
}//class
```

1. Open the file **GenericClassLab1.java** from the **Code** folder.
2. Complete the implementation of the constructor, setter and getter methods, and the **print** method.
3. Create a test program called **TestGenericClassLab1.java** having the **main** method.
4. Create an object of **GenericClassLab1** with the **Integer** data type and print values.
5. Next, create an object of **GenericClassLab1** with the **Double** data type and print values.
6. Finally, create an object of **GenericClassLab1** with the **String** data type and print values.