Programs on Java Lambda Expressions

//LambdaDemo.java

```
// A functional interface.
interface MyNumber {
double getValue();
}
class LambdaDemo {
  public static void main(String args[]) {
   MyNumber myNum; // declare an interface reference
   myNum = () -> 123.45;

   // Call getValue(), which is provided by the previously assigned lambda expression.
   System.out.println("A fixed value: " + myNum.getValue());

  // Here, a more complex expression is used.
   myNum = () -> Math.random() * 100;

   // These call the lambda expression in the previous line.
   System.out.println("A random value: " + myNum.getValue());
   System.out.println("Another random value: " + myNum.getValue());

   // A lambda expression must be compatible with the method
   // defined by the functional interface. Therefore, this won't work:
  // myNum = () -> "123.03"; // Error!
   /* the lambda expression must be compatible with the abstract method that it is intended
to implement. For this reason, the commented-out line at the end of the preceding program
is illegal because a value of type String is not compatible with double, which is the return
type required by getValue( ).
  */
 }
}
```

OUTPUT:

```
A fixed value: 123.45
A random value: 88.90663650412304
Another random value: 53.00582701784129
```

//LambdaDemo2.java

```
// Demonstrate a lambda expression that takes a parameter.
// Another functional interface.
interface NumericTest {
boolean test(int n);
}
class LambdaDemo2 {
  public static void main(String args[]) {
    // A lambda expression that tests if a number is even.
    NumericTest isEven = (n) -> (n % 2) == 0;

    if (isEven.test(10)) System.out.println("10 is even");
    if (!isEven.test(9)) System.out.println("9 is not even");

     // Now, use a lambda expression that tests if a number is non-negative.
     NumericTest isNonNeg = (n) -> n >= 0;

     if (isNonNeg.test(1)) System.out.println("1 is non-negative");
     if (!isNonNeg.test(-1)) System.out.println("-1 is negative");
  }
}
```

OUTPUT:

10 is even
9 is not even
1 is non-negative
-1 is negative

//LambdaDemo3.java

```
// Demonstrate a lambda expression that takes two parameters.
interface NumericTest2 {
  boolean test(int n, int d);
}

class LambdaDemo3 {
   public static void main(String args[]) {
     // This lambda expression determines if one number is a factor of another.

     NumericTest2 isFactor = (n, d) -> (n % d) == 0;

     if(isFactor.test(10, 2))
        System.out.println("2 is a factor of 10");
     if(!isFactor.test(10, 3))
        System.out.println("3 is not a factor of 10");
   }
}
```

OUTPUT:

2 is a factor of 10

3 is not a factor of 10

//Block demo
//Sometimes there is a need for <u>one huge code block instead of single value</u>
//BlockLambdaDemo.java

```
interface NumericFunc {
  int func(int n);
}
class BlockLambdaDemo {
  public static void main(String args[]) {
    // This block lambda computes the factorial of an int value.

    NumericFunc factorial = (n) -> {
                                        int result = 1;
                                        for(int i=1; i <= n; i++)
                                                result = i * result;
                                        return result;
                                    };


    System.out.println("The factoral of 3 is " + factorial.func(3));
    System.out.println("The factoral of 5 is " + factorial.func(5));
  }
}
```

OUTPUT:
The factorial of 3 is 6
The factorial of 5 is 120

```
// A generic functional interface.
interface SomeFunc<T> {
  T func(T t);
}

class GenericFunctionalInterfaceDemo {
   public static void main(String args[]) {

       // Use a String-based version of SomeFunc.
       SomeFunc<String> reverse = (str) -> {
                                                String result = "";
                                                int i;
                                               for(i = str.length()-1; i >= 0; i--)
                                                 result += str.charAt(i);
                                               return result;
                                             };

       System.out.println("Lambda reversed is " + reverse.func("Lambda"));
       System.out.println("Expression reversed is " +reverse.func("Expression"));

      // Now, use an Integer-based version of SomeFunc.
      SomeFunc<Integer> factorial = (n) -> {
                                                int result = 1;
                                                for(int i=1; i <= n; i++)
                                                result = i * result;
                                                return result;
                                              };

      System.out.println("The factoral of 3 is " + factorial.func(3));
      System.out.println("The factoral of 5 is " + factorial.func(5));
  }
}
```

OUTPUT:

Lambda reversed is adbmaL
Expression reversed is noisserpxE
The factorial of 3 is 6
The factorial of 5 is 120