LambdaExpressions

Descriptive Scenario 1:
Task: Write a lambda expression that accepts two integers and returns their sum.
Requirement: Use the predefined functional interface implement and test the lambda.

```
package com.lambda.expressions;
import java.util.function.BiFunction;
/*@FunctionalInterface
interface Addition{
	int add(int a,int b);
}*/
public class Example1_28_04 {
	public static void main(String[] args) {
	BiFunction<Integer,Integer,Integer> biFunctional=(a,b)->a+b;
	System.out.println(biFunctional.apply(10,20));
	}
}
```

Outputs:

```
30
```

Descriptive Scenario 2:
Task: Create a lambda expression that takes no arguments and prints "Processing complete."
Requirement: Use the predefined functional interface value.

```
package com.lambda.expressions;
import java.util.function.Supplier;
public class Example2_28_04 {
	public static void main(String[] args) {
		Supplier<String> supplier=()->"Processing Complete";
		System.out.println(supplier.get());
```

```
	}
}
```

Descriptive Scenario 3:
Task: Write a lambda expression that checks whether a given integer is even.
Requirement: Use the predefined functional interface Predicate . The lambda should return true if the number is even, otherwise false .

```
package com.lambda.expressions;
import java.util.function.Predicate;
public class Example3_28_04 {
	public static void main(String[] args) {
		Predicate<Integer> predicate=(num)->num%2==0;
		System.out.println(predicate.test(7));
	}
}
```

Descriptive Scenario 4:
Task: Create a lambda expression that takes a String and returns its length.
Requirement: Use the predefined functional interface Function to implement and test this functionality.

```
package com.lambda.expressions;
import java.util.function.Function;
```

```
public class Example4_28_04 {
	public static void main(String[] args) {
		Function<String,Integer> function=(str)->str.length();
		System.out.println(function.apply("Divya"));
	}
}
```

Descriptive Scenario 5:
Task: Develop a lambda expression that takes a floating-point number (whether it is positive or negative).
Requirement: Use the predefined functional interface message like "Positive" or "Negative".

```
package com.lambda.expressions;
import java.util.function.Consumer;
public class Example5_28_04 {
	public static void main(String[] args) {
		Consumer<Float> consumer=(num)->
			/*if(num>0) {
				System.out.println("Positive");
			}
			else if(num<0){
				System.out.println("Negative");
			}
			else {
				System.out.println("Zero");
			}
		};
		consumer.accept(10.5f);
```

```
            consumer.accept(0.0f);
            consumer.accept(-90.5f);*/

        System.out.println(num>0?"Positive":num<0?"Negative":"Zero");
                    consumer.accept(10.5f);
                    consumer.accept(0.0f);
                    consumer.accept(-90.5f);
        }
}
```