Lambda Expressions

Guide for UIL Written Exam

The following is meant as a guide only. This guide is an introduction to the concept of lambda expressions. Contestants should seek out resources that provide a thorough explanation of the nature and use of lambda expressions that goes beyond this guide.

The following is quoted directly from Wikipedia (https://en.wikipedia.org/wiki/Anonymous_function#Java)

> A lambda expression consists of a comma separated list of the formal parameters enclosed in parentheses, an arrow token (->), and a body. Data types of the parameters can always be omitted, as can the parentheses if there is only one parameter. The body can consist of one statement or a statement block.

```
// with no parameter
() -> System.out.println("Hello, world.")

// with one parameter (this example is an
identity function).
a -> a

// with one expression
(a, b) -> a + b

// with explicit type information
(long id, String name) -> "id: " + id + ", name:"
+ name

// with a code block
(a, b) -> { return a + b; }
```

Before you can begin to use a lambda expression there must first be a functional interface. A functional interface is any interface that contains only one abstract method. Once a functional interface has been defined, a variable of that type can be declared and a lambda expression can be assigned to that variable.

Here are a few examples that demonstrate how to use lambda expressions.

1. Simple output.

```
public class LambdaExamples {

	public static void main(String[] args) {
		FunctionalInterface lamb = s->System.out.print(s);
		lamb.doSomething("UIL");
	}

	interface FunctionalInterface{
		public void doSomething(String str);
	}
}
```

`s->System.out.print(s)` is the lambda expression in this example. `s` serves as the argument that is passed to the formal parameter `str` in the `doSomething` method defined in the `FunctionalInterface` interface. `System.out.print(s);` serves as the implementation of the body of the `doSomething` method. This class will print `UIL` to the console.

2. The body of the lambda expression can be more than one line. If this is the case, the body must be enclosed within curly braces as shown here.

```
FunctionalInterface lamb = s->{System.out.println(s);
                               System.out.print(s.length());};
lamb.doSomething("UIL");
```

Changing the body of the lambda expression in this way will cause the program to print

```
UIL
3
```

3. A lambda expression must have the same number of arguments as the abstract method has parameters. When the lambda expression has more than one argument, the arguments must be enclosed in parenthesis. Here is an example.

```
public static void main(String[] args) {
FunctionalInterface lamb = (s,t)->{System.out.println(s);
                                   System.out.println(t);
                                   System.out.println(s+" and "+t);};
lamb.doSomething("UIL","CS");
}

interface FunctionalInterface{
	public void doSomething(String str1,String str2);
}
```

The output for this code segment would be

```
UIL
CS
UIL and CS
```

4. One final example using this code segment shows how to create a lambda expression with no arguments and a one line body.

```
public static void main(String[] args) {
      FunctionalInterface lamb = ()->System.out.println("UIL and CS");
      lamb.doSomething();
      }

interface FunctionalInterface{
      public void doSomething();
}
```

Of course, this code will print `UIL and CS`.

The java.util.function package.

The Java API describes the java.util.function package like this.

> "The interfaces in this package are general purpose functional interfaces used by the JDK, and are available to be used by user code as well. While they do not identify a complete set of function shapes to which lambda expressions might be adapted, they provide enough to cover common requirements."

A list of these interfaces can be found here:
https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/function/package-summary.html

If the programmer needs to accomplish one of the common tasks covered by the interfaces available in the java.util.function package they only need to import the package and then use the interfaces.

The UIL Written Exam will be limited to the following interfaces in the java.util.function package:

- Predicate and BiPredicate
- Function and BiFunction
- Supplier
- Consumer and BiConsumer

A description of these interfaces will be available in the STANDARD CLASSES AND INTERFACES – SUPPLEMENTAL REFERENCE supplied to each contestant for each contest.

Examples using Consumer

The Java API describes the Consumer interface as follows:

> `public interface` **`Consumer<T>`**
> Represents an operation that accepts a single input argument and returns no result. Unlike most other functional interfaces, `Consumer` is expected to operate via side-effects.
>
> This is a functional interface whose functional method is `accept(Object)`.

The method description is

> `void accept(T t)`
> Performs this operation on the given argument.
>
> **Parameters:** `t` - the input argument

1. Example that just prints.

```
import java.util.function.Consumer;
public class LambdaExamples {
	public static void main(String[] args) {
		Consumer<String> print = s -> System.out.print(s);
		print.accept("Hello!");
	}
}
```

First the Consumer interface must be imported. A variable named print of type Consumer that is parameterized to be a String is declared and the lambda expression `s -> System.out.print(s);` is assigned to that variable. The last line uses the object print to call the method accept and the string "Hello!" is printed in the console.

2. Example to do some arithmetic.

```
import java.util.function.Consumer;
public class LambdaExamples {
	public static void main(String[] args) {
		int num = 10;
		Consumer<Integer> print = i -> {i++;System.out.print(i);};
		print.accept(num);
	}
}
```

This example accepts an Integer, adds one to it and prints the result. In this case, 11.

forEach method

Most data structures that are of type `Collection` contain a `forEach` method that accepts a `Consumer` object as an argument. The `forEach` method will then perform the action specified by the lambda expression assigned to the argument on all elements in the `Collection`.

We can use the `forEach` method to simply print all of the elements in an `ArrayList`.

```
import java.util.function.Consumer;
import java.util.ArrayList;
public class LambdaExamples {
	public static void main(String[] args) {
		ArrayList<Integer> list = new ArrayList<Integer>();
		list.add(5);list.add(12);list.add(18);
		list.add(9);list.add(4);list.add(25);
		list.forEach(i -> System.out.println(i));
	}
}
```

We can change the lambda expression so that our code will print each value in an `ArrayList` times 2.

```
list.forEach(i -> {i*=2;System.out.println(i);});
```