

# Lambda Expressions

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** test [~/Desktop/test] - .../src/lab/Example1.java
- Toolbar:** Includes tabs for Example1.java, Example2.java, Example2Ans.java, Example3.java, Example3Ans.java, ExampleFirst.java, and Exercise4.java.
- Project Explorer:** Shows a file tree with Example1.java selected.
- Code Editor:** Displays the following Java code:

```
1 package lab;
2
3 @FunctionalInterface
4 interface MyFunctionalInterface {
5     //A method with no parameter
6     public String sayHello();
7 }
8
9 public class Example1 {
10
11     public static void main(String args[]) {
12         // lambda expression
13         MyFunctionalInterface msg = () -> {
14             return "Hello";
15         };
16         System.out.println(msg.sayHello());
17     }
18 }
19
20
21
```

The code defines a functional interface `MyFunctionalInterface` with a single method `sayHello`. It then creates a lambda expression that returns the string "Hello" and assigns it to the variable `msg`. Finally, it prints the value of `msg.sayHello()`.

**Bottom Status Bar:** Shows the following information:  
Event Log (1)  
Terminal, Zeppelin, Spark monitoring, Messages, Run, TODO  
Build completed successfully in 6 s 107 ms (5 minutes ago)  
9:14 CRLF UTF-8 4 spaces AWS: No credentials selected

test [~/Desktop/test] - .../src/lab/Example2.java

The screenshot shows a Java IDE interface with the following details:

- Title Bar:** test [~/Desktop/test] - .../src/lab/Example2.java
- Toolbar:** Includes tabs for Example2, Run, Stop, Refresh, and Search.
- Code Editor:** Displays the content of Example2.java. The code implements an interface ICalculateService and overrides its calculateTotal method to sum a list of integers. It also contains a main method to demonstrate the functionality.

```
1 package lab;
2 import ...
3
4 interface ICalculateService{
5     public int calculateTotal(List<Integer> listOfMarks);
6 }
7
8 public class Example2 implements ICalculateService{
9     @Override
10    public int calculateTotal(List<Integer> listOfMarks) {
11        int total = 0;
12        for (Integer mark : listOfMarks) {
13            total = total + mark;
14        }
15        return total;
16    }
17
18    public static void main(String[] args) {
19        Example2 example2 = new Example2();
20        ArrayList<Integer> listOfMarks = new ArrayList<>();
21        listOfMarks.add(85);
22        listOfMarks.add(75);
23        listOfMarks.add(60);
24        listOfMarks.add(80);
25        listOfMarks.add(100);
26
27        int total = example2.calculateTotal(listOfMarks);
28        System.out.println("Total is = " + total);
29    }
30
31 }
```

- Sidebar:** Includes sections for Project, AWS Explorer, and Z-Structure.
- Bottom Status Bar:** Shows the build status (Build completed successfully), terminal information, and event log.

test [~/Desktop/test] - .../src/lab/Example2Ans.java

```
1 package lab;
2 import ...
3
4 @FunctionalInterface
5 interface ICalculateServices{
6     public int calculateTotal(List<Integer> listOfMarks);
7 }
8
9
10
11
12 public class Example2Ans{
13
14     public static void main(String[] args) {
15
16         ArrayList<Integer> listOfMarks = new ArrayList<>();
17         listOfMarks.add(85);
18         listOfMarks.add(75);
19         listOfMarks.add(60);
20         listOfMarks.add(80);
21         listOfMarks.add(100);
22
23         ICalculateService iCalculateService = total -> listOfMarks.stream().mapToInt(Integer::intValue).sum();
24
25         System.out.println("Total is " + iCalculateService.calculateTotal(listOfMarks));
26     }
27
28 }
```

Example2Ans : main()

Terminal Zeppelin Spark monitoring Messages Run TODO Event Log

Build completed successfully in 6 s 107 ms (4 minutes ago)

17:29 CRLF UTF-8 Tab\* AWS: No credentials selected

test [~/Desktop/test] - .../src/lab/Example3.java

```
1 package lab;
2
3 import ...
4
5 @FunctionalInterface
6 interface IAverageServices {
7     public int getAverage(List<Integer> listOfMarks);
8 }
9
10 public class Example3 {
11
12     public static void main(String[] args) {
13
14         ArrayList<Integer> listOfMarks = new ArrayList<>();
15         listOfMarks.add(90);
16         listOfMarks.add(80);
17         listOfMarks.add(70);
18         listOfMarks.add(60);
19         listOfMarks.add(100);
20
21         IAverageServices iAverageServices =
22             total -> (listOfMarks.stream().mapToInt(Integer::intValue).sum() /
23                         (listOfMarks.size()));
24
25         System.out.println("Average is " + iAverageServices.getAverage(listOfMarks));
26     }
27
28 }
29
30 }
```

Example3 > main()

Terminal Zeppelin Spark monitoring Messages Run TODO Event Log

Build completed successfully in 6 s 107 ms (4 minutes ago)

27:1 CRLF UTF-8 Tab\* AWS: No credentials selected

test [~/Desktop/test] - .../src/lab/Example3Ans.java

```
1 package lab;
2
3 import ...
4
5 interface IAverageService {
6     public int getAverage(List<Integer> listOfMarks);
7 }
8
9
10 public class Example3Ans implements IAverageService{
11
12     @Override
13     public int getAverage(List<Integer> listOfMarks) {
14
15         int total = 0;
16         for (Integer mark : listOfMarks) {
17             total = total + mark;
18         }
19         return (total/listOfMarks.size());
20     }
21
22
23     public static void main(String[] args) {
24
25         Example3Ans example3Ans = new Example3Ans();
26
27         ArrayList<Integer> listOfMarks = new ArrayList<-->();
28         listOfMarks.add(90);
29         listOfMarks.add(80);
30         listOfMarks.add(70);
31         listOfMarks.add(60);
32         listOfMarks.add(100);
33
34         System.out.println("Average is = " + example3Ans.getAverage(listOfMarks));
35     }
36
37 }
```

AWS Explorer

Z:Structure

Example3Ans

Terminal Zeppelin Spark monitoring Messages Run TODO Event Log

Build completed successfully in 6 s 107 ms (4 minutes ago)

11:14 CRLF UTF-8 Tab\* AWS: No credentials selected

test [~/Desktop/test] - .../src/lab/ExampleFirst.java

```
1 package lab;
2
3 import java.util.stream.IntStream;
4
5 public class ExampleFirst {
6
7     public static void main(String[] args) {
8         loopOldWay();
9         System.out.println();
10        loopNewWay();
11    }
12
13    public static void loopOldWay() {
14
15        for (int num = 1; num <= 10; num++) {
16            System.out.println("loop old print " + num);
17        }
18    }
19
20    public static void loopNewWay() {
21
22        IntStream.iterate( seed: 10, num -> num - 1).limit(10).
23        forEach(num -> System.out.println("loop new print " + num));
24    }
25
26}
```

ExampleFirst

Terminal Zeppelin Spark monitoring Messages Run TODO Event Log

Build completed successfully in 6 s 107 ms (4 minutes ago)

5:14 CRLF UTF-8 Tab\* AWS: No credentials selected

```

test [~/Desktop/test] - .../src/lab/Exercise4.java
Example1.java Example2.java Example2Ans.java Example3.java Example3Ans.java ExampleFirst.java Exercise4.java
1 package lab;
2
3 import java.util.stream.IntStream;
4
5 public class Exercise4 {
6
7     public static void main(String[] args) {
8
9         System.out.println("Threads old approach");
10
11         Runnable r1 = new Runnable() {
12             @Override
13             public void run() {
14                 for (int i = 0; i <= 10; i++) {
15                     System.out.println("Runnable Threads Old Way " + i);
16                 }
17             }
18         };
19         new Thread(r1).start();
20
21         System.out.println("Threads new approach");
22
23         Runnable r2 = () -> {
24             IntStream.iterate( seed: 0, i -> i + 1).limit(10).forEach(i -> {
25                 System.out.println("Runnable Threads new way 1 " + i);
26             });
27         };
28         new Thread(r2).start();
29
30         new Thread(() -> IntStream.iterate( seed: 0, i -> i + 1).limit(10).forEach(i -> {
31             System.out.println("Threads new way 2 " + i);
32         })).start();
33     }
34
35 }

```

AWS Explorer

Z-Structure

Exercise4

Terminal Zeppelin Spark monitoring Messages Run TODO Event Log

Build completed successfully in 6 s 107 ms (4 minutes ago)

5:14 CRLF UTF-8 Tab\* AWS: No credentials selected

## Solution for Multiple Inheritance

```

interface ITestable1{
    public default void readAnnotations(){
        System.out.println("Read Annotations 01");
    }
}

interface ITestable2{
    public default void readAnnotations(){
        System.out.println("Read Annotations 02");
    }
}

public class Annotate implements ITestable1, ITestable2{
    public static void main(String[] args) {
        Annotate diff = new Annotate();
        diff.readAnnotations();
    }

    @Override
    public void readAnnotations() {
        ITestable1.super.readAnnotations();
        ITestable2.super.readAnnotations();
    }
}

```

Activate Windows  
Copyright © 2014 Microsoft Windows

eduscope Lecture Capture

SLIIT - Faculty of Computer Science

## Summary of Method References & Lambda Expressions



Type	Syntax	Method Reference	Lambda expression
Reference to a static method	<code>Class::staticMethod</code>	<code>String::valueOf</code>	<code>s -&gt; String.valueOf(s)</code>
Reference to an instance method of a particular object	<code>instance::instanceMethod</code>	<code>s::toString</code>	<code>O -&gt; "string".toString()</code>
Reference to an instance method of an arbitrary object of a particular type	<code>Class:instanceMethod</code>	<code>String::toString</code>	<code>s -&gt; s.toString()</code>
Reference to a constructor	<code>Class::new</code>	<code>String::new</code>	<code>O -&gt; new String()</code>

## Summary of Method References



Method Reference Type	Syntax	Example
static	<code>ClassName::StaticMethodName</code>	<code>String::valueOf</code>
constructor	<code>ClassName::new</code>	<code>ArrayList::new</code>
specific object instance	<code>objectReference::MethodName</code>	<code>x::toString</code>
arbitrary object of a given type	<code>ClassName::InstanceMethodName</code>	<code>Object::toString</code>

## Method References



### Four Types of Method References

- 1) Method reference to an **instance method** of an object – `object::instanceMethod`
- 2) Method reference to a **static method** of a class – `Class::staticMethod`
- 3) Method reference to an **instance method** of an arbitrary object of a particular type – `Class::instanceMethod`
- 4) Method reference to a **constructor** – `Class::new`

# 1. Method Reference to an instance method

The screenshot shows a Java IDE interface with two code editor panes and a console window.

**Code Editor 1 (Left):**

```
package method.reference;

interface IReference {
    void display();
}

public class First {
    public void myMethod() {
        System.out.println("Instance Method");
    }

    public static void main(String[] args) {
        First first = new First();
        IReference iReference = first::myMethod;
        iReference.display();
    }
}
```

**Code Editor 2 (Bottom Left):**

```
1 interface Test{
2     int m();
3
4     default void display(){
5         System.out.println("Display");
6     }
7
8     static void staticMethod(){
9         System.out.println("Static method");
10    }
11 }
12
13 public class Sample1 {
14
15     interface IPerson{
16         double calculate(double no1, double no2);
17     }
18
19     public static void main(String[] args) {
20         Test test = () -> 42;
21         System.out.println(test.m());
22         test.display();
23         Test.staticMethod();           I
24
25
26
27         IPerson add = (no1, no2) -> {return (no1 + no2);};
28         IPerson min = (no1, no2) -> {return (no1 - no2);};
29         IPerson mul = (no1, no2) -> {return (no1 * no2);};
30         IPerson dev = (no1, no2) -> {return (no1 / no2);};
31
32
33 }
```

**Console Window (Top Right):**

```
<terminated> First [Java Application] C:\Program Files\Java\jdk\bin\java First
Instance Method
```

**Code Editor 3 (Bottom Right):**

```
1 Prob... 2 Java...
<terminated> Sample1
#2
Display
Static method
```

**Bottom Status Bar:**

eduscope Lect

The screenshot shows a Java development environment with three tabs open: Sample1.java, Lambda.java, and Binding.java. The Sample1.java tab is active and displays the following code:

```
1 interface Test{
2     int m();
3
4     default void display(){
5         System.out.println("Display");
6     }
7 }
8
9 public class Sample1 {
10
11     interface IPerson{
12         double calculate(double no1, double no2);
13     }
14
15     public static void main(String[] args) {
16         IPerson add = (no1, no2) -> {return (no1 + no2);};
17         IPerson min = (no1, no2) -> {return (no1 - no2);};
18         IPerson mul = (no1, no2) -> {return (no1 * no2);};
19         IPerson dev = (no1, no2) -> {return (no1 / no2);};
20
21         System.out.println(add.calculate(10, 20.5));
22         System.out.println(min.calculate(10, 20.5));
23         System.out.println(mul.calculate(10, 20.5));
24         System.out.println(dev.calculate(10, 20.5));
25
26         Test test = () -> 42;
27         System.out.println(test.m());
28     }
29 }
30
31 }
```

The code defines an interface `Test` with a method `m()`. It also contains a `default void display()` method that prints "Display". The `Sample1` class implements `IPerson` with methods for addition, subtraction, multiplication, and division. It also contains a `main` method that prints the results of these operations and calls the `m()` method of the `Test` interface.

Discover Your Future

```

Lambda.java X MethodReference.java DefaultMethods.java Anonymous.java Problems Console X
<terminated> Lambda [Java App]
10 + 5 = 15
10 - 5 = 5
10 x 5 = 50
10 / 5 = 2

1 interface MathOperation {
2     int operation(int a, int b);
3 }
4
5 public class Lambda {
6
7     public static void main(String[] args) {
8
9         Lambda lambda = new Lambda();
10
11         // with type declaration
12         MathOperation addition = (int a, int b) -> a + b;
13
14         // with out type declaration
15         MathOperation subtraction = (a, b) -> a - b;
16
17         // with return statement along with curly braces
18         MathOperation multiplication = (int a, int b) -> {
19             return a * b;
20         };
21
22         // without return statement and without curly braces
23         MathOperation division = (int a, int b) -> a / b;
24
25         System.out.println("10 + 5 = " + lambda.operate(10, 5, addition));
26         System.out.println("10 - 5 = " + lambda.operate(10, 5, subtraction));
27         System.out.println("10 x 5 = " + lambda.operate(10, 5, multiplication));
28         System.out.println("10 / 5 = " + lambda.operate(10, 5, division));
29     }
30
31     private int operate(int a, int b, MathOperation mathOperation) {
32         return mathOperation.operation(a, b);
33     }
34 }

```

Activate Windows  
Go to Settings to activate Windows.

**eduscope Lecture Capture**

SLUIT - Faculty of Computing  
Discover Your

## Lambda Syntax

```

(int x) -> x+1           // Single declared-type argument
(int x) -> { return x+1; } // same as above
(x) -> x+1               // Single inferred-type argument, same as below
x -> x+1                 // Parenthesis optional for single inferred-type case

(String s) -> s.length()   // Single declared-type argument
(Thread t) -> { t.start(); } // Single declared-type argument
s -> s.length()           // Single inferred-type argument
t -> { t.start(); }        // Single inferred-type argument

(int x, int y) -> x+y     // Multiple declared-type parameters
(x,y) -> x+y              // Multiple inferred-type parameters
(x, final y) -> x+y       // Illegal: can't modify inferred-type parameters
(x, int y) -> x+y          // Illegal: can't mix inferred and declared types

```



**eduscope Lecture Capt**



The image shows a Java code editor window with three tabs. The top tab is 'Sample1.java'.

```

1 interface IPerson{
2     double calculate(double no1, double no2);
3 }
4
5 interface Test{
6     int m();
7 }
8 public class Sample1 {
9
10 |
11     public static void main(String[] args) {
12         IPerson add = (no1, no2) -> {return (no1 + no2);};
13         IPerson min = (no1, no2) -> {return (no1 - no2);};
14         IPerson mul = (no1, no2) -> {return (no1 * no2);};
15         IPerson dev = (no1, no2) -> {return (no1 / no2);};
16
17         System.out.println(add.calculate(10, 20.5));
18         System.out.println(min.calculate(10, 20.5));
19         System.out.println(mul.calculate(10, 20.5));
20         System.out.println(dev.calculate(10, 20.5));
21
22         Test test = () -> 42;
23         System.out.println(test.m());
24
25     }
26
27 }
28

```

The second tab is also 'Sample1.java'.

```

1 interface IPerson{
2     double calculate(double no1, double no2);
3 }
4 public class Sample1 {
5
6     public static void main(String[] args) {
7         IPerson add = (no1, no2) -> {return (no1 + no2);};
8         IPerson min = (no1, no2) -> {return (no1 - no2);};
9         IPerson mul = (no1, no2) -> {return (no1 * no2);};
10        IPerson dev = (no1, no2) -> {return (no1 / no2);};
11
12        System.out.println(add.calculate(10, 20.5));
13        System.out.println(min.calculate(10, 20.5));
14        System.out.println(mul.calculate(10, 20.5));
15        System.out.println(dev.calculate(10, 20.5));
16    }
17
18 }
19

```

The third tab is 'Sample1.java'.

```

1 interface IPerson{
2     double calculate(double no1, double no2);
3 }
4 public class Sample1 {
5
6     public static void main(String[] args) {
7         IPerson iPerson = (no1, no2) -> {return (no1 + no2);};
8
9         System.out.println(iPerson.calculate(10, 20.5));
10    }
11
12 }
13

```

On the right side of the editor, there is a terminal window showing the output of the Java application:

```

<terminated> Sample1 [Java Application] C:\Users\...
80.5
-10.5
205.0
0.4878048780487805

```

Below the terminal, another terminal window shows:

```

<terminated> Sample1 [Java Application] C:\Users\...
30.5

```

The screenshot shows a Java code editor with two tabs: 'Sample1.java' and 'Sample1.java'. Both tabs contain the same Java code, demonstrating the use of a lambda expression to implement an interface.

```
Sample1.java
1 interface IPerson{
2     void display(String name);
3 }
4 public class Sample1 {
5
6     public static void main(String[] args) {
7         IPerson iPerson = (name) -> {System.out.println("Your name is = " + name);}
8
9         iPerson.display("Udara");
10    }
11
12 }
13
```

```
Sample1.java
1 interface IPerson{
2     void display(String name);
3 }
4 public class Sample1 {
5
6     public static void main(String[] args) {
7         IPerson iPerson = name -> System.out.println("Your name is = " + name);
8
9         iPerson.display("Udara");
10    }
11
12 }
13
```

The code defines an interface `IPerson` with a single method `display`. The `Sample1` class implements this interface. In the `main` method, it creates a lambda expression that prints the string "Your name is = " followed by the argument `name`. This lambda expression is then assigned to the `iPerson` variable and called with the argument "Udara".