

# Lambda Expression

Bright Gee Varghese

# Lambda Expression

- A *lambda expression* is, essentially, an anonymous (that is, unnamed) method.
- Lambda expressions are also commonly referred to as *closures*.
- A *functional interface* is an interface that contains one and only one abstract method.

- *lambda operator* or the *arrow operator*, is  $\rightarrow$ .
- The left side specifies any parameters required by the lambda expression. (If no parameters are needed, an empty parameter list is used.)
- On the right side is the *lambda body*, which specifies the actions of the lambda expression.
- This lambda expression takes no parameters, thus the parameter list is empty. It returns the constant value 123.45

$() \rightarrow 123.45$



`double myMeth() { return 123.45; }`

$(n) \rightarrow (n \% 2) == 0$

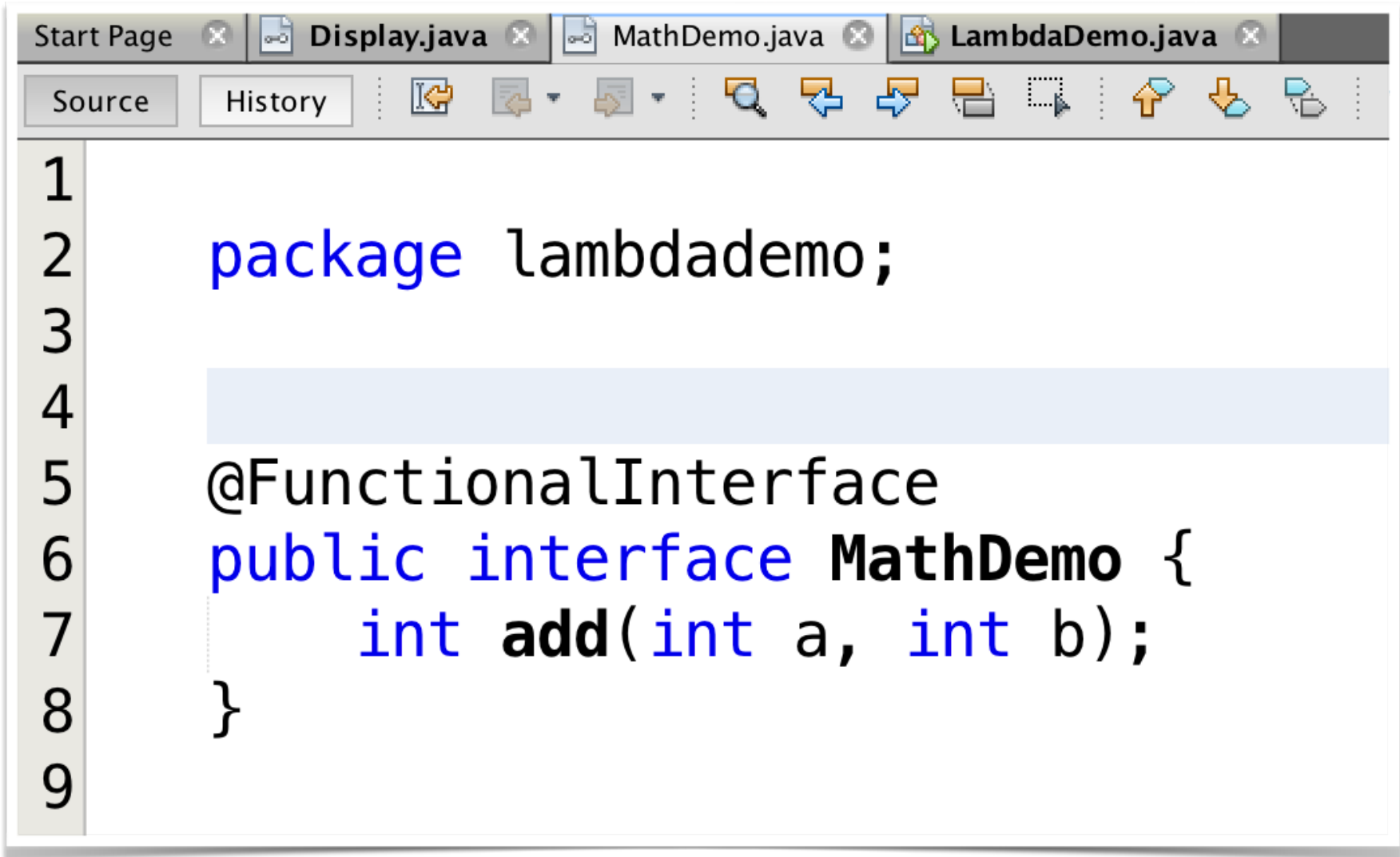


This lambda expression returns **true** if the value of parameter **n** is even.

# Functional Interfaces

- A functional interface is an interface that specifies only one abstract method.

Create an interface which has one abstract method

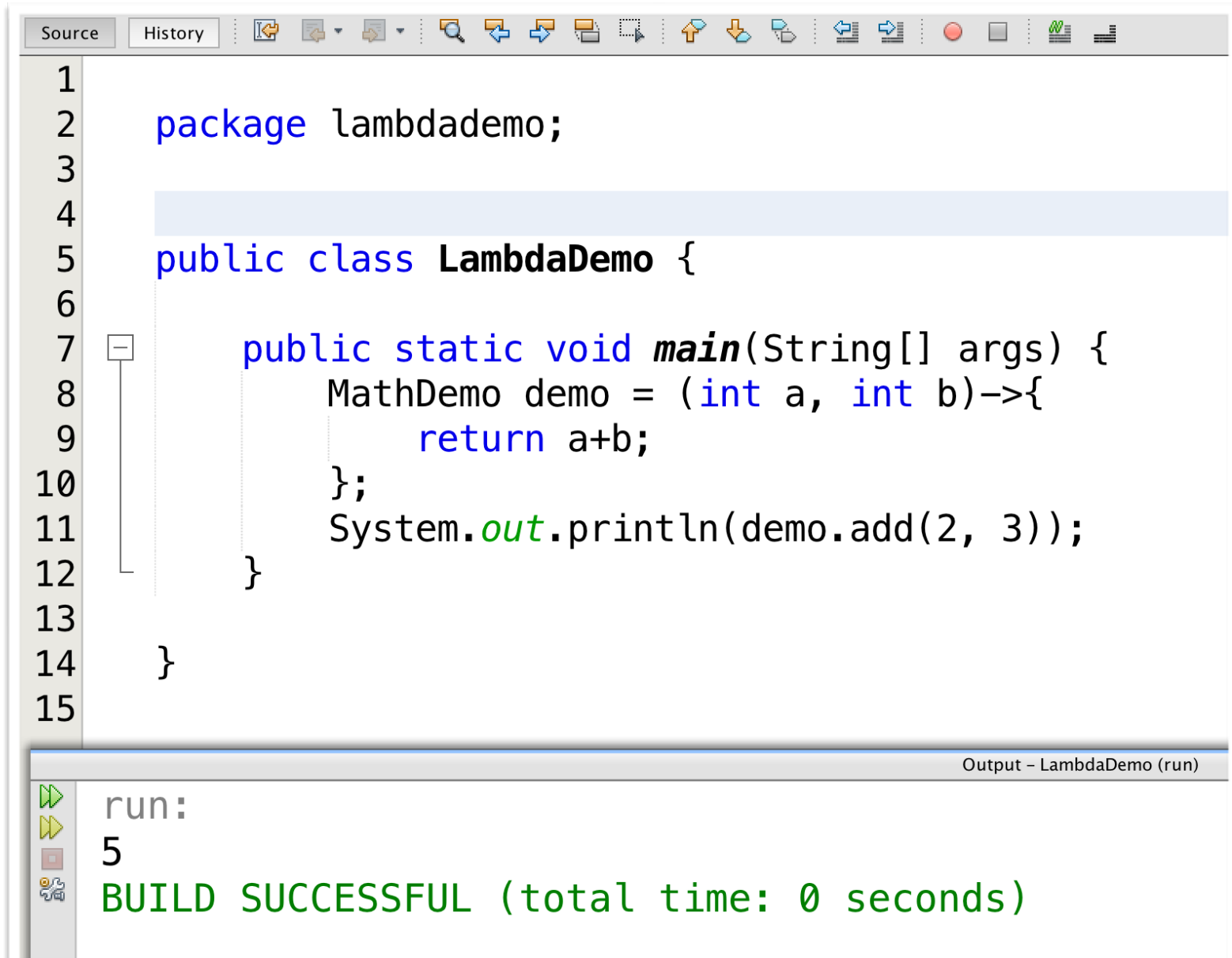


The screenshot shows an IDE window with three tabs: 'Start Page', 'Display.java', 'MathDemo.java', and 'LambdaDemo.java'. The 'MathDemo.java' tab is active. The editor displays the following Java code:

```
1  
2 package lambdademmo;  
3  
4  
5 @FunctionalInterface  
6 public interface MathDemo {  
7     int add(int a, int b);  
8 }  
9
```

The code defines a package named 'lambdademmo' and a functional interface named 'MathDemo' with a single abstract method 'add' that takes two integers and returns an integer. The IDE interface includes a toolbar with various icons for navigation and editing.

# With braces and return



The screenshot shows an IDE window with a source code editor and an output console. The source code editor displays the following Java code:

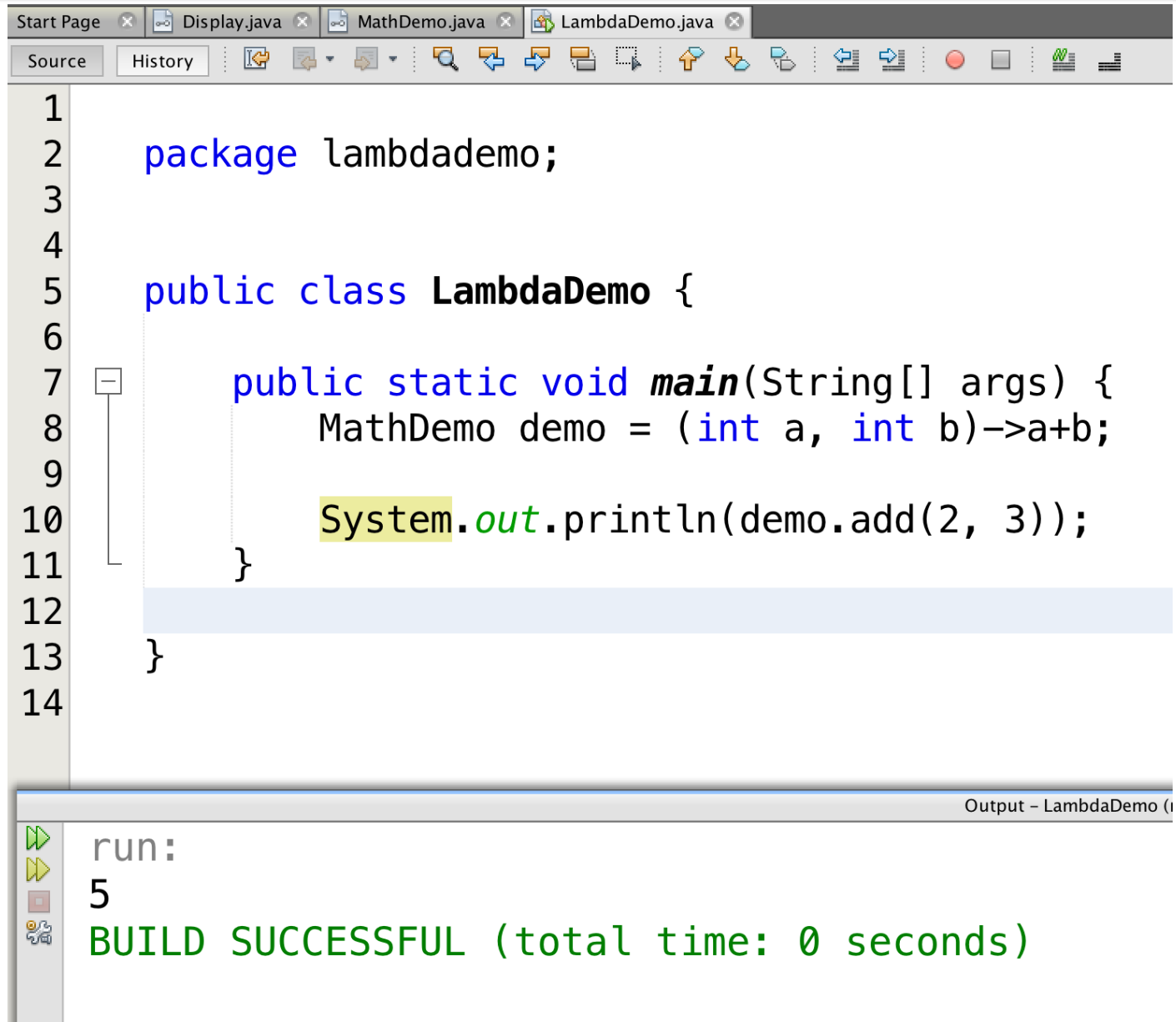
```
1
2 package lambd_demo;
3
4
5 public class LambdaDemo {
6
7     public static void main(String[] args) {
8         MathDemo demo = (int a, int b)->{
9             return a+b;
10        };
11        System.out.println(demo.add(2, 3));
12    }
13
14 }
15
```

The output console at the bottom shows the result of running the code:

```
run:
5
BUILD SUCCESSFUL (total time: 0 seconds)
```

The output console title is "Output - LambdaDemo (run)".

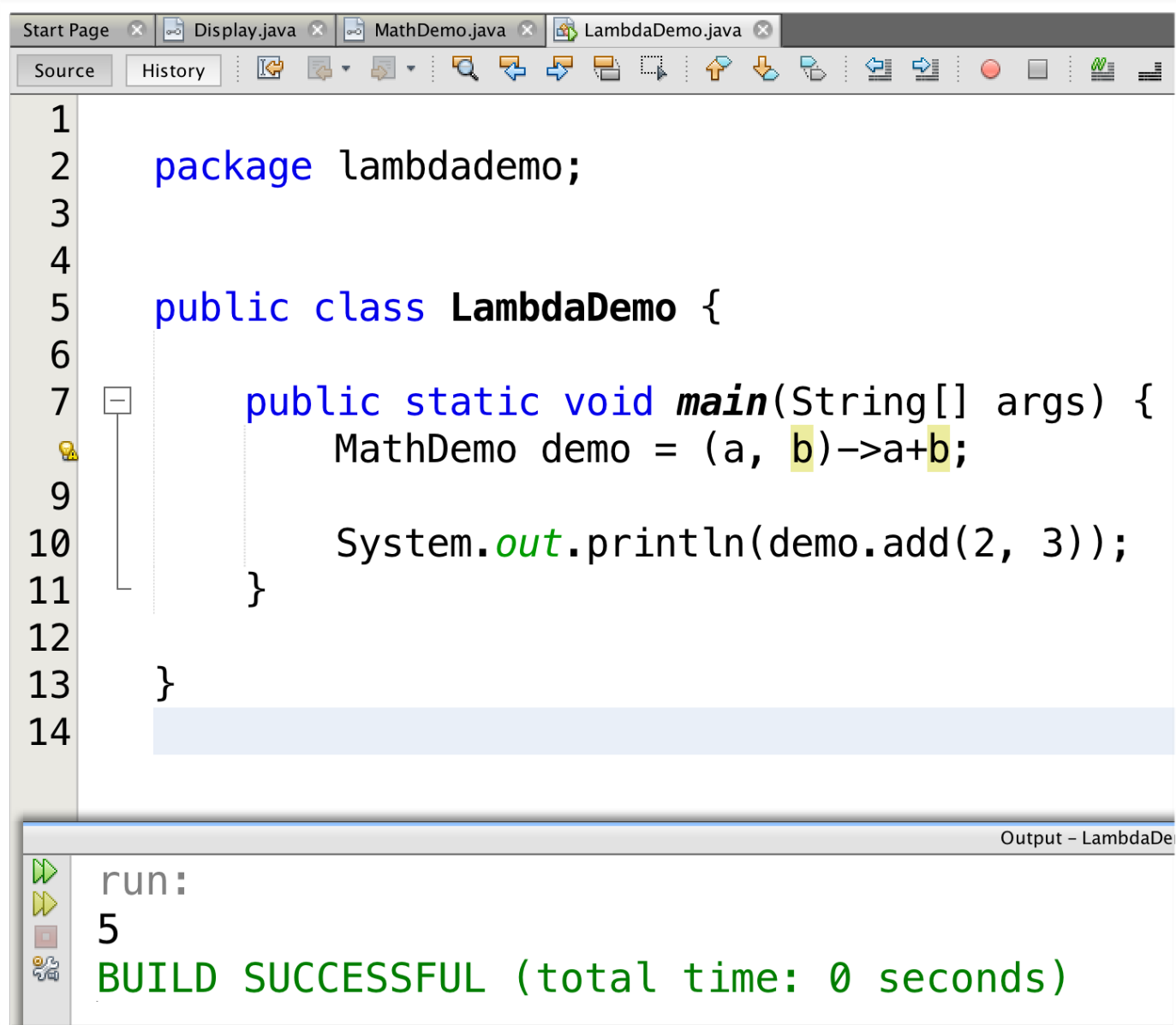
# Without braces and no return



```
1
2 package lambd-demo;
3
4
5 public class LambdaDemo {
6
7     public static void main(String[] args) {
8         MathDemo demo = (int a, int b) -> a + b;
9
10        System.out.println(demo.add(2, 3));
11    }
12
13 }
14
```

run:  
5  
BUILD SUCCESSFUL (total time: 0 seconds)

# Without braces, no return and no parameter type



```
1
2 package lambd_demo;
3
4
5 public class LambdaDemo {
6
7     public static void main(String[] args) {
8         MathDemo demo = (a, b) -> a + b;
9
10        System.out.println(demo.add(2, 3));
11    }
12
13 }
14
```

run:  
5  
BUILD SUCCESSFUL (total time: 0 seconds)



Start Page × Display.java × MathDemo.java × LambdaDemo.java ×

Source

History



1

2

3

4

5

6

7

8

```
package lambd_demo;
```

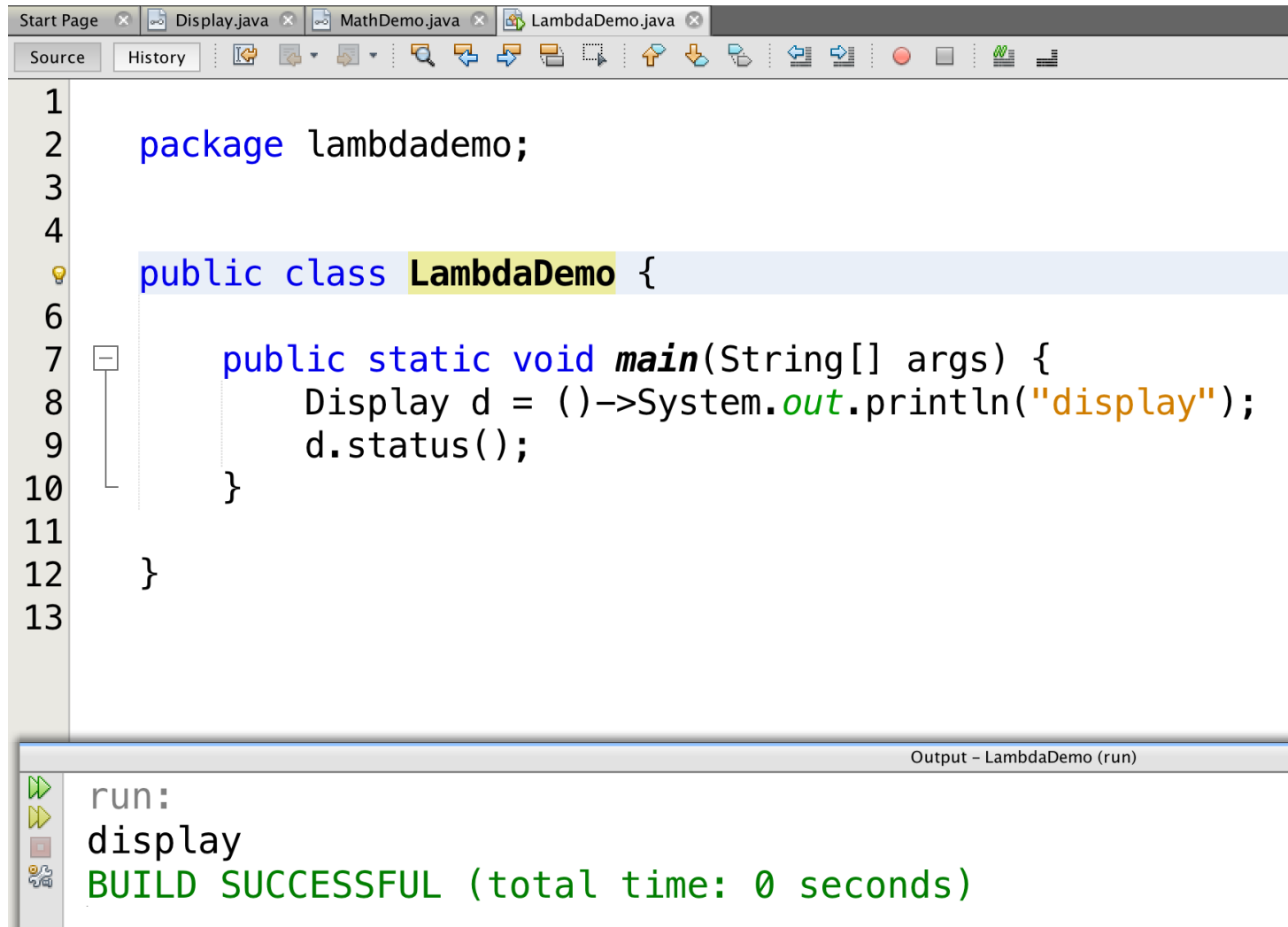
```
@FunctionalInterface
```

```
public interface Display {
```

```
    void status();
```

```
}
```

# optional braces - only if one statement



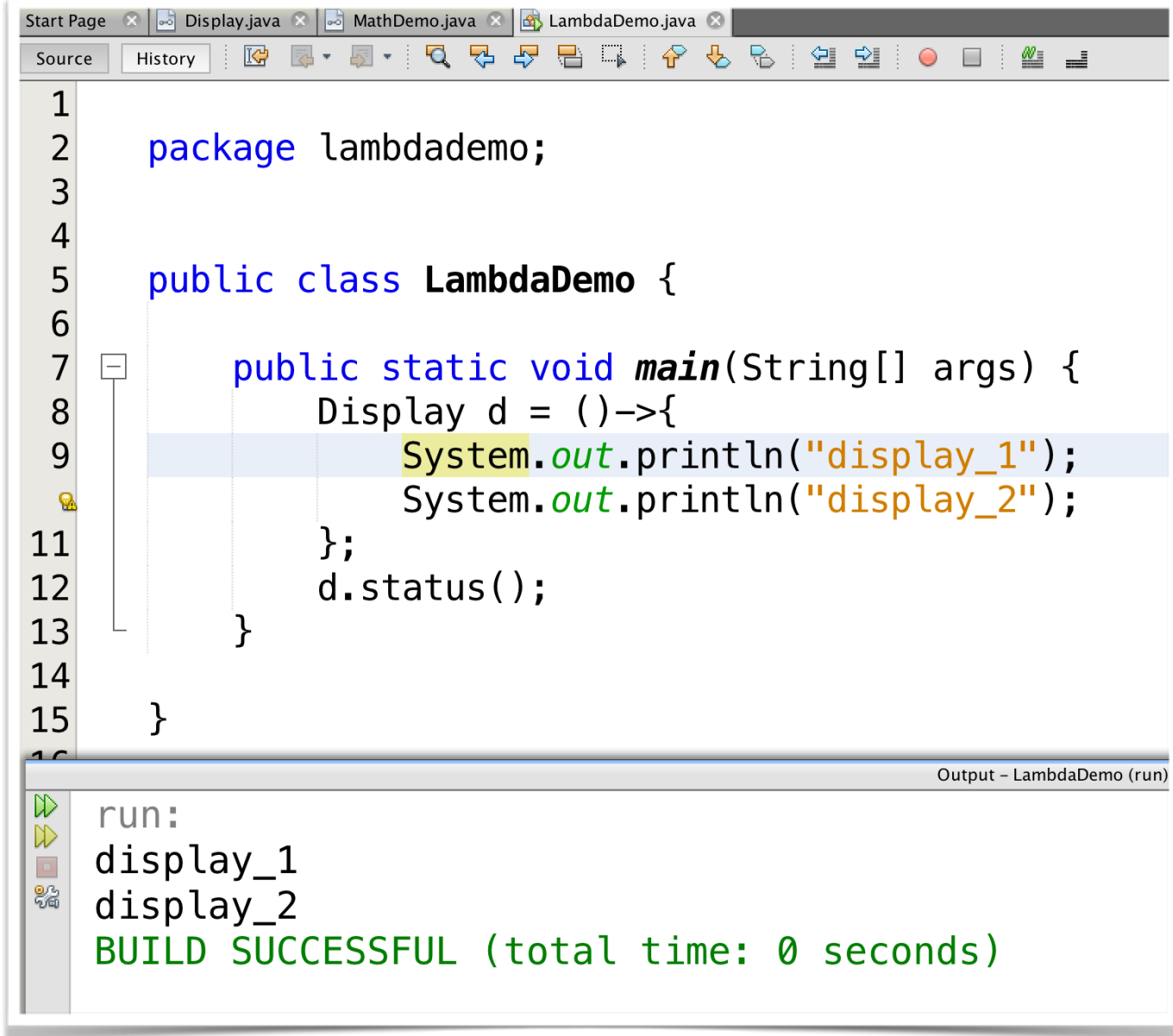
The screenshot shows an IDE with four tabs: Start Page, Display.java, MathDemo.java, and LambdaDemo.java. The LambdaDemo.java tab is active, showing the following code:

```
1  
2 package lambd_demo;  
3  
4  
5 public class LambdaDemo {  
6  
7     public static void main(String[] args) {  
8         Display d = ()->System.out.println("display");  
9         d.status();  
10    }  
11  
12 }  
13
```

The code is syntactically correct. The `main` method contains a lambda expression `()->System.out.println("display");` which is assigned to a `Display` object `d`, and then `d.status();` is called. The IDE's output window at the bottom shows the result of running the program:

```
run:  
display  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Mandatory braces - if more than one statement



The screenshot shows an IDE window with four tabs: Start Page, Display.java, MathDemo.java, and LambdaDemo.java. The LambdaDemo.java tab is active, showing the following code:

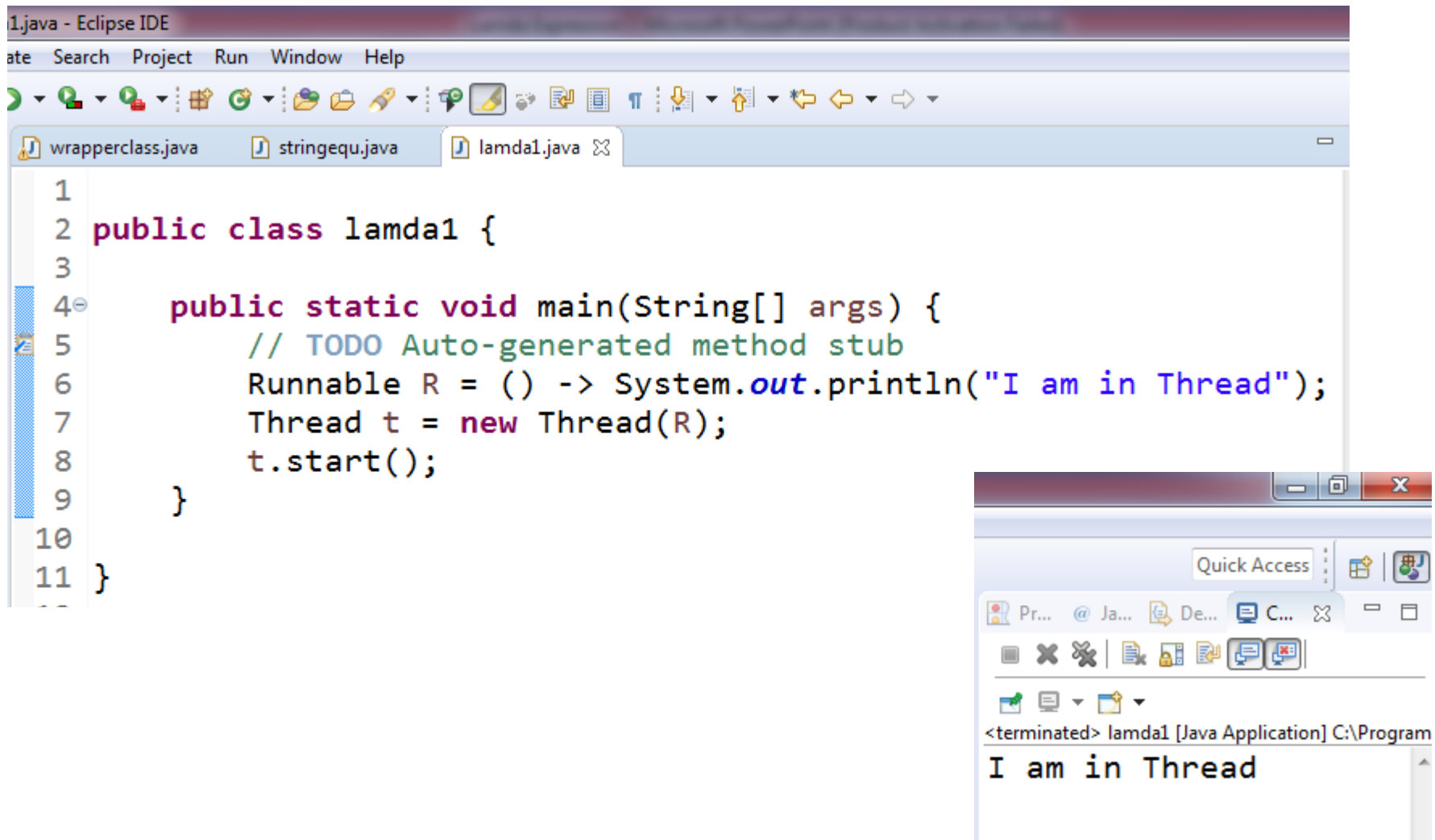
```
1
2 package lambd_demo;
3
4
5 public class LambdaDemo {
6
7     public static void main(String[] args) {
8         Display d = ()->{
9             System.out.println("display_1");
10            System.out.println("display_2");
11        };
12        d.status();
13    }
14
15 }
```

The code is syntactically correct. The IDE's output window at the bottom shows the following output:

```
run:
display_1
display_2
BUILD SUCCESSFUL (total time: 0 seconds)
```

The output window title is "Output - LambdaDemo (run)".

# Thread – Runnable Interface



```
1.java - Eclipse IDE
File Edit Search Project Run Window Help
wrapperclass.java stringequ.java lamda1.java ✕

1
2 public class lamda1 {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         Runnable R = () -> System.out.println("I am in Thread");
7         Thread t = new Thread(R);
8         t.start();
9     }
10
11 }
```

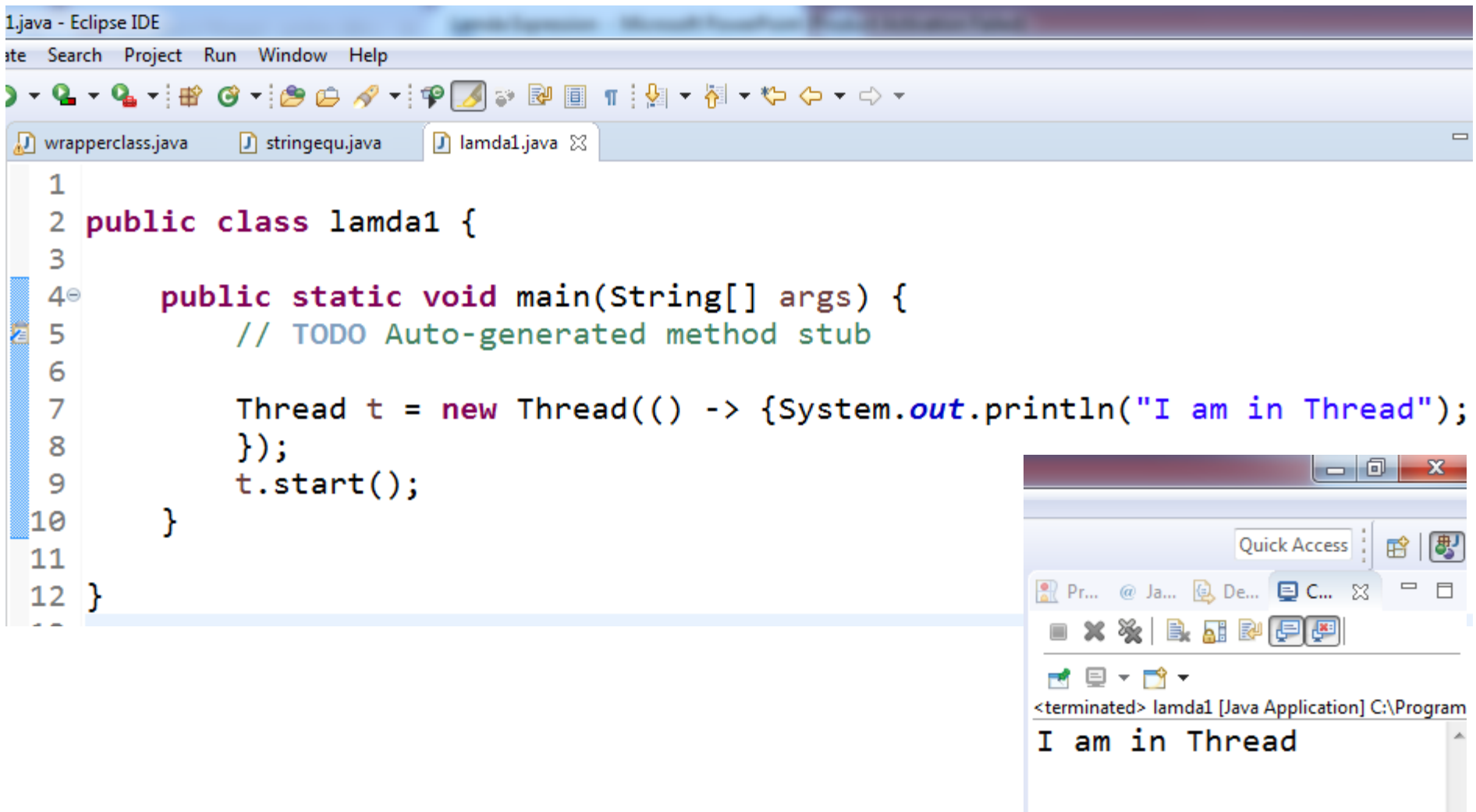
Quick Access

Pr... @ Ja... De... C... ✕

<terminated> lamda1 [Java Application] C:\Program

I am in Thread

# Runnable Interface within Thread



The screenshot displays the Eclipse IDE with the file `lamda1.java` open. The code defines a public class `lamda1` with a `main` method. Inside the `main` method, a `Thread` object `t` is created using a lambda expression that prints "I am in Thread". The thread is then started with `t.start()`.

```
1 public class lamda1 {
2
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         Thread t = new Thread(() -> {System.out.println("I am in Thread");
8         });
9         t.start();
10    }
11
12 }
```

Below the code editor, the Eclipse console window is visible, showing the output of the program:

```
<terminated> lamda1 [Java Application] C:\Program
I am in Thread
```