

# Overview of Java Lambda Expressions

**Douglas C. Schmidt**

**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://www.dre.vanderbilt.edu/~schmidt)**

**Professor of Computer Science**

**Institute for Software  
Integrated Systems**

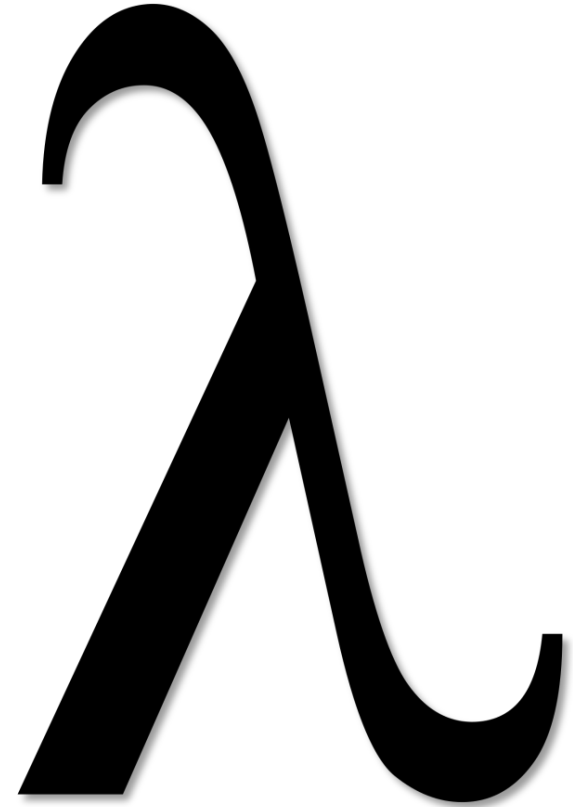
**Vanderbilt University  
Nashville, Tennessee, USA**



# Learning Objectives in this Part of the Lesson

---

- Understand how lambda expressions provide a foundational functional programming feature in Modern Java

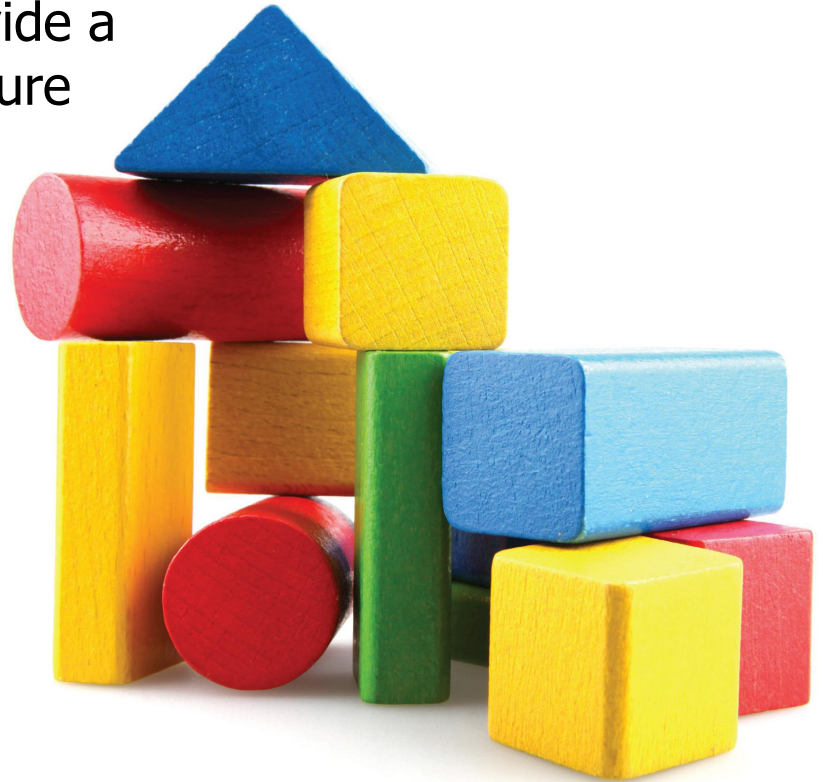


We show simple examples that highlight lambda expression syntax & semantics

# Learning Objectives in this Part of the Lesson

---

- Understand how lambda expressions provide a foundational functional programming feature in Modern Java
  - Know how lambda expressions can be used to represent a wide range of code block usages in Java



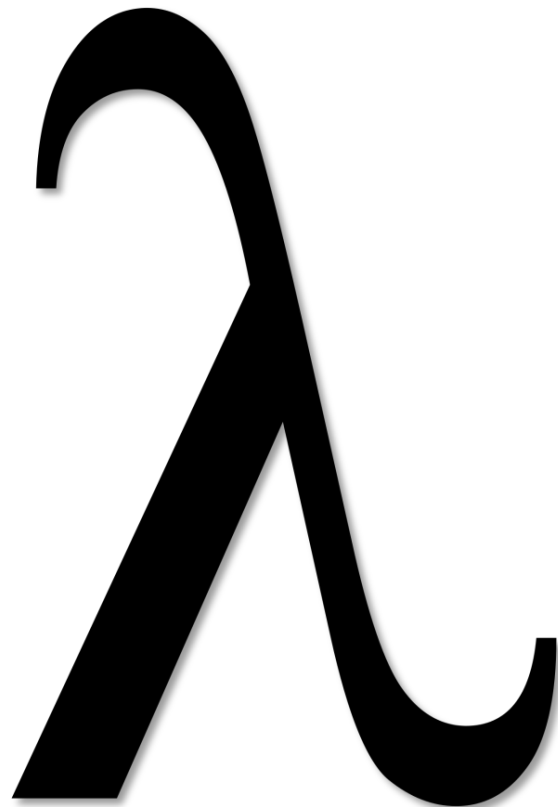
---

# Overview of Java Lambda Expressions

# Overview of Java Lambda Expressions

---

- A *lambda expression* is an unnamed block of code (with optional parameters)



---

See [www.drdoobbs.com/jvm/lambda-expressions-in-java-8/240166764](http://www.drdoobbs.com/jvm/lambda-expressions-in-java-8/240166764)

# Overview of Java Lambda Expressions

---

- A *lambda expression* is an unnamed block of code (with optional parameters)

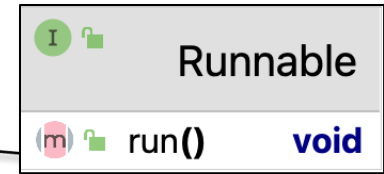
```
Thread t = new Thread(() ->
    System.out.println
    ("hello world"));
```

# Overview of Java Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters)

```
Thread t = new Thread( () ->
    System.out.println
        ("hello world") );
```

*Thread's constructor expects an instance of type Runnable*



# Overview of Java Lambda Expressions

---

- A *lambda expression* is an unnamed block of code (with optional parameters)

*This lambda expression takes no parameters at all, i.e., "()"*

```
Thread t = new Thread( () ->
    System.out.println
    ("hello world"));
```



# Overview of Java Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters)

*This lambda expression takes no parameters at all, i.e., "()"*

```
Thread t = new Thread( () ->
                        System.out.println
                        ("hello world"));
```

**COMING SOON**

We'll show examples later where lambda expressions take parameters

# Overview of Java Lambda Expressions

---

- A *lambda expression* is an unnamed block of code (with optional parameters)

*The arrow is a syntactic construct that separates the (possibly empty) parameter list from the lambda body*

```
Thread t = new Thread(() ->
                        System.out.println
                        ("hello world"));
```

# Overview of Java Lambda Expressions

---

- A *lambda expression* is an unnamed block of code (with optional parameters)

*This lambda Runnable body defines the computation*

```
Thread t = new Thread(() ->  
    System.out.println  
    ("hello world"));
```



# Overview of Java Lambda Expressions

---

- A *lambda expression* is an unnamed block of code (with optional parameters)

```
Thread t = new Thread(() ->  
    System.out.println  
    ("hello world"));
```

*There's no need for curly braces when the lambda body is one expression or is just a "void" method invocation*

# Overview of Java Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters)

*Lambda expressions are very compact since they focus solely on computation(s) to perform*

```
Thread t = new Thread(() ->  
    System.out.println  
    ("hello world"));
```



# Overview of Java Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters)

*Conversely, this anonymous inner class requires more code to write each time*

```
Thread t = new Thread(() ->
    System.out.println
        ("hello world"));
```

VS

```
new Thread(new Runnable() {
    public void run() {
        System.out.println("hello world");
    }});
```



# Overview of Java Lambda Expressions

---

- Lambda expressions can also work with multiple parameters

```
String[] nameArray = {"Barbara", "James", "Mary", "John",  
                      "Robert", "Michael", "Linda", "james", "mary"};
```

```
Arrays.sort(nameArray, (String s, String t) ->  
                      s.compareToIgnoreCase(t));
```

# Overview of Java Lambda Expressions

---

- Lambda expressions can work with multiple parameters, e.g.

```
String[] nameArray = {"Barbara", "James", "Mary", "John",  
                      "Robert", "Michael", "Linda", "james", "mary"};
```



*Array of names represented as strings*

```
Arrays.sort(nameArray, (String s, String t) ->  
                    s.compareToIgnoreCase(t));
```



# Overview of Java Lambda Expressions

---

- Lambda expressions can work with multiple parameters, e.g.

```
String[] nameArray = {"Barbara", "James", "Mary", "John",  
                      "Robert", "Michael", "Linda", "james", "mary"};
```

```
Arrays.sort(nameArray, (String s, String t) ->  
                      s.compareToIgnoreCase(t));
```

*The sort comparison operator is a lambda expression that ignores case*

# Overview of Java Lambda Expressions

---

- A *lambda expression* can be stored, passed, & executed later

```
Runnable r =  
    () -> System.out.println("hello world");
```

# Overview of Java Lambda Expressions

---

- A *lambda expression* can be **stored**, passed, & executed later

```
Runnable r =  
    () -> System.out.println("hello world");
```



*You can store a lambda expression into a variable*

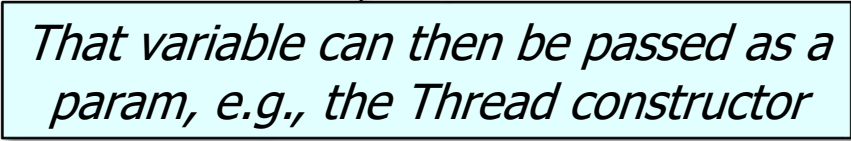
# Overview of Java Lambda Expressions

---

- A *lambda expression* can be stored, **passed**, & executed later

```
Runnable r =  
    () -> System.out.println("hello world");
```

```
Thread t = new Thread(r);
```



*That variable can then be passed as a param, e.g., the Thread constructor*

# Overview of Java Lambda Expressions

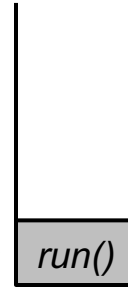
- A *lambda expression* can be stored, passed, & **executed later**

```
Runnable r =
```

```
    () -> System.out.println("hello world");
```

```
Thread t = new Thread(r);
```

```
t.start();
```



Runtime  
thread  
stack

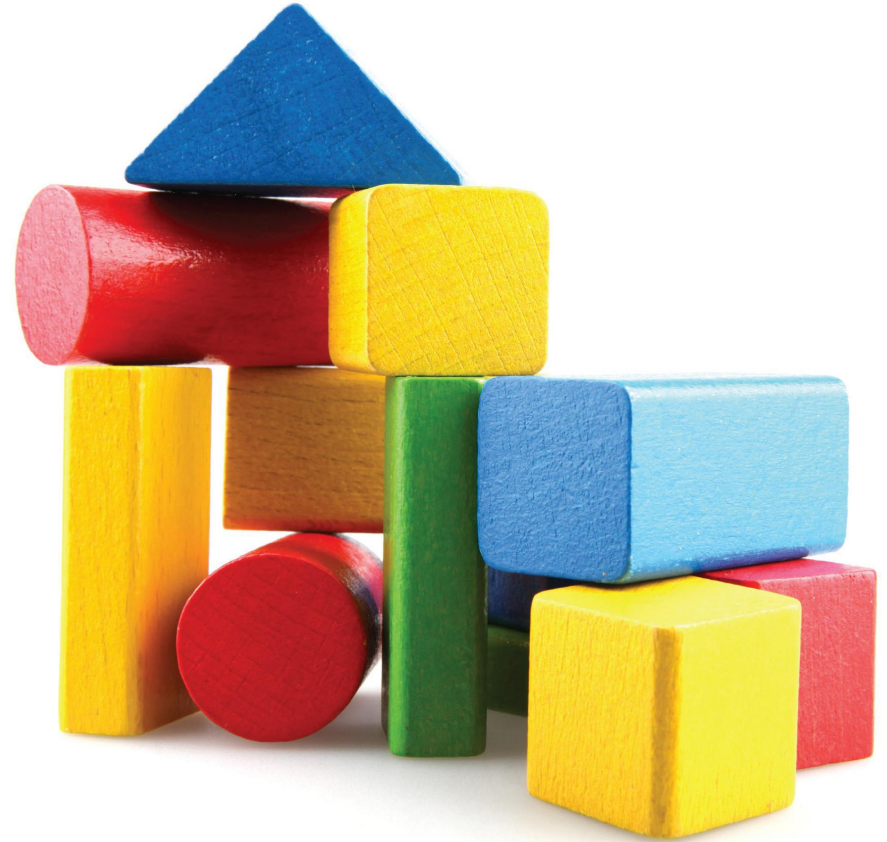
*This computation executes in a new Java thread when it's started*

---

# Java Lambda Expressions Cover a Wide Range of Usages

# Java Lambda Expressions Cover a Wide Range of Usages

- Lambda expressions can be used to represent a wide range of code block usages in Java



# Java Lambda Expressions Cover a Wide Range of Usages

- Lambda expressions can be used to represent a wide range of code block usages in Java, e.g.
- Runnable & Callable tasks

*Submit a lambda expression that concurrently multiplies two BigFraction objects*

```
String f1 =  
    "62675744/15668936";  
String f2 = "609136/913704";  
  
Future<BigFraction> f =  
    commonPool().submit(() -> {  
        BigFraction bf1 =  
            new BigFraction(f1);  
        BigFraction bf2 =  
            new BigFraction(f2);  
        return bf1.multiply(bf2);  
    });  
...
```



# Java Lambda Expressions Cover a Wide Range of Usages

- Lambda expressions can be used to represent a wide range of code block usages in Java, e.g.
  - Runnable & Callable tasks
  - Comparator & filter functions for collections

```
List<String> fruits =  
    new ArrayList<>();
```

```
fruits  
    .add(new String("Apple"));  
fruits  
    .add(new String("Orange"));  
fruits  
    .add(new String("Pear"));  
fruits  
    .add(new String("Banana"));
```

*Sort a List of Java  
String objects*

```
fruits.sort((f1, f2) ->  
            f1.compareTo(f2))
```

# Java Lambda Expressions Cover a Wide Range of Usages

- Lambda expressions can be used to represent a wide range of code block usages in Java, e.g.
  - Runnable & Callable tasks
  - Comparator & filter functions for collections
  - Event listeners & handlers

*Post a "toast" on Android when a user clicks a button*

```
public class MainActivity
    extends Activity {
    protected void onCreate
        (Bundle bundle) {
        Button button =
            findViewById(R.id.button);

        Button
            .setOnClickListener
                (view -> { Toast
                    .makeText(this,
                        "Button clicked!",
                            LENGTH_SHORT) .show();
                });
    }
}
```

# Java Lambda Expressions Cover a Wide Range of Usages

- Lambda expressions can be used to represent a wide range of code block usages in Java, e.g.
  - Runnable & Callable tasks
  - Comparator & filter functions for collections
  - Event listeners & handlers
  - Thread pools & concurrency constructs

```
ThreadFactory factory = r -> {  
    Thread t = new Thread(r);  
    t.setPriority  
        (Thread.NORM_PRIORITY);  
    return t;  
};
```



*Create a new Thread with  
the designated priority*

---

# End of Overview of Java Lambda Expressions