Overview of Java 8 Lambda Expressions & Method References

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Lesson

- Recognize foundational functional programming features in Java 8, e.g.,
  - Lambda expressions
Learning Objectives in this Lesson

- Recognize foundational functional programming features in Java 8, e.g.,
  - Lambda expressions
  - Method (& constructor) references
Learning Objectives in this Lesson

- Recognize foundational functional programming features in Java 8, e.g.,
  - Lambda expressions
  - Method (& constructor) references

Several concise examples are used to showcase foundational Java 8 features
Overview of Lambda Expressions & Method References
Overview of Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later

```java
new Thread(() ->
    System.out.println("hello world"))
    .start();
```
Overview of Lambda Expressions

- A *lambda expression* is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() ->
    System.out.println("hello world")).start();
```

*This lambda expression takes no parameters (i.e., "()") & defines a computation that will run in a separate Java thread*
Overview of Lambda Expressions

- A **lambda expression** is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() ->
    System.out.println("hello world"))
    .start();
```

Lambda expressions are compact since they just focus on computation(s) to perform
Overview of Lambda Expressions

• A **lambda expression** is an unnamed block of code (with optional parameters) that can be stored, passed around, & executed later, e.g.,

```java
new Thread(() ->
    System.out.println("hello world"))
.start();
```

**VS**

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

```java
new Thread(new Runnable() {
    public void run() {
        System.out.println("hello world");
    }
}).start();
```
Overview of Method References

- A method reference is a compact, easy-to-read handle for a method that already has a name.

<table>
<thead>
<tr>
<th>Kind</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::InstanceMethodName</td>
</tr>
<tr>
<td>3. Reference to an instance method of an arbitrary object of a given type</td>
<td>ContainingType::methodName</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
</tr>
</tbody>
</table>

See [docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html](docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html)
A method reference is a compact, easy-to-read handle for a method that already has a name.

Overview of Method References

<table>
<thead>
<tr>
<th>Kind</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::InstanceMethodName</td>
</tr>
<tr>
<td>3. Reference to an instance method of an arbitrary object of a given type</td>
<td>ContainingType::methodName</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
</tr>
</tbody>
</table>
Overview of Method References

- Method references are more compact than anonymous inner classes & lambda expressions
Overview of Method References

- Method references are more compact than anonymous inner classes & lambda expressions, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s,String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }});

VS

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

VS

Arrays.sort(nameArray, String::compareToIgnoreCase);
```

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex1](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex1)
Overview of Method References

- Method references are more compact than anonymous inner classes & lambda expressions, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }
});
```

VS

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

VS

```java
Arrays.sort(nameArray, String::compareToIgnoreCase);
```

Lots of syntax for anonymous inner class
Overview of Method References

- Method references are more compact than anonymous inner classes & lambda expressions, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s,String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }});
```

VS

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

VS

```java
Arrays.sort(nameArray, String::compareToIgnoreCase);
```

See docs.oracle.com/javase/tutorial/java/generics/genTypeInference.html
Overview of Method References

• Method references are more compact than anonymous inner classes & lambda expressions, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }
});
```

VS

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

VS

```java
Arrays.sort(nameArray, String::compareToIgnoreCase);
```

Method reference is even more compact
Overview of Method References

• Method references are more compact than anonymous inner classes & lambda expressions, e.g.,

```java

Arrays.sort(nameArray, new Comparator<String>(){
    public int compare(String s, String t) { return s.toLowerCase().compareTo(t.toLowerCase()); }});

VS

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

VS

Arrays.sort(nameArray, String::compareToIgnoreCase);
```

It’s good practice to use method references whenever you can!
Overview of Method References

• The contents of a collection or array can be printed in various ways

```java
```
Overview of Method References

• The contents of a collection or array can be printed in various ways

```java
```

• System.out.println() can be used to print out an array

```java
System.out.println(Arrays.asList(nameArray));
```

prints

```
[Barbara, James, Mary, John, Linda, Michael, Linda, james, mary]
```

See hwww.tutorialspoint.com/java/util/arrays_aslist.htm
Overview of Method References

• The contents of a collection or array can be printed in various ways
  ```java
  ```

• System.out.println() can be used to print out an array

• Java 8’s forEach() method can be used in conjunction with a stream & method reference to iterate without using a loop
  ```java
  Stream.of(nameArray).forEach(System.out::print);
  ```
  prints
  ```
  BarbaraJamesMaryJohnLindaMichaelLindajamesmary
  ```

End of Overview of Java 8 Lambda Expressions & Method References