Overview of Java 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 Part of the Lesson

• Understand foundational functional programming features in Java, e.g.,
  • Lambda expressions
  • Method (& constructor) references

Several examples showcase foundational Java function programming features
Overview of Method References
Overview of Method References

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

<table>
<thead>
<tr>
<th>Kind</th>
<th>Syntax</th>
<th>Method Reference</th>
<th>Lambda Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
<td>String::valueOf</td>
<td>s -&gt; String.valueOf(s)</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::instanceMethodName</td>
<td>s::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>3. Reference to instance method of an arbitrary object of</td>
<td>ContainingType::methodName</td>
<td>String::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>a given type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
<td>String::new</td>
<td>() -&gt; new String()</td>
</tr>
</tbody>
</table>
Overview of Method References

- A compact, easy-to-read handle for a method that already has a name
- It’s shorthand syntax for a lambda expression that executes one method

<table>
<thead>
<tr>
<th>Kind</th>
<th>Syntax</th>
<th>Method Reference</th>
<th>Lambda Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
<td>String::valueOf</td>
<td>s -&gt; String.valueOf(s)</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::instanceMethodName</td>
<td>s::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>3. Reference to instance method of an arbitrary object of a given type</td>
<td>ContainingType::methodName</td>
<td>String::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
<td>String::new</td>
<td>() -&gt; new String()</td>
</tr>
</tbody>
</table>
Overview of Method References

- A compact, easy-to-read handle for a method that already has a name
- It’s shorthand syntax for a lambda expression that executes one method

<table>
<thead>
<tr>
<th>Kind</th>
<th>Syntax</th>
<th>Method Reference</th>
<th>Lambda Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
<td>String::valueOf</td>
<td>s -&gt; String.valueOf(s)</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::instanceMethodName</td>
<td>s::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>3. Reference to instance method of an arbitrary object of a given type</td>
<td>ContainingType::methodName</td>
<td>String::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
<td>String::new</td>
<td>() -&gt; new String()</td>
</tr>
</tbody>
</table>
Overview of Method References

- A compact, easy-to-read handle for a method that already has a name
- It’s shorthand syntax for a lambda expression that executes one method

<table>
<thead>
<tr>
<th>Kind</th>
<th>Syntax</th>
<th>Method Reference</th>
<th>Lambda Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::&lt;br&gt;staticMethodName</td>
<td>String::valueOf</td>
<td>s -&gt; String.valueOf(s)</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::&lt;br&gt;instanceMethodName</td>
<td>s::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>3. Reference to instance method of an arbitrary object of a given type</td>
<td>ContainingType::&lt;br&gt;methodName</td>
<td>String::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::&lt;br&gt;new</td>
<td>String::new</td>
<td>() -&gt; new String()</td>
</tr>
</tbody>
</table>
Overview of Method References

- A compact, easy-to-read handle for a method that already has a name
- It’s shorthand syntax for a lambda expression that executes one method

<table>
<thead>
<tr>
<th>Kind</th>
<th>Syntax</th>
<th>Method Reference</th>
<th>Lambda Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
<td>String::valueOf</td>
<td>s -&gt; String.valueOf(s)</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::instanceMethodName</td>
<td>s::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>3. Reference to instance method of an arbitrary object of a given type</td>
<td>ContainingType::methodName</td>
<td>String::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
<td>String::new</td>
<td>() -&gt; new String()</td>
</tr>
</tbody>
</table>
### Overview of Method References

- A compact, easy-to-read handle for a method that already has a name
- It’s shorthand syntax for a lambda expression that executes one method

<table>
<thead>
<tr>
<th>Kind</th>
<th>Syntax</th>
<th>Method Reference</th>
<th>Lambda Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reference to a static method</td>
<td>ContainingClass::staticMethodName</td>
<td>String::valueOf</td>
<td>s -&gt; String.valueOf(s)</td>
</tr>
<tr>
<td>2. Reference to an instance method of a particular object</td>
<td>containingObject::instanceMethodName</td>
<td>s::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>3. Reference to instance method of an arbitrary object of a given type</td>
<td>ContainingType::methodName</td>
<td>String::toString</td>
<td>s -&gt; s.toString()</td>
</tr>
<tr>
<td>4. Reference to a constructor</td>
<td>ClassName::new</td>
<td>String::new</td>
<td>() -&gt; new String()</td>
</tr>
</tbody>
</table>
Benefits of Method References
Benefits of Method References

- Method references are more compact than alternative mechanisms

Java method references support more concise “behavior parameterization”

See [blog.indrek.io/articles/java-8-behavior-parameterization](http://blog.indrek.io/articles/java-8-behavior-parameterization)
Benefits of Method References

- Method references are more compact than alternative mechanisms, 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  Method references are even more compact & readable

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

See [www.gravytrain.co.uk/blog/java-8-an-introduction-to-method-references](http://www.gravytrain.co.uk/blog/java-8-an-introduction-to-method-references)
Benefits of Method References

- Method references are more compact than alternative mechanisms, 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

```
Method references also promote code reuse
```

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

The Arrays.sort() implementation doesn’t change, but the params do!
Benefits of Method References

- Method references are more compact than alternative mechanisms, 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  

*Replacing one comparison with another is easy, a la the Strategy pattern.*

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

Benefits of Method References

- Method references are more compact than alternative mechanisms, 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::compareTo);
```

It’s therefore good practice to use method references whenever you can!
Applying Method References in Practice
Method references can be used to print a collection or array in various ways.

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

Array of names represented as strings
Applying Method References in Practice

- Method references can be used to print a collection or array in various ways
  ```java
  ```

- System.out.println() can be used to print out an array
  ```java
  System.out.println(List.of(nameArray));
  ```
  prints
  ```
  [Barbara, James, Mary, John, Linda, Michael, Linda, james, mary]
  ```

See docs.oracle.com/javase/8/docs/api/java/io/PrintStream.html#println
Applying Method References in Practice

- Method references can be used to print a collection or array in various ways

```java
```

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

```java
System.out.println(List.of(nameArray));
```

prints

![Fact] Factory method returns a fixed-size list backed by the array.

[Barbara, James, Mary, John, Linda, Michael, Linda, james, mary]
Applying Method References in Practice

- Method references can be used to print a collection or array in various ways
  ```java
  ```
- System.out.println() can be used to print out an array
- Java’s forEach() methods can be used to print out values of an array

Method references can be used to print a collection or array in various ways:

```java
```

- System.out.println() can be used to print out an array.
- Java’s forEach() methods can be used to print out values of an array, e.g.
  - In conjunction with a stream & method reference:
    ```java
    Stream.of(nameArray).forEach(System.out::println);
    
    prints
    Factory method that creates a stream from an array
    BarbaraJamesMaryJohnLindaMichaelLindajamesmary
    ```

See [docs.oracle.com/javase/9/docs/api/java/util/List.html#immutable](http://docs.oracle.com/javase/9/docs/api/java/util/List.html#immutable)
Applying Method References in Practice

- Method references can be used to print a collection or array in various ways

```java
```

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

- Java’s forEach() methods can be used to print out values of an array, e.g.

  - In conjunction with a stream & method reference

```java
Stream.of(nameArray).forEach(System.out::println);
```

prints

Barbara James Mary John Linda Michael Linda James Mary

Performs method reference action on each stream element

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#forEach
Applying Method References in Practice

- Method references can be used to print a collection or array in various ways

```java
```

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

- Java’s `forEach()` methods can be used to print out values of an array, e.g.
  - In conjunction with a stream & method reference
  ```java
  List.of(nameArray).forEach(System.out::println);
  ```
  prints
  ```
  Factory method converts an array into a list.
  BarbaraJamesMaryJohnLindaMichaelLindajamesmary
  ```

See [docs.oracle.com/javase/9/docs/api/java/util/List.html#of](https://docs.oracle.com/javase/9/docs/api/java/util/List.html#of)
Applying Method References in Practice

- Method references can be used to print a collection or array in various ways
  
  ```java
  ```

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

  - Java’s forEach() methods can be used to print out values of an array, e.g.
    - In conjunction with a stream & method reference
    - In conjunction with a collection (e.g., List)

  ```java
  List.of(nameArray).forEach(System.out::println);
  ```

  prints

  ```
  Performs method reference action on each list element
  ```

  BarbaraJamesMaryJohnLindaMichaelLindajamesmary

See [docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html#forEach](https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html#forEach)
Applying Method References in Practice

• Method references can be used to print a collection or array in various ways
  String[] nameArray = {"Barbara", "James", "Mary", "John", 
                      "Robert", "Michael", "Linda", "james", "mary"};

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

• Java’s forEach() methods can be used to print out values of an array, e.g.
  • In conjunction with a stream & method reference
  • In conjunction with a collection (e.g., List)
  • forEach() on a stream differs slightly from forEach() on a collection

See stackoverflow.com/a/23232560
Method references can be used to print a collection or array in various ways:

```java
```

- System.out.println() can be used to print out an array.
- Java’s forEach() methods can be used to print out values of an array, e.g.
  - In conjunction with a stream & method reference.
  - In conjunction with a collection (e.g., List).
- forEach() on a stream differs slightly from forEach() on a collection.
  - e.g., forEach() ordering is undefined on a stream, whereas it’s defined for a collection.

See stackoverflow.com/a/23232560
End of Overview of Java Method References