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

Kind	Syntax	Method Reference	Lambda Expression
1. Reference to a static method	ContainingClass:: staticMethodName	String::valueOf	s -> String.valueOf(s)
2. Reference to an instance method of a particular object	containingObject:: instanceMethodName	s::toString	s -> s.toString()
3. Reference to instance method of an arbitrary object of a given type	ContainingType:: methodName	String::toString	s -> s.toString()
4. Reference to a constructor	ClassName::new	String::new	() -> new String()

See docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

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

Kind	Syntax	Method Reference	Lambda Expression
1. Reference to a static method	ContainingClass:: staticMethodName	String::valueOf	s -> String.valueOf(s)
2. Reference to an instance method of a particular object	containingObject:: instanceMethodName	s::toString	s -> s.toString()
3. Reference to instance method of an arbitrary object of a given type	ContainingType:: methodName	String::toString	s -> s.toString()
4. Reference to a constructor	ClassName::new	String::new	() -> new String()

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

Kind	Syntax	Method Reference	Lambda Expression
1. Reference to a static method	ContainingClass:: staticMethodName	String::valueOf	s -> String.valueOf(s)
2. Reference to an instance method of a particular object	containingObject:: instanceMethodName	s::toString	s -> s.toString()
3. Reference to instance method of an arbitrary object of a given type	ContainingType:: methodName	String::toString	s -> s.toString()
4. Reference to a constructor	ClassName::new	String::new	() -> new String()

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

Kind	Syntax	Method Reference	Lambda Expression
1. Reference to a static method	ContainingClass:: staticMethodName	String::valueOf	s -> String.valueOf(s)
2. Reference to an instance method of a particular object	containingObject:: instanceMethodName	s::toString	s -> s.toString()
3. Reference to instance method of an arbitrary object of a given type	ContainingType:: methodName	String::toString	s -> s.toString()
4. Reference to a constructor	ClassName::new	String::new	() -> new String()

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

Kind	Syntax	Method Reference	Lambda Expression
1. Reference to a static method	ContainingClass:: staticMethodName	String::valueOf	s -> String.valueOf(s)
2. Reference to an instance method of a particular object	containingObject:: instanceMethodName	s::toString	s -> s.toString()
3. Reference to instance method of an arbitrary object of a given type	ContainingType:: methodName	String::toString	s -> s.toString()
4. Reference to a constructor	ClassName::new	String::new	() -> new String()

Overview of Method References

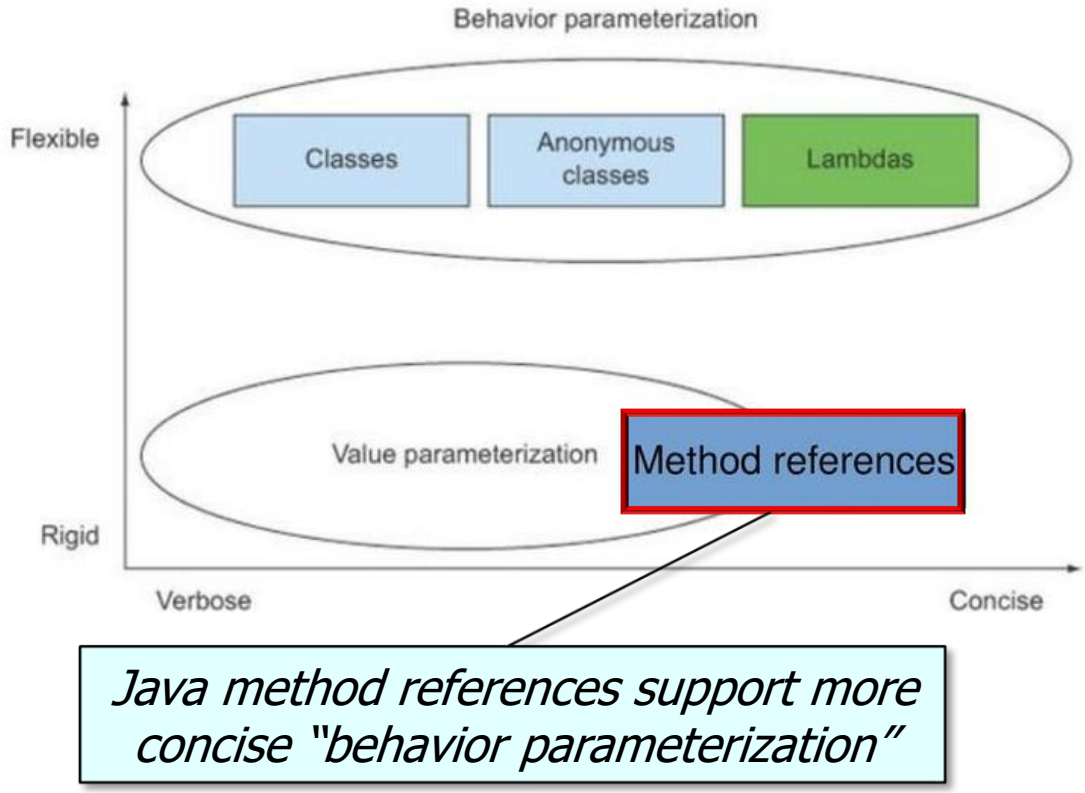
- A compact, easy-to-read handle for a method that already has a name
  - It's shorthand syntax for a lambda expression that executes one method

Kind	Syntax	Method Reference	Lambda Expression
1. Reference to a static method	ContainingClass:: staticMethodName	String::valueOf	s -> String.valueOf(s)
2. Reference to an instance method of a particular object	containingObject:: instanceMethodName	s::toString	s -> s.toString()
3. Reference to instance method of an arbitrary object of a given type	ContainingType:: methodName	String::toString	s -> s.toString()
4. Reference to a constructor	ClassName::new	String::new	() -> new String()

Benefits of Method References

Benefits of Method References

- Method references are more compact than alternative mechanisms



See blog.indrek.io/articles/java-8-behavior-parameterization

Benefits of Method References

- Method references are more compact than alternative mechanisms, e.g.,

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

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

Benefits of Method References

- Method references are more compact than alternative mechanisms, e.g.,

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

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 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.,

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

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

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

See en.wikipedia.org/wiki/Strategy_pattern

Benefits of Method References

- Method references are more compact than alternative mechanisms, e.g.,

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

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::compareTo);
```

It's therefore good practice to use method references whenever you can!

Applying Method References in Practice

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"};
```

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

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

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

```
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

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

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

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

Factory method returns a fixed-size list backed by the array.

prints

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

See www.baeldung.com/java-9-collections-factory-methods

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

See www.javaworld.com/article/2461744/java-language/java-language-iterating-over-collections-in-java-8.html

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

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

prints

Factory method that creates a stream from an array

```
BarbaraJamesMaryJohnLindaMichaelLindajamesmary
```

See 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

```
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

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

prints

```
BarbaraJamesMaryJohnLindaMichaelLindajamesmary
```

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

```
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)

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

Factory method converts an array into a list.

prints

```
BarbaraJamesMaryJohnLindaMichaelLindajamesmary
```

See 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

```
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)

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

prints

```
BarbaraJamesMaryJohnLindaMichaelLindajamesmary
```

Performs method reference action on each list element

See 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

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
      - 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