JAVA 101: LEARN JAVA
By Jeff Friesen, JavaWorld
NOV 12, 2019 11:27 AM PST

About
A beginner's library for learning about essential Java programming concepts, syntax, APIs, and packages.

ADVANCED JAVA LANGUAGE FEATURES

Get started with method references in Java

Use method references to simplify functional programming in Java

`Function<String, MRDemo> function = MRDemo::new;` causes the compiler to look for a constructor that takes a `String` argument, because `Function`'s `apply()` method requires a single (in this context) `String` argument. Executing `function.apply("some name")` results in `"some name"` being passed to `MRDemo(String name)`.

Compile Listing 5 and run the application. You'll observe output that's similar to the following:

```
MRDemo(String name) called with some name
MRDemo@2f92e0f4
```

References to instance methods in superclass and current class types

Java's `super` keyword, which can be used only in instance contexts, references an overridden method in a superclass. You may occasionally need to create a method reference that refers to a superclass method. You can do so via the following syntax:

```
className.super::instanceMethod
```

Java's `this` keyword, which can be used only in instance contexts, invokes a constructor or references an instance method in the current class. You may occasionally need to create a method reference that refers to a method in the current class. You can do so via the following syntax:

```
this::instanceMethod
```

Listing 6 presents an application that demonstrates both syntaxes.

Listing 6. MRDemo.java (version 6)

```
import java.util.function.Consumer;
class Superclass
{
   void print(String msg)
   {
      System.out.printf("Superclass print(): %s%n", msg);
   }
}
class Subclass extends Superclass
{
   Subclass()
   {
      Consumer<String> consumer = Subclass.super::print;
      consumer.accept("Subclass.super::print");
      consumer = this::print;
      consumer.accept("this::print");
   }
   @Override
   void print(String msg)
   {
      System.out.printf("Subclass print(): %s%n", msg);
   }
}
public class MRDemo
{
   public static void main(String[] args)
   {
      new Subclass();
   }
}
```

Listing 6 introduces the `Superclass` and `Subclass` classes, where `Subclass` extends `Superclass`. `Subclass` introduces a constructor and overrides `Superclass`'s `void print(String msg)` method. I've also introduced an `MRDemo` class whose `main()` method drives the application by instantiating `Subclass`.

`Subclass`'s constructor assigns a method reference to `Superclass`'s `print()` method to a `Consumer<String>` variable (`void print(String)` matches `void accept(String)`'s return type and parameter list), and invokes the method. Next, the constructor assigns a method reference to `Subclass`'s `print()` method to this variable and invokes the method.

Compile Listing 6 and run the application. You'll observe the following output:

```
Superclass print(): Subclass.super::print
Subclass print(): this::print
```

In conclusion

Lambdas and method references help us express functionality in a compact manner, which makes source code easier to understand. In this tutorial, I've shown you how to use method references to refer to a class's static methods, bound and unbound non-static methods, and constructors, as well as how to use them to refer to instance methods in superclass and current class types. Keep practicing with lambdas and method references, and get in the habit of using these features instead of the more verbose anonymous classes.

Jeff Friesen teaches Java technology (including Android) to everyone.

Follow