Object Oriented Programming in Java

Lambda Expressions, Methods Refrences

Suhel Hammoud

Writing Lambda Expressions as Method References

- A lambda expression implements the only abstract method of a functional interface.
- Often called "anonymous methods".
- Can be assigned to variables, passed around, or returned.
- Sometimes lambdas are just method calls, e.g.:

```java
Consumer<String> printer = s -> System.out.println(s);
```

- This is where method references come in.

Your First Method Reference

- A lambda can be simplified if it just refers to an existing method.
- Example:

```
Consumer<String> printer = System.out::println;
```

- This is an unbound method reference.
- 4 categories of method references:
    - Static
    - Bound
    - Unbound
    - Constructor

Writing Static Method References

- A reference to a static method.
- Example:

```
DoubleUnaryOperator sqrt = a -> Math.sqrt(a);
DoubleUnaryOperator sqrt = Math::sqrt;
```

- General syntax: `RefType::staticMethod`
- Works with multiple arguments:

```
IntBinaryOperator max = (a, b) -> Integer.max(a, b);
IntBinaryOperator max = Integer::max;
```

Writing Unbound Method References

Methods That Do Not Take Any Argument

- Example:

```
Function<String, Integer> toLength = s -> s.length();
Function<String, Integer> toLength = String::length;
```

- Looks like a static call, but isn't.
- Useful for calling getters:

```
Function<User, String> getName = user -> user.getName();
Function<User, String> getName = User::getName;
```

Unbound Method References with Parameters

- Example:

```
BiFunction<String, String, Integer> indexOf = (sentence, word) -> sentence.indexOf(word);
BiFunction<String, String, Integer> indexOf = String::indexOf;
```

- Syntax: `RefType::instanceMethod`
- Type signature of the method reference helps determine the arguments.

Writing Bound Method References

- The object is fixed in the reference.
- Example:

```
Consumer<String> printer = System.out::println;
```

- Bound to System.out.
- Compare with unbound:

```
Function<User, String> getName = User::getName;
User anna = new User("Anna");
String name = getName.apply(anna);
```

- Syntax: expr::instanceMethod (expr is an actual object or expression)

Writing Constructor Method References

- Refers to a class constructor.
- Example:

```
Supplier<List<String>> newListOfStrings = () -> new ArrayList<>();
Supplier<List<String>> newListOfStrings = ArrayList::new;
```

- Diamond operator is optional, but if you use it then specifying the type:

```
Supplier<List<String>> newListOfStrings = ArrayList<String>::new;
```

Constructor References with Parameters

- Can refer to different constructors:

```
Function<Integer, List<String>> newListOfNStrings = size -> new ArrayList<>(size);
Function<Integer, List<String>> newListOfNStrings = ArrayList::new;
```

- Same syntax `ArrayList::new`, but refers to different constructors.
- Always infer intent from the functional interface's signature.

Bound vs Unbound Method References in Java

In Java method references, the **key difference** between a **bound** and **unbound** method reference is:

	Bound Method Reference	Unbound Method Reference
Object instance	Already known and fixed inside the reference.	Not known yet; will be passed later as a parameter.
Syntax	`instance::methodName`	`ClassName::methodName`
Example	`System.out::println`	`String::toLowerCase`
Lambda Equivalent	`x -> instance.methodName(x)`	`(obj, args...) -> obj.methodName(args...)`
When used	When you already have the specific object.	When the object to call the method on will come later.

Wrapping Up Method References

Name	Syntax	Lambda Equivalent
Static	`RefType::staticMethod`	`(args) -> RefType.staticMethod(args)`
Bound	`expr::instanceMethod`	`(args) -> expr.instanceMethod(args)`
Unbound	`RefType::instanceMethod`	`(obj, rest) -> obj.instanceMethod(rest)`
Constructor	`ClassName::new`	`(args) -> new ClassName(args)`

- Method references simplify lambda expressions.
- IDEs often suggest method reference replacements.