From Collections to Streams in Java 8 Using Lambda Expressions

Lambda Expressions and Functional Interfaces

José Paumard

@JosePaumard | blog.paumard.org

What Is This Course About?

- Lambda Expressions
- Building new functions and API
- The new Collection Framework
- Map / filter / reduce & collections
- The new Stream API

What You Will Learn

- To write lambda expressions in Java 8
- To leverage them to create new API
- To efficiently use the new collections
- To avoid useless computations in implementing map / filter / reduce
- To build and use simple streams

Agenda

- Lambda expressions and functional interfaces
- Writing functions with lambdas
- The new collection framework
- Implementing map / filter / reduce
- The Stream API

Targeted Audience

This is a Java course

Good knowledge of the language

Basic knowledge of the main APIs

Generics

Collection API

Agenda

- Lambda expressions in Java 8
- Method references
- How to create new API
- Use case: the `Comparator` interface

Lambda Expressions in Java 8

How to write, build and use lambdas

What Is Lambda Expression?

- Let us introduce lambda expressions on examples

First Example

- A basic Comparator

```
Comparator<String> comparator = new Comparator<String>() {

   public int compare(String s1, String s2) {
      return Integer.compare(s1.length(), s2.length());
   }
};
```

- We can use it to sort an array of Strings

```
Arrays.sort(tabStrings, comparator);
```

Second Example

- A Runnable

```
Runnable r = new Runnable() {

   public void run() {
      int i = 0;
      while (i++ < 10) {
         Sytem.out.println("It works!");
      }
   }
};
```

- We can execute this in another thread

```
Executors.newSingleThreadExecutor().execute(r);
```

What Did We Do?

- We wrote some code in an anonymous class
- And we passed it to another piece of code
- That executed it “later”
- And in another context (thread)

- We passed code as a parameter
- And we used anonymous class, because it is the only way to do it in Java

Let Us Rewrite Our Code

- The Comparator we wrote

```
Comparator<String> comparator = new Comparator<String>() {

    public int compare(String s1, String s2) {
        return Integer.compare(s1.length(), s2.length());
    }
};
```

- Becomes:

```
Comparator<String> comparator =
    (String s1, String s2) ->
        Integer.compare(s1.length(), s2.length());
```

Let Us Rewrite Our Code

- The Runnable we wrote is a bit more tricky

```
Runnable r = new Runnable() {

    public void run() {
        int i = 0;
        while (i++ < 10) {
            Sytem.out.println("It works!");
        }
    }
}
```

Let Us Rewrite Our Code

- The Runnable we wrote is a bit more tricky

```
Runnable r = () -> {
        int i = 0;
        while (i++ < 10) {
            Sytem.out.println("It works!");
        }
    };
```

Let Us Rewrite Our Code

- In the case there is a returned value:

```
(String s1, String s2) -> {
    System.out.println("I am comparing strings");
    return Integer.compare(s1.length(), s2.length());
}
```

Some Remarks

- One can put modifiers on the parameters of a lambda expression
  - The final keyword
  - Annotations
- It is not possible to specify the returned type of a lambda expression

Some Remarks

- We can also omit the types of the parameters, this code:

```
(String s1, String s2) -> {
    System.out.println("I am comparing strings");
    return Integer.compare(s1.length(), s2.length());
}
```

- Becomes:

```
(s1, s2) -> {
    System.out.println("I am comparing strings");
    return Integer.compare(s1.length(), s2.length());
}
```

Method References

An alternative syntax for lambda expressions

Method References

- There is another way of writing a lambda expression

Method References

- A first example

```
Function<Person, Integer> f = person -> person.getAge() ;
```

```
Function<Person, Integer> f = Person::getAge ;
```

Method References

- A second example

```
BinaryOperator<Integer> sum = (i1, i2) -> i1 + i2 ;
                            = (i1, i2) -> Integer.sum(i1, i2) ;
```

```
BinaryOperator<Integer> sum = Integer::sum ;
```

- And we also have:

```
BinaryOperator<Integer> max = Integer::max ;
```

Method References

- A third example:

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

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

Where Are We?

- So far a lambda expression is a new syntax
- And a new way of writing instances of anonymous classes
- An alternative syntax: method references

- Let us talk about the *type* of a lambda expression

How to Create New API

Lambdas + default methods + static methods

Interfaces Have Been Modified in Java 8

- Default methods:

```
public interface Iterable<T> {

    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
}
```

Interfaces Have Been Modified in Java 8

- Static methods:

```
@FunctionalInterface
public interface Function<T, R> {

    R apply(T t) ;

    static <T> Function<T, T> identity() {
        return t -> t;
    }
}
```

Live coding

How to take existing API and add new patterns to them

Using default and static methods

Using lambda expressions

Live Coding Summary

- The Comparator example
  - How to rewrite a legacy interface
  - To build a new API, simple to use
  - With new patterns!

Summary

- Quick lambda introduction
- Method references: an alternative syntax to write lambda expressions
- How to take an old interface and build a new API from it