Implementing Design Patterns Using Java 8 Lambda Expressions

INTRODUCING DEFAULT METHODS TO CHAIN AND COMPOSE FUNCTIONS

José Paumard
PHD, JAVA CHAMPION, JAVA ROCK STAR

@JosePaumard https://github.com/JosePaumard

How to design API with Java 8

- using lambda expressions

- based on the use of functional interfaces

- leveraging default and static methods

- following the classical design patterns

Writing better code for better applications!

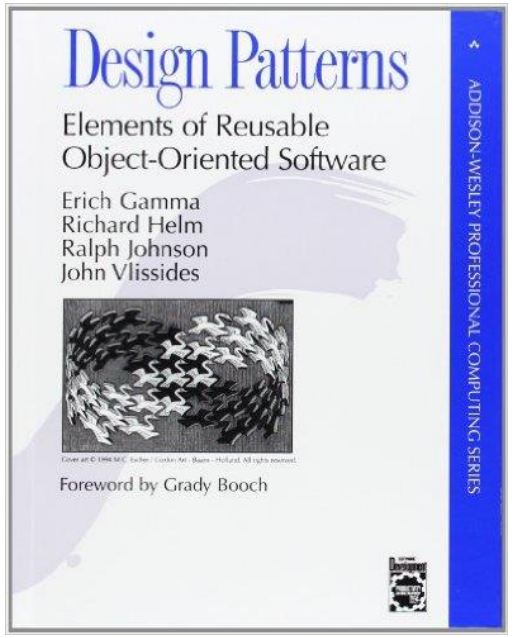
This is a Java 8 course

- basic knowledge of Java 8
- how to write lambda expressions
- what is a functional interface
- some knowledge of design patterns

From Collections to Streams in Java 8 Using Lambda Expressions

Design Patterns in Java:

- The big picture
- Creational
- Structural
- Behavioral



Agenda of the Course

- How to chain and compose Consumer, Predicate and Function
- How to compose Comparator and Function
- Implementation of the Factory, Registry and Builder patterns
- Implementation of the Visitor pattern
- How to use partial application of functions for a Validator pattern

Agenda

First, let us chain and compose simple lambda expressions

Consumers

Predicates

Functions

Chaining Consumers

```java
@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);
}

Consumer<String> c1 = s -> System.out.println("c1 = " + s);
Consumer<String> c2 = s -> System.out.println("c2 = " + s);
```

How to create a third consumer

That would pass the object it consumes

First to the consumer c1, and then to the consumer c2

Demo

Let us do that in a live demo!

Adding a default method

On a functional interface

Provides a way to chain lambdas

Do not forget to fail fast!

Combining and Negating Predicates

```java
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);
}

Predicate<String> p1 = s -> s != null;
Predicate<String> p2 = s -> !s.isEmpty();
```

How to create a third predicate

That would be true if s is both non-null and non-empty

This predicate would be the logical AND of p1 and p2

Demo

Let us do that live!

Default methods can combine lambdas

They can modify the behavior of lambdas

Chaining and Composing Functions

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

    R apply(T t);
}

Function<Meteo, Integer> f1 = meteo -> meteo.getTemperature();
Function<Integer, Integer> f2 = t -> t*9/5 + 32;
```

How to create a third function

That would return the temperature in Fahrenheit directly

It could use chaining or composition

Demo

Let us go back to the IDE!

Difference between chaining

And composition

Composition of only for functions

Factory methods

Module Wrap Up

What did you learn?

Chaining of lambda expressions

Implemented with default methods

Factory methods to create lambdas

Composition: the 1st fundamental operation of functional programming