Java Fundamentals 2 - Day 5

Recap

- Java Design Patterns
- GoF?

Recap Exercise

- Create a small application using one of the 23 patterns.

Functional Programming

- Remember programming paradigms?
    - Imperative Programming
    - Declarative Programming
    - Object Oriented Programming
    - *Functional Programming*

Spoiler

Imperative programming is a paradigm describing **HOW** the program should do something by explicitly specifying each instruction (or statement) step by step, which mutate the program's state.

Declarative programming is a paradigm describing **WHAT** the program does, without explicitly specifying its control flow.

What is Functional Programming?

- Originates from Lambda Calculus
    - Strong roots in mathematics
    - Functions are purely computational (input and output and no side-effects)
    - Data is often immutable

See https://en.wikipedia.org/wiki/Lambda_calculus

Basics

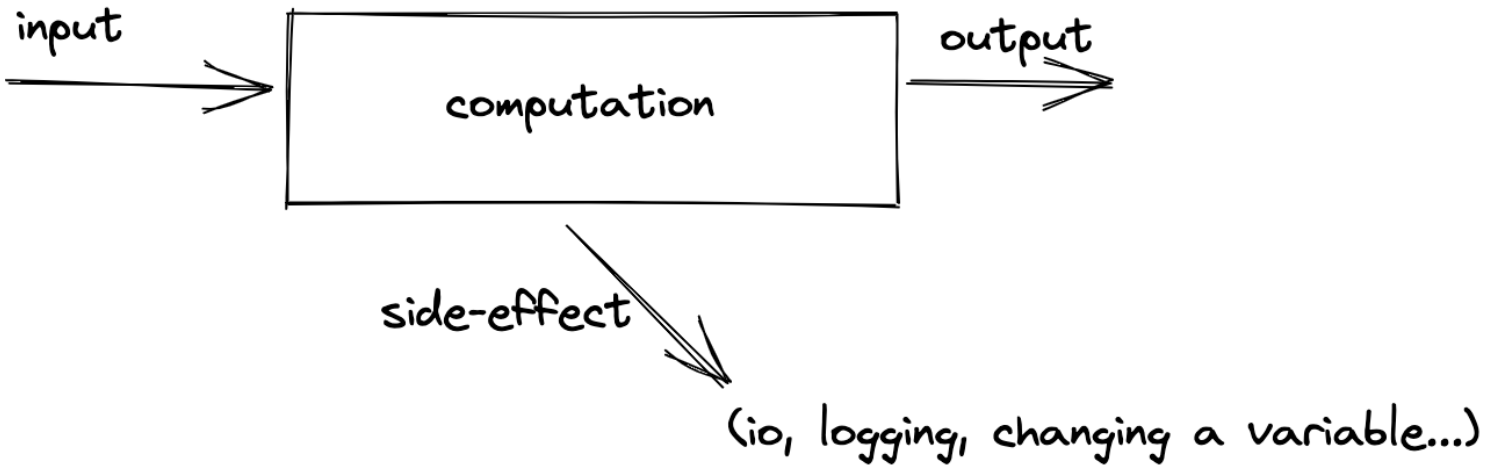
- Functional programming contains the following key concepts:
    - Functions as first class objects
    - Pure functions
    - Higher order functions
- Pure functional programming has a set of rules to follow too:
    - No side effects
    - Calling a function multiple times with same arguments always yields same result
    - No state
    - Immutable variables
    - Favour recursion over looping

Pure functions

- Pure functions only have input and output

Pure Function

Impure Function



Lambda Expression Syntax

```
(argument-list) -> {function-body}
```

Where:

- Argument-list: It can be empty or non-empty as well.
- Arrow notation/lambda notation: It is used to link arguments-list and body of expression.
- Function-body: It contains expressions and statements for lambda expression.

Lets break it down

A simplest lambda expression contains a single parameter and an expression

```
parameter -> expression
```

```
p1 -> p1 * 2 + 1
```

!

To use more than one parameter, wrap them in parentheses

```
(parameter1, parameter2) -> expression
```

!

An expression cannot contain variables, assignments or statements such as if or for. In order to do more complex operations, a code block can be used with curly braces

```
(parameter1, parameter2) -> { code block }
```

```
(p1, p2) -> {
    if (p1 > p2) {
        return p1;
    } else {
        return p2;
    }
```

```
}
```

!

And even without a parameter

```
() -> { code block }
```

```
() -> {
    return 42;
}
```

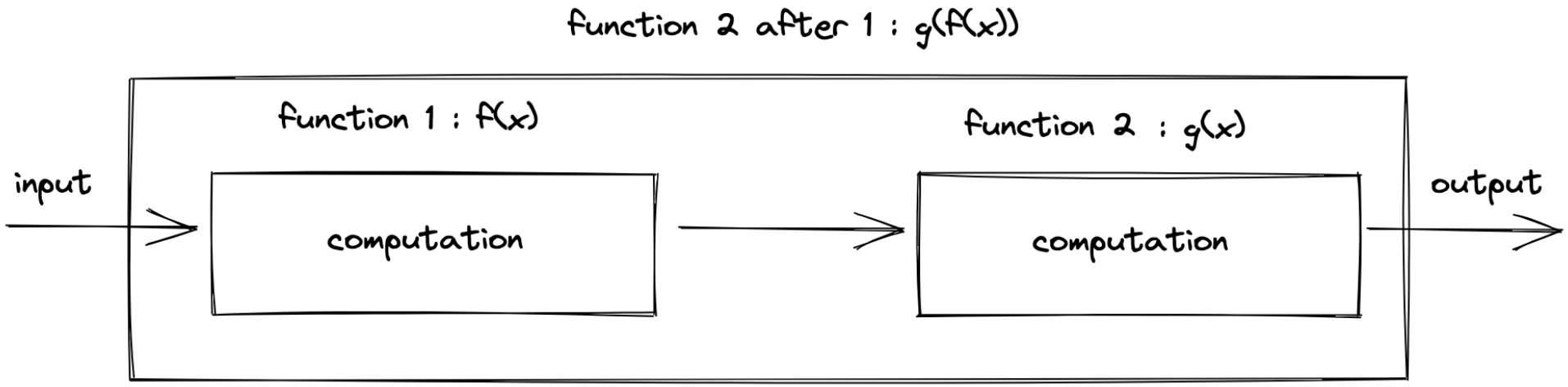
Note that when a function has no arguments and it follows the rules for pure functional programming the result must be a constant expression (e.g. 42 or 37 * 59 or …)

Lambda expressions advantages:

- Code readability / maintainability
    - Reasoning about pure functions is easy
- Sequential and parallel execution
- Pass behavior into functions
- Reduces lines of code (loc)

Why are functions important?

- Functions can be composed to larger functions



Functional Programming in Java

- In Java, FP also helps reduce boilerplate
- Let’s sort a list without FP

```
var numbers = Arrays.asList(8, 3, 2, 1, 7, 9, 5, 6);

// We create an anonymous inner class, implementing Comparator
numbers.sort(new Comparator<Integer> () {
  public int compare(Integer a, Integer b) {
    return a - b;
  }
});
```

- Feel free to feel disgusted, or even horrified
- What is the important part of the sorting-code?

Using FP to sort the list

- FP allows the use of Lambda’s to write the same code like this:

```
var numbers = Arrays.asList(8, 3, 2, 1, 7, 9, 5, 6);

// We create a lambda function to do the sorting
numbers.sort( (a, b) -> a - b );
```

- This code functions *exactly* the same way
- The lambda "implements" `Comparator`
- We only write what matters

Lambda’s

- Lambda’s are "anonymous functions"
- They have a parameter list and either 1 expression, or a block

```
// A lambda with no parameters, and a fixed value
var lambda1 = () -> 4;

// A lambda with one parameter
var lambda2 = (a: Integer) -> a + 1;

// A lambda with two parameters
var lambda3 = (a, b) -> "Test: " + a + " and " + b;

// A lambda with a block
var lambda4 = (a: boolean, b: String) -> {
  // Here we can do all sorts of things here, like in a normal block

  // ...
  // A block does end with a return
  return "Test: " + a + " and " + b;
};
```

Where can lambda's be used?

- `Comparator<T>` is not the only place we can use lambda's
    - `Function<T, R>`
    - `IntConsumer`
- Any `interface` with *exactly one* abstract method can be used for lambda's

```
// The FunctionalInterface Comparator has this abstract method signature
public interface Comparator<T> {
  public int compare(T o1, T o2);
}

// Which allows you to write a lambda like this:
Comparator<Integer> lambda = (o1, o2) -> o1 - o2;

// Or, with full typing:
Comparator<Integer> lambda2 = (o1: Integer, o2: Integer) -> o1 - o2;
```

Funtional Interfaces

- In Java, any *Functional Interface* can be implemented with a lambda
- The method-signature in the interface dictates the form of the lambda (number and type of parameters)
- When compiling, Java matches the signature to the lambda to check if it fits

Functional Programming in Java

- Lambda’s can be actions that are passed to frameworks
- Labmda’s can be used to add custom functionality to frameworks
- Lambda’s are used to store dynamic behaviour
    - Lambda’s can be stored in collections as well

Expressions

Example without lambda expression

```
@FunctionalInterface
interface AddInterface {
    void add(int a, int b);
}

public class FunctionalInterfaceExample {
    public static void main(String args[]) {
        //without lambda, AddInterface implementation using anonymous class
        AddInterface addInterface = new AddInterface() {
            public void add(int a, int b) {
                System.out.println(a + b);
            }
        };
        addInterface.add(10, 20);
    }
}
```

Example without lambda expression 2

```
@FunctionalInterface
interface Drawable {
    public void draw();
}

public class LambdaExpressionExample {
    public static void main(String[] args) {
        int width = 10;

        //without lambda, Drawable implementation using anonymous class
        Drawable d = new Drawable() {
            public void draw() {
                System.out.println("Drawing " + width);
            }
        };
        d.draw();
    }
}
```

Exercise without lambda expression

Create an app using a @FunctionalInterface but without a lambda expression

To roll a dice:)

Example with lambda expression

```
@FunctionalInterface
interface AddInterface2 {
    void add(int a, int b);
}

public class LambdaExpressionExample {
    public static void main(String args[]) {
        //Using lambda expressions
        AddInterface2 addInterface = (a, b) -> {
            System.out.println(a + b);
        };
        addInterface.add(10, 20);
    }
}
```

Example with lambda expression 2

```
@FunctionalInterface  //It is optional
interface Drawable2 {
    public void draw();
}

public class LambdaExpressionExample2 {
    public static void main(String[] args) {
        int width = 10;

        //with lambda
        Drawable2 d2 = () -> {
            System.out.println("Drawing " + width);
        };
        d2.draw();
    }
}
```

Exercise with lambda expression

Create an app using a @FunctionalInterface but with a lambda expression

To roll a dice:)

Example with lambda expression and a return

```
interface Sayable {
    public String say();
}

public class LambdaExpressionExample3 {
    public static void main(String[] args) {
        Sayable s = () -> {
            return "I have nothing to say.";
        };
        System.out.println(s.say());
    }
}
```

Example with lambda expression, a single parameter and a return

```
interface Sayable2 {
    public String say(String name);
}

public class LambdaExpressionExample4 {
    public static void main(String[] args) {

        // Lambda expression with single parameter.
        Sayable2 s1 = (name) -> {
            return "Hello, " + name;
        };
        System.out.println(s1.say("Bob"));

        // You can omit function parentheses
        Sayable2 s2 = name -> {
            return "Bye, " + name;
        };
        System.out.println(s2.say("Alice"));
    }
}
```

Example with lambda expression, a single parameter and a return 2

With multiple statements

```
@FunctionalInterface
interface Sayable3 {
    String say(String message);
}

public class LambdaExpressionExample8 {
    public static void main(String[] args) {

        // You can pass multiple statements in lambda expression
        Sayable3 person = (message) -> {
            String str1 = "I would like to say, ";
            String str2 = str1 + message;
            return str2;
        };
        System.out.println(person.say("time is precious."));
    }
}
```

Exercise with lambda expression, a single parameter and a return

To roll a dice with ..5 sides:)

Example with lambda expression, a multiple parameters and a return

```
interface Addable {
    int add(int a, int b);
}

public class LambdaExpressionExample5 {
    public static void main(String[] args) {

        // Multiple parameters in lambda expression
        Addable ad1 = (a, b) -> (a + b);
        System.out.println(ad1.add(10, 20));

        // Multiple parameters with data type in lambda expression
        Addable ad2 = (int a, int b) -> (a + b);
        System.out.println(ad2.add(100, 200));
    }
}
```

Example with lambda expression, a multiple parameters and a return 2

To use a lambda expression in a method, the method should have a parameter with a single-method interface as its type

```
interface StringFunction {
    String run(String str);
}

public class MainForEachWithInterface {
    public static void main(String[] args) {
        StringFunction exclaim = (s) -> s + "!";
        StringFunction ask = (s) -> s + "?";
        printFormatted("Hello", exclaim);
        printFormatted("Hello", ask);
    }
    public static void printFormatted(String str, StringFunction format) {
        String result = format.run(str);
        System.out.println(result);
    }
}
```

Example with lambda expression, a multiple parameters and a return, or (no) return

In Java lambda expression, if there is only one statement, you may or may not use return keyword. You must use return keyword when lambda expression contains multiple statements.

```
interface Addable2 {
    int add(int a, int b);
}

public class LambdaExpressionExample6 {
    public static void main(String[] args) {

        // Lambda expression without return keyword.
        Addable2 ad1 = (a, b) -> (a + b);
        System.out.println(ad1.add(10, 20));

        // Lambda expression with return keyword.
        Addable2 ad2 = (int a, int b) -> {
            return (a + b);
        };
        System.out.println(ad2.add(100, 200));
    }
}
```

ForEach

Example using forEach to print

Lambda expressions are usually passed as parameters to a function.

```
public class MainForEach {
    public static void main(String[] args) {
        List<Integer> numbers = List.of(5, 9, 8, 1);

        numbers.forEach((n) -> {
            System.out.println(n);
        });
    }
}
```

Example using forEach to print 2

```
public class LambdaExpressionExample7 {
    public static void main(String[] args) {

        List<String> patterns = List.of(
            "Factory", "Strategy", "Adapter", "Decorator",
            "Facade", "Builder", "Iterator"
        );

        patterns.forEach(
                (n) -> System.out.println(n)
        );
    }
}
```

Exercise

Create a List with names and

1. Print each name when using a for index loop
2. Print each name when using a for object loop
3. Print each name when using a list’s foreach loop
4. Sort the list by name’s length and print each name

Example using forEach to print with Consumer

Lambda expressions can be stored in variables

```
public class MainForEachWithConsumer {
    public static void main(String[] args) {
        List<Integer> numbers = List.of(5, 9, 8, 1);

        Consumer<Integer> method = (n) -> {
            System.out.println(n);
        };

        numbers.forEach(method);
    }
}
```

Sort

Example use lambda to sort lists

We can use lambda expression to sort list using Comparator.

```
@AllArgsConstructor
@ToString
@Getter
class Product {
    private int id;
    private String name;
    private float price;

}

public class LambdaExpressionExample10 {
    public static void main(String[] args) {
        List<Product> list = new ArrayList<Product>();

        //Adding Products
        list.add(new Product(1, "Asteroids", 25000f));
        list.add(new Product(3, "Pong", 300f));
        list.add(new Product(2, "Tetris", 150f));
        list.add(new Product(4, "Space Invaders", 150f));
        System.out.println("Sorting on the basis of name...");

        // implementing lambda expression
        Collections.sort(list, (p1, p2) -> {
            return p1.getName().compareTo(p2.getName());
        });

        for (Product p : list) {
```

```
            System.out.println(p.getId() + " " + p.getName() + " " + p.getPrice());
        }
    }
}
```

Exercise

Create an application with a list with game scores. Each score has a username, a score and a creationDateTime. Sort and print the list sorted by score. Sort and print the list sorted by creationDateTime.

Hint

- Use LocalDateTime for creationDateTime

Filter

Example use lambda to filter lists

We can use lambda expression to filter list using Stream's filter.

```
@AllArgsConstructor
@Getter
@ToString
class Product2 {
    private int id;
    private String name;
    private float price;

}

public class LambdaExpressionExample11 {
    public static void main(String[] args) {
        List<Product2> list = new ArrayList<>();
        list.add(new Product2(1, "Samsung A5", 17000f));
        list.add(new Product2(3, "Iphone 6S", 65000f));
        list.add(new Product2(2, "Sony Xperia", 25000f));
        list.add(new Product2(4, "Nokia Lumia", 15000f));
        list.add(new Product2(5, "Redmi4 ", 26000f));
        list.add(new Product2(6, "Lenevo Vibe", 19000f));

        // using lambda to filter data
        Stream<Product2> filtered_data = list.stream().filter(p -> p.getPrice() > 20000);

        // using lambda to iterate through collection
        filtered_data.forEach(
                product -> System.out.println(product.getName() + ": " + product.getPrice())
        );
```

```
    }
}
```

Exercise

Create an application with a list with game scores. Each score has a username, a score and a date. Filter and print the list with scores above 100. Filter and print the list with scores after the first of January this year.

Extra

Example use lambda to create threads

We can use lambda expression to run threads using Runnable.

```
public class LambdaExpressionExample9 {
    public static void main(String[] args) {

        //Thread Example without lambda
        Runnable r1 = new Runnable() {
            public void run() {
                System.out.println("Thread1 is running...");
            }
        };
        Thread t1 = new Thread(r1);
        t1.start();
        //Thread Example with lambda
        Runnable r2 = () -> {
            System.out.println("Thread2 is running...");
        };
        Thread t2 = new Thread(r2);
        t2.start();
    }
}
```

Example use lambda handle a click on a swing button:)

```
public class LambdaEventListenerExample12 {
    public static void main(String[] args) {
        JTextField tf = new JTextField();
        tf.setBounds(50, 50, 150, 20);
        JButton b = new JButton("click");
        b.setBounds(80, 100, 70, 30);

        // lambda expression implementing here.
        b.addActionListener(e -> {
            tf.setText("hello swing");
        });

        JFrame f = new JFrame();
        f.add(tf);
        f.add(b);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(null);
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
```

Exercise

Run the code above and see what it does :)

Katas

https://github.com/zupzup/java8-functional-katas