

CONCURENT AND FUNCTIONAL PROGRAMMING

SI4

Pascal URSO

FUNCTIONAL PROGRAMMING INTRODUCTION

History, languages, usage, immutability

HISTORY

- Main models of computation
 - Lambda calculus (A. Church, 1930s)
 - Turing machine (A. Turing, 1936)
 - Petrin nets (1962) / Khan process networks (1974)
- Main programming paradigm families
 - Functional programming (what?)
 - Imperative programming (how?)
 - Concurrent programming (when?)

LAMBDA CALCULUS

- A formal system in mathematical logic
- Syntax (lambda terms)
 - x – variable
 - $(\lambda x. M)$ – function definition (aka abstraction).
 - $(M N)$ – function application, with M and N lambda terms.
- Operations
 - $(\lambda x. M[x]) \rightarrow (\lambda y. M[y])$ – α -conversion (renaming)
 - $((\lambda x. M) E) \rightarrow (M[x := E])$ – β -reduction



FUNCTIONAL PROGRAMING

- Functional programming languages
 - LISP dialects (fully parenthesized prefix notation), J. McCarthy 1958
 - Common Lisp, Scheme, Clojure, Racket
 - Erlang (Prolog-style, distributed), 1988
 - Haskell, OCaml, Scala (strong static typing)
 - Many others...
- Functional programming in imperative languages
 - C++ (since 11), C#, JavaScript, PHP (since 5.3), Python, Go, Rust, **Java (since 8)**, ...

MAIN CONCEPTS OF FUNCTIONAL PROGRAMMING

- Pure functions
 - No side-effect!, idempotence → **Thread-Safe**
- Recursion
 - Loop are recursion, tail recursion
- First-class and higher-order functions
 - Functions as argument or result
- *Optionally:*
 - *Type systems*
 - *Lazy evaluation*

ADVANTAGES OF FUNCTIONAL PROGRAMMING

- For algorithmic
 - Focus on what, not how
 - Higher order programming
- For computer engineering
 - Better tests (pure function)
 - Many design patterns are functional (MVC, factory, decorator, strategy, ...)
- For interactive systems
 - Handlers are functions
- For distributed systems
 - Non-mutable states
 - No race condition
 - Easier to reason about
 - Efficient use of resources
- Micro-services, AWS lambda, are functions

PURE FONCTIONS

- Computer Science 101
- How to order a list ?
 - *Return a new list that contains the same elements ordered*
- **Purely functional data structure**
 - **Strongly immutable**
- In Java :
 - Immutable collections libraries (Guava, Eclipse, ...)
 - Collections.unmodifiableXXX
 - Records (since Java 16)

JAVA RECORDS

- **Immutable data classes**
- Fields declaration (private, final)
- Automatic definition of
 - Constructor
 - Accessors
 - toString
 - equals, hashCode

```

public class Person {

    private final String name;
    private final String address;

    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, address);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        } else if (!(obj instanceof Person)) {
            return false;
        } else {
            Person other = (Person) obj;
            return Objects.equals(name, other.name)
                && Objects.equals(address, other.address);
        }
    }

    @Override
    public String toString() {
        return "Person [name=" + name + ", address=" + address + "]";
    }

    // standard getters
}

```

EXAMPLE

VS

```

public record Person (String name, String address) {}

```

SOME DETAILS

- Record inherit from java.lang.Record (no other)
- Record are final (no inheritance)
- Record can be generic
- Accessor are fieldName() (not getFieldName())
- You can personalise automatic defintions
- You can add (class) methods / constructors / class fields

```
record Rectangle(double length, double width) {  
    public Rectangle(double length, double width) {  
        if (length <= 0 || width <= 0) {  
            throw new java.lang.IllegalArgumentException(  
                String.format("Invalid dimensions: %f, %f", length, width));  
        }  
        this.length = length;  
        this.width = width;  
    }  
}
```

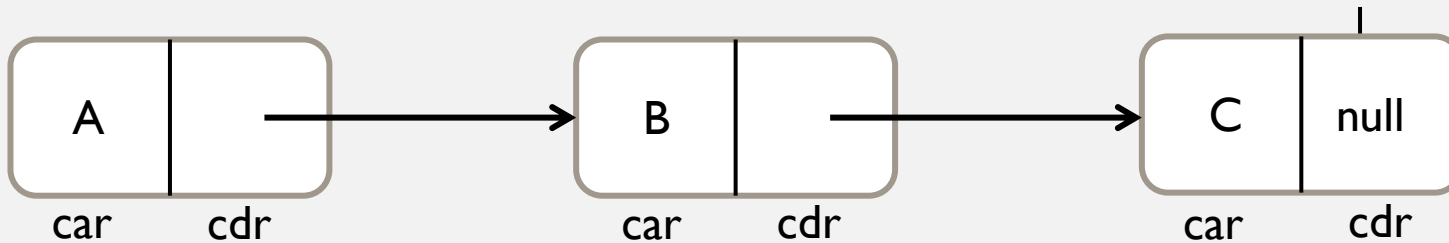
Or better :

```
record Rectangle(@GreaterThanZero double length,  
                @GreaterThanZero double width) { }
```

IMMUTABLE LIST

- Singly linked list

```
public record Lst<T>(T car, Lst<T> cdr) {  
}
```



```
Lst<String> L = new Lst<>( car: "A", new Lst<>( car: "B", new Lst<>( car: "C", cdr: null)));  
System.out.println(L.car());  
System.out.println(L.cdr().car());  
System.out.println(L.cdr().cdr().car());
```

LIST USAGE

- Recursive style
- Immutable data structure

```
public static <T> Lst<T> replaceAll(Lst<T> l, T x, T y) {  
    if (l == null) {  
        return null;  
    } else if (l.car().equals(x)) {  
        return new Lst<>(y, replaceAll(l, x, y));  
    } else {  
        return new Lst<>(l.car(), replaceAll(l, x, y));  
    }  
}
```