

# Data Oriented Programming in Java

<https://github.com/forax/dop-examples>

Rémi Forax  
Université Gustave Eiffel – Sept 2022



CALVIN & HOBBS © BIL WATTERSON

Don't believe what I'm saying !

I'm a huge proponent of designing your code around the data, rather than the other way around [...]

Bad programmers worry about the code. Good programmers worry about data structures and their relationships.

-- *Linus Torvalds*

Did Linus just disrespect OOP ?



class-room-as-picasso

# Structure definitions in Java (let's go back to school)

# Structure definitions (Java 1.0)

A **class** mixed

- Data definition (mutable fields)
- Behaviors (methods)

```
class Drawing {  
    private String title;  
    public Drawing(String title) { this.title = title; }  
    public int price() { return title.length * 100; }  
}
```

# Structure definitions (Java 8)

A **lambda** is an anonymous function

- data ?
- one function

```
interface Article {  
    int price();  
}
```

...

```
public static Article drawing(String title) { return () -> title.length * 100; }
```

# Structure definitions in Java 8+

class (mutable data + methods)

lambda (immutable data + one function)



# Structure definitions (Java 17)

A **record** is pure data ?

- immutable data
- methods ?

```
record Drawing(String title) {}
```

```
...
```

```
public static int price(Drawing drawing) {  
    return drawing.title() * 100;  
}
```

# Structure definitions in Java 17+

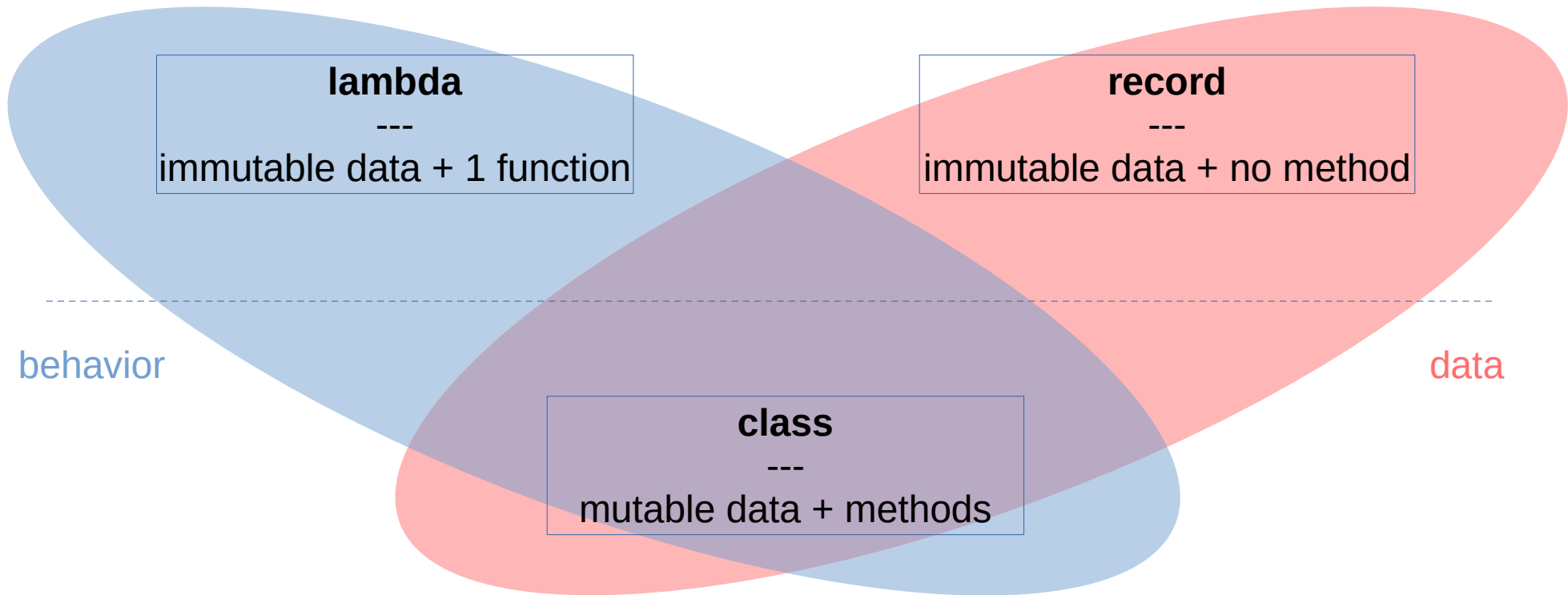
class (mutable data + methods)

lambda (immutable data + one function)

record (immutable data + no\* methods)

\* you can if you want but that not the main style

# Structure definitions in Java 17+





fantastic-unicorn-as-picasso

What is a record and  
where to use them ?

DEMO !

(1 / 3)



matching-as-picasso

# Pattern Matching

# Virtual Polymorphism

```
interface Article {  
    int price();  
}  
  
record Drawing(String title) implements Article {  
    public int price() { return title.length() * 100 }  
}  
  
enum Maker { DISNEY, ACME }  
record Toy(Maker maker) implements Article {  
    public int price() {  
        return switch(maker) {  
            case DISNEY -> 1_000;  
            case ACME -> 130;  
        };  
    }  
}
```

# Pattern Matching

```
interface Article { int price(); }  
record Drawing(String title) implements Article { }  
enum Maker { DISNEY, ACME }  
record Toy(Maker maker) implements Article { }  
  
int price(Article article) {  
    return switch(article) {  
        case Drawing drawing -> drawing.title().length() * 100;  
        case Toy toy ->  
            switch(toy.maker()) {  
                case DISNEY -> 1_000;  
                case ACME -> 13;  
            };  
        default -> ... // FIXME  
    };  
}
```



# Wadler's Expression Problem

## Polymorphism

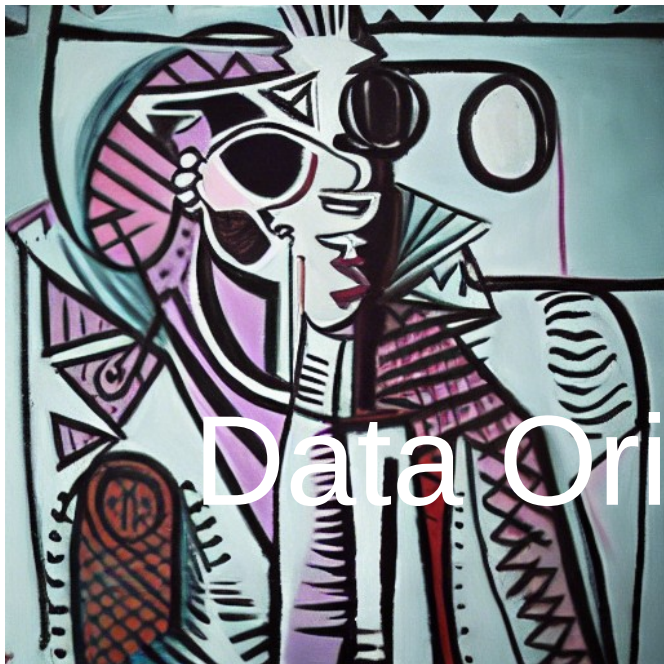
- add new subtypes
- No new operation

## Pattern Matching

- Add new operations
- No new subtypes

We can not get both :(

DEMO !  
(2 / 3)



dope-as-picasso

# Data Oriented Programming\*

\* warning overloaded term !

# DOP Principle

Data are more important than code

When data are updated, code should follow

- Compilers should flag where updates need to be done

# DOP in Java

Switch Expression must be **exhaustive**

- Add sealed types
- Switch statement get a warning

**Record Pattern** test the shape of a record

# DOP in Java

```
sealed interface Article permits Drawing, Toy { int price(); }
```

```
record Drawing(String title) implements Article { }
```

```
enum Maker { DISNEY, ACME }
```

```
record Toy(Maker maker) implements Article { }
```

```
int price(Article article) {  
    return switch(article) {  
        case Drawing(String title) - > title.length() * 100;  
        case Toy(Maker maker) - >  
            switch(toy.maker()) {  
                case DISNEY - > 1_000;  
                case ACME - > 13;  
            };  
        // no default here !  
    };  
}
```

# We steal everything !

From Scala (2004) / Kotlin (2011)

- record come from “case class” / “data class”
- sealed keyword

# We steal everything !

From Scala (2004) / Kotlin (2011)

- record come from “case class” / “data class”
- sealed keyword

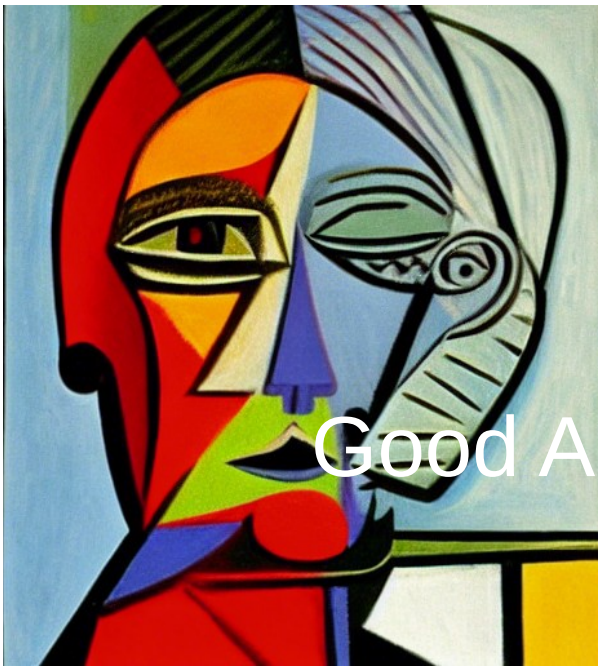
From Standard ML (1983)

**datatype** article

= Drawing **of** String

| Toy **of** Maker





Good Artists Copy, Great Artists Steal  
-- *Pablo Picasso*

picasso-as-picasso

DEMO !  
(3 / 3)



oops-as-picasso

# OOP vs DOP

# OOP vs DOP

Not on the same interfaces

- Not-sealed: users can provide new implementations
- Sealed: all implementations are known

Not at the same level

- OOP works at API level
- DOP works at implementation level

# OOP and DOP

Not on the same interfaces

Not at the same level

=> OOP and DOP should work together



# Patterns in Pattern Matching (everything is in preview)

patterns-as-picasso

# Type Pattern (Java 17)

Equivalent to an instanceof check  
but type safe !

```
Article article = ...  
switch(article) {  
    case Toy toy -> ...  
    case Drawing drawing -> ...  
}
```

# Record Pattern (Java 19)

Check the shape of a record (DOP)

```
Article article = ...  
switch(article) {  
    case Toy(Maker maker) toy -> ...  
    case Drawing(String title) -> ...  
}
```



# Var Pattern (Java 19)

Ask the compiler to infer the type

```
Article article = ...  
switch(article) {  
    case Toy(var maker) -> ...  
    case Drawing(var title) -> ...  
}
```

# *Null Pattern (Java 19)*

Not really a pattern

If there is a case null, switch allows null

```
String s = ...  
switch(s) {  
    case null -> ...  
    case String s -> ...  
}
```

# *Guard Pattern (Java 19)*

No really a pattern, attached to a case

Article article = ...

**switch**(article) {

**case** Toy(var maker) **when** maker == Maker.DISNEY -> ...

**case** Toy toy -> ...   // for the other makers

**case** Drawing d -> ...

}

# Instanceof vs switch

All patterns also works with instanceof

```
record House(String... persons) {  
    public boolean equals(Object o) {  
        return o instanceof House(var persons) &&  
            Arrays.equals(this.persons, persons);  
    }  
    // + hashCode  
}
```

# Any Pattern '\_' (Java 20)

I don't care about that value

```
Point point = ...  
switch(point) {  
    case Point(var x, _) - > ...  
    case Point(_, _) - > ...  
}
```

Also works in lambda, for, catch and try-with-resources !

# Primitive Pattern (Java 20)

Is a primitive cast is safe ?

```
int value = ...  
switch(value) {  
    case short -> ... // values in Short.MIN_VALUE ... MAX_VALUE  
    case int i -> ...  
}
```

I don't like it but I'm not the spec lead !

# Array Pattern (Java 20)

Has several items (exact count)

```
String[] array = ...  
switch(array) {  
    case String[] { var i1, var i2 } -> ... // if two items  
    case String[] array -> ... // other arrays  
}
```

# Constant Pattern (Java 21+)

Use a constant (syntax still in flux)

```
Point p = ...  
switch(point) {  
  case Point(0, 0) -> ...  
  case Point _ -> ...  
}
```



# And more ...

Pattern assignment

```
Point(int x, int y) = point;
```

Deconstruction pattern

Record pattern but on interface class and enum

Named pattern

```
switch(optional) {  
  case Optional.of(var value) -> ...  
  case Optional.empty() -> ...  
}
```



questions-as-picasso

# Questions ?

<https://github.com/forax/dop-examples>

All images have been dreamed by Stable Diffusion