Exploring Other Design Patterns in Java

Esteban Herrera
JAVA ARCHITECT

@eh3rrera www.eherrera.net

Overview

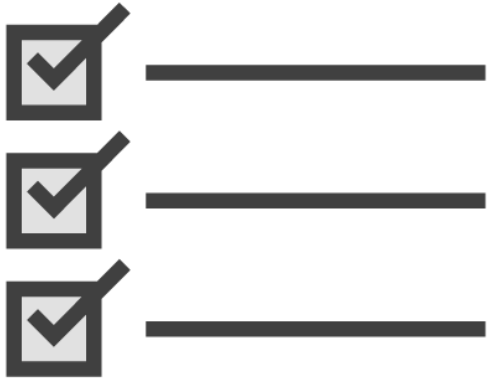
Other patterns

- Enterprise development
- Functional programming
- Reactive programming

Specializations of the Gang of Four design patterns

Enterprise Development Patterns

Enterprise Software Development



- Data persistence
- Concurrency
- Integrate with other applications

Patterns in Enterprise Software

In recent years there's been a small but useful growth in describing patterns for the development of enterprise systems. On this page I keep a list of the most notable catalogs on these patterns and some thoughts on the broad interrelationships between them.

There's no formal organization tying these writers together, but we do have a strong informal connection - frequently reviewing each others' work. We've often wondered if we should set up some more organized group, but haven't really summoned up enough energy around it to actually make anything happen. Just writing our own work is quite hard enough!

Different people have different expectations about what patterns are good for and why they are interesting. I described my view of this in a column for IEEE Software.

I'm listing the catalogs he[illegible] comfortable with. I don't i[illegible]

http://bit.ly/entpatterns

Catalogs

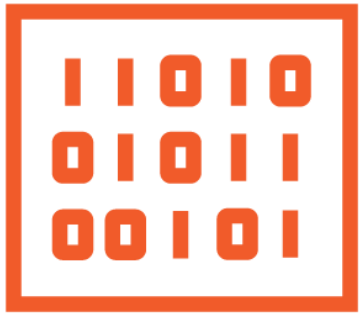
MVC Pattern

It's an architectural pattern

It dates back to the late seventies

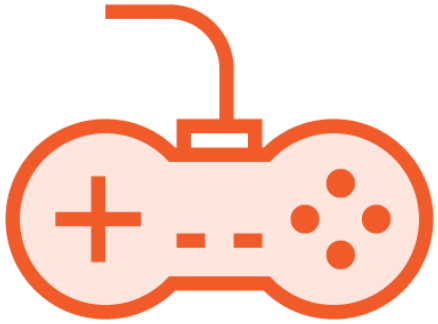
It can be described in terms of the GoF patterns

MVC



Model

View



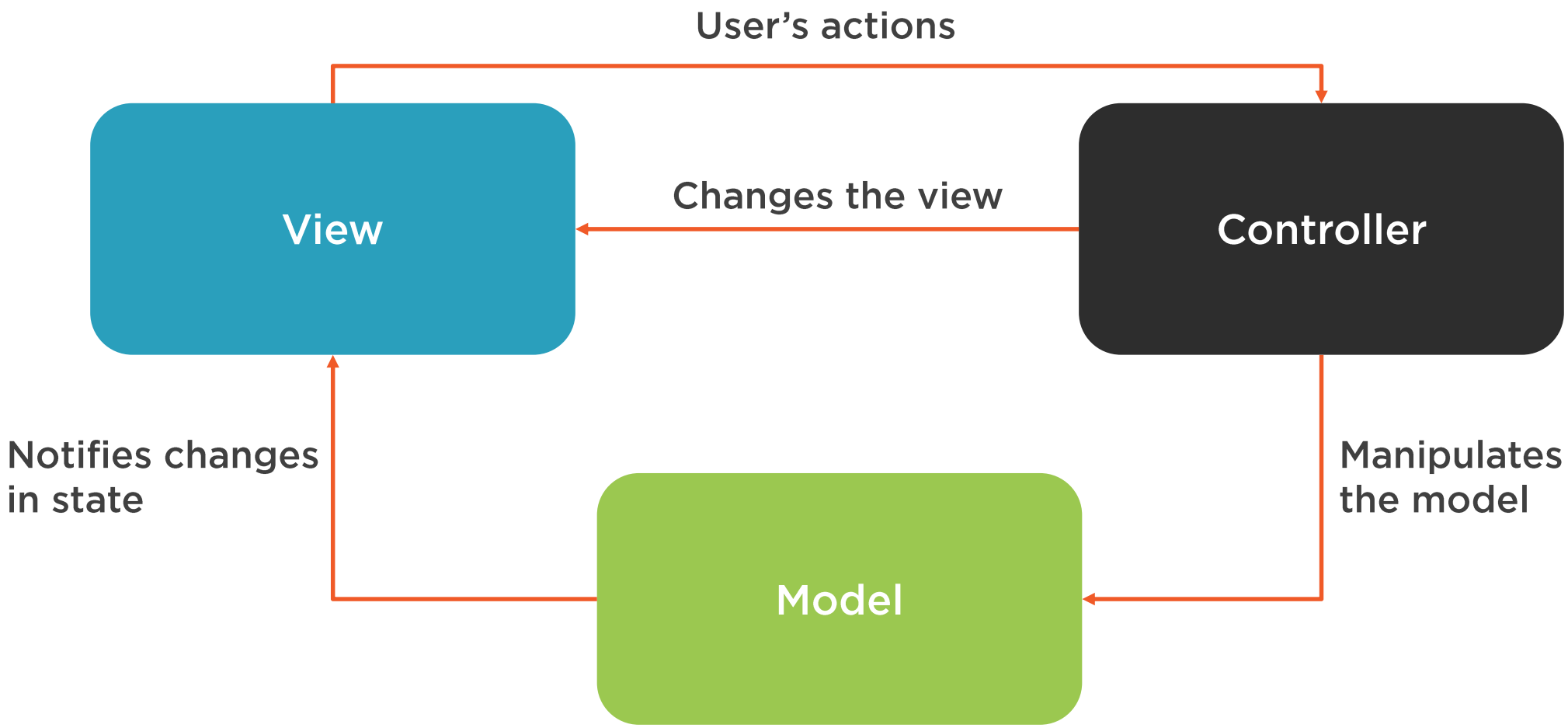
Controller

Separation of Concerns

Business logic

Presentation

Model View Controller



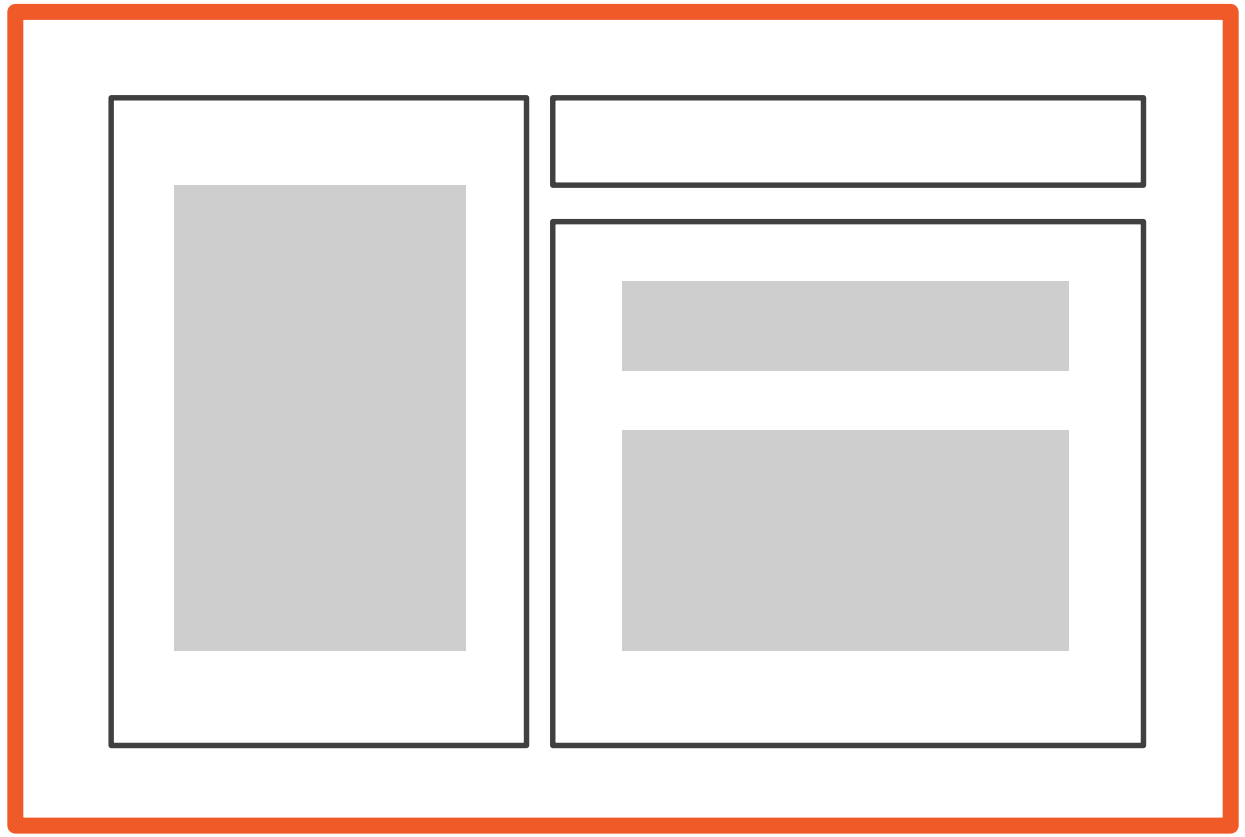
Understanding MVC as Patterns

Composite

Strategy

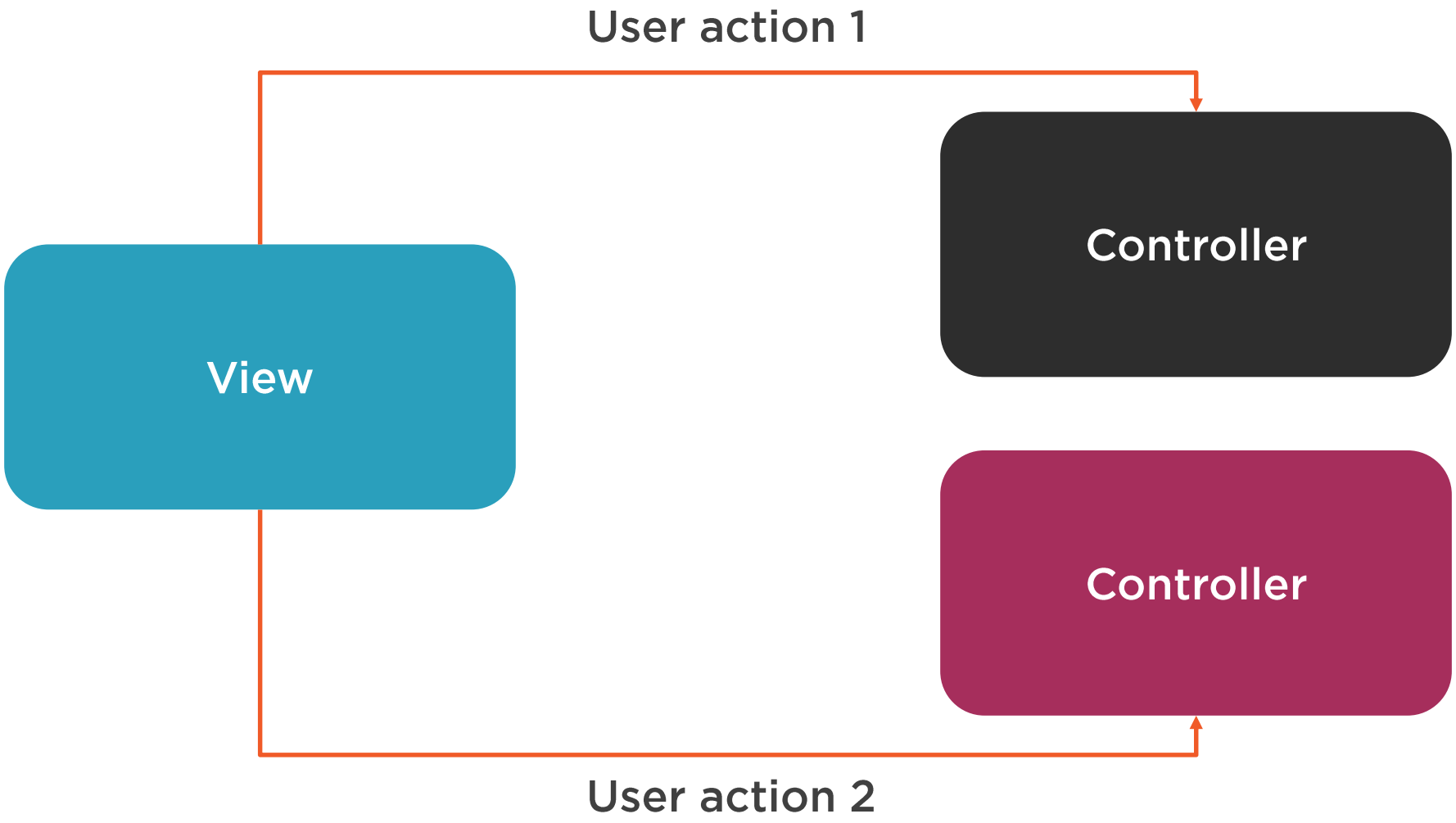
Observer

Composite

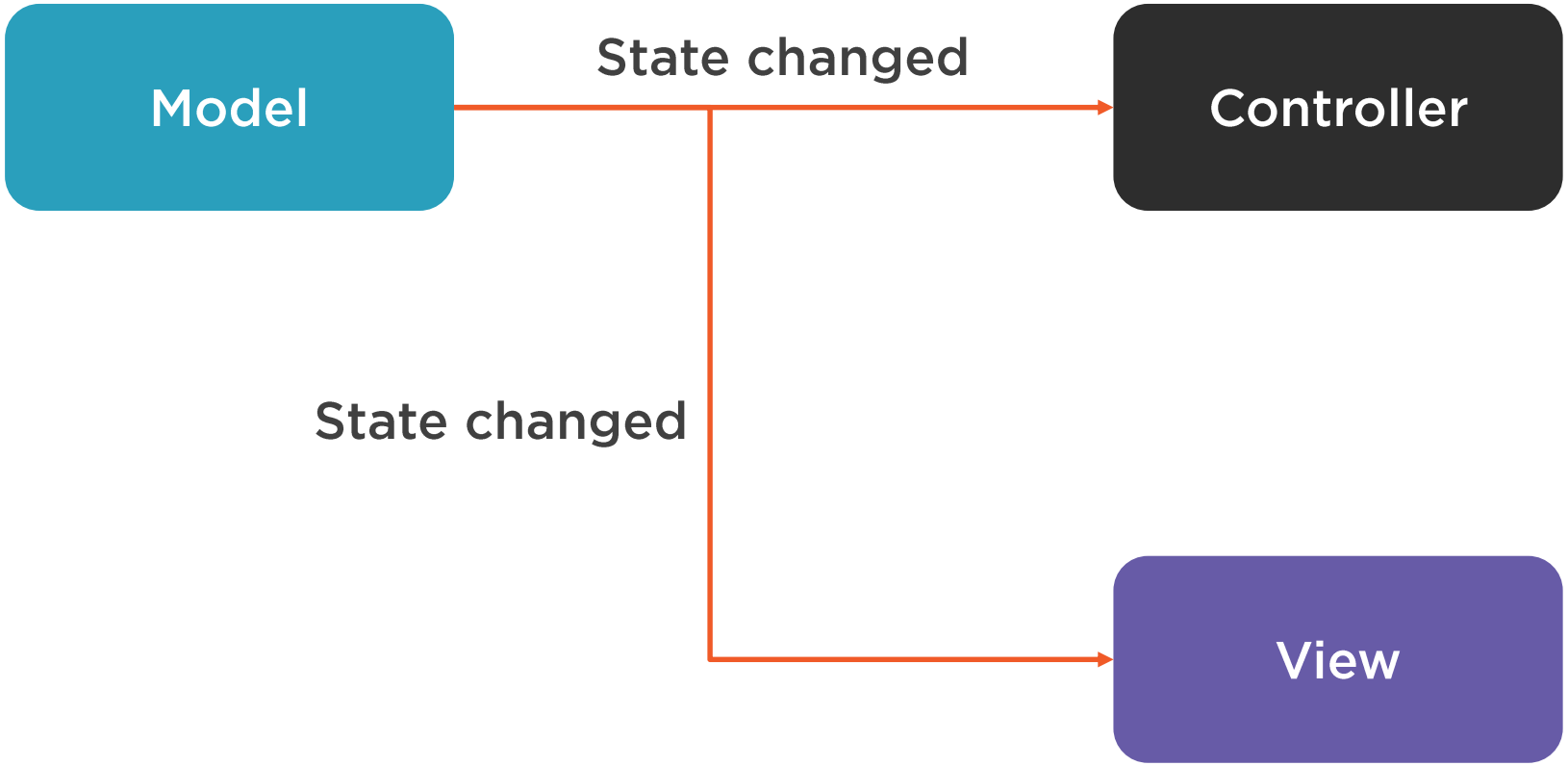


View

Strategy

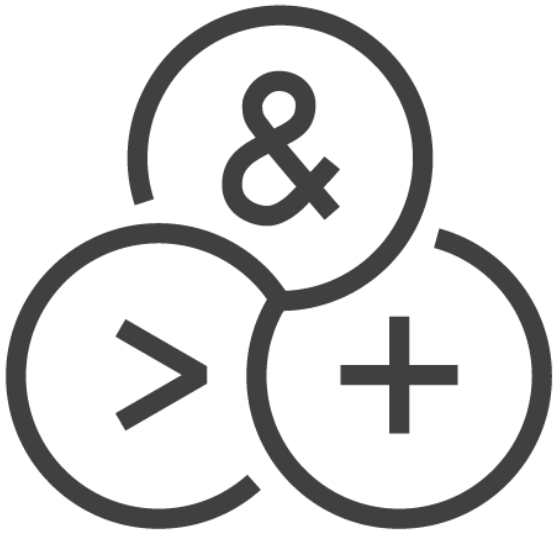


Observer



Functional Programming Patterns

Functional Programming



High order functions

Pure functions

- Have a single responsibility
- Have no side effects
- Are referentially transparent

Design Patterns in Functional Programming?

Functional programming is a different paradigm

- Java is not really a functional language

Most OOP design patterns are irrelevant in functional programming

The choice of programming language is important because it influences one's point of view. Our patterns assume Smalltalk/C++-level language features, and that choice determines what can and cannot be implemented easily.

Design Patterns: Elements of Reusable Object-Oriented Software

```
object MyObject {
    def printMessage(message: String) {
        println("Message: $message");
    }
}
```

Scala Objects

Replace the Singleton pattern

Functional Patterns

MapReduce

- Break tasks into smaller ones (map) and aggregate the result (reduce)

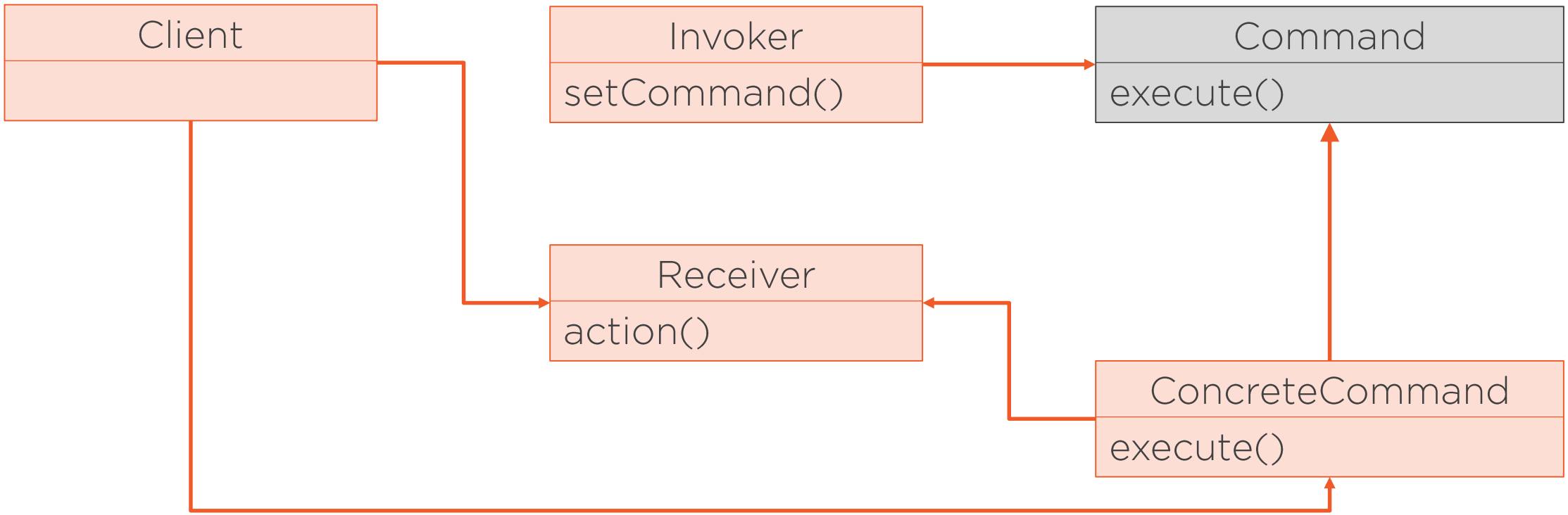
Memoization

- Cache the result of a function

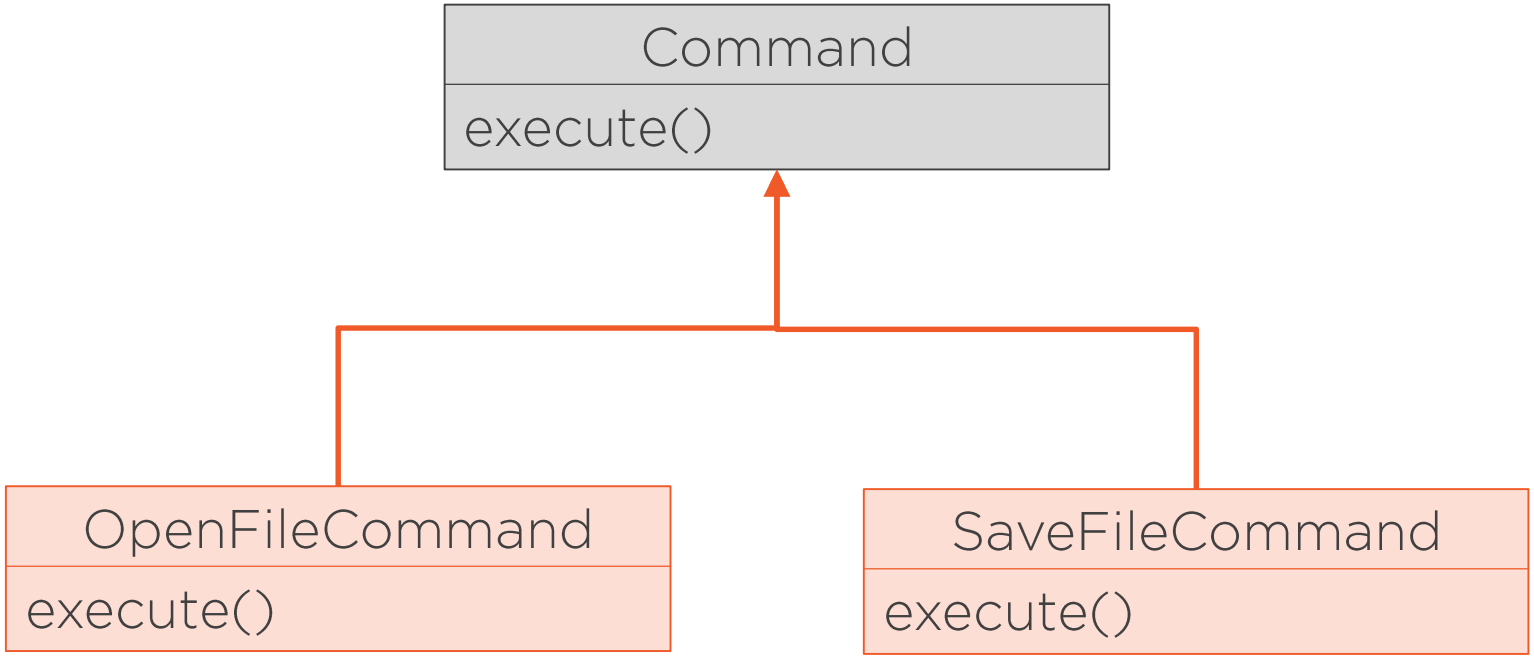
Monad

- Container type that defines rules of interaction and provides composing functions

The Command Pattern



The Command Pattern



Command Pattern with Anonymous Classes

```
menu.executeAction(new Command() {
    void execute() {
      openFile("notes.txt");
    }
});
```

Command Pattern with Lambda Expressions

```
menu.executeAction(() -> openFile("notes.txt"));
```

See GoF patterns as one way to understand functional programming concepts.

Design principles apply regardless of the programming style.

```
void String formatString(String s,
        Function<String, String> func) {
    return func.apply(s);
}
```

Strategy Pattern

Understand high-level functions

```
formatString(s, s -> s.toUpperCase());
formatString(s, s -> s.toLowerCase());
```

Strategy Pattern

Understand high-level functions

```
Pizza p =
    new ExtraCheesePizza(new PepperoniPizza(new BasicPizza()));
```

Decorator Pattern

Understand function composition

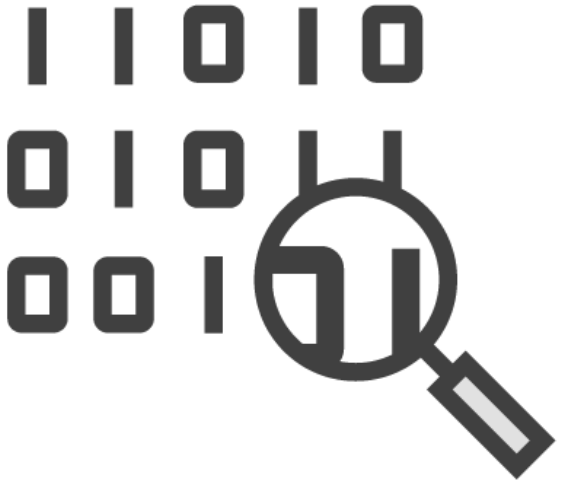
```
Function<Pizza, Pizza> extraCheesePizza = p -> {/* */};
Function<Pizza, Pizza> pepperoniPizza = p -> {/* */};
Pizza p = extraCheesePizza.
            .andThen(pepperoniPizza)
            .apply(new BasicPizza());
```

Decorator Pattern

Understand function composition

Reactive Programming Patterns

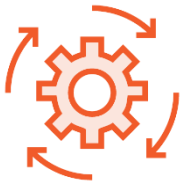
Reactive Programming



Event-driven

Data flow

Reactive Programming Model



Non-blocking

Asynchronous

Declarative

Example

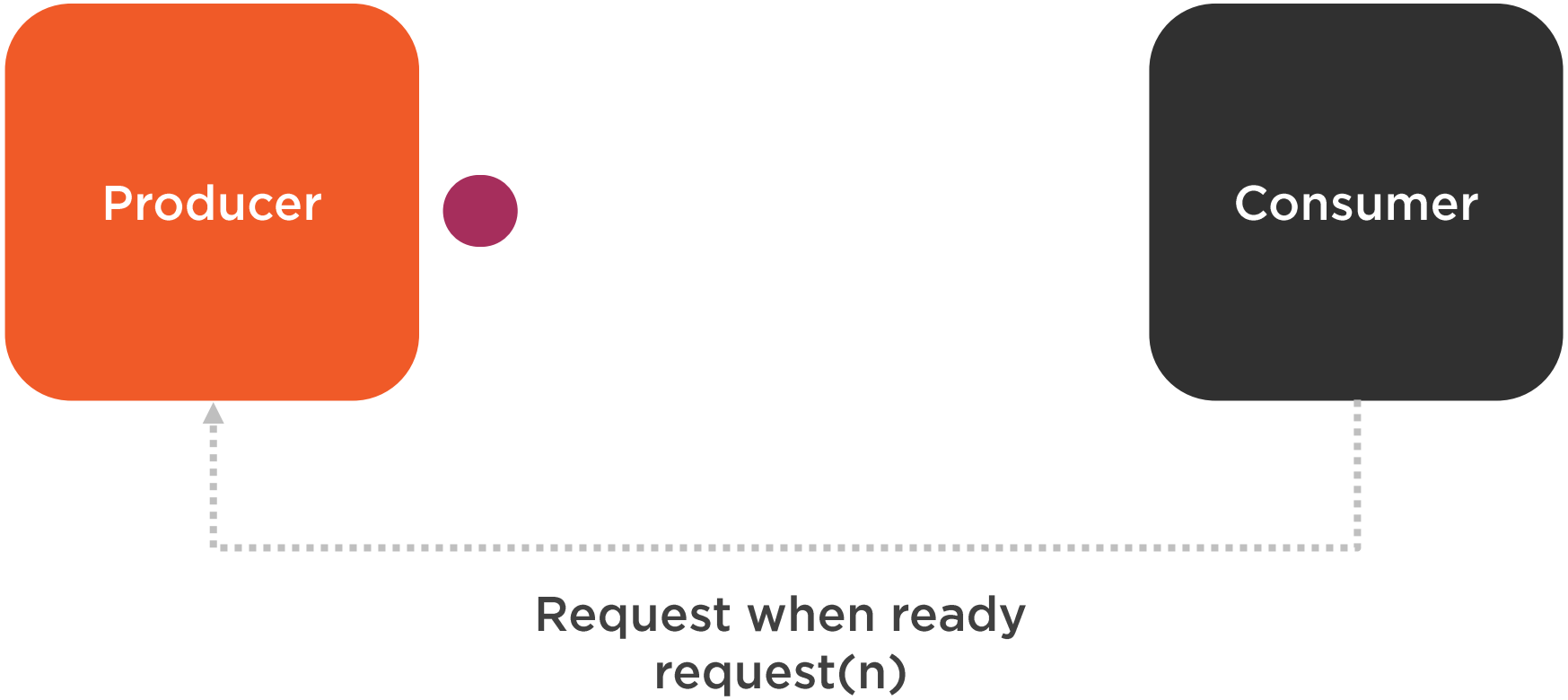
```
String[] letters = {"hello", "world"};

Observable.from(words)
   .map(String::toUpperCase)
   .subscribe(word -> System.out.println(word));
```

The reactive programming model is based on the Observer and Iterator patterns.

Observer Pattern

Observer Pattern

Backpressure



Observer/Iterator

Observer/Iterator

Reactive Programming Patterns?

Asynchronous patterns

- Already implemented in reactive programming libraries

Reactive design patterns

- Architecture of systems according to the Reactive manifesto

Things to Remember

Enterprise development

- Model View Controller
  - Composite
  - Strategy
  - Observer

Functional programming

- Object-oriented patterns are irrelevant in functional programming
- They can help you understand functional programming concepts

Reactive programming

- Based on Observer and Iterator patterns