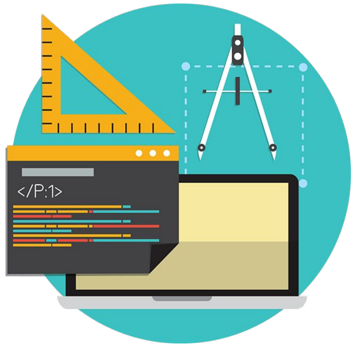


OOP / Design
OOAD / FP Principles

DIEGO PACHECO

About me...

- ❑ Cat's Father
- ❑ Principal Software Architect
- ❑ Agile Coach
- ❑ SOA/Microservices Expert
- ❑ DevOps Practitioner
- ❑ Speaker
- ❑ Author

diegopacheco

@diego_pacheco

http://diego-pacheco.blogspot.com.br/

Diego Pacheco

Building Applications with Scala

Write modern, scalable, and reactive applications with the power of Scala

Packt>

Diego Pacheco

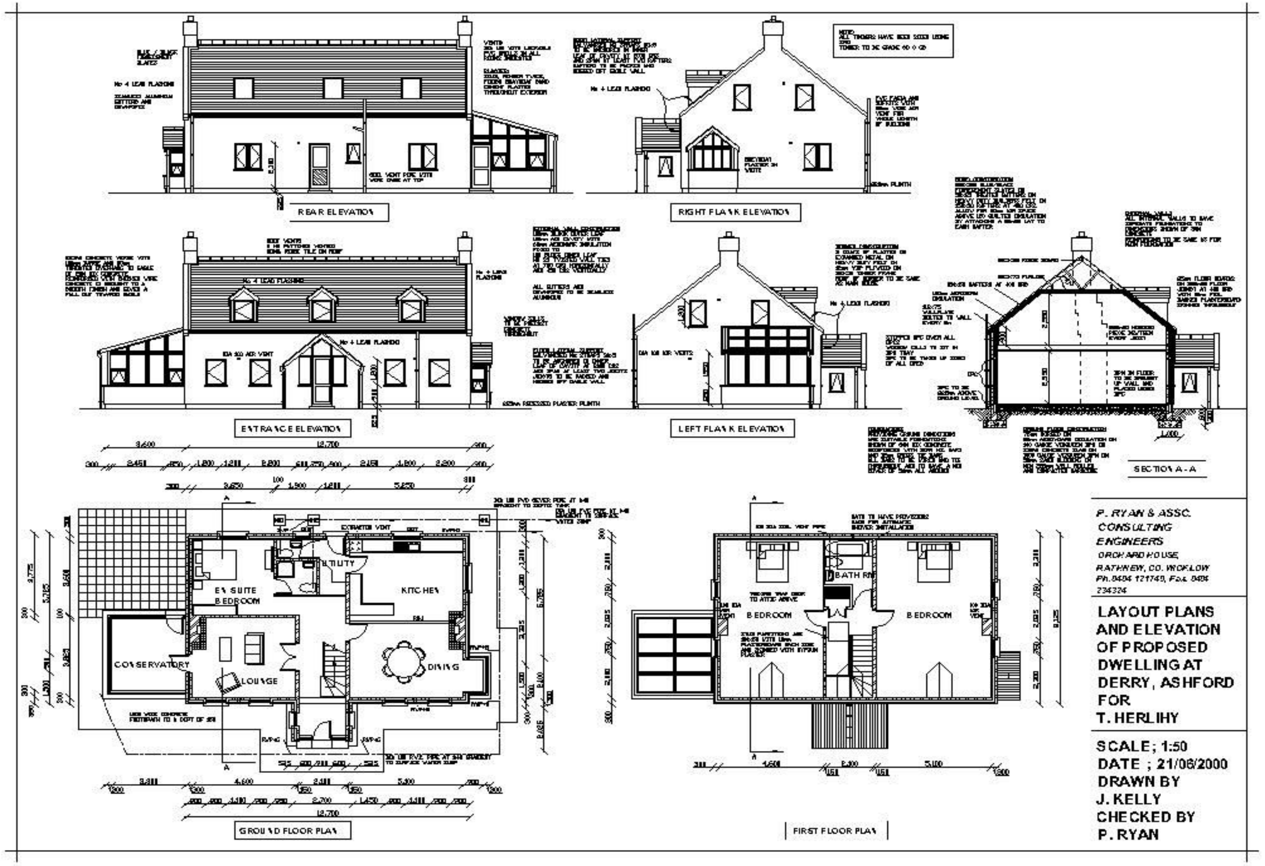
Building Effective Microservices

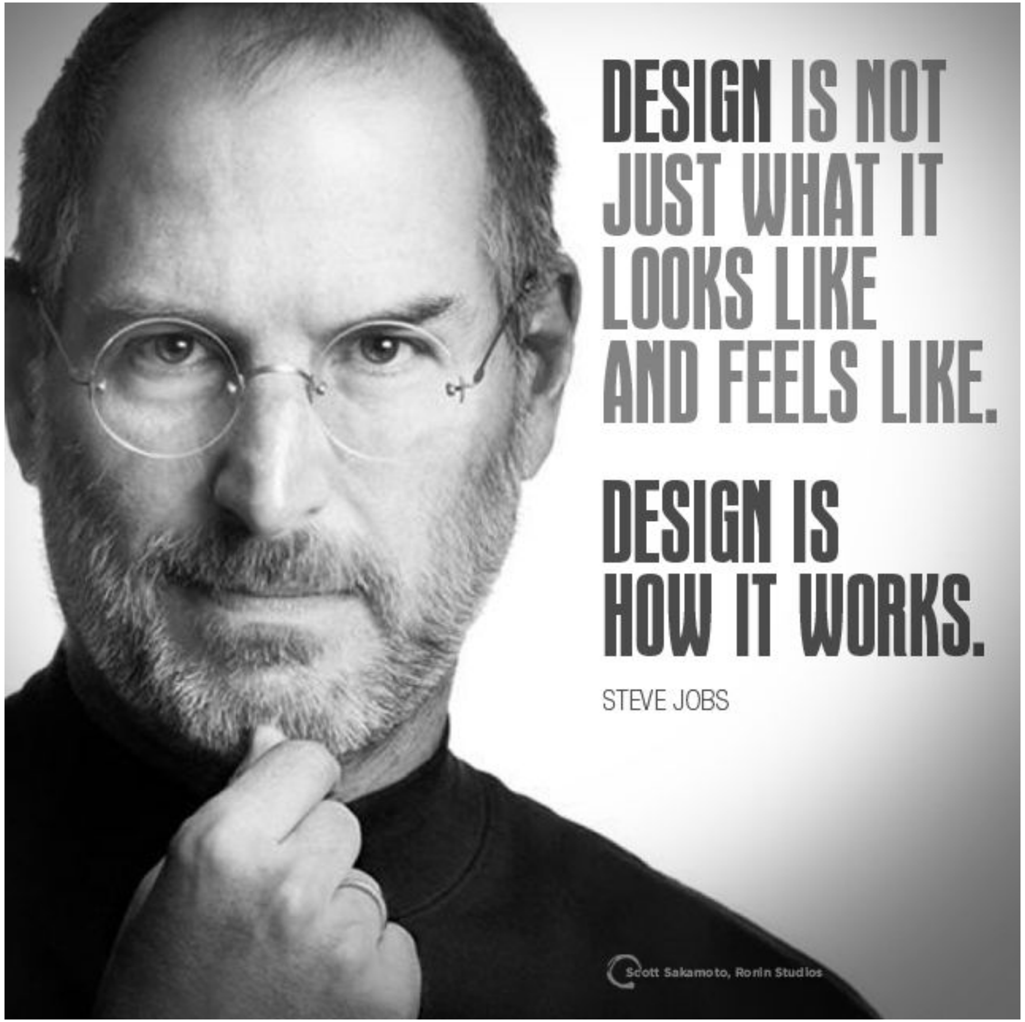
Explore microservices and their implementation hands-on

Packt>

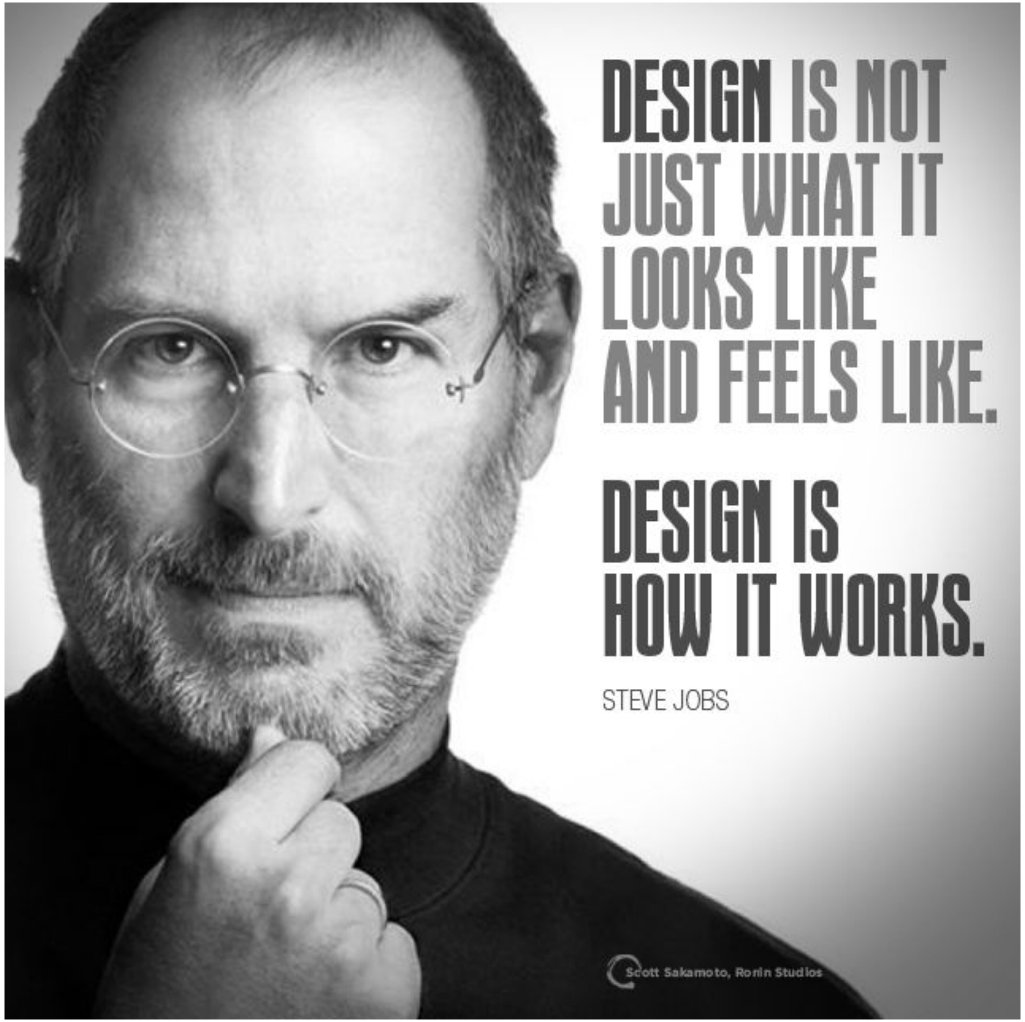
https://diegopacheco.github.io/

Relationship between Software Design & Architecture



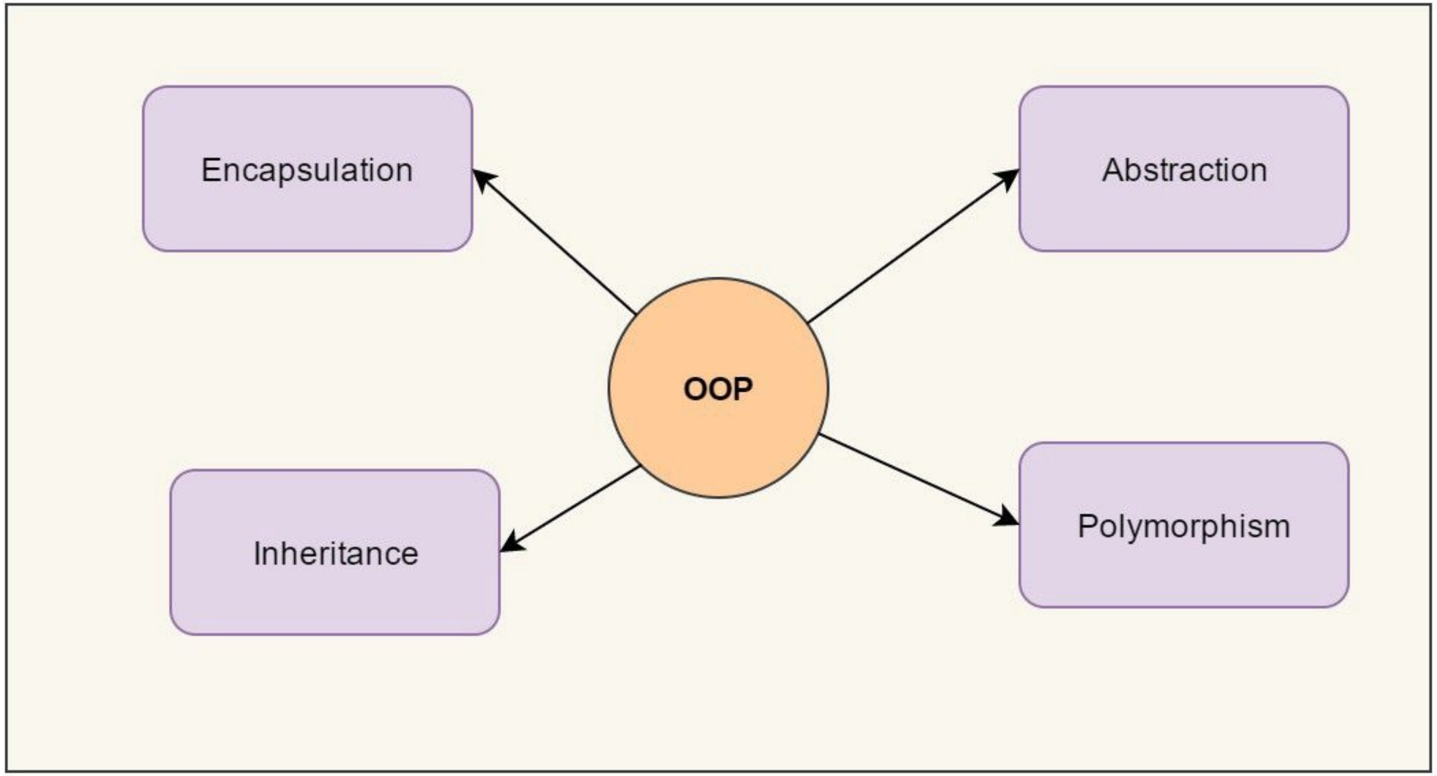

DESIGN IS NOT JUST WHAT IT LOOKS LIKE AND FEELS LIKE.
DESIGN IS HOW IT WORKS.
STEVE JOBS
Scott Sakamoto, Ronin Studios

- ❑ Use Cases
- ❑ Concept
- ❑ Structure & Behavior
- ❑ Rationale
- ❑ Trade offs
- ❑ Think about:
  - ❑ User
  - ❑ Machine
  - ❑ Cloud
  - ❑ Developers
- ❑ Simple
- ❑ Avoid BDUF
- ❑ Refactoring


DESIGN IS NOT JUST WHAT IT LOOKS LIKE AND FEELS LIKE.
DESIGN IS HOW IT WORKS.
STEVE JOBS
Scott Sakamoto, Ronin Studios

- ❑ Properties:
  - ❑ Feasibility
  - ❑ Testability
  - ❑ Maintainability
  - ❑ Debuggability
  - ❑ Observability
  - ❑ Flexibility
  - ❑ Extensibility
- ❑ Principles:
  - ❑ Loose Coupling
  - ❑ High Cohesion
  - ❑ Isolation
  - ❑ SOC

O
O
P
Object
Oriented
Programming

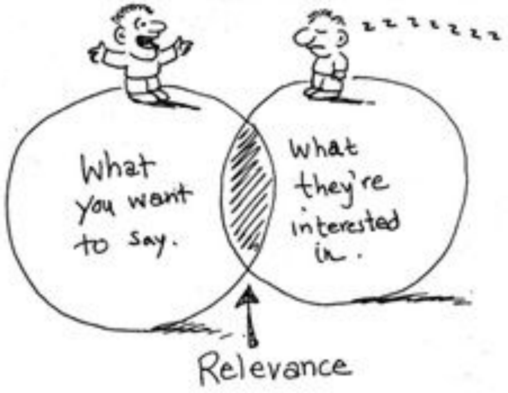


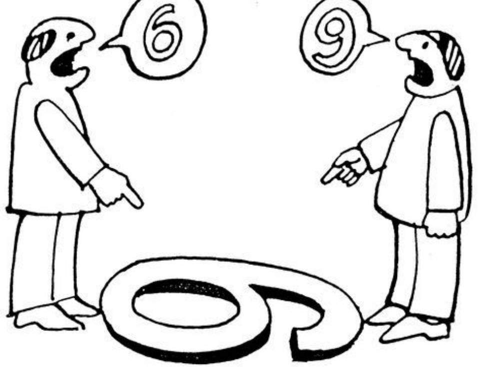
Four Pillars of Object Oriented Programming

What is abstraction?

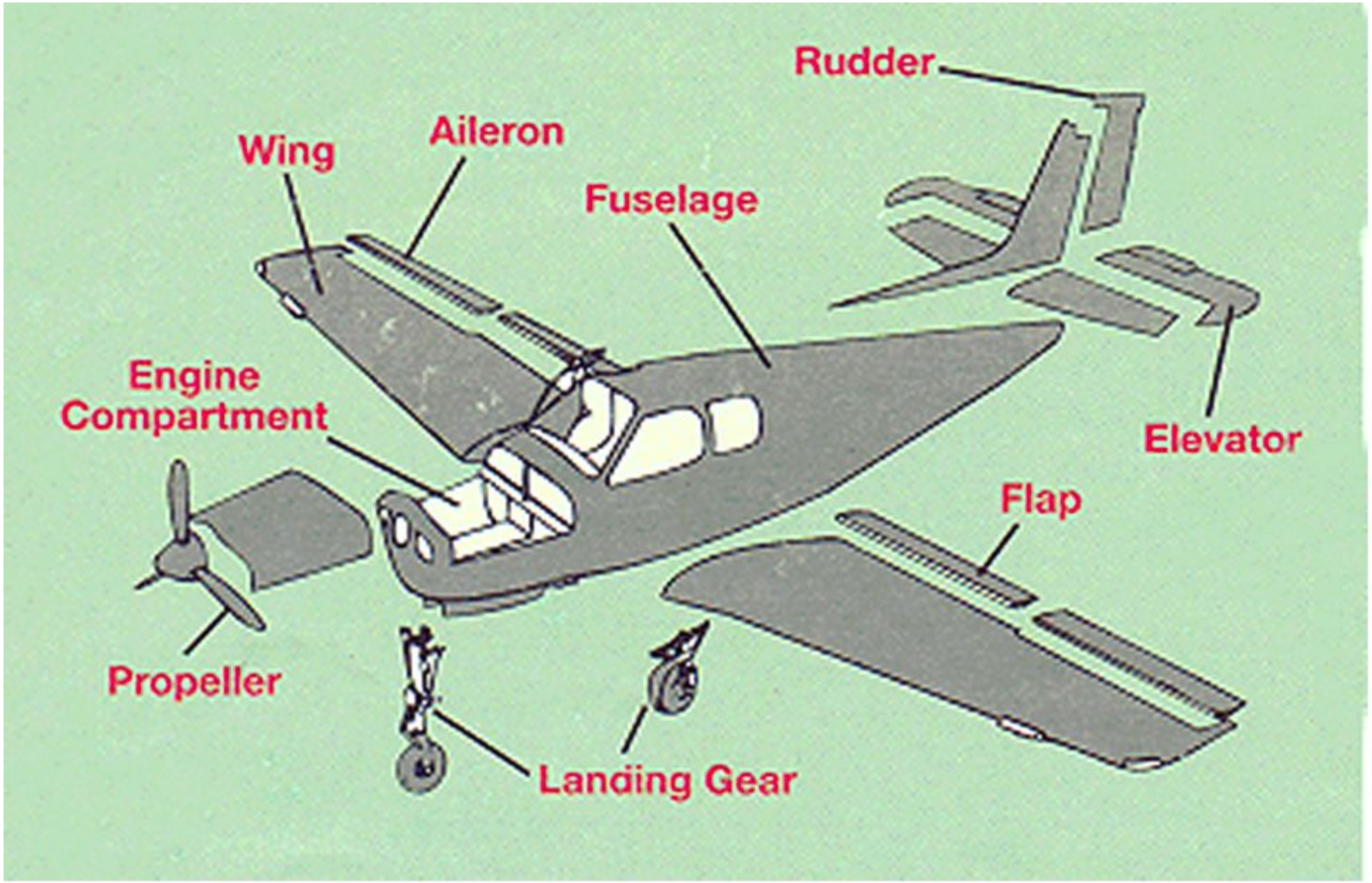
"Abstraction is the **elimination** of the irrelevant and the **amplification** of the essential."

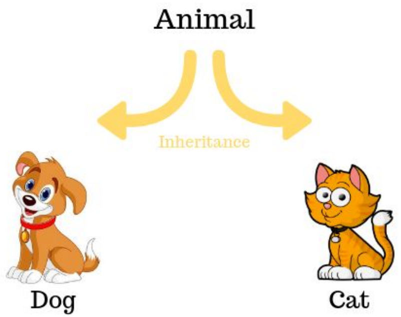
- *Bob Martin*





Thinking about "Concepts" and Big Picture

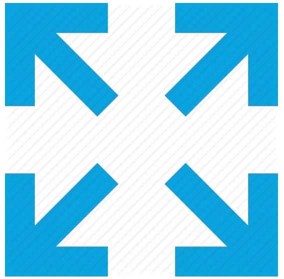


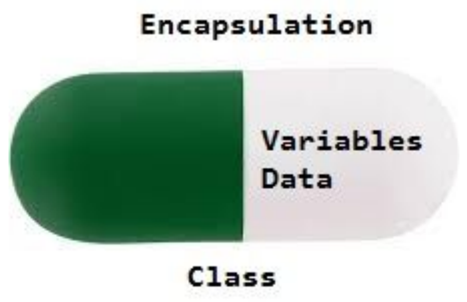


Inheritance: Reuse Structure, Behavior and Code and / or Extend, retrieve from parent.

Code Reuse

"Productivity"


Extend



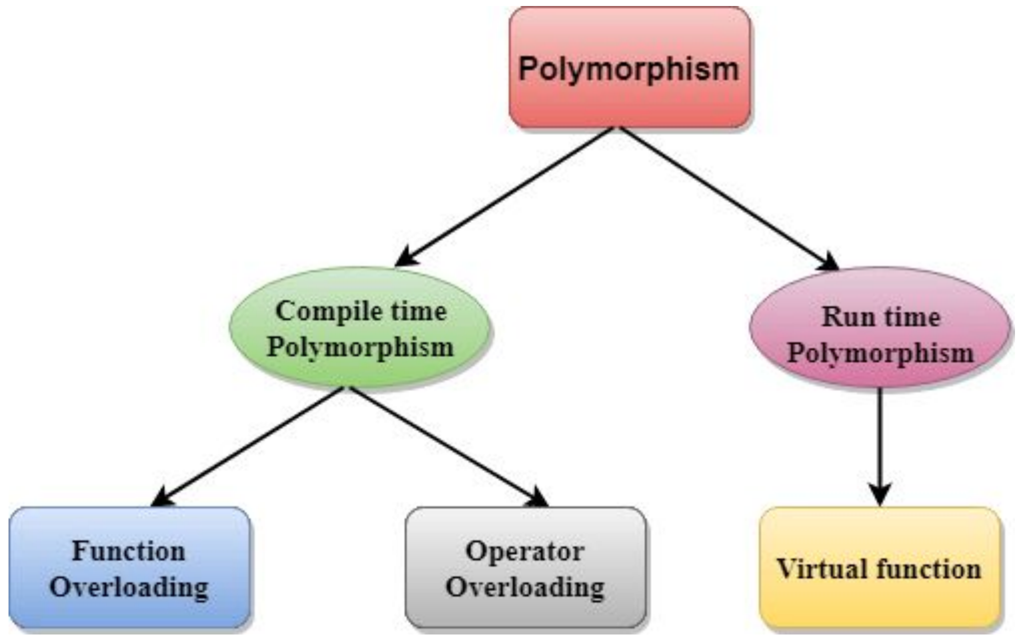
Encapsulation is like ISOLATION but in a SMALL CLASS level. Abstract State and Impl details.

Control

Hidden

Safety

Polymorphism: Best way to Kill IFs in OOP.

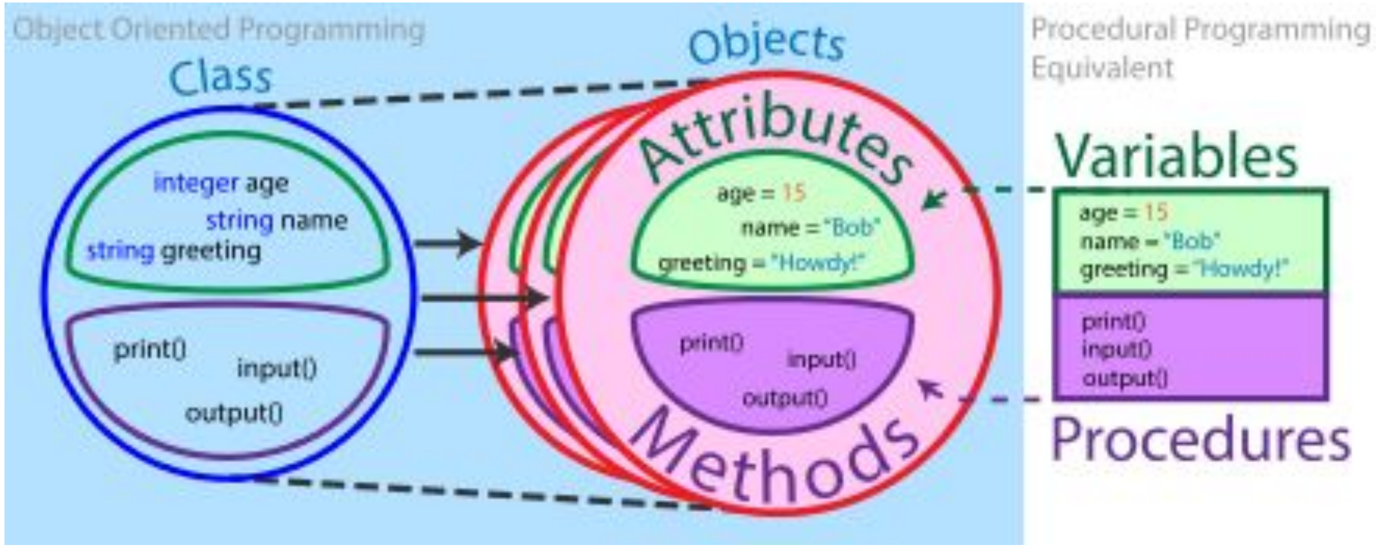


Flow Control

Extensibility

Isolation

OOP

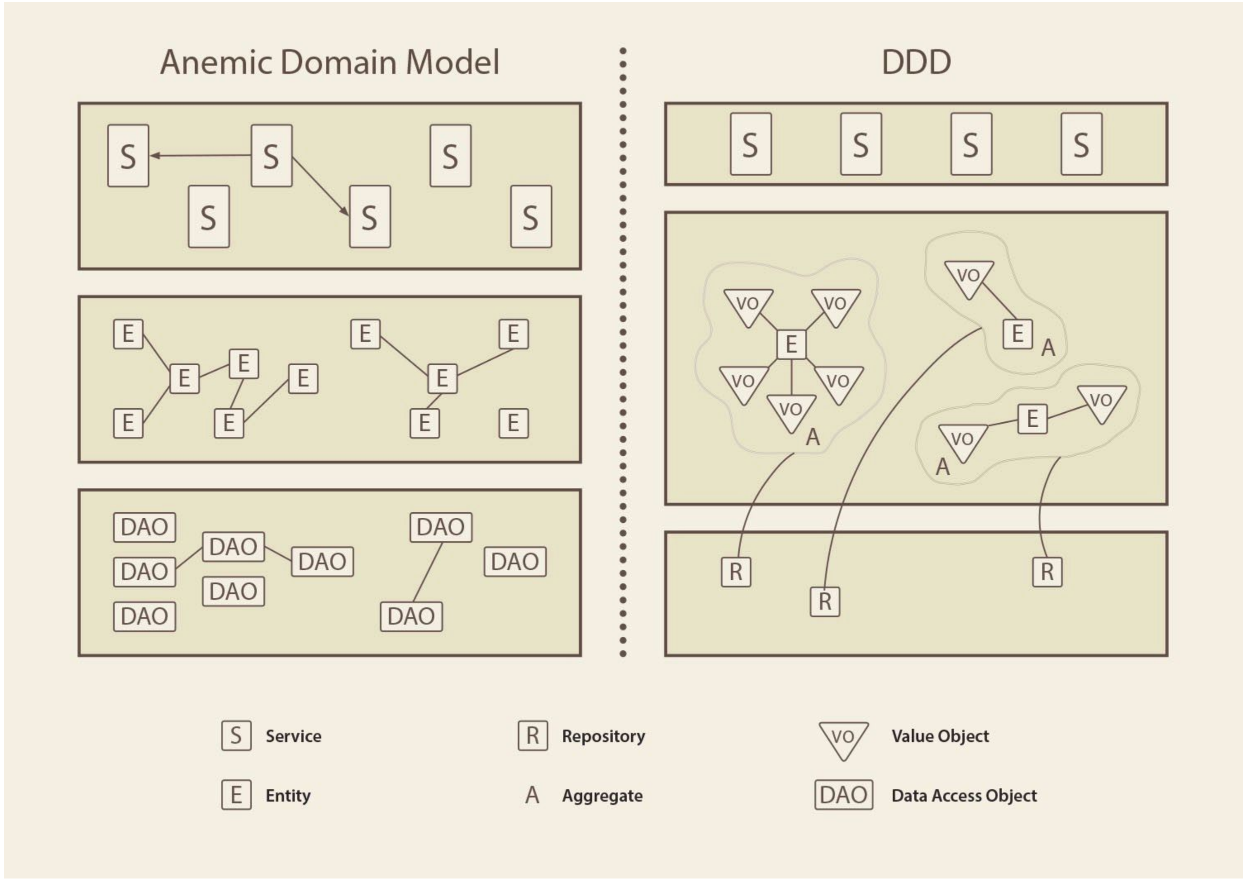


CLASS == Data + Functions

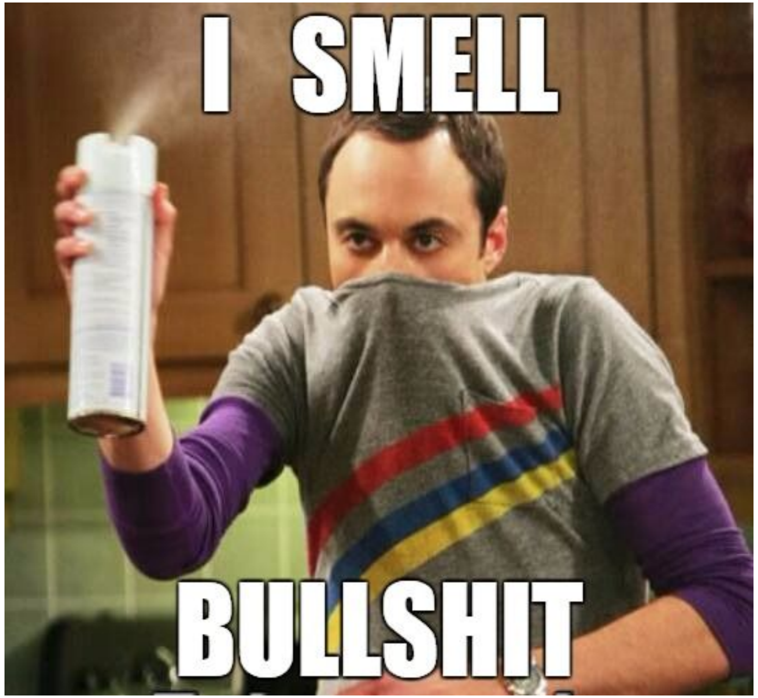
OOP by nature is very DDD

OOP == DDD // hold on, but that's not how we code...

DDD vs Anemic Model



IF smell like BS probably is...



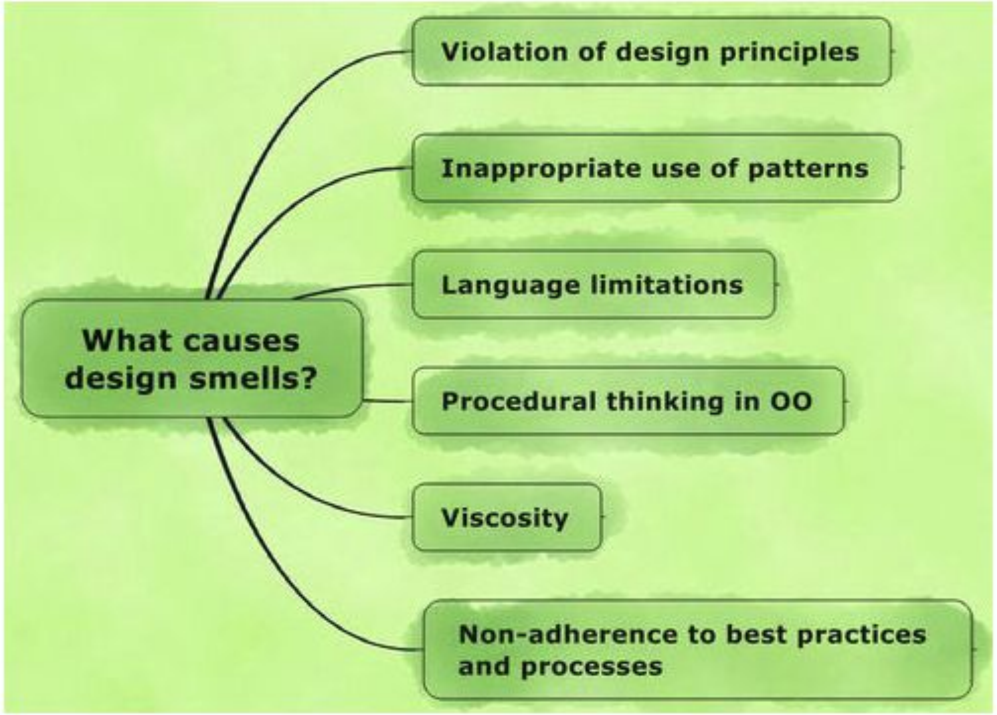

What causes design smells?
Violation of design principles
Inappropriate use of patterns
Language limitations
Procedural thinking in OO
Viscosity
Non-adherence to best practices and processes

FIGURE 2.1 Common causes of design smells.

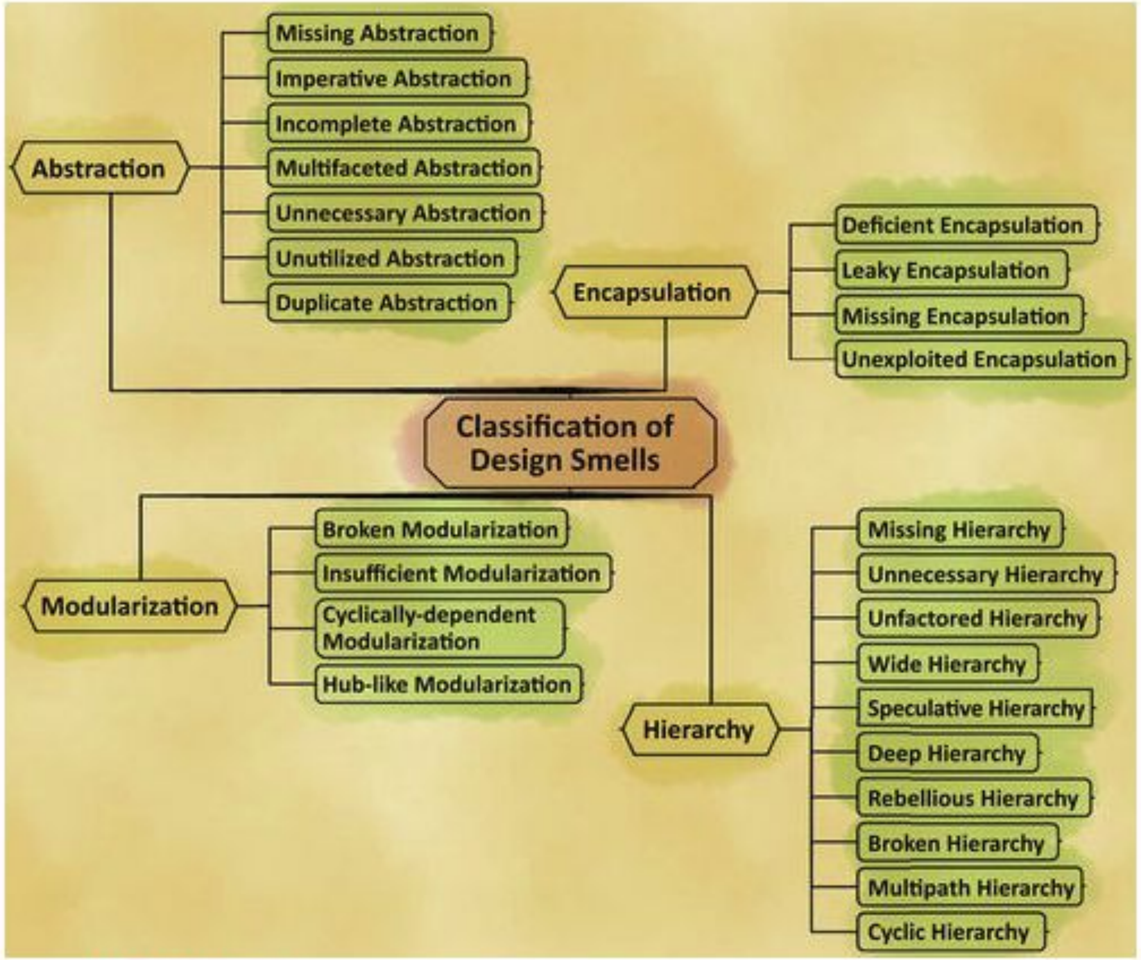
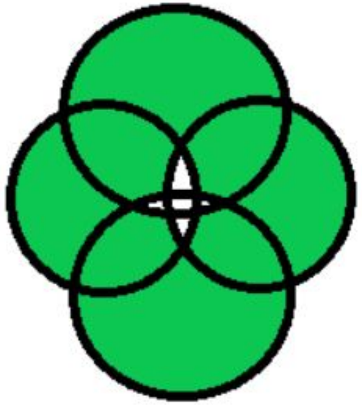
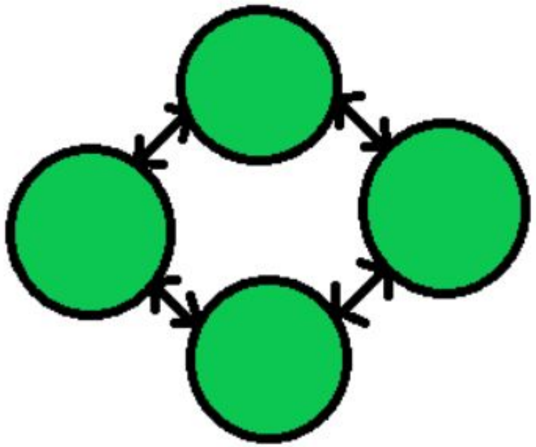


FIGURE 2.3 Classification of design smells.

Loose Coupling Principle





Tight coupling:

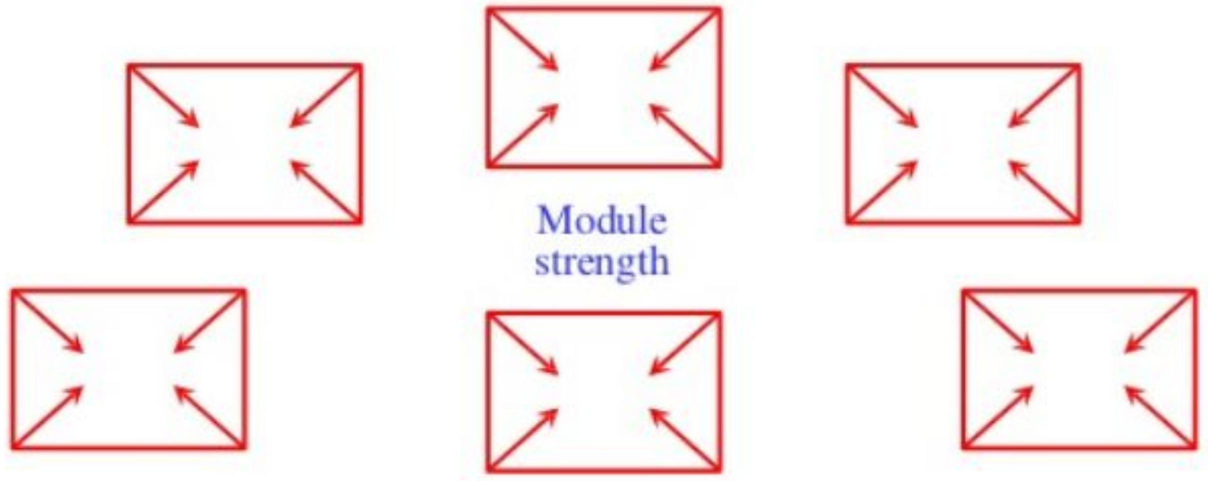
1. More Interdependency
2. More coordination
3. More information flow

Loose coupling:

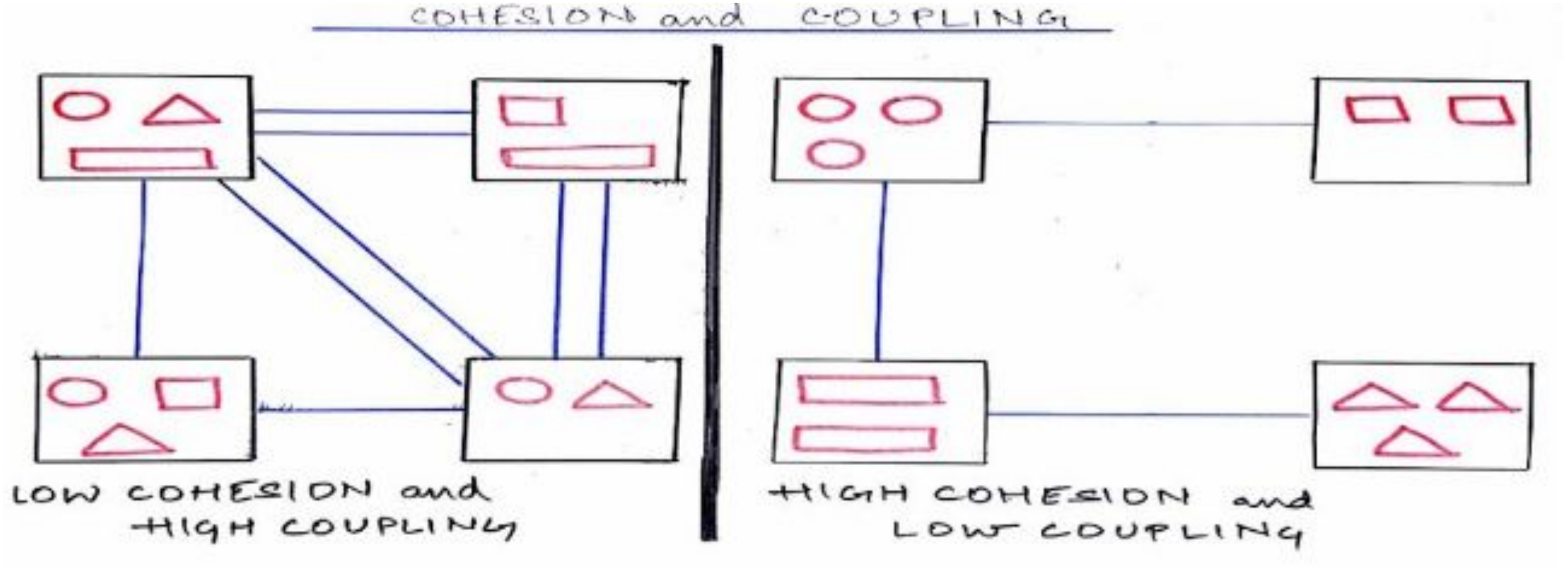
1. Less Interdependency
2. Less coordination
3. Less information flow

High Cohesion Principle

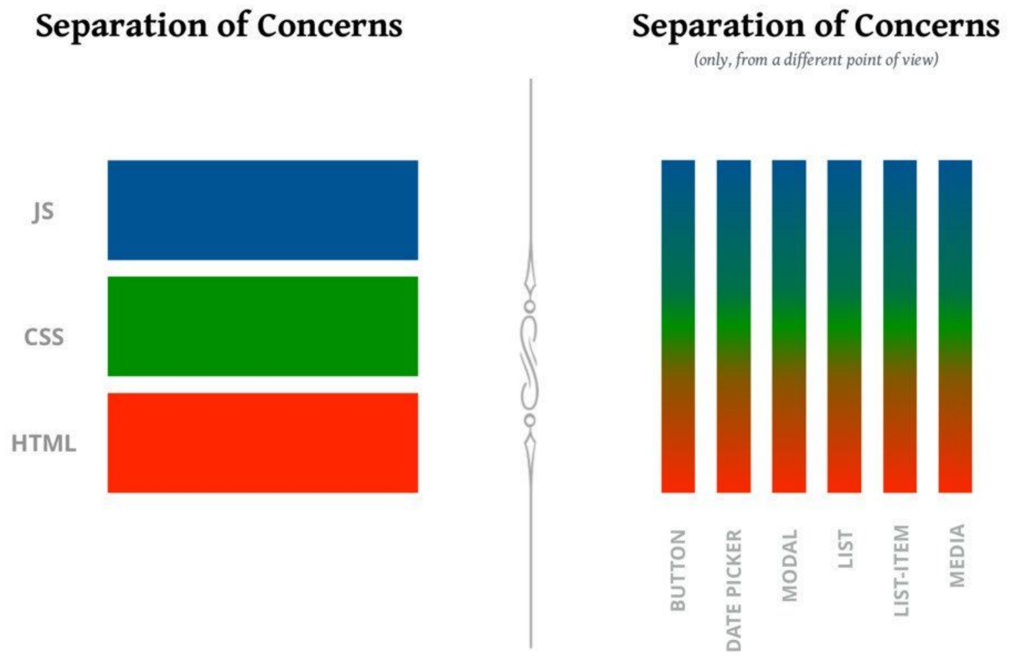
Cohesion is a measure of the degree to which the elements of a module are functionally related.



High Cohesion Principle


COHESION and COUPLING
LOW COHESION and HIGH COUPLING
HIGH COHESION and LOW COUPLING

SOC: Separation of Concerns



OOAD

- It's a methodology.
- Easily could lead to BDUF/Waterfall.
- The good this is:
  - Tell you to think before code
  - Consider
    - Analysis
    - Design
  - There is no right or wrong.

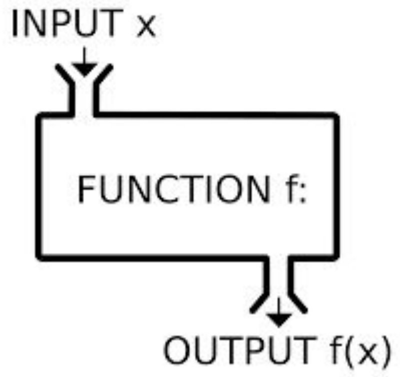
FP

There are no Classes

- ❑ Just Functions and Data together
- ❑ Wait that sound like OOP / DDD
- ❑ RE-use functions / focus on problem.
- ❑ Declarative not Imperative
- ❑ "forget" (for, ifs, vars)

FP

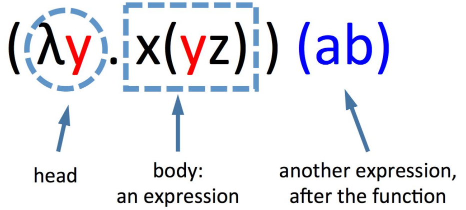
Functions are 1st class citizen



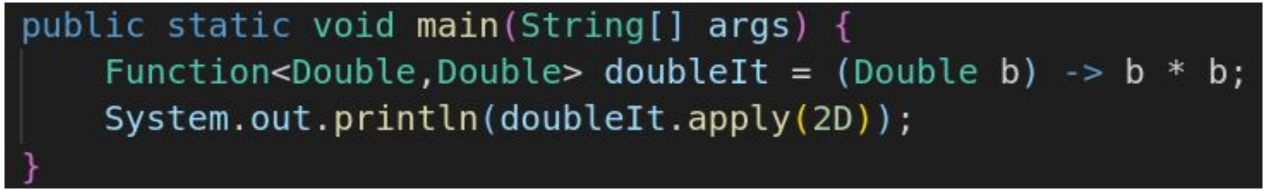
- ❏ Syntax Sugar / Abstractions
- ❏ Less Code / more focus on the problem.
- ❏ Great for algorithms / Computations.

FP

Lambda Calculus



- ❑ Formal Math System
- ❑ Alonzo Church in the 1930s
- ❑ Used in CS, Math, Philosophy, linguistics.

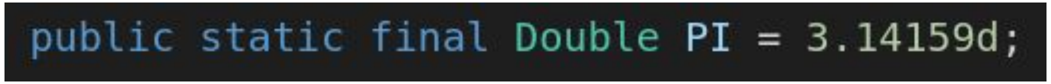


```
public static void main(String[] args) {
    Function<Double,Double> doubleIt = (Double b) -> b * b;
    System.out.println(doubleIt.apply(2D));
}
```

Immutability

π

- ❑ It's a like a constant, never change
- ❑ Mutations? New value / function
- ❑ Reduce bugs (shared global state)
- ❑ Super important for concurrency
- ❑ Java i.g: String, Integer, Double, Boolean, Byte (All Wrapper Classes)

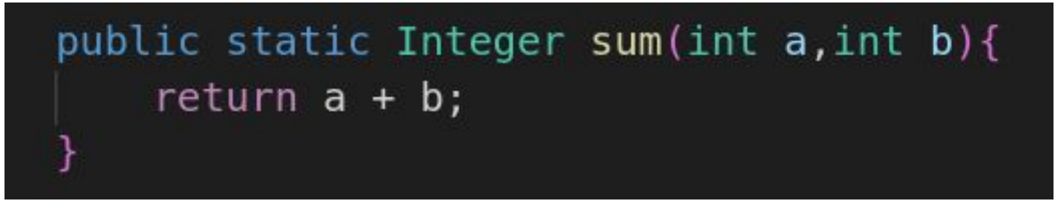


```java
public static final Double PI = 3.14159d;
```

FP

No Side Effects / Disciplined State

- ❑ Pure Functions
- ❑ Same input, same results
- ❑ No Side Effects (IO)

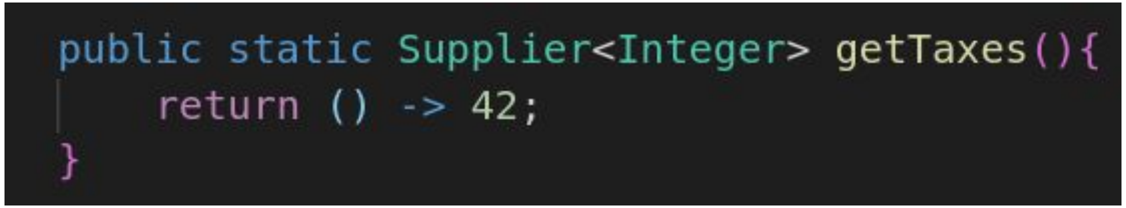


```java
public static Integer sum(int a,int b){
    return a + b;
}
```

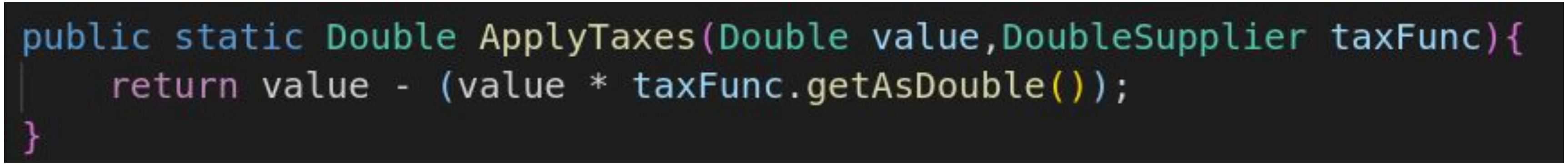
FP

High Order Functions

- ❑ Function as Data Types
- ❑ Pass Functions as Parameters
- ❑ Return functions from Functions
- ❑ Great for Laziness and Composition

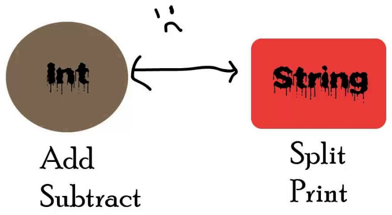


```java
public static Supplier<Integer> getTaxes(){
    return () -> 42;
}
```



```java
public static Double ApplyTaxes(Double value,DoubleSupplier taxFunc){
    return value - (value * taxFunc.getAsDouble());
}
```

FP



Type System

- ❑ Set of Rules
- ❑ Relational Algebra
- ❑ Relies on the Compiler
- ❑ Reduce Bugs / improve safety
- ❑ ...If we go to an extreme you can't code and your system will break in production anyway.

Exercises

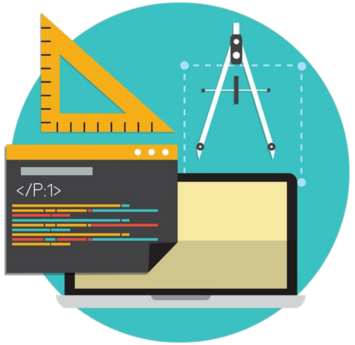
TAX processing system: The system need process different TAX per product per state. TAX table:

CAR: RS: 40%, SP: %30%, SC:10%

Food: RS: 50%, SP: %40%, SC:10%

Beer: RS: 60%, SP: %20%, SC:10%

Given a list of products in a Sales OS the system should be able to tell how much TAX we will need pay buy product. We also need to produce a report with: A) top 3 sells, B) top 1 state charging taxer per products, C) top 3 salesman. The system also should be responsible for generate TAX reports for the go in the FLAT FILE format: ID | STATE | PRODUCTID | SALE | TAX. You need provide **2 Designs & Implementation with Java 8** (OOP and FP).



OOP / Design
OOAD / FP Principles

Diego Pacheco