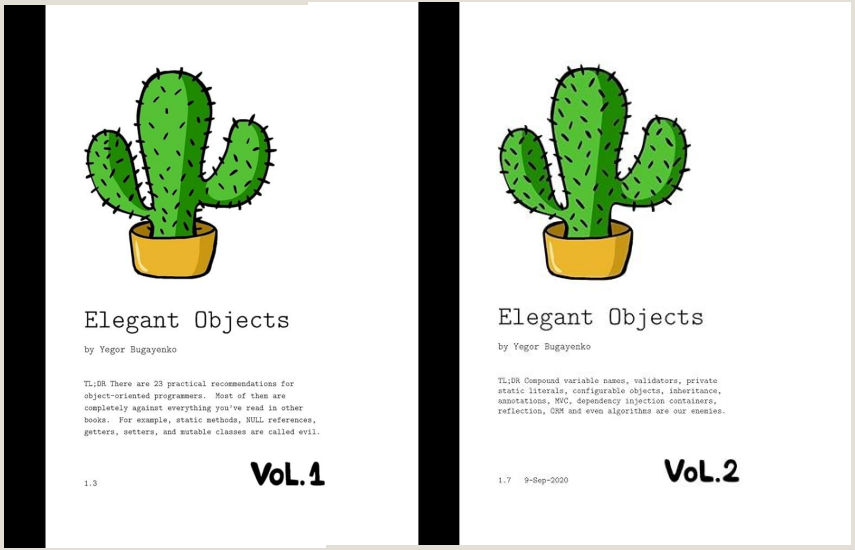
# Proper OOP

Diego Pacheco

# Let’s talk about Java.

OOP
>
Java

# Java is even Fully OOP.




# Markething Effect
# +
# Industry Trend
# ==
# Java or OOP?

**Objection 1.** Data structure and functions should not be bound together

**Objection 2.** Everything has to be an object.

**Objection 3.** In an OOPL data type definitions are spread out all over the place.

**Objection 4.** Objects have private state.


Group Chats ...


https://www.cs.otago.ac.nz/staffpriv/ok/Joe-Hates-OO.htm

Never Breaking APIs

+

Huge Community + Enterprise

==

A lot of value

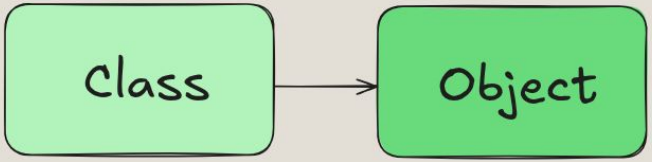
# Agile is not about...

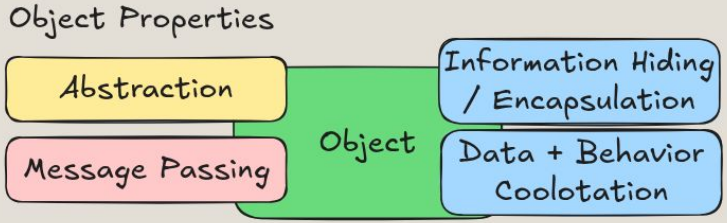
- ❑ JIRA
- ❑ Scrum masters
- ❑ Story Points
- ❑ SAFE
- ❑ Meetings

# OPP is not about...

- ❑ Inheritance (subtyping is ok)
- ❑ Polymorphism (nice and cool)
- ❑ Getters/Setters
- ❑ Enums
- ❑ Complexity (lead by bad abstractions)

# Classes vs Objects



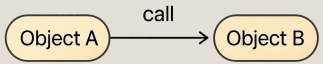




- A way to get objects
- Classes are not the problem.
- Functions, classes, packages, jars, services are just options.
- Who does not like options?

# Principles
## (or maybe key properties)

Encapsulation / Information Hiding

Data + Behavior Colocation




Message Passing

Proper Abstractions

# Principles
# Data + Behavior Colocation

Data + Objects Colocation

- ❏ Core of OOP and DDD
- ❏ What usually break this principle: enums, getters/setters, ifs, switch, annotations.
- ❏ Breaking this principle == Procedural / Imperative Programing (what FP fights)

# Principles
# Data + Behavior Colocation

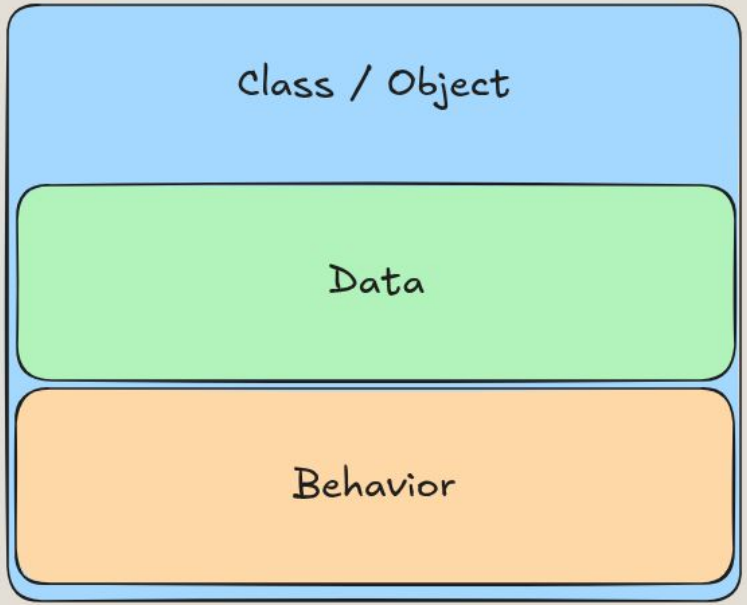
Data + Behavior Colocation

- ❏ Classes are often “Naked”
- ❏ … also known as “Anemic Model”
- ❏ Which leads to leaking abstractions (code into the client - if/enum effect).

https://martinfowler.com/bliki/AnemicDomainModel.html
https://github.com/diegopacheco/java-pocs/tree/master/pocs/proper-oop-media-poc

# Principles
# Data + Behavior Colocation

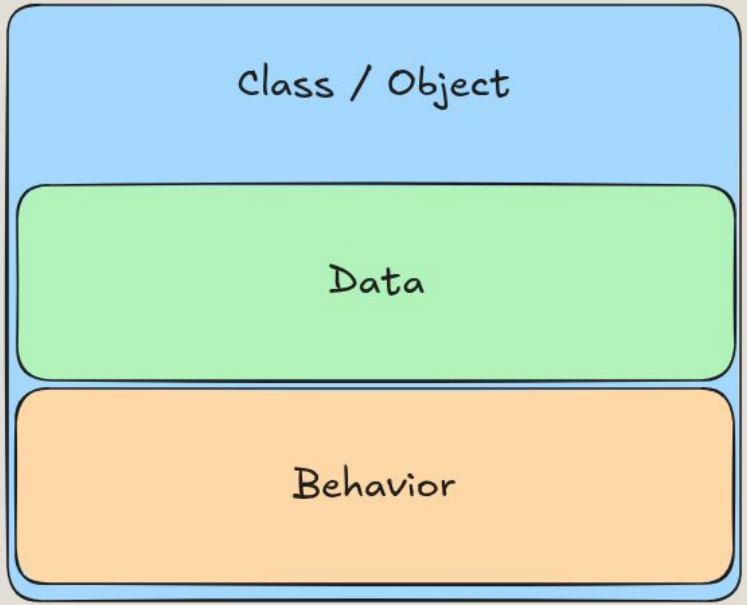
Data + Behavior Colocation

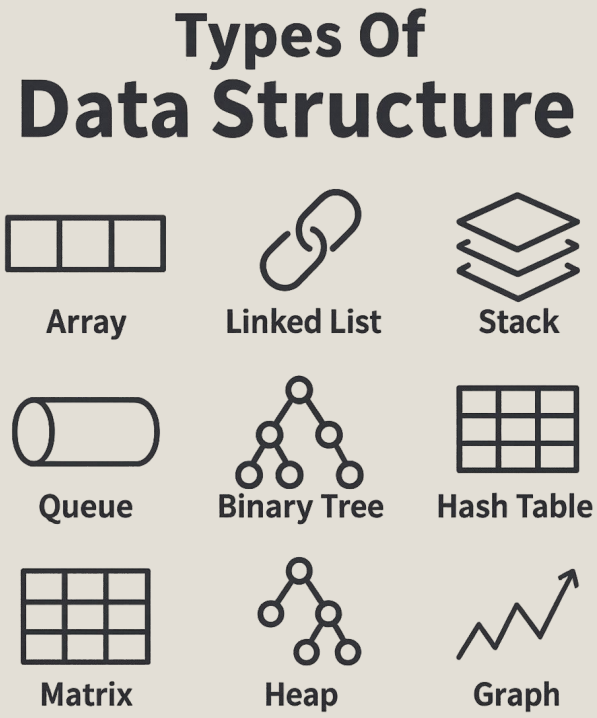



- Usually not followed.
- When done right is powerful and always desired.

# Principles
# Data + Behavior Colocation

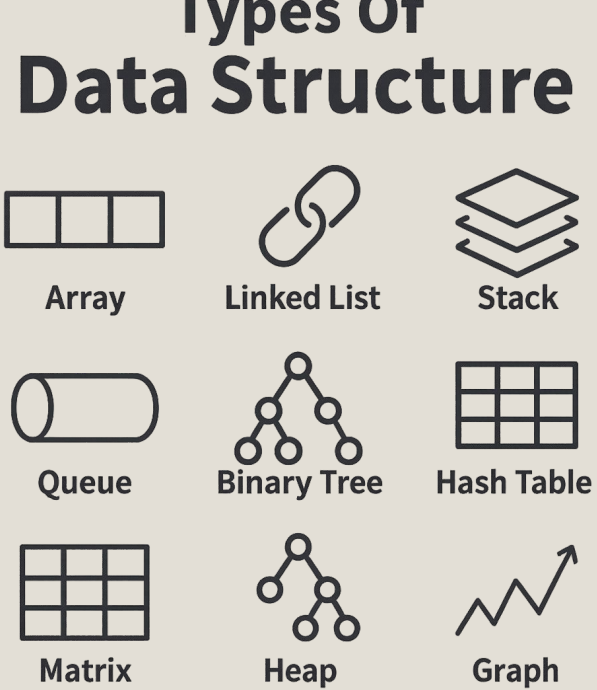
Data + Behavior
Colocation



# Principles
# Data + Behavior Colocation

Data + Behavior Colocation




- Generic push(oop) vs global push(fp). Centralized vs distributed.
- Standard sdk vs Business Programs

# Principles
# Encapsulation + Information Hiding

Encapsulation / Information Hiding

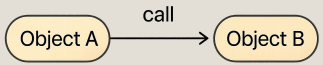
- ❏ Hide Data
- ❏ Hide Behavior
- ❏ Hide Complexity
- ❏ Provide value
- ❏ “Do Something”
- ❏ … delegate often is a anti-pattern.

# Principles Encapsulation + Information Hiding

Encapsulation / Information Hiding

- ❏ Code duplication is lack of this principle
- ❏ Better systems, better design requires moving things around.
- ❏ Think about the consumer (ifs, switch, exceptions, return, void, enums).
- ❏ Tip: Play with separate/together.
- ❏ Reuse should not happen by accident

# Principles
# Message Passing




Message Passing

- ❏ Delivers Decoupling
- ❏ It’s a protocol
- ❏ It’s a contract
- ❏ Every class has it.
- ❏ Requires a lot of thinking.
- ❏ Has consequences.

# Principles
# Proper Abstractions

Proper
Abstractions

- ❏ Hard to do it
- ❏ When is wrong:
  - ❏ Leaking details
  - ❏ Everything breaks the contract
  - ❏ Violates the other principles
  - ❏ When we do things we don't understand

# Principles
# Proper Abstractions

Proper
Abstractions

- ❏ Abstractions require knowledge
- ❏ Abstractions require thinking
- ❏ Abstractions can break over time
- ❏ When done right is desirable.
- ❏ Humans (and AI) can break anything that is good and turn into bad.

# Misunderstood (Used wrong)

## Step 1

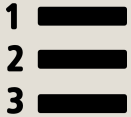
Polymorphism when Force

Reube by hierarchy.

## Step 2

Getters / Setters

Because is the “standard”.




## Step 3

Enums

Nulls

Aspects

All over the place

## Step 4

Don’t learn proper OOP Design

Barely talk about Design

Just ignore Design

## Step 5

Mass adoption
Lots of anti-patterns
Over decades
Just “Append”
Result in bad abstractions
And Complexity.


## Goal

Replicate

Things you don't

Understand.

No questions

asked!

# Root Issues

| | How manifest | How we'll respond |
|---|---|---|
| 1 | Bad Abstractions - By applying patterns people do not understand. | 1. Understand core principles first<br>2. Understand patterns (well)<br>3. Lots of practices and mentorship |
| 2 | Lack of Critical Thinking - By no time to think, just more features, no time to refactoring. | 1. Make time to think<br>2. Compare options with tradeoffs<br>3. Do tons of Pocs and Think (a lot)<br>4. Review with mentor |
| 3 | Lack of proper Learning - by no trainings, not retrospectives, no hard conversations, or just LGTM. | 1. Prioritize Learning (all times)<br>2. Do formal trainings<br>3. Do hard review and critique all that is in place<br>4. Review with mentor |

# Benefits Proper OOP

## Dy doing  it right you get

### Maintenance

Code that can be better maintained. Therefore easier to change and the evolve the system(s) over time.

### Decoupling and Cohesion

System(s) get more robust, they don't break easier with refactorings, testing is easier, things can be changed independently with less merges.

### Proxy to Happiness

Working in code that you love, and you are proud is the best thing ever. Always refactoring that code and keeping this state is true definition of purity and Goodness.

## The solution

# How to make it better

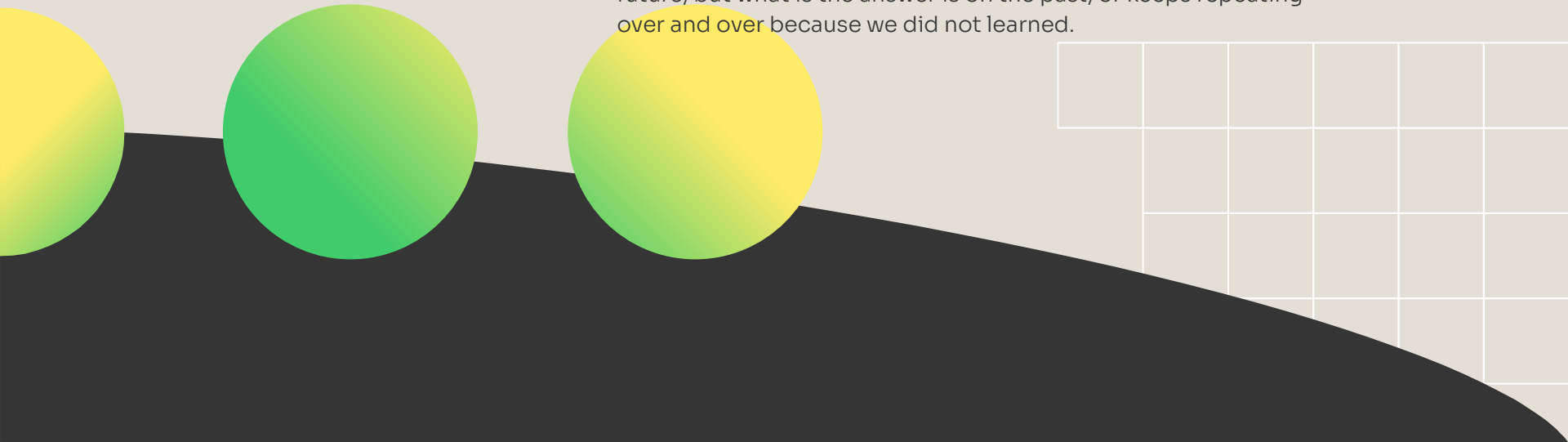
1. **Make sure you understand the core principles:** Is you read you did not understand, is you practice 100x and got review and approved by mentor - them you get it.
2. **Tradeoff different options**: Always do design POCs prsatice with different options, together vs apart, pattern A vs Pattern B - don't take things for granted.
3. **Do lots of Drawing**: UML or not, what matters is to think, review what is behind what your doing, always understand 10 levels deeper.

# The solution

**4. Practice**: Do it before coding something. What is the first thing you do after understanding the problem? It should be the design, how you do design? Just hack the exist code? What is your design process?

**5. Beware of 6 and half-dozen**: React now has no classes, but really, how much a component is different from a class? Having no classes does no fix the problem, having classes also dont fix the problem (classitis).

**6. Learn about old things**: We always think the answer is on the future, but what is the answer is on the past, or keeps repeating over and over because we did not learned.



# Thank you

# Proper OOP

Diego Pacheco