

# Object-Oriented

## Analysis and Design

YEGOR BUGAYENKO

Lecture #4 out of 16

80 minutes

The slidedeck was presented by the author in this [YouTube Video](#)

All visual and text materials presented in this slidedeck are either originally made by the author or taken from public Internet sources, such as web sites. Copyright belongs to their respected authors.

**Before:**

Structured Programming  
C, Pascal, FORTRAN, ALGOL  
CASE  
Waterfall

**After:**

Object-Oriented Programming  
C++, Object Pascal, Java  
RUP

Object Model by Grady Booch

SOLID Principles by Uncle Bob

Design by Contract by Bertrand Meyer

Don't Repeat Yourself (DRY)

You Ain't Gonna Need It (YAGNI)

Inversion of Control

OOP vs. Functional Programming

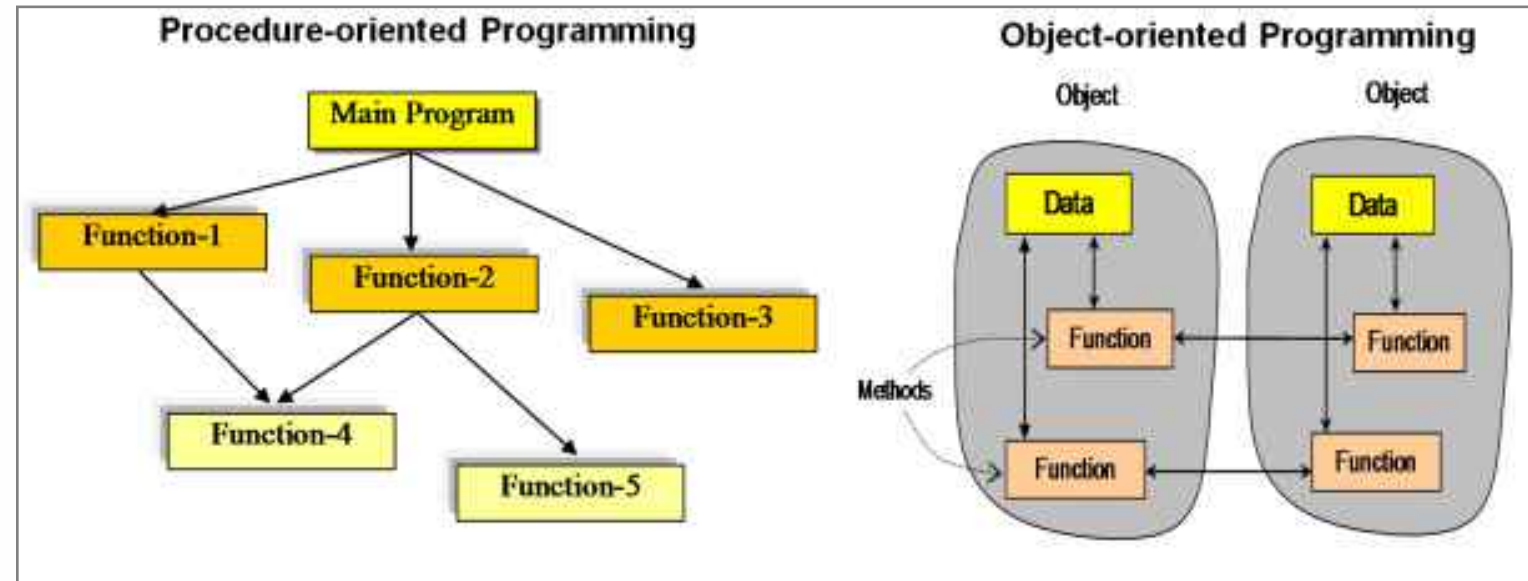
Books, Venues, Call-to-Action

Chapter #1:

# Object Model by Grady Booch

[ [Abstraction](#) Encapsulation Modularity Hierarchy ]

# Abstraction



[ [Abstraction](#) Encapsulation Modularity Hierarchy ]

## Some Principles of OOP

Abstraction

Encapsulation

Modularity

Hierarchy

Polymorphism

Inheritance

Composition

Delegation

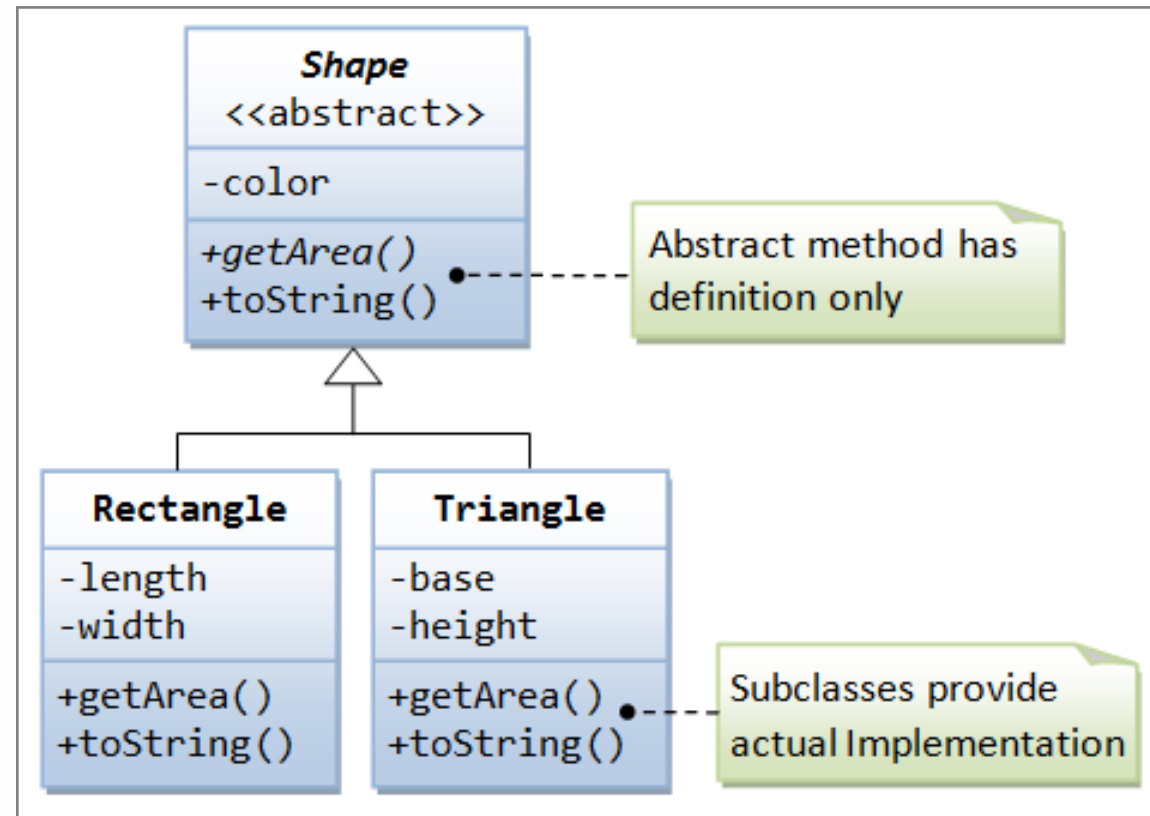
Subtyping

Data Hiding

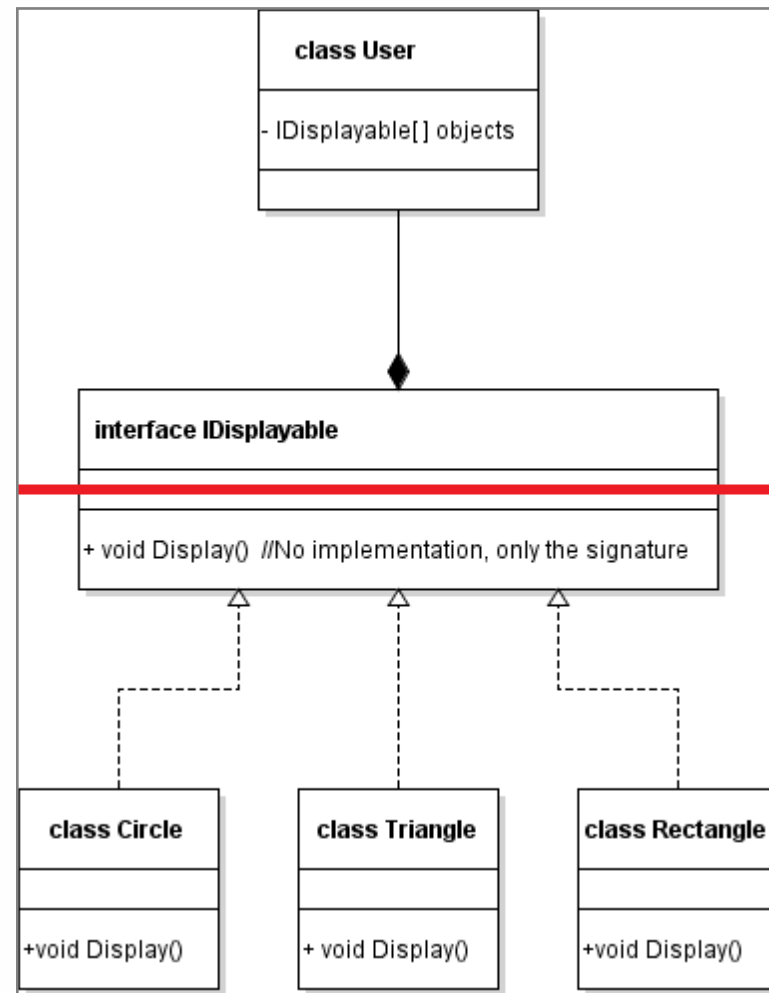
Separation of Concerns

[ [Abstraction](#) Encapsulation Modularity Hierarchy ]

## Abstraction & Polymorphism



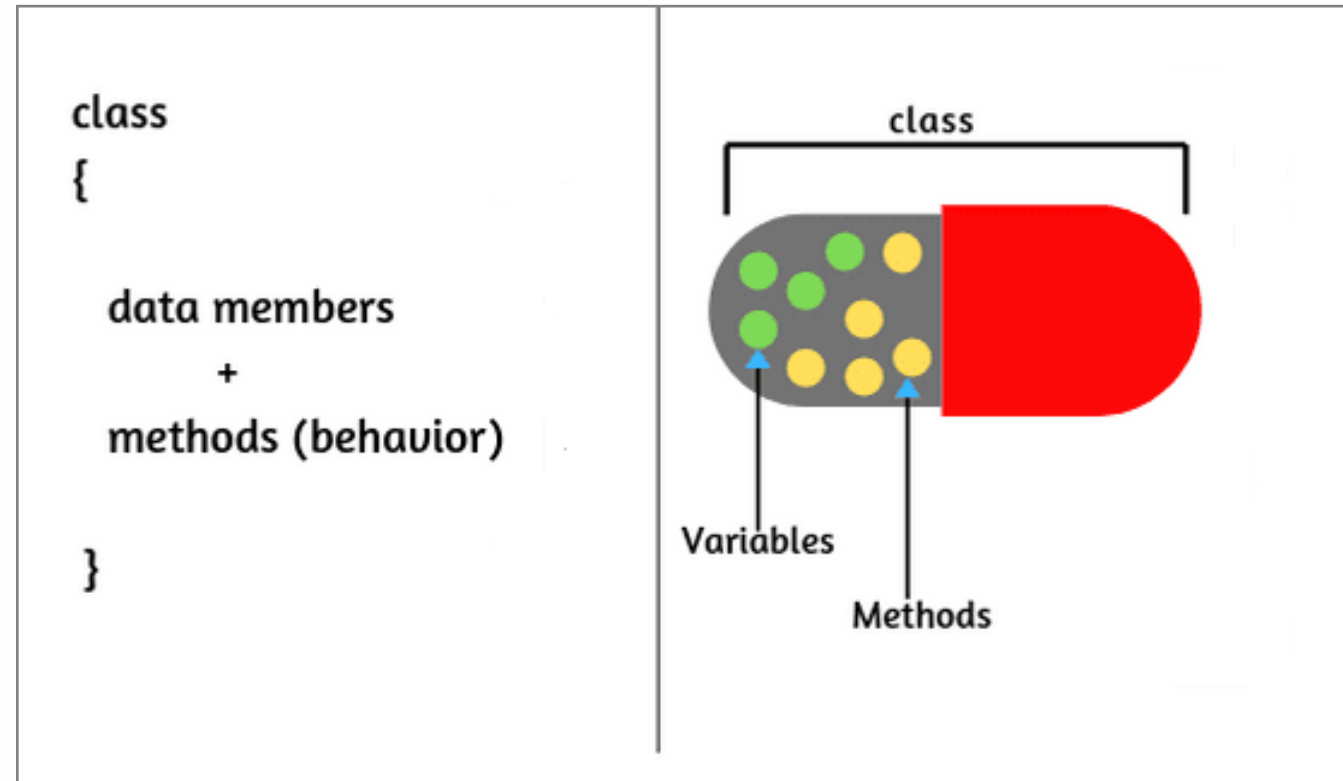
[ Abstraction Encapsulation Modularity Hierarchy ]



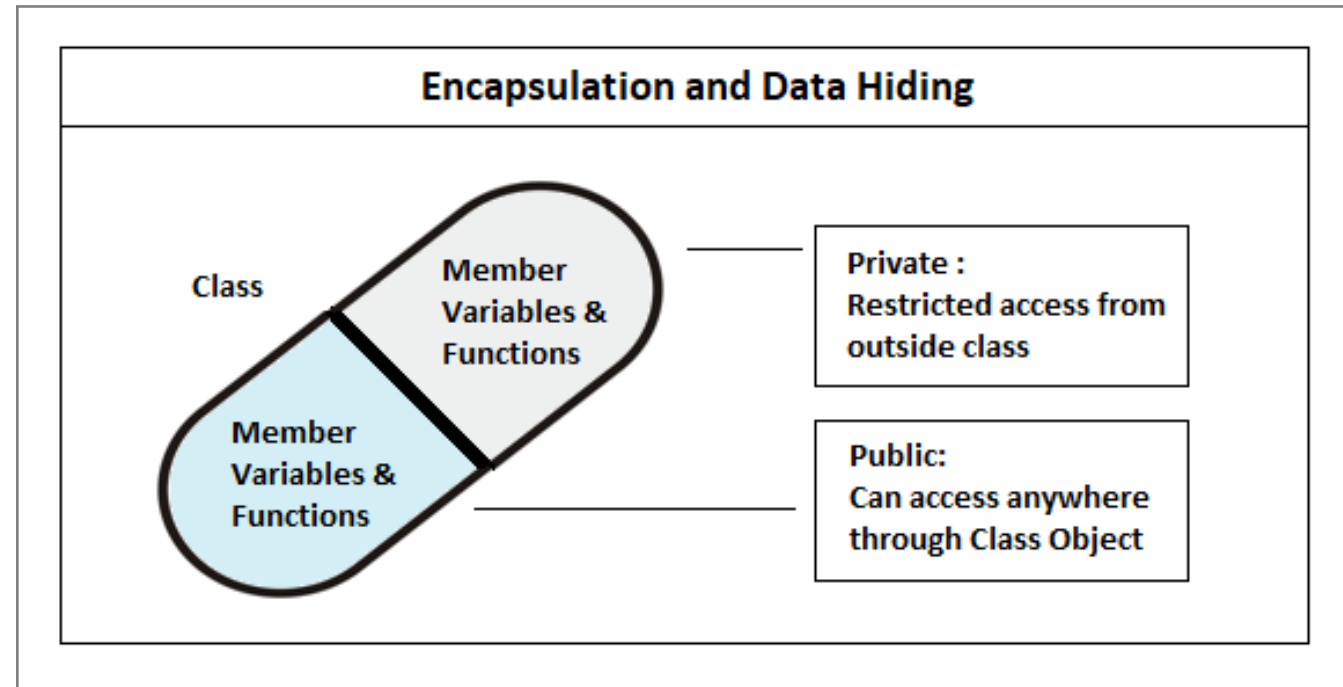


[ Abstraction [Encapsulation](#) Modularity Hierarchy ]

# Encapsulation



[ Abstraction [Encapsulation](#) Modularity Hierarchy ]



[ Abstraction [Encapsulation](#) Modularity Hierarchy ]

## “Encapsulation Covers Up Naked Data” (2016)

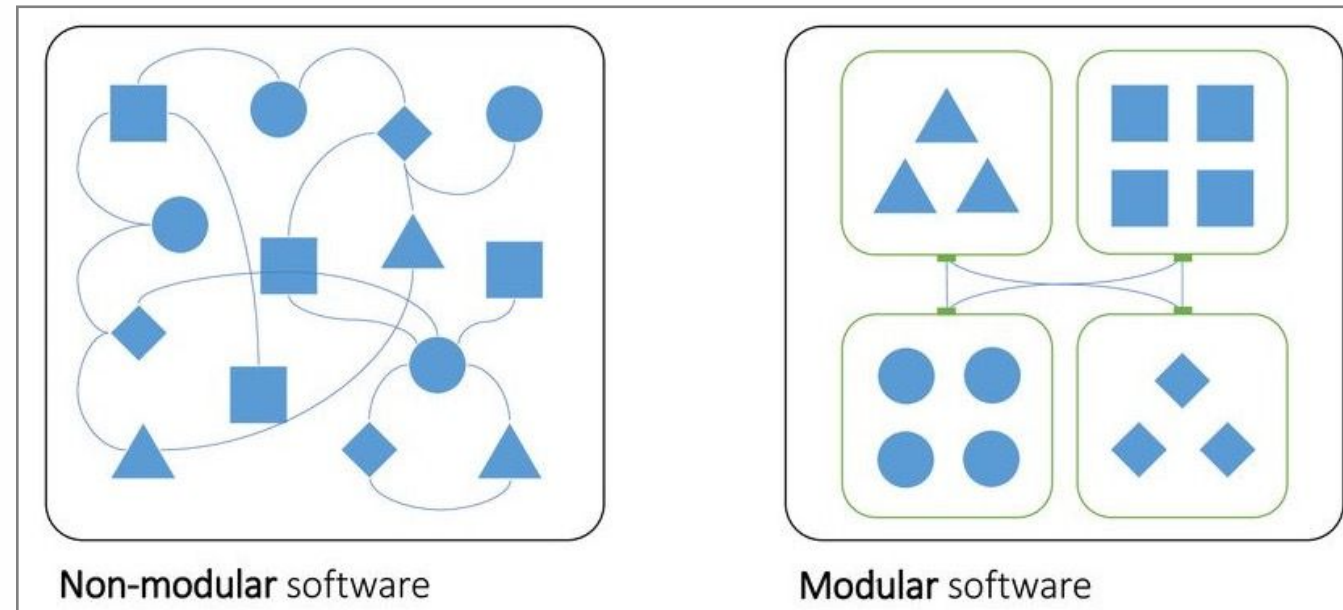


```
class Temperature {  
    private int t;  
    public int getT() { return this.t; }  
    public void setT(int t) { this.t = t; }  
}
```

```
class Temperature {  
    private int t;  
    public String toString() {  
        return String.format("%d F", this.t);  
    }  
}
```

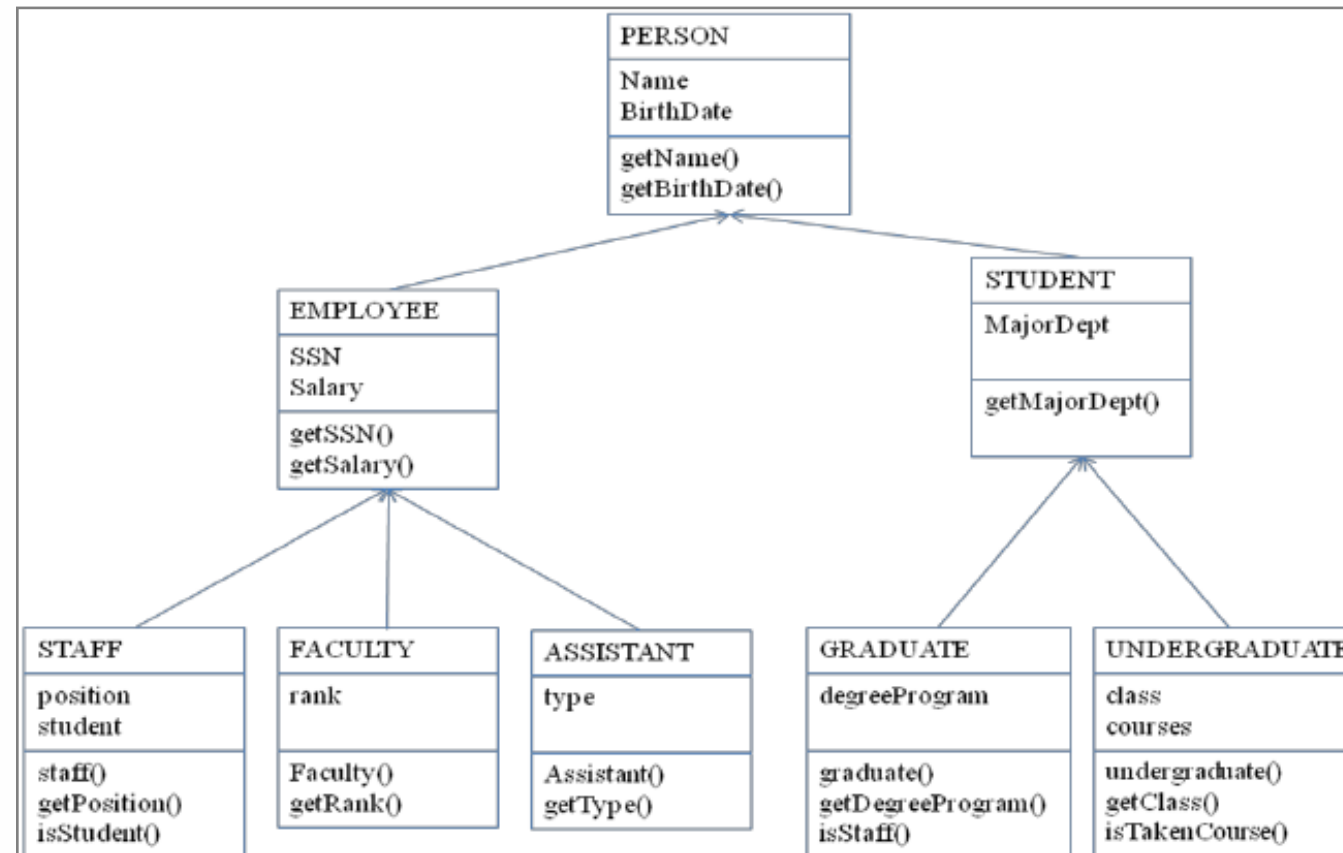
[ Abstraction Encapsulation [Modularity](#) Hierarchy ]

## Modularity



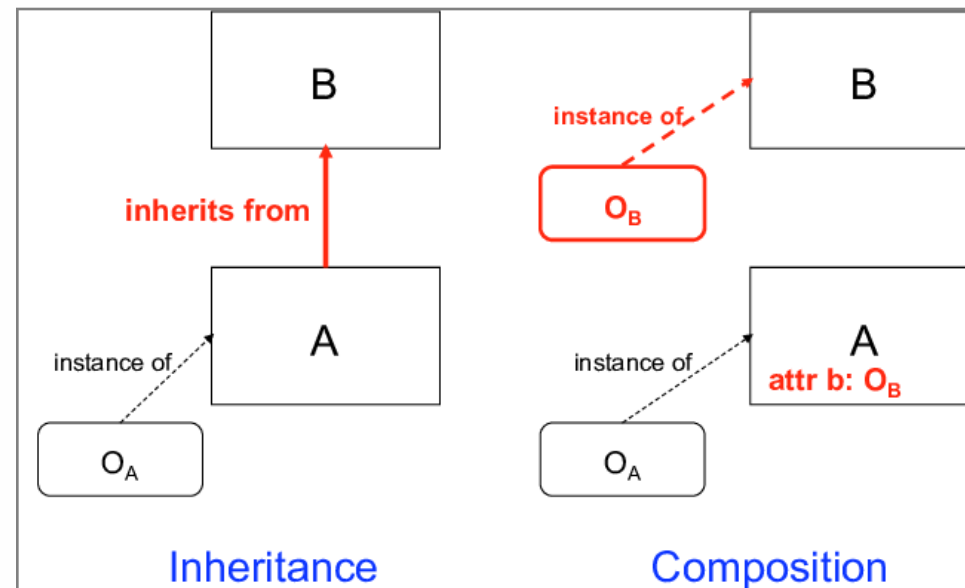
[ Abstraction Encapsulation Modularity [Hierarchy](#) ]

# Hierarchy



[ Abstraction Encapsulation Modularity [Hierarchy](#) ]

## Inheritance vs. Composition



[ Abstraction Encapsulation Modularity [Hierarchy](#) ]

“Prefer composition over inheritance?” (2008) at StackOverflow



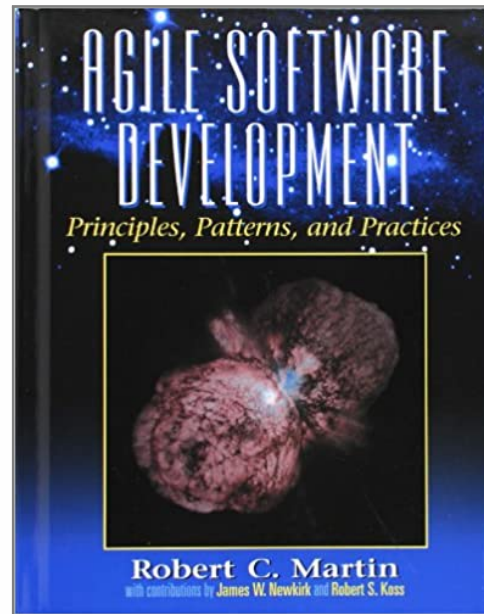
“Inheritance Is a Procedural Technique for Code Reuse” (2016)



Chapter #2:

## SOLID Principles by Uncle Bob





“The attitude that agile developers have toward the design of the software is the same attitude that surgeons have toward sterile procedure. Sterile procedure is what makes surgery possible. Without it, the risk of infection would be far too high to tolerate. Agile developers feel the same way about their designs.”

— *Agile Software Development. Principles, Patterns, and Practices*, Robert Martin

## **SOLID** principles:

- **S**ingle Responsibility Principle
- **O**pen Close Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle



“Robert C. Martin is most recognized for developing many software design principles and for being a founder of the influential Agile Manifesto” — Wikipedia

<https://www.cleancoder.com>

[@unclebobmartin](#) on Twitter

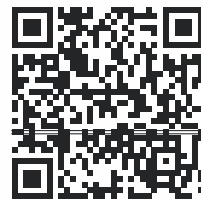
## Single Responsibility Principle (SRP)

“A class should have only one reason to change.”

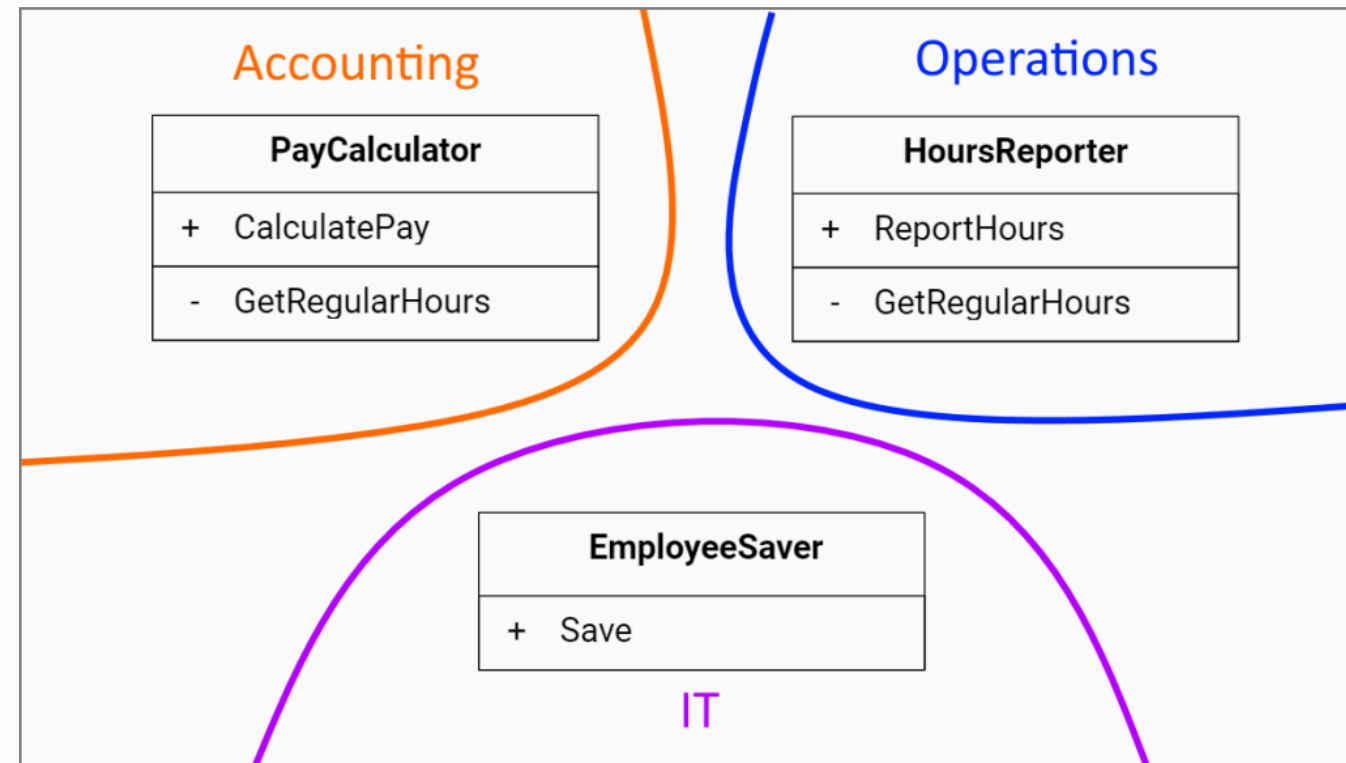
“Every module, class or function in a computer program should have responsibility over a single part of that program’s functionality, and it should encapsulate that part.”

“Each module should be responsible for each role.”

“SRP is a Hoax” (2017)

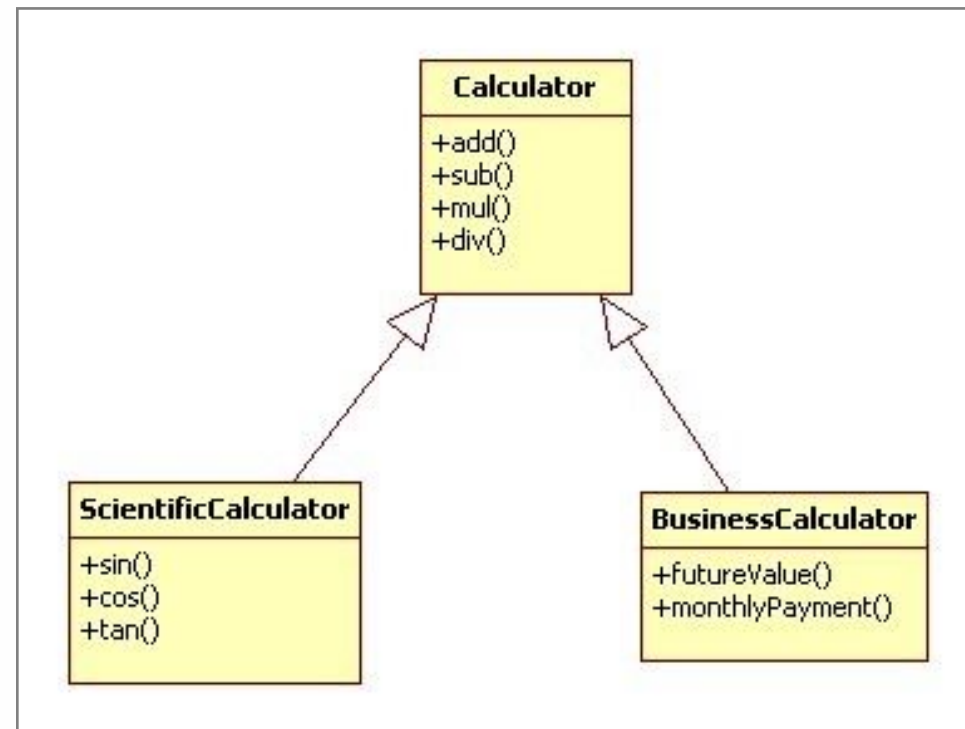


[ [SRP](#) OCP LSP Duck ISP DIP ]



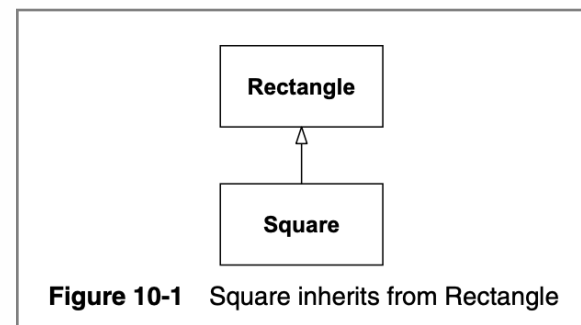
## Open-Close Principle (OCP)

“Software entities should be open for extension, but closed for modification.” — Robert Martin



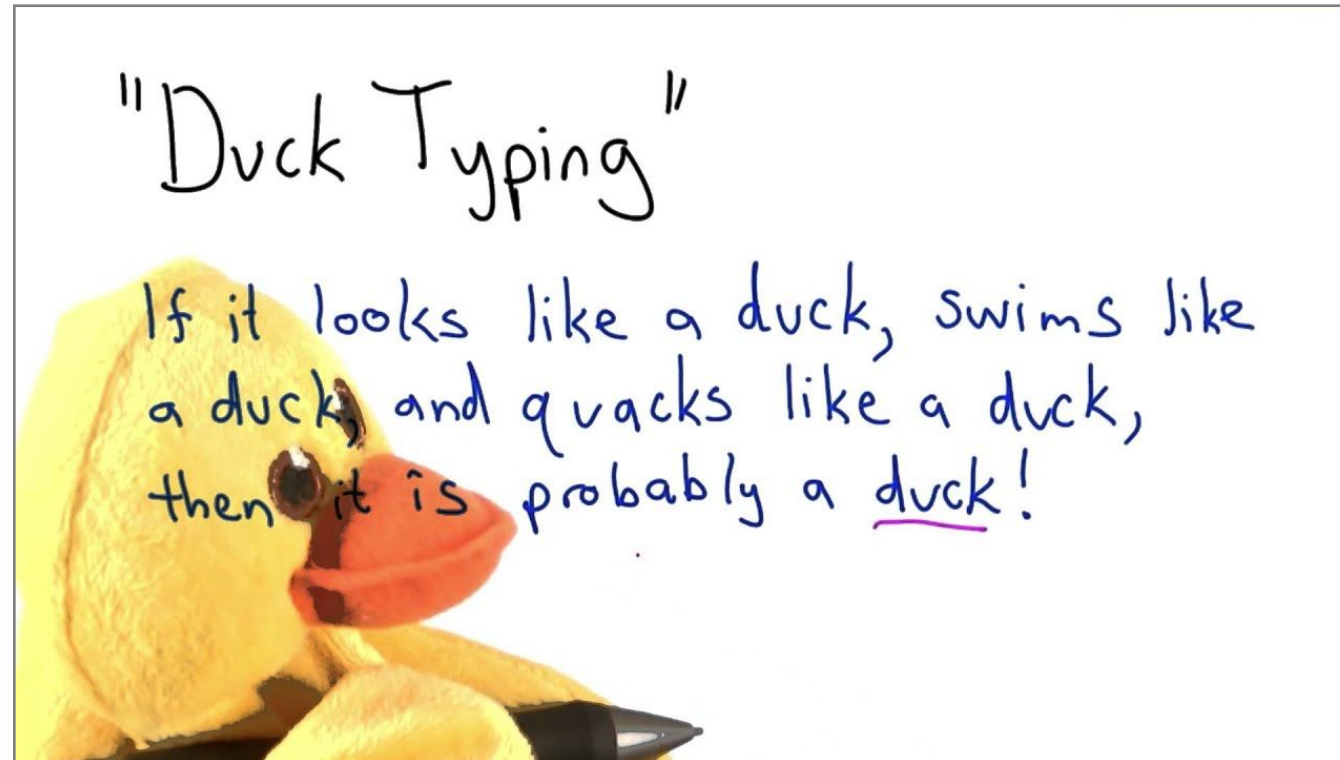
## Liskov Substitution Principle (LSP)

“If for each object  $o_1$  of type  $S$  there is an object  $o_2$  of type  $T$  such that for all programs  $P$  defined in terms of  $T$ , the behavior of  $P$  is unchanged when  $o_1$  is substituted for  $o_2$  then  $S$  is a subtype of  $T$ .” — Barbara Liskov



“The LSP makes it clear that in OOD, the IS-A relationship pertains to behavior that can be reasonably assumed and that clients depend on.”

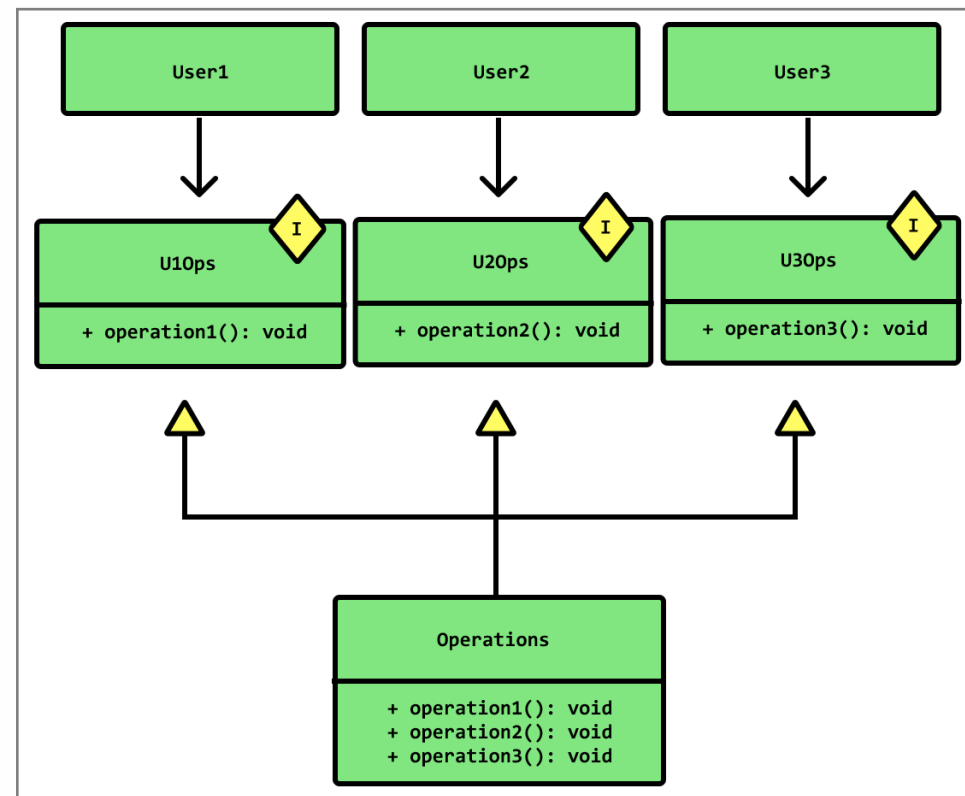
## Duck Typing





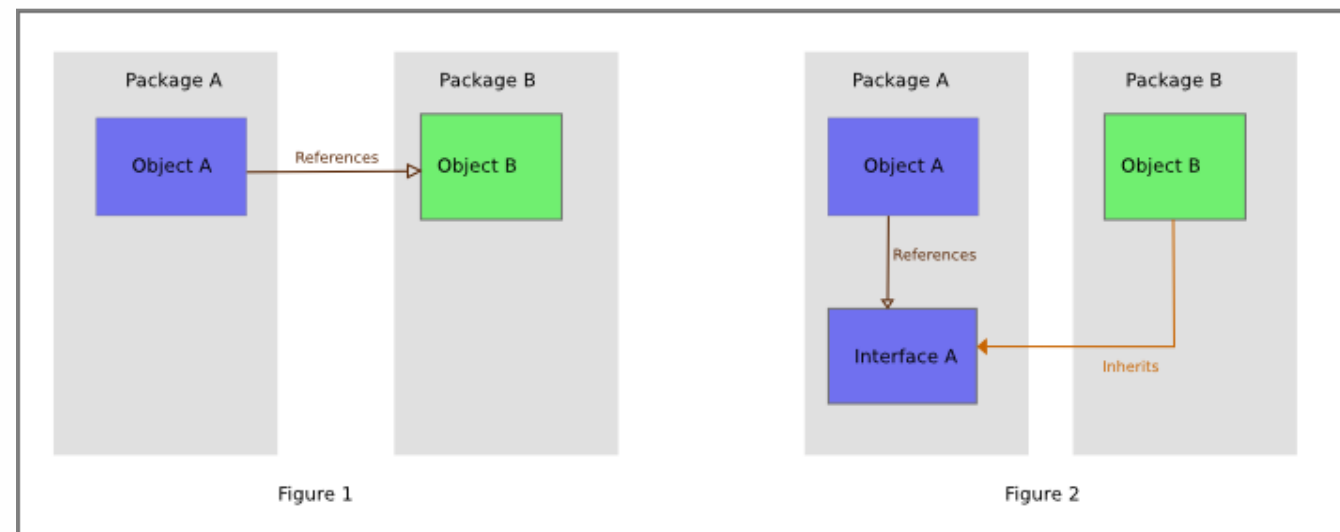
## Interface Segregation Principle (ISP)

“Clients should not be forced to depend on methods that they do not use.”  
— Robert Martin



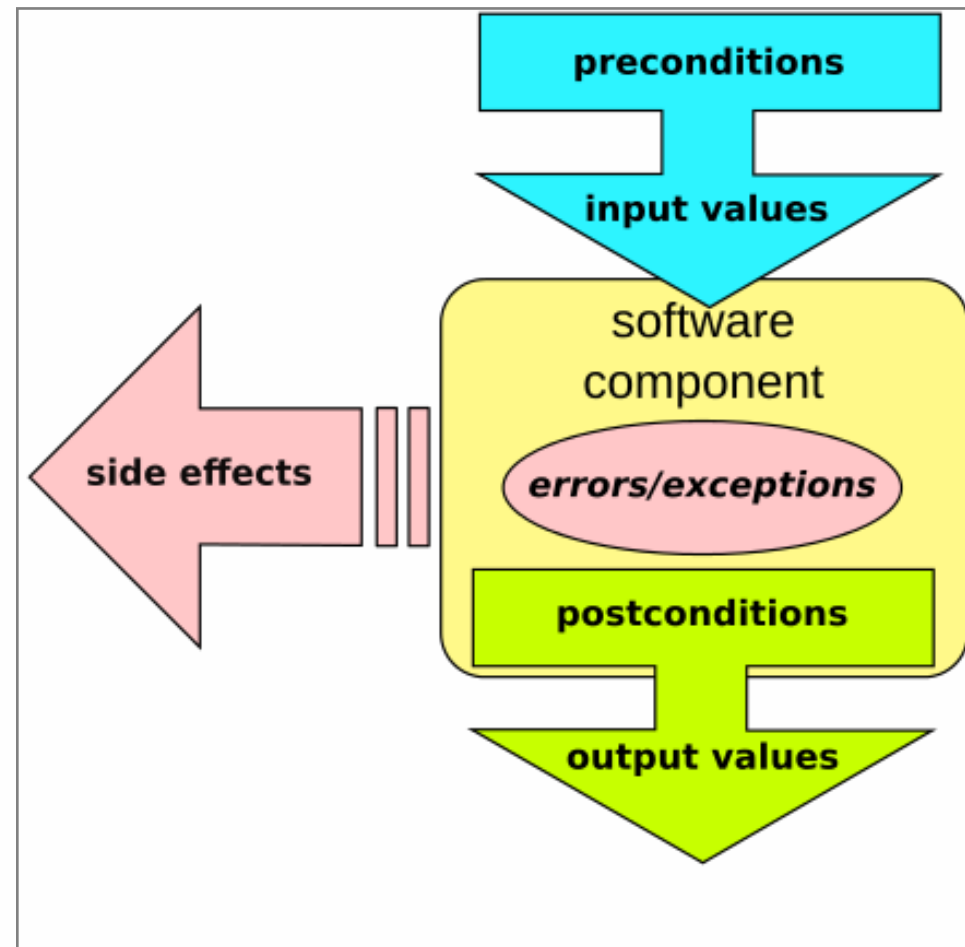
## Dependency Inversion Principle (DIP)

“a) High-level modules should not depend on low-level modules. Both should depend on abstractions. b) Abstractions should not depend on details. Details should depend on abstractions.” — Robert Martin



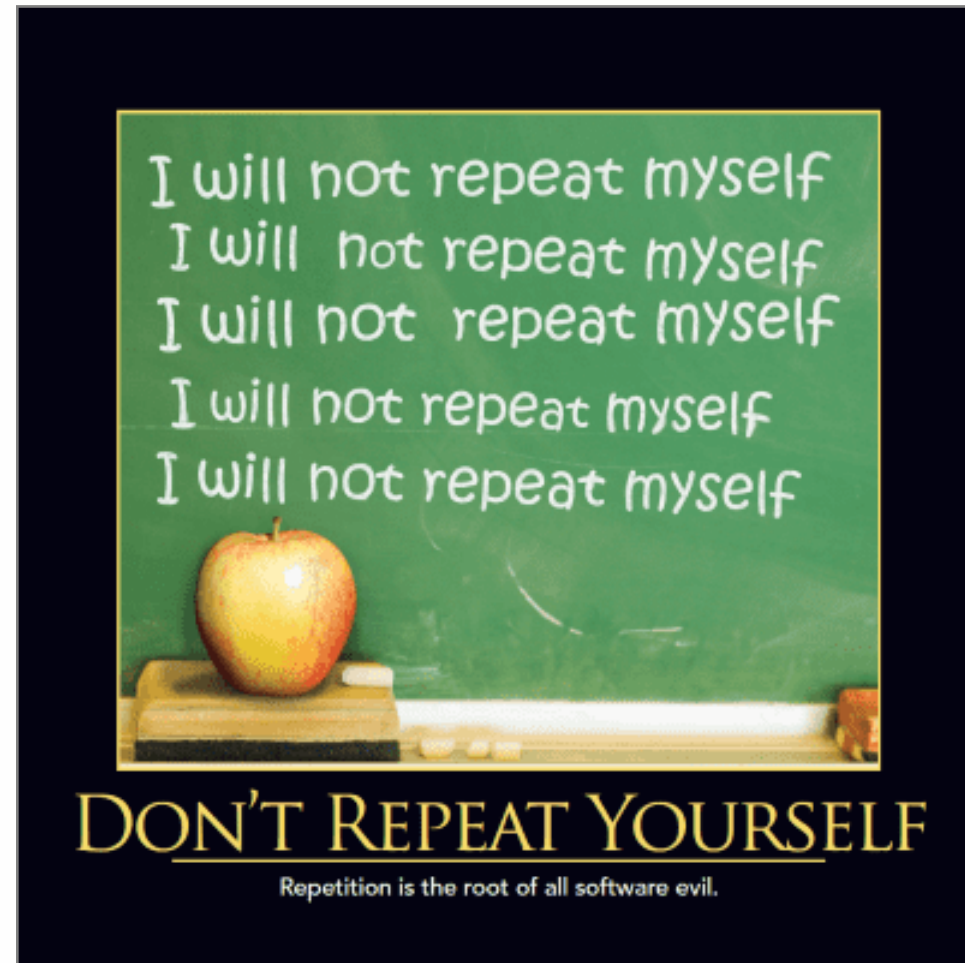
Chapter #3:

## Design by Contract by Bertrand Meyer



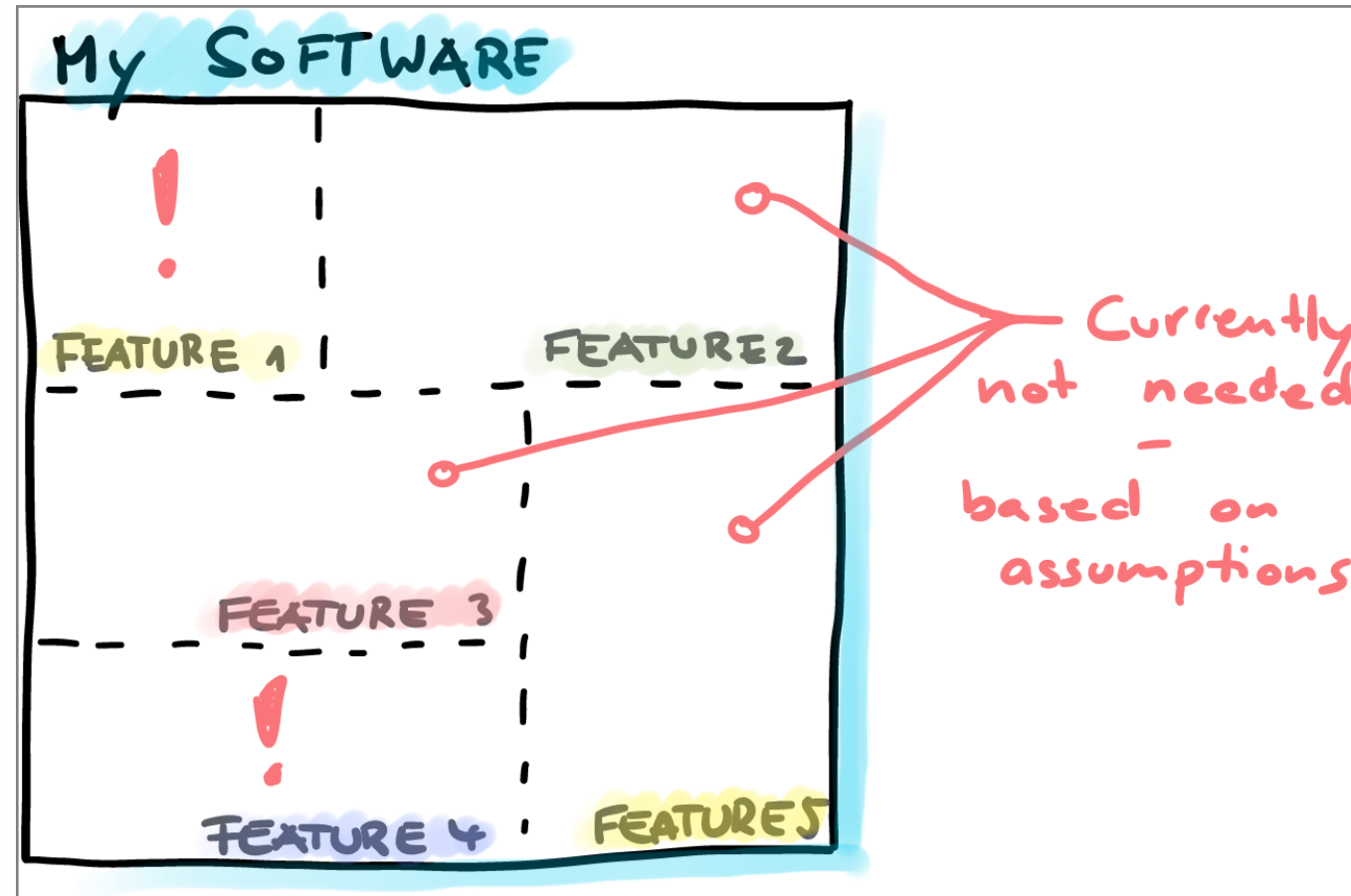
Chapter #4:

## Don't Repeat Yourself (DRY)



Chapter #5:

## You Ain't Gonna Need It (YAGNI)





Chapter #6:

## Inversion of Control

## “How Does Inversion of Control Really Work?” (2017)



Look at this code:

```
| print(book.title());
```

It is very straight forward: we retrieve the title from the book and simply give it to the `print()` procedure, or whatever else it might be. *We* are in charge, the *control* is in our hands.

In contrast to this, here is the *inversion*:

```
| print(book);
```

We give the entire book to the procedure `print()` and it calls `title()` when it feels like it. That is, we *delegate* control.

Chapter #7:

## OOP vs. Functional Programming

### **OO pattern/principle**

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

### **FP pattern/principle**

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Oh my, functions again!
- Functions
- Functions ☐

(c) Uncle Bob



Edsger W. Dijkstra (1989)

“TUG LINES,” Issue 32, August 1989

“Object oriented programs are offered as alternatives to correct ones” and “Object-oriented programming is an exceptionally bad idea which could only have originated in California.”



Alan Kay (1997)

The Computer Revolution hasn't happened yet

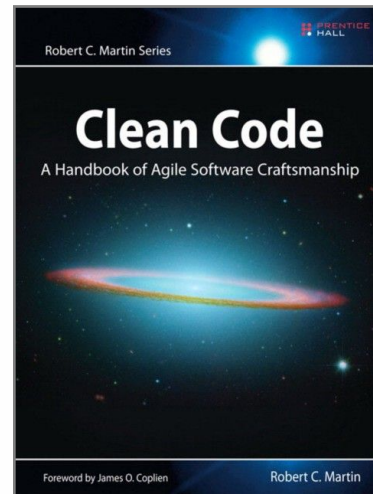
“I invented the term object-oriented, and I can tell you I did not have C++ in mind.” and “Java and C++ make you think that the new ideas are like the old ones. Java is the most distressing thing to happen to computing since MS-DOS.” ([proof](#))

“What’s Wrong With  
Object-Oriented Programming?”  
(2016)



Chapter #8:

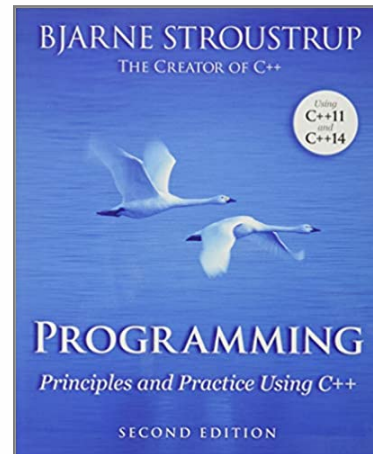
## Books, Venues, Call-to-Action



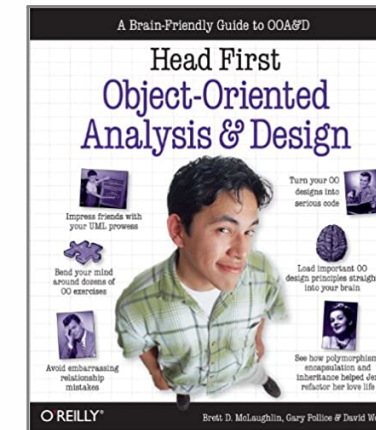
“Clean Code: A Handbook of Agile Software Craftsmanship” by ROBERT C. MARTIN



“Object-Oriented Analysis and Design with Applications” by GRADY BOOCH ET AL.



“Programming: Principles and Practice Using C++” by BJARNE STROUSTRUP



“Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D” by BRETT McLAUGHLIN ET AL.



## Where to publish:

SPLASH: ACM SIGPLAN conference on Systems, Programming, Languages, and Applications.

## Call to Action:

Analyze one of the apps you've written recently and find out which design principles you've used and where.

## Still unresolved issues:

- How to fix object-oriented programming?
- How to enforce it automatically?
- How to eliminate root causes of violations?
- How to make better programming languages?

# Bibliography