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

Before:

Structured Programming
C, Pascal, FORTRAN, ALGOL
CASE
Waterfall

After:

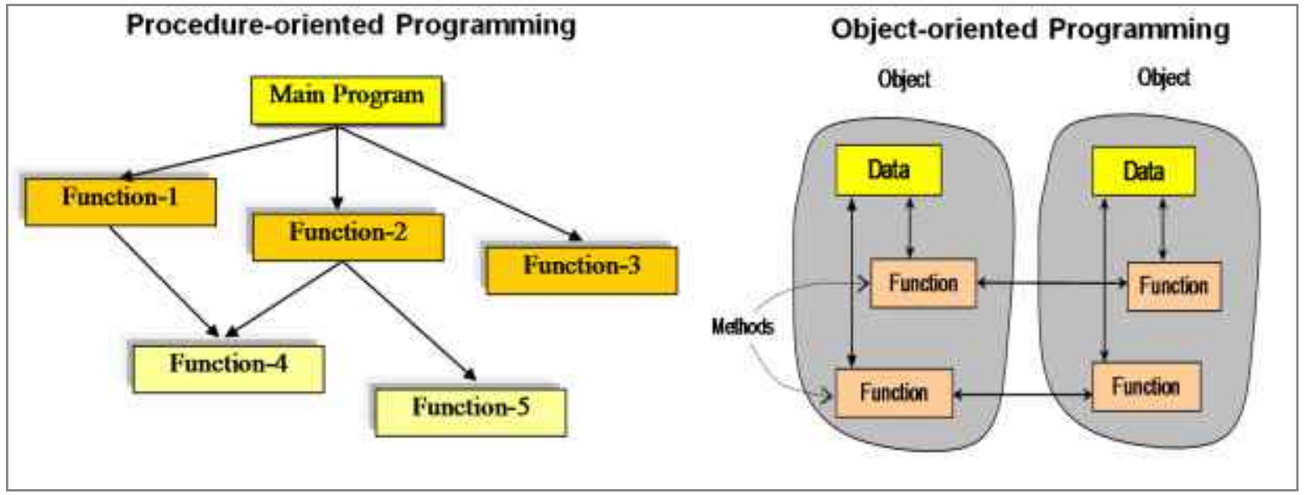
Object-Oriented Programming
C++, Object Pascal, Java
RUP

- 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

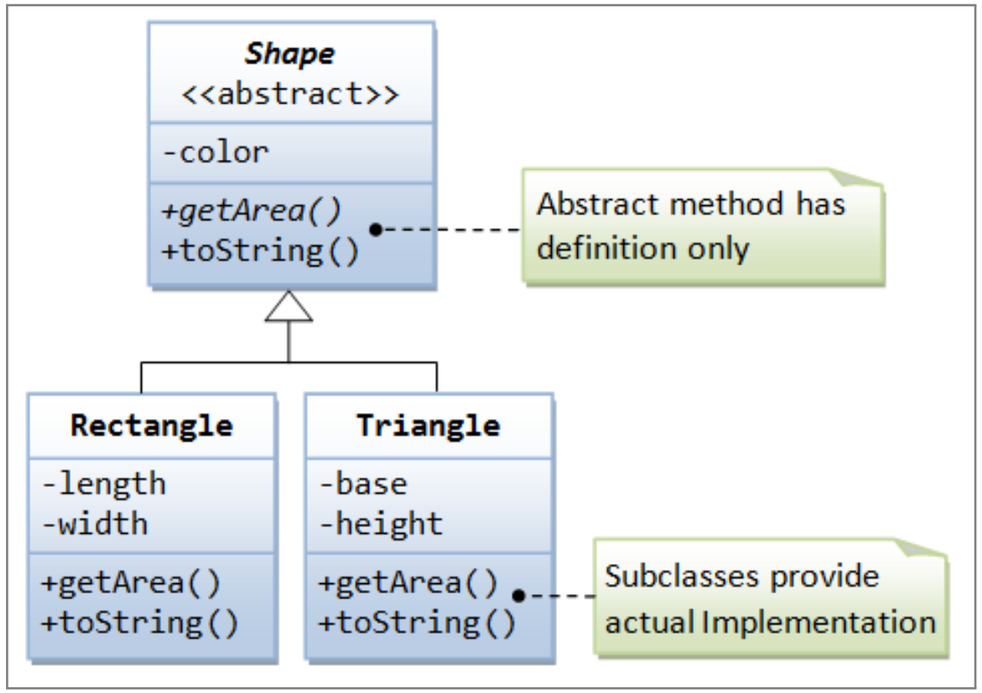


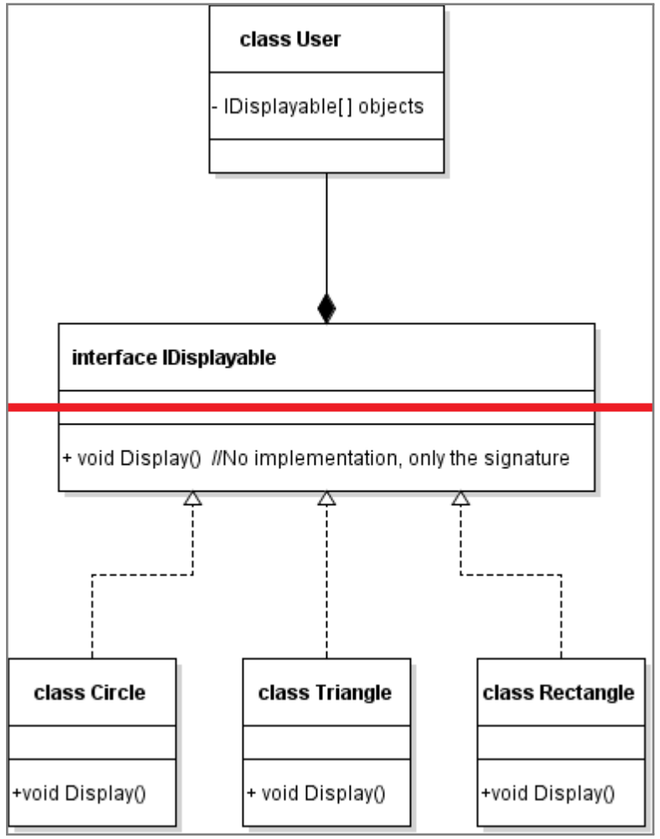
Some Principles of OOP

Abstraction
Encapsulation
Modularity
Hierarchy

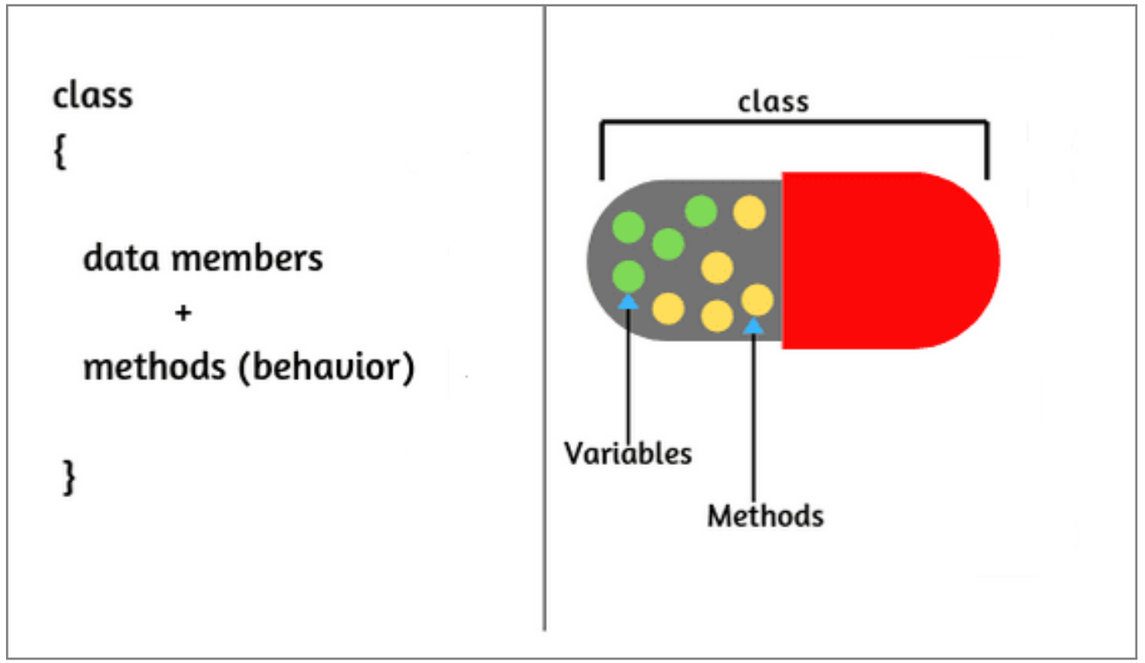
Polymorphism
Inheritance
Composition
Delegation
Subtyping
Data Hiding
Separation of Concerns

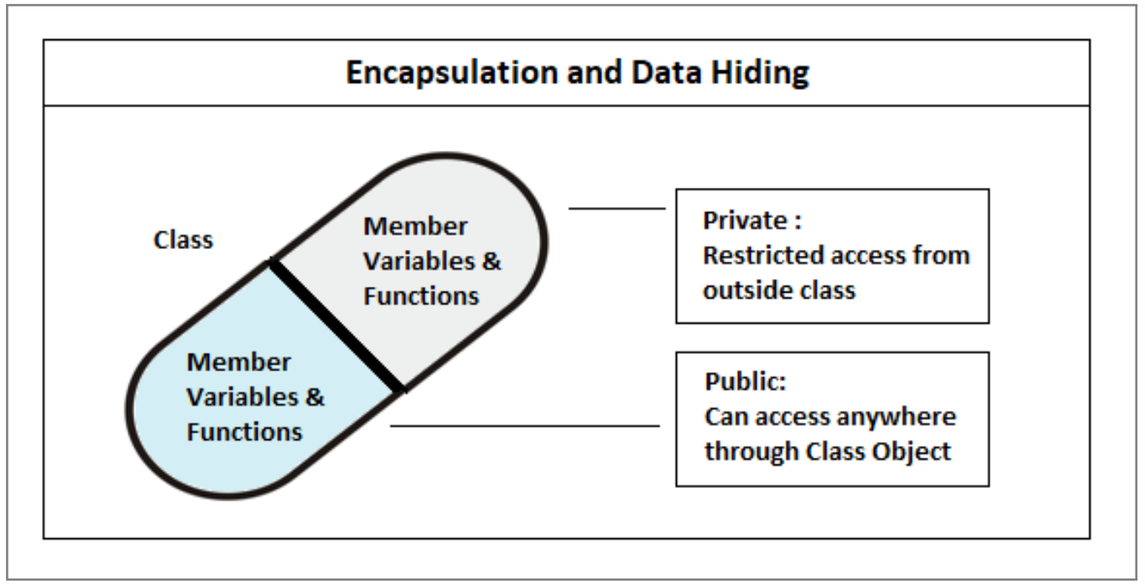
Abstraction & Polymorphism




class User
- IDisplayable[] objects
interface IDisplayable
+ void Display() //No implementation, only the signature
class Circle
+void Display()
class Triangle
+ void Display()
class Rectangle
+void Display()

Encapsulation



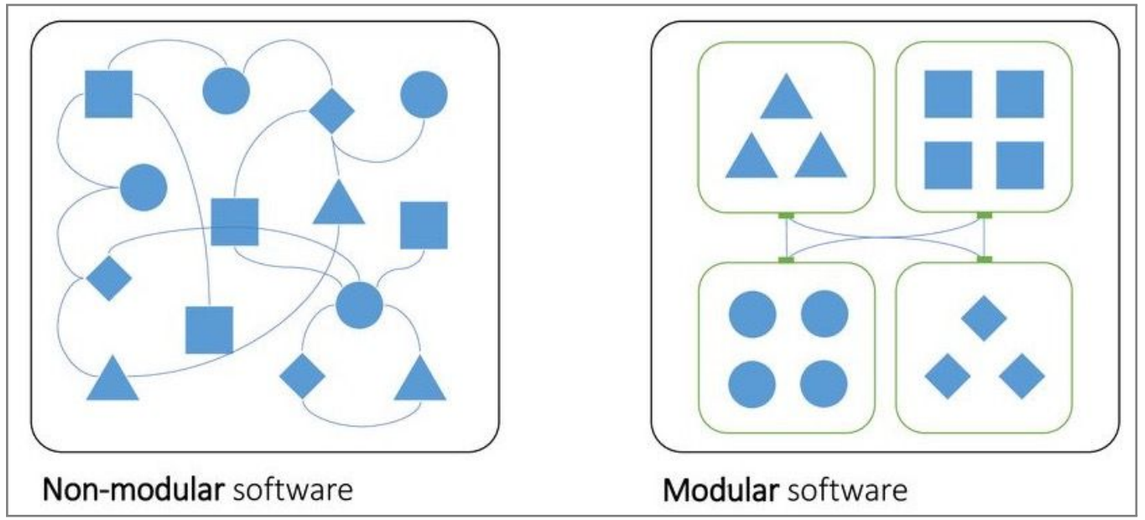

Encapsulation and Data Hiding
Class
Member Variables & Functions
Member Variables & Functions
Private : Restricted access from outside class
Public: Can access anywhere through Class Object

“Encapsulation Covers Up Naked Data” (2016)

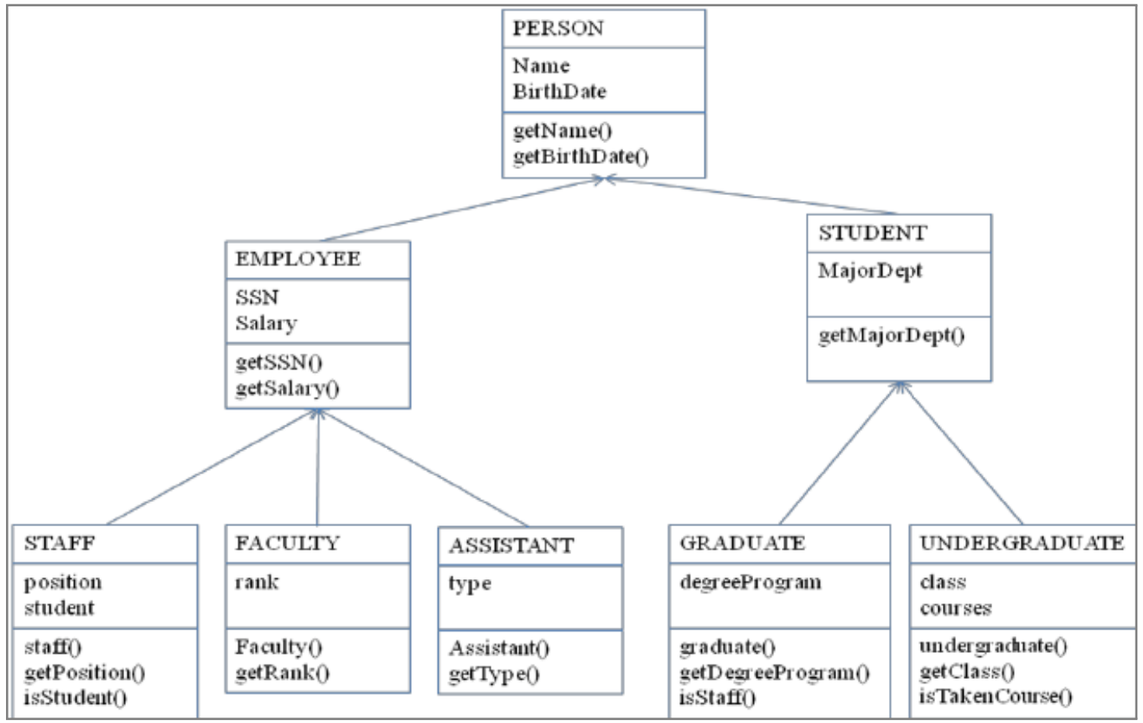
```
class Temperature {
  private int t;
  public int getT() { return this.t; }
  public void setT(int t) { this.t = t; }
}
```

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

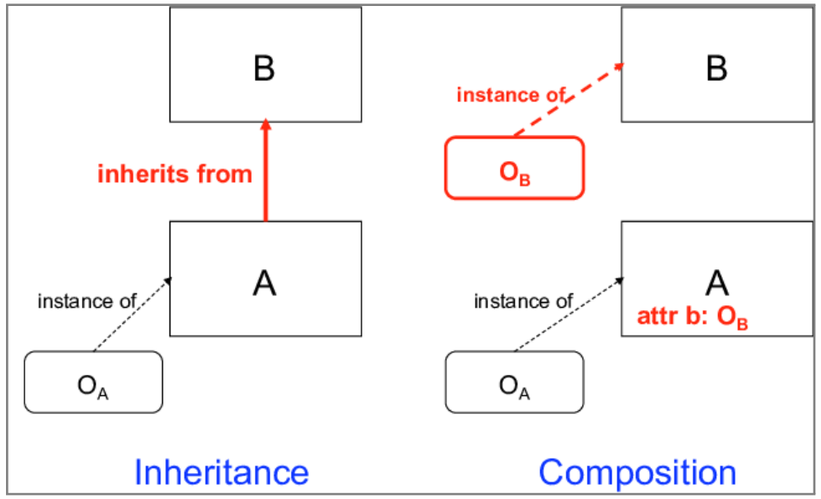
Modularity



Hierarchy


PERSON
Name
BirthDate
getName()
getBirthDate()
EMPLOYEE
SSN
Salary
getSSN()
getSalary()
STUDENT
MajorDept
getMajorDept()
STAFF
position
student
staff()
getPosition()
isStudent()
FACULTY
rank
Faculty()
getRank()
ASSISTANT
type
Assistant()
getType()
GRADUATE
degreeProgram
graduate()
getDegreeProgram()
isStaff()
UNDERGRADUATE
class
courses
undergraduate()
getClass()
isTakenCourse()

Inheritance vs. Composition

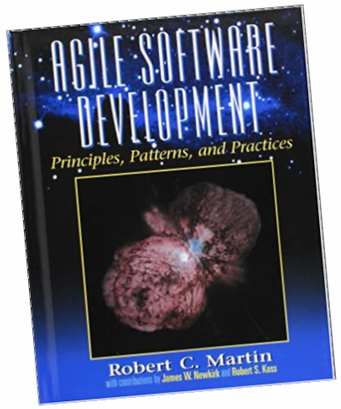


“Prefer composition over inheritance?” (2008) at StackOverflow

“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.”

— Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices.* Prentice Hall, 2002. doi:10.5555/515230

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

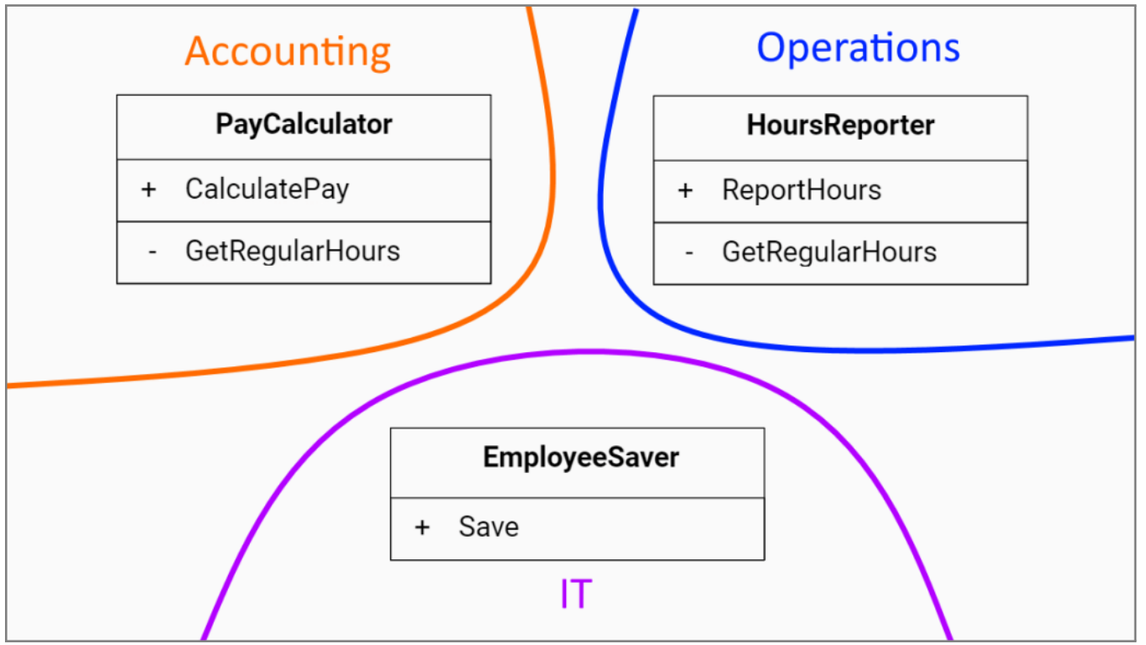
Single Responsibility Principle (SRP)

"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."

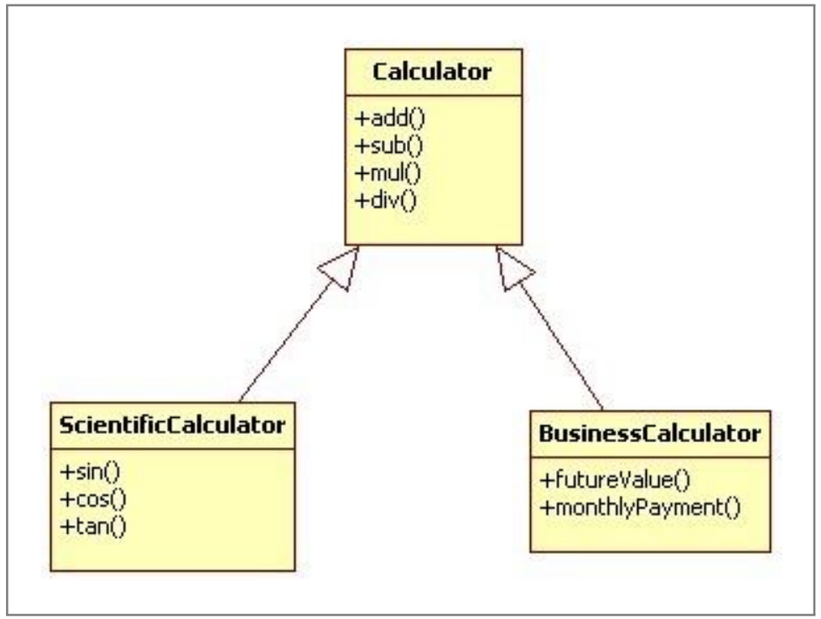
"Each module should be responsible for each role."

"SRP is a Hoax" (2017)


Accounting
PayCalculator
+ CalculatePay
- GetRegularHours
Operations
HoursReporter
+ ReportHours
- GetRegularHours
EmployeeSaver
+ Save
IT

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

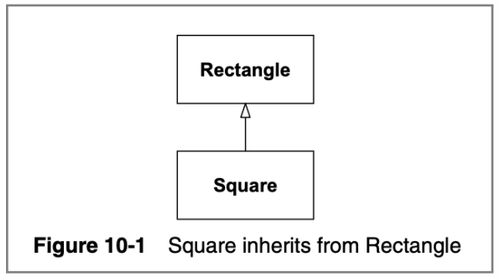
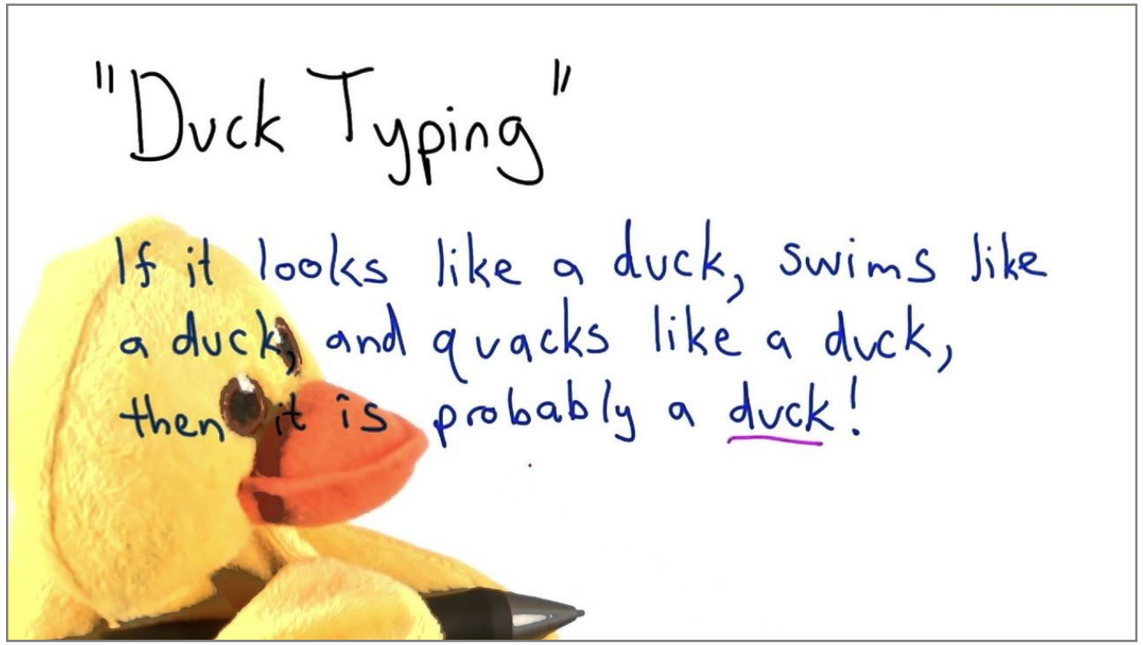


Figure 10-1 Square inherits from Rectangle

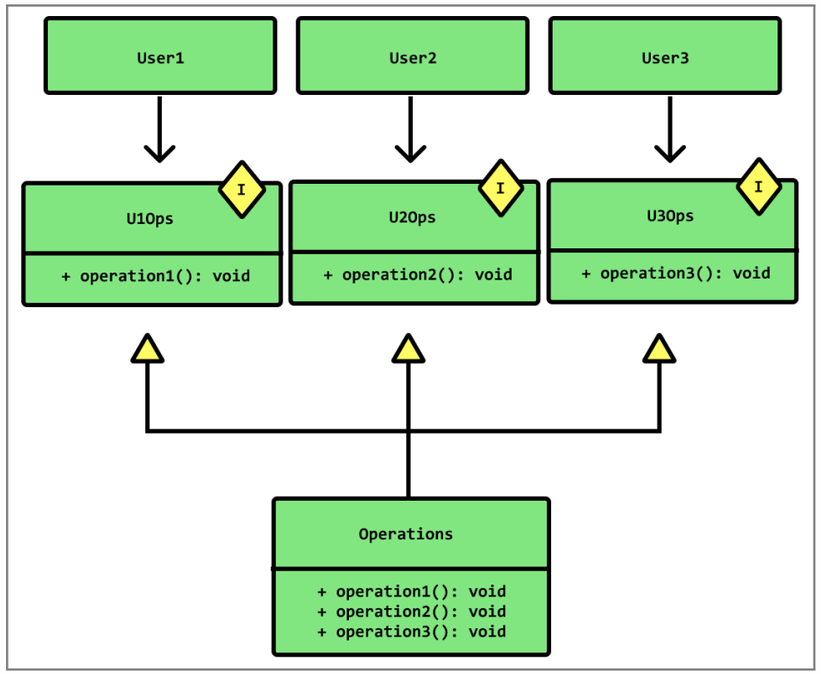
"The LSP makes it clear that in OOD, the IS-A relationship pertains to behavior that can be reasonably assumed and that clients depend on."

Duck Typing


"Duck Typing"
If it looks like a duck, swims like a duck, and quacks like a duck, then it is probably a duck!

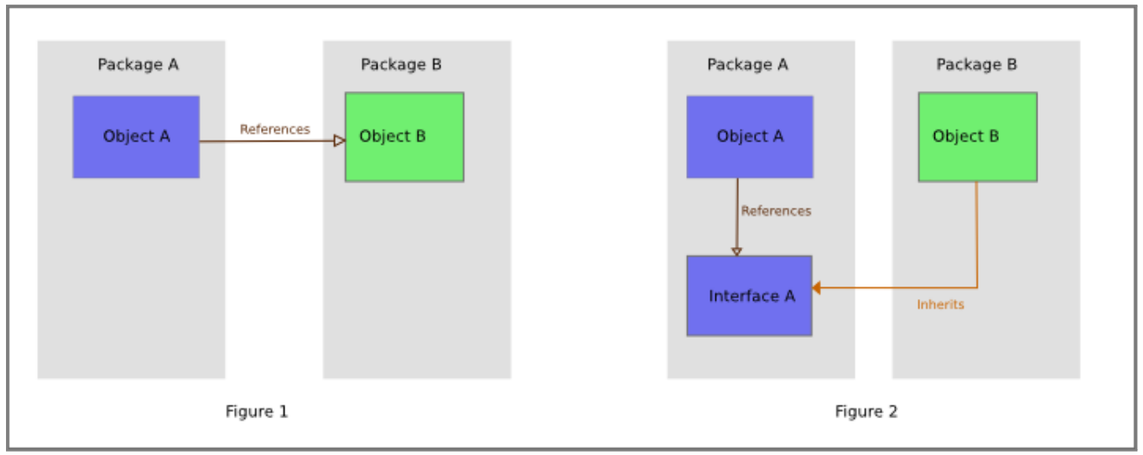
Interface Segregation Principle (ISP)

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



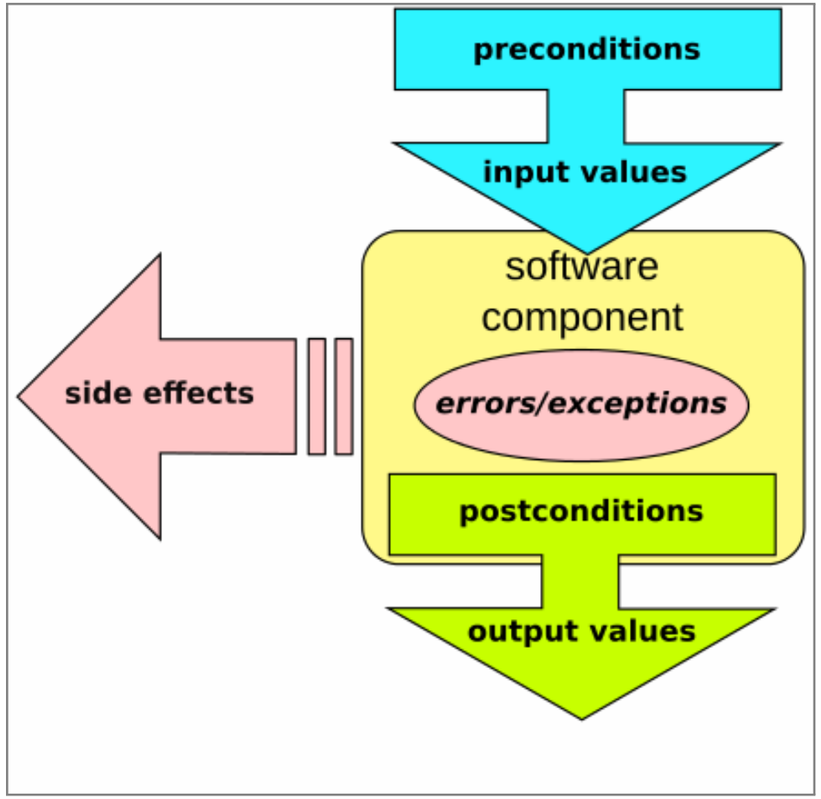
Dependency Inversion Principle (DIP)

“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



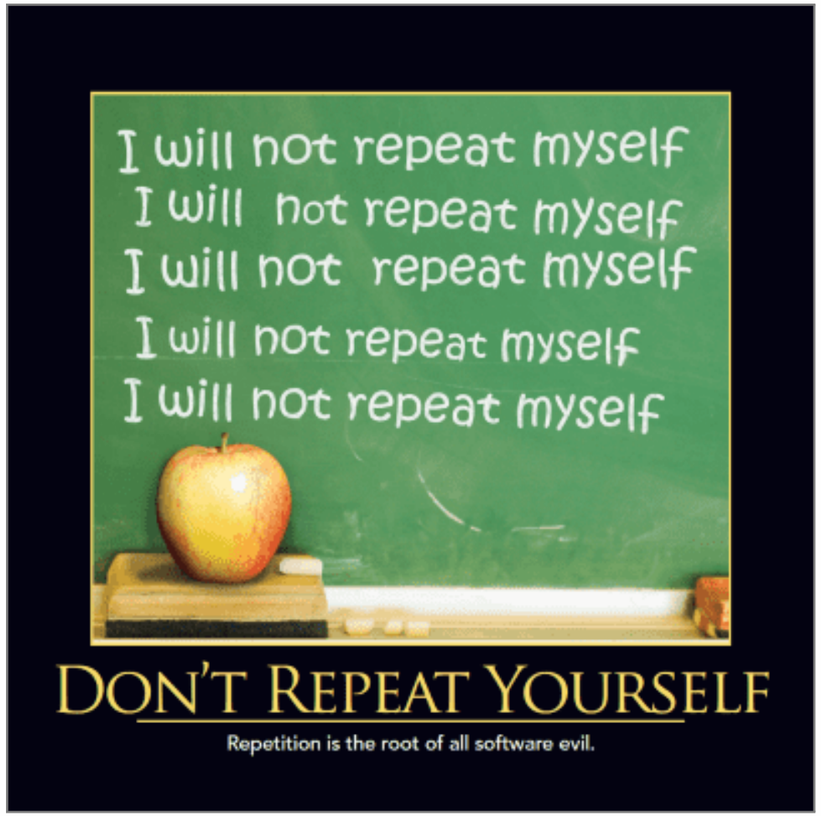
Chapter #3:

Design by Contract by Bertrand Meyer


preconditions
input values
software component
errors/exceptions
side effects
postconditions
output values

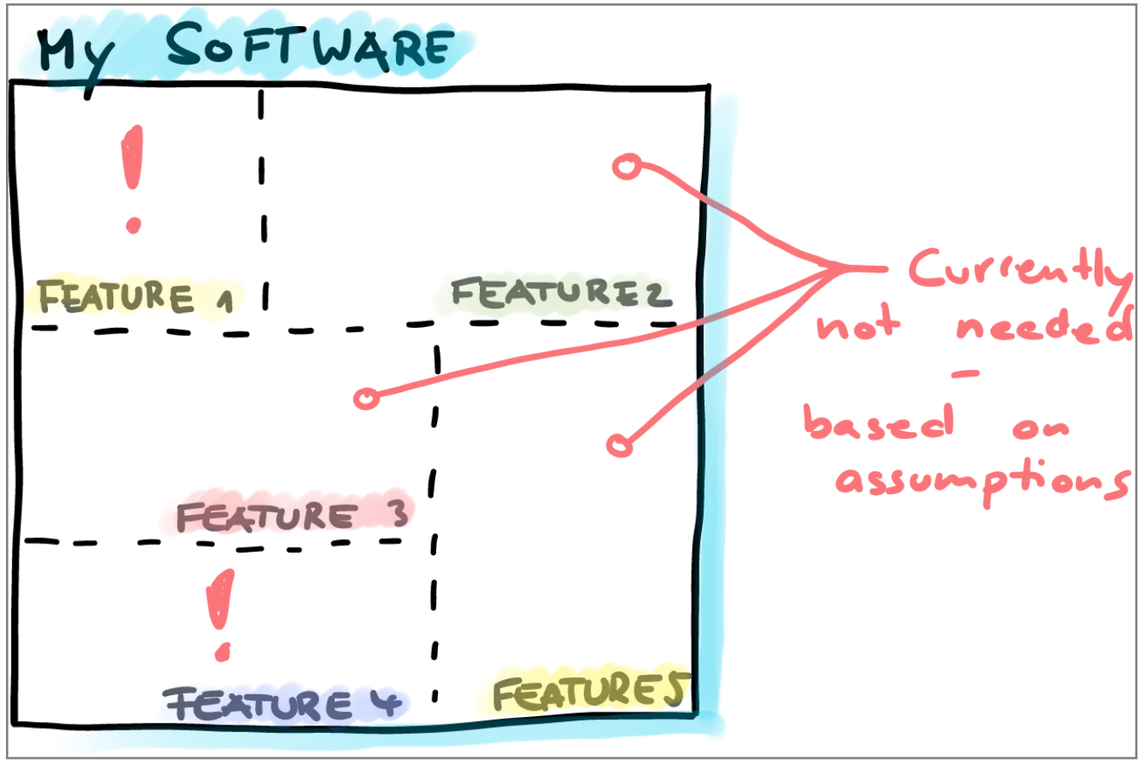
Chapter #4:

Don’t Repeat Yourself (DRY)


I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
I will not repeat myself
DON'T REPEAT YOURSELF
Repetition is the root of all software evil.

Chapter #5:

You Ain't Gonna Need It (YAGNI)


My SOFTWARE
FEATURE 1
FEATURE 2
FEATURE 3
FEATURE 4
FEATURE 5
Currently not needed - based on assumptions

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	FP pattern/principle
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions □

(c) Uncle Bob

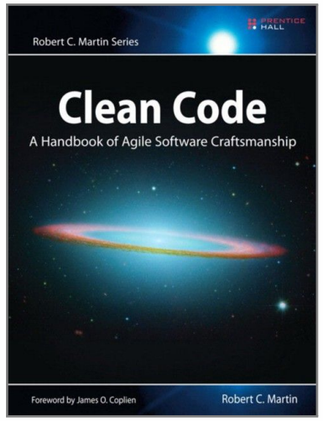
Edsger W. Dijkstra (1989)
"TUG LINES," Issue 32, August 1989
"Object oriented programs are offered as alternatives to correct ones" and "Object-oriented programming is an exceptionally bad idea which could only have originated in California."

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)

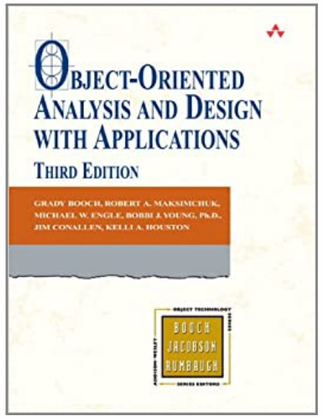
"What's Wrong With Object-Oriented Programming?" (2016)

Chapter #8:

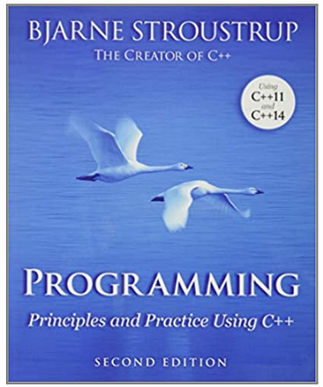
Books, Venues, Call-to-Action



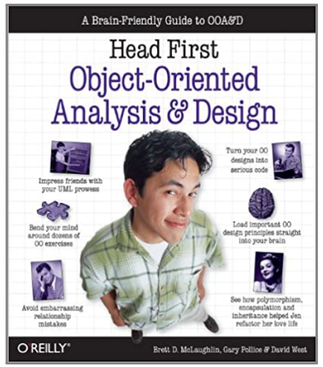
Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008. doi:10.5555/1388398



Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 1994. doi:10.5555/1407387



Bjarne Stroustrup. *Programming: Principles and Practice Using C++*. Pearson Education, 2 edition, 2014



Brett McLaughlin, Gary Pollice, and David West. *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*. O'Reilly Media, 2006

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

Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen, and Kelli A. Houston. *Object-Oriented Analysis and Design With Applications*. Addison-Wesley, 1994. doi:10.5555/1407387.

Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002. doi:10.5555/515230.

Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008. doi:10.5555/1388398.

Brett McLaughlin, Gary Pollice, and David West. *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*. O'Reilly Media, 2006.

Bjarne Stroustrup. *Programming: Principles and Practice Using C++*. Pearson Education, 2 edition, 2014.