



# SOLID

O. DYSHLEVYI



# S.O.L.I.D. PRINCIPLES

S.O.L.I.D. is an acronym introduced by **Robert C. Martin**

5 basic patterns of object-oriented programming and design:

S

- Single responsibility principle

O

- Open/Closed principle

L

- Liskov substitution principle

I

- Interface segregation principle

D

- Dependency inversion principle



ONE CHEF CAN'T RUN THE WHOLE RESTAURANT  
SINGLE RESPONSIBILITY PRINCIPLE

# SINGLE RESPONSIBILITY PRINCIPLE

*"There should never be more than one reason for a class to change."*

— **Robert Martin**



A class should concentrate on doing one thing.

The class seems to be doing too much, is too big and too complicated. The easiest way to fix this is to split the class

## SRP. PROS & CONS



### Pros:

- Helps to follow DRY
- Lowers chances to change single class
- Class names correspond to what they do



### Cons:

- May complicate system by adding too many new classes

# TYPICAL SRP VIOLATIONS

Mixing business logic  
with infrastructure

Low cohesion  
(Class/module/method  
solves several  
unrelated tasks)

Verbose logging or  
formatting within a  
class



# SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should



TRYING NEW SHOES DOESN'T REQUIRE YOU TO SAW YOUR FEET OFF  
OPEN CLOSED PRINCIPLE



# OPEN/CLOSED PRINCIPLE

*"Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."*

— **Robert Martin**



Design patterns that help following the principle:

- Template Method
- Strategy
- Decorator



## Pros:

- Adding new functionality without legacy code being changed

## Cons:

- May complicate system because of amount of classes being grown

## TYPICAL OCP VIOLATIONS

Class with a switch statement or if-else condition with some business logic behind it

Unstable class interface used in a lot of places in code



# OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat



# LISKOV SUBSTITUTION PRINCIPLE

*"Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."*

— **Robert Martin**



Subclasses should behave nicely when used in place of their base class



### Pros:

- Unambiguously defined contract all implementers should follow
- Allows to interchange implementation correctly, without side effects

### Cons:

- Sometimes it takes too much effort to define right class hierarchy

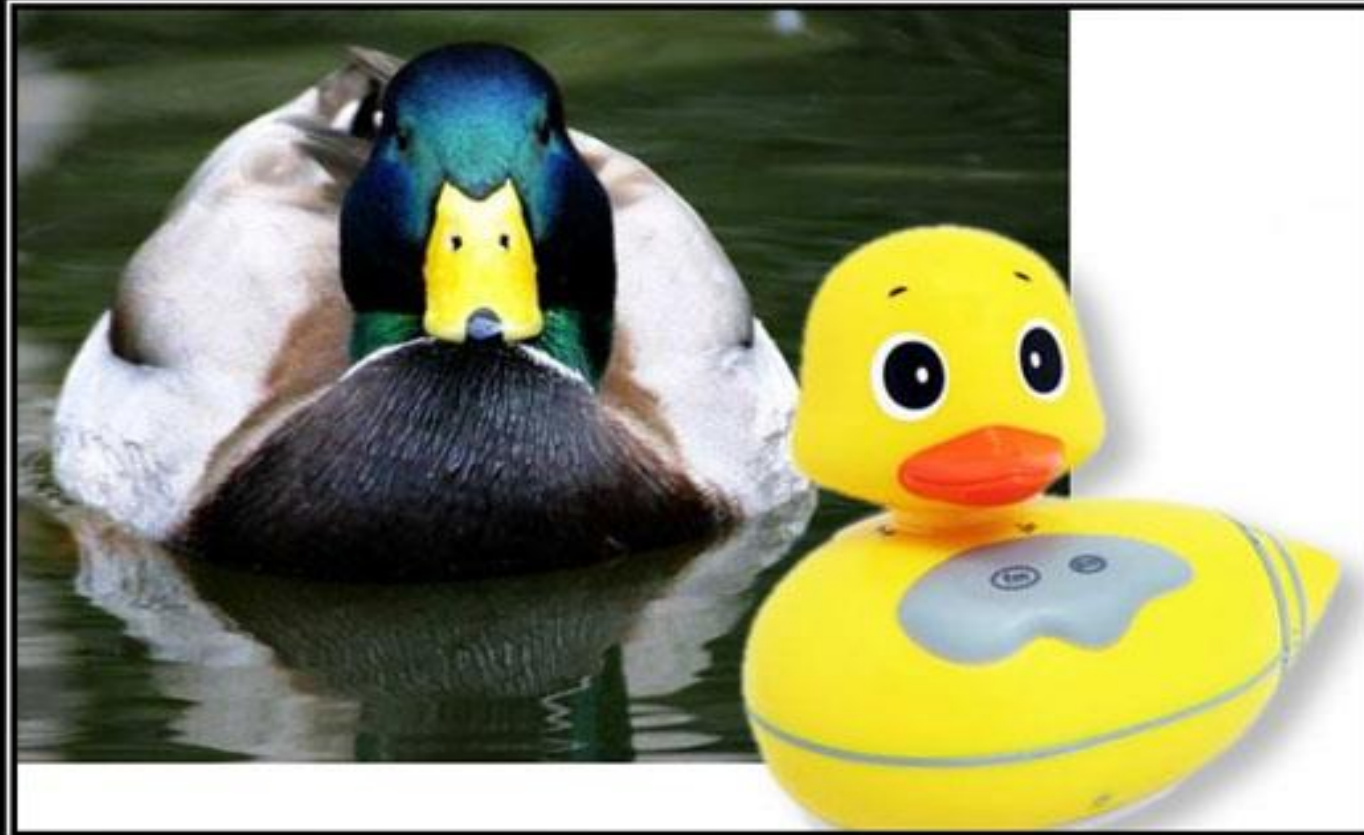
## LSP. PROS & CONS

Behaviour of derived classes is not in consistency with behavior of base class (e.g. generation of exceptions not defined by base class)

Blurred contract of base class impedes implementation of consistent behavior by derived classes

TYPICAL LSP VIOLATIONS





# LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You  
Probably Have The Wrong Abstraction



YOU WANT ME TO PLUG THIS IN WHERE?

INTERFACE SEGREGATION

# INTERFACE SEGREGATION PRINCIPLE

*"Clients should not be forced to depend upon interfaces that they don't use."*

— **Robert Martin**



*"Many client specific interfaces are better than one general purpose interface"*



### Pros:

- No need to implement unnecessary methods
- Client code sees only what it should see



### Cons:

- Adding additional interfaces

## ISP. PROS & CONS

---

Method has  
parameters of derived  
class, even though it's  
enough to use base  
class

Class has 2 or more  
different types of  
clients

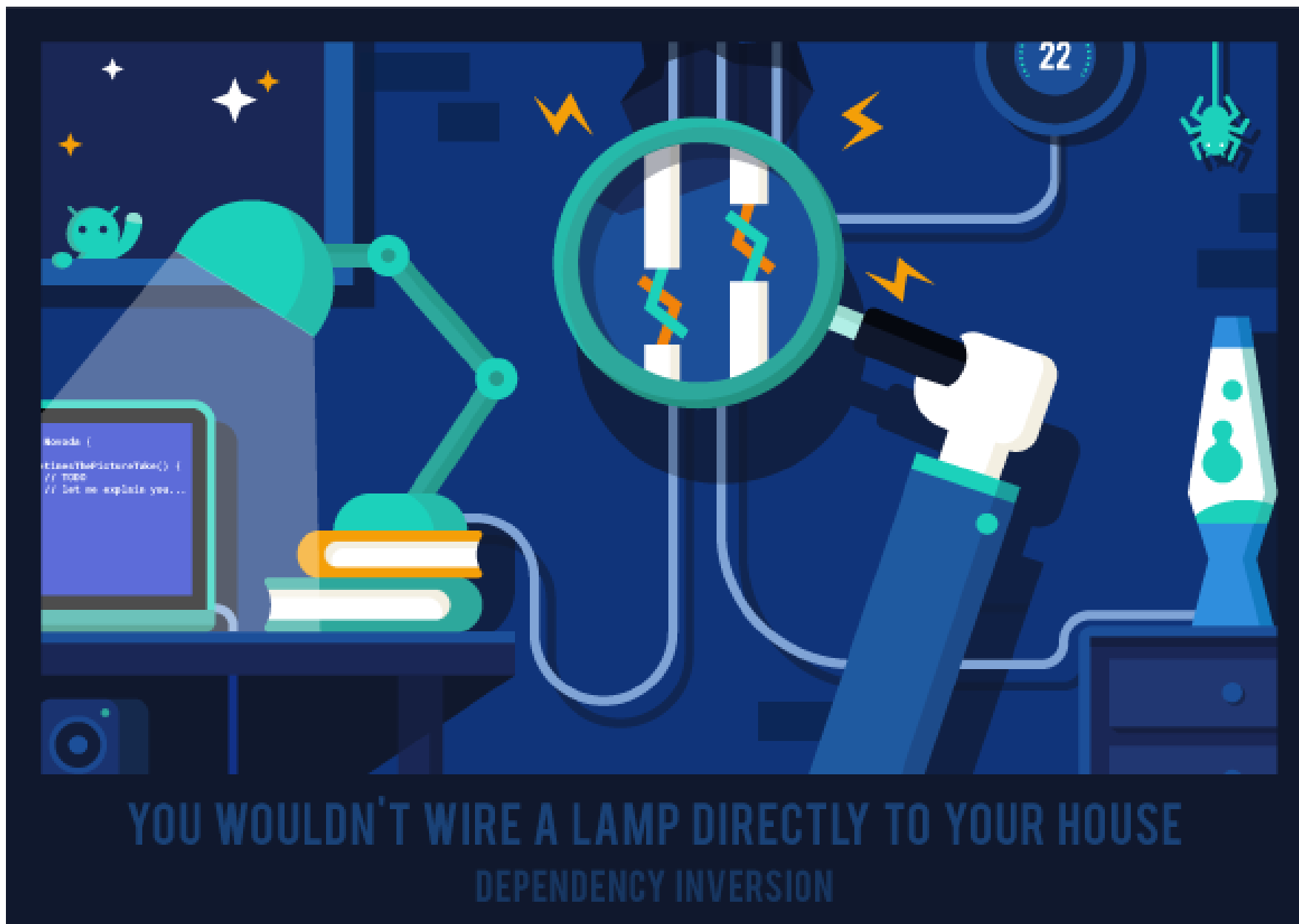
Unimplemented  
methods inside of class

## TYPICAL ISP VIOLATIONS



# INTERFACE SEGREGATION PRINCIPLE

You Want Me To Plug This In, Where?



YOU WOULDN'T WIRE A LAMP DIRECTLY TO YOUR HOUSE

DEPENDENCY INVERSION

# DEPENDENCY INVERSION PRINCIPLE

*If a class has dependencies on other classes, it should rely on the dependencies' interfaces rather than their concrete types”*

— **Robert Martin**



Dependency injection is the method of following this principle.






### Pros:

- Classes don't depend on concrete implementation
- Allows easily change implementation
- Allows write good unit tests

### Cons:

- Creating additional interfaces
- Constructor madness

DI. PROS & CONS



Low-level classes directly  
communicate with high-  
level classes (e.g. models  
know about user interface  
or infrastructure code  
knows about business logic)

## TYPICAL DIP VIOLATIONS



## DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

