

# Module 1: “The SOLID Principles”



TEKNOLOGISK  
INSTITUT

# SOLID



Beautiful, cartoonish illustrations by Ugonna Thelma from "[THE S.O.L.I.D. PRINCIPLES IN PICTURES](#)"

# Agenda

- ▶ Introducing SOLID
- ▶ Single Responsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ *(Optional) Lab 1*
- ▶ Workshop A.1

# SOLID is...

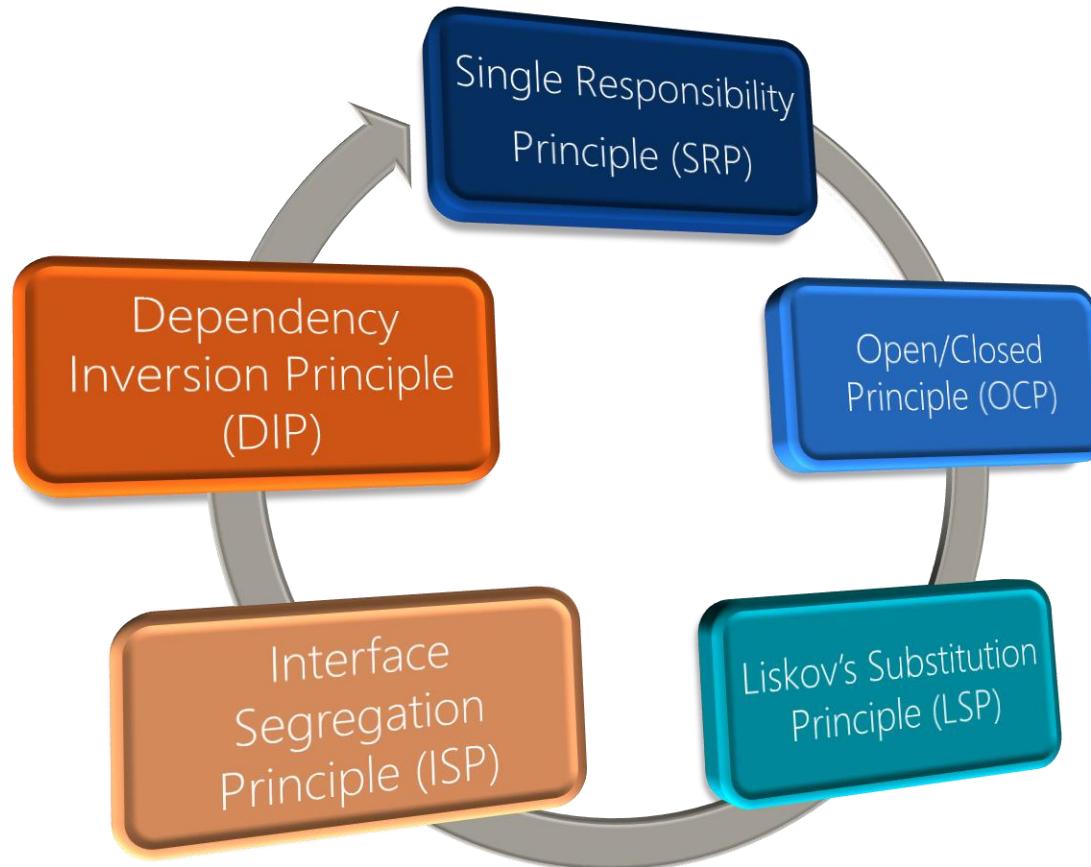
- ▶ ... Five fundamental "commandments" for OOP
- ▶ ... Coined by Robert C. Martin a.k.a. "Uncle Bob"
- ▶ ... Programming language-agnostic
- ▶ ... Not a framework or package!
  
- ▶ ... Maintainability!

*"Always code as if the  
guy who ends up  
maintaining your code  
will be a violent  
psychopath who  
knows where you live"*

- John F. Woods (1991)



# The Five Principles of SOLID



# Agenda

- ▶ Introducing SOLID
- ▶ **Single Reponsibility Principle (SRP)**
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ *(Optional) Lab 1*
- ▶ Workshop A.1

# Single Responsibility Principle (SRP)

*Each class should only have a single responsibility.*

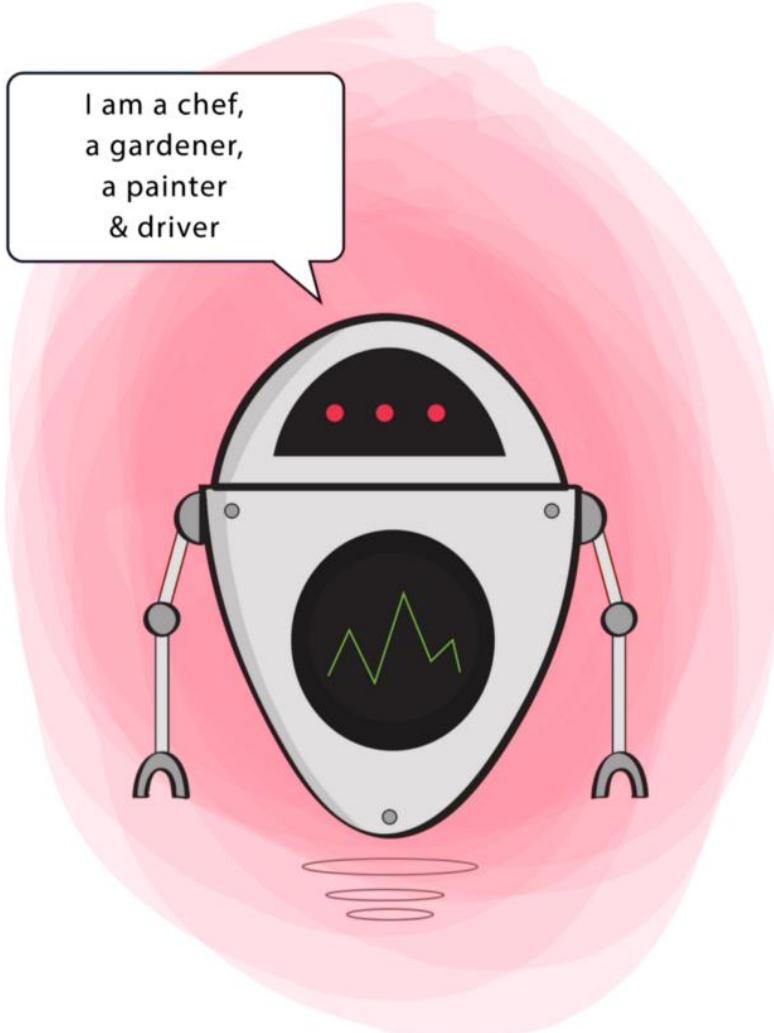
*Each class should have only one reason to change*

# What Does That Mean Exactly?

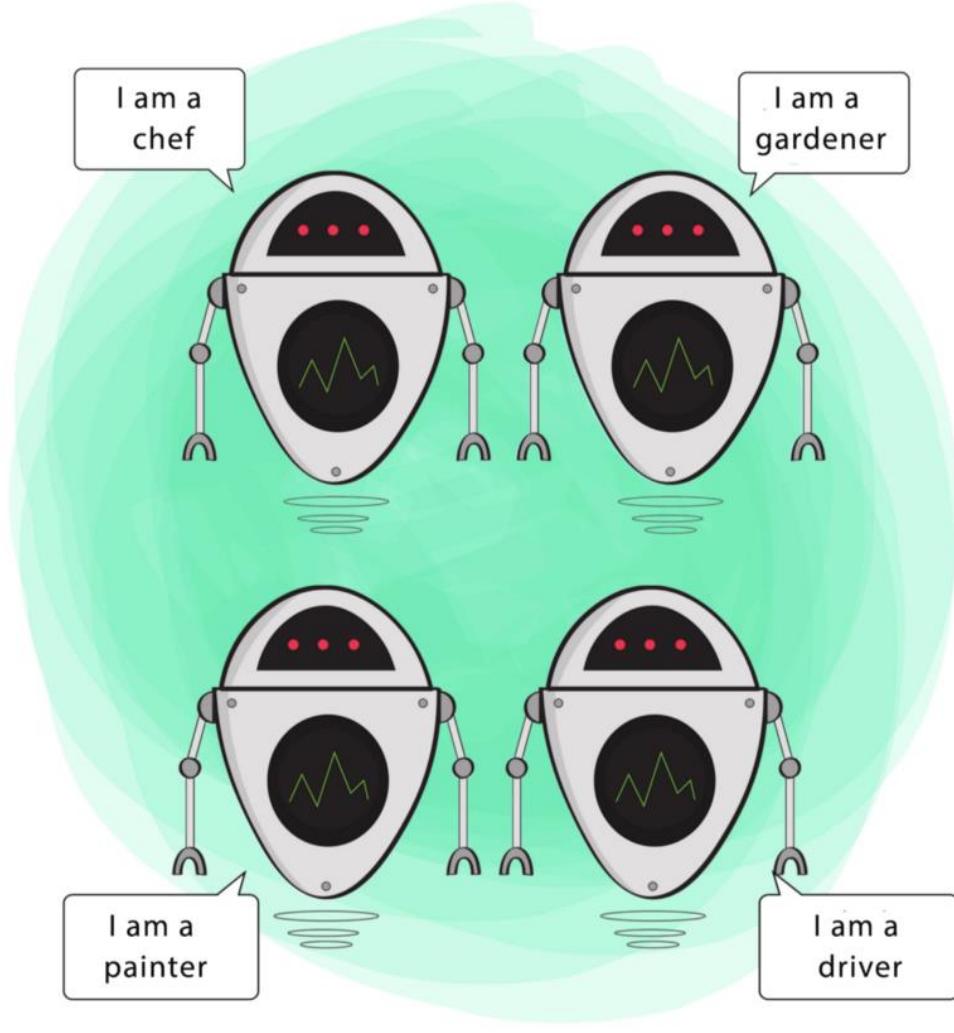
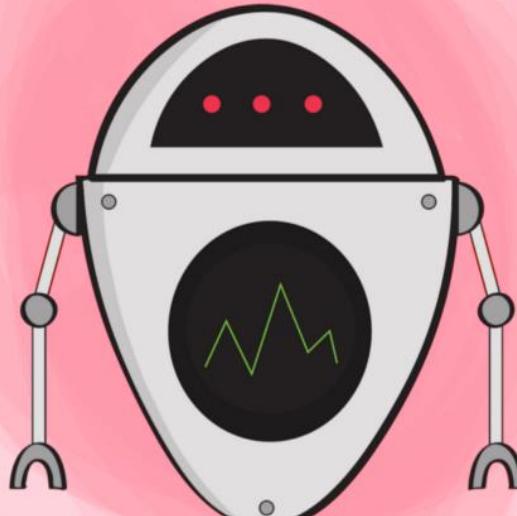
*For each class there should be only one requirement which, when changed, incurs a change to that class*

# SRP in Summary

- ▶ Idea
  - Avoid God classes and Swiss army knife classes
- ▶ Why?
  - Small classes are easy to understand, modify, and debug
  - Small classes are hard to get wrong ☺
  - Supports team collaboration
- ▶ Consequences
  - 4-5 times more classes – but small, simple classes!
  - Functionality will appear as classes



I am a chef,  
a gardener,  
a painter  
& driver



I am a  
chef

I am a  
gardener

I am a  
painter

I am a  
driver



Single Responsibility



# Agenda

- ▶ Introducing SOLID
- ▶ Single Responsibility Principle (SRP)
- ▶ **Open/Closed Principle (OCP)**
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ *(Optional) Lab 1*
- ▶ Workshop A.1

# Open/Closed Principle (OCP)

*Software entities should be open for extension, but closed for modification*

# What Does That Mean Exactly?

*When a class is done, it is done!*

*You add new functionality.*

*You derive from existing functionality.*

*You plug in new functionality into existing.*

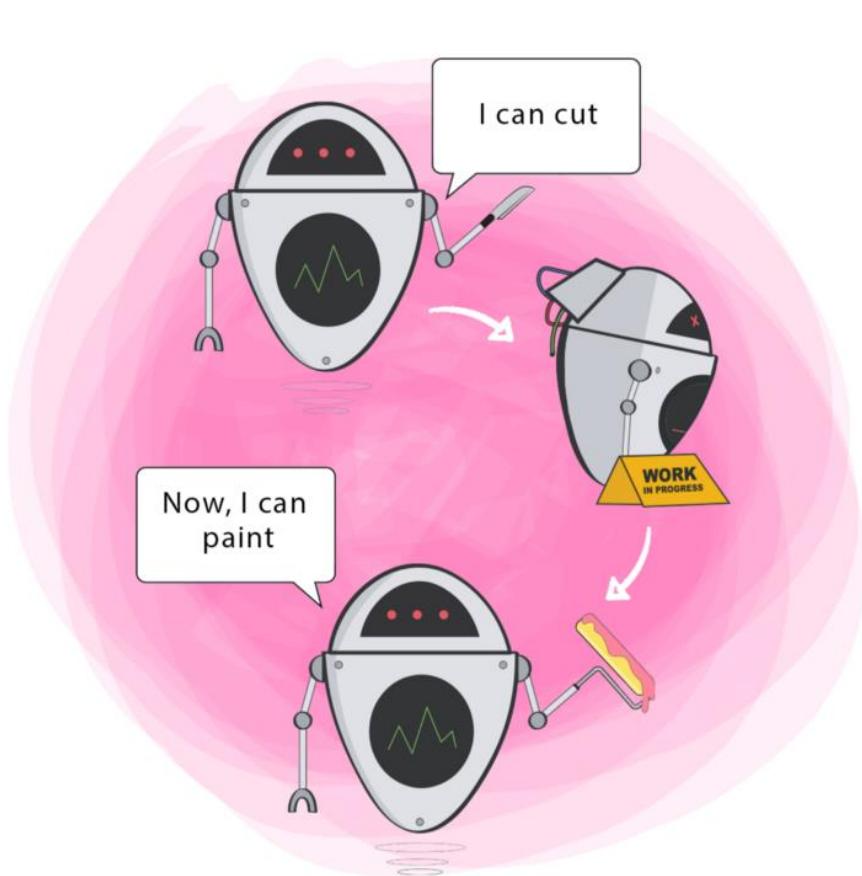
*There should be no cascading  
modifications throughout classes!*

# Abstract Base Classes or Interfaces?

- ▶ Bertrand Meyer's original definition
  - Based on (abstract) classes and inheritance
  - Can lead to quirky and multiple levels of inheritance
  - Puts large responsibility on author of base class
- ▶ The modern interpretation (a.k.a. "Polymorphic")
  - Interfaces
  - Allows swapping out complete implementations to avoid quirkiness

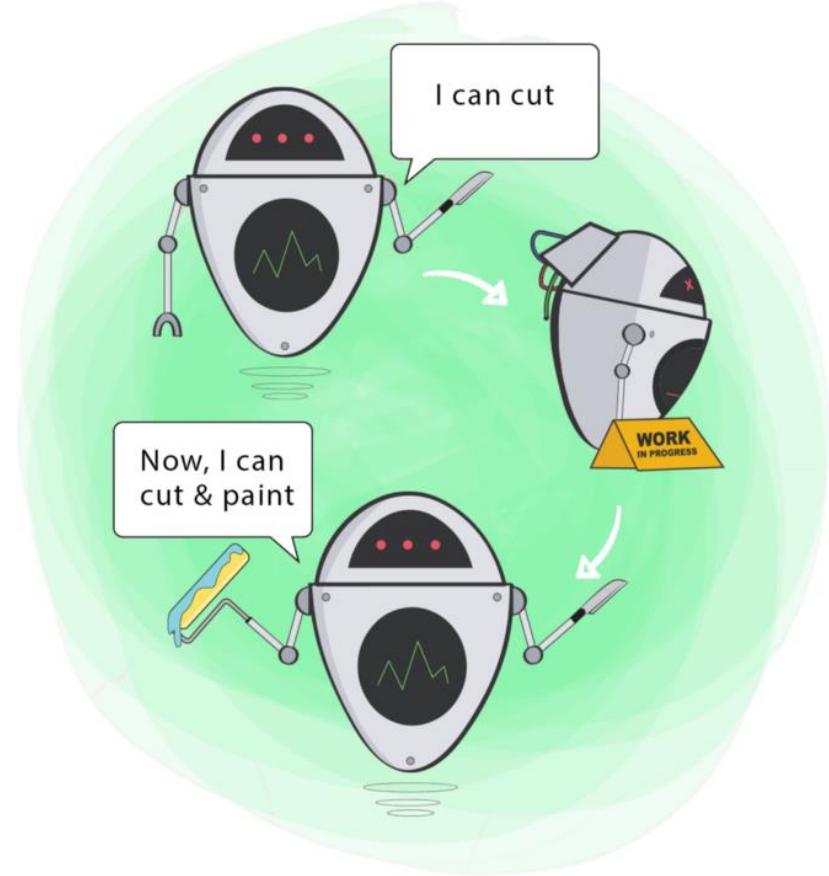
# OCP in Summary

- ▶ Idea
  - Add, derive or plug-in new functionality without changing existing classes
- ▶ Why?
  - Everything that worked before still works!
  - No accidental errors to existing code
  - Easier to locate newly introduced errors
  - Supports team collaboration
- ▶ Consequences
  - Changes are easy to locate and review
  - Existing tests still work when new requirements are added



✗

Open-Closed



✓

# Agenda

- ▶ Introducing SOLID
- ▶ Single Responsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ **Liskov's Substitution Principle (LSP)**
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ *(Optional) Lab 1*
- ▶ Workshop A.1

# Liskov Substitution Principle (LSP)

*If  $S$  is a subtype of  $T$ , then objects of type  $T$  may be replaced with objects of type  $S$  without breaking the program*

# What Does That Mean Exactly?

*Any derived class should be substitutable for its base class.*

*When you add new functionality, don't make any changes which cause existing code to break.*

*Essentially: "Behave well!"*

# LSP Variance Rules ~ Signature

- ▶ Contravariance of the method arguments in a subtype
- ▶ Covariance of the method return type in a subtype
- ▶ No new exceptions can be thrown by the subtype (unless they are part of the existing exception hierarchy)

# LSP Contract Rules ~ Behavior

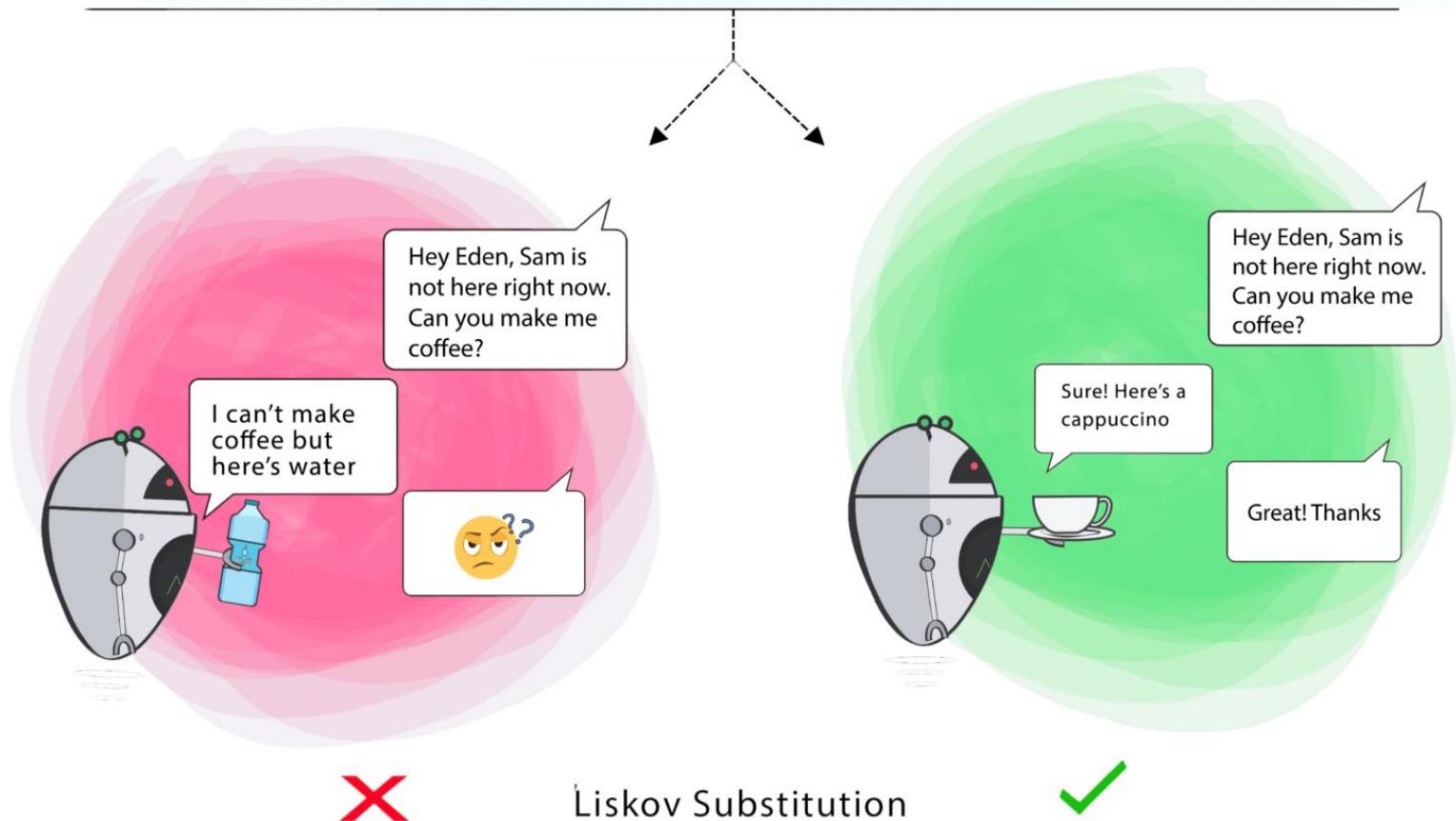
- ▶ Preconditions cannot be strengthened in a subtype
- ▶ Postconditions cannot be weakened
- ▶ Invariants of the base type must be preserved in a subtype
- ▶ History constraint: Mutability vs. Immutability

# But...

- ▶ What about (pure) abstract base classes?
- ▶ What about interfaces?
- ▶ They have no existing implementation!

# LSP in Summary

- ▶ Idea
  - Make sure your subtypes behave well within the existing program
- ▶ Why?
  - Due to SRP and OCP you will swap functionality all the time.
  - Swapping in new functionality should not break your program
  - Essentially, LSP is a vital enabler for the other SOLID rules
- ▶ Consequences
  - Must implement all methods and properties in subclasses in the "spirit" of the existing program
  - Understand (and respect) the data invariants of the base classes
  - Nothing breaks...! ☺



# Agenda

- ▶ Introducing SOLID
- ▶ Single Reponsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ **Interface Segregation Principle (ISP)**
- ▶ Dependency Inversion Principle (DIP)
- ▶ *(Optional) Lab 1*
- ▶ Workshop A.1

# Interface Segregation Principle (ISP)

*A client should not be forced to depend upon methods it doesn't use*

# What Does That Mean Exactly?

*Break interfaces into smaller, more focused interfaces.*

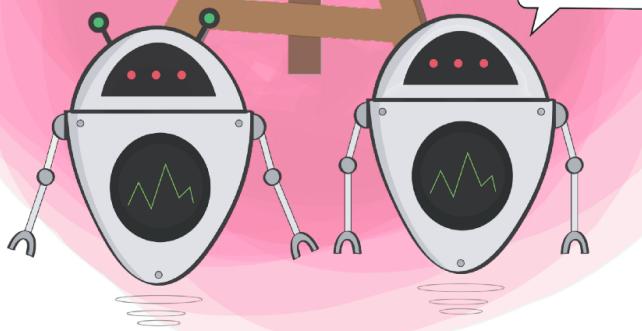
*(Can still combine smaller interfaces using interface inheritance, though)*

# ISP in Summary

- ▶ Idea
  - Make your interfaces small and focused
    - This includes method parameters as well!
- ▶ Why?
  - Bloated interfaces probably violate SRP (or LSP)
  - Prevents references to unused dependencies
- ▶ Consequences
  - Interfaces become easier to implement
  - Classes and components have fewer dependencies

EXERCISES		
All Robots		
- spin around		
- rotate arms		
- wiggle antennas		

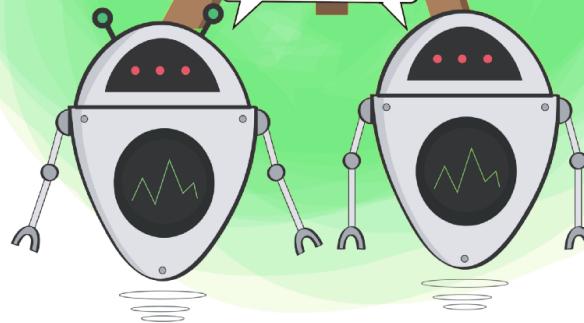
Oops! But I don't have antennas



Interface Segregation

EXERCISES		
Robots that can spin around	Robots that can rotate arms	Robots that can wiggle antennas
- spin around	- rotate arms	- wiggle antennas

Awesome!



# Agenda

- ▶ Introducing SOLID
- ▶ Single Reponsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ **Dependency Inversion Principle (DIP)**
- ▶ *(Optional) Lab 1*
- ▶ Workshop A.1

# Dependency Inversion Principle (DIP)

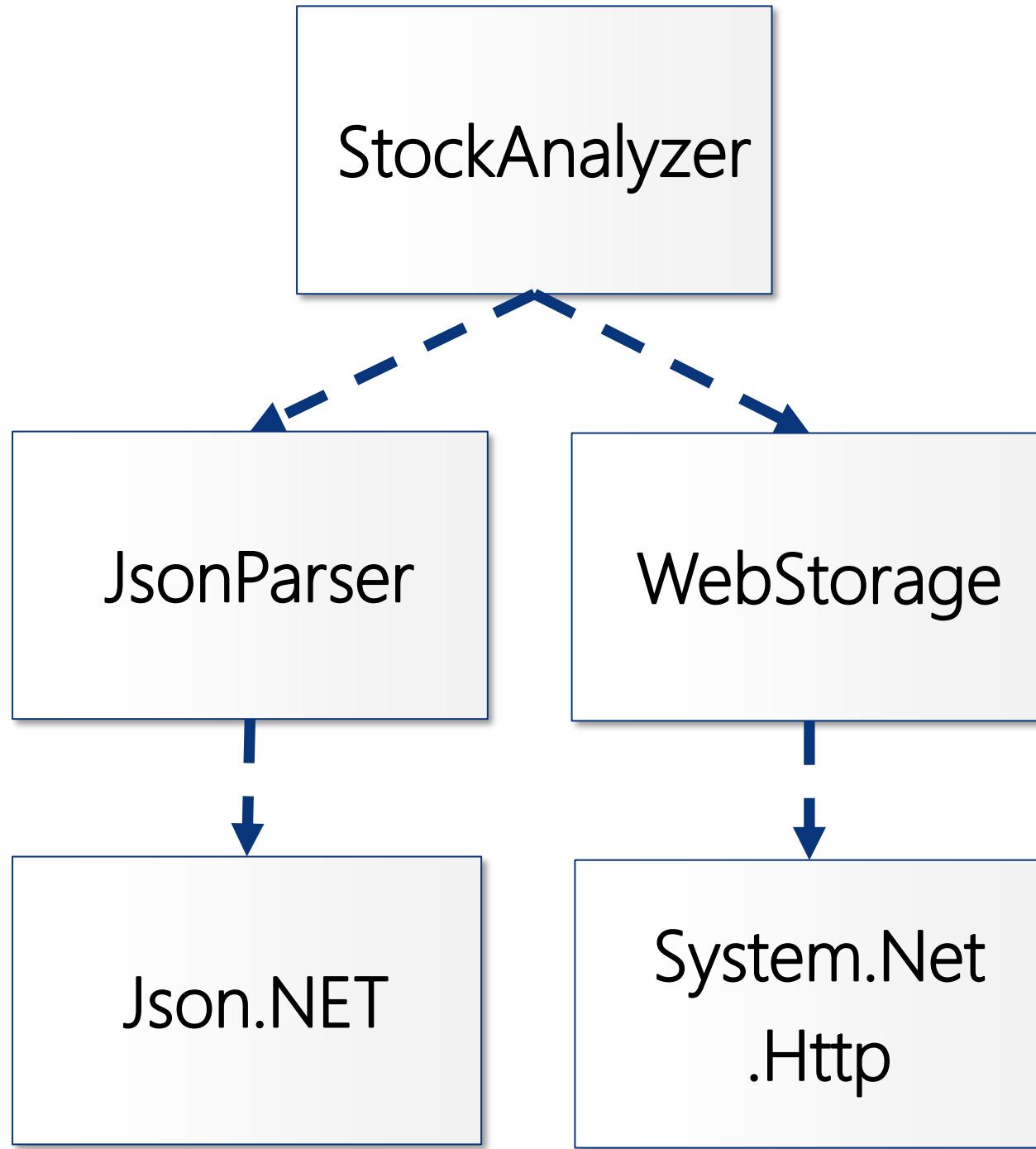
*High-level modules should not depend on low-level modules. Both should depend on abstractions.*

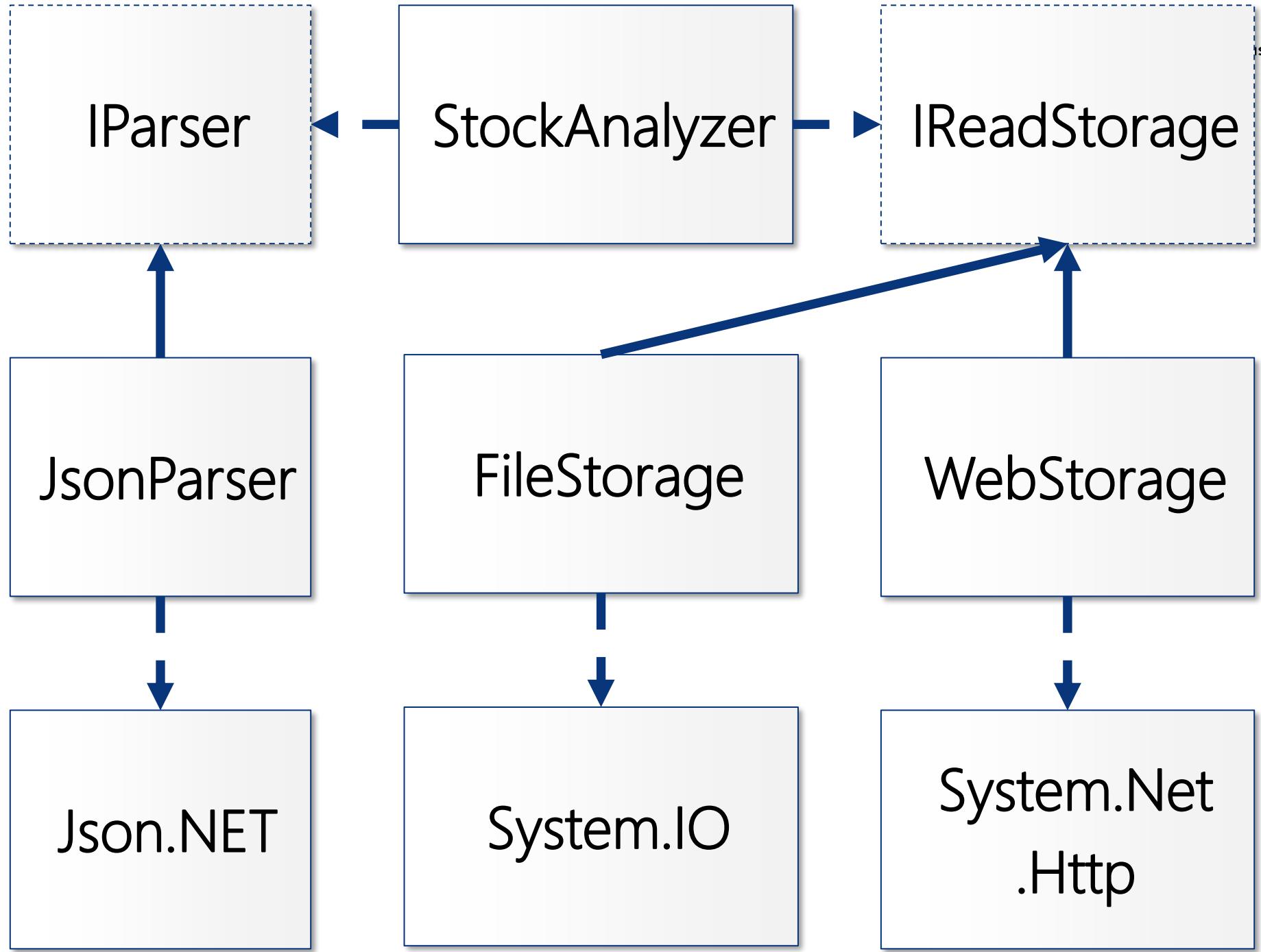
*Abstractions should not depend upon details. Details should depend upon abstractions.*

# What Does That Mean Exactly?

*Ensure that your classes do not depend upon specific implementations. That way you have the freedom to swap implementations and behavior.*

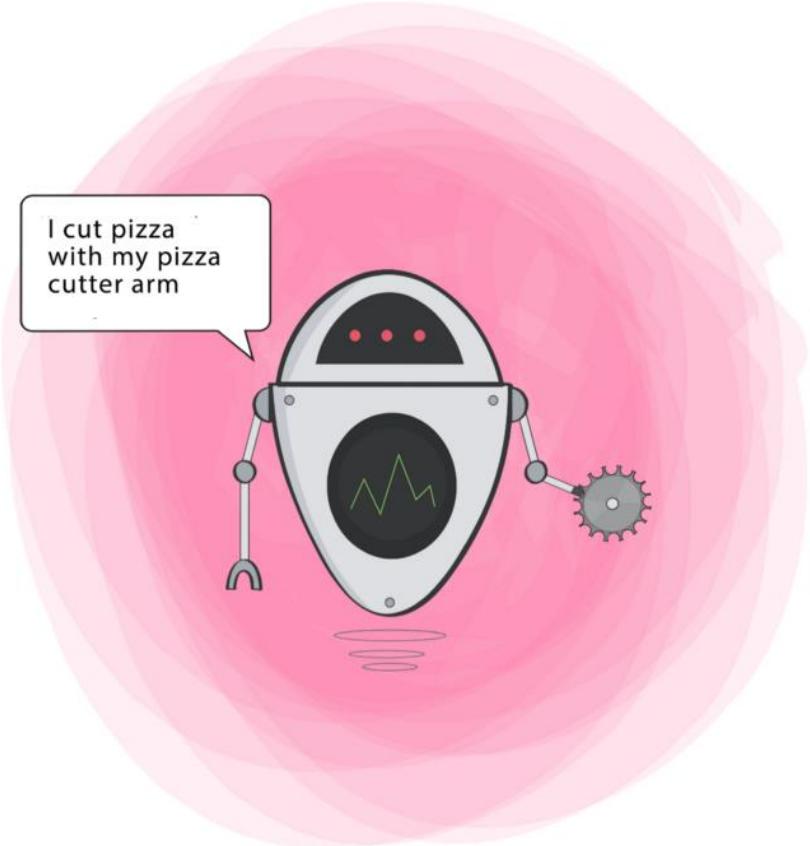
*A class' dependencies are supplied to the class – not created by the class itself!*



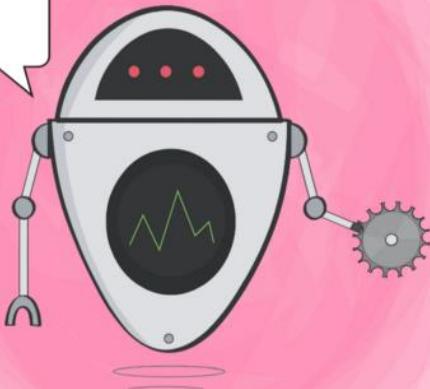


# DIP in Summary

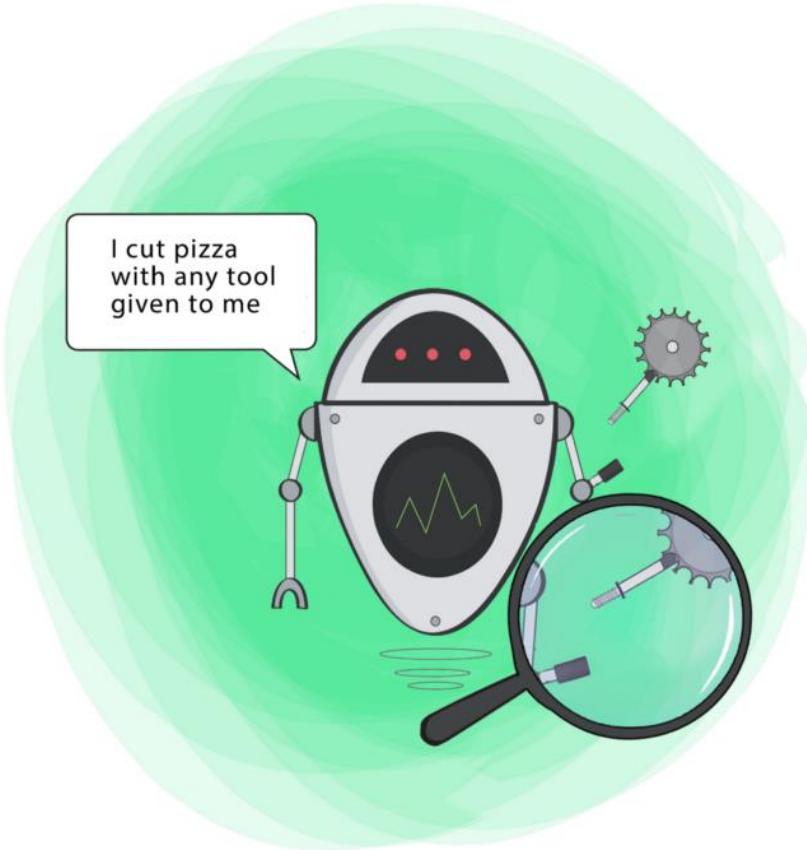
- ▶ Idea
  - Don't depend on concrete implementations! Only depend upon abstractions
  - Feed the dependencies needed into a class' constructor
- ▶ Why?
  - Maximize freedom to change implementations, because a class will never depend upon specific implementations – only their abstraction
  - Testability
  - Minimize dependencies and dependencies' dependencies
- ▶ Consequences
  - Classes will become loosely coupled
  - Your program becomes eligible for Dependency Injection



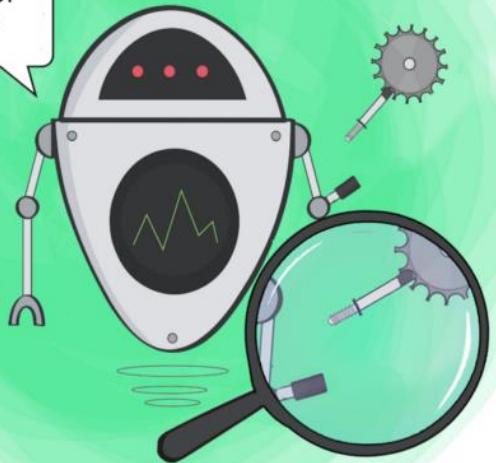
I cut pizza  
with my pizza  
cutter arm



## Dependency Inversion



I cut pizza  
with any tool  
given to me



# Agenda

- ▶ Introducing SOLID
- ▶ Single Responsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ ***(Optional) Lab 1***
- ▶ Workshop A.1

# Agenda

- ▶ Introducing SOLID
- ▶ Single Responsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ *(Optional) Lab 1*
- ▶ **Workshop A.1**

# Workshop A.1:

## Initial Setup and Inspection of Project



# Summary

- ▶ Introducing SOLID
- ▶ Single Responsibility Principle (SRP)
- ▶ Open/Closed Principle (OCP)
- ▶ Liskov's Substitution Principle (LSP)
- ▶ Interface Segregation Principle (ISP)
- ▶ Dependency Inversion Principle (DIP)
- ▶ Workshop A.1



**WINCUBATE**



Jesper Gulmann Henriksen  
PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31  
Email : [jgh@wincubate.net](mailto:jgh@wincubate.net)  
WWW : <http://www.wincubate.net>

Ringgårdsvej 4A  
8270 Højbjerg  
Denmark