



# **OBJECT ORIENTED PROGRAMMING AND S.O.L.I.D. PRINCIPLES**

Universidad de La Laguna ~ PAI 2020-2021  
Viren Sajju Dhanwani Dhanwani and J. Daniel Escánez Expósito

# WHO ARE WE?



Viren S. Dhanwani  
alu0101230948  
@virensdd



J. Daniel Escáñez  
alu0101238944  
@jdanielescanez



# CONTENTS

- **Object Oriented Programming terms**
  - Classes and Objects
  - Instance and class variables/methods
- **OOP Features**
  - Encapsulation
  - Composition
  - Delegation
  - Inheritance
  - Polymorphism
  - Abstraction



# CONTENTS

- **OOP Principles**
  - **K.I.S.S.** (Keep It Simple, Stupid)
  - **D.R.Y.** (Don't Repeat Yourself)
  - **Y.A.G.N.I.** (You Aren't Going to Need It)
  - **S.O.L.I.D.**
    - Single Responsibility
    - Open/Closed
    - Liskov Substitution
    - Interface Segregation
    - Dependency Inversion

# REPOSITORY

PAI-ULL / 2020-2021-pai-trabajo-oodesign-solid-viren-dhanwani-daniel-escanez

generated from ULL-ESIT-INF-PAI-2020-2021-Trabajos-Alumnado

Watch 1

Star 0

Fork 0

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

master

1 branch

0 tags

Go to file

Add file

Code

al0101230948 Presentation added

d1209ad 7 days ago

25 commits

bib	README updated	7 days ago
slides	Presentation added	7 days ago
src	README updated	7 days ago
.eslintrc.cjs	eslint configured	7 days ago
.gitignore	Initial commit	11 days ago
LICENSE	Initial commit	11 days ago
README.md	README updated	7 days ago
package-lock.json	eslint configured	7 days ago
package.json	eslint configured	7 days ago

README.md

OOP and SOLID Principles

Authors

Viren Sajju Dhanwani Dhanwani José Daniel Escánez Expósito

PAI Lab assignment (lab, class presentation with slides) structure

About

2020-2021-pai-trabajo-oodesign-solid-viren-dhanwani-daniel-escanez created by GitHub Classroom

Readme

MIT License

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Contributors 2

al0101230948 Viren Sajju Dhanwa...

github-classroom[bot]

Languages

JavaScript 100.0%

# OBJECT ORIENTED PROGRAMMING TERMS

# OOP TERMS: CLASSES AND OBJECTS



# OOP TERMS: INSTANCE AND CLASS VARIABLES

● ● ● Intance Variables ~ Viren S. Dhanwani and J. Daniel Esc...

```
export class Point2D {  
  constructor(abscissa = 0, ordinate = 0) {  
    this.abscissa_ = abscissa;  
    this.ordinate_ = ordinate;  
  }  
}
```

INSTANCE

CLASS

```
Welcome to Node.js v14.16.0.  
Type ".help" for more information.  
> Math.PI  
3.141592653589793  
> []
```



# OOP TERMS: INSTANCE AND CLASS METHODS

```
Instance and class methods ~ Viren S. Dhanwani and J. Daniel Escáñez

class Point {
  ...

  static staticDistance(a = new Point(), b = new Point()) {
    const ABSCISSA_DISTANCE = a.getAbscissa() - b.getAbscissa();
    const ORDINATE_DISTANCE = a.getOrdinate() - b.getOrdinate();
    return Math.sqrt(ABSCISSA_DISTANCE ** 2 + ORDINATE_DISTANCE ** 2);
  }

  instanceDistance(anotherPoint = new Point()) {
    const ABSCISSA_DISTANCE = this.abscissa_ - anotherPoint.getAbscissa();
    const ORDINATE_DISTANCE = this.ordinate_ - anotherPoint.getOrdinate();
    return Math.sqrt(ABSCISSA_DISTANCE ** 2 + ORDINATE_DISTANCE ** 2);
  }

  ...
}

const ORIGIN = new Point();
const POINT = new Point(3, 4);

console.log(Point.staticDistance(ORIGIN, POINT)); // 5
console.log(ORIGIN.instanceDistance(POINT)); // 5
```

# OBJECT ORIENTED PROGRAMMING FEATURES



# OOP FEATURES: ENCAPSULATION

Encapsulation ~ Viren S. Dhanwani and J. Daniel Escáñez

```
class Counter {  
  #count = 0;  
  click() {  
    this.#count += 1;  
  }  
  getCount() {  
    return this.#count;  
  }  
}  
  
const myCounter = new Counter();  
console.log(myCounter.getCount());  
  
for (let i = 0; i < 4; i++) {  
  myCounter.click();  
  console.log(myCounter.getCount());  
}
```

# PUBLIC, PROTECTED AND PRIVATE IN JS (EXPERIMENTAL FEATURE)


Public and private field declarations are an experimental feature (stage 3) proposed at TC39, the JavaScript standards committee. Support in browsers is limited, but the feature can be used through a build step with systems like Babel.

## Browser compatibility


[Report problems with this compatibility data on GitHub](#)

	Desktop						Mobile						Node.js
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Node.js
Private class fields	74	79	No ★	No	62	No	74	74	No ★	53	No	No	12.0.0

 Full support

 No support

★ See implementation notes.

 User must explicitly enable this feature.

[https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Classes/Private\\_class\\_fields](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Classes/Private_class_fields)

# PUBLIC, PROTECTED AND PRIVATE IN JS (EXPERIMENTAL FEATURE)

- In the constructor
  - PUBLIC VARIABLE
    - `variableName`
  - PROTECTED VARIABLE
    - `variableName_`
- In the class body
  - PRIVATE VARIABLE
    - `#variableName`

**Browser compatibility**

[Report problems with this compatibility data on GitHub](#)

	Desktop						Mobile						Server-side
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	Node.js
Private class fields	74	79	No ★ 🚩	No	62	No	74	74	No ★ 🚩	53	No	No	12.0.0

Full support

No support

★ See implementation notes.

🚩 User must explicitly enable this feature.

[https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Classes/Private\\_class\\_fields](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Classes/Private_class_fields)

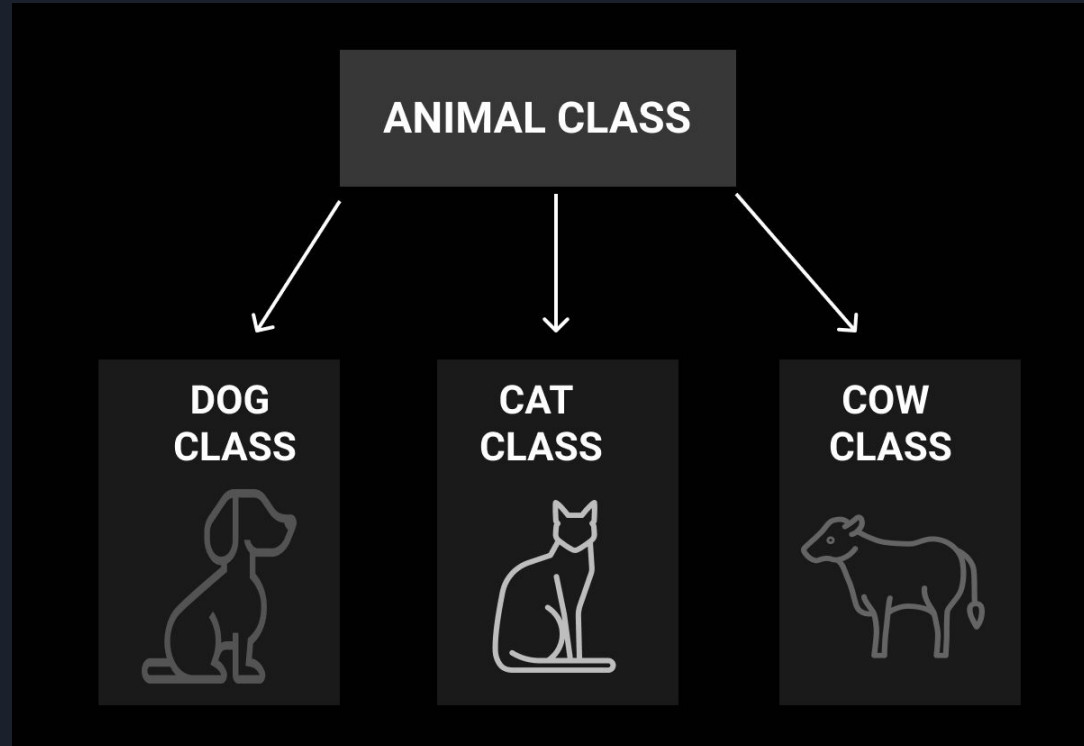


# OOP FEATURES: COMPOSITION AND DELEGATION

Composition and Delegation ~ Viren S. Dhanwani and J. Daniel Escáñez

```
export class SegmentComposition {  
  constructor(firstPoint = new Point2D(), secondPoint = new Point2D()) {  
    this.firstPoint_ = firstPoint;  
    this.secondPoint_ = secondPoint;  
  }  
  length() {  
    return this.firstPoint_.instanceDistance(this.secondPoint_);  
  }  
}
```

# OOP FEATURES: INHERITANCE





# OOP FEATURES: INHERITANCE

Inheritance ~ Viren S. Dhanwani and J. Daniel Escáñez

```
class Person {
  constructor(first, last, age, gender, interests) {
    this.name_ = {
      first,
      last,
    };
    this.age = age;
    this.gender_ = gender;
    this.interests_ = interests;
  }
  /** Outputs the person greeting and saying her name */
  greeting() {
    console.log('Hi! I\'m ' + this.name_.first);
  }
  /** Outputs the person farewell and her name */
  farewell() {
    console.log(this.name_.first + ' has left the building. Bye for now!');
  }
}
```





# OOP FEATURES: INHERITANCE

Inheritance ~ Viren S. Dhanwani and J. Daniel Escáñez

```
class Teacher extends Person {  
  constructor(first, last, age, gender, interests, subject, grade) {  
    this.name = {  
      first,  
      last  
    };  
    this.age = age;  
    this.gender = gender;  
    this.interests = interests;  
    // subject and grade are specific to Teacher  
    this.subject = subject;  
    this.grade = grade;  
  }  
}
```

# OOP FEATURES: INHERITANCE

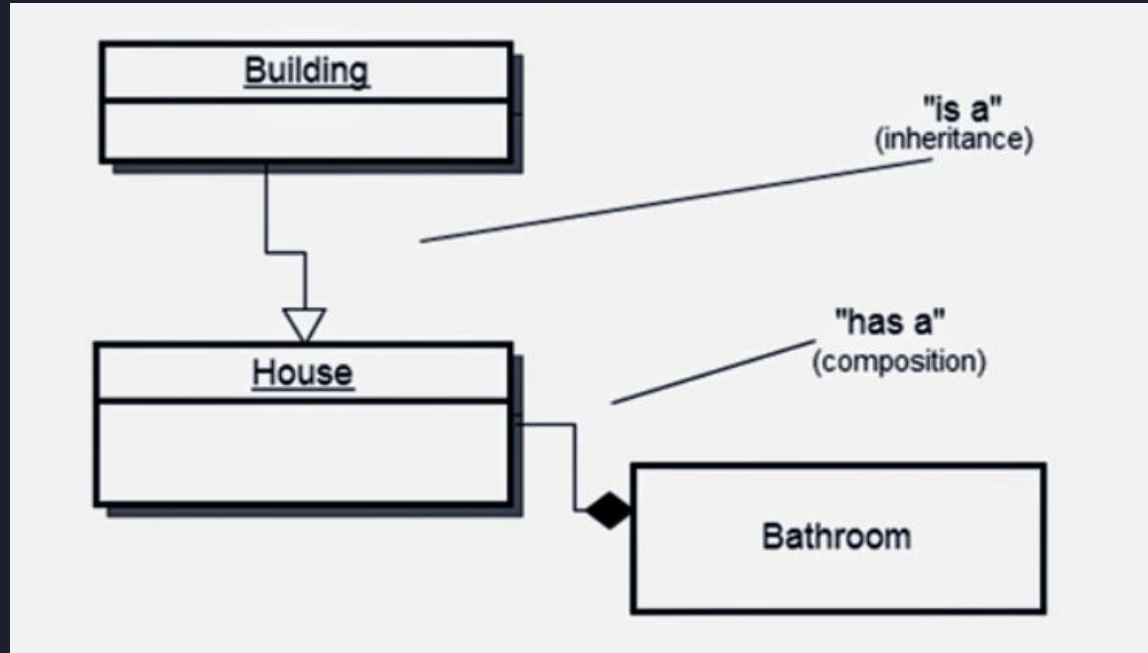
Inheritance ~ Viren S. Dhanwani and J. Daniel Escáñez

```
class Teacher extends Person {
  constructor(first, last, age, gender, interests, subject, grade) {
    this.name = {
      first,
      last
    };
    this.age = age;
    this.gender = gender;
    this.interests = interests;
    // subject and grade are specific to Teacher
    this.subject = subject;
    this.grade = grade;
  }
}
```

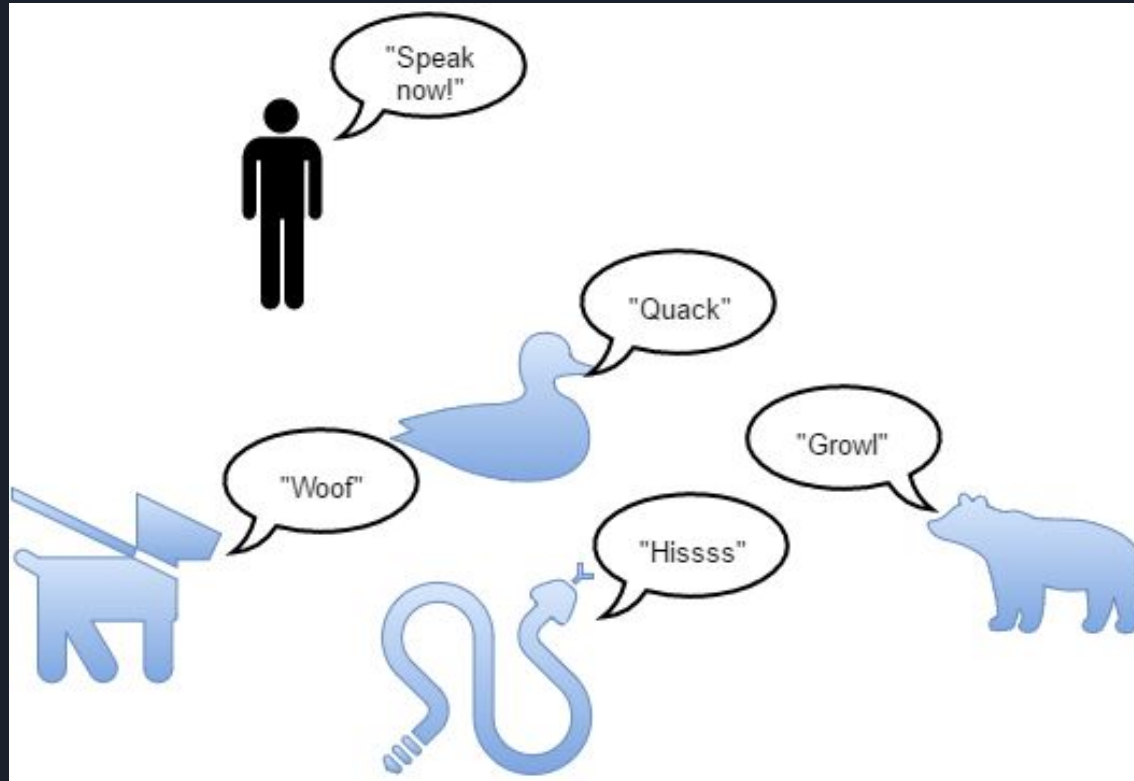
Inheritance ~ Viren S. Dhanwani and J. Daniel Escáñez

```
class Teacher extends Person {
  constructor(first, last, age, gender, interests, subject, grade) {
    super(first, last, age, gender, interests);
    // Subject and grade are specific to Teacher
    this.subject = subject;
    this.grade_ = grade;
  }
}
```

# INHERITANCE VS COMPOSITION



# OOP FEATURES: POLYMORPHISM





# OOP FEATURES: ABSTRACTION

```
Abstraction ~ Viren S. Dhanwani and J. Daniel Escáñez

class Employee {
  constructor() {
    if (this.constructor === Employee) {
      throw new Error('Object of Abstract Class cannot be created');
    }
  }
  display() {
    throw new Error('Abstract Method has no implementation');
  }
}

class Manager extends Employee {
  display() {
    console.log('I am a Manager');
  }
}

const MANAGER = new Manager();
MANAGER.display(); // -> I am a Manager
```

# OBJECT ORIENTED PROGRAMMING PRINCIPLES

# K.I.S.S. (Keep It Simple, Stupid)



# D.R.Y. (Don't Repeat Yourself)

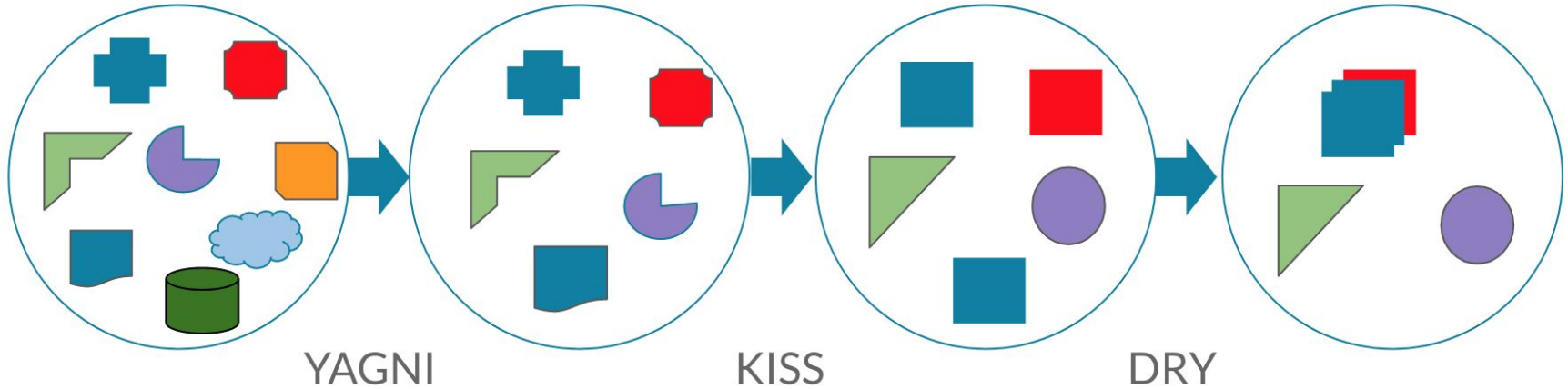




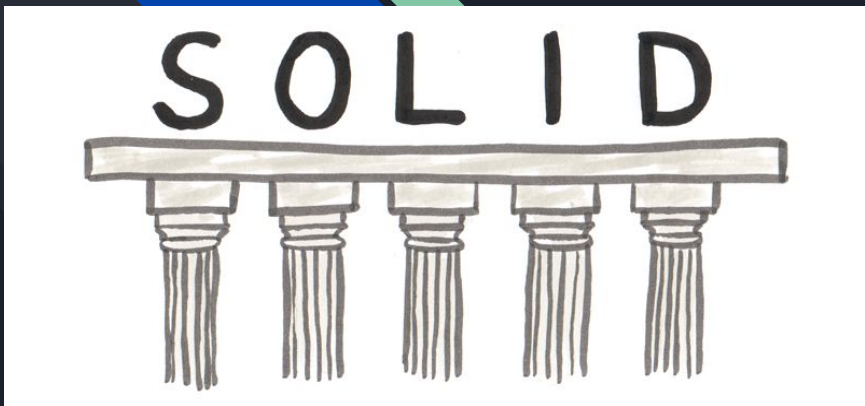
# Y.A.G.N.I. (You Aren't Going to Need It)



# Y.A.G.N.I. - K.I.S.S. - D.R.Y.



# SOLID PRINCIPLES



## **S**ingle Responsibility Principle

A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)



## **O**pen / Closed Principle

A software module (it can be a class or method) should be open for extension but closed for modification.



## **L**iskov Substitution Principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.



## **I**nterface Segregation Principle

Clients should not be forced to depend upon the interfaces that they do not use.



## **D**ependency Inversion Principle

Program to an interface, not to an implementation.

## SOLID: Single Responsibility



“There should never be more than a reason for a class to change”

Robert C. Martin

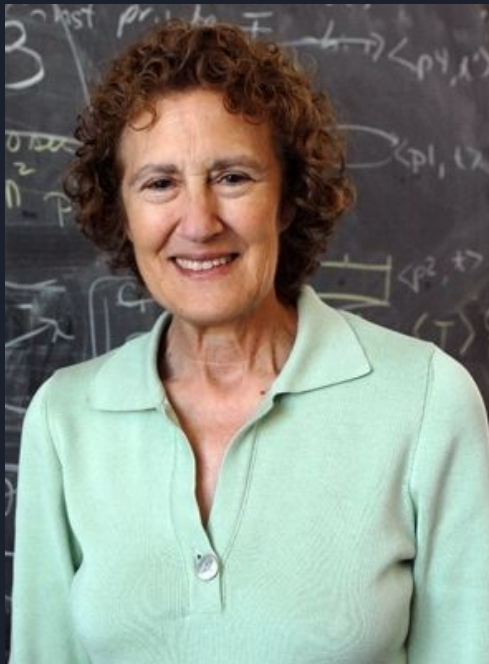
# SOLID: Open/Closed



“Software entities should be open for extension but closed for modification”

Bertrand Meyer

# SOLID: Liskov Substitution



Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program

Barbara Liskov

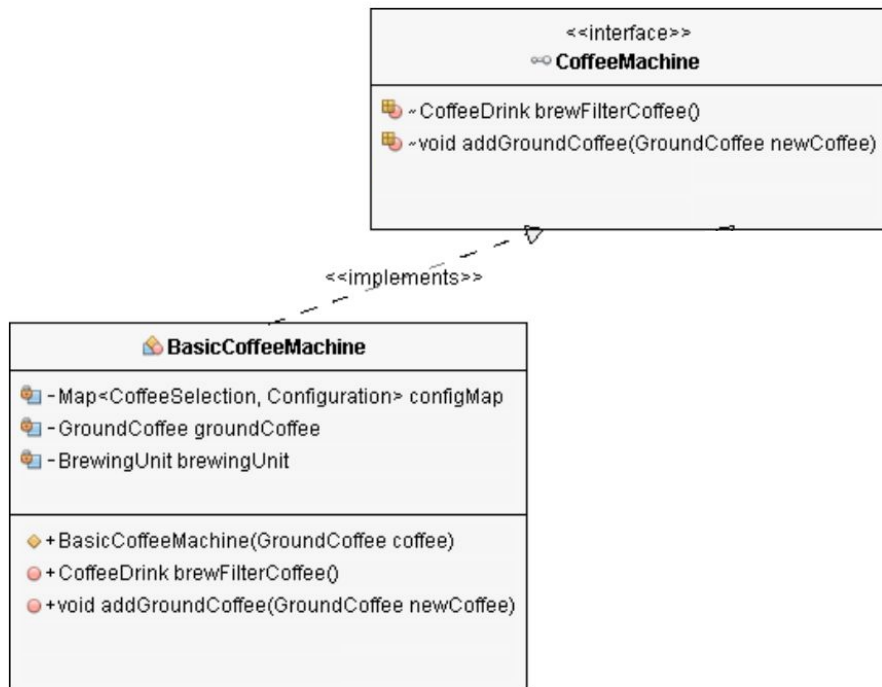
# SOLID: Interface Segregation



“Clients should not be forced to depend upon the interfaces that they do not use”

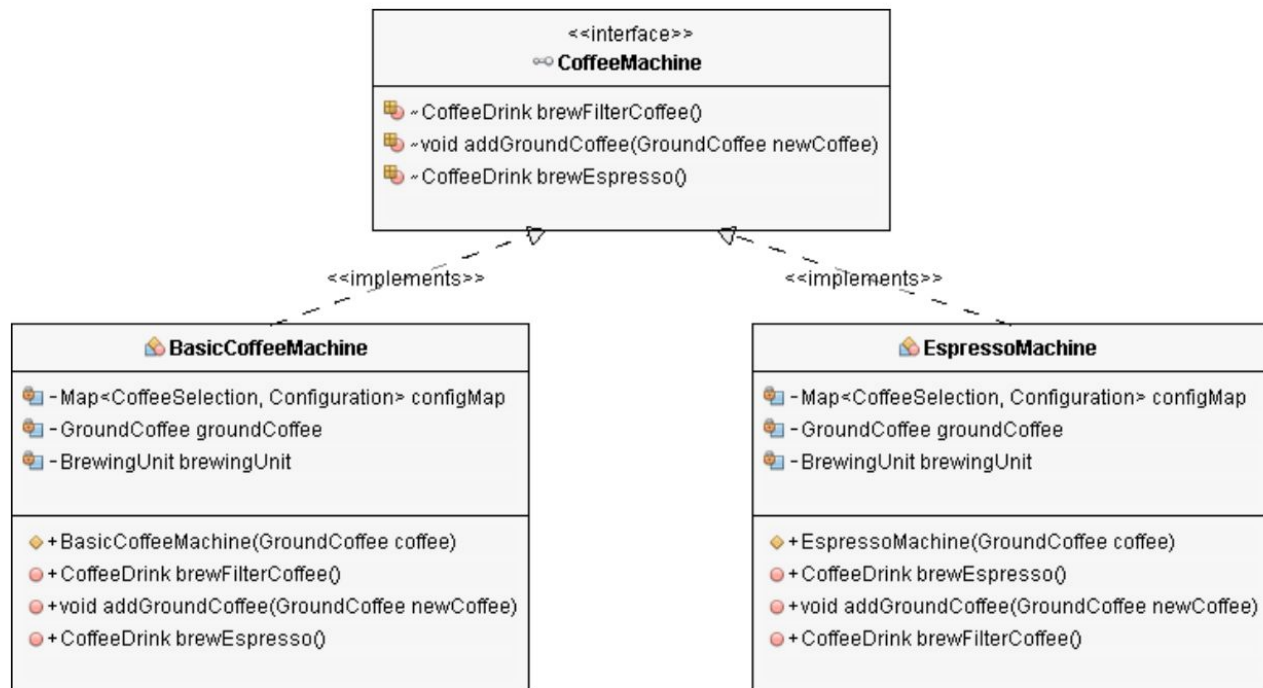
Robert C. Martin

# SOLID: Interface Segregation

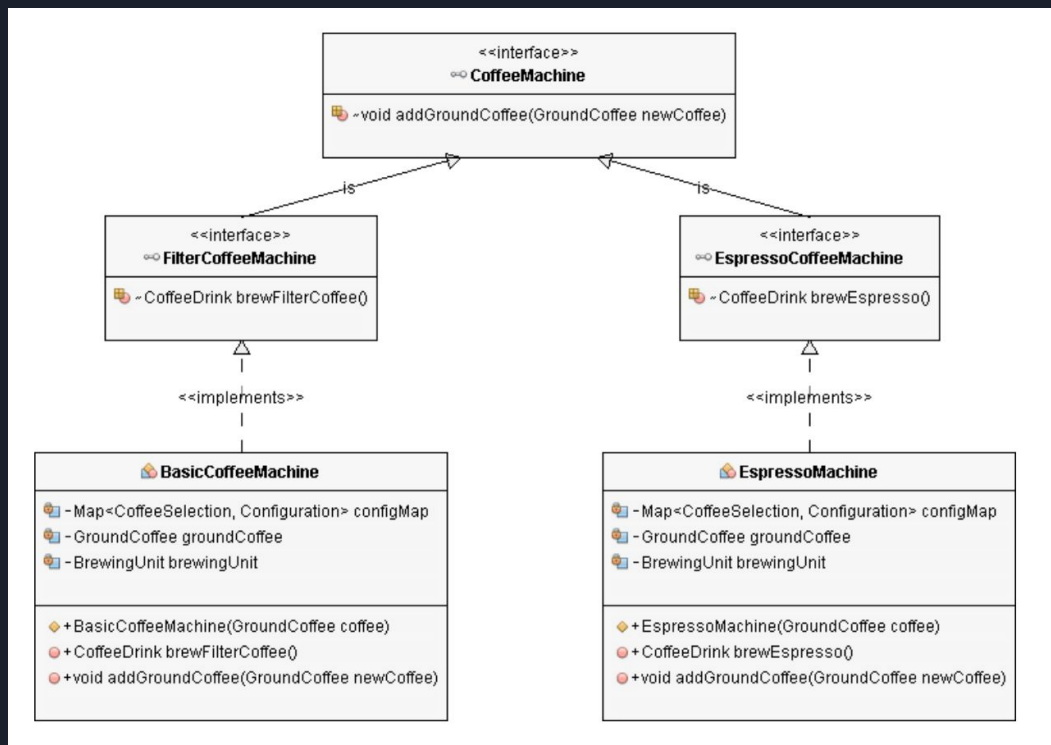




# SOLID: Interface Segregation



# SOLID: Interface Segregation



# SOLID: Interface Segregation



“Clients should not be forced to depend upon the interfaces that they do not use”

Robert C. Martin

# SOLID: Dependency Inversion

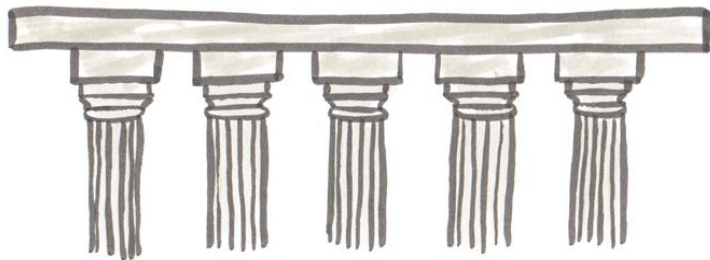


- A. “High-level modules should not depend on low-level modules. Both should depend on abstractions”
- B. Abstractions should not depend on details. Details should depend on abstractions

Robert C. Martin

# SUMMARY

SOLID



## **S**ingle Responsibility Principle

A class should have only a single responsibility (i.e. only one potential change in the software's specification should be able to affect the specification of the class)



## **O**pen / Closed Principle

A software module (it can be a class or method) should be open for extension but closed for modification.



## **L**iskov Substitution Principle

Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.



## **I**nterface Segregation Principle

Clients should not be forced to depend upon the interfaces that they do not use.



## **D**ependency Inversion Principle

Program to an interface, not to an implementation.

# CONTACT US



Viren S. Dhanwani  
alu0101230948  
@virensdd



J. Daniel Escáñez  
alu0101238944  
@jdanielescanez



# BIBLIOGRAPHY

[OOP and SOLID Principles Repository](#)

[Classes - JavaScript | MDN](#)

[S.O.L.I.D The first 5 principles of Object Oriented Design with JavaScript](#)

[Object-oriented programming in JavaScript #1. Abstraction](#)

[JavaScript: Object Modelling with Behavior Delegation](#)

[JavaScript Class Inheritance](#)

[KISS, DRY, and Code Principles Every Developer Should Follow](#)

[Pablo's Topic of the Month – March: SOLID Principles](#)

# QUESTIONS?