The benefits of SOLID in software development

Ruben Agudo Santos (GS-AIS-HR)

Table of contents

- What is SOLID?
- Single Responsibility Principle
- Open-Closed Principle
- Liskov's Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle
- Why we should care
- QA
- Bibliography

What is SOLID?

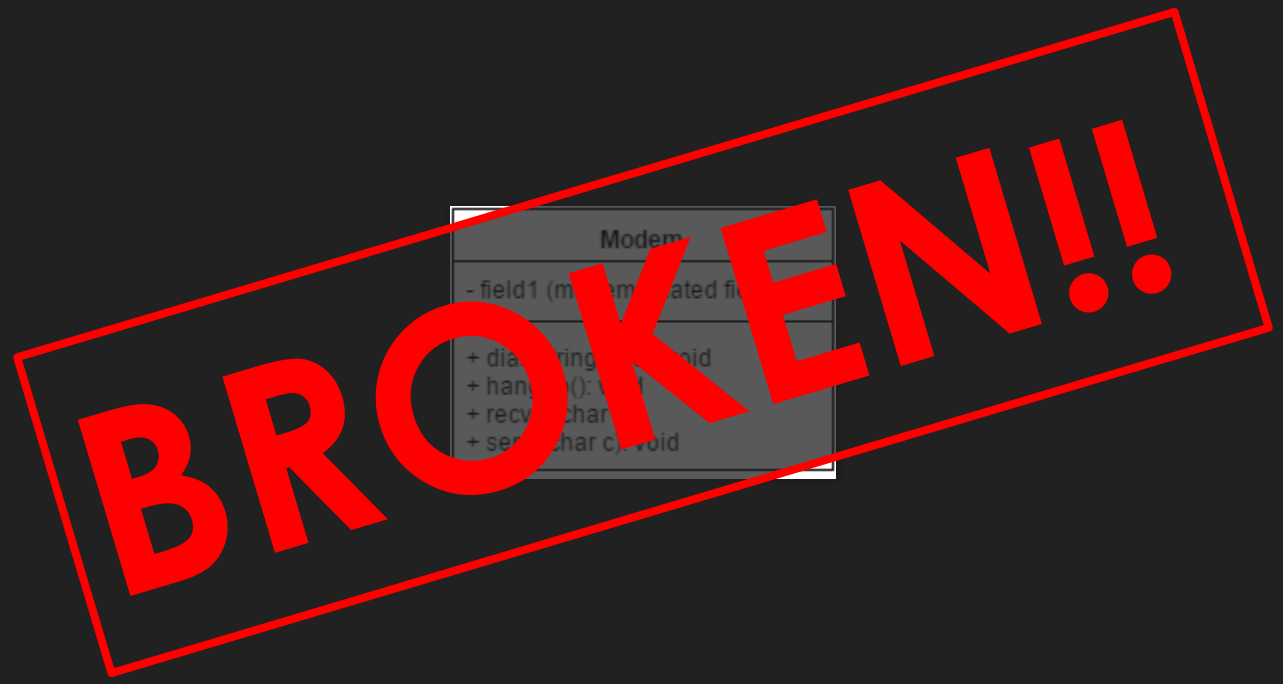
Not a state of matter

Uncle Bob

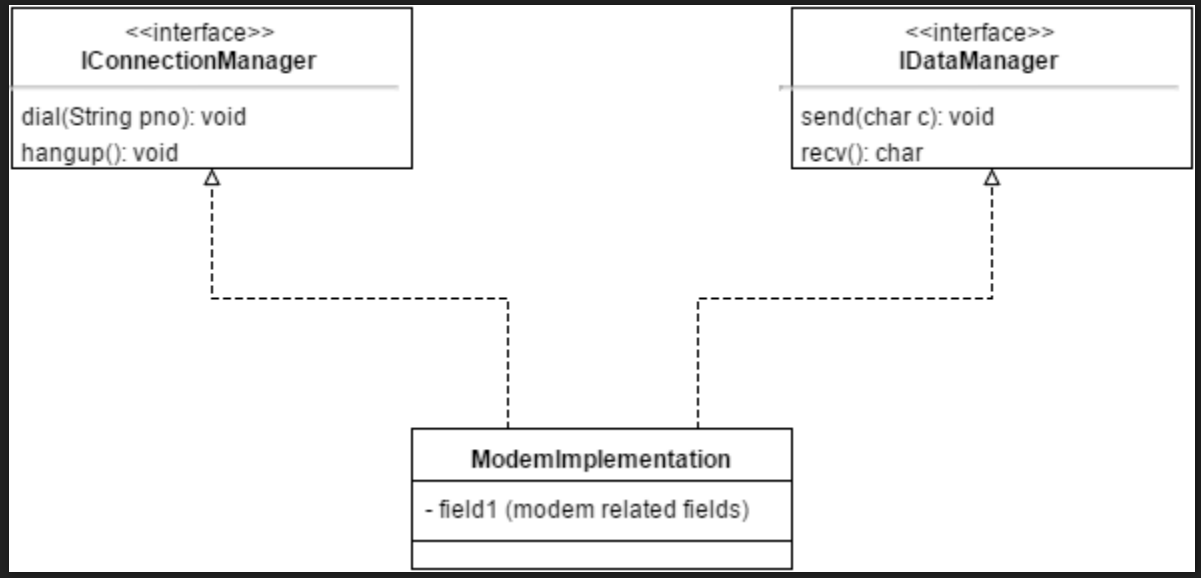
Single Responsibility Principle

"THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE"

Single Responsibility Principle



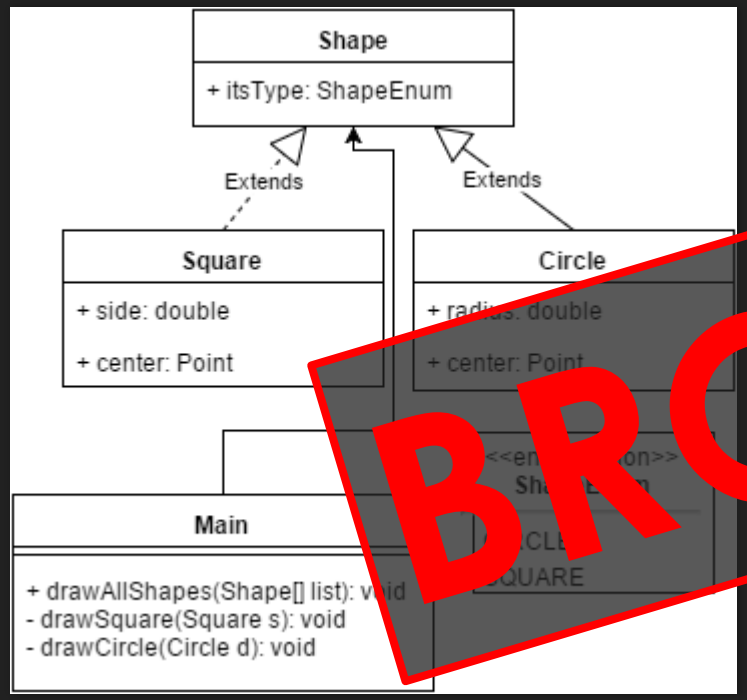
Single Responsibility Principle



Open-Closed Principle

"SOFTWARE ENTITIES SHOULD BE OPEN FOR EXTENSION, BUT CLOSED FOR MODIFICATION"

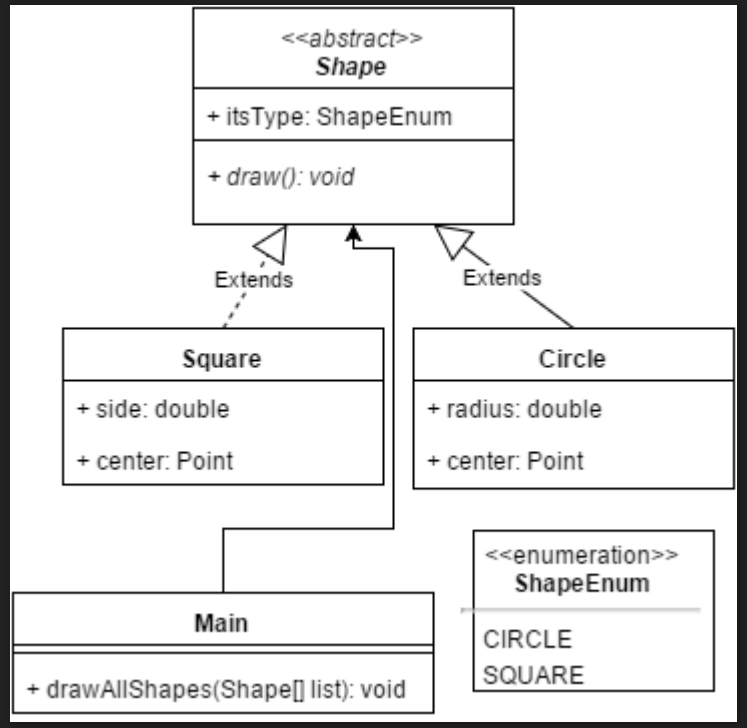
Open-Closed Principle



BROKEN!!

```
public void DrawAllShapes(Shape[] shapes) {
  for (int i = 0; i < shapes.length; i++) {
    Shape s = shapes[i];
    switch(s.itsType) {
      case Shape.SQUARE:
        drawSquare((Square) s);
        break;
      case Shape.CIRCLE:
        drawCircle((Circle) s);
        break;
    }
  }
}
```

Open-Closed Principle




```
public void DrawAllShapes(Shape[] shapes) {
  for (int i = 0; i < shapes.length; i++) {
    Shape s = shapes[i];
    s.draw();
  }
}
```

Liskov Substitution Principle

"FUNCTIONS THAT USE POINTERS OR REFERENCES TO BASE CLASSES MUST BE ABLE TO USE OBJECTS OF DERIVED CLASSES WITHOUT KNOWING IT"

Liskov Substitution Principle

Liskov Substitution Principle

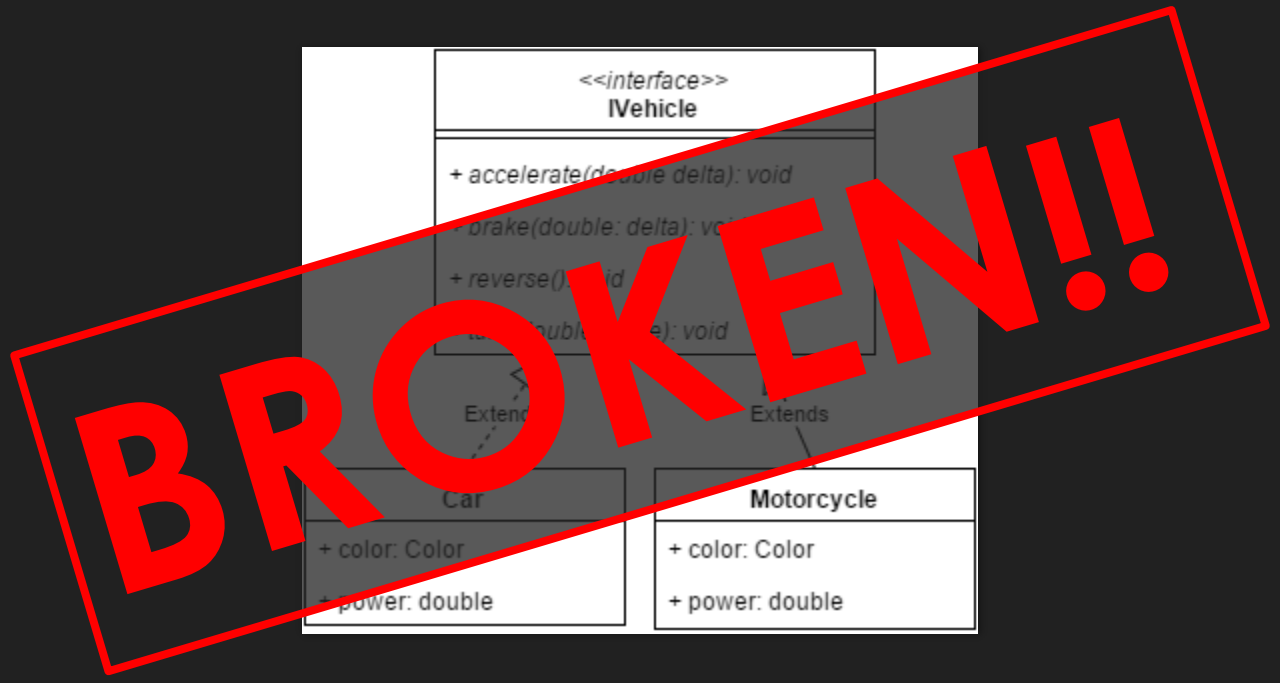
Moral of the story:

- Model the classes based on behavior.
- *"...when redefining a routine [in a derivative], you may only replace its precondition by a weaker one, and its postcondition by a stronger one."*

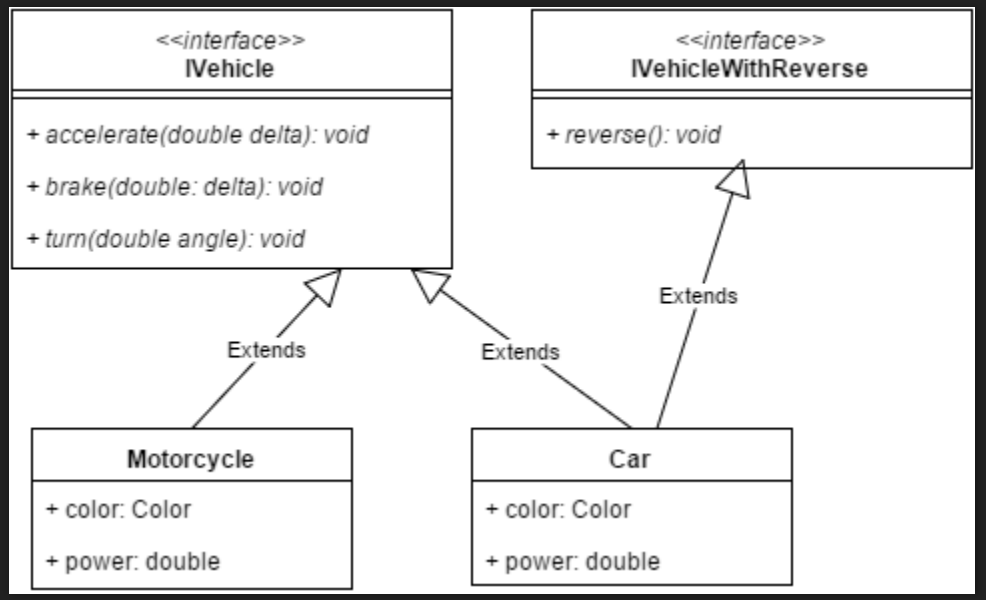
Interface Segregation Principle

"CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES THAT THEY DO NOT USE"

Interface Segregation Principle



Interface Segregation Principle

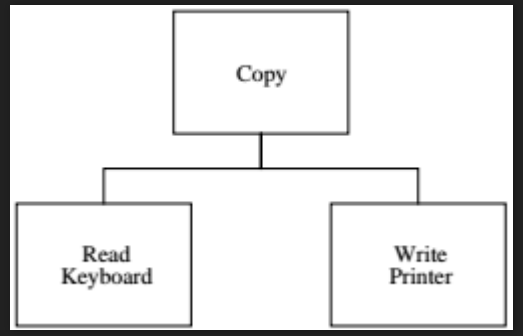


Dependency Inversion Principle

A. *"HIGH LEVEL MODULES SHOULD NOT DEPEND UPON LOW LEVEL MODULES. BOTH SHOULD DEPEND UPON ABSTRACTIONS"*

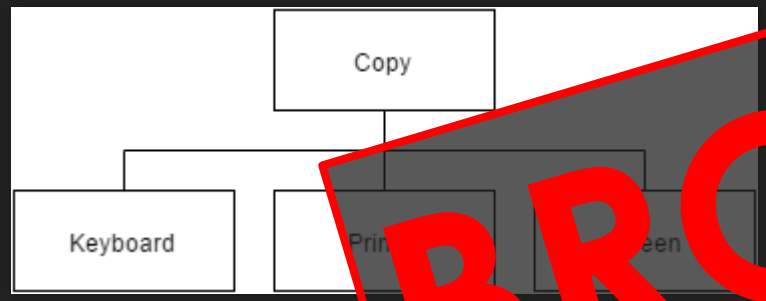
B. *"ABSTRACTIONS SHOULD NOT DEPEND UPON DETAILS. DETAILS SHOULD DEPEND UPON ABSTRACTIONS"*

Dependency Inversion Principle




```
void Copy() {
  int c;
  while ((c = readKeyboard()) != EOF) {
    writePrinter(c);
  }
}
```

Dependency Inversion Principle

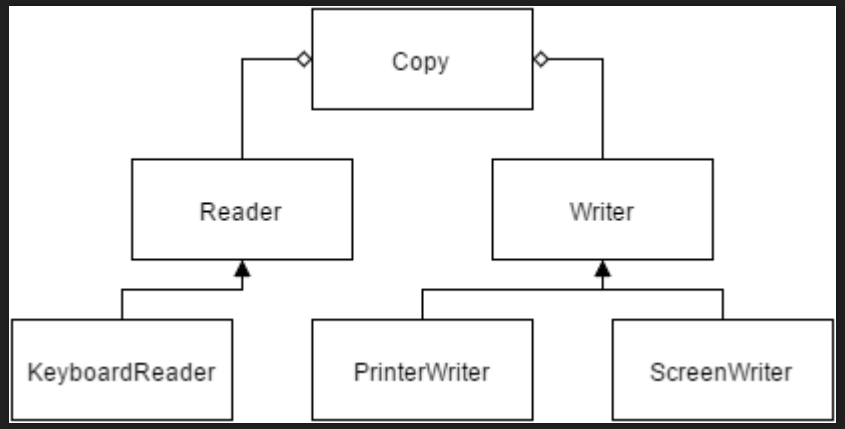



```
enum OutputDevice {printer, disk};

void Copy(OutputDevice dev) {
  int c;
  while ((c = ReadKeyboard()) != EOF)
    if (dev == printer) WritePrinter(c);
    else WriteDisk(c);
}
```

BROKEN!!

Dependency Inversion Principle




```
abstract class Reader {
  public abstract int read();
};

abstract class Writer {
  public abstract void write(char);
};

void copy(Reader r, Writer w) {
  int c;
  while((c=r.Read()) != EOF)
    w.Write(c);
}
```

Why we should care

- Cleaner code (Remember Ben Wolff's presentation)
- Code smells are kept away
- Codebase that is maintainable and expandable
- Usually integrated in Agile methodologies, i.e. Scrum

QA

THANKS FOR COMING!

Bibliography

Robert C. Martin

http://www.objectmentor.com/resources/articles/srp.pdf

http://www.objectmentor.com/resources/articles/ocp.pdf

http://www.objectmentor.com/resources/articles/lsp.pdf

http://www.objectmentor.com/resources/articles/isp.pdf

http://www.objectmentor.com/resources/articles/dip.pdf

Duck Panel

https://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/

Robert C. Martin Photo

Uploaded by Tim-bezhashvyly, under CC BY-SA 4.0