SOLID

“The critical design tool for software development is a mind well educated in design principles”

THE SOLID ACRONYM

- The SOLID acronym is composed of the first letters of 5 design principles:

S	Single Responsibility Principle	(SRP)
O	Open Closed Principle	(OCP)
L	Liskov's Substitution Principle	(LSP)
I	Interface Segregation Principle	(ISP)
D	Dependency Inversion Principle	(DIP)

SINGLE RESPONSIBILITY PRINCIPLES (SRP)

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

SINGLE RESPONSIBILITY PRINCIPLES

A class should only have one reason to change

- *…why?*

*“A functional unit on a given level of abstraction should only be responsible for a **single aspect** of a system’s requirements.*

*An aspect of requirements is a trait or property of requirements, **which can change independently of other aspects**.”*

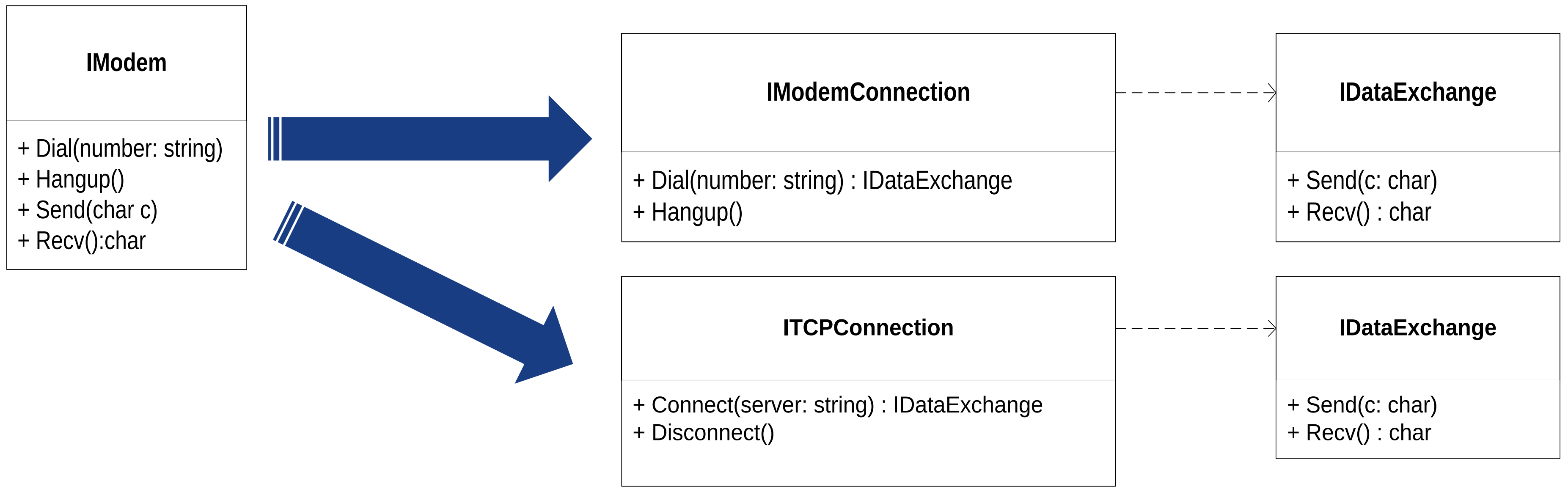
SINGLE RESPONSIBILITY PRINCIPLES

- If a class has several responsibilities (reasons to change), changes because of one of them may have adverse effects on others.

- Breaking SRP may lead to designs that are hard to test, maintain, change etc.

SRP - MODEM

* What is the problem with IModem?



SRP – PERSON

```
public class Person {
    public string FirstName { get; set; }
    public string LastName  { get; set; }
    public Gender Gender { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string Format(string formatType) {
        switch(formatType) {
            case "JSON":
               // implement JSON formatting here
               return jsonFormattedString;
            case "FirstAndLastName":
              // implementation of first & lastname formatting here
              return firstAndLastNameString;
            default:
              // implementation of default formatting
              return defaultFormattedString;
        }
    }
}
```

THE OPEN-CLOSED PRINCIPLE (OCP)

OPEN-CLOSE PRICIPLE

Software entities should be open for extension but closed for modification

- The OCP helps us develop SW systems that can cope with changing requirements in the future
- Let’s have a look…

OPEN-CLOSE DISSECTED

Software entities should be open for extension but closed for modification

- *"Open for extension"*: You should expect requirements to change, and allow this change to be implemented through extension of the existing design.

- *"Closed for modification"*: When the requirements calls for extension of your code, it should not be necessary to modify

OCP NEWBIE DIALOG

You want to implement changes, but without changing things?

Basically, yes…

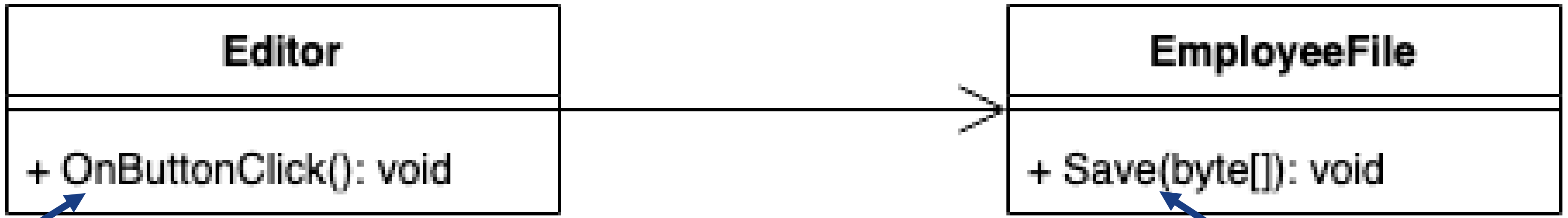
WTF? Implementing changes without changing stuff?!?

Yes. Implementing changes do not necessarily mean “changing”. It often means “extending”.

Sigh…

Here, let me show you…

STORAGE WITHOUT OCP



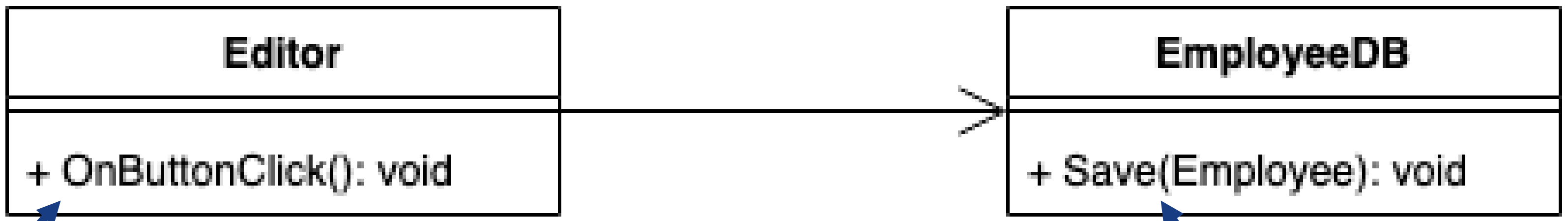
```
public class Editor {
  private EmployeeFile storage;
  Editor() {
    storage = new EmployeeFile()
  }
  public void OnButtonClick() {
    for (attr in employee) {
      var data = attr.Format();
      var bytes = // convert data to bytes
      storage.Save(bytes);
    }
  }
}
```

```
public class EmployeeFile {
  public void Save(byte[] b) {
    fileStream.Write(b, 0, b.length);
  }
}
```

NEW REQUIREMENTS

Need to support database

STORAGE WITHOUT OCP



```
public class Editor {
  private EmployeeDb storage;
  Editor() {
    storage = new EmployeeDB()
  }
  public void OnButtonClick() {
    storage.save(employee);
  }
}
```

```
public class EmployeeDB {
  public void Save(Employee e) {
    db.Save(e);
  }
}
```

OCP NEWBIE DIALOG

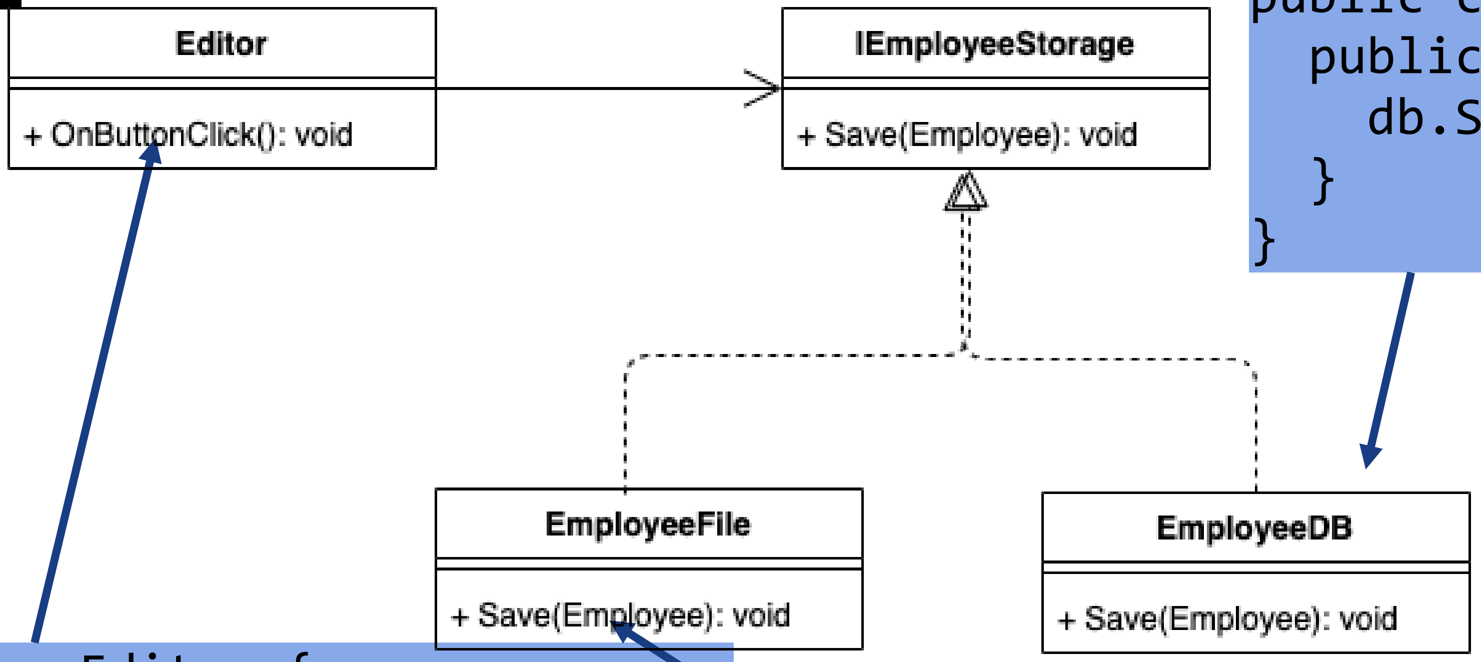
*Damn…what went wrong? I didn't want to change **`Editor`**, only the storage it uses!*

***`Editor`** was too tightly coupled to **`Storage`**, therefore extensions to the storage type used rippled through – **`Editor`** was not closed for modifications when changes were needed…*

But how could I ever avoid that? It is a whole new storage type I am trying to use. And what if the supported storage is changed again? Or if we need to support several storages?

*We could apply OCP and make **`Editor`** open for extension (the new storage or storages) but closed for modifications…let me show you!*

STORAGE WITH OCP



```
public class EmployeeDB {
  public void Save(Employee e) {
    db.Save(e);
  }
}
```

```
public class Editor {
  private IEmployeeStorage storage;
  Editor(IEmployeeStorage s) {
    storage = s;
  }
  public void OnButtonClick() {
      storage.Save(employee);
    }
  } }
```

```
public class EmployeeFile {
  public void Save(Employee e) {
    for (attr in e) {
      var data = attr.Format();
      var b = // convert data
      fileStream.Write(b, 0, b.length);
    }
} }
```

RECAP

- SOLID: 5 principles for good OO design
- S: SRP – Single Responsible Principle. A class should have exactly one reason to change
- O: OCP – Open-Closed Principle. A class should be open for extension but closed for modifications.

QUESTIONS?

AARHUS
UNIVERSITY