Interface Segregation Principle

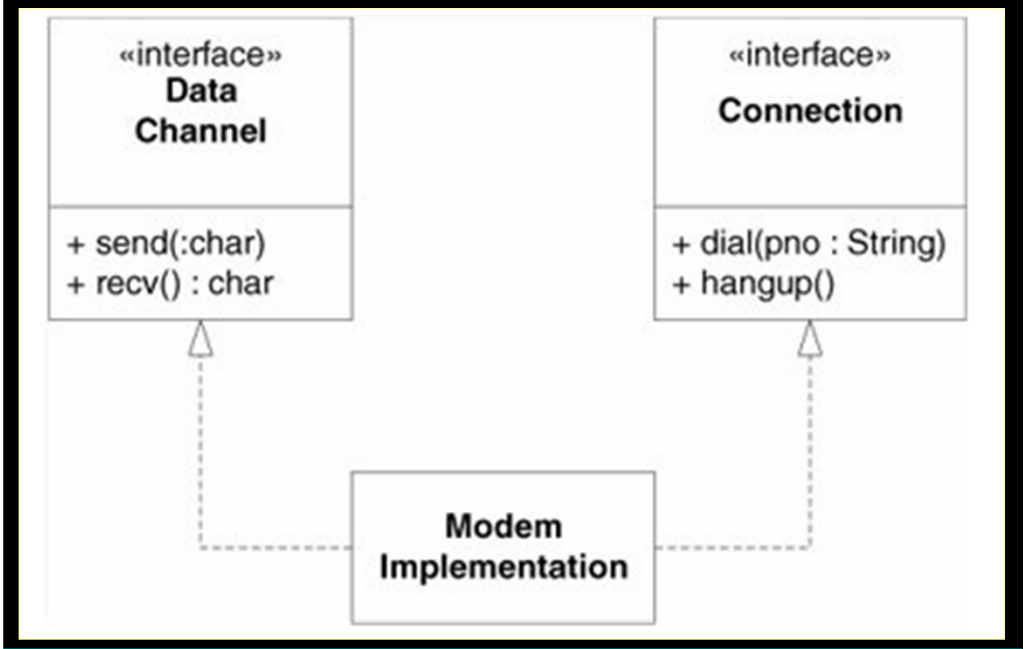
- Interfaces should be as fine-grained as possible.
  - Any problem:

```
public interface Modem {
  public void dial(String pno);
  public void hangup();
  public void send(Char c);
  public char recv();
}
```

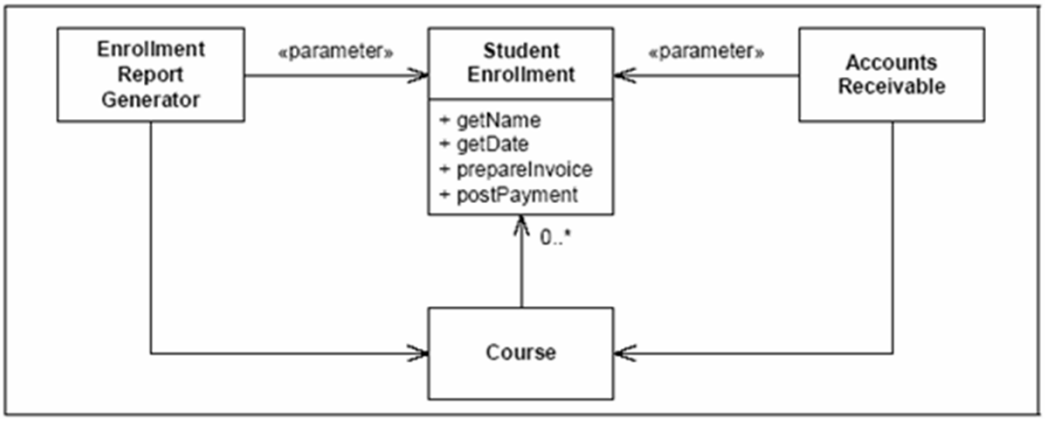

Clients should not be forced to depend upon the interfaces that they do no use. – Robert Martin

If a class implements an interface with multiple methods, but in one of the methods throws notSupportedException, then this principle is violated.

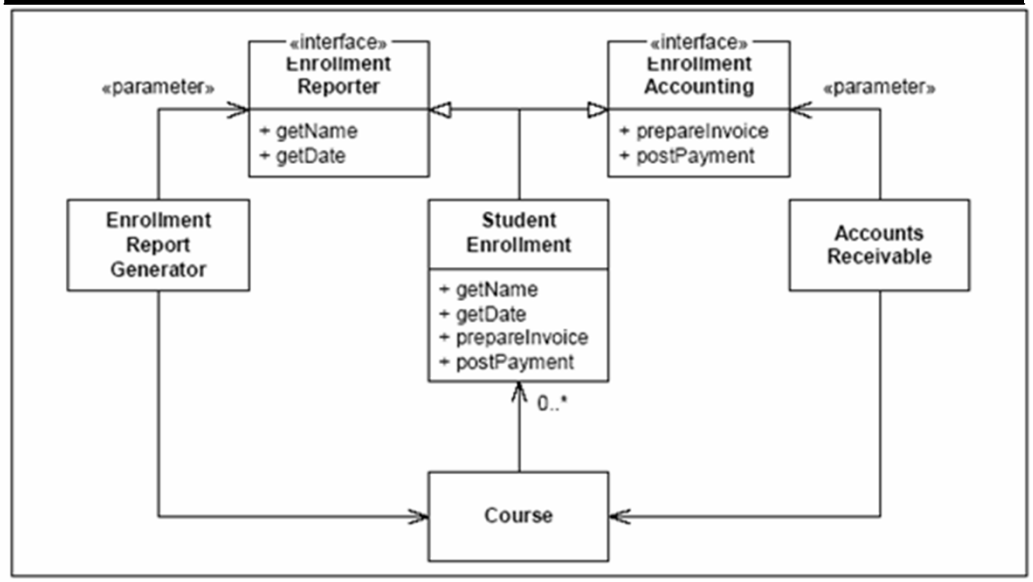
ISP implemented



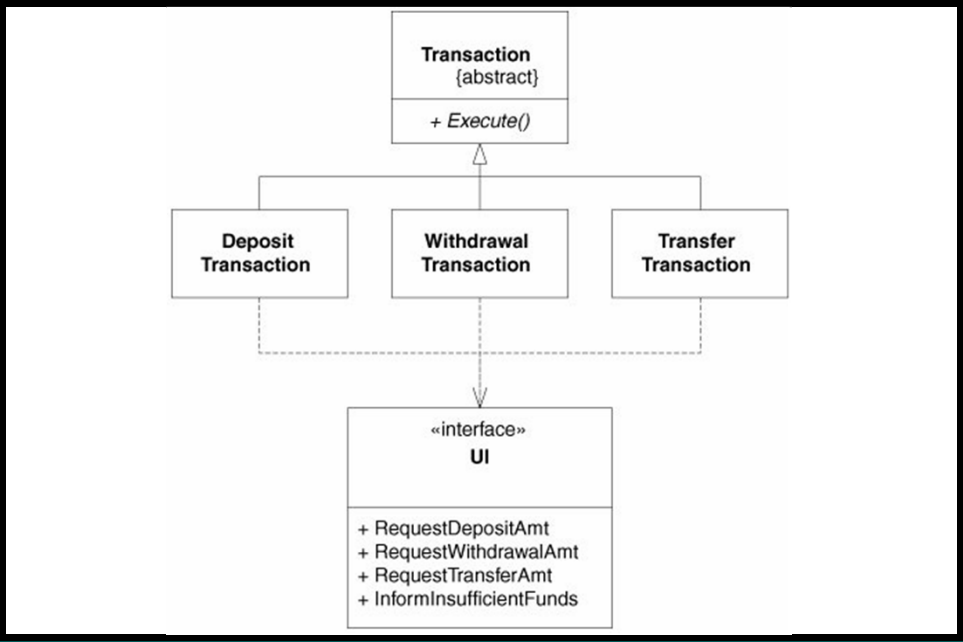
ISP violated



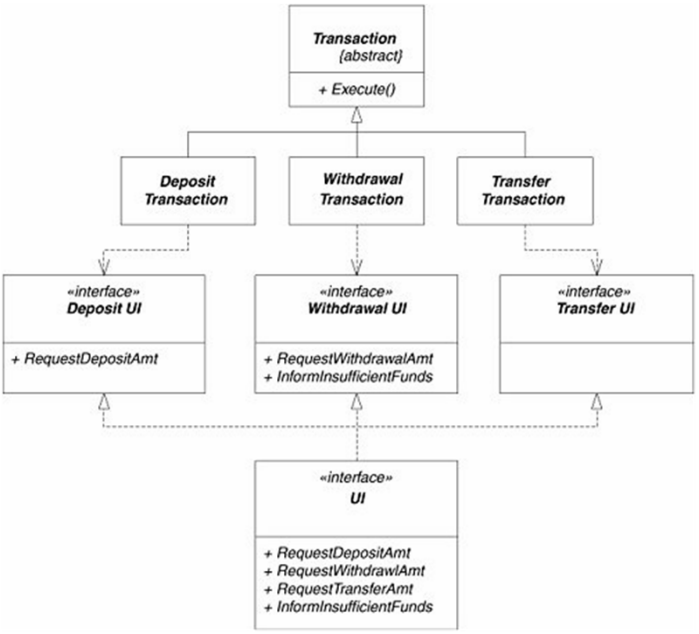
ISP implemented

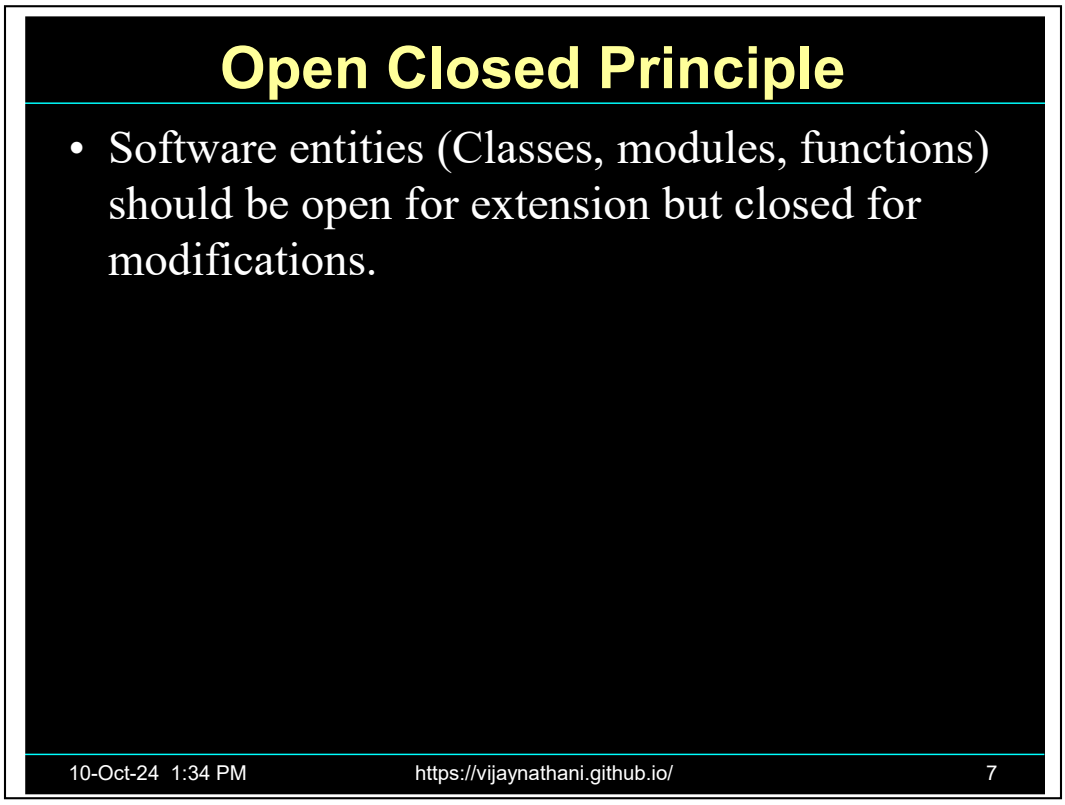


ISP violated



ISP implemented





It should be possible to change the environment of a class without changing the class.

A class should be closed for modification, but open for extension. When you change a class, there is always a risk that you will break something. But if instead of modifying the class you extend it with a sub-class, that's a less risky change.

Guidelines

- Keep things that vary separately from things that are common.

Q45 – LoanHandler

Q44 – FILE1, DATABASE1

Q39 - Scheduler

Q83 – Cooker

Q84 – ChooseFontDialog

Dependency Inversion Principle

- Program to an interface and not to an implementation
  - Any problem?

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

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

No variable should hold a reference to a concrete class.

No class should derive from a concrete class.

No method should override an implemented method of any of its base classes.

It is OK to depend on stable classes like String, Integer, JPanel, etc.

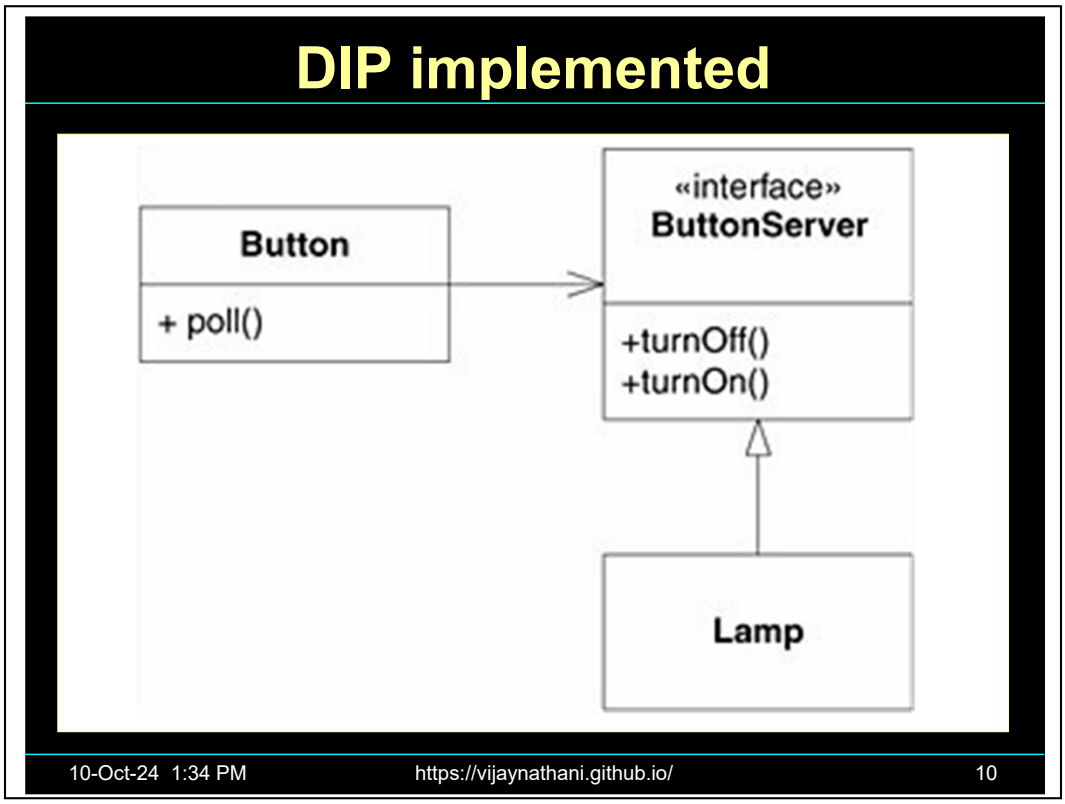
Avoids designs that are rigid, Fragile and Immobile.

Example: We have a classes: Copy, Keyboard, Printer. The Copy reads from keyboard and prints to the printer. If we use DIP, we have an abstraction for Keyboard and Printer. So we can add input and output devices later on.

Example: Collections in Java.

Example: Customer is a class. Employee is a class. Employees are now allowed to purchase on credit. We need an interface Buyer.

For every variable use the maximum abstract type possible.



Advantages

- Clients are unaware of the specific class of the object they are using
- One object can be easily replaced by another
- Object connections need not be hardwired to an object of a specific class, thereby increasing flexibility
- Loosens coupling
- Increases likelihood of reuse
- Improves opportunities for composition since contained objects can be of any class that implements a specific interface

Disadvantages

- Modest increase in design complexity