CS 525 - ASD

Advanced Software Development

MS.CS Program

Department of Computer Science

Rene de Jong, MsC.

Maharishi University
OF MANAGEMENT

CS 525 - ASD

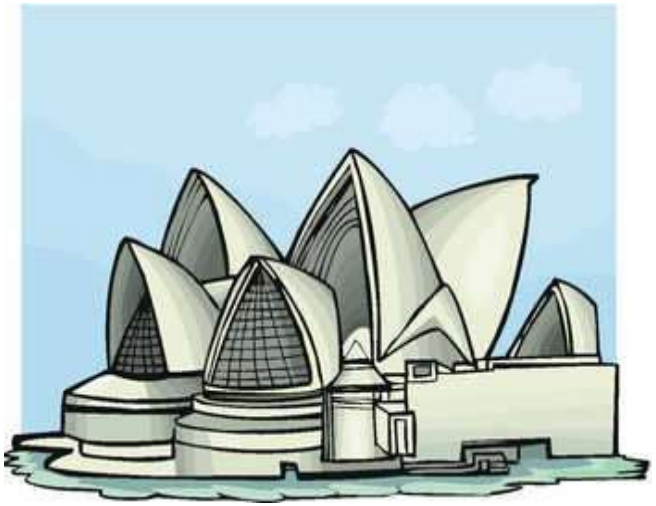
Advanced Software Development

Maharishi University
OF MANAGEMENT

Design principles

- Keep it simple
- Keep it flexible
- Loose coupling
- Separation of concern
- Information hiding
- Principle of modularity
- DRY: Don’t repeat yourself
- Encapsulate what varies
- Solid
  - **S**ingle Responsibility Principle (SRP)
  - **O**pen-Closed Principle (OCP)
  - **L**iskov Substitution Principle (LSP)
  - **I**nterface Segregation Principle (ISP)
  - **D**ependency Inversion Principle (DIP)

Keep it simple

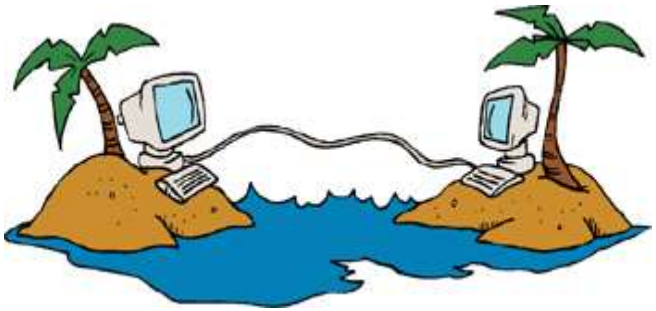


Keep it flexible

- Everthing changes
  - Business
  - Technical
- More flexibility leads to more complexity

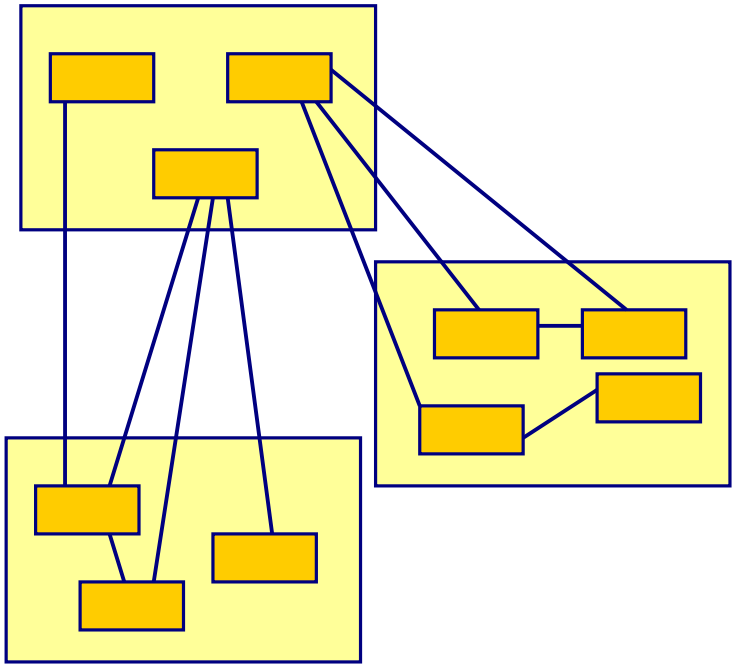
Loose coupling

- Different levels of coupling
  - Technology
  - Time
  - Location
  - Data structure
- You need coupling somewhere
  - Important is the level of coupling

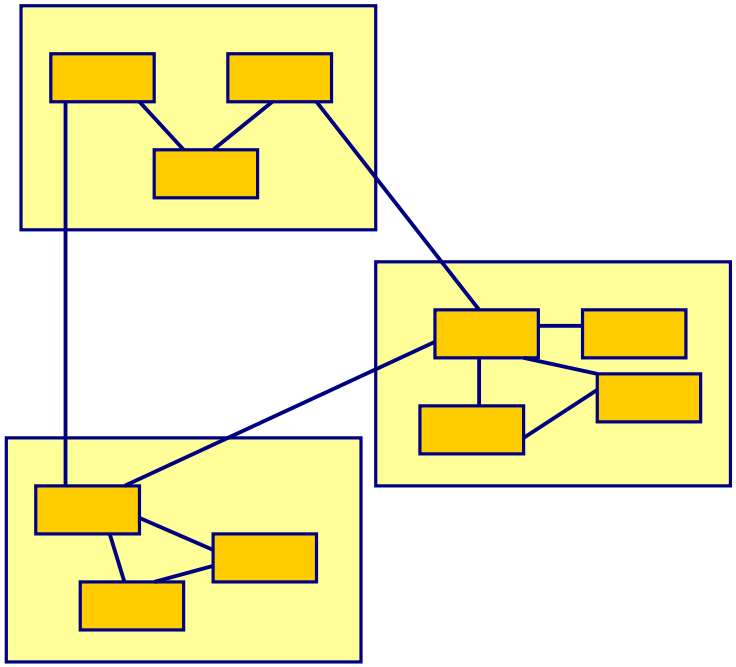


High cohesion, low coupling

- High coupling, low cohesion



- High cohesion, low coupling



Separation of concern

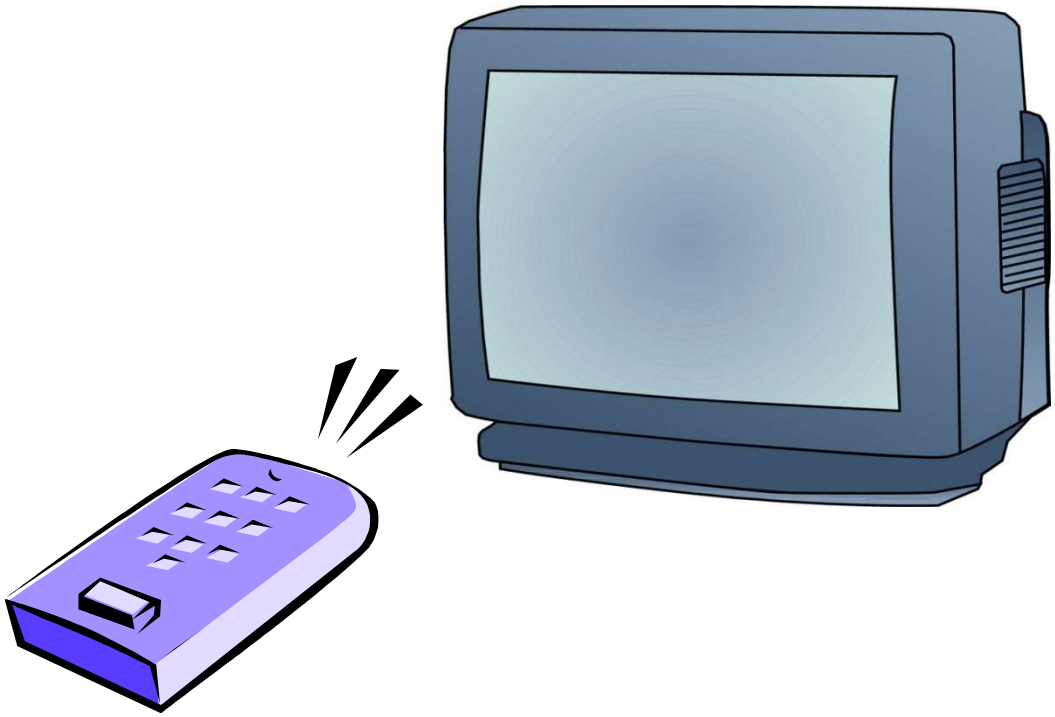
- Separate technology from business
- Separate stable things from changing things
- Separate things that need separate skills
- Separate business process from application logic
- Separate implementation from specification

Information hiding

- Black box principle
- Hide implementation behind an interface

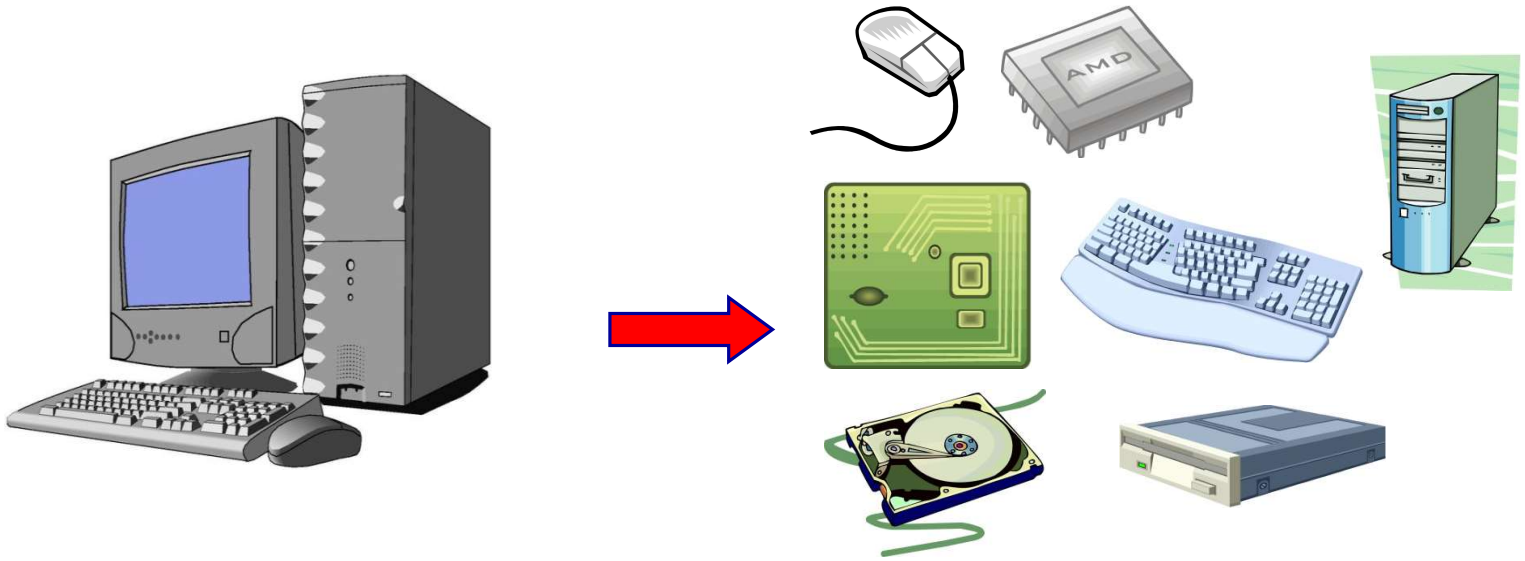
Program to an interface, not an implementation.

- Client app is decoupled from knowing the details of the concrete implementation.



Principle van modularity

- Decomposition
- Devide a big complex problem is smaller parts
- Use components that are
  - Better understandable
  - Independent
  - Reusable
- Leads to more flexibility
- Makes finding and solvings bugs easier

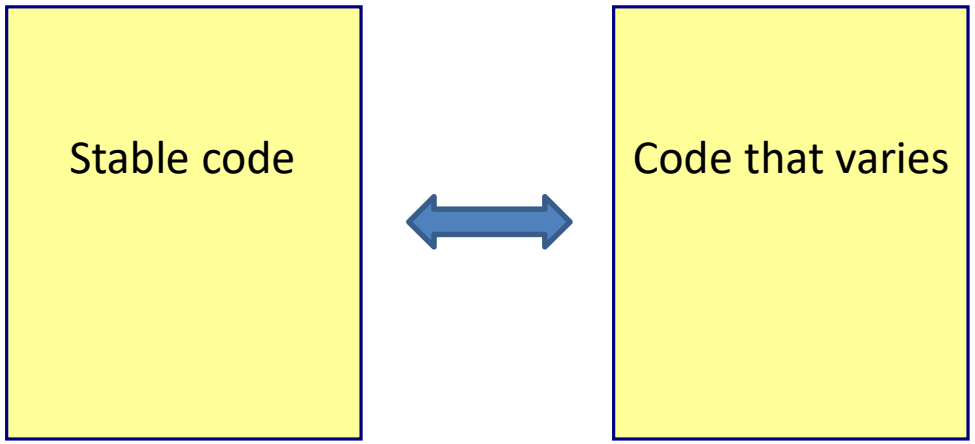


DRY: Don’t Repeat Yourself

- Write functionality at one place, and only at one place
- Avoid code scattering

Encapsulate what varies

- Take the parts that vary and encapsulate them, so that later you can alter or extend the parts that vary without affecting the parts that don’t.

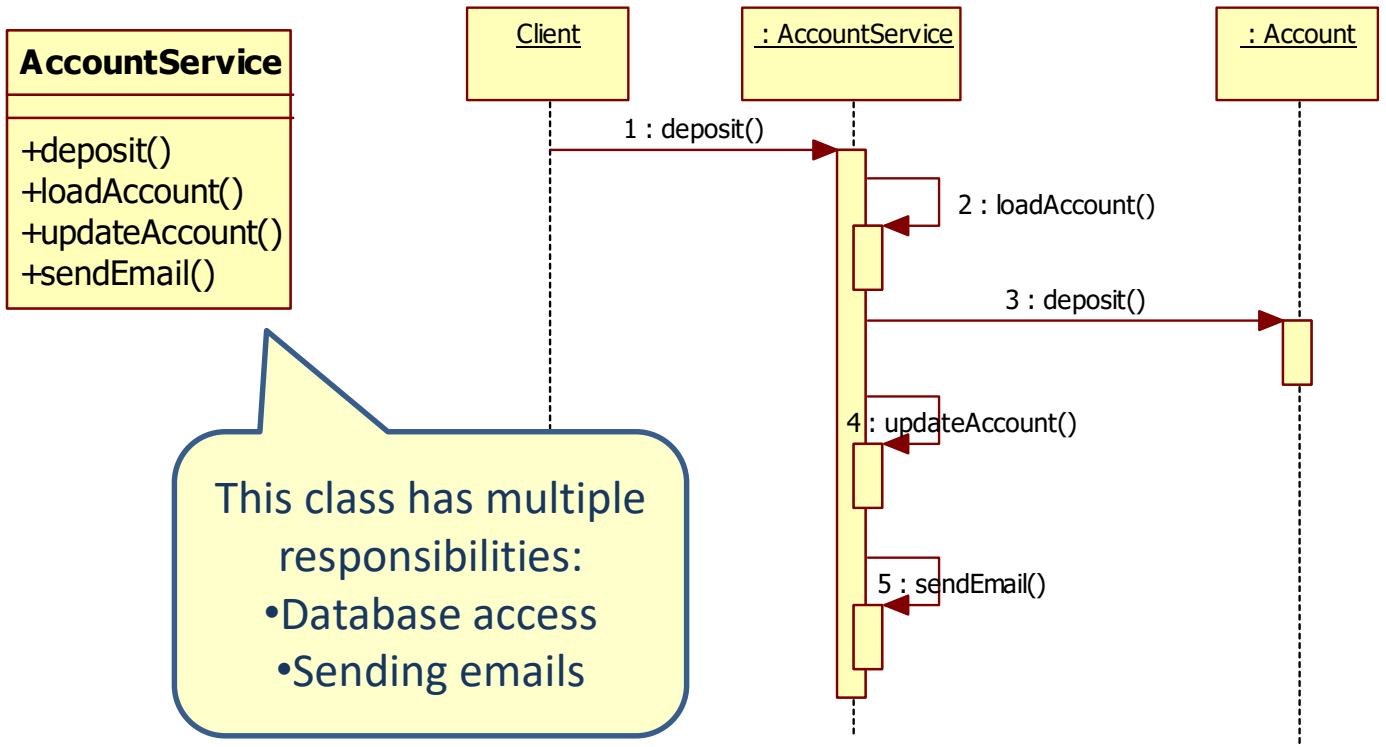


SOLID

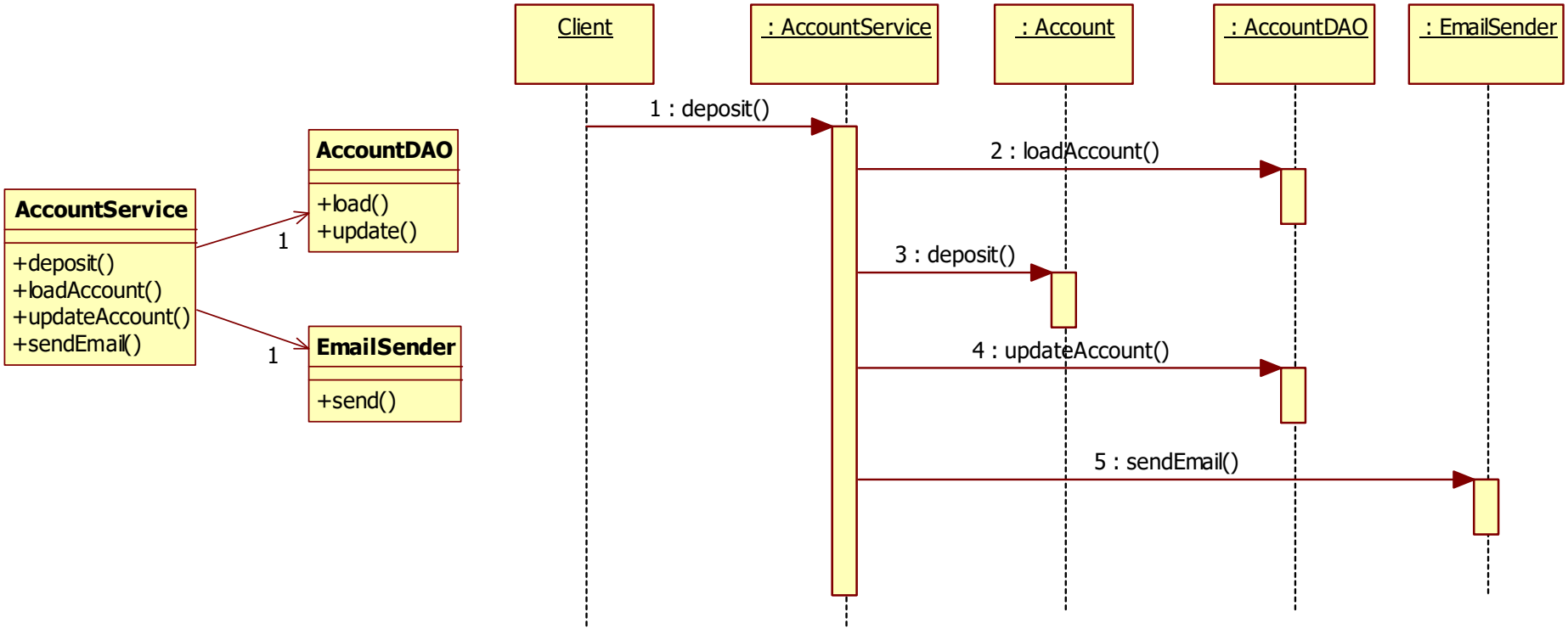
- **S**ingle Responsibility Principle (SRP)
- **O**pen-Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)

Single Responsibility Principle

- A class has only one responsibility
  - There should never be more than one reason for a class to change.

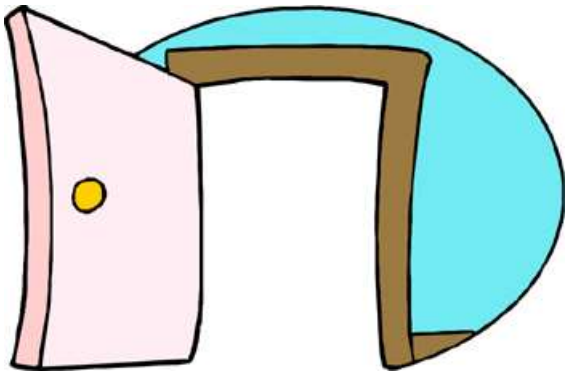


Single Responsibility Principle



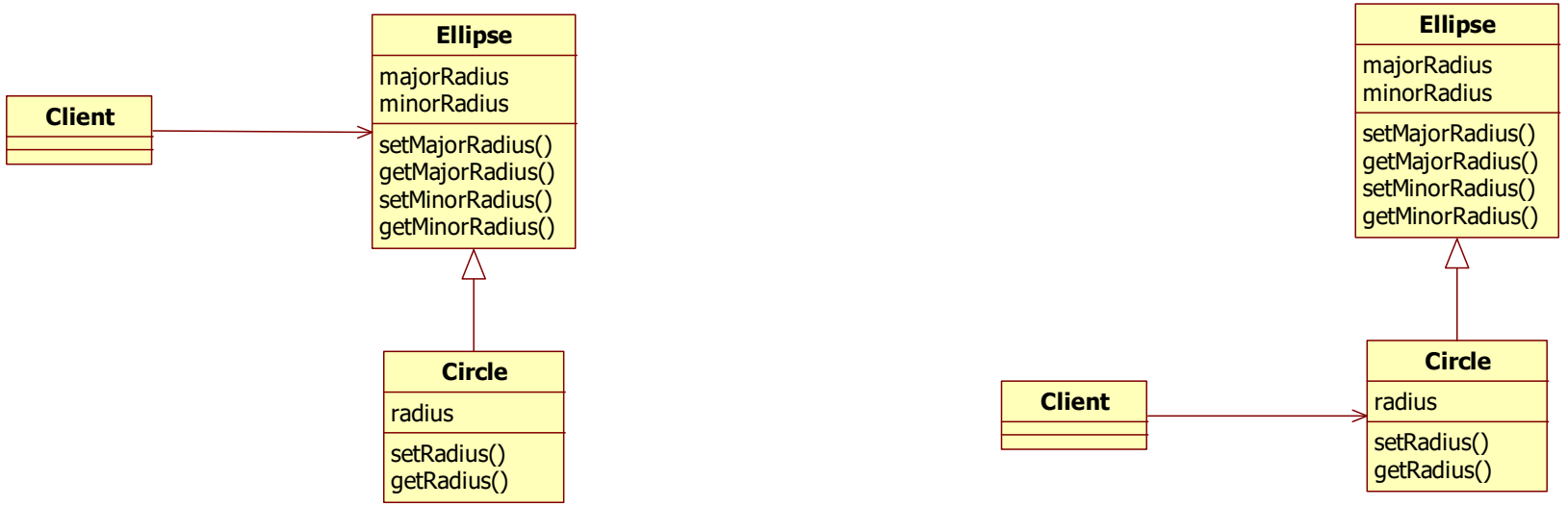
Open-closed principle (OCP)

- Your design should be open for extension, but closed for change
  - We want to add new code as much as possible, and we want to avoid changing working, and tested code

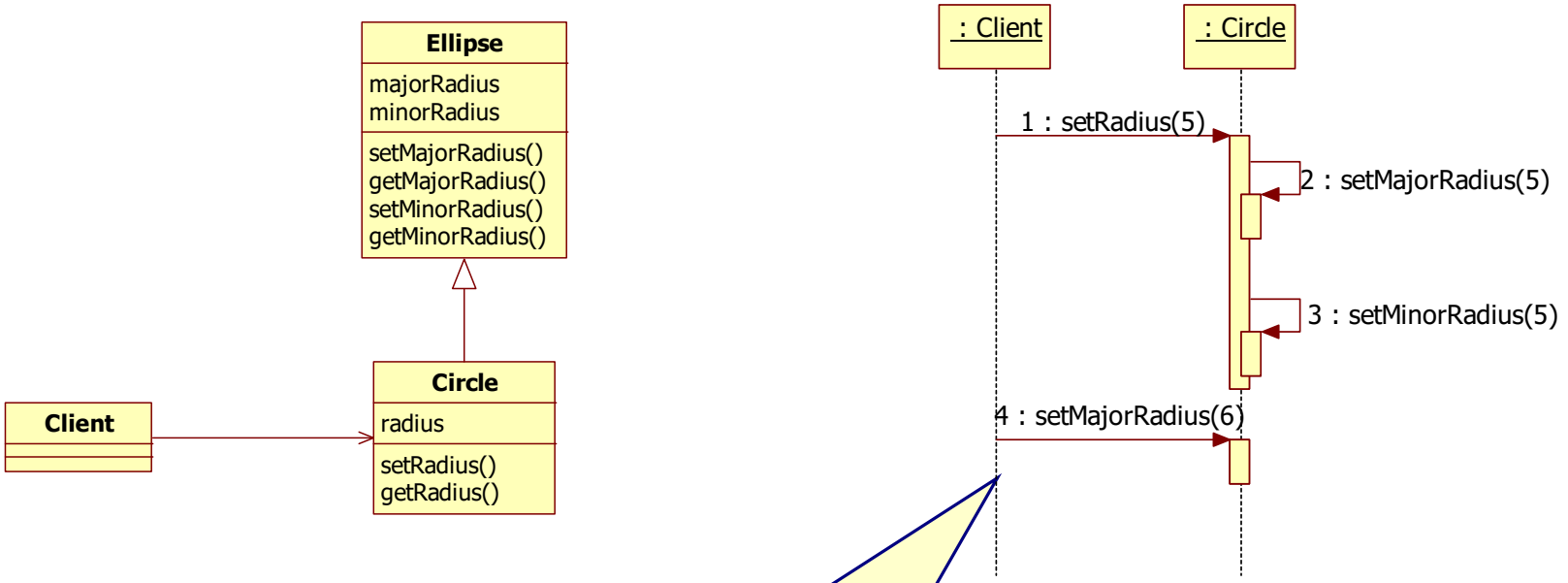


Liskov Substitution Principle

- It should always be possible to substitute a base class for a derived class without any change in behavior.



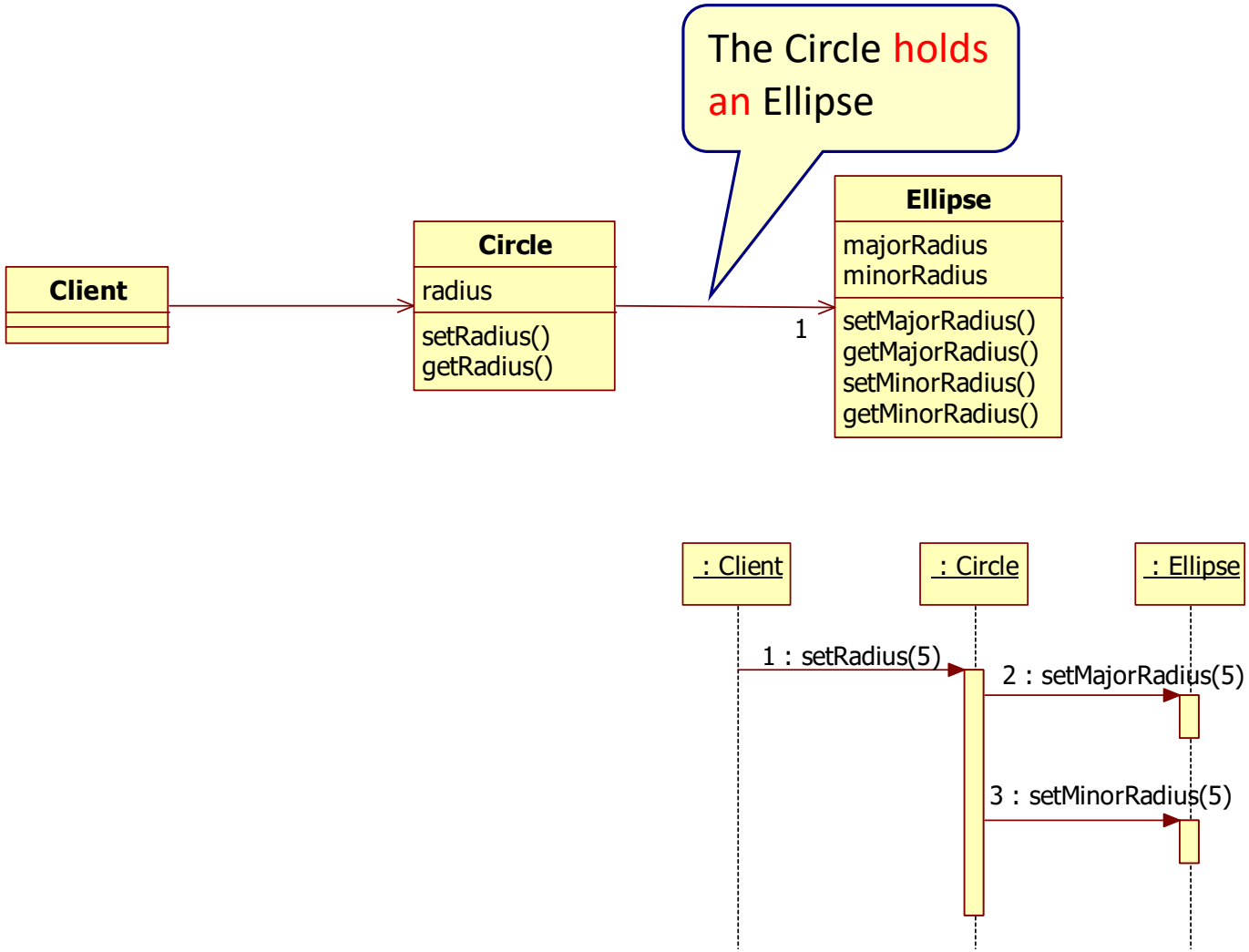
Liskov Substitution Principle Example



The Circle also inherits the methods SetMajorRadius() and SetMinorRadius().

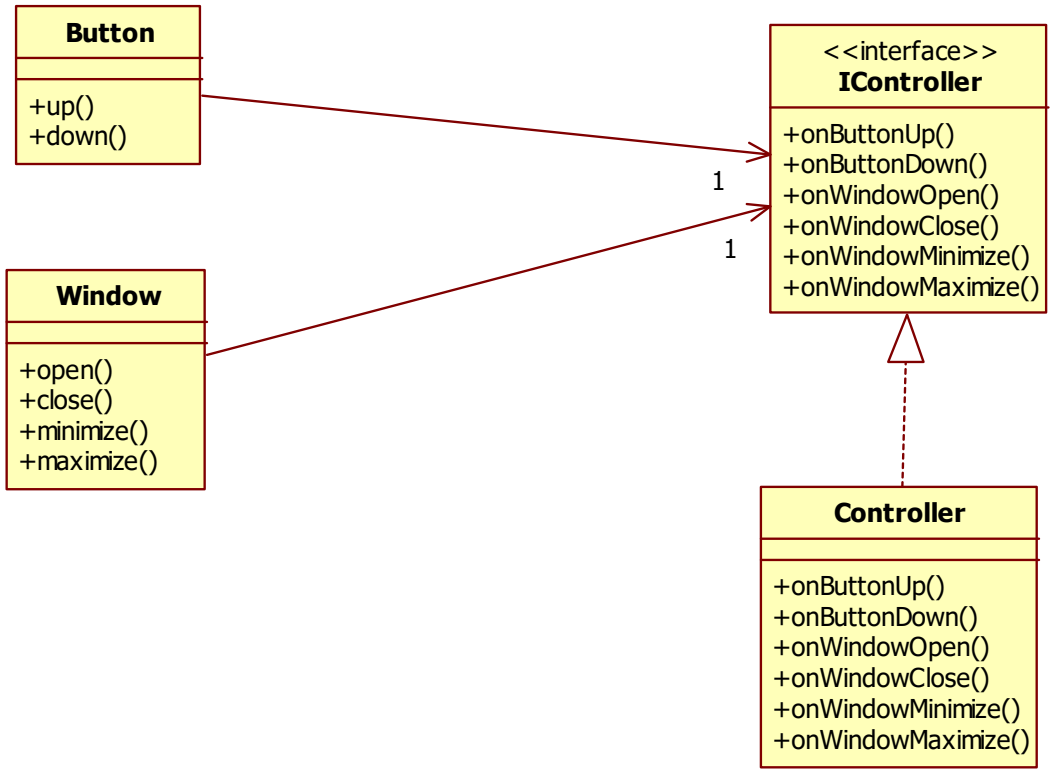
You cannot replace an Ellipse with a Circle!

Solution: composition

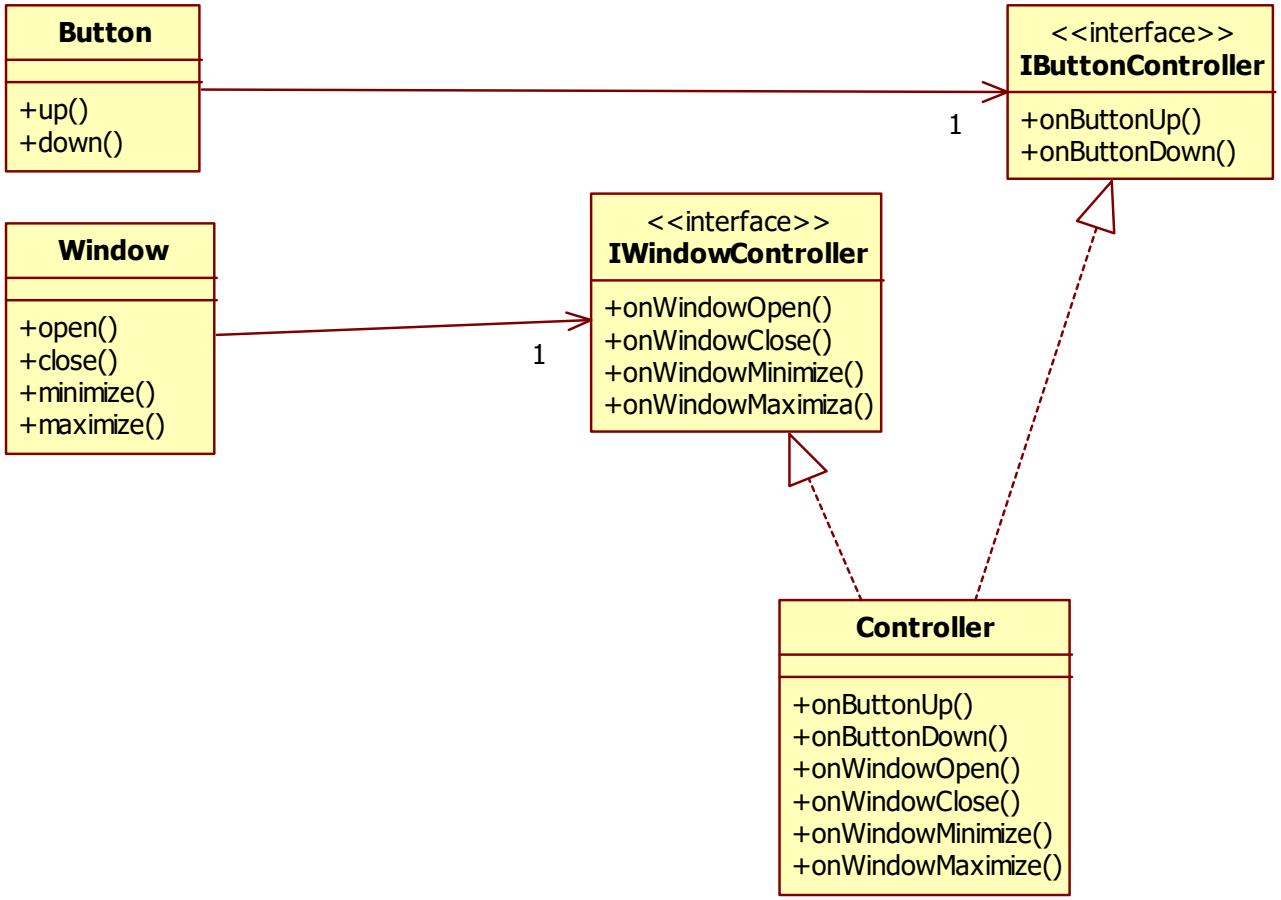


Interface Segregation Principle (ISP)

- Clients should not be forced to depend on methods they do not use

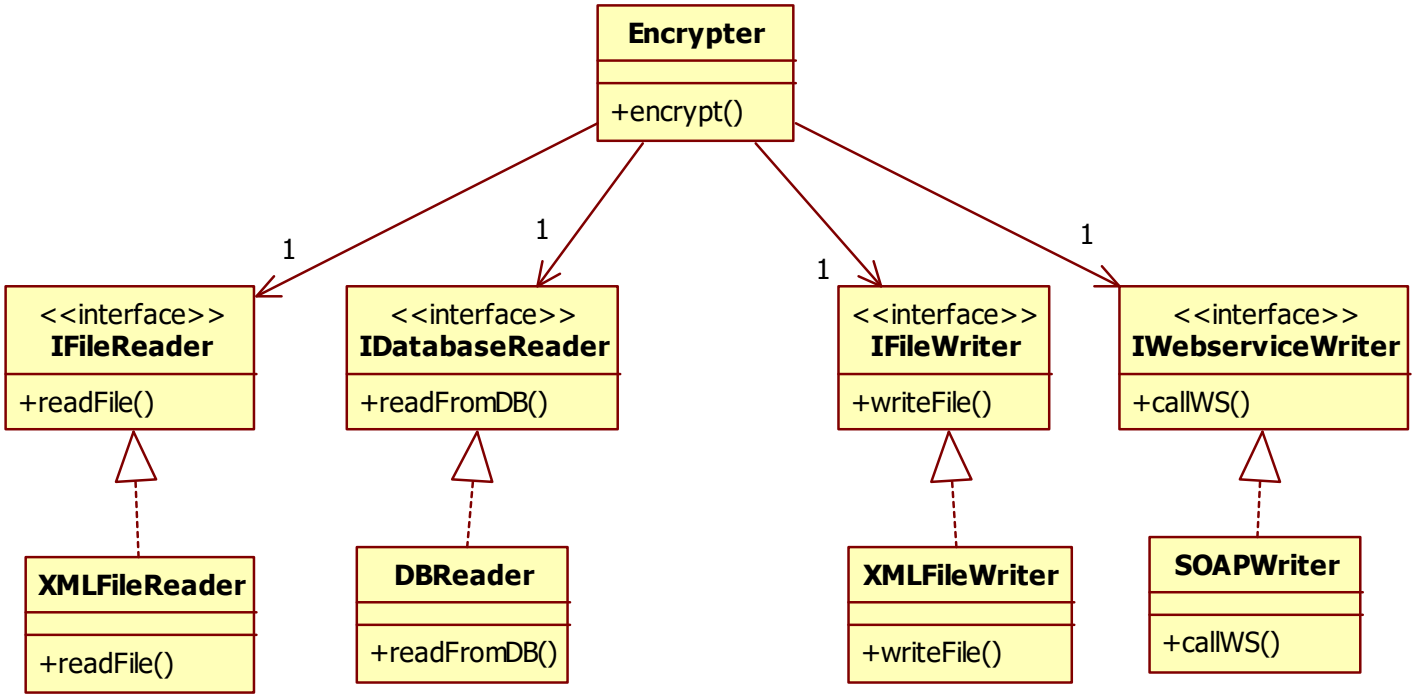


Interface Segregation Principle (ISP)

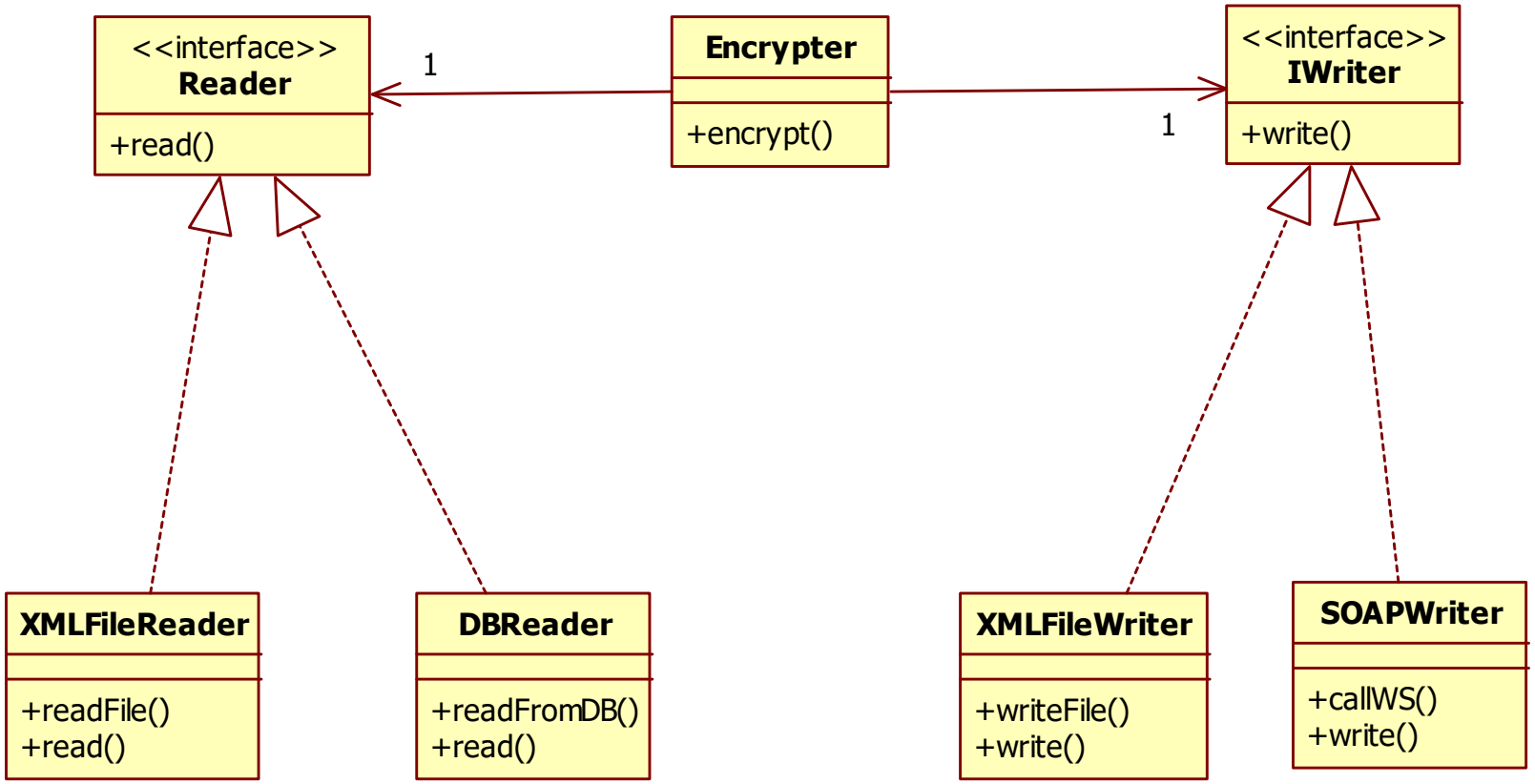


Dependency Inversion Principle (DIP)

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



Dependency Inversion Principle (DIP)



Design principles

- Keep it simple
- Keep it flexible
- Loose coupling
- Separation of concern
- Information hiding
- Principle of modularity
- DRY: Don’t repeat yourself
- Encapsulate what varies
- Solid
  - **S**ingle Responsibility Principle (SRP)
  - **O**pen-Closed Principle (OCP)
  - **L**iskov Substitution Principle (LSP)
  - **I**nterface Segregation Principle (ISP)
  - **D**ependency Inversion Principle (DIP)

Main point

- A good designed system is often simple and easy to modify.
- The unified field is the underlying field at the basis of all relative creation.