CSE 4513

Lec – 6

Software Design

3.Assume, the velocity of your team is 100 Story points. You have 20 user stories (US1-US20) in your project backlog. You have estimated the user stories to have a difficulty/complexity expressed in story points as follows:

- Each of US1 to US5 equals 2 story points - Each of US6 to US10 equals 3 story points
- Each of US11 to US15 equals 6 story points - Each of US16 to US20 equals 14 story points

a) If you have a team of 4 developers and weekly sprints (1 week = 5 days = 40 hours), which user stories would you be able to implement in the next sprint and achieve the highest possible value without violating your capacity (effort) constraint?

b) How would your result change if the following needs to be implemented with highest priority?

- US6 must be implemented together with US11 - US7 must be implemented together with US16 - US8 must be implemented together with US17

Solution

=100

If you give me a program that **works** perfectly but is impossible to change, then it won’t work when the requirements change, and I won’t be able to make it work. Therefore the program will become useless.

If you give me a program that **does not work** but is easy to change, then I can make it work, and keep it working as requirements change. Therefore the program will remain continually useful.

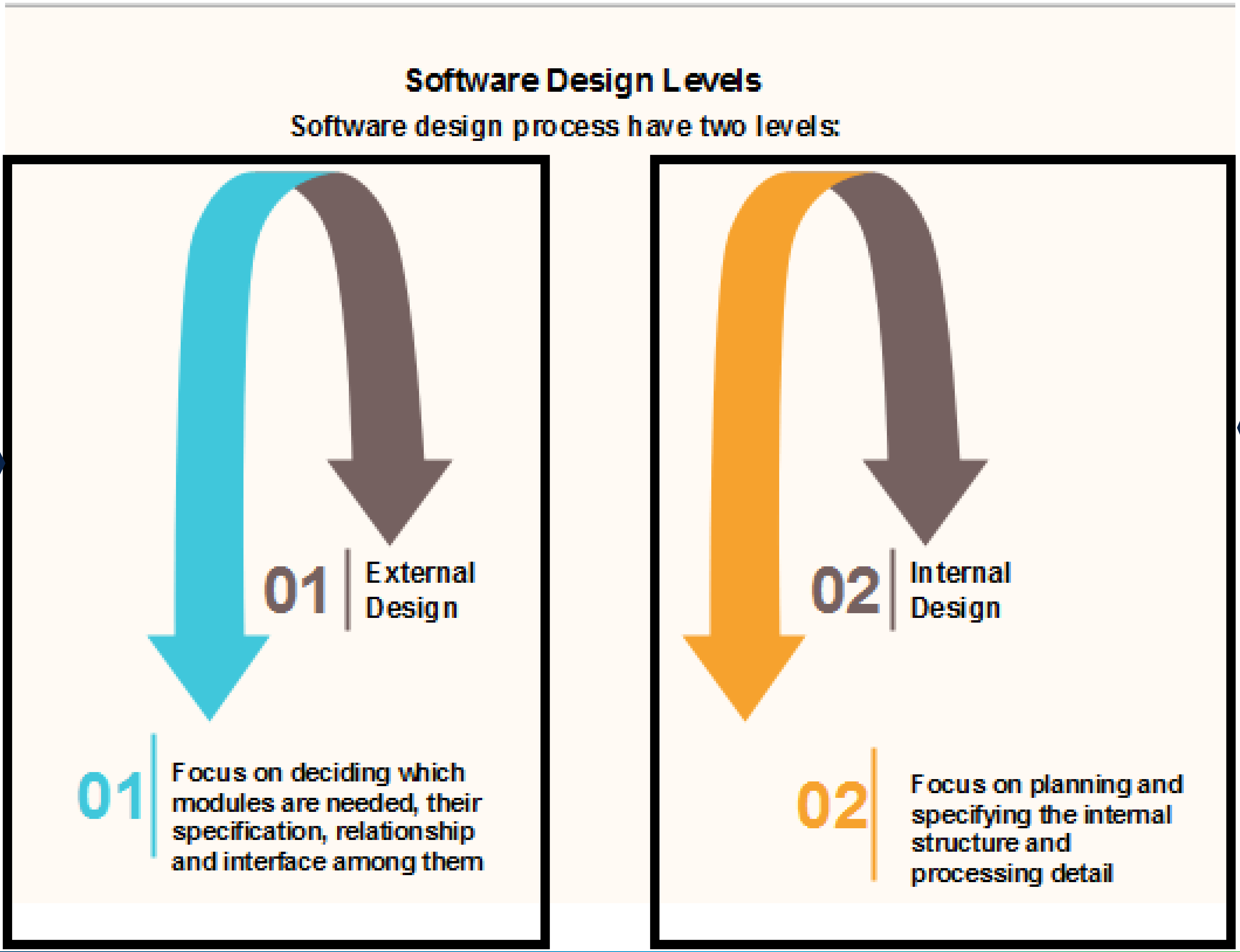
What is Software Design

is a mechanism to **transform user requirements** into some suitable form, which **helps the programmer in software coding and implementation**.

- ✓ It deals with representing the client's requirement, as described in SRS (Software Requirements Specification) document, into a form, i.e., easily implementable using programming language.
- ✓ moves the concentration from the problem domain to the solution domain.
- ✓ In software design, we consider the system to be a set of components or modules with clearly defined behaviors & boundaries.

SW Design Level

High Level Design



Software Design Levels

Software design process have two levels:

01 | External Design

01 | Focus on deciding which modules are needed, their specification, relationship and interface among them

02 | Internal Design

02 | Focus on planning and specifying the internal structure and processing detail

Detail Level Design

Objectives of SW Design

01

Correctness
should be correct as per requirement

02

Completeness
should have all components like data structures, modules, and external interfaces, etc.

03

Efficiency
Resources should be used efficiently by the program.

04

Flexibility
Able to modify on changing needs.

05

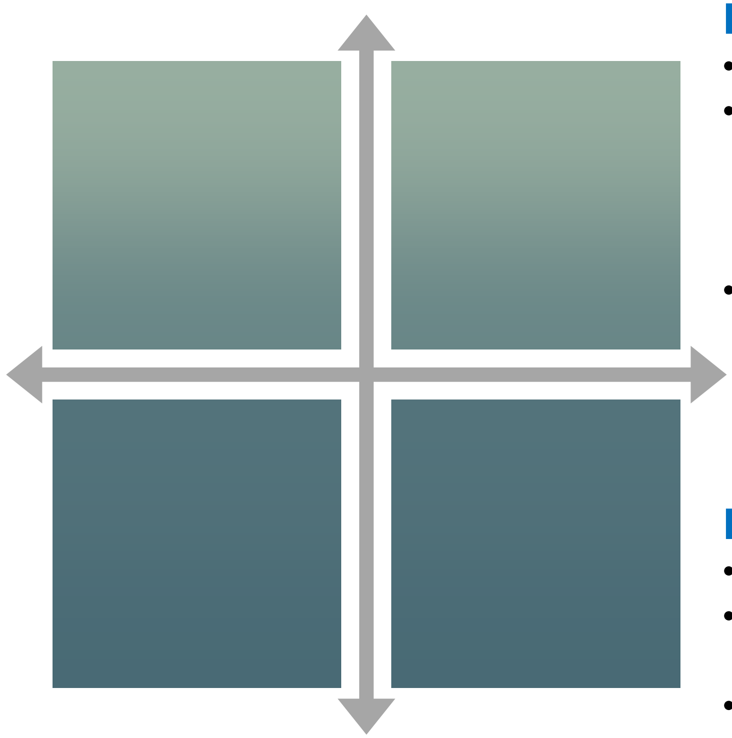
Consistency
There should not be any inconsistency in the design..

06

Maintainability
should be so simple so that it can be easily maintainable by other designers.

Software Design Principles

- ✓ Software design principles are concerned with providing means to handle the complexity of the design process effectively.
- ✓ Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.
- ✓ The key software design principles are:



SOLID
- Pls check next Slide

DRY
- Don't Repeat Yourself
- each small pieces of knowledge (code) may only occur exactly once in the entire system.
- This helps us to write scalable, maintainable and reusable code.

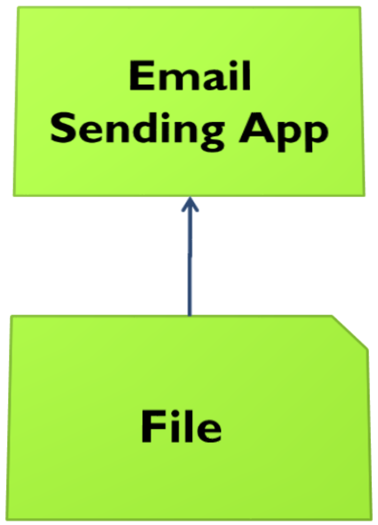
YAGNI
- You ain't gonna need it
- always implement things when you actually need them
- never implements things before you need them.

KISS
- Keep it simple, Stupid!
- keep each small piece of software simple
- unnecessary complexity should be avoided.

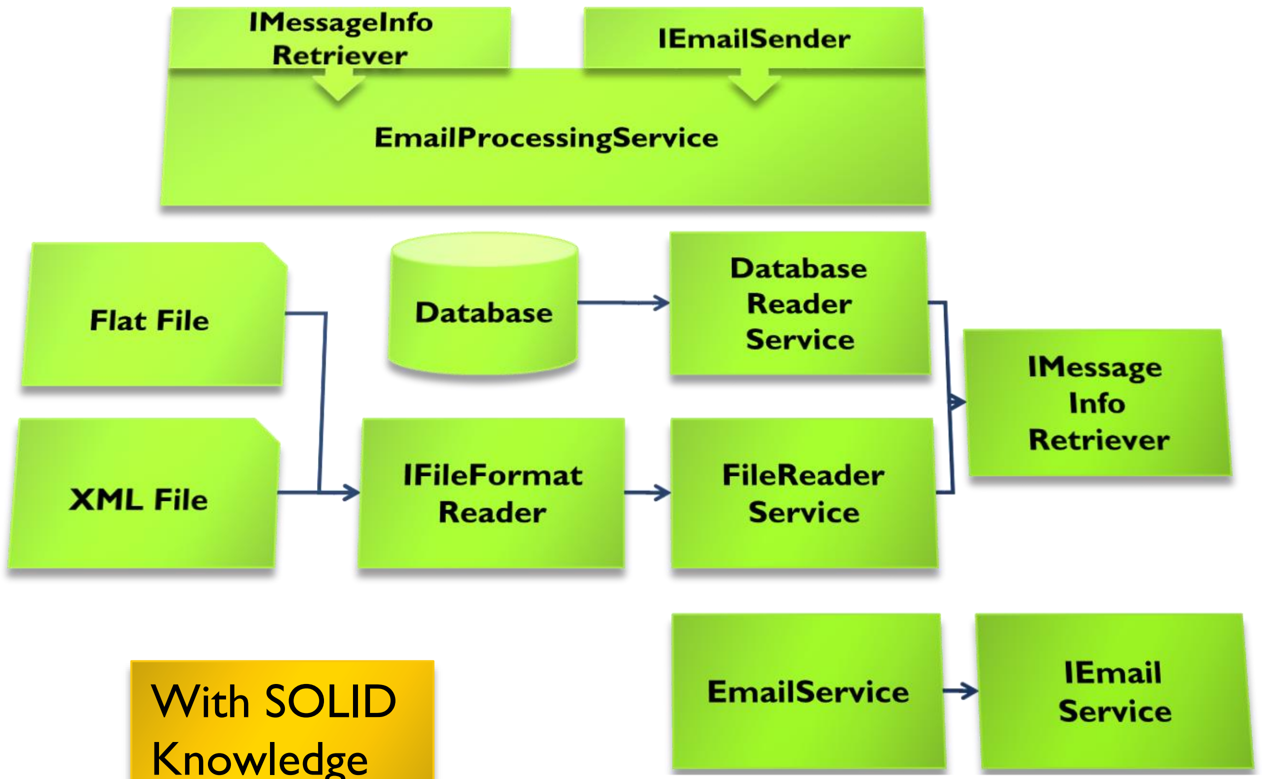
Software Design Principles - SOLID

- ✓ In Object Oriented Programming (OOP), SOLID is an acronym, introduced by Michael Feathers, for five design principles used to make software design more understandable, flexible, and maintainable.
- ✓ There are five SOLID principles:
  - ➢ Single Responsibility Principle (SRP)
  - ➢ Open Closed Principle (OCP)
  - ➢ Liskov Substitution Principle (LSP)
  - ➢ Interface Segregation Principle (ISP)
  - ➢ Dependency Inversion Principle (DIP)

WHY SOLID



Without SOLID Knowledge



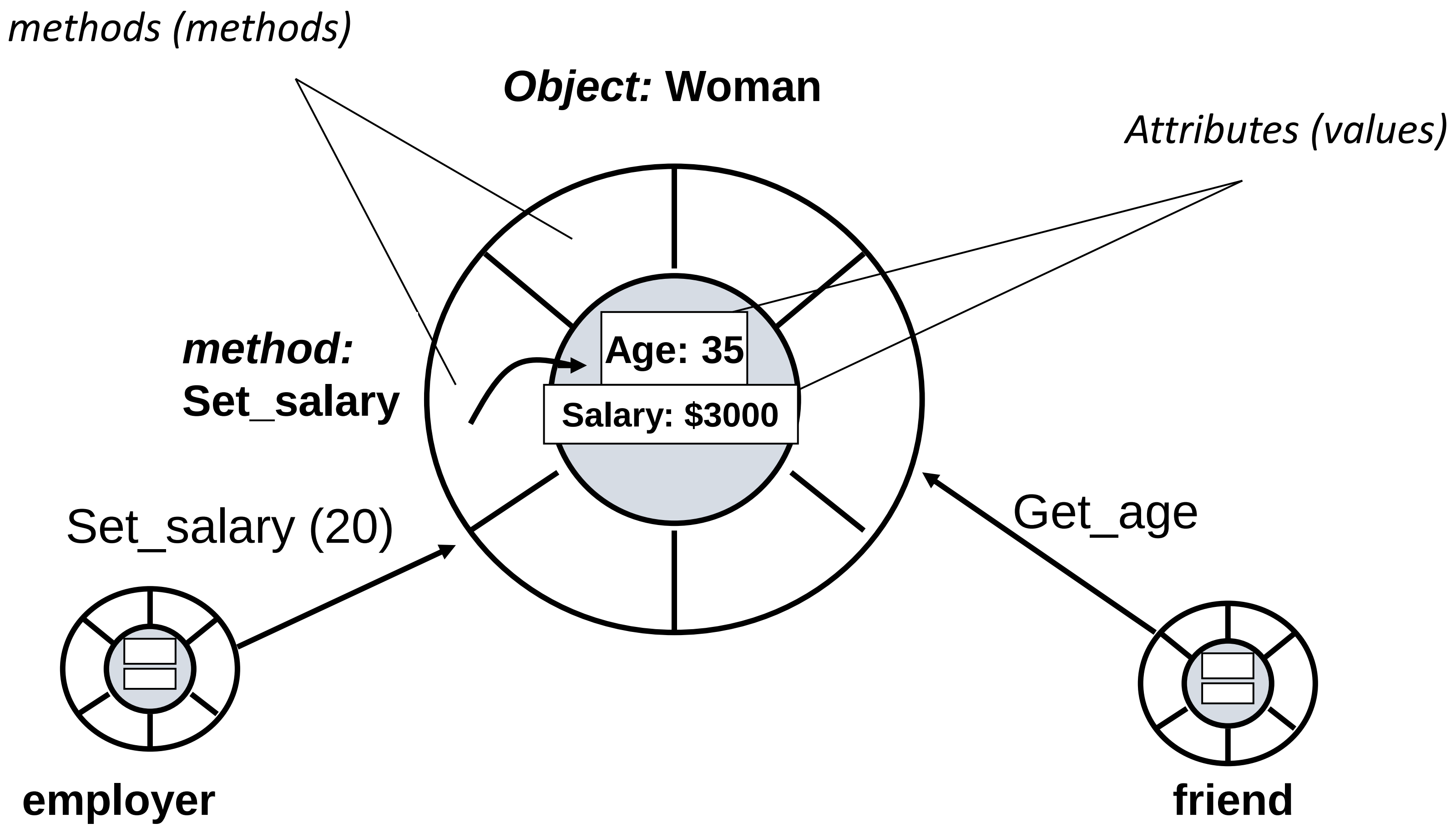
With SOLID Knowledge

Recap OO Concept

OBJECTS

- ✓ Object: Complex data type that has an **identity**, contains other data types called **attributes** and modules of code called **operations** or **methods**
- ✓ Attributes and associated values are **hidden** inside the object.
- ✓ Any object that wants to obtain or change a value associated with other object, must do so by sending a **message** to one of the objects (invoking a method)

Objects


methods (methods)
Object: Woman
Attributes (values)
method:
Set_salary
Age: 35
Salary: $3000
Set_salary (20)
Get_age
employer
friend

ENCAPSULATION

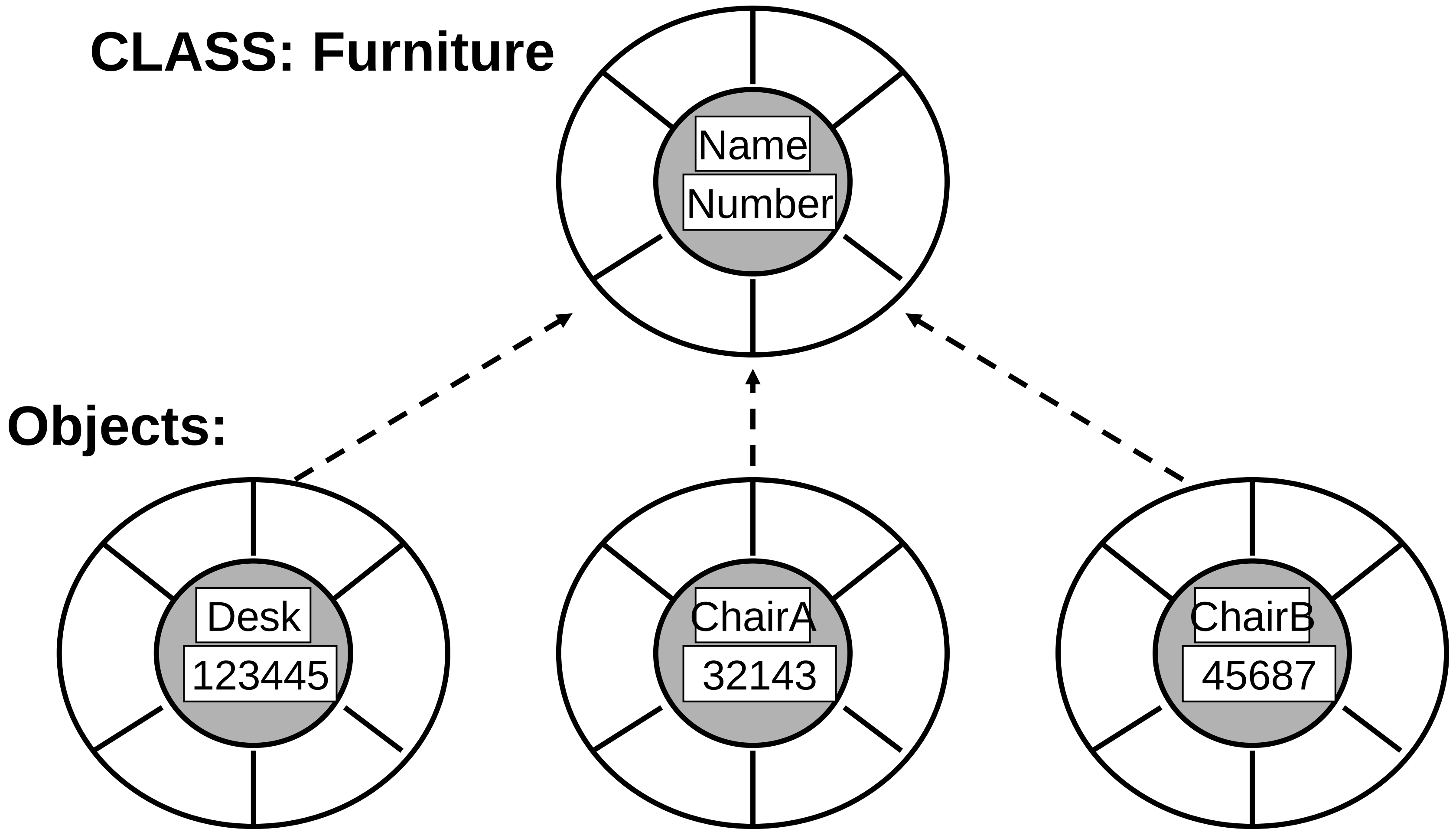
- Each objects methods manage it’s own attributes.
- This is also known as ***hiding.***
- An object **A** can learn about the values of attributes of another object **B**, only by invoking the corresponding method (message) associated to the object **B**.

- Example:
  - Class: Lady
  - Attributes: Age, salary
  - Methods: get_age, set_salary

Classes

- ✓Classes are **templates** that have methods and attribute names and type information, but no actual values!
- ✓Objects are generated by these classes and they actually contain values.
- ✓We design an application at the class level.
- ✓When the system is running objects are created by classes as they are needed to contain state information.
- ✓When objects are no longer needed by the application, they are eliminated.

CLASS & OBJECTS



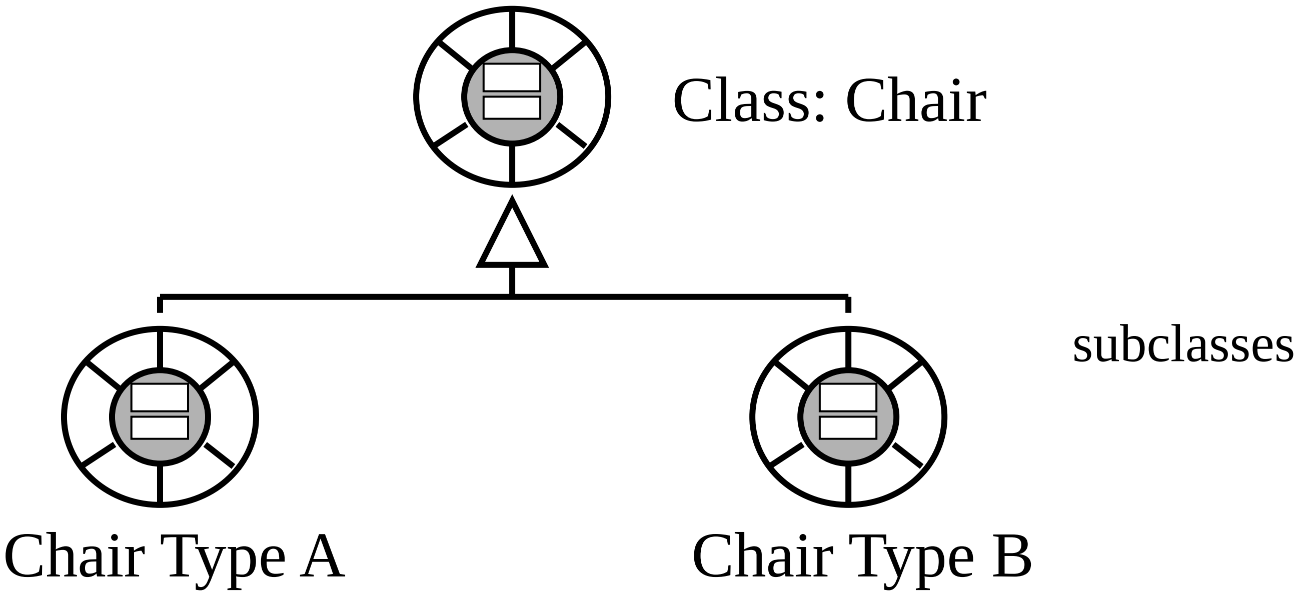
Message Passing & Associations

- ✓Methods are associated with classes but classes don’t send messages to each other.
- ✓Objects send messages.
- ✓A **static diagram (class diagram)** shows classes and the logical associations between classes, it doesn´t show the movement of messages.
- ✓An **association** between two classes means that the objects of the two classes can send messages to each other.
- ✓**Aggregation**: when an object contains other objects (a part-whole relationship)

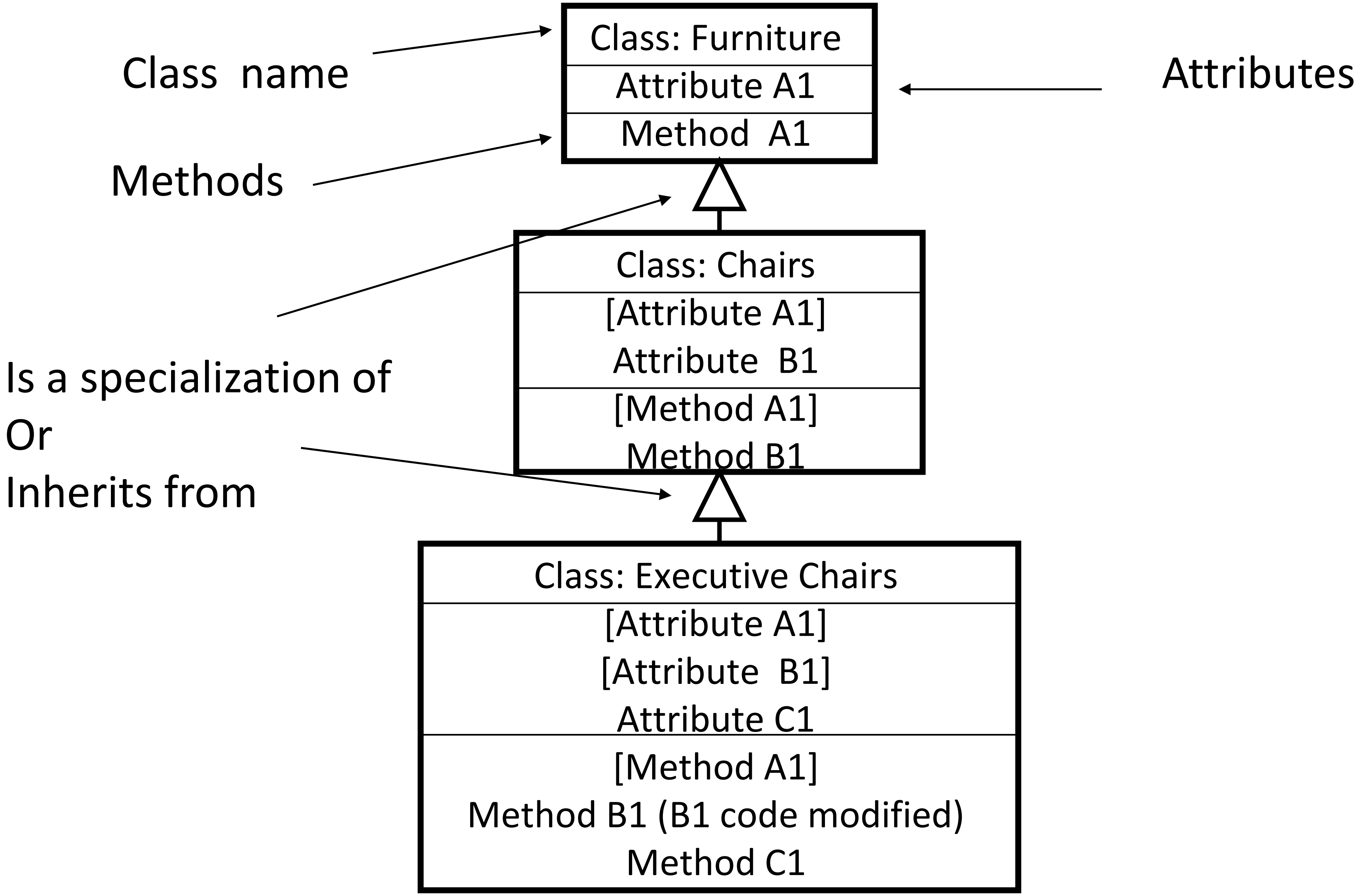
Class Hierarchies & Inheritance

✓Classes can be arranged in hierarchies so that more classes inherit attributes and methods from more abstract classes

Class hierarchy diagrams



CLASS INHERITANCE & SPECIALIZATION



PUBLIC, PRIVATE & PROTECTED

- ✓Attributes can be public or private:
  - ➢ Private: it can only be accessed by its own methods
  - ➢ Public: it can be modified by methods associated with any class (violates encapsulation)
- ✓Methods can be public, private or protected:
  - ➢ Public: it’s name is exposed to other objects.
  - ➢ Private: it can’t be accessed by other objects, only internally
  - ➢ Protected: (special case) only subclasses that descend directly from a class that contains it, know and can use this method.

POLIMORPHISM

- ✓Means that the same method will behave differently when it is applied to the objects of different classes
- ✓It also means that different methods associated with different classes can interpret the same message in different ways.
- ✓Example: an object can send a message PRINT to several objects, and each one will use it’s own PRINT method to execute the message.

Next Class

SOLID Again