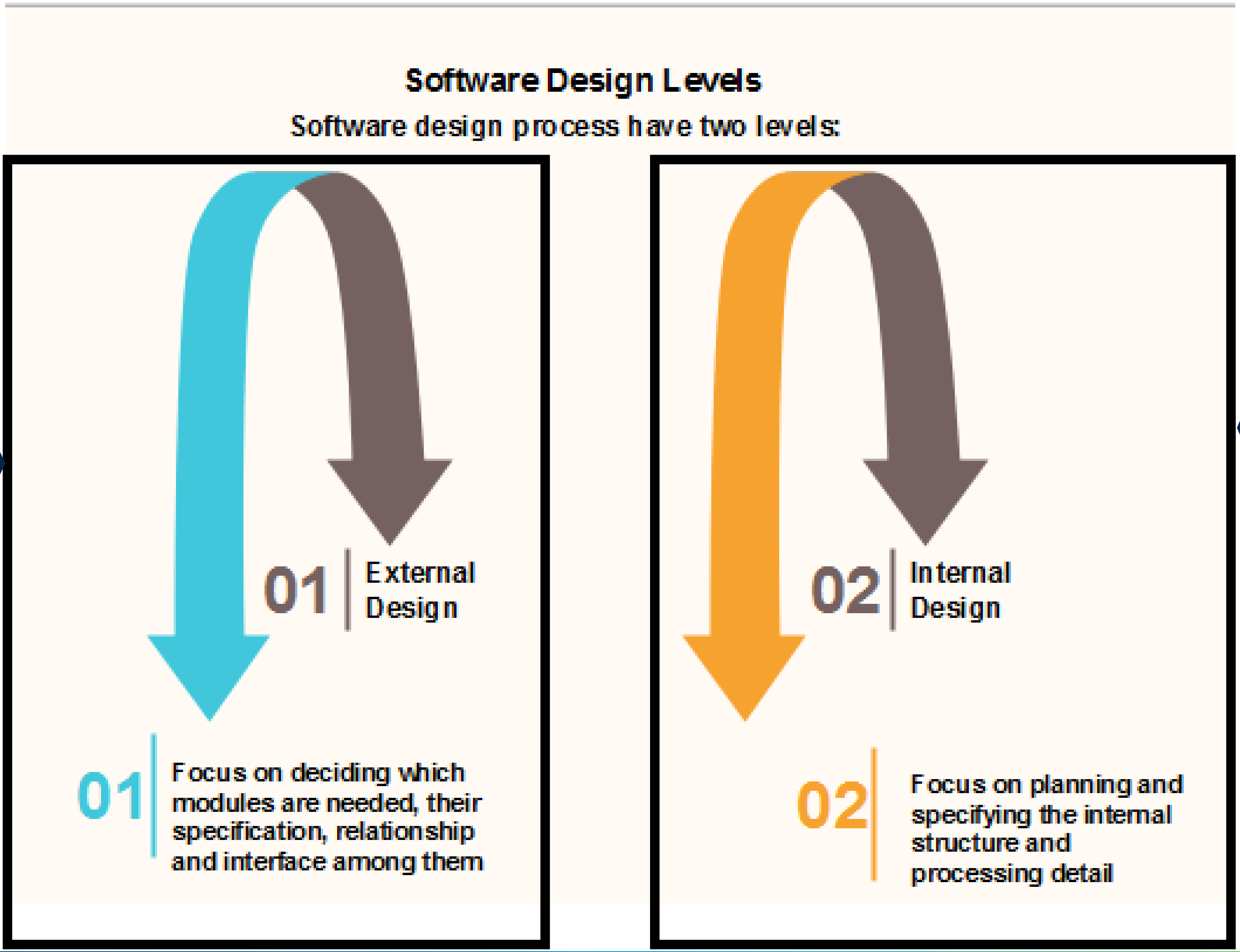
What is Software Design

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 Requirement 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



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

High Level Design

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

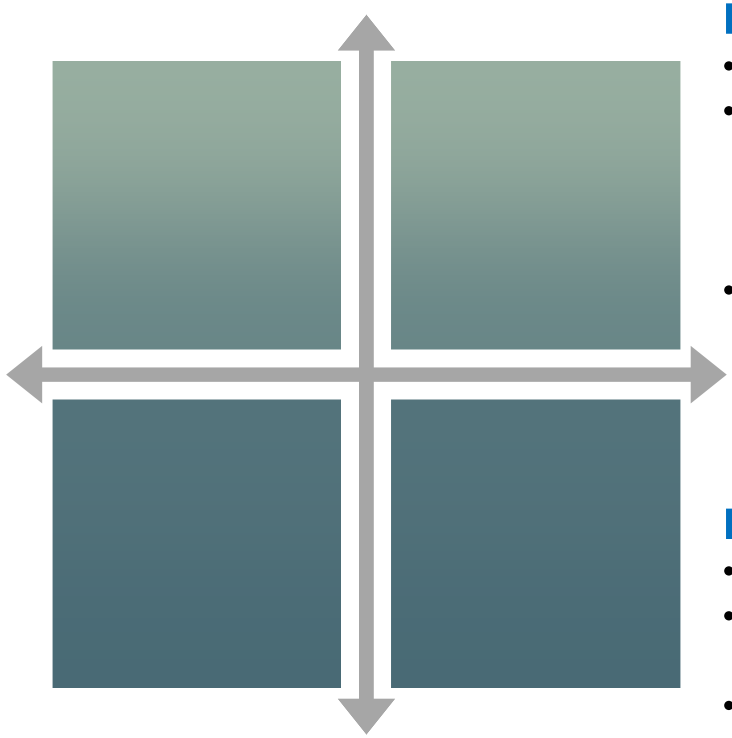
- ✓ Software design principles are concerned with providing means to handle the complexity of the design process effectively.
- ✓ Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.
- ✓ The key software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

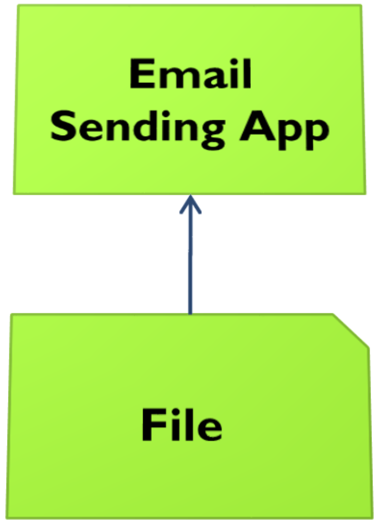
KISS

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

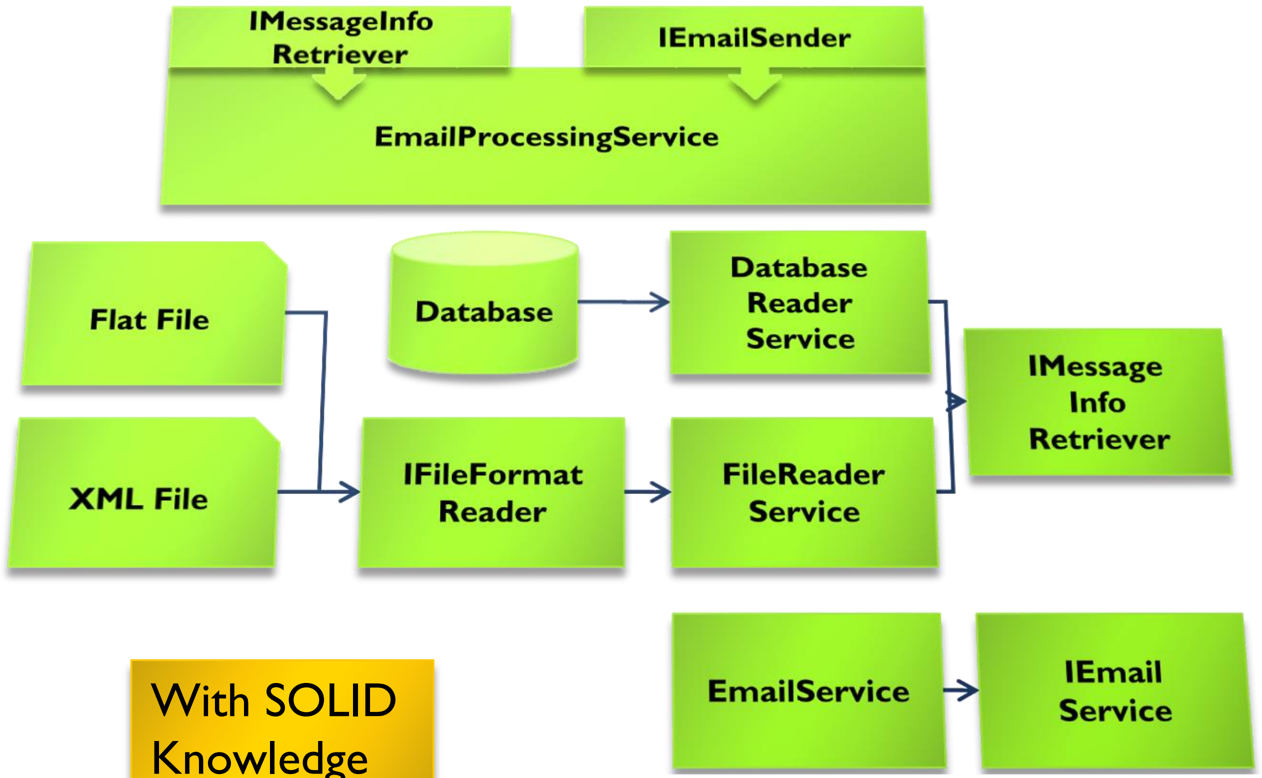
Software Design Principles - SOLID

- ✓ 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

Classes and Objects

- ✓ A **class** is a template for objects.
- ✓ A **class** defines object properties including a valid range of values, and a default value.
- ✓ A **class** also describes object behavior.
- ✓ An **object** is a member or an "instance" of a class.
- ✓ An **object** has a state in which all of its properties have values that you either explicitly define or that are defined by default settings
- ✓Objects are generated by the classes and they actually contain values.
- ✓We design an application at the class level.

ENCAPSULATION

- The ability to protect some components of the object from external entities ("private").
- Encapsulation is achieved when each object keeps its state **private**, inside a class.
- Other objects don’t have direct access to this state. Instead, they can only call a list of public functions — called methods.
- 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**.

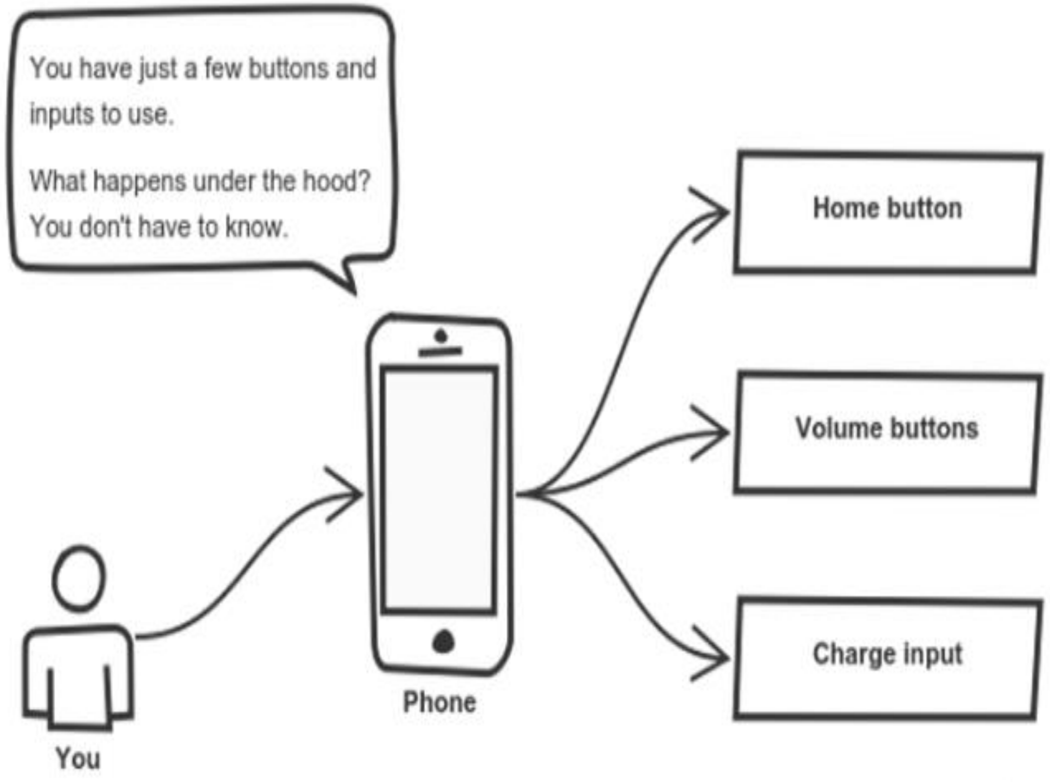
Message Passing & Associations

- ✓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.

ABSTRACTION

- ✓ Abstraction can be thought of as a natural extension of encapsulation.
- ✓ maintaining a large codebase for years — with changes along the way — is difficult.
- ✓ Abstraction is a concept aiming to ease this problem.
- ✓ Applying abstraction means that each object should **only** expose a high-level mechanism for using it.
- ✓ This mechanism should hide internal implementation details. It should only reveal operations relevant for the other objects.

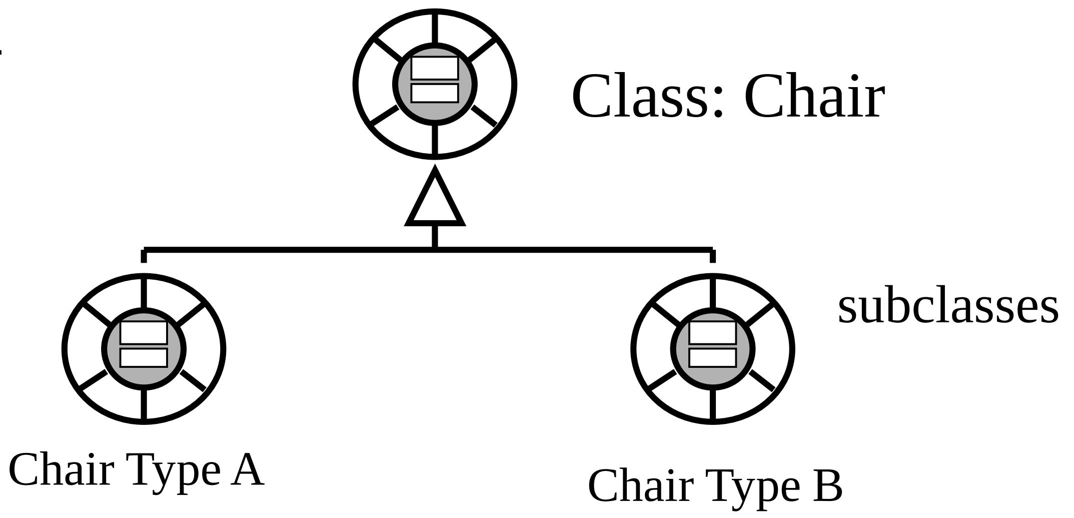
Think about your mobile again


You have just a few buttons and inputs to use.
What happens under the hood? You don't have to know.
Home button
Volume buttons
Charge input
Phone
You

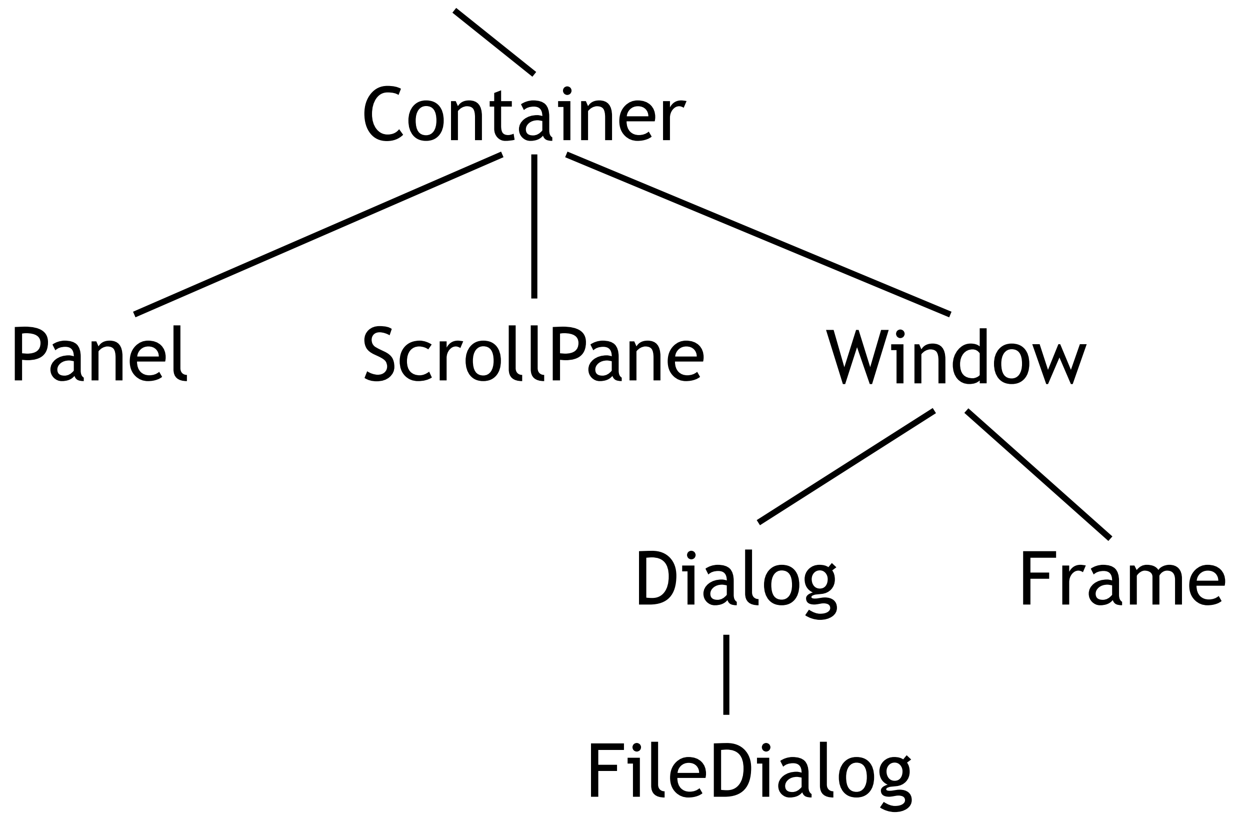
Class Hierarchies & Inheritance

- ✓ Objects are often very similar. They share common logic. But they’re not **entirely** the same
- ✓ Every class, except Object, has a superclass
- ✓ A class may have several ancestors, up to Object
- ✓ When you define a class, you specify its superclass
  - ✓ If you don’t specify a superclass, Object is assumed
- ✓ Every class may have one or more subclasses
- ✓ Inheritance is important since it leads to the reusability of code.

Class hierarchy diagrams

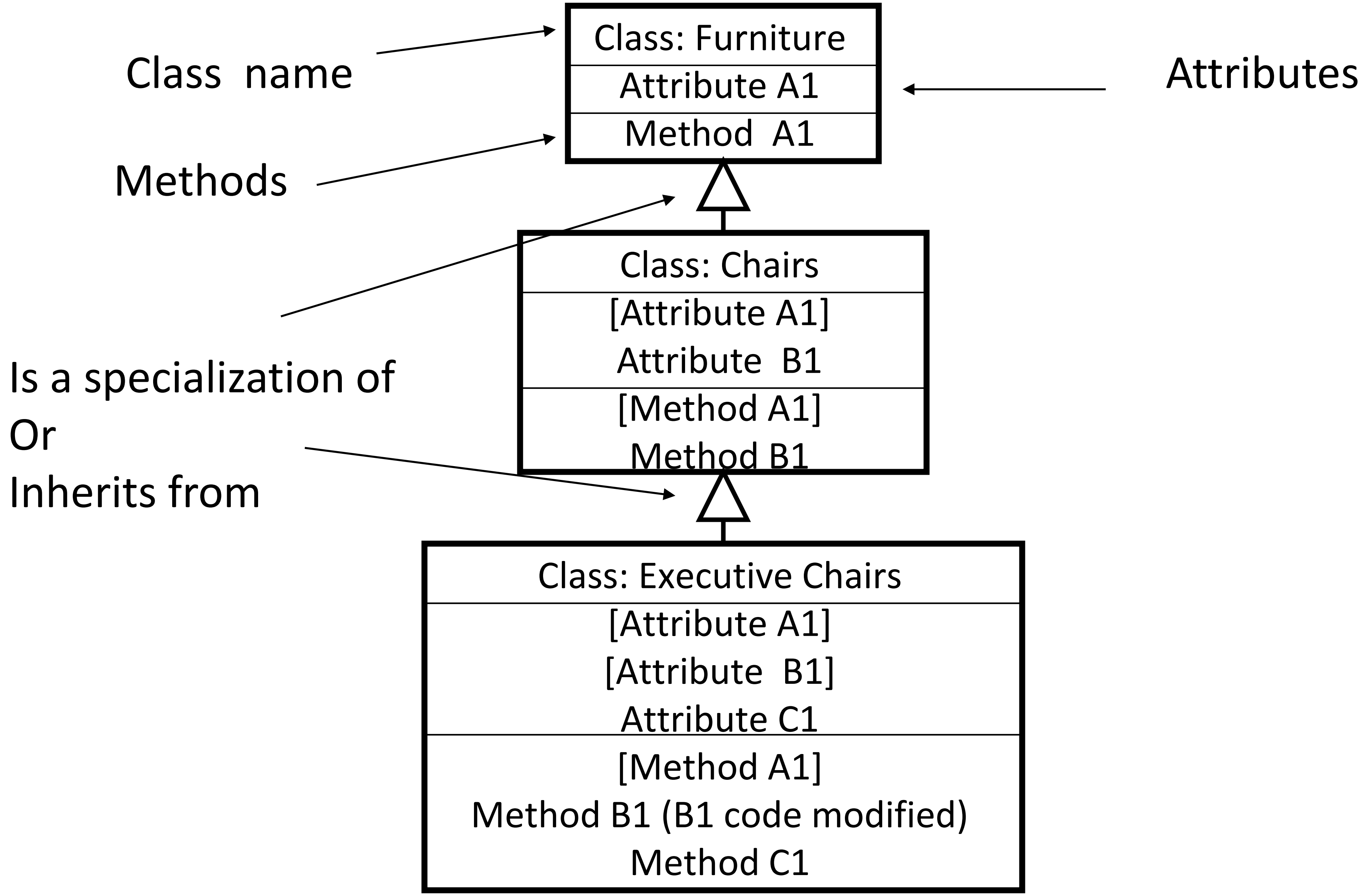


MORE EXAMPLE



A FileDialog is a Dialog is a Window is a Container

CLASS INHERITANCE & SPECIALIZATION



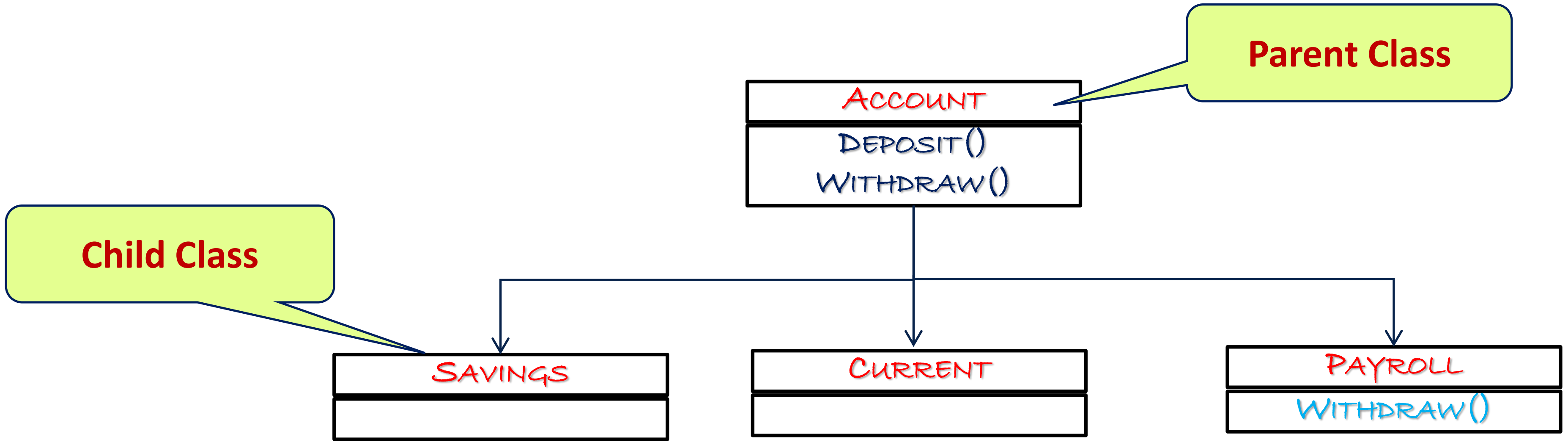
Public, Private & Protected

- ✓ 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
- ✓ 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.

POLYMORPHISM

- ✓Inheritance lets users inherit attributes and methods, and polymorphism uses these methods to perform different tasks.
- ✓Means that the same method will behave differently when it is applied to the objects of different classes

LET'S TAKE AN EXAMPLE



The operation of deposit and withdraw is same for Savings and Current accounts. So the inherited methods from Account class will work. However the withdraw method need to be modified for payroll class.

when the "withdrawn" method for saving account is called a method from parent account class is executed.

But ,when the "Withdraw" method for the payroll account is called withdraw method defined in the privileged class is executed. This is **Polymorphism in OOPs.**

Method Overriding

- Method Overriding is redefining a super class method in a sub class.
- **Rules for Method Overriding**
- The method signature i.e. method name, parameter list and return type have to match exactly.
- The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa.