CMPS 251

Object Oriented Principles

Dr. Abdelkarim Erradi

Computer & Engineering Science Dept.

QU

Outline

1. What is Object Oriented Programming (OOP)?

2. OO Principles: Modularity, Abstraction, Encapsulation, Inheritance and Polymorphism

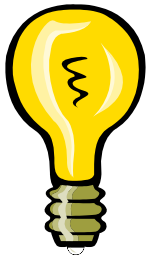
What is Object Oriented Programming (OOP)?

Back

What is OOP?

- Object Oriented Programming (OOP):
  - **Programming paradigm that uses objects and their interactions to design and develop computer programs**
  - A **class** = a definition of an object
  - An **object** = an instance of a class
  - A running program can be seen as a **collection of objects collaborating** to perform a given task
  - OOP = a set of principles (Abstraction, Encapsulation, Inheritance, Polymorphism) guiding software construction
  - Classes allow the software developer to represent real-world concepts in their software design
  - A class **encapsulate** attributes and methods

Example Classes



LightBulb

- **attributes**
  - on (true or false)
- **methods**
  - switch on
  - switch off
  - get state

Car

- **attributes**
  - color
  - made
  - kms travelled
  - made year
- **methods**
  - start
  - accelerate
  - stop
  - get kms travelled
  - get made year

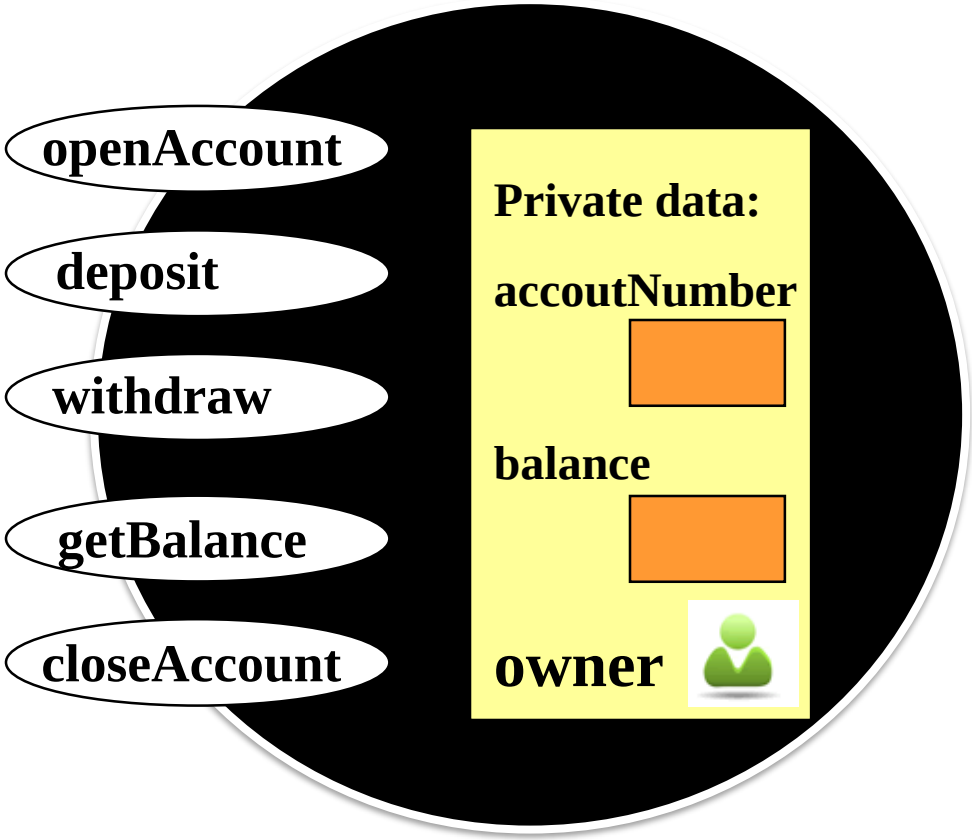
BankAccount

- **attributes**
  - balance
- **methods**
  - deposit
  - withdraw
  - get balance

Note

- **Each object (i.e., class instance) has its own values for its attributes** (e.g., different accounts can have different balances)

BankAccount Example



Account class contains attributes and methods

An Object has:

- **Attributes** – information about the object
- **Methods** – functions the object can perform
- **Relationships** with other objects
  - e.g., A BankAccount has an Owner

Classes

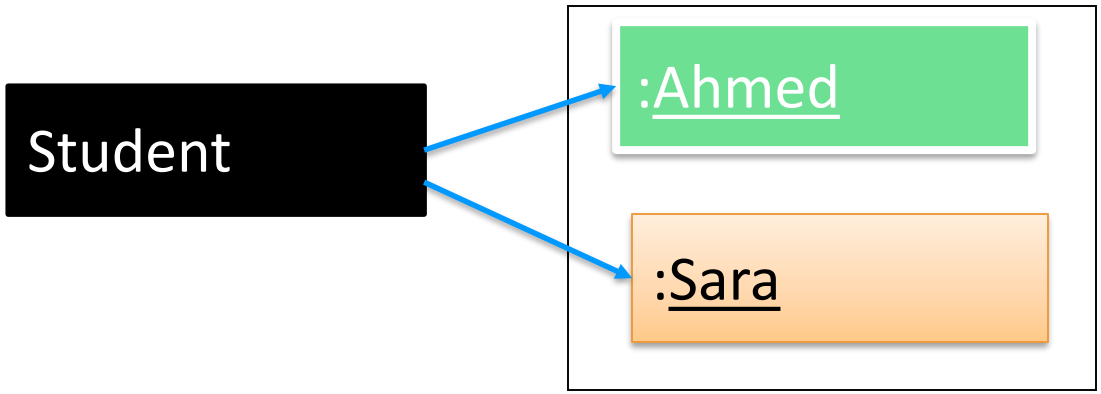
- A class is a **programmer-defined data type** and **objects are variables of that type**
  - Classes allow us to create new data types that are well suited to an application.
  - You create objects by **instantiating** a class

  e.g., Student quStudent;

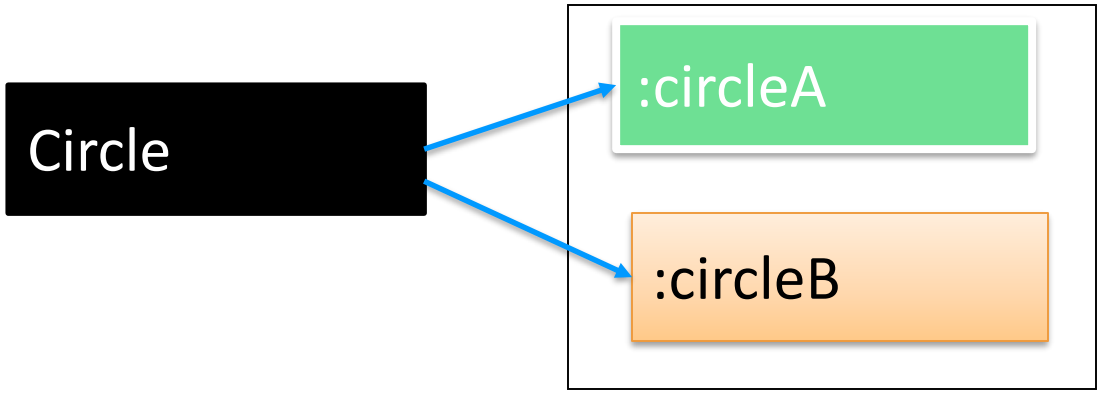
  This declares quStudent object of type Student.

- A class contains private **attributes** and public **methods**

Class vs. Object



Ahmed and Sara are objects of class Student



circleA and circleB are objects of class Circle

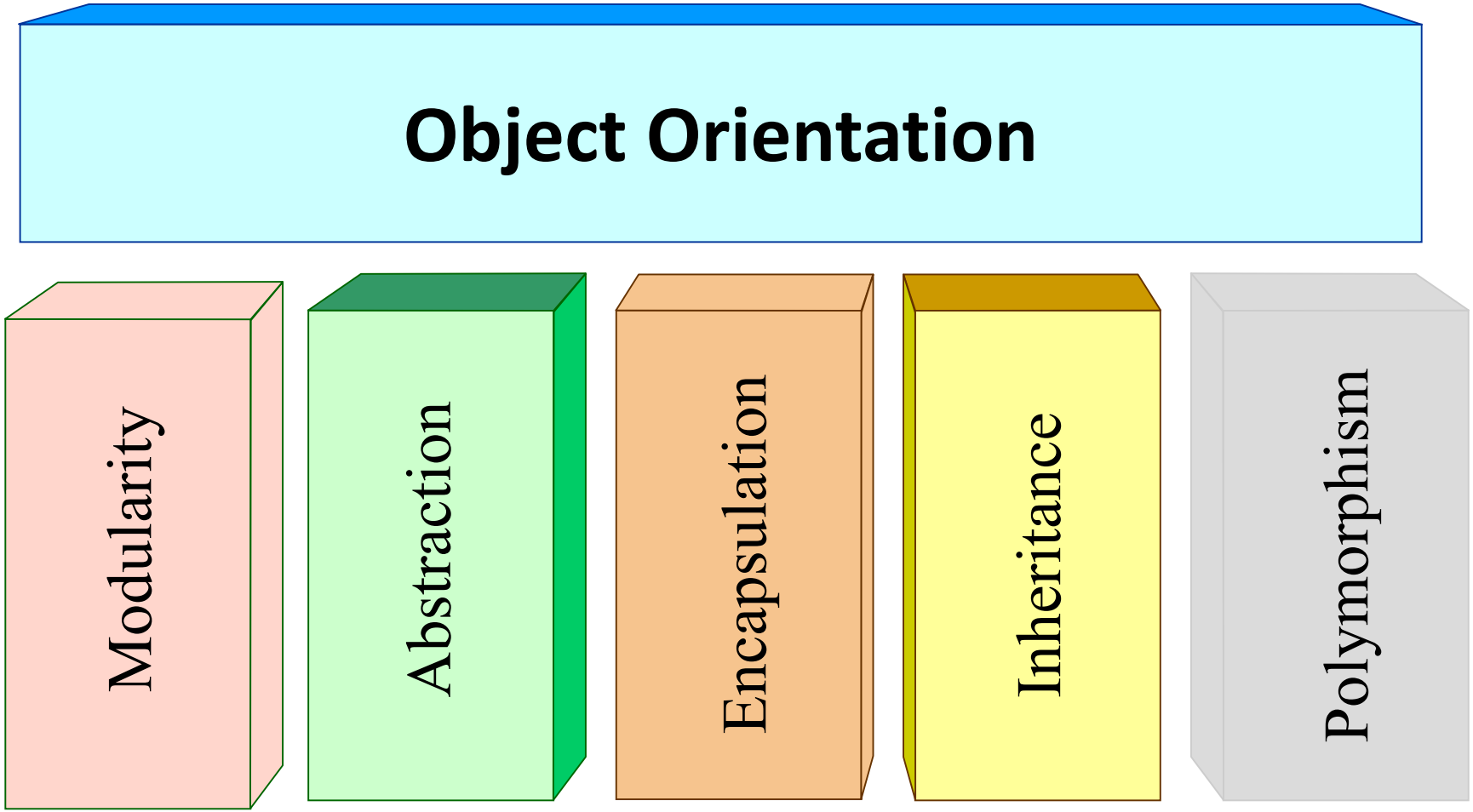
- Object is an instance of a class.

Very important to understand

OOP Principles

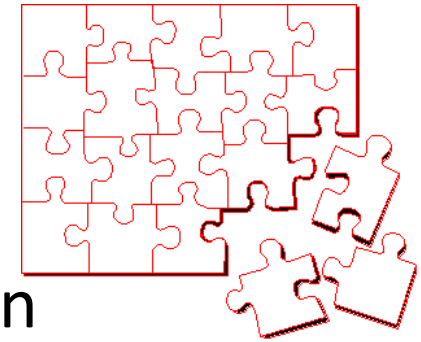
Back

OOP Principles



Modularity is a must!

- **To reduce complexity, we need to break a program into smaller pieces**
  - Facilitate the design, implementation, operation and maintenance of large programs
  - **Permits reuse** of logic
  - Ease **maintainability** and understandability



- Two ways to perform decomposition:
  - **Functional** (or Procedural) decomposition
  - **Object-oriented** decomposition

Two ways to divide and conquer!

Functional decomposition

- We think in terms of sequence of steps to solve the problem
- Break down a program into a set of functions
- Each function handles a **single logical "chunk" of the solution**

$\Rightarrow$ **A program is a collection of one or more collaborating functions $\{f_0, f_1, \ldots. f_n\}$**

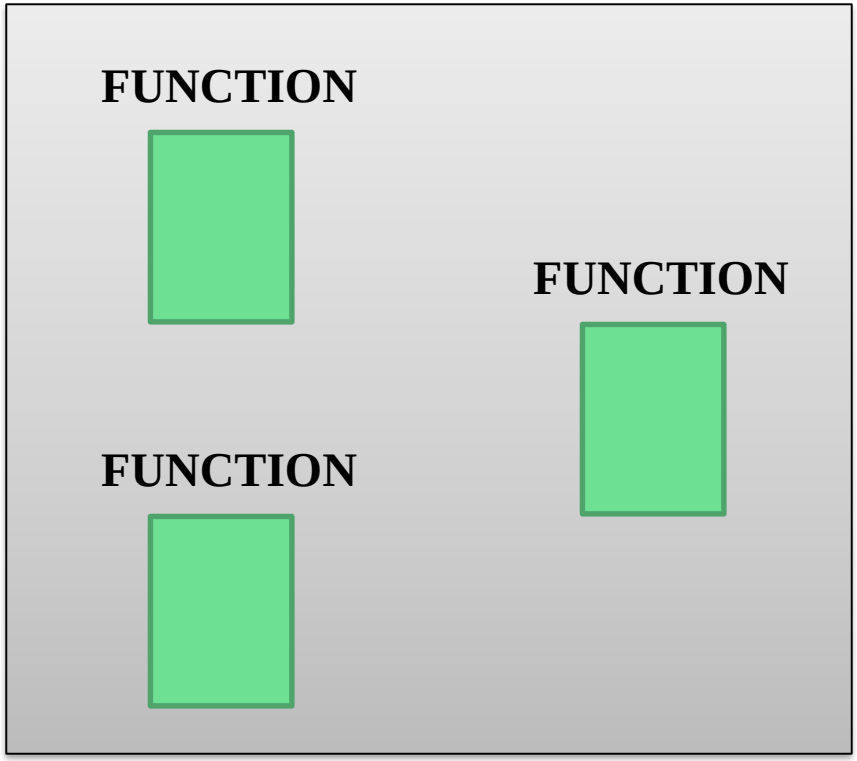
Object-oriented decomposition

- We think of a program as a set of objects that interact
- Each object has some **attributes** and **methods**

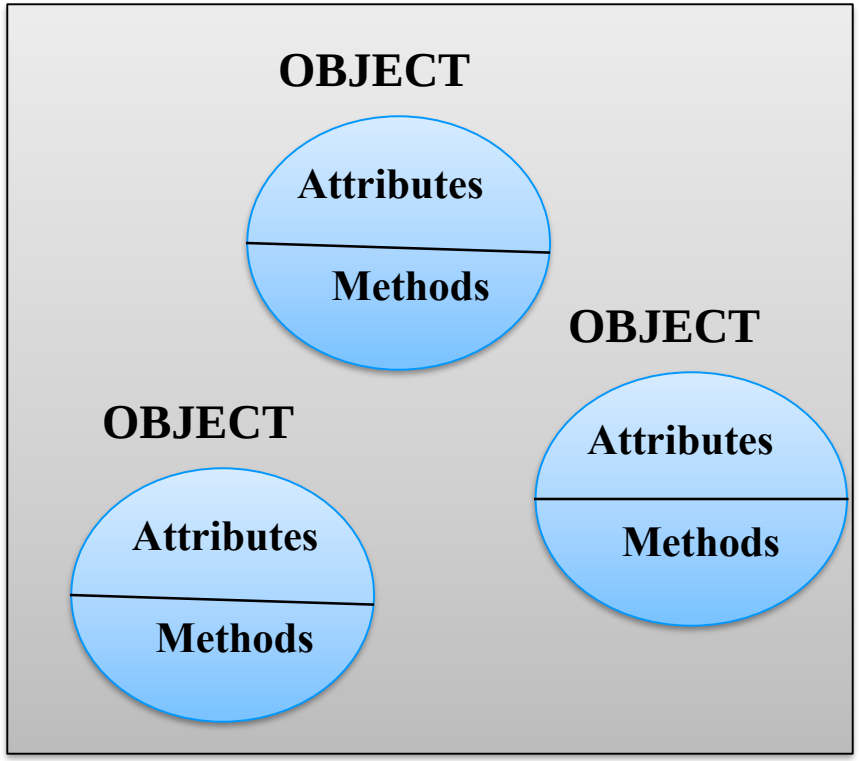
=> **A program is a collection of one or more cooperating objects $\{O_0, O_1, \ldots. O_n\}$**

Functional decomposition vs. Object-oriented decomposition

Functional Decomposition



Object-Oriented Decomposition



Functional decomposition vs. Object-oriented decomposition

- When using functional decomposition, functions and data are separated. The key problems are:
  - Poor modeling of things in the real world
  - Difficulty of creating new Data Types

- In the physical world we deal with **objects** such as people and cars. Real-world objects have both:

Attributes & Methods

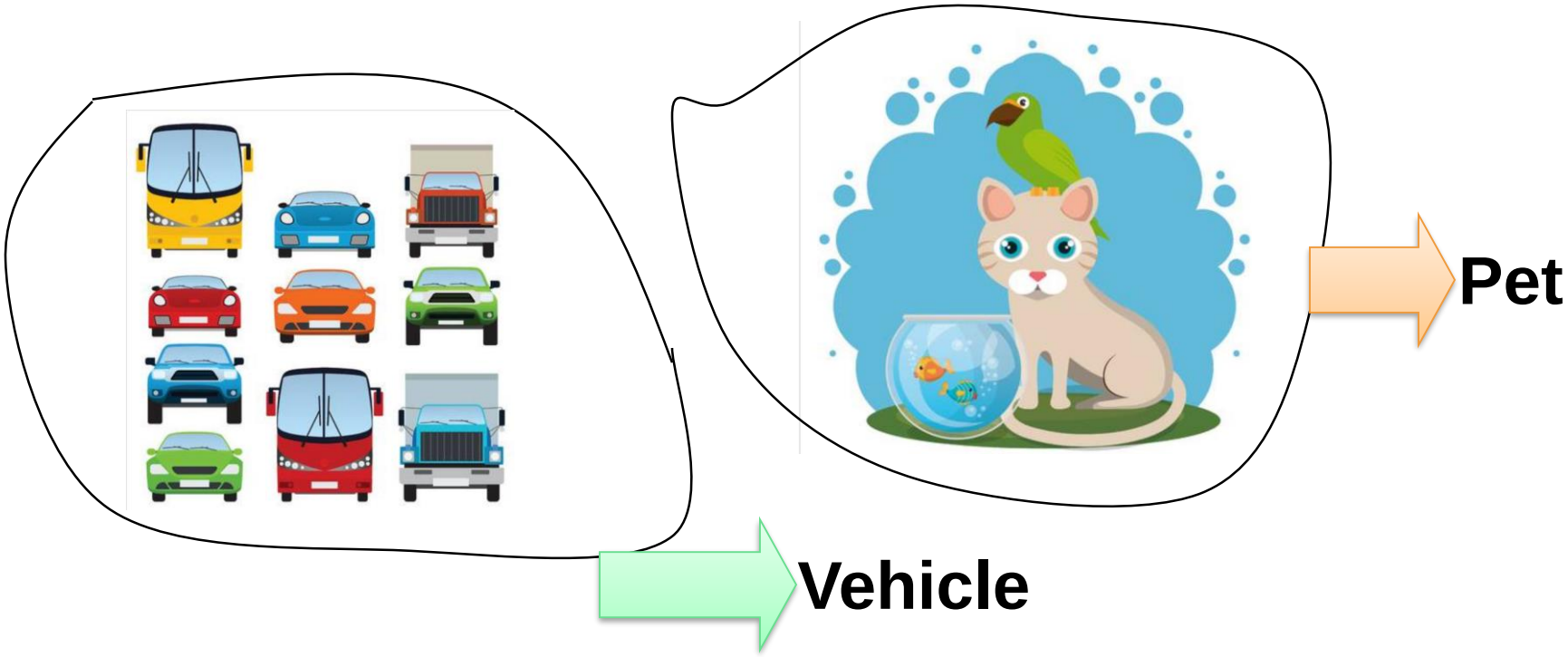
  - OOP allows creating new Data Types that represent real world objects and concepts such as Student, Course…

Abstraction

- A **meaningful grouping** of similar objects can be ABSTRACTED to a class
- Abstraction allow creating classes (i.e., new data types) that are well suited to an application.
  - Abstraction allows us to model our system using the concepts and terminology of the problem domain (i.e., the problem to be solved)
  - Software classes are inspired from the domain concepts
- Abstraction allows us to manage complexity by creating a simplified representation of something
  - Concentrating on the essential characteristics needed in the application we are developing

ABSTRACTION

- **Abstraction =** meaningful grouping of objects



- A grouping of objects can be **ABSTRACTED** to a <u>class</u>

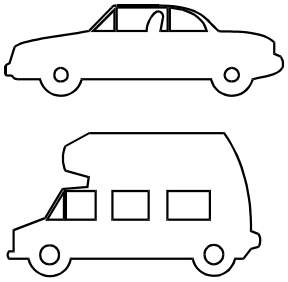
Classes = abstraction of objects with the same attributes and behavior

Person Objects

Abstract Into →

Person Class
Attributes: `Name, Age`
Operations: `Speak(), Listen(), Walk()`

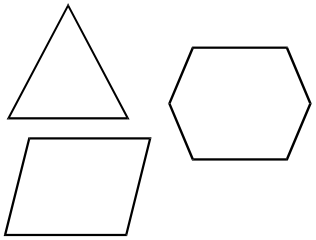
Vehicle Objects



Abstract Into →

Vehicle Class
Attributes: `Name, Model, Color`
Operations: `Start(), Stop(), Accelerate()`

Polygon Objects



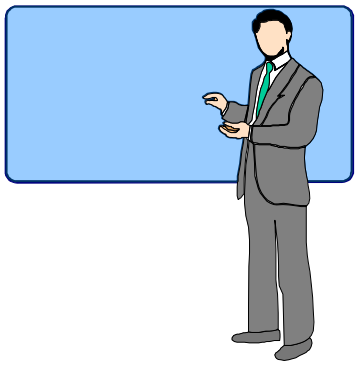
Abstract Into →

Polygon Class
Attributes: `Vertices, Border, Color, FillColor`
Operations: `Draw(), Erase(), Move()`

Example: Abstraction for a Student Registration System

Student

Schedule



Instructor

Section
(e.g., 9:00 am to 10am
Sunday-Tuesday-Thursday)

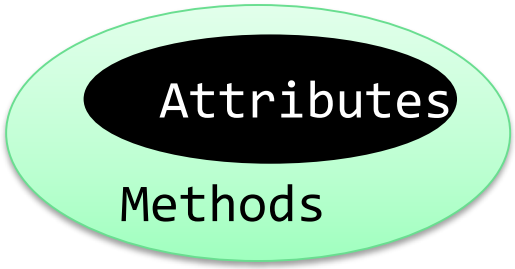
Course (e.g. Algebra)

Encapsulation

- **Encapsulation** = to **combine attributes and methods into a single unit** called an **class**
- Make attributes private (hidden) and provide getters / setters

  ⇒ Attributes are safe from any accidental change
- Hide the class implementation from clients

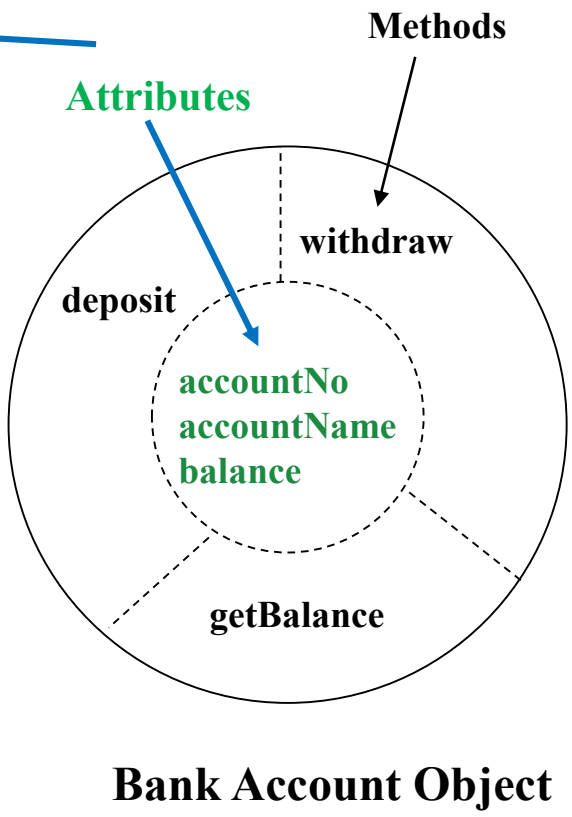
  ⇒ Clients access the object via **public interface**



Encapsulation - Example

```
public class Account {
    private int accountNo;
    private String accountName;
    private double balance;

    public void deposit(double amount) {
        balance += amount;
    }
    public void withdraw(double amount) {
        balance -= amount;
    }
    public double getBalance() {
        return balance;
    }
}
```

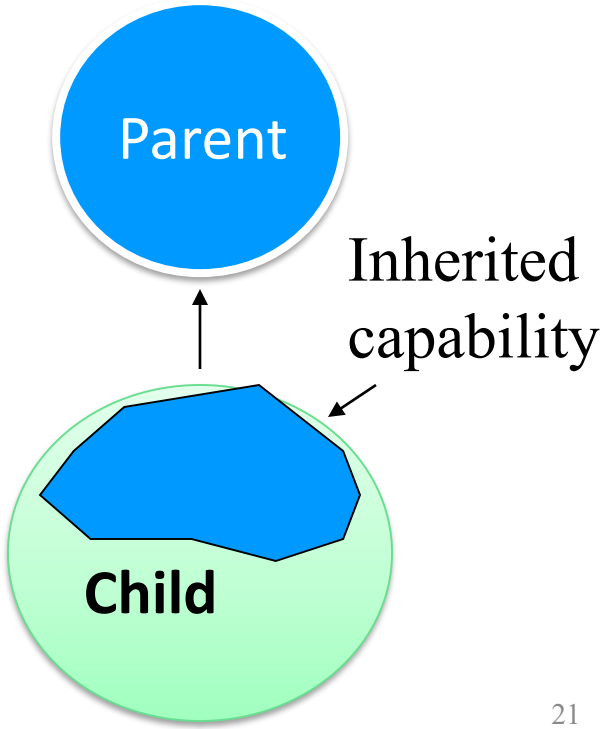


Bank Account Object

Inheritance

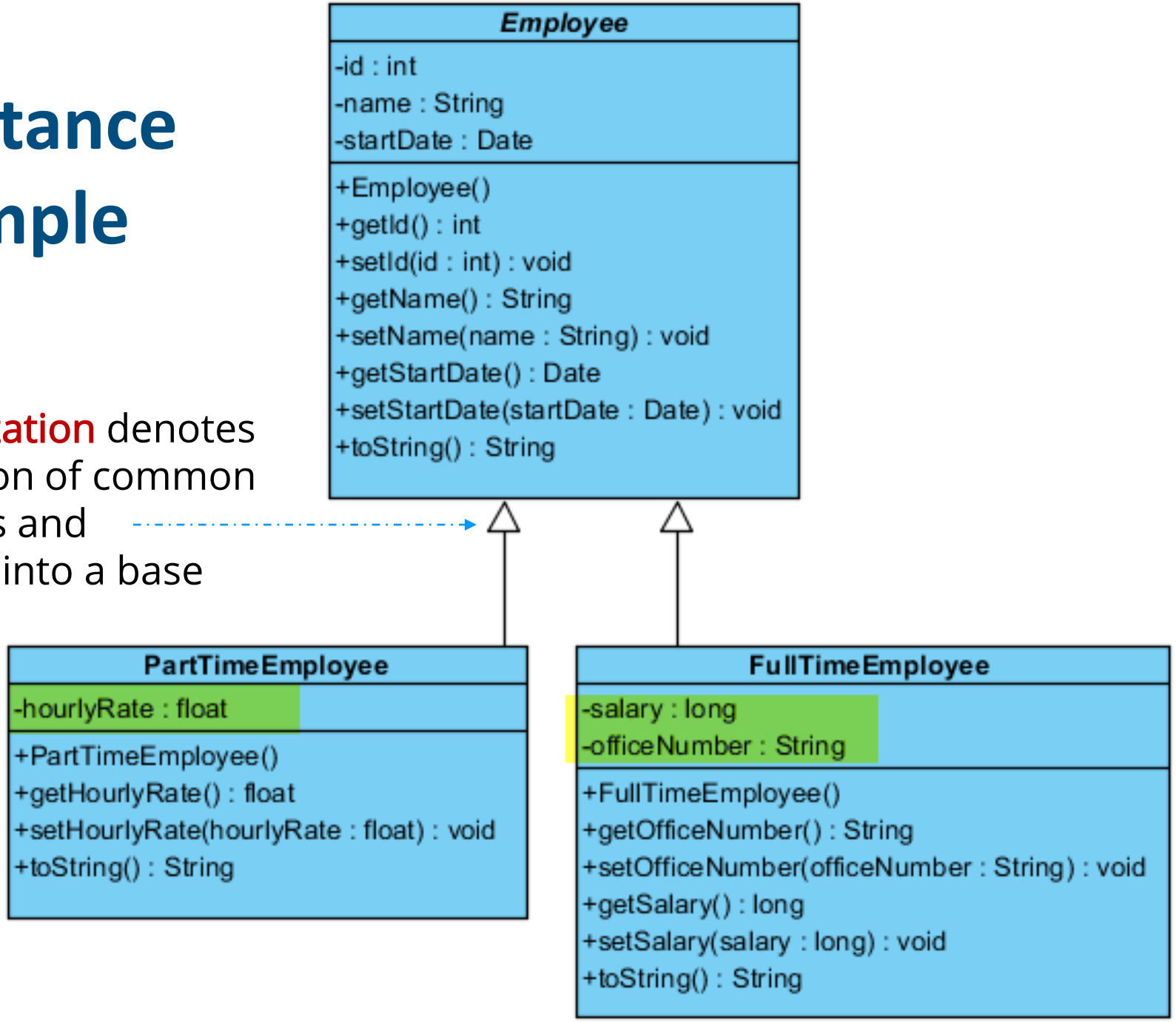
- Organize classes in **inheritance hierarchies**
  - A subclass inherits its parent's attributes and methods

- Inheritance leverages the similarities among classes
  - This allows **reuse** since the implementation is not repeated



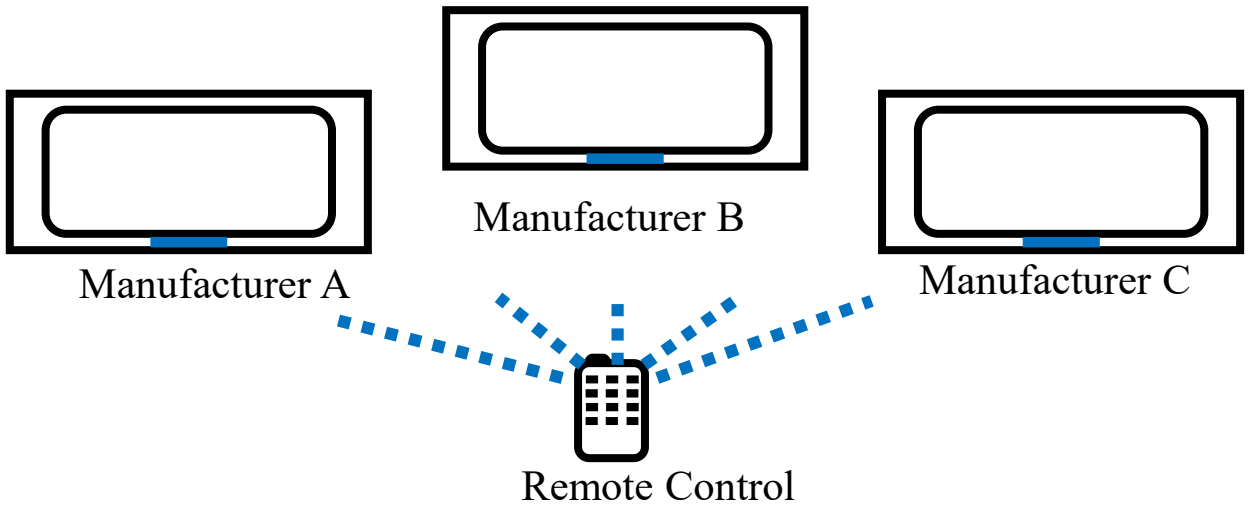
Inheritance Example

Generalization denotes abstraction of common attributes and methods into a base class

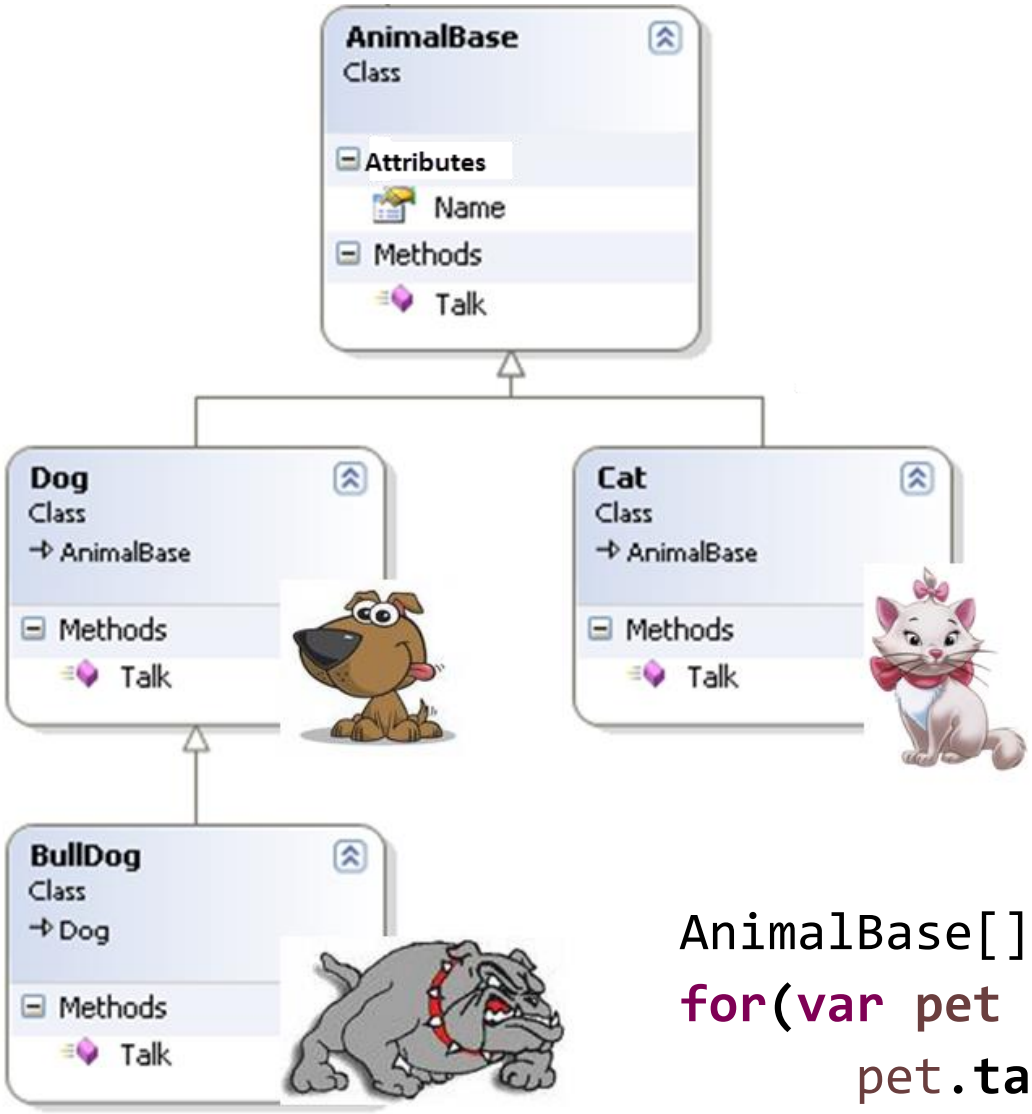


Polymorphism

- Objects of **different types** can be accessed through the **same interface**
  - Within an inheritance hierarchy, a subclass can override a method of its superclass to have a **different implementation** (e.g., calculateArea in a rectangle is implemented differently in a circle)



Polymorphism Example



Note that all animals have **Talk** method but the **implementation is different**:

- Cat says Meowww!
- Dog says: Arf! Arf!
- BullDog : Aaaarf! Aaaarf!

```
AnimalBase[] pets = {cat, dog, bulldog};
for(var pet : pets) {
        pet.talk();
}
```

Benefits of OOP (1 of 2)

- **Better understandability** since objects within a program often model real-life objects in the problem to be solved
- **High degree of organization and modularity** of the code

  → Easier to partition the work in a project based on objects

  →This fits the needs of large projects

Benefits of OOP (2 of 2)

- **Encapsulation**:

+ Reduces software complexity

  To use an class you just need to know its public interface and can ignore the details of how it is implemented

+ Protect attributes from accidental changes

- **Inheritance**:

+ Increase reuse & Eliminates redundant code

→Save development time and get higher productivity

- **Polymorphism**:

+ Makes it possible to call methods with different implementations using one interface

Summary

- OOP is a powerful and widely used programming style
- Key OOP principles are: **Abstraction**, **Encapsulation**, **Inheritance** and **Polymorphism**
- Enables **easy mapping** of real world objects to objects in the program
- Applications built using OOP are **flexible to change**, well organization, and allow code **reuse**
- More info @ “OOP Concepts” section in Oracle Java Tutorial http://download.oracle.com/javase/tutorial/java/