CMPS 251

Object Oriented Principles

Read Chapters 3 & 6

Dr. Abdelkarim Erradi

Computer & Engineering Science Dept.

QU

Outline

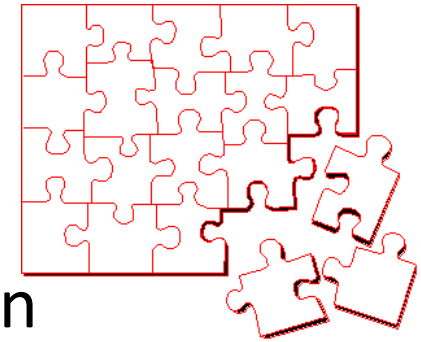
1. Approaches of Program Decomposition
2. What is Object Oriented Programming (OOP)?
3. OO Pillars: Modularity, Abstraction, Encapsulation, Inheritance and Polymorphism

Approaches of program decomposition

Back

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**

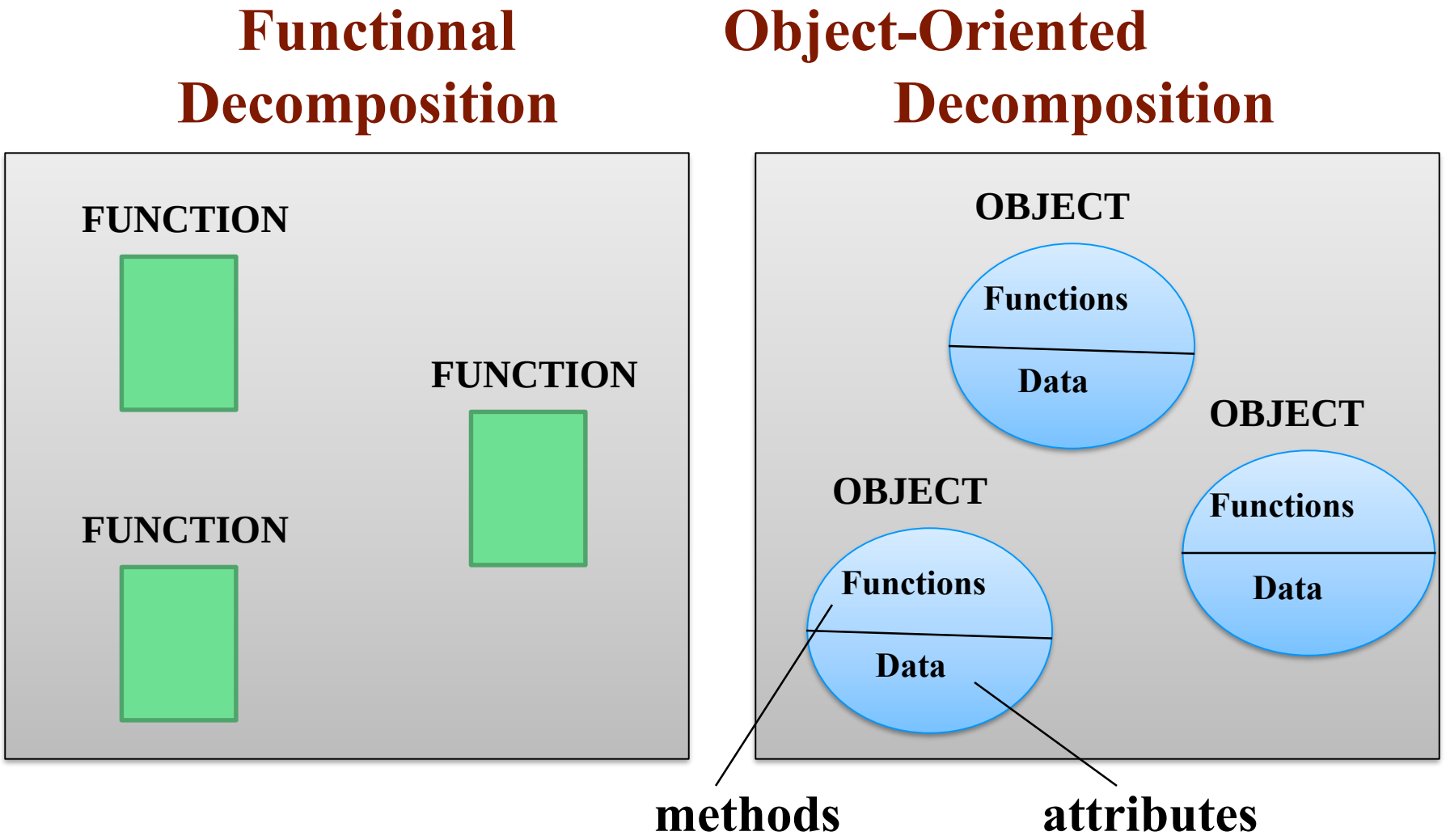
$\Rightarrow$ **A program is a collection of one or more collaborating functions $\{f_0, f_1, \ldots. f_n\}$**

Object-oriented decomposition

- We think of a program as a set of autonomous objects that collaborate to perform some goal
- Each object has some **attributes** and **functions**

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

Functional decomposition vs. Object-oriented decomposition



Key problem with Functional Programming = Poor Real World Modeling & Difficulty of creating new Data Types

- Functions and data are separated

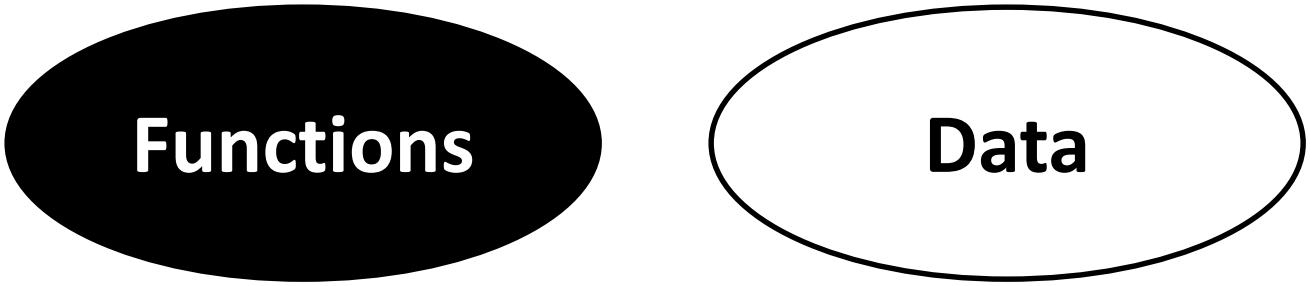
  ⇒**provide a poor modeling of things in the real world**

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

Data & Functions

- OOP allow you to create new Data Types that represent real world objects and concepts such as Student, Date…

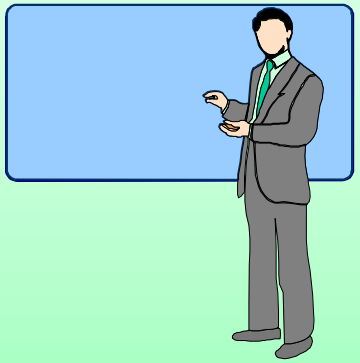
Procedural Programming Model



Functions & Data are arranged separately

OOP Programming

Functions
+
Data



Real Objects have both Functions & Data

Functional Decomposition

- **FOCUS** is on actions and algorithms.
- **BEGINS** by breaking the solution into a series of major steps. This process continues until each **subproblem** cannot be divided further or has an obvious solution
- **UNITS** are ***functions*** representing **algorithms**
- **DATA** plays a secondary role in support of actions to be performed

Object-oriented decomposition

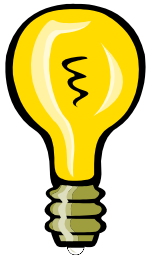
- **FOCUS** is on entities called **objects**. The objects encapsulate data and functions that manipulate that data
- **BEGINS** by identifying the major objects in the problem, and choosing appropriate **data items** and **functions** on those objects
- **UNITS** are ***objects***. Programs are collections of objects that communicate with each other
- **DATA** plays a leading role. Algorithms are used to implement functions on the objects and to enable interaction of objects with each other

What is Object Oriented Programming (OOP)?

What is OOP?

- Object Oriented Programming (OOP):
  - **Programming paradigm that uses "objects" and their interactions to design and develop computer programs**
  - OOP = a set of principles (Abstraction, Encapsulation, Inheritance, Polymorphism) guiding software construction
  - Objects allow the software developer to represent real-world concepts in their software design
  - A running program can be seen as a **collection of objects collaborating** to perform a given task
  - The objects **encapsulate** data and functions that manipulate that data

Examples of Objects



LightBulb

- **attributes**
  - on (true or false)
- **methods**
  - switch on
  - switch off
  - check if on

Car

- **attributes**
  - color
  - liters of petrol in tank
  - kms run so far
  - current speed (km/h)
- **methods**
  - accelerate
  - stop
  - get petrol level
  - get odometer reading

QNB

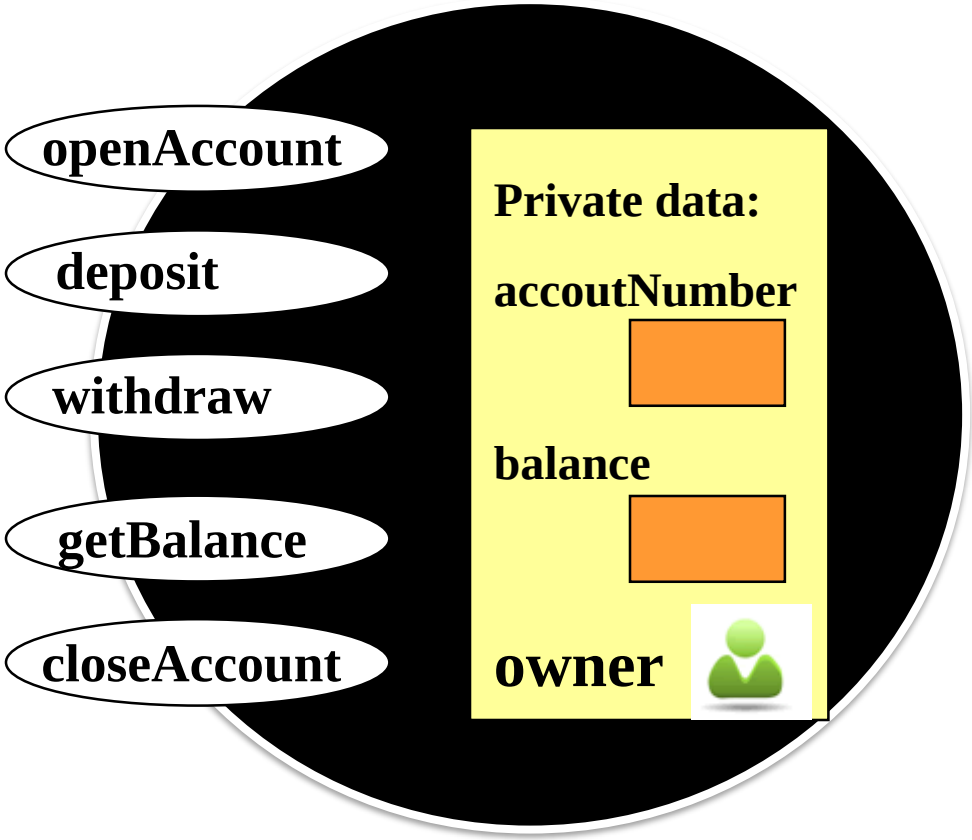
BankAccount

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

Note

- **each object is an "instance" of that "class" of object**
- **each instance has its own values for its attributes**
  - e.g., different accounts can have different balances

BankAccount Example



BankAccount 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

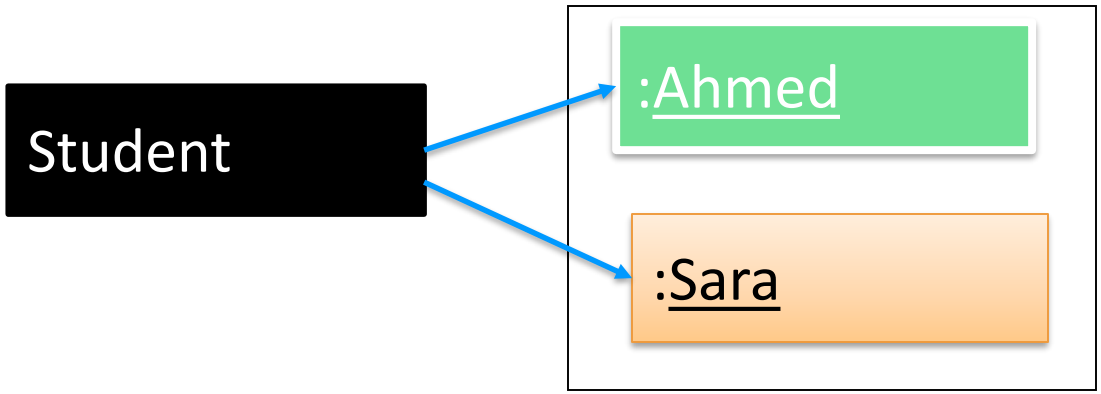
- A class is a **programmer-defined data type** and **objects are variables of that type**
  - Classes allow us to create new data types that are well suited to an application.
  - You create objects by **instantiating** a class

  e.g., Student quStudent;

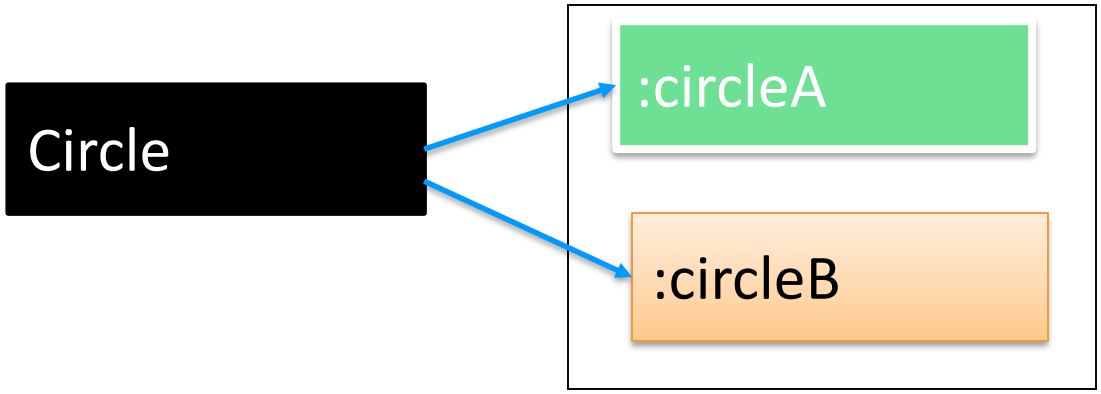
  This declares quStudent object of type Student.

- 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

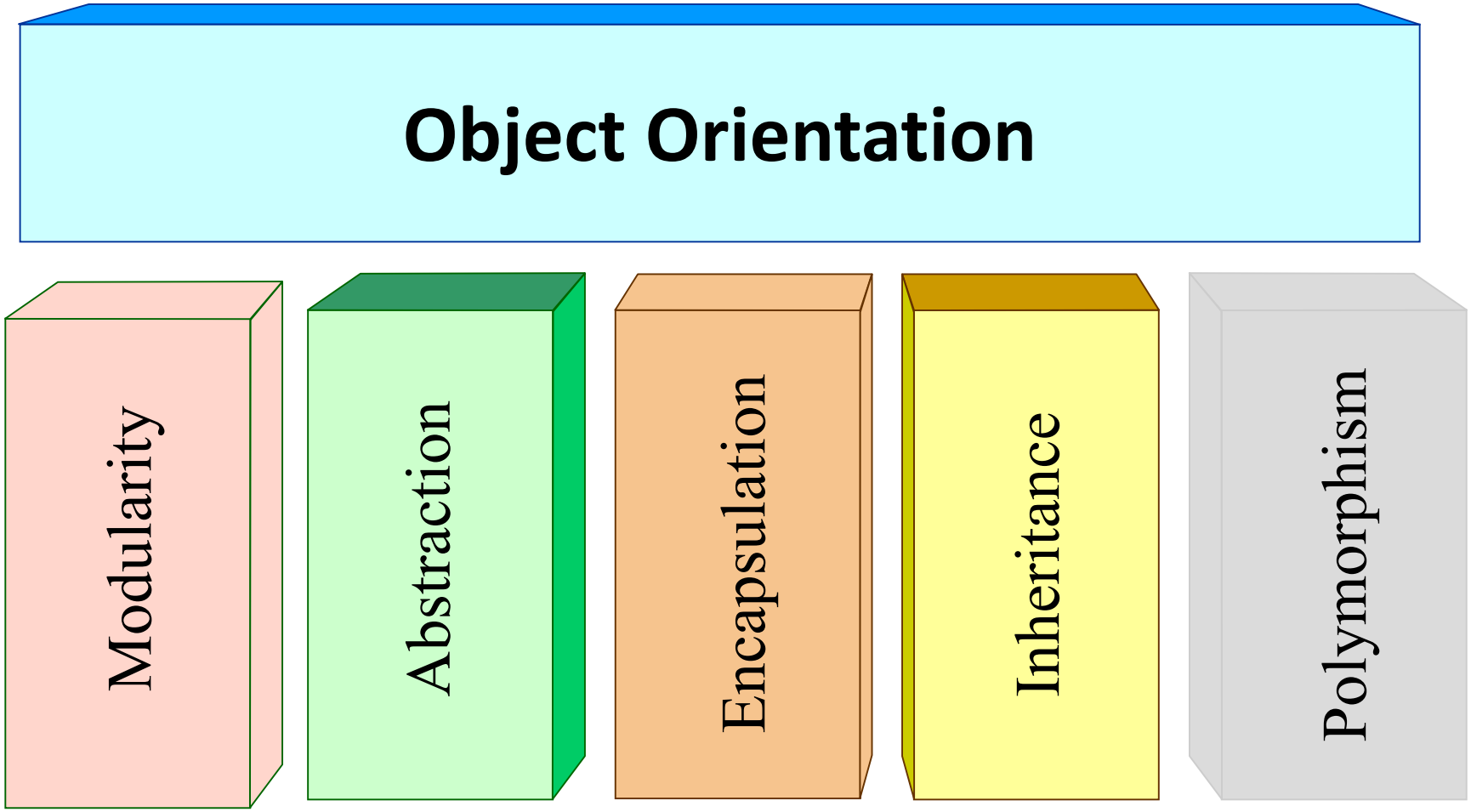
- Object is an instance of a class.

OOP Pillars

Very important to understand and master

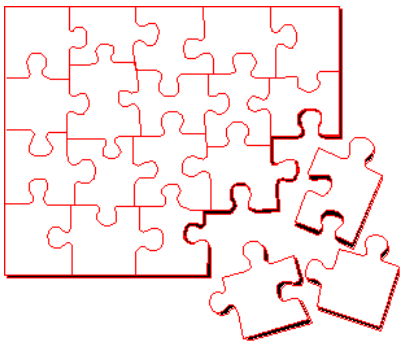
Back

Basic Pillars of Object Orientation



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



- Object-oriented decomposition is widely used:

  => We think of a program as a set of autonomous objects

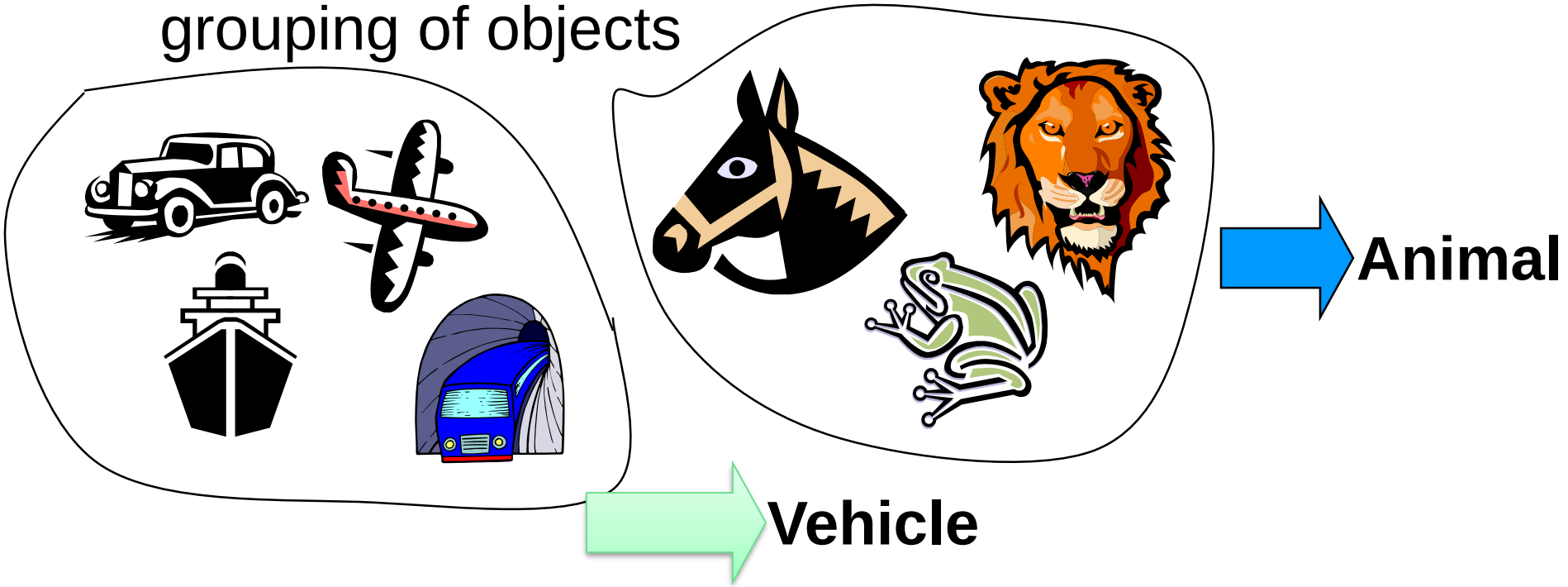
  {O_0, O_1, …. O_n} that collaborate to fulfill the requirements

Abstraction

- The technique of creating new data types that are well suited to an application.
  - OO allows us to model our system using the concepts and terminology of the problem domain
  - 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

CLASSIFICATION and ABSTRACTION

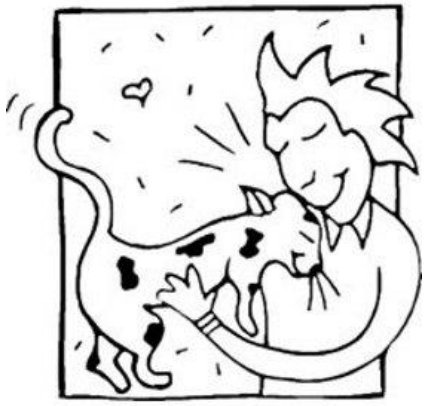
- **CLASSIFICATION** starts with meaningful grouping of objects



- A good collection of well classified objects can be **ABSTRACTED** to classes

AN OBJECT IS ABSTRACTED TO A CLASS

- A CLASS is a DEFINITION, a TEMPLATE, for the objects => A class is a *type* of thing
- NOTE: A CLASS IS NOT A COLLECTION OF OBJECTS.



A "CAT" in general is a CLASS
***Your* cat and *His* cat are specific objects**

- The role of a class is to define the attributes (state) and methods (behavior) of its instances.
- The class **Car**, for example, defines the attribute color.
  - Each individual car (object) will have a value for this property, such as "maroon," "yellow" or "white"

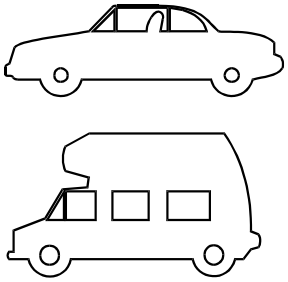
Classes = abstraction of objects with the same attributes and behavior

Person Objects

Abstract Into →

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

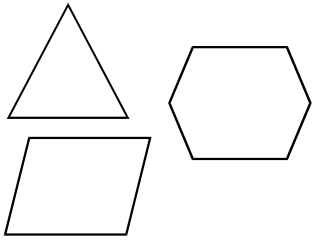
Vehicle Objects



Abstract Into →

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

Polygon Objects



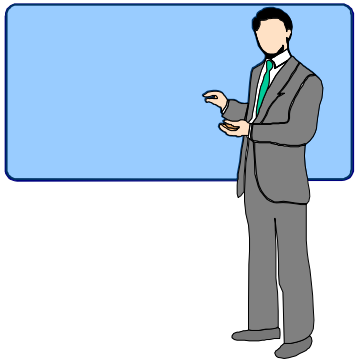
Abstract Into →

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

Example: Abstraction for a Student Registration System

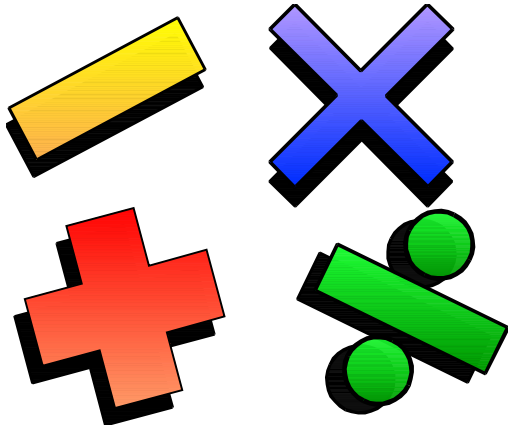
Student

Schedule



Professor

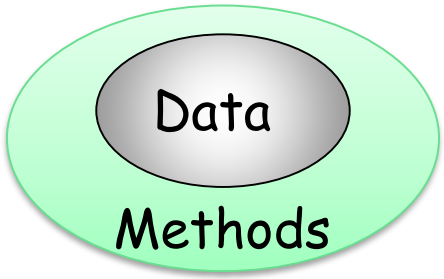
Course Offering (9:00 am to 10am Sunday-Tuesday-Thursday)



Course (e.g. Algebra)

Encapsulation

- **Encapsulation** = to **combine attributes and methods into a single unit** called an **object**
  - Hiding implementation from clients
    - Clients access the object via **public interface**
  - The data is *hidden*, so it is safe from any accidental alteration. Methods are used to access the Object's data

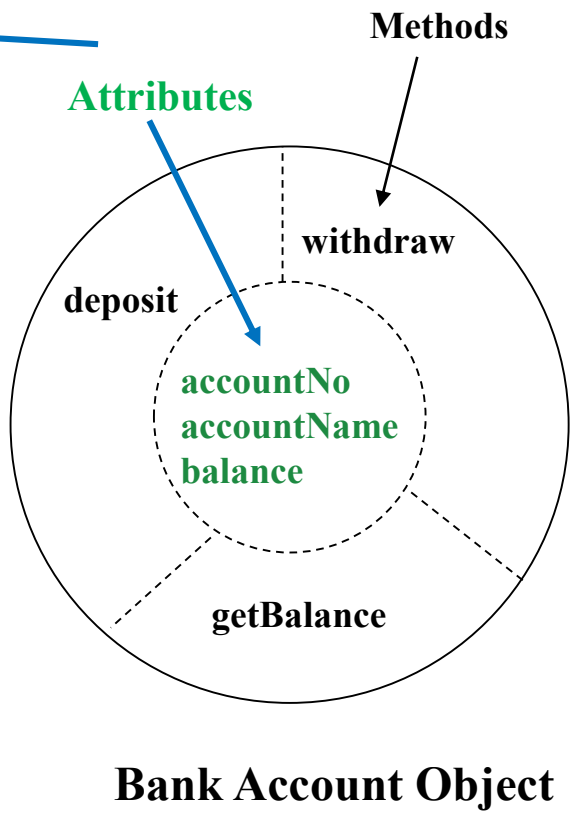


Encapsulation is the foundation of OOP

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;
    }
}
```

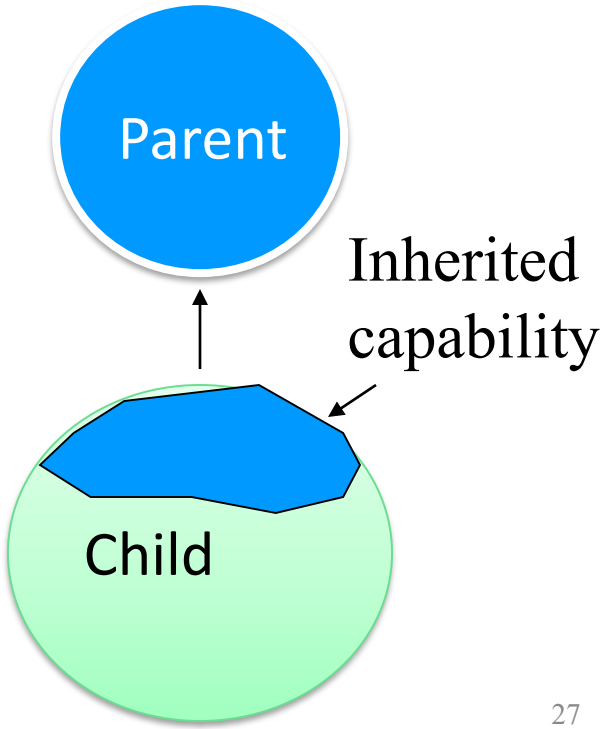


Bank Account Object

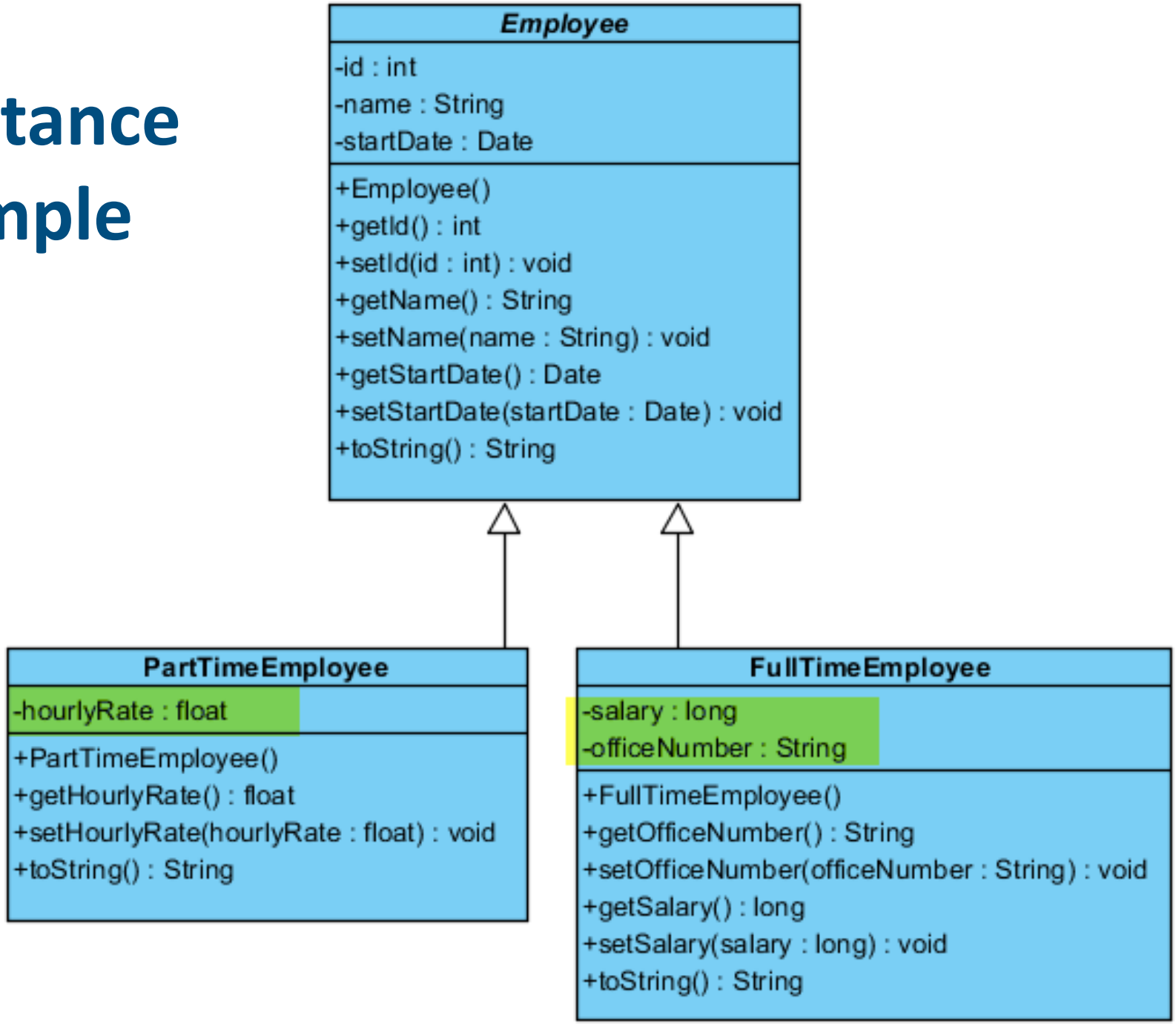
Inheritance

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

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

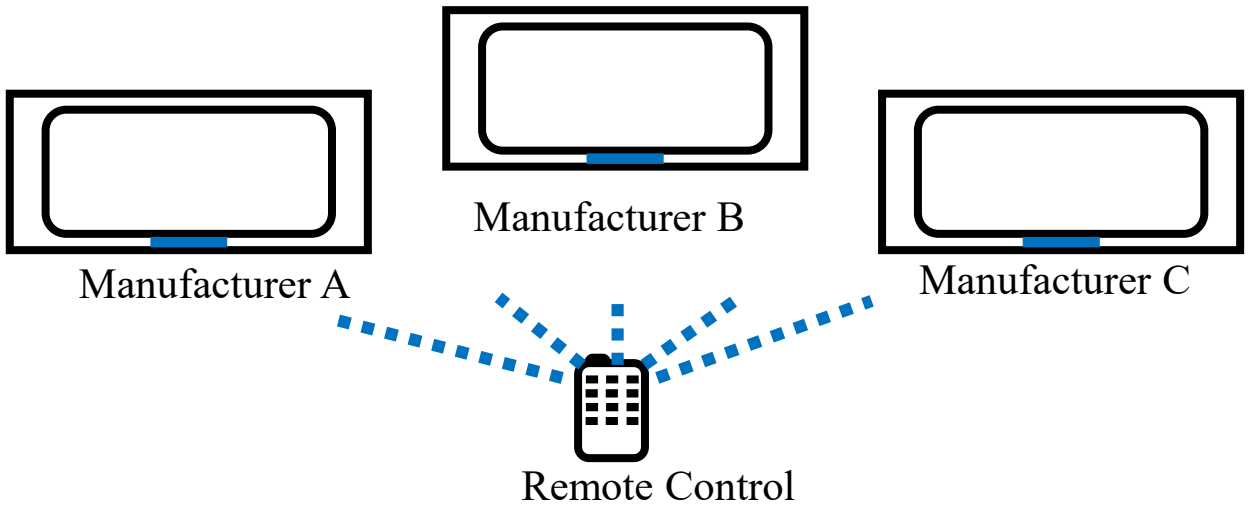


Inheritance Example

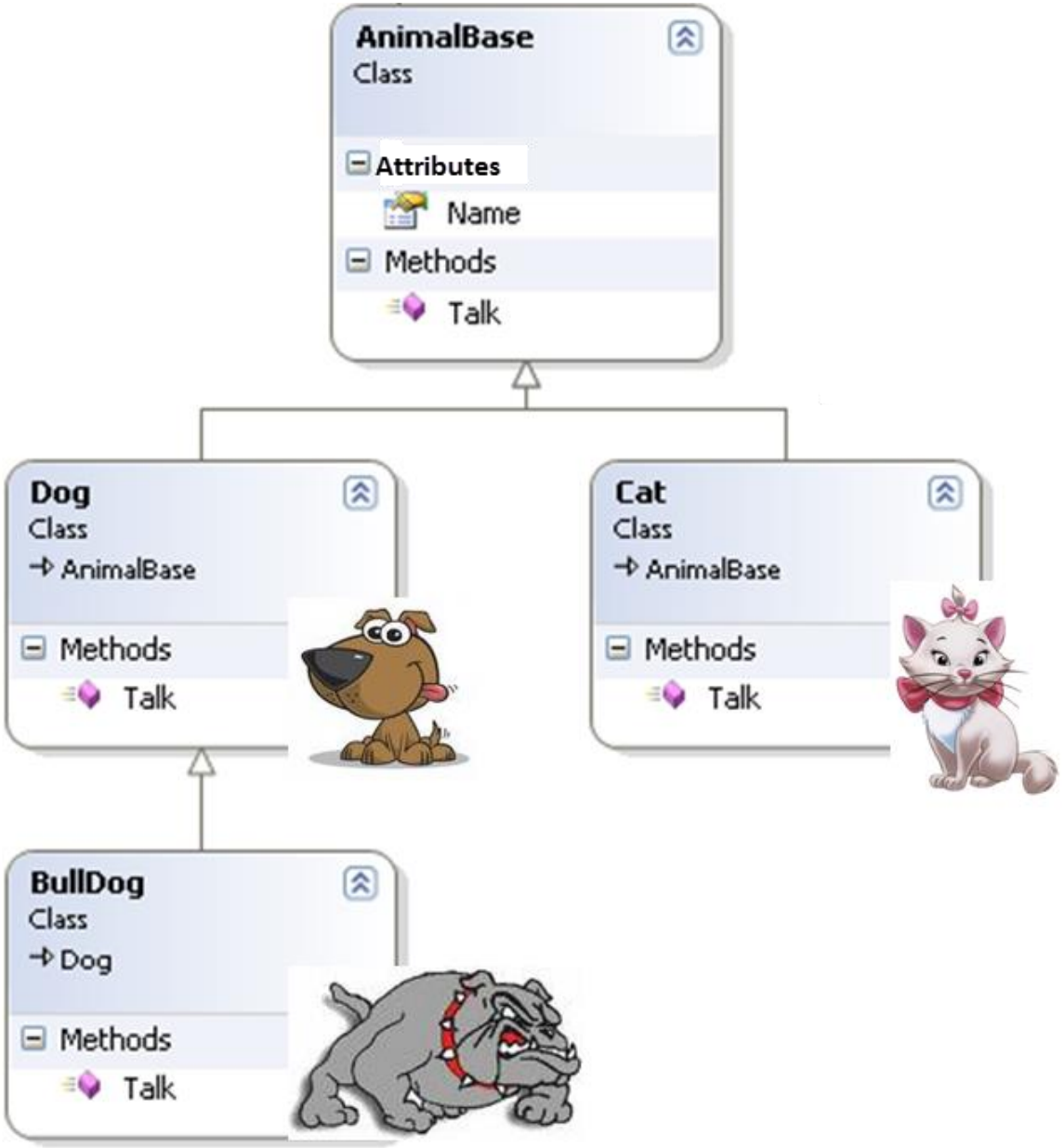


What Is Polymorphism?

- The ability to hide many different implementations behind a single interface:
  - The capability of a method to do different things based on the object that it is acting upon (e.g., calculating area in a rectangle is done differently from 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!

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 object you just need to know its public interface and can ignore the details of how it is implemented

+ Improves the resiliency of the system, i.e. its ability to adapt to change

- **Inheritance**:

+ Eliminates redundant code and extend the use of existing classes.

→ Save development time and get higher productivity.

- **Polymorphism:**

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

OOP Summary

- A software can be seen as a **collection of objects collaborating** to perform a given task
- Objects are alive:
  - They know their **attributes**
  - They can do things using their **methods**
  - They exist in different states
  - Each object is unique, it is not any other object.
- Objects live in communities
  - They exchange messages
  - They have relationships with each other
- Classes are blueprints of objects
- Object are instances of classes

Summary

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