Introduction to Object-Oriented Programming

TRAN THANH TUNG

Objectives

Go through most of the basic concepts in Object-Oriented programming

Procedural programming

- Distinct functions or procedures with inputs and output
- Sometimes data is global

Normally, data and functions are separated

Some procedural code …

```
struct coordinate
{
      int x, y;
};
char map[100][100];
int room_width = 62, room_height = 25, food = 0;
int food_x = 0, food_y = 0, delay = 100, ax = 2, ay = 0, point = 3, tmpnum, score[100];
bool gameplay = true;
string name, player[100], tmpstr;
coordinate snake[100];
```

Some procedural code …

```
void setBoard();
void setPlayerShip(char & choice);
void setComShip();
void showBoard();
void showBoard2(int, int, int, int);
void playerAttack(int &x, int &y);
void comAttack(int &x, int &y);
char convert(int n);
void Convert(char ch[],int &,int &, int &, int &);
```

Problems with procedural programming

If we need hundreds of functions, how can you manage the list of functions?

- Sort them in alphabetic order?

If we have hundreds of variables accessed by many different functions,

- How to organize them: Using struct?
- How to make sure there is no unintended modification?

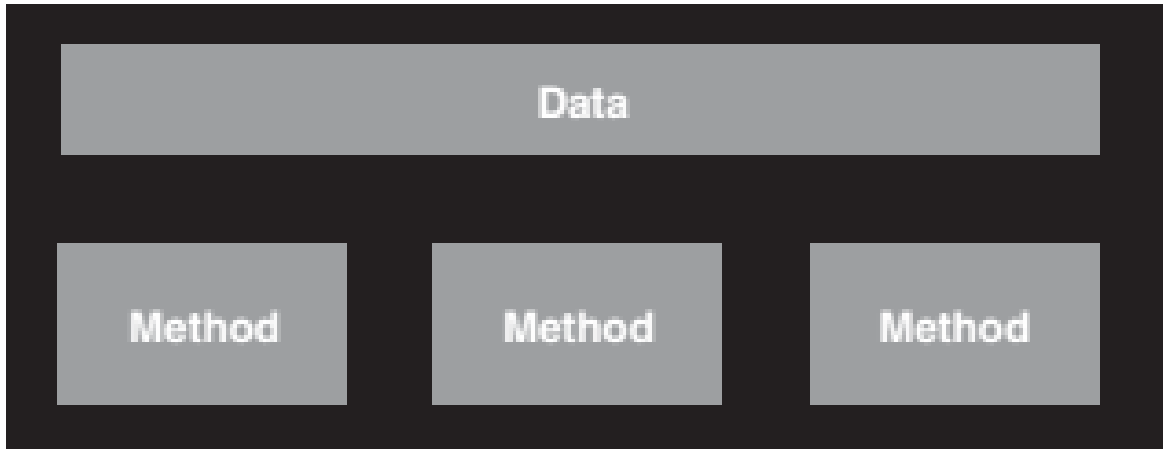
Object-oriented programming addresses these issues

Object-Oriented programming

Object is an entity that contains both *data* and *behaviors*

Ex: a person has

- attributes
  eye color, height, …
- behaviors
  walking, talking, …



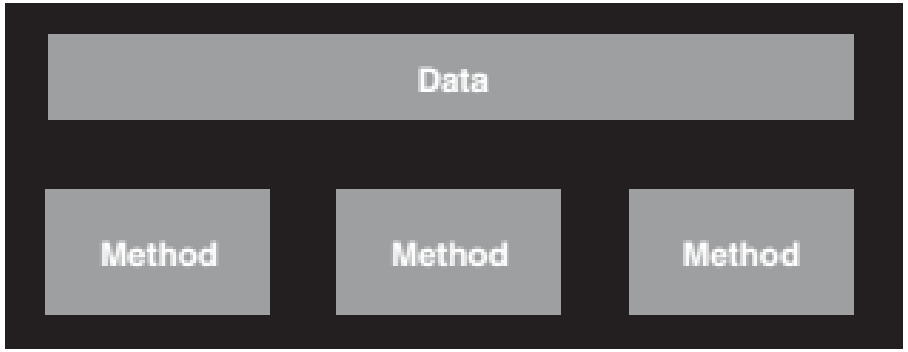
Object-Oriented programming

- Functions related to an object (an entity) are grouped together in one place

- Data related to an entity are stored in a representing object of the entity

In OO term, the combining is

called ***data encapsulation***



Data hiding

In OO terminology,

- data is referred to as ***attributes***

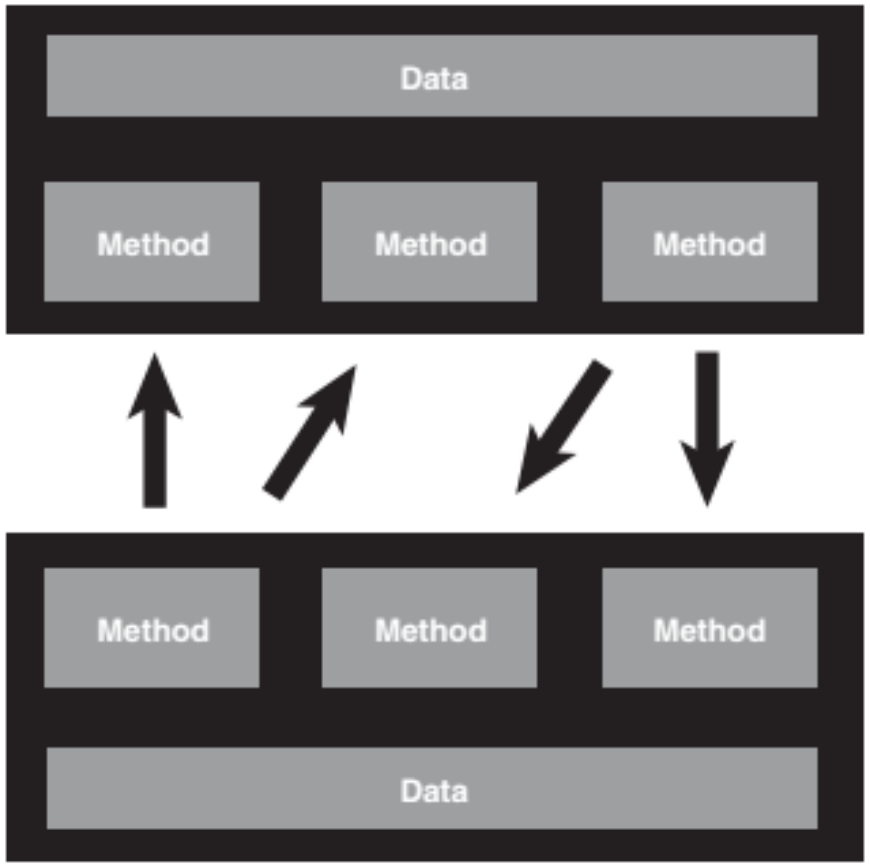
- behaviors are referred to as ***methods***

Access to attributes and methods -- ***members*** -- of an object can be controlled

> "Restricting access to certain members is called ***data hiding***"

no restriction on members access is a poorly OO designed

Objects communication



Objects send messages to each other

- A message = a method call

→ The caller do not care how the called method is implemented

→ Implementation of called methods can be changed without making change to callers

Object in detail

- The ***state*** of an object is represented by the data stored within the object – object attributes

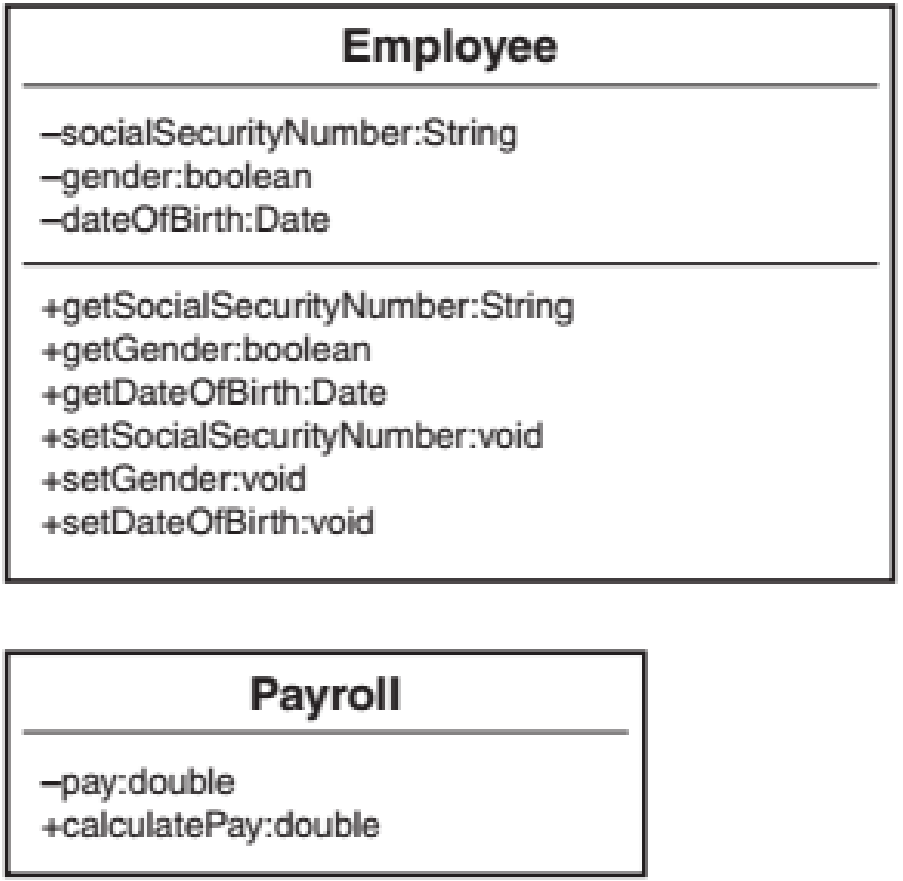
- The ***behavior*** of an object is what the object can do – object methods
  - Each attribute would have corresponding methods, such as **`getAge()`** and **`setAge()`**
  - They are the ***getter*** and the ***setter*** of the attribute

- For example:

```
private int Age;
public void setAge(int value) { Age = value;}
public int getAge() { return Age;}
```

  - The concept of getters and setters supports the concept of data hiding – providing controlled access to object's data.
    - Other object should not manipulate data within other objects

UML - Class diagram



Class

Class is a blueprint for an object

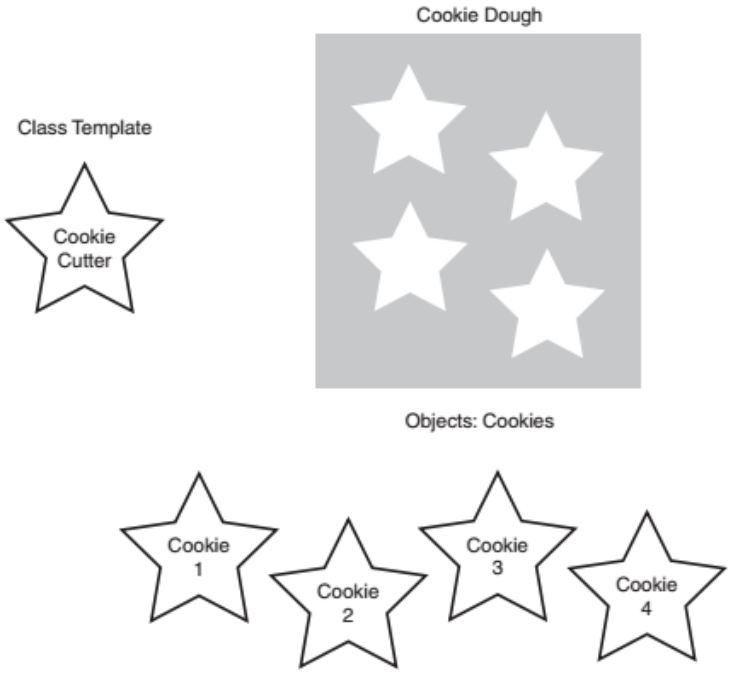
When we ***instantiate*** an object, we use a class as the basic to build the object.

→ An object cannot be instantiated without a class

→ The class comes first, then objects

Classes can be thought of as the templates

Classes ***define*** attributes and methods that ***all*** objects created with the class will posse



Class

```
public class Person{

    //Attributes
    private String name;
    private String address;

    //Methods
    public String getName(){
        return name;
    }
    public void setName(String n){
        name = n;
    }

    public String getAddress(){
        return address;
    }
    public void setAddress(String adr){
        address = adr;
    }

}
```

Access Designation

When a data or a method is defined as ***`public`***,
other objects can directly access it

When a data or a method is defined as ***`private`***,
only that specific object can access it

Another access ***modifier*** is ***`protected`***

We will learn about access modifiers later

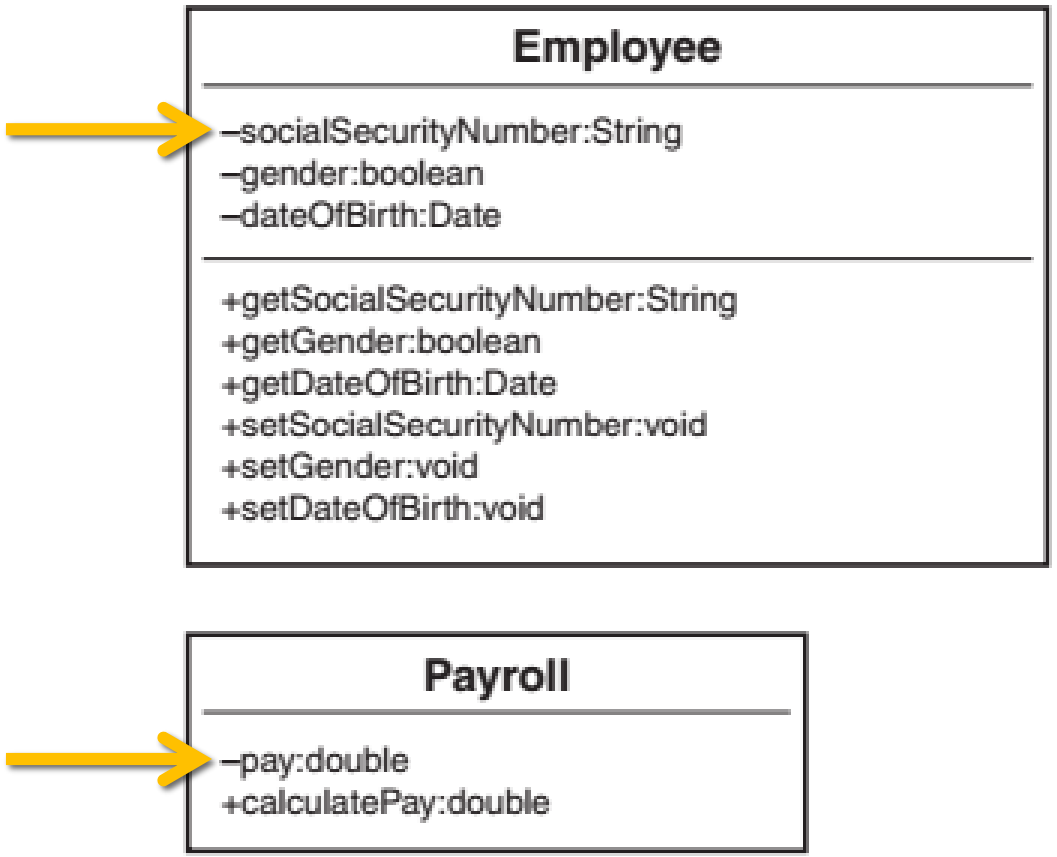
Messages

Object A ***sends a message*** to Object B:
Object A ***invokes*** a method of Object B

Object B's response is defined by
the return value of the method

Only *public* methods can be invoked

Class diagram, again



Encapsulation and data hiding

One of the primary advantages of using objects

Object need not to reveal all its attributes and behaviors

Object should only reveal the ***interfaces*** the other objects must have to interact with it.

Interface

The class design specifies the interfaces

Methods that are part of the interface as designated as **public**

For hiding data, all attributes should be declared as **private**

The interface describes how users of the class interact with the class

Interface/implementation paradigm

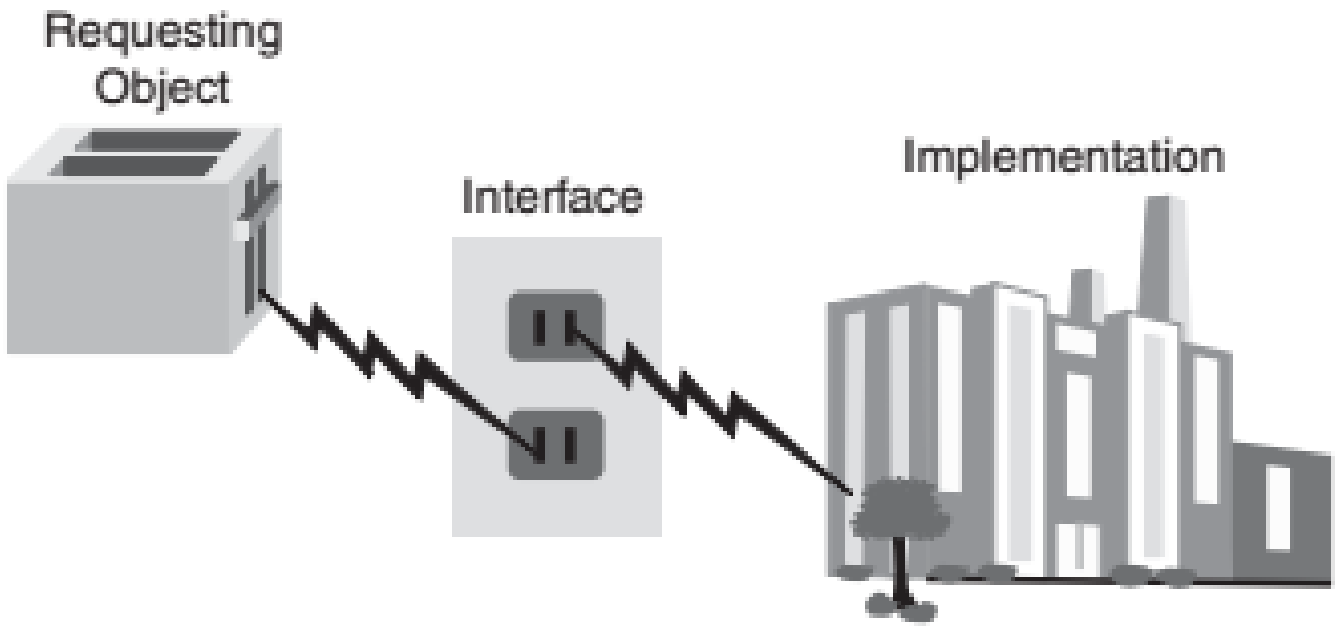
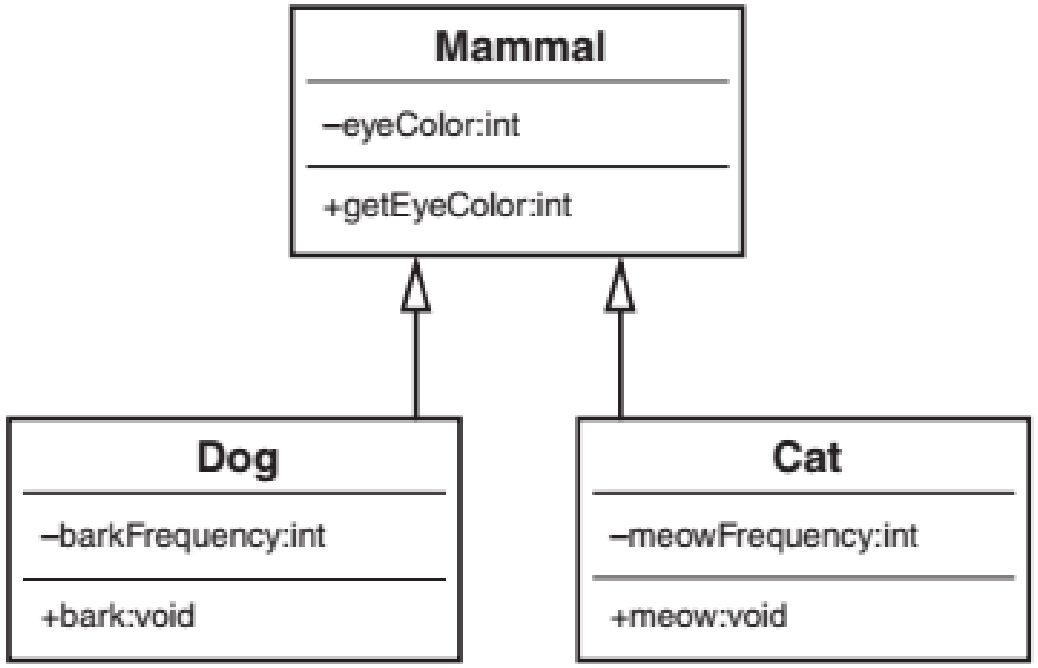


Figure 1.12 Power plant example.

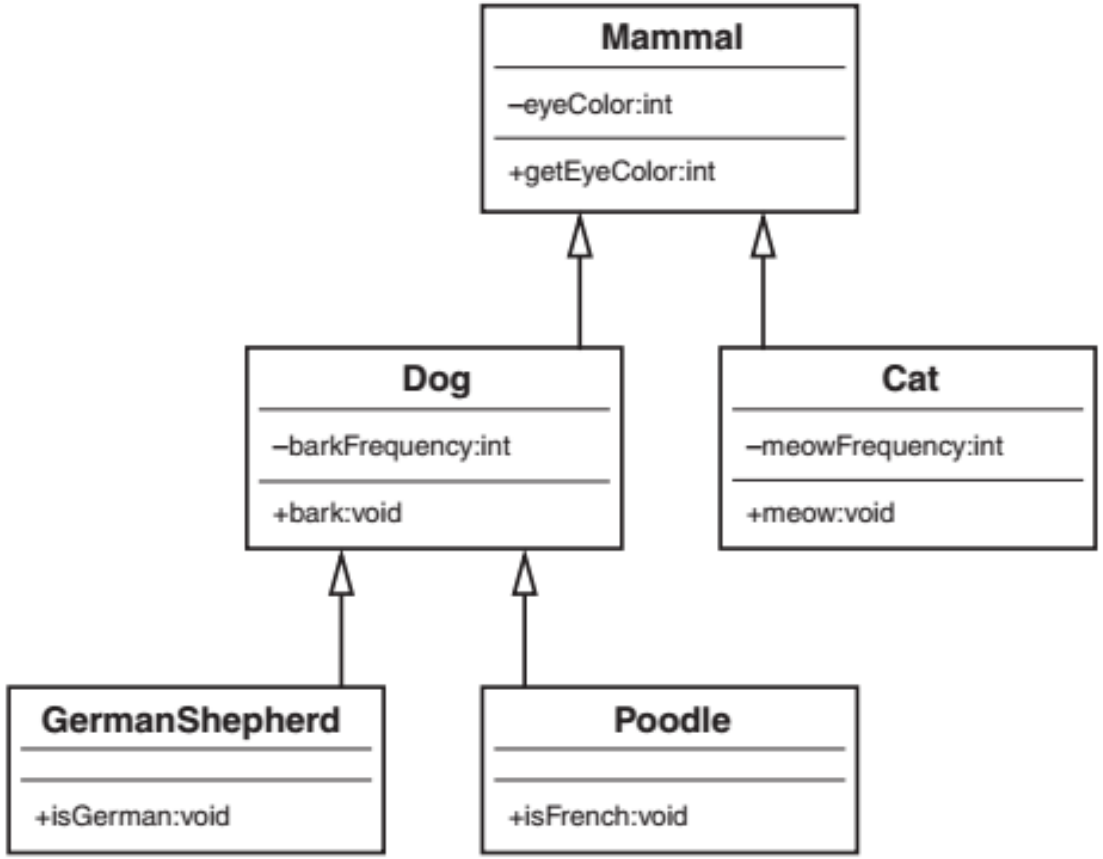
Inheritance

Inheritance provides one of the most powerful features of OOP: ***Code reuse***

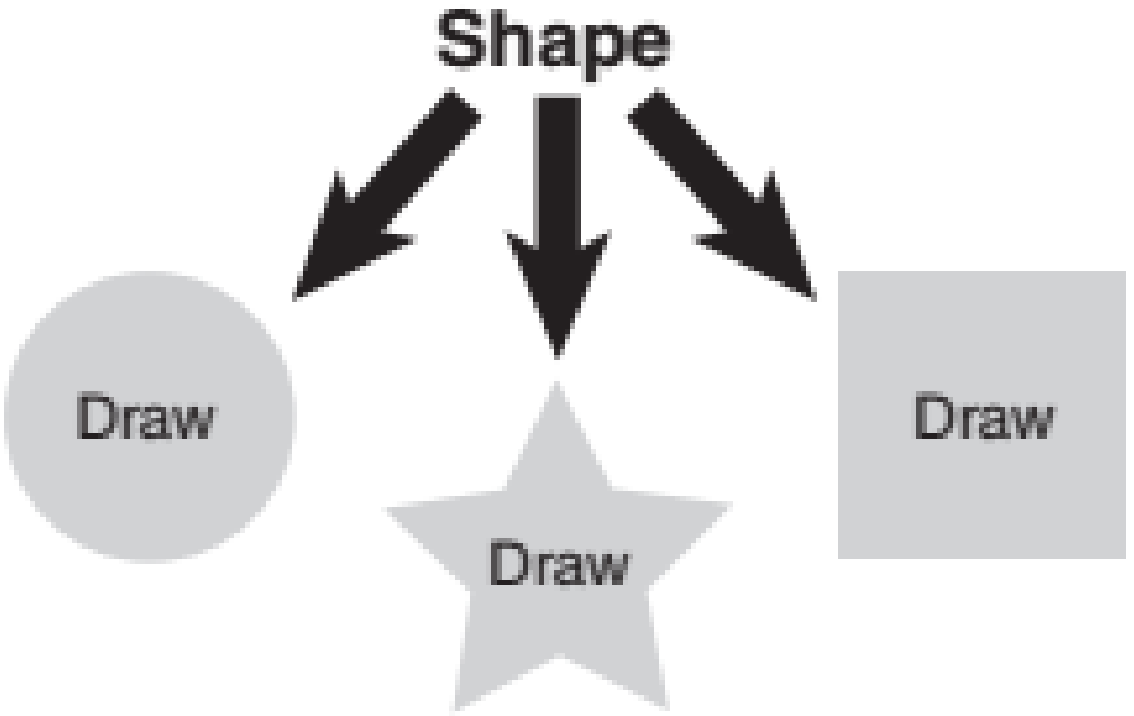
Inheritance allows a class to ***inherit*** the attributes and the behaviors of another class



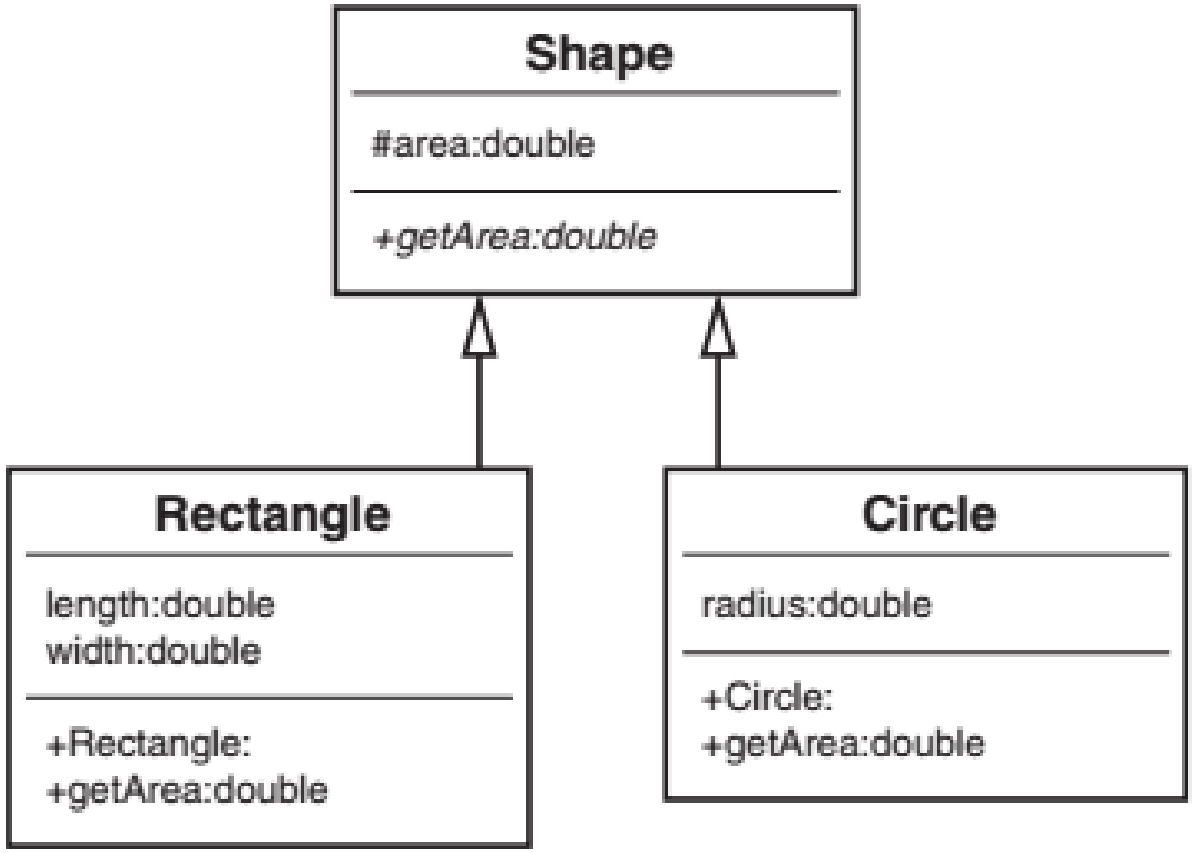
Inheritance



Is-a Relationship



Polymorphism



Composition – has-a relationship

Object can contain other objects

For example:

- A computer object contains a video card, a keyboards, and drives

In OOP, composition:

objects are built, or composed from other objects

Conclusion

- Class – Object
- Attributes – Methods
- Encapsulation
- Data hiding and access modifiers
- Interface – implementation
- Polymorphism
- Inheritance and is-a relationship
- Composition and has-a relationship