Object oriented Analysis &Design
面向对象分析与设计

Lecture_01 面向对象概述

主讲：姜宁康 博士

日期：2018/11/7

- 1.6、 “面向对象”思考方式的核心特征（一）
  - 封装 Encapsulation
  - 继承 Inheritance
  - 多态 Polymorphism

1）封装 Encapsulation

- Encapsulation
  - is the process of hiding the implementation details of an object隐藏了一个对象的实现细节
  - The internal state is usually not accessible by other objects 内部的状态也不为其他对象所访问
  - The only access to manipulate the object data is through its interface 对象的数据只能通过接口去访问
- Encapsulation allows objects to be viewed as ‘black boxes’ 封装使得对象可以被看成一个“黑盒子”
- It protects an object’s internal state from being corrupted by other objects. 保护数据
- Also, other objects are protected from changes in the object implementation. 一个对象实现方法的改变，不影响其他相关对象
- Communication is achieved through an ‘interface’ 对象间通过“接口”进行通信

1）封装 Encapsulation

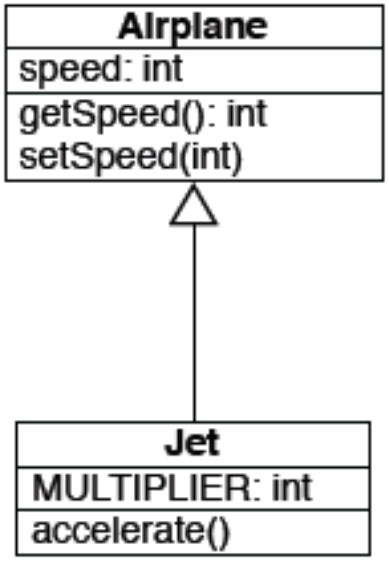
- 要封装什么？
  - 内部的、不想让其他人随意了解的信息
  - 可以封装类的属性，如，“人” 这个类，封装个人的工资、资产、年龄等信息
  - 可以封装类的方法，如，“人”如何赚钱（）？如何消磨时间（）？
- 我们为什么要封装
  - 保护隐私
  - 保护数据安全
  - 隔离复杂度 （内部实现细节不对外公开）。如“空调”，封装了制冷的过程，对人提供了一个制冷的按钮
- **面向对象的封装有四种方式（可见性）**
  - **Public**
  - **Private**
  - **Protected**
  - **Package**

1）封装 Encapsulation

- Rule:
  - An object should only reveal the interfaces needed to interact with it. Details not pertinent to the use of the object should be hidden from other objects.
    - This is a general rule of encapsulation
- Suggestion:
  - Getters and Setters
    - supports the concept of data hiding
    - Because other objects should not directly manipulate data within another object
- For example,
  - if a user needs access to an attribute,
    - a method is called to return the attribute (a getter)

2）继承 Inheritance

- The basic principle is simple:
  - A class gets the state and behaviour of another class and adds additional state and behaviour




```
1 public class Airplane {
2
3   private int speed;
4
5   public Airplane(int speed) {
6   this.speed = speed;
7 }
8
9 public int getSpeed() {
10    return speed;
11 }
12
13 public void setSpeed(int speed) {
14    this.speed = speed;
15 }
16
17 }
```

2）继承 Inheritance

```
1 public class Jet extends Airplane {
2
3 private static final int MULTIPLIER = 2;
4
5 public Jet(int id, int speed) {
6 super(id, speed);
7 }
8
9 public void setSpeed(int speed) {
10 super.setSpeed(speed * MULTIPLIER);
11 }
12
13 public void accelerate() {
14 super.setSpeed(getSpeed() * 2);
15 }
16
17 }
18
```

Note:
extends keyword indicates inheritance
super() and super keyword is used to refer to superclass

No need to define getSpeed() method; its inherited!

setSpeed() method overrides behavior of setSpeed() in Airplane

subclass can define new behaviors, such as accelerate()

2）继承 Inheritance

- How do you express inheritance in C++?

- 例

```
class Car : public Vehicle {
 public:
   ...
 };
```

We state the above relationship in several ways:

- Car is "a kind of a" Vehicle
- Car is "derived from" Vehicle
- Car is "a specialized" Vehicle
- Car is a "subclass" of Vehicle
- Car is a "derived class" of Vehicle
- Vehicle is the "base class" of Car
- Vehicle is the "superclass" of Car

3）多态 Polymorphism

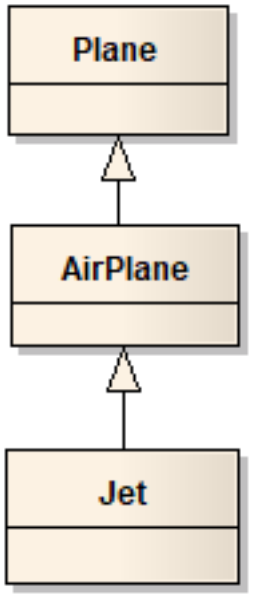
- 本意:有多种形态 “Having many forms”
  - “When one class inherits from another, then polymorphism allows a subclass to stand in for the superclass.” 当一个类从另一个类继承而来，多态使得子类可以代替父类
  - The sender of a stimulus doesn’t need to know the receiver’s class消息发送方不需要知道消息接收方属于那个子类
  - Different receivers can interpret the message in their own way同一类族的接收者可以按自己的方式处理消息
- 推论Consequence
  - different receivers can response to the same stimulus based on their interpretation同一类族的接收者可以按自己的方式处理同一个消息
  - So we can have a variety of objects who process the same piece of data in different ways.有多种对象可以按自己的方式处理相同的数据

3）多态 Polymorphism

- Implication: both of these are legal statements
  - Airplane plane = new Airplane()
  - Airplane plane = new Jet()

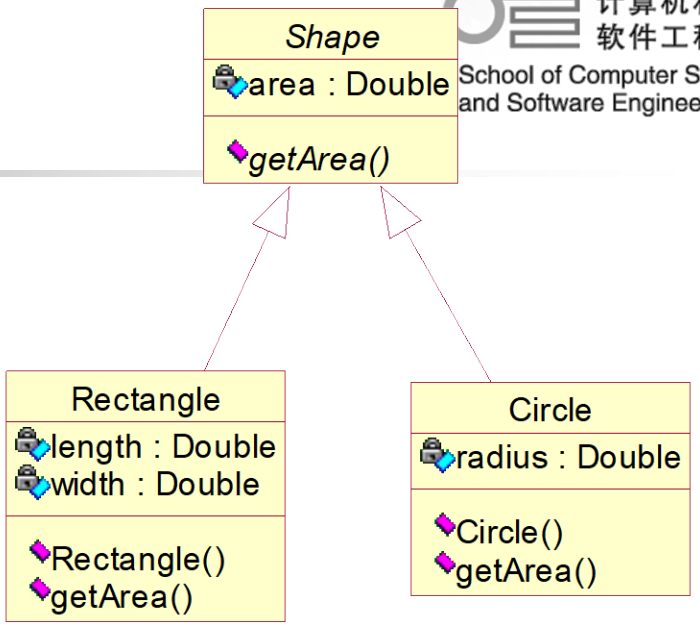
这是多态的核心思想，是设计模式的基础！

“使用指向父类的指针或者引用，能够调用子类的对象”



3）多态 Polymorphism


```
public abstract class Shape{
   private double area;
   public abstract double getArea();
}
```




```
public class Circle extends Shape{
 double radius;
   public Circle(double r) {
      radius = r;
   }
    public double getArea() {
        area = 3.14*(radius*radius);
        return (area);
    };}
```

```
public class Rectangle extends Shape{
   double length;    double width;
    public Rectangle(double l, double w){
        length = l;         width = w;
    }
    public double getArea() {
        area = length*width;
        return (area);
    };
}
```

```
//instantiate the Shape classes
Circle circle = new Circle(5);
Rectangle rectangle = new Rectangle(4,5);
```

```
//add these Shape classes to the stack:
stack.push(circle);
stack.push(rectangle);
```

```
//empty the stack, and do not have to care //about what kind of Shape classes are in it:
while ( !stack.empty()) {
   Shape shape = (Shape) stack.pop();
   System.out.println ("Area = " +
            shape.getArea());
}
```

- 本讲结束