

## 2.1 OOP设计方法及特征

### 结构化（面向过程）程序设计

- 20世纪70年代
- 是一种自上而下、逐步细化的模块化程序设计方法。
- N.Wirth的观点：算法+数据结构=程序

### 面向对象程序设计

- 20世纪80年代，面向对象程序设计：
  - 运用以对象作为基本元素的方法，用计算机语言描述并处理一个问题。
  - 程序由一系列对象组成。
- 对象
- 对象可以是人们要研究的 **任何事物**，包括具体的（如人、计算机、程序、直线）和抽象（如思想、规则、计划）。
- 每一类对象都有 **属性**（如大小、形状、重量）和 **行为**（行走、转弯、捕食）。
- 类：同类对象 **公共的** 属性和行为。

### 类与对象

- 类是具有相同数据结构（属性）和相同操作功能（行为）白的对象的集合，它规定了这些对象的公共属性和行为方法。
- 对象是类的一个实例，例如，汽车是一个类，而行驶在公路上的车辆则是一个对象。
- 抽象与具体的关系



Mr Lin is human.  
Or  
Human is Mr Lin.

### 例子：类和对象

```
class CHumanKind {
private: //人类的属性
    char name[20];
    long ID;
    char sex;
public: //人类的行为 (方法)
    void eat();
    void wear();
    void reside();
    void traffic();
};
```

```
class CDog {
private: //狗的属性
    char name[20];
    char variety[20];
    char color[12];
    float weight;
public: //狗的行为 (方法)
    void eat();
    void run();
    void jump();
    void speak();
};
```

```
void demo1() {
    CHumanKind human1; // BLX
    CDog dog1; // WC

    human1.wear();
    dog1.speak();
}
```

## 基本特征：抽象abstract

- 抽象是指对一类对象进行概括，抽出它们共同的性质并加以描述的过程。

抽象应结合业务



```
class CChinchillas {
private:
    char name[20];
    char area[20];
    char color[12];
    float weight;
public:
    void eat();
    void sleep();
    void run();
    void jump();
    void speak();
};
```

## 基本特征：封装(encapsulation)

- 封装是将抽象得到的属性数据和行为代码 **结合**，形成一个具有类特征的整体。
- 封装避免了外部与对象之间的 **影响**。
  - 对象无法影响外部数据。
  - 外部对对象的修改只能通过public方法。

```
class CDog {  
private:  
    char name[20];  
    char variety[20];  
    char color[12];  
    float weight;  
public:  
    void run();  
    void jump();  
    void speak();  
};
```

外部无法访问

外部接口

## 基本特征：继承(inheritance)

- 继承是指一个新类可以从现有的类派生而来，很好地解决了软件的可重用性问题。

## 基本特征：多态 (polymorphism)

- 多态性是指类中具有相似功能的不同函数使用同一个名称来实现，并允许不同类的对象对同一消息作出不同的响应。

```
1  #include<bits/stdc++.h>  
2  using namespace std;  
3  class Solution  
4  {  
5  public:  
6      int sum(int num1, int num2)  
7      {  
8          return num1 + num2;  
9      }  
10 } s;  
11 int main()  
12 {  
13     // int a;  
14     int a = s.sum(2, 4);  
15     cout<<a;  
16     return 0;
```

```
17     }  
}
```

## V2

```
1  #include <iostream>  
2  using namespace std;  
3  class Solution  
4  {  
5  public:  
6      int sum(int num1, int num2)  
7      {  
8          return num1 + num2;  
9      }  
10 };  
11 int main()  
12 {  
13     Solution s;  
14     cout<<s.sum(1,2);  
15     return 0;  
16 }
```