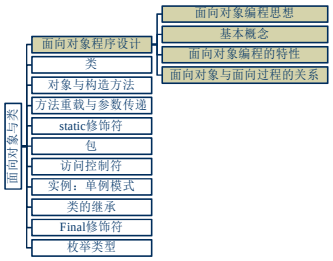
第四章 面向对象与类(1)

第四章：面向对象与类

面向对象与类
- 4.1面向对象程序设计
- 4.2类
- 4.3对象与构造方法
- 4.4方法重载与参数传递
- 4.5static修饰符
- 4.6包
- 4.7访问控制符
- 4.8实例：单例模式
- 4.9类的继承
- 4.10Final修饰符
- 4.11 枚举类型

第四章：面向对象与类



4.1面向对象编程思想

面向对象是一种程序设计方法，用于解决面向过程设计面对复杂的大型软件业务时存在的问题：

- **与人类惯用思维不一致**
  面向功能和面向客体
- **软件难于维护和扩展**
  功能或行为不稳定
- **可重用性不足。**

面向对象编程思想

- **面向对象是一种程序设计方法：直接以问题域中的事物（客体）为中心来观察和分析问题。**将待解问题直接用一个个相互作用、相互驱动的对象来表示，它引入类的概念实现更高一级的抽象和封装。
- **其他面向对象思想：设计模式、敏捷编程、软件重构**
- **面向对象包括OOA(分析)+OOD（设计）+OOP（编程）**
  - **从现实世界对象作为起点进行分析，（根据用户需求）**
  - **抽象到计算机世界的类型（封装）**
  - **并整合重现为计算机内的交互对象 （相互传递消息）系统。**

4.1.2基本概念

面向对象概念：

（1）系统中一切事物皆对象（object）。
（2）相同或类似的对象归为一个类型(class)。
（3）在类型之间可能有继承关系(inherited)和组合关系。
（4）对象之间可以互通消息(message)，形成动态联系。

4.1.2基本概念

1、对象（Object）
可以从不同的角度来理解对象。
（1）从系统分析者角度，对象是现实生活中客观世界的实体或概念，具有确切功能，并能够为其他对象提供服务。

（2）从开发者角度，对象是由数据（事物的属性）和作用于数据的操作（事物的行为）构成的整体。

（3）从使用者角度，软件系统中的每个对象对应着现实世界中的具体对象。

4.1.2基本概念

2、类(Class)
类是对某一类事物的抽象描述，它是具有相同特征的多个对象的模板，强调的是对象间的共性。

不同类之间可能存在一定的关系，通常有以下两种：
（1）特殊与一般的关系：类A是类B的一种（is-a）。例如猫是动物的一种。
（2）包含关系：类A中有一个类B（has-a），例如汽车中有引擎。

4.1.2基本概念

消息（Message）：对象之间通过发送消息建立动态联系。

- 消息包含三部分内容：
  （1）接受消息的对象。
  （2）消息的名称，即要完成哪个操作。
  （3）操作所需要的附加信息，即传入方法的参数。
- 向对象发送消息相当于调用该对象的方法。
  如 System.out.println(“This is a test”)。
- 。

4.1.3 面向对象编程主要特征

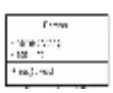
特点：
1封装：实现数据隐藏，将对象的使用者和设计者分开
2继承：可支持代码的重用和扩展
3 多态：单一的接口或方法具有不同的动作。

好处：
1 稳定性好 （stabilization）
2 重用性好 （reusability）
3 维护性好 （maintainability）

4.2 面向对象基本特征：封装、继承和多态

封装

- 它将对象的属性和行为封装起来构成新的类型，并隐藏内部实现细节，只向用户提供对象的外部可调用操作，如图4-1所示，这就是封装思想。
- **封装最大的好处是降低了软件系统的耦合程度。实现了代码的可重用性和可维护性。**



4.2 封装、继承和多态

继承

继承性是类与类之间的一种关系，通过继承，可以在无需重新编写原有类的情况下，实现代码的扩展和重用。
- **具有父类的全部属性和行为**
- **能对继承的属性和行为进行修改和扩充。**
- **极大提升了可重用性和可维护性**

4.2 封装、继承和多态

多态：指类的某个行为具有不同的表现形式。

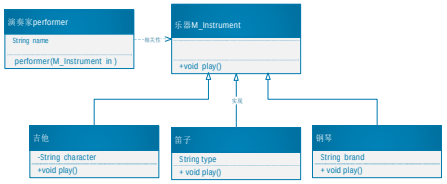
1、单个类：多态性在单个类中表现为方法重载：一个类可以有多个名字相同、形参列表不同的方法，在使用时由传递给它们的实参来决定使用哪个方法。

Tool 类

+void show(int num, String str)
+void show(int num)
+void show(String str)

3、多态

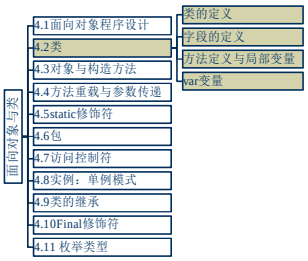
2、在多个类中主要表现为继承结构中的方法覆盖：父类和子类中具有相同的方法头，不同的代码实现，运行时再决定调用哪个方法。



4.1.4面向对象和面向过程思想的关系

- 具体实现类方法时，仍然会用到面向过程的思维方式
- 面向对象如果离开了面向过程，就无法从抽象思维层面落实到实现。

4.2 类



4.2.1 基本类的定义

。定义形式如下：

```
[访问权限修饰符][其他修饰符] class 类名 [extends 父类名] [implements 接口列表]
{
[访问权限修饰符][其他修饰符]  [字段]
[访问权限修饰符][其他修饰符]  [初始化块]
[访问权限修饰符][其他修饰符]  [构造方法]
[访问权限修饰符][其他修饰符]  [普通方法]
}
```

举例--类的定义

```
class Rectangle {
 private int width=0, height=0, area=0;    //成员变量
 public Rectangle(int w, int h){     //构造方法
      width = w;
     height = h;
     area=w*h;
 }
 public int getArea(){
      return area;
 }
 public void drawRect() {
  for (int i = 0; i < height; i++) {
     for (int j = 0; j < width; j++)
        System.out.print("*");
     System.out.println();
     }
 }}
```

4.2.2 字段定义

- 其定义的基本形式：

[访问权限修饰符][其他修饰符]数据类型 变量名1[=初值1]，…变量名n[=初值n];

如：int width;

- 作用范围：是整个类体，不受定义位置限制，且无需显式赋值，可以被类中的方法访问。

- 注意：

对于字段，如果定义时不赋初值，系统会自动赋一个默认值，

数值类型为0/0.0，

boolean类型为false,

字符类型为 ‘\u0000’,

引用类型为null。

4.2.3 方法定义与局部变量

1、定义方法

方法描述了对象所具有的行为，必须先定义后调用。定义形式如下：

[修饰符] 返回值类型 方法名([类型1 形参1, 类型2 形参2,…]) [throws 异常列表]{

方法体

}

说明：

（1）返回值类型：若无返回值则为void，若有返回值，用return 返回结果。

（2）形参列表：方法头中的变量称为形式参数，简称形参。形参就像是占位符，调用方法时会给参数传递一个值，这个值称为实际参数，简称实参。当然，方法中可以不包含参数。方法名和参数列表一起构成方法签名。

（3）throws异常列表列出在方法执行中可能出现的异常。

4.2.3 方法定义与局部变量

2、局部变量

局部变量声明在方法内部，而且编程人员必须对它进行初始化操作。

- 定义形式为：

[final] 数据类型 变量名1[=初值1]，…变量名n[=初值n];

- 说明：

局部变量存于栈中的，它随方法的调用而产生，方法结束则消失。具体可分为以下三种：

（1）方法体内定义的变量：作用域从声明处到方法结束。

（2）方法的形参：作用域为整个方法体

（3）语句块中声明的变量：作用域从声明处到语句块结束。

。

4.2.3 方法定义与局部变量

3、局部变量隐藏字段

如果方法中的局部变量与字段（成员变量）同名，根据就近原则，该方法访问此变量时，直接访问的是局部变量，而非字段。如：

```
class temp{
        int i=0; //成员变量
        void showInfo(){
                int i=30;//局部变量
                System.out.println("i is :"+i);
 }
}
```

【例4.2】方法、变量的定义和作用域

上述代码有意设计的较为复杂，实际应用中通常不会这样设计。

```
class FieldDemo{
        public int compare(int a){ //合法
           if (a>var1)
                return 1;
           else if (a==var1)
                return 0;
           else  return -1;
           }
        public void showInfo(){
           int var1=30;
           System.out.println(a); //非法,a是方法compare(int a)的局部变量
           System.out.println("var1 is :"+var1);//合法，隐藏字段，var1=30
           System.out.println("var1 is :"+this.var1);//合法，显示的是字段var1
           {
             int var1=40;  // 非法，同一方法中，即使语句块嵌套，也不能定义同名变量
             int var2=50;
             System.out.println("var2 is:"+var2); //合法，在var2的有效范围内
           }
           System.out.println("var2 :"+var2); //非法，超出了var2所属语句块范围。
          }
        //合法，字段作用域不受定义位置限制
         private  int  var1=(int)(Math.random()*10);
 }
```

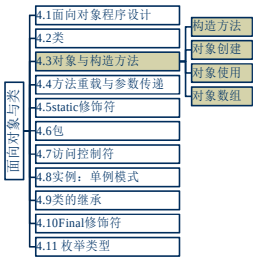
4.2.4var局部变量

var相当于一种动态类型，是Java10新特性，用它来定义局部变量。

- var 定义变量的语法： var 变量名 = 初始值;
- 如：var a = 20;
- var类型是编译器根据变量所赋的值来推断类型，适用于
  - （1）只能用于局部变量上，不允许定义类的成员变量。
  - （2）声明时必须初始化，不能在以后再赋初始值。
    - 例如 var a ; a=5 / /错误
  - （3）不能用作方法参数。
  - （4）var变量如果赋给一个返回值不直观的表达式时，不能用var定义变量。
    - 例如： var a={1,2,3} / /错误，因为var需要一个显式的目标类型

```
public class Test {
    public static void main(String[] args) {
      for(var v:args)
      System.out.println(v);
    }
}
```

4.3 对象与构造方法



4.3.1 构造方法

- 构造方法是一种特殊的方法，它专门用于创建对象，完成对象的初始化工作。
- 构造方法有以下特殊之处。

（1） 构造方法的方法名与类名相同。
（2） 构造方法没有返回类型，也不能有void。
（3） 构造方法用new操作符调用，主要作用是初始化对象。
（4） 在Java中，每个类都至少有一个构造方法，如果没有显示地定义构造方法，Java会自动提供一个缺省的构造方法(形参列表为空，空实现体)。

4.3.1 构造方法

1．构造方法的定义
构造方法形式如下：
[修饰符] 方法名（[形式参数列表]）[throws异常列表]{
方法体
}
如：在Sample类中定义一个构造方法如下：

```
public Sample(int a){
        System.out.println("My Constructor");
        }
```

4.3.1 构造方法

2．默认的构造方法

- 如果类中没定义构造方法，系统会自动为用户提供一个无参的默认构造方法，确保每个Java类都至少有一个构造方法，该构造方法是空操作，

- 默认构造方法生成规则如下：

（1）若类中不含任何构造方法，则系统提供一个默认构造方法
（2）只要在类中编写了任何一个构造方法，系统就不会提供默认构造方法。

4.3.1 构造方法

例如：

```
public class Sample1{
    public static void main(String arg[]){
        Sample1  s1=new Sample1();
        }
    }
```

```
public class Sample2{
    public Sample2(int a){
        System.out.println("My Constructor");
        }
    public static void main(String arg[]){
        Sample2  s1=new Sample2();  //非法，因为不存在无参构造方法
        Sample2  s2=new Sample2(3);
        }
    }
```

4.3.2 对象创建

1. 对象声明

类是一种引用类型，声明的对象变量称为引用（reference）变量，但声明一个引用变量时，并没有对象生成，声明形式如下：

类名 对象名列表;

例如：

Rectangle rec1, rec2;

Rectangle rec3=null //空引用null，不指向任何对象

4.3.2 对象创建

2. 对象创建

- 使用new操作符调用构造方法创建对象，具体格式如下：

new 构造方法（[参数]）；

其作用是在内存中为此对象分配内存空间，并返回对象的引用（相当于对象的存储地址）。

例如：new Rectangle(12, 6);

- 我们可以将对象声明与创建过程合而为一。具体形式如下：

类名 引用变量=new构造方法（[参数]）；

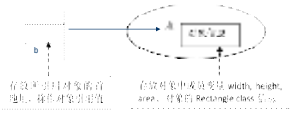
例如：

Rectangle rec1= new Rectangle(12, 6);

4.3.2 对象创建

2. 对象创建

Rectangle rec1= new Rectangle(12, 6);



在没有开启逃逸分析的情况下，对象实体都是分配在堆上（逃逸分析可参看附录）。

4.3.3 对象使用

1、访问对象成员

创建新的对象之后，可以通过分量运算符“.”来访问对象成员

注意：构造方法不能通过分量运算符调用。

- 具体形式如下：

对象引用.成员变量

对象引用.方法名(实际参数列表)

- 例如：Rectangle r1=new Rectangle(12, 6);

r1.drawRect();

4.3.3 对象使用

方法调用时需注意

[对象名].方法名（实参1，实参2……）

- 调用方法的方法称为主调方法，被调用方法为被调方法。主调和被调方法不在一个对象里则需要显式说明被调对象名。
- 方法调用时，形参和实参的个数要一致。类型方面，形参和实参要兼容，若无法自动转换，被视为语法错误。

求 sum(n)=n! +n²+1

```
class Cal{
    private long fac(int end) {
        long mul=1;
        for(int i=1;i<=end;i++)
            mul=mul*i;
        return mul;
    }
    public long sum(int n) {
        return n*n+1+ fac(n);
    }
}
public class TestCal{
    public static void main(String arg[]) {
        Cal demo=new Cal();
        long result=demo.sum(7);
        System.out.println("sum(7) is :"+result);
    }}
```

4.3.3 对象使用

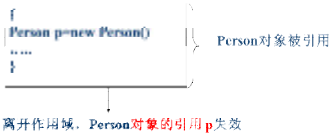
- 注意：空对象
  当对象引用指向一个空的对象，引用值为null，程序无法通过这个引用调用对象属性和方法。

  如：
  Cal demo;
  demo.sum(7);
  会发生编译错误。

4.3.3 对象使用

2、引用及对象的生命周期

- 当引用变量离开它的作用域（定义它的大括号“{}”）则会失效，
- 但如果所指对象还有其他引用指向它，则对象还可以继续被使用，
- 可如果没有任何引用指向一个对象时，对象将成为垃圾对象，不能再被使用，**JVM提供了垃圾回收机制，根据一定的策略回收垃圾对象**。



对象引用及对象作用域举例

```
class Person{
 String name;
 private Person(String name){
 this.name=name;
}
 void showInfo(){
  System.out.println("my age is :"+name);
 }
 public static Person getP(String name){
   //ptemp作用域在getP方法中
   Person ptemp=new Person(name);
   return ptemp; //引用值被传递出去
}}
public class TestPerson{
 public static void main(String arg[]){
 Person p=Person.getP("张三");
 p.showInfo();
}}
```

4.3.3 对象使用

3、对象的比较

- “==” 用来判断运算符两边是否是引用的同一个对象，即比较的是对象引用值（或对象实体的地址）。
- 要比较两个对象的内容是否相等，即对象实体值，必须在类里实现专门的方法，所有类的父类Object中有一个方法equals()。

```
String str1=new String("China");
String str2=new String("China");
System.out.println(str1==str2); //返回false
System.out.println(str1.equals(str2)); //返回true
```

4.3.4 对象数组

- 对象数组就是一个数组中的所有元素都是对象，声明对象数组与普通基本类型数组一样.

```
Person[] p=new Person[4];
```

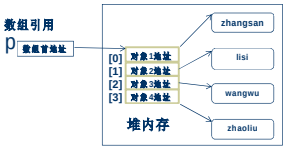
- 因对象数组中每个元素都是对象，所以每个元素都需单独实例化（还需用new实例化每个元素）

4.3.4 对象数组

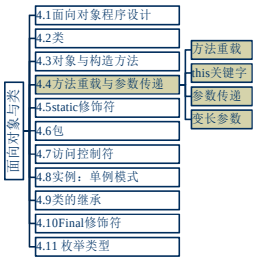
```
Person[] p;
p=new Person[4];
```

如果不赋值，p[0]…p[3]开始时值为NULL

```
p[0]=Person.getP(“zhangsan”);
p[1]=Person.getP(“lisi”);
p[2]=Person.getP(“wangwu”);
p[3]=Person.getP(“zhaoliu”);
```



第四章：面向对象与类



4.4.1 方法重载

1. 什么是方法的重载

- 定义方法时使用相同的方法名，不同的形参列表，叫方法重载（overloading）
方法重载是实现“多态”的一种方法。
- 形参列表不同是指参数个数不同，或者对应位置上参数类型不同。
- 重载方法返回类型、修饰符可以相同，也可不同，它不决定是否是重载方法。
- 例如 protected int add(double x, double y) {return x+y;}
public static double add(int x, int y) {return x+y;}

```
public class Person{
    private String name;
    private int age;

    Person(String n,int a){
        name=n;
        age=a;
    }
    void sayHello(){ //不带参数的sayHello()方法
        System.out.println("Hello! My name is "+name);
    }
    void sayHello(Person p2){   //带参数的sayHello()方法
       System.out.printf("Hello %s,My name is %s",p2.name, name);
    }
    public static void main(String args[]) {
       Person per1=new Person("zhang san",20);
       Person per2=new Person("li si",30);
       //调用重载的两个sayHello（）方法
       per1.sayHello();
       per1.sayHello(per2);
    }}
```

【运行结果】
Hello! My name is zhang san
Hello li si, My name is zhang san

4.4.1 方法重载

1. 什么是方法的重载

- 定义方法时使用相同的方法名，不同的形参列表，叫方法重载（overloading）
方法重载是实现“多态”的一种方法。
- 形参列表不同是指参数个数不同，或者对应位置上参数类型不同。
- 重载方法返回类型、修饰符可以相同，也可不同，它不决定是否是重载方法。
- 例如 protected int add(double x, double y) {return x+y;}
public static double add(int x, int y) {return x+y;}

4.4.1 方法重载

2、定位重载函数的顺序
重载方法在被调用时，也有可能出现实参与多个方法的形参兼容的情况。

```
public class OrderTest{
     public static double add(double d, double d2){
           System.out.println("in double");
            return d+d2;
     }
     public static int add(int i, int i2){
           System.out.println("in int");
           return i+i2;
     }
     public static void main(String arg[]){
           byte b1=4,b2=5;
          System.out.println("sum is :"+add(b1,b2));
     }
     }
```

【运行结果】
in int
sum is :9

4.4.1 方法重载

2、定位重载函数的顺序

原则：
(1) 查找同名方法，没有则报错
(2) 比较形参和实参的数目是否相等，如果多个方法符合条件，这些方法进入候选集
(3)候选集中
- 如果对应位置上的每个参数类型完全匹配，为最佳方法，
- 如果无匹配的可以通过扩展转换找出最佳匹配方法，选择原则为：源类型与目标类型的距离越近越好。

4.4.1 方法重载

The Person's null, the age is 0
The Person's xiao ming,the age is 0
The Person's xiao ming,the age is 20

3、应用：构造方法重载

```
class Person2{
    private String name;
    private int age;
    Person2(){
            System.out.printf("The Person's %s, the age is %d\n",name,age);
            }
    Person2(String n){
            name=n;
            System.out.printf("The Person's %s,the age is %d\n",name,age);
            }
    Person2(String n ,int a){
            name=n;
            age=a;
            System.out.printf("The Person's %s,the age is %d\n",name,age);
            }
    public static void main(String args[]){
            Person2 per1=new Person2();
            Person2 per2=new Person2("xiao ming");
            Person2 per3=new Person2("xiao ming",20);
            }}
```

4.4.2 this关键字

this表示当前对象

- 当通过一个对象引用调用它的成员方法时，系统会将当前对象的别名this传递到被调方法中，所以，this只能在成员方法中可见。

- this关键字通常用在下面三种场合。
  （1）使用this访问对象成员
  （2）构造方法中，用this调用本类的另一构造方法
  （3）返回当前对象

4.4.2 this关键字

（1）使用this访问对象成员
特别是局部变量和成员变量重名时，利用this可以限定某个变量是成员变量。

```
class Point{
        int x,y;
        public Point(){
        }
        public Point(int x,int y){
                this.x=x;
                this.y=y;
                }
        public int getX(){
                return x; //也可以写成 return this.x
     }
        public int getY(){return y;}
  }
```

4.4.2 this关键字

（2）构造方法中，用this调用本类的另一构造方法

在一个构造方法中，调用另一个重载的构造方法：形式为：this（[实参]）。这条语句必须是构造方法的第一条语句，且只能出现一次。如：

```
  public class Person3 {
      private String name;
      private int age;
      public Person3(){
              System.out.println("in a constructor");
      }
      public Person3(String name, int age){
              this();
              this.name=name;
              this.age=age;
  } }
```

4.4.2 this关键字

（3）返回当前对象

在方法中，利用return this可以返回当前对象，从而可以继续调用该类或其子类的成员。

```
public class ThisDemo{
      ThisDemo m1() {
              System.out.println("in m1");
              return this;
      }
      void m2() {
              System.out.println("in m2");
      }
      public static void main(String arg[]) {
              ThisDemo d=new ThisDemo();
              d.m1().m2();
```

this关键字只能用在对象方法中，不能用在静态方法中，后续4.5节会说明。