Java程序设计基础

类和对象

类和对象串主要内容

- **6.1 面向对象（重点）**
- **6.2 Java 中的类（重点）**
- **6.3 构造方法（重点）**
- 6.4 对象的初始化过程（难点）
- **6.5 重载（重点）**

教学目标

- 能理解类和对象的概念
- 能准确区分类和对象的不同及联系
- 能正确使用类和对象开发程序
- 能理解封装的概念
- 能准确使用封装增加程序的安全性
- 能理解构造方法的概念
- 能正确使用this区分重名变量

教学目标

- 能正确使用this调用同类中的其他构造方法
- **能理解对象初始化时内存的演变过程**
- 能理解重载的概念
- 能正确使用重载提高程序的复用性
- **能正确使用类和对象解决实际问题**

面向对象编程

- 面向对象（面向对象编程的简称，Object Oriented Programming，OOP）是一种对现实世界理解和抽象的方法，是计算机编程技术发展到一定阶段后的产物
- **通过面向对象的方式，将现实世界的事物抽象成对象，将现实世界中的关系抽象成关联、继承、实现、依赖等关系，从而帮助人们实现对现实世界的抽象与建模**
- 通过面向对象的方法，更利于人们理解，更利于对复杂系统进行分析、设计与开发。同时，面向对象也能有效提高编程的效率，通过封装技术和消息机制可以像搭积木一样快速开发出一个全新的系统

五子棋

- 要编写一个五子棋的游戏，用面向对象的设计思路，其分析步骤如下

  棋盘部分：负责绘制基本画面以及黑棋、白棋走完后的画面

  黑棋、白棋：除了颜色不一样外，其行为是一样的

  规则部分：负责判定输赢和犯规

- 思考：如果用面向过程的设计思路，应该如何分析？

五子棋

- 总的来说:

  OOP 以对象来划分系统

  OOP 将属性和行为放入对象

  OOP 通过调用对象使其执行行为来完成功能

  OOP 使程序易于维护和扩展

类与对象的概念

- **在编程世界中，万物皆对象。例如，一辆具体的汽车、一间具体的房子、一张具体的支票、一个具体的桌子也是对象，甚至一项具体的计划、一个具体的思想都是对象。**
- **在 Java 面向对象编程中，将这些对象的属性仍然称为属性，将对象具有的行为称为方法（Mathod）。例如，教师具有姓名、性别、年龄、学历这些属性，小轿车具有品牌、颜色、价格等属性，这些属性具体的值称为属性值。教师具有讲课、批改作业等行为，小轿车具有行驶、停止、喇叭响等行为，这些行为称为方法。可见，方法是一种动态的行为，而属性是一种静态的特征**
- 属性通常是名词，行为或方法通常是动词
- 什么是类？类是对具有相同属性和相同行为的对象的抽象。

类的基本语法

- 类定义的语法形式

```
public class 类名｛
    //定义类属性
    属性1类型:属性1名;
    属性2类型:属性2名;
    …
    //定义方法
    方法1定义
    方法2定义
    …
}
```

- **类的主要内容分两部分，第一部分是类的属性定义，在前面的课程中学习过，在类内部、方法外部定义的变量称为成员变量，也可以称为成员属性，或简称为“属性”。第二部分是类的方法定义，通过方法的定义，描述类（对象）具有的动态行为，这些方法也可以称为成员方法，或简称为“方法”。**

类的基本语法

- 定义好一个类后，就可以根据这个类创建（实例化）对象了。类就相当于一个模板，可以创建多个对象。创建对象的语法形式如下。

  类名 对象名 = new 类名();

- 之后再通过．操作符来引用对象的属性和方法，具体的语法形式如下。

  对象名.属性；

  对象名.方法()；

初始封装

- 封装的目的是简化编程和增强安全性。
- 简化编程是指，封装可以让使用者不必了解具体类的内部实现细节，而只是要通过提供给外部访问的方法来访问类中的属性和方法。例如 Java API 中的 Arrays.sort()方法，该方法可以用于给数组进行排序操作，开发者只需要将待排序的数组名放到 Arrays.sort()方法的参数中，该方法就会自动的将数组排好序。
- 增强安全性是指，封装可以使某个属性只能被当前类使用，从而避免被其他类或对象进行误操作。例如在 Student.java 的程序中，Student 的 stuAge 属性是 public 的形式体现的，但这样做实际存在着安全隐患：TestStudent 类（或在访问修饰符可见范围内的其他类）完全可以随意的对 stuAge 进行修改，

封装如何保证数据的安全性

- 封装

  1 用 private 禁止其他类直接访问属性；

  2 给 1 中的属性新增两个 public 修饰的 setter 和 getter 方法，供其他类安全的访问。

  setter 方法用于给属性赋值，而 getter 访问用于获取属性的值。并且一般而言，

  setter 方法的名字通常是 set+属性名，getter 方法的名字通常是 get+属性名。

封装如何保证数据的安全性

- 示例

```
public class Student {
    private int stuAge ;
    public int getStuAge() {
        return stuAge;
    }
    public void setStuAge(int age) {
        if( age >0 && age <110)
            stuAge = age;
        else
            age = 0 ;
    }
}
```

构造方法的基本语法

- **构造方法（也称为构造函数）是一种特殊的方法，它具有以下特点。**

  构造方法的方法名必须与类名相同

  构造方法没有返回类型，在方法名前不声明返回类型（包括 void）

- 构造方法的语法形式如下

  [访问修饰符] 类名([参数列表])；

this 关键字

- this 是 Java 的一个关键字，它表示“指向当前对象的引用（简称为 ‘当前对象’）”

- 在实际开发时，this 通常用于以下两个场景。

  区分成员变量和局部变量，避免命名的烦恼。

  调用其他构造方法。

使用 this 关键字区分成员变量

- 可以通过 this 来区分名称相同的不同变量，如下所示。

```
private int age ;
...
public void setAge(int age) {
    ...
    this.age = age ;
}
```

在this.age = age ;这句代码中，因为 this 代表着当前对象，因此this.age就代表了当前对象的 age 属性；而age没有 this 修饰，就会根据变量的作用域，默认为方法参数中的 age

使用 this 关键字调用其他构造方法

- 可以使用 this 调用类中的其他构造方法，但这种this()语句必须写在构造方法的一行

```
public Student() {
    this(" WangYun" );
}
public Student(String name) {
    this(name,23);
}
public Student(String name, int age) {
    ...
}
```

- 需要注意的是，所有的构造方法之间不能循环调用，否则会出现类似“死循环”的现象。例如以上代码，如果在最后一个构造方法的第一行也加上this()，那么三个构造方法就会无限制的彼此调用。

对象初始化简介

- 对象的初始化，实际就是先在堆内存中申请一块用于存放对象属性值的空间，然后再给这些属性值赋上默认值、程序员期望的数据或者用户指定的数据。当堆内存中的这块空间有了值以后，再在栈空间中申请一块空间并存放引用变量，然后用栈中的引用变量指向堆中的对象，最终就可以通过栈中的引用变量访问或者修改堆中的对象了

对象初始化过程中内存的演变

- 阅读以下代码

```
public class Student {
    private String name = "";
    private int age = -1;
    private int sex = -1;
    private int grade = -1;
    //使用初始化块初始化
    {
        System.out.println("使用初始化块初始化");
        this.name = "雷静";
        this.age = 22;
        this.sex = 2;
        this.grade = 4;
    }
```

对象初始化过程中内存的演变

```
    //无参构造方法
    public Student() {
        System.out.println("使用无参构造函数初始化");
    }
    //构造方法，用户初始化对象的成员变量
    public Student(String name, int age, int sex, int grade) {
        System.out.println("使用有参构造函数初始化");
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.grade = grade;
    }
   //省略了Student类中的其他方法
}
```

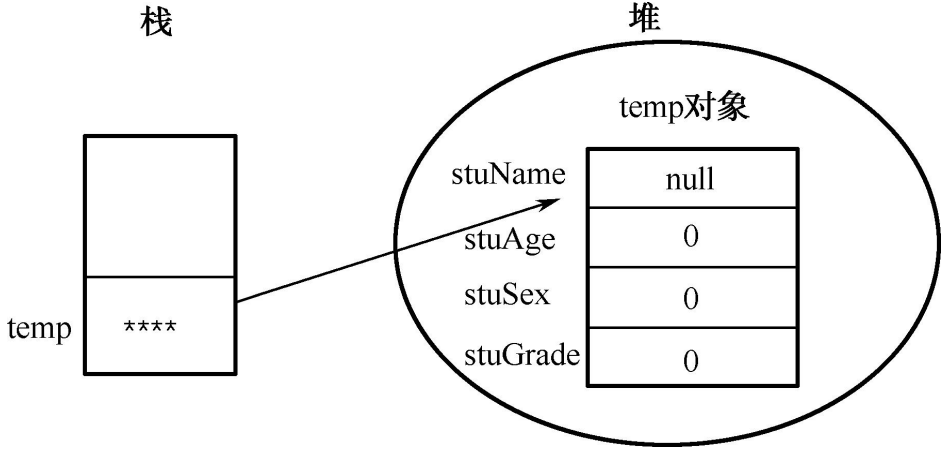
对象初始化过程中内存的演变

```
public class TestStudent{
    public static void main(String[] args){
        Student temp = new Student();
        System.out.println(temp.getName() + " 正在读大学" + temp.getGrade() + "年级");
        //构造方法初始化成员变量在初始化块执行之后
        Student wangYun = new Student("王云",22,1,4);
        System.out.println(wangYun.getName() + " 正在读大学" + wangYun.getGrade() +
"年级");
    }
}
```

对象初始化过程中内存的演变

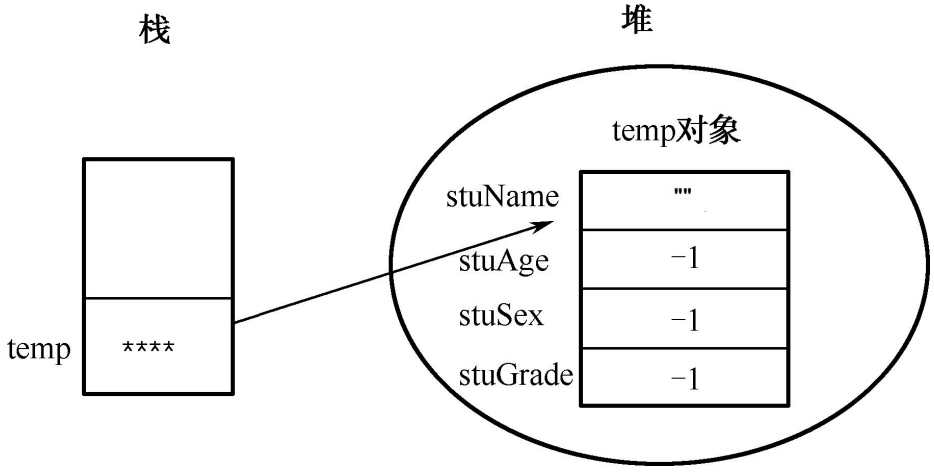
- 以上代码，在对象初始化时的内存演变过程如下。

（1）“Student temp= new Student();”执行时，首先需要在堆内存中申请空间，用于存放对象的实例，这片空间上成员变量的值全部为默认值：name的值是null、age的值是0……



对象初始化过程中内存的演变

（2）紧接着，执行声明初始化（由设计该类的开发者指定），例如在Student类中“private int age = -1;”代表程序员希望age属性用值-1覆盖默认值



对象初始化过程中内存的演变

（3）初始化块初始化。

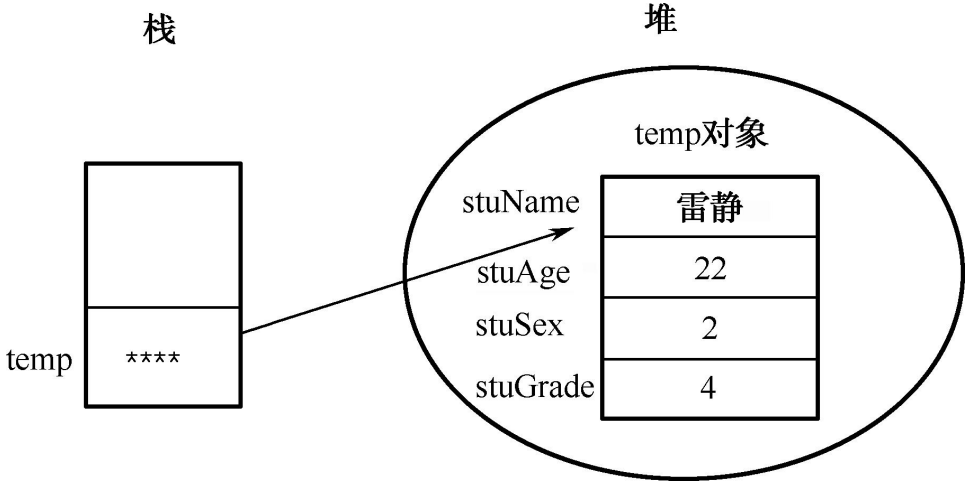
初始化块就是在类的下一级（与成员变量和成员方法同级）用一对大括号括起来的代码块，语法形式如下：

```
{
    代码块
}
```

对象初始化过程中内存的演变

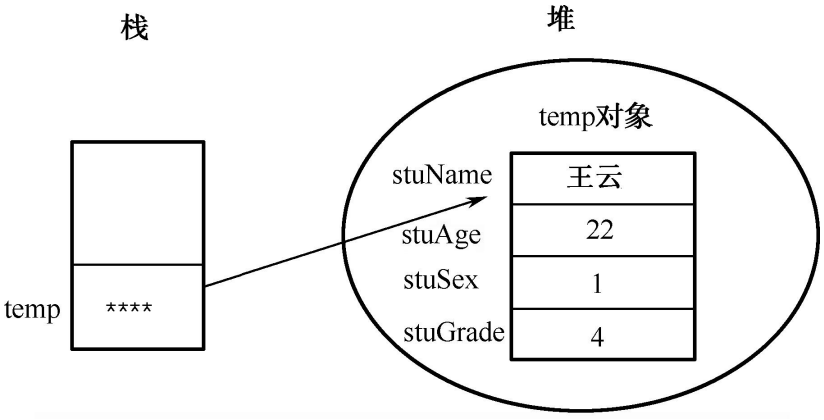
初始化块可以用来覆盖类的成员变量的值，初始化块的执行时机是发生在“声明初始化”之后，在“构造器初始化”之前。例如，程序中的如下初始化代码，就用于给成员变量再次赋值

```
{
    this.name = "雷静";
    this.age = 22;
    this.sex = 2;
    this.grade = 4;
}
```



对象初始化过程中内存的演变

（4）构造器初始化，例如“Student temp = new Student("王云", 22, 1, 4);”，在默认初始化，声明初始化，初始化块之后，再此用构造器覆盖各个属性的值



重载

- 在同一个类中，可以有两个或两个以上的方法具有相同的方法名，但它们的参数列表不同。这种形式被称为重载（overload）。所谓参数列表不同包括以下三种情形。

  参数的数量不同。

  参数的类型不同。

  参数的顺序不同。

重载

- 必须注意的是，仅返回值不同不能视为方法重载，还会得到一个语法错误。而且重载的方法之间只是“碰巧”名称相同罢了，不具备连带效应。既然方法名称相同，在使用相同的名称调用方法时，编译器怎么确定调用哪个方法呢？这就要靠传入参数的不同确定具体调用哪个方法。

- 一个类的多个构造方法被视为重载，因为多个构造方法的方法名相同，而参数列表不同，符合重载的定义。

笔试题

- 下面对于类和对象的描叙错误的是（　　）？

A 类可以产生对象

B 对象可以抽象出类

C 类是抽象的，对象是具体的

D 对类细分后的产物就是对象

笔试题

- 下面对于构造方法的描叙正确的是（　　）　？

　　A　类中必须显式定义构造方法

　　B 构造方法的返回类型是void

　　C 构造方法名和类名完全一致，并且不能带任何参数

　　D　一个类中可以定义多个构造方法

笔试题

- 在某个类中存在一个方法：void　getSort(int　x),以下能作为这个方法的重载声明的是（　　）？

A　public getSort(float x)

B　int getSort(int y)

C　double getSort(int x,int y)

D　void get(int x, int y)

笔试题

- 重写与重载的关系是（ ）？

  A 重写只有出现在父类与子类之间，而重载可以出现在同一个类中

  B 重写方法可以有不同的方法名，而重载方法必须是相同的方法名

  C final修饰的方法可以被重写，但不能被重载

  D 重写与重载是同一个概念的不同描述