

第三章 Java面向对象程序设计-1



3.1 面向对象程序设计方法概述

3.2 类与对象

3.3 对象初始化和回收

3.4 应用举例

3.1 面向对象程序设计方法概述



■ 对象 (object)

● 现实世界

- 万物皆对象
- 都具有各自的属性，对外界都呈现各自的行为

● 程序

- 一切都是对象
- 都具有标识 (identity), 属性和行为(方法)
 - 通过一个或多个变量来保存其状态
 - 通过方法(method) 实现其行为

3.1 面向对象程序设计方法概述



■ 类 (class)

- “人以类聚，物以群分”
- 将属性及行为相同或相似的对象归为一类
- 类可以看成是对象的抽象，是对象的集合，代表了此类对象所具有的**共有属性**和**行为**。
- 在面向对象的程序设计中，**每一个对象都属于某个特定的类**。

```
import java.math.*;
class Circle
{
    private double radius;

    Circle(double rads)
    {
        radius = rads;
    }
    public double getArea()
    {
        return radius * radius * Math.PI;
    }
}
```

Demo: TestCircle.java

```
public class TestCircle
{
    public static void main(String[] args)
    {
        Circle c = new Circle(20.0);
        System.out.println("The area is: "+c.getArea());
    }
}
```



3.1 面向对象程序设计方法概述



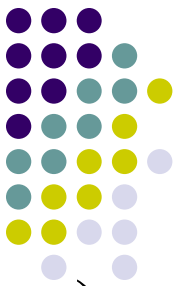
■ 结构化程序设计

- 对应的典型计算机语言, 例如: **C**
- 面向操作(**action**)的
- **函数**(方法)是程序的基本单位

■ 面向对象程序设计

- 对应的典型计算机语言, 例如: **Java**
- 面向**对象**(**object**)的
- **类**(**class**)是程序的基本单位
 - **方法**(函数)被封装在类中
 - **数据**也常常被封装在类中

3.1 面向对象程序设计方法概述



- **面向对象程序设计 (Object-Oriented Programming)**
 - 与结构化程序设计方法相比，更符合人类认识现实世界的思维方式
 - 已成为程序设计/软件分析与设计的主流方向
 - 涉及的主要概念
 - 抽象——abstract
 - 封装——encapsulation
 - 继承——inheritance
 - 多态 —— polymorphism

3.1.1 抽象



■ 抽象 (abstract)

- 忽略问题中与当前目标无关的方面，以便更充分地注意与当前目标有关的方面
- 计算机软件开发中所使用的抽象有
 - 过程抽象
 - 数据抽象

3.1.1 抽象



■ 过程抽象

- 将整个系统的功能划分为若干部分，强调功能完成的过程和步骤，而隐藏其具体的实现
- 基于过程抽象的两个标准程序设计技术
 - 过程分解
 - 递归技术

3.1.1 抽象



■ 数据抽象

- 将需要处理的数据和这些数据上的操作结合在一起，抽象成不同的抽象数据类型
- 每个抽象数据类型既包含了数据，也包含了针对这些数据的操作
- 相对于过程抽象，数据抽象是更为合理的抽象方法

3.1.1 抽象



■ 举例1:

● 钟表- Clock

➤ 数据(属性)

- `int Hour; int Minute; int Second;`

➤ 方法(行为)

- `SetTime(); ShowTime();`

3.1.1 抽象



■ 举例2:

● 人 - person

➤ 数据(属性)

- `char *name; char *gender; int age; int id;`

➤ 方法(行为)

- 生物行为
`Eat(), Step(), ...`
- 社会行为
`Work(), Study(), ...`

3.1.1 抽象



■ Java

```
import java.math.*;
class Circle
{
    private double radius;

    Circle(double rads)
    {
        radius = rads;
    }
    public double getArea()
    {
        return radius * radius * Math.PI;
    }
}
```

3.1.2 封装



■ 封装 (encapsulation)

- 是一种信息隐蔽技术
- 将数据和基于数据的操作封装在一起
- 用户只能看到对象的封装接口信息，对象的内部细节对用户是隐蔽的
- 封装的目的在于将对象的使用者和设计者分开，使用者不必知道行为实现的细节，只需使用设计者提供的消息来访问对象

3.1.2 封装



■ Java

```
import java.math.*;
class Circle
{
    private double radius;

    Circle(double rads)
    {
        radius = rads;
    }
    public double getArea()
    {
        return radius * radius * Math.PI;
    }
}
```

3.1.2 封装



■ 封装的定义

- 清楚的边界

- 所有对象的内部信息被限定在这个边界内

- 接口

- 对象向外界提供的方法，外界可以通过这些方法与对象进行交互

- 受保护的内部实现

- 功能的实现细节，不能从类外访问。

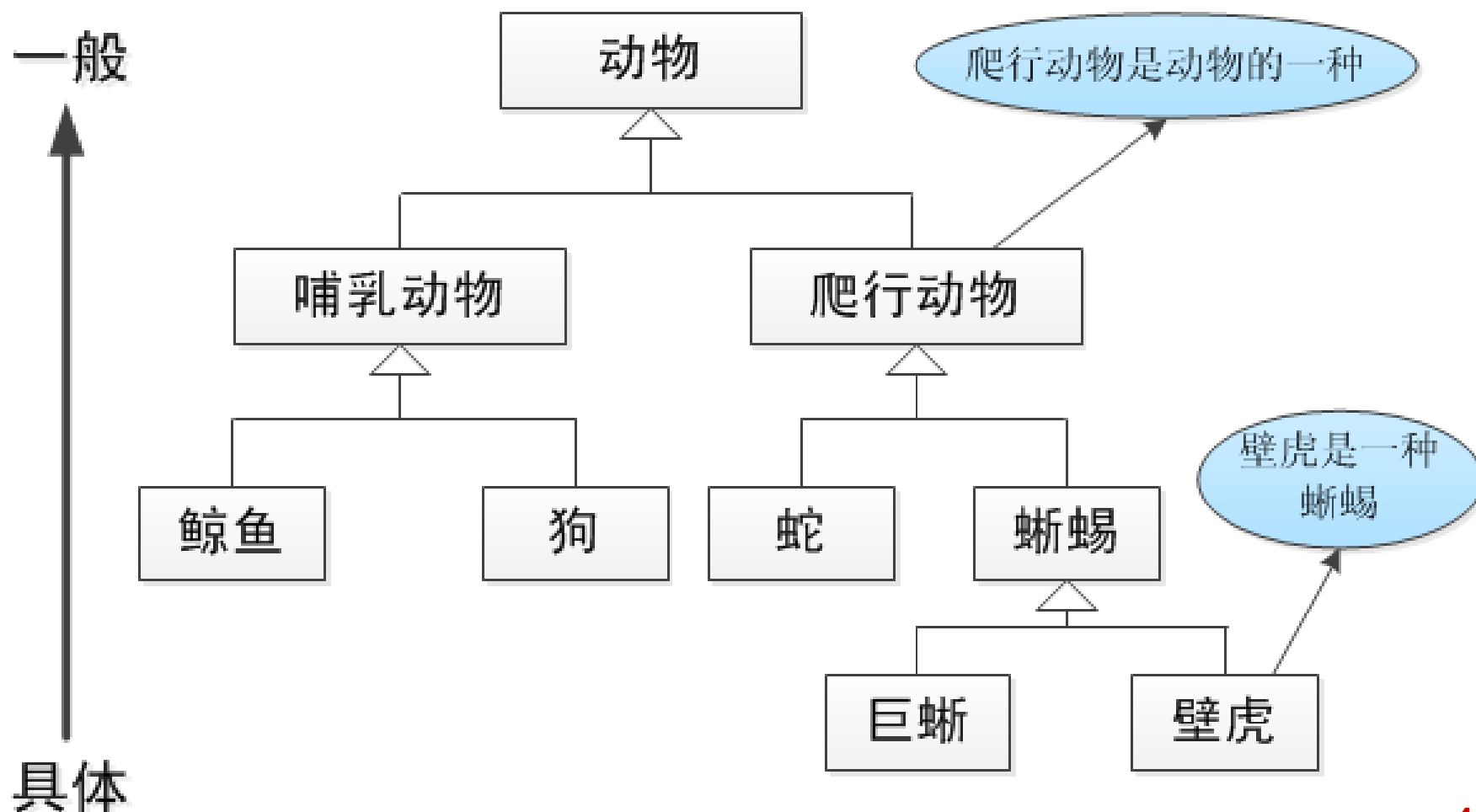
■ 封装的意义

- 在面向对象程序设计中，类封装了数据及对数据的操作，是程序中的最小模块

3.1.3 继承



■ 动物类层次举例



3.1.3 继承



■ 继承 (inheritance)

- 是指新的类可以获得已有类（称为超类/基类/父类）的属性和行为，称新类为已有类的派生类（也称为子类）
- 继承过程中派生类继承了基类的特性，包括属性和方法
- 派生类也可修改继承的方法或增加新的方法，使之更适合特殊的需要
- 有助于解决软件的可重用性问题，使程序结构清晰，降低了编码和维护的工作量

3.1.3 继承



■ Employee类继承Person类

```
public class Person {  
    public String name;  
    public String getName() {  
        return name;  
    }  
}
```

```
public class Employee extends Person {  
    public int employeeNumber;  
    public int getEmployeeNumber() {  
        return employeeNumber;  
    }  
}
```

3.1.3 继承



■ 单继承

- 任何一个派生类都只有单一的直接父类
- 类层次结构为树状结构

■ 多继承

- 一个类可以有一个以上的直接父类
- 类层次结构为网状结构，设计及实现比较复杂

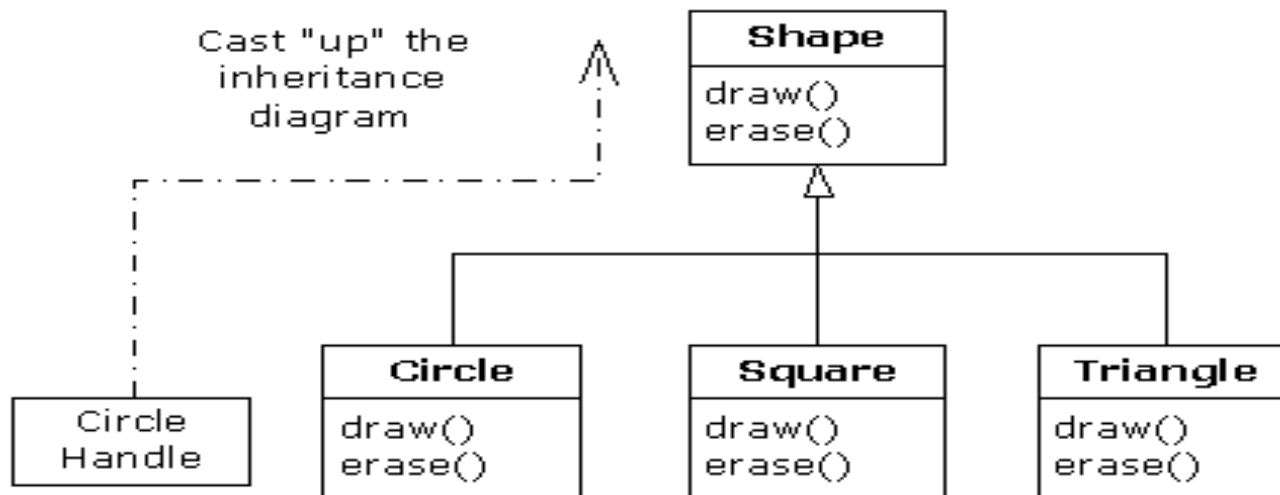
■ Java语言仅支持单继承

3.1.4 多态



■ 多态 (polymorphism)

- 一个程序中同名的不同方法共存
- 不同类的对象可以响应同名的消息(方法)，具体的实现方法却不同
- **Circle.draw(); Triangle.draw(); Square.draw();**
- 使语言具有灵活、抽象、行为共享、代码共享的优势，很好地解决了应用程序方法同名问题



第三章 Java面向对象程序设计-1



3.1 面向对象程序设计方法概述

3.2 类与对象

3.3 对象初始化和回收

3.4 应用举例

3.2 类与对象



■ 类与对象

- 在程序中，对象是通过一种抽象数据类型来描述的，这种抽象数据类型称为类(class)
- 一个类是对一组相同/相似对象的描述。
- 类是构造对象的模板
- 对象是类的具体实例 (instance)

3.2 类与对象



3.2.1 类的声明

3.2.2 对象的声明与引用

3.2.3 数据成员

3.2.4 方法成员

3.2.5 方法成员重载

3.2.6 成员方法参数

3.2.1 类的声明



- 编写类的“模板”
- 声明形式

[**public**] [**abstract** | **final**] **class** 类名称

[**extends** 父类名称]

[**implements** 接口名称列表]

{

public/private 数据类型 变量名; //变量成员声明初始化

public/private 数据类型 方法名（参数列表）; //方法声明及方法体

}

3.2.1 类的声明



class关键字用来定义一个类

Circle是类名称

```
import java.math.*;
class Circle
{
    double radius;
    Circle()
    {
    }
    Circle(double rads)
    {
        radius = rads;
    }
    double getArea()
    {
        return radius * radius * Math.PI;
    }
}
```

数据域

构造方法

方法

3.2.1 类的声明



■ 关键字

- **class**

- 表明其后声明的是一个类

- **extends**

- 如果所声明的类是从某一父类派生而来，那么，父类的名字应写在extends之后

- **implements**

- 如果所声明的类要实现某些接口，那么，接口的名字应写在implements之后

3.2.1 类的声明



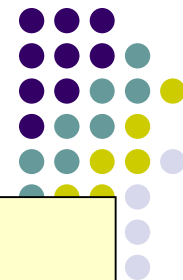
■ 修饰符

- 可以有多个，用来限定类的使用方式
- **public**
 - 表明此类为公有类
- **abstract**
 - 指明此类为抽象类
- **final**
 - 指明此类为终结类

■ 类的声明体

- 变量成员声明及初始化
 - 可以有多个
- 方法声明及方法体
 - 可以有多个

3.2.1 类的声明



```
public class Clock
{ // 成员变量
    int hour ;
    int minute ;
    int second ;

    // 成员方法
    public void setTime(int newH, int newM, int newS)
    {
        hour=newH ;
        minute=newM ;
        second=newS ;
    }
    public void showTime ()
    {
        System.out.println(hour+": "+minute+": "+second) ;
    }
}
```

3.2 类与对象



3.2.1 类的声明

3.2.2 对象的声明与引用

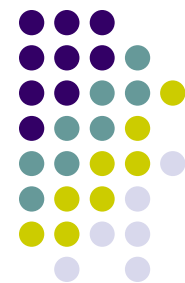
3.2.3 数据成员

3.2.4 方法成员

3.2.5 方法成员重载

3.2.6 成员方法参数

3.2.2 对象的声明与引用



■ 变量和对象

- 变量除了存储基本数据类型的数据，还能存储**对象**的引用，
- “引用”一个对象的变量称为“**引用类型**”的变量，简称“**对象变量**”或“引用变量”。
- 类的对象也称为**类的实例(instance)**

3.2.2 对象的声明与引用



■ 对象的声明 (declaration)

- 格式

类名 变量名;

- 例如Clock是已经声明的类名，则下面语句声明的变量aclock将用于存储该类对象的引用：

Clock aclock;

- 声明一个对象变量时，实际上并没有创建一个对象，此变量=null。

3.2.2 对象的声明与引用



■ 对象的创建 (creation)

- 生成对象的格式:

new <类名> (...);

➤ 例如: **aclock = new Clock();**

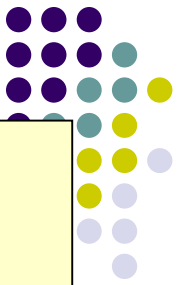
- 其作用是:

➤ 在内存中为此对象分配内存空间

➤ 返回对象的引用(reference, 相当于对象的存储地址)

- 引用变量可以被赋以空值

➤ 例如: **aclock = null;**



```
public class TestClock {  
    public static void main(String[] args){  
        Scanner in = new Scanner(System.in);  
        // get input  
        System.out.print("What is the hour? ");  
        int in_hour = in.nextInt();  
  
        System.out.print("What is the minute? ");  
        int in_minute = in.nextInt();  
  
        System.out.print("What is the second? ");  
        int in_second = in.nextInt();  
  
        Clock c1 = new Clock();  
        c1.setTime(in_hour, in_minute, in_second);  
        System.out.print("The current time is: ");  
        c1.showTime();  
    }  
}
```



3.2.2 对象的声明与引用

■ 小结

1、创建对象(生成对象)

Clock c1 = new Clock();

a) 定义对象变量: **类名** **对象名**;

Clock c1; 声明一个引用变量时并没有对象生成

b) 分配内存: **new** 运算符

返回对象的引用(reference, 相当于对象的存储地址)

c) 初始化对象: 即初始化对象中的实例变量。

对象可以被赋以空值: 例如: **c1=null;**

Clock c1 = new Clock();

2、指派对象引用

1、创建对象

3.2.2 对象的声明与引用



2、使用对象

例：获取Clock对象的时间：`c1.showTime()`。

a) 访问对象的成员

调用对象的方法：`c1.showTime()`;

引用对象的数据：`c1.hour`;

b) 对象做类的成员

c) 方法中使用对象：方法参数和返回值

3、释放对象

- 自动垃圾回收（标记）；`finalize()`

3.2 类与对象



3.2.1 类的声明

3.2.2 对象的声明与引用

3.2.3 数据成员

3.2.4 方法成员

3.2.5 方法成员重载

3.2.6 成员方法参数

3.2.3 数据成员



■ 数据成员

- 表示Java类的状态
- 声明数据成员必须给出变量名及其所属的类型，
同时还可以指定其他特性
- 在一个类中成员变量名是唯一的
- 数据成员的类型可以是Java中任意的数据类型(简单类型，类，接口，数组)
- 分为实例变量和类变量

3.2.3 数据成员



■ 声明格式

[**public** | **protected** | **private**]

[**static**][**final**][**transient**][**volatile**]

变量数据类型 变量名1[=变量初值],
变量名2[=变量初值], ... ;

● 格式说明

- **public**, **protected**, **private**为访问控制符
- **static**指明这是一个静态成员变量
- **final**指明变量的值不能被修改
- **transient**指明变量是临时状态
- **volatile**指明变量是一个共享变量

3.2.3 数据成员



■ 实例变量(Instance Variables)

- 没有static修饰的变量称为实例（对象）变量
- 用来存储所有实例都需要的属性信息，不同实例的属性值可能会不同
- 可通过下面的表达式访问对象属性的值

<对象名>.<对象变量名>

例如： `c1.hour;`

3.2.3 数据成员



■ 例子

- 定义一个表示圆的类，保存在文件**Circle.java** 中。
- 然后编写测试类，保存在文件**ShapeTester.java**中，并与**Circle.java**放在相同的目录下



```
import java.math.*;
public class Circle
{
    public double radius;
    public double getArea()
    {
        return radius * radius * Math.PI;
    }
}
```

```
class ShapeTester
{
    public static void main(String args[])
    {
        Circle aCircle = new Circle();
        System.out.println(aCircle);
        System.out.println("radius = " + aCircle.radius);
    }
}
```

3.2.3 数据成员



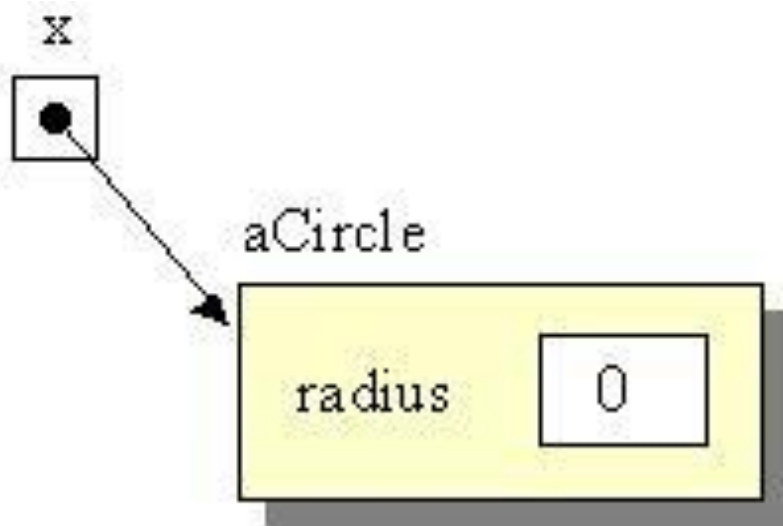
- 编译后运行结果如下：

Circle@592fa617

radius = 0

- 解释

- @之后的数值为x所指的对象的存储地址
- x的值及对象的状态如图



3.2.3 数据成员



- 声明一个表示矩形的类，保存在`Rectangle.java`中；编写测试类，保存在`ShapeTester.java`中，二文件保存在相同的目录下

```
public class Rectangle {  
    double width = 10.128;  
    double height = 5.734;  
}
```

```
public class ShapeTester {  
    public static void main(String args[]) {  
        Circle    x;  
        Rectangle y;  
        x = new Circle();  
        y = new Rectangle();  
        System.out.println(x + "    " + y);  
    }  
}
```

3.2.3 数据成员

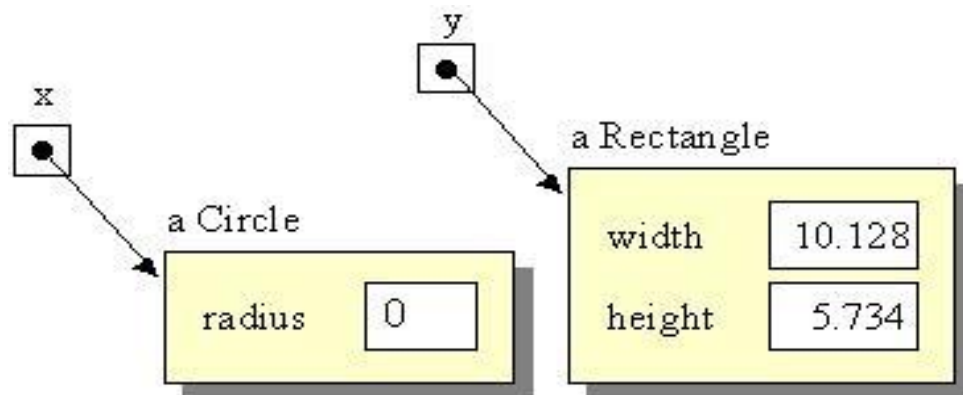


■ 编译后运行结果如下：

Circle@82f0db45 Rectangle@92d342df

■ 解释

- Circle及Rectangle类对象的状态如图
- @之后的数值为x所指的对象的存储地址



3.2.3 数据成员



- 对ShapeTester类进行修改，使两个实例具有不同的实例变量值

```
public class ShapeTester {  
    public static void main(String args[]) {  
        Circle x;  
        Rectangle y, z;  
        x = new Circle();  
        y = new Rectangle();  
        z = new Rectangle();  
        x.radius = 50;  
        z.width = 68.94;  
        z.height = 47.54;  
        System.out.println(x.radius + ";" + y.width + ";" + z.width);  
    }  
}
```

3.2.3 数据成员

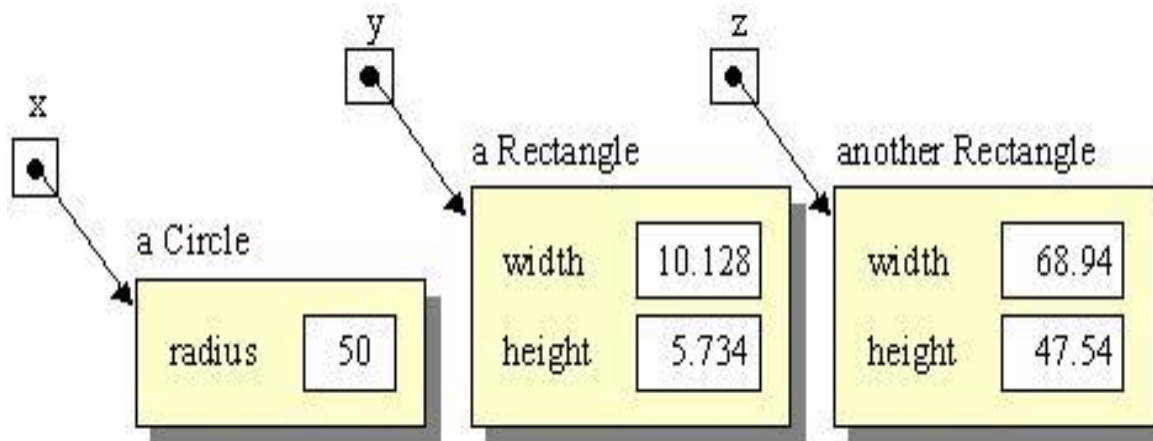


■ 编译后运行结果如下：

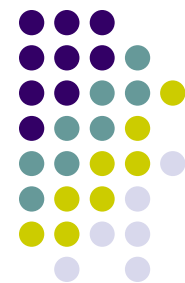
50; 10.128; 68.94

■ 解释

- Circle及Rectangle类对象的状态如图




3.2.3 数据成员



■ 例子：地址簿程序

- 一个人的地址通常包括以下信息：
姓名，省份，城市，街道，门牌号，邮政编码
- 采用过程化的程序设计方法，使用简单变量存储，则存储两个人地址的代码如下



```
public static void main(String args[]) {  
    String name1, name2;  
    int  roomNumber1, roomNumber2;  
    String streetName1,streetName2;  
    String city1, city2;  
    String province1, province2;  
    String postalCode1, postalCode2;  
    name1 = " Zhang Li";  
    roomNumber1 = 15;  
    streetName1 = " Tsinghua East  Road";  
    city1 = " Beijing";  
    province1 = "Beijing";  
    postalCode1 = " 100084";  
    name2 = " Li Hong";  
    roomNumber2 = 2;  
    streetName2 = " BeiNong";  
    city2 = " Beijing";  
    province2 = " Beijing";  
    postalCode2 = " 102206";    //...do something interesting  
}
```


3.2.3 数据成员



- 采用面向对象的程序设计方法，则需要首先声明 **Address** 类如下

```
public class Address {  
    String name;  
    int roomNumber;  
    String streetName;  
    String city;  
    String province;  
    String postalCode;  
    //方法成员略  
}
```

■ 主方法改写如下

```
public static void main(String args[]) {  
    Address address1 = new Address(), address2 = new Address();  
    address1.name = "Zhang Li";  
    address1.streetNumber = 15;  
    address1.streetName = "Tsinghua East Road";  
    address1.city = "Beijing";  
    address1.province = "Beijing";  
    address1.postalCode = "100084";  
    address2.name = "Li Hong";  
    address2.streetNumber = 2;  
    address2.streetName = "BeiNong";  
    address2.city = "Beijing";  
    address2.province = "Beijing";  
    address2.postalCode = "102206";  
    //...do something interesting  
}
```

3.2.3 数据成员



■ 类变量——Static变量

- 也称为**静态变量**，声明时需加**static**修饰符
- 不管类的对象有多少，类变量只存在一份，在整个类中只有一个值
- **类初始化的同时就被赋值**
- 适用情况
 - 类中所有对象都相同的属性
 - 经常需要共享的数据
 - 系统中用到的一些常量值
- 引用格式
<类名 | 对象名>.<类变量名>

3.2.3 数据成员



- 对于一个圆类的所有对象，计算圆的面积时，都需用到 π 的值，可在Circle类的声明中增加一个类属性PI

```
public class Circle {  
    static double PI = 3.14159265;  
    int radius;  
}
```

- 当生成Circle类的实例时，在每一个实例中并没有存储PI的值，PI的值存储在类中

3.2.3 数据成员



■ 对类变量进行测试

```
public class ClassVariableTester {  
    public static void main(String[] args) {  
        Circle x = new Circle();  
        System.out.println(x.PI);  
        System.out.println(Circle.PI);  
        Circle.PI = 3.14;  
        System.out.println(x.PI);  
        System.out.println(Circle.PI);  
    }  
}
```

• 测试结果

3.14159265
3.14159265
3.14
3.14

■ 推荐使用类名来直接访问静态数据。

- 声明一个Point类，有两个私有变量保存点坐标，一个类变量保存已有点的个数（Demo: Point.java）



```
public class Point {  
    private int x;  
    private int y;  
    public static int pointCount=0;  
    public Point(int x, int y) {  
        this.x = x; this.y = y; pointCount++;  
    }  
}
```

- 测试类
(Demo:PointTester.java)

```
class PointTester {  
    public static void main(String[] args) {  
        Point p = new Point(1, 1);  
        System.out.println(p.pointCount);  
        Point q = new Point(2, 2);  
        System.out.println(q.pointCount);  
        System.out.println(q.pointCount == Point.pointCount);  
        System.out.println(Point.pointCount);  
    }  
}
```

测试结果

1

2

true

2

3.2.3 数据成员



- 数据成员——final修饰符
 - 用途：定义常量
 - 实例变量和类变量都可被声明为final
 - final实例变量必须在每个构造方法结束之前赋初值，以保证使用之前会被初始化
 - final类变量必须在声明的同时初始化
 - final变量一旦被初始化便不可改变

3.2 类与对象



3.2.1 类的声明

3.2.2 对象的声明与引用

3.2.3 数据成员

3.2.4 方法成员

3.2.5 方法成员重载

3.2.6 成员方法参数

3.2.4 方法成员



■ 方法成员

- 定义类的行为

- 一个对象能够做的事情
- 能够从一个对象取得的信息

- 可以没有，也可以有多个；

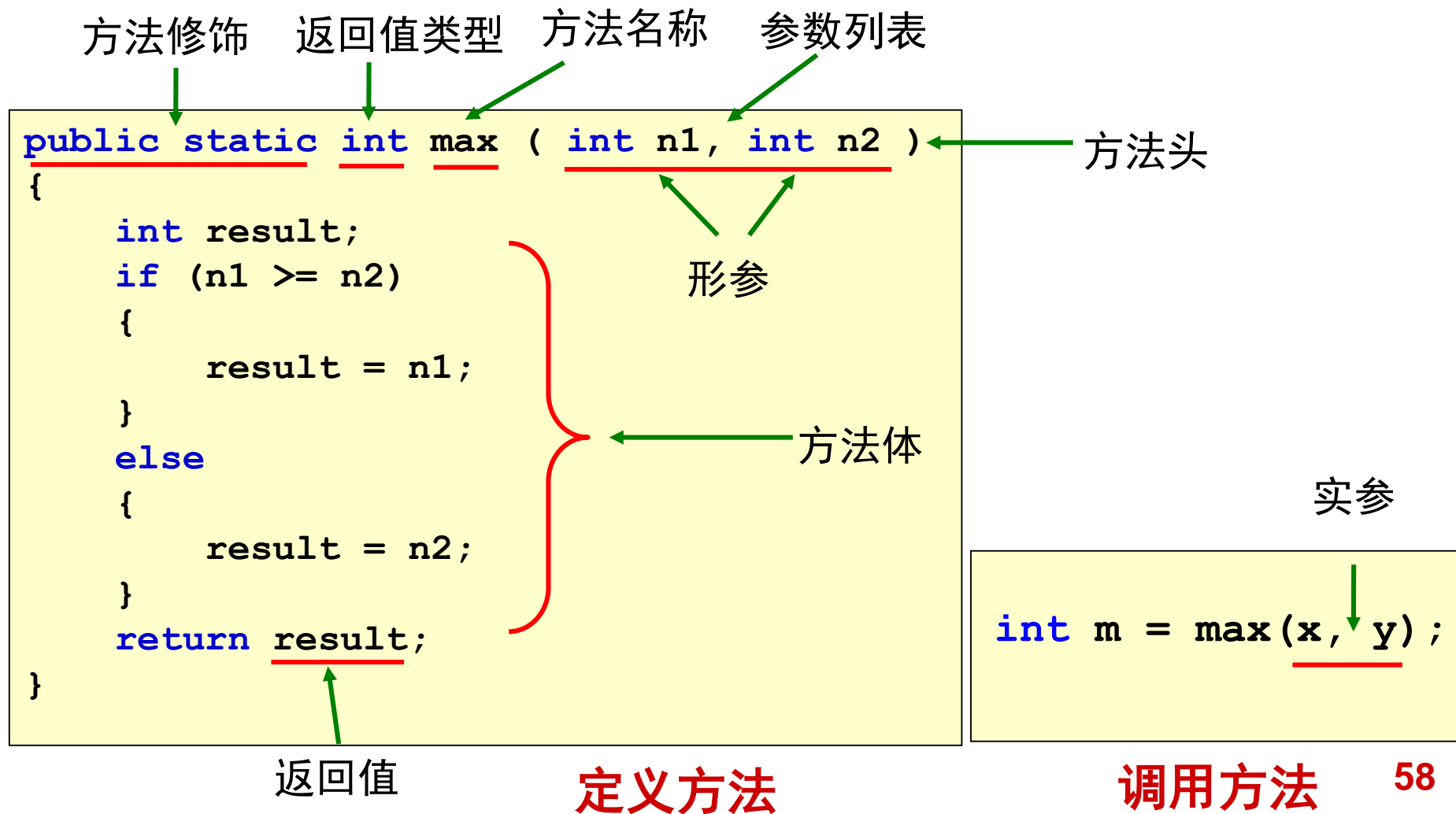
- 一旦在类中声明了方法，它就成为了类声明的一部分

- 分为实例方法和类方法

3.2.4 方法成员



- 方法（Method）是为执行一个操作而组合在一起的语句组





3.2.4 方法成员

- 与C++程序不同的是：每个方法都必须隶属于某个类
- 声明格式

```
[public | protected | private]
[static][final][abstract][native][synchronized]
返回类型 方法名 ([参数列表]) [throws exceptionList] {
    方法体
}
```

- 返回类型: 返回数据的数据类型
 - 除了构造方法，一般要求返回类型: 要么为void，要么为任意Java数据类型
 - 最多只能返回一个值
- 方法名: 任何一个合法的标识符
- 参数列表:
 - 用逗号(,) 分隔开；参数类型与返回值类型同，可以为空

3.2.4 方法成员



- 格式说明

- 方法修饰

- `public`、`protected`、`private` 为存取控制符
- `static`: 指明方法是一个类方法
- `final`: 指明方法是一个终结方法
- `abstract`: 指明方法是一个抽象方法
- `native`: 用来集成java代码和其它语言的代码
- `synchronized`: 用来控制多个并发线程对共享数据的访问

3.2.4 方法成员



- 格式说明(续)

- 返回类型

- 方法返回值的类型，可以是任意的Java数据类型
- 当不需要返回值时，返回类型为void
- 除了void外，方法中必须包含return语句

- 参数类型

- 简单数据类型，
- 引用类型(数组、类或接口)
- 可以有多个参数，也可以没有参数，方法声明时的参数称为形式参数

- 方法体

- 方法的实现
- 包括局部变量的声明以及所有合法的Java指令
- 局部变量的作用域只在该方法内部

- throws exceptionList

- 用来处理异常

3.2.4 方法成员



■ 方法调用

- 给对象发消息意味着调用对象的某个方法
 - 从对象中取得信息
 - 修改对象的状态或进行某种操作
 - 进行计算及取得结果等

- 调用格式

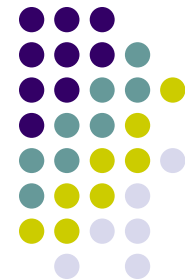
<对象名>.<方法名>（〔参数列表〕）

称点操作符 “.” 前面的<对象名>为消息的接收者
(receiver)

- 参数传递

- 值传递：参数类型为基本数据类型时
- 引用传递：参数类型为对象类型或数组时

3.2.4 方法成员



■ 实例方法

- 表示特定对象的行为
- 声明时前面不加static修饰符
- 使用时需要发送给一个类实例

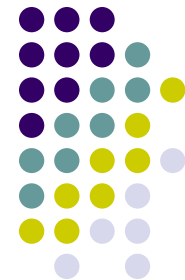


■ 举例：在Circle类中声明计算周长的方法

```
public class Circle {  
    static double PI = 3.14159265;  
    int radius;  
    public double circumference() {  
        return 2 * PI * radius;  
    }  
}
```

- 由于radius是实例变量，在程序运行时，Java会自动取其接收者对象的属性值
- 也可将**circumference**方法体改为：
`return 2 * PI * this.radius;`
关键字**this**代表此方法的接收者对象

3.2.4 方法成员



■ 独特的对象引用：**this**

- 对象中的所有数据字段都是通过this指针间接引用的。
- 同一类中的方法可以相互调用，其中隐含了一个this调用。

3.2.4 方法成员



- 方法调用测试

```
public class CircumferenceTester {  
    public static void main(String args[]) {  
        Circle c1 = new Circle();  
        c1.radius = 50;  
        Circle c2 = new Circle();  
        c2.radius = 10;  
        double circum1 = c1.circumference();  
        double circum2 = c2.circumference();  
        System.out.println("Circle 1 has circumference " + circum1);  
        System.out.println("Circle 2 has circumference " + circum2);  
    }  
}
```

3.2.4 方法成员



■ 运行结果

Circle 1 has circumference 314.159265

Circle 2 has circumference 62.831853

■ 说明

- 在使用实例方法时，需要将其发送给一个实例对象（也称给对象发送一条消息），radius的值即是接收者对象的值
- 在执行c1.circumference()时，radius的值为c1的radius属性值；
- 在执行c2.circumference()时，radius的值为c2的radius属性值

3.2.4 方法成员



- 在Circle类及Rectangle类中声明计算面积的方法area()

```
public class Circle {  
    static double PI = 3.14159265;  
    int radius;  
    public double circumference() {  
        return 2 * PI * radius;  
    }  
    public double area() {  
        return PI * radius * radius;  
    }  
}
```

```
public class Rectangle {  
    double width;  
    double height;  
    public double area() {  
        return width * height;  
    }  
}
```



- 声明测试类AreaTester，对Circle类及Rectangle类的area()方法进行测试

```
public class AreaTester {  
    public static void main(String args[]) {  
        Circle c = new Circle();  
        c.radius = 50;  
        Rectangle r = new Rectangle();  
        r.width = 20;  
        r.height = 30;  
        System.out.println("Circle has area " + c.area());  
        System.out.println("Rectangle has area " + r.area());  
    }  
}
```

■ 运行结果

Circle has area 7853.981625

Rectangle has area 600.0

■ 说明

- 不同的类中可以声明相同方法名的方法
- 使用时，系统会根据接收者对象的类型找到相应类的方法

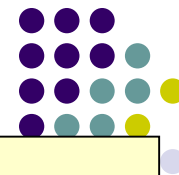
3.2.4 方法成员



- 举例：在Circle类中增加方法改变圆的半径。
(带参数的方法)

```
public class Circle {  
    static double PI = 3.14159265;  
    int radius;  
    public double circumference() {  
        return 2 * PI * radius;  
    }  
    public double area() {  
        return PI * radius * radius;  
    }  
    public void enlarge (int factor) {  
        radius = radius * factor;  
    }  
}
```

■ 测试类EnlargeTester



```
public class EnlargeTester {  
    public static void main(String args[]) {  
        Circle c1 = new Circle();  
        c1.radius = 50;  
        Circle c2 = new Circle();  
        c2.radius = 10;  
        System.out.println("Circle 1 的周长: " + c1.circumference());  
        System.out.println("Circle 2 的周长: " + c2.circumference());  
        c2.enlarge(4);  
        System.out.println("Circle 2 扩大后的周长: " + c2.circumference());  
    }  
}
```

■ 运行结果

Circle 1 的周长: 314.159265

Circle 2 的周长: 62.831853

Circle 2 扩大后的周长: 251.327412

- 举例：在Circle类中增加fitsInside方法判断一个圆是否在一个长方形内，需要以Rectangle类的对象作为参数。（以对象作为参数的方法）



```
public class Circle {  
    static double PI = 3.14159265;  
    int radius;  
    public double circumference() {  
        return 2 * PI * radius;  
    }  
    public void enlarge(int factor) {  
        radius = radius * factor;  
    }  
    public boolean fitsInside (Rectangle r) {  
        return (2 * radius <= r.width) && (2 * radius <= r.height);  
    }  
}
```


■ 测试类InsideTester



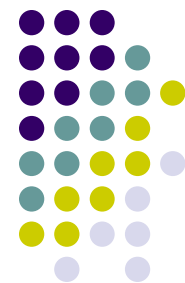
```
public class InsideTester {  
    public static void main(String args[]) {  
        Circle c1 = new Circle();  
        c1.radius = 8;  
        Circle c2 = new Circle();  
        c2.radius = 15;  
        Rectangle r = new Rectangle();  
        r.width = 20;  
        r.height = 30;  
        System.out.println("Circle1 fits inside Rect:" + c1.fitsInside(r));  
        System.out.println("Circle2 fits inside Rect:" + c2.fitsInside(r));  
    }  
}
```

■ 运行结果

Circle 1 fits inside Rectangle: true

Circle 2 fits inside Rectangle: false

3.2.4 方法成员



■ 类方法——static方法

- 也称为**静态方法**，表示类中对象的共有行为
- 声明时，前面**需加static修饰符**
- 不能被声明为抽象方法
- 类方法可以在不建立对象的情况下用类名直接调用，也可用类实例调用

3.2.4 方法成员



- 举例：将摄氏温度(centigrade)转换成华氏温度(fahrenheit)
 - 转换公式为 $\text{fahrenheit} = \text{centigrade} * 9 / 5 + 32$
 - 除了摄氏温度值及公式中需要的常量值，此功能不依赖于具体的类实例的属性值，因此可声明为类方法。
 - 转换方法centigradeToFahrenheit放在类Converter中

```
public class Converter {  
    public static int centigradeToFahrenheit(int cent) {  
        return (cent * 9 / 5 + 32);  
    }  
}
```

- 方法调用

```
Converter.centigradeToFahrenheit(40);
```

3.2.4 方法成员



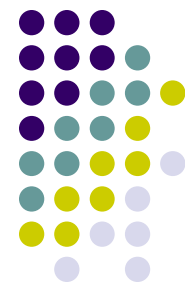
■ 类的静态方法

```
class Employee
{
    String name;
    long salary;
    short employee_id;
    static int total_employees;

    static void clear(){
        total_employees=0;
    }
}
```

- 在访问本类的成员时，静态方法只能访问静态成员！

3.2 类与对象



3.2.1 类的声明

3.2.2 对象的声明与引用

3.2.3 数据成员

3.2.4 方法成员

3.2.5 方法成员重载

3.2.6 成员方法参数

3.2.5 方法成员重载



```
import java.math.*;
class Circle
{
    double radius = 1.0;
    Circle()
    {
    }
    Circle(double rads)
    {
        radius = rads;
    }
    double getArea()
    {
        return radius * radius * Math.PI;
    }
}
```

3.2.5 方法成员重载



■ 方法重载（overloading）

- 一个类中**名字相同**的多个方法
- 这些方法的**参数必须不同**，Java可通过参数列表的不同来辨别重载的方法
 - 或者参数个数不同
 - 或者参数类型不同
- 返回值可以相同，也可以不同
- 重载的价值在于它允许通过使用一个方法名来访问多个方法

3.2.5 方法成员重载



- 在编译时就能够被识别
- 针对： **同一个类内的同名方法**
- 例如：

```
public int square(int x)
{
    return x * x;
}
public double square(double x)
{
    return x * x;
}
```

- 识别标志：
 - 参数的个数、类型、数据类型的排列顺序
 - 返回值不能做为识别的标志？


```
public class OverloadTest
```

Demo: OverloadTest.java

```
{
```

```
    public static void main(String[] args) {
```

```
        System.out.printf("The maximum between %d and %d is %d\n",  
                           1, 2, max(1, 2));
```

```
        System.out.printf("The maximum between %f and %f is %f\n",  
                           1.0, 2.0, max(1.0, 2.0));
```

```
        System.out.printf("The maximum between %f, %f, and %f is %f\n",  
                           1.0, 2.0, 3.0, max(1.0, 2.0, 3.0));
```

```
    }
```

```
    public static int max ( int n1, int n2 ) {
```

```
        if (n1 >= n2) return n1;
```

```
        else return n2;
```

```
    }
```

```
    public static double max ( double n1, double n2 ) {
```

```
        return (n1 >= n2) ? n1 : n2;
```

```
    }
```

```
    public static double max( double n1, double n2, double n3 ) {
```

```
        return max(max(n1, n2), n3);
```

```
    }
```

```
}
```

3.2.5 方法成员重载



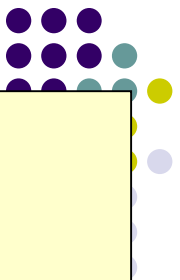
■ 歧义调用 (ambiguous invocation)

- 有时一个方法调用会有两个或更多可能的匹配，编译器无法判断哪个更为合适。这称为歧义调用。
- 歧义调用会产生编译错误，考虑如下代码：

```
max(int, double) { ..... }
```

```
max(double, int) { ..... }
```

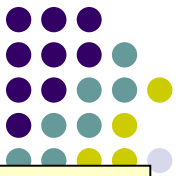
都可能与`max(1, 2)`匹配。由于两个方法谁也不比谁更合适，所以这个调用是有歧义的，导致编译错误。



- 举例：歧义调用

```
public class AmbiguousOverloadTest
{
    public static void main(String[] args)
    {
        System.out.println(max(1, 2));
    }
    public static double max ( double n1, int n2 )
    {
        if (n1 >= n2) return n1;
        else return n2;
    }
    public static double max ( int n1, double n2 )
    {
        if (n1 >= n2) return n1;
        else return n2;
    }
}
```

- System.out.println(max(1, (double)2)); 或
System.out.println(max((double)1, 2));



■ 判断：哪些是重载方法？

```
public class OverloadExample
```

```
{
```

```
    public static int max ( int n1, int n2 )
```

```
    {
```

```
        // ... ..
```



参数表不变，只改变返回值，不是重载

```
    }
```

```
    public static double max ( int n1, int n2 )
```

```
    {
```

```
        // ... ..
```



只改变了参数表中的形参的命名，类型未变化，不是重载

```
    }
```

```
    public static int max ( int a1, int a2 )
```

```
    {
```

```
        // ... ..
```



```
    }
```

```
    public static int max( String s1, short s2)
```

```
    {
```

```
        // ... ..
```

```
    }
```

```
}
```

3.2 类与对象



3.2.1 类的声明

3.2.2 对象的声明与引用

3.2.3 数据成员

3.2.4 方法成员

3.2.5 方法成员重载

3.2.6 成员方法参数

3.2.6 成员方法参数



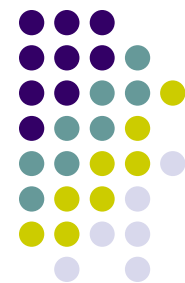
■ 参数名

- 方法的参数名可以和类的成员变量同名
- ```
class Circle
{
 int x, y, radius;
 public Circle (int x, int y, int radius)
 {
 this.x = x;
 this.y = y;
 this.radius = radius;
 }
}
```

### ■ 参数类型

- 任何合法的数据类型传递到方法中
- 不能将一个方法作为参数传给另一个方法

## 3.2.6 成员方法参数



### ■ 传值与传地址(引用)

- 在方法中，简单数据类型的参数是传值的，而不是引用变量本身。
  - 好处：带来一定的安全性，使方法不能随意改变作用域外的变量。
  - 缺点：有时需要改变一个或多个参数的值，用return也只能返回一个值。
- 对象的引用是对象在内存中的地址，从而使方法中的局部变量与调用者中的变量指向了同一个内存位置。

## ■ 举例：通过值传递



```
public class PassByValueExample {
 public static void main(String[] args) {
 int x = 1;
 int y = 2;
 System.out.printf("Before invoke the swap method: x = %d, y = %d\n", x, y);
 swap(x, y);
 System.out.printf("After invoke the swap method: x= %d, y = %d\n", x, y);
 }

 public static void swap (int n1, int n2) {
 System.out.printf("\t Inside the swap method \n");
 System.out.printf("\t \t Before swap: n1 = %d, n2 = %d\n", n1, n2);
 int temp = n1;
 n1 = n2;
 n2 = temp;
 System.out.printf("\t \t After swap: n1 = %d, n2 = %d\n", n1, n2);
 }
}
```



## 3.2.6 成员方法参数



### ■ 值传递

输出结果:

Before invoke the swap method:  $x = 1, y = 2$

Inside the swap method

Before swap:  $n1 = 1, n2 = 2$

After swap:  $n1 = 2, n2 = 1$

After invoke the swap method:  $x = 1, y = 2$

## 3.2.6 成员方法参数



- 可以看到，这次 swap 失败了... x、y 的值并没有被交换
- 这是因为 Java 的方法调用是“通过值传递参数的”！
- 在方法调用时，实参的值，被copy给形参。在方法内部改变形参的值，并不会影响到实参的值。

```
public static void main (String[] args)
{
 int x = 1;
 int y = 2;
 swap(x, y);
}
```

实参

```
public static void swap(int n1, int n2)
{
 int temp = n1;
 n1 = n2;
 n2 = temp;
}
```

形参


copy

## 3.2.6 成员方法参数



### ■ 给方法传递对象参数

- 对象可以作为方法的参数，进行传递
- 使用“按值传递”的规则，进行传递
  - 给基本类型值传值：（传值给参数）
  - 给引用类型（对象）值传值：（值是对象的引用）



```
public class Circle3
{
 private double radius;
 public Circle3(double newRadius)
 {
 radius = newRadius;
 }
 public double getRadius()
 {
 return radius;
 }
 public void setRadius(double newRadius)
 {
 radius = newRadius;
 }
 public double getArea()
 {
 return radius * radius * Math.PI;
 }
}
```

```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is " + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c, int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius() + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

# 跟踪执行过程 - 1



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*myCircle*

对象引用

Circle3对象  
radius=1

# 跟踪执行过程 - 2



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*myCircle* →

对象引用

Circle3对象  
radius=1

*n* →

5

# 跟踪执行过程 - 3



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*myCircle* →

对象引用

Circle3对象  
radius=1

*n* →

5



# 跟踪执行过程 - 4



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*myCircle*



对象引用



*c*



对象引用



*n*



5

*times*



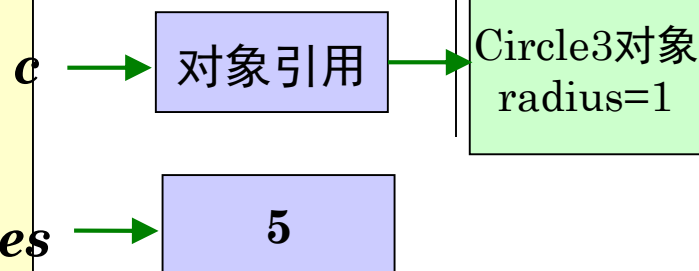
5

Circle3对象  
radius=1

# 跟踪执行过程 - 5



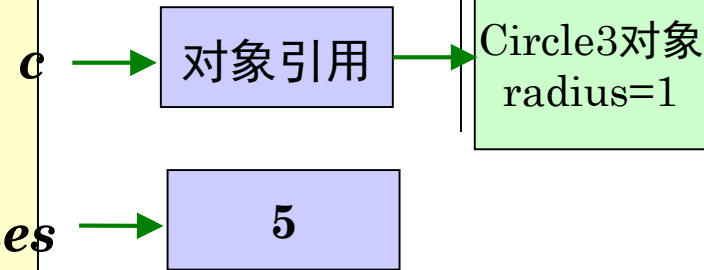
```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```



# 跟踪执行过程 - 6



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```



# 跟踪执行过程 - 7



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*c*

对象引用

Circle3对象  
radius=2

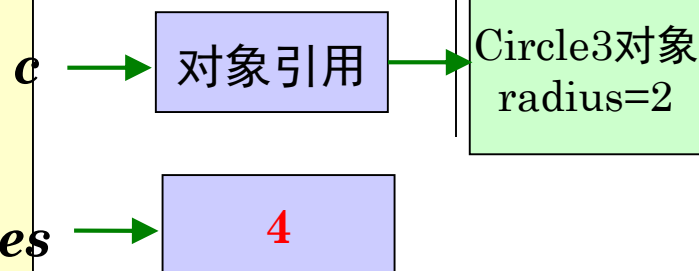
*times*

5

# 跟踪执行过程 - 8



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

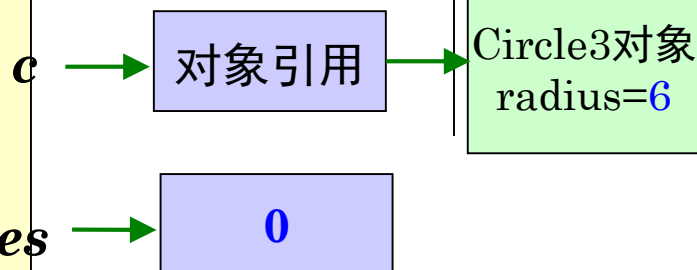


继续执行.....

# 跟踪执行过程 - 9



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

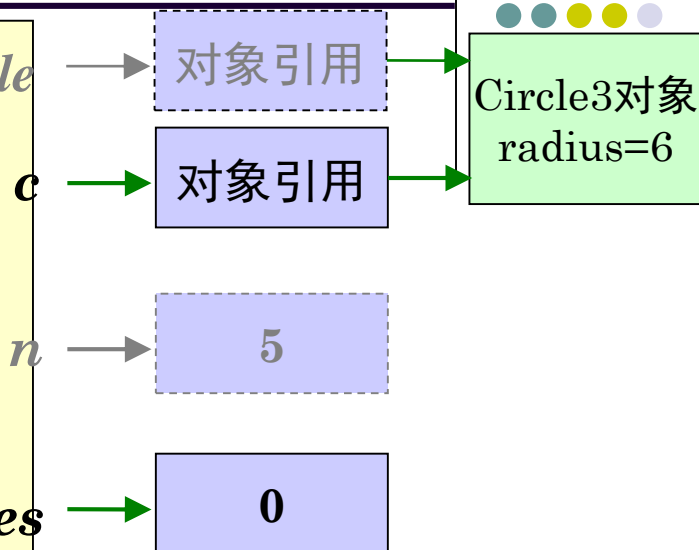


# 跟踪执行过程 - 10



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*myCircle*



# 跟踪执行过程 - 11



```
public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}
```

*c*

对象引用

Circle3对象  
radius=6

*times*

5



```

public class TestPassObject
{
 public static void main(String[] args)
 {
 Circle3 myCircle = new Circle3(1);
 int n = 5;
 printAreas(myCircle, n);
 System.out.print("\n" + "Radius is "
 + myCircle.getRadius());
 System.out.println(", n is " + n);
 }
 public static void printAreas(Circle3 c,
 int times)
 {
 System.out.println("Radius \t\tArea");
 while (times >= 1)
 {
 System.out.println(c.getRadius()
 + "\t\t" + c.getArea());
 c.setRadius(c.getRadius() + 1);
 times--;
 }
 }
}

```

```

public class Circle3
{
 private double radius;
 public Circle3(double newRadius)
 {
 radius = newRadius;
 }
 public double getRadius()
 {
 return radius;
 }
 public void setRadius(double newRadius)
 {
 radius = newRadius;
 }
 public double getArea()
 {
 return radius * radius * Math.PI;
 }
}

```

输出结果:

| Radius                | Area               |
|-----------------------|--------------------|
| 1.0                   | 3.141592653589793  |
| 2.0                   | 12.566370614359172 |
| 3.0                   | 28.274333882308138 |
| 4.0                   | 50.26548245743669  |
| 5.0                   | 78.53981633974483  |
| Radius is 6.0, n is 5 |                    |

## 3.2.6 成员方法参数



### ■ 基本数据类型变量和引用类型变量的区别

#### 基本类型:

`int i = 1` → `i` 1

#### 对象类型:

`Circle c = new Circle()` → `c` 

Reference

 → 

c:Circle  
radius:1.0

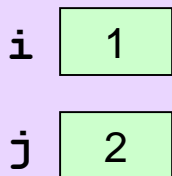
## 3.2.6 成员方法参数



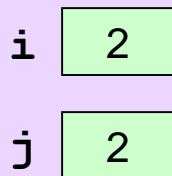
### ■ 拷贝基本数据类型变量和引用类型变量

基本类型赋值:  $i = j$

赋值之前

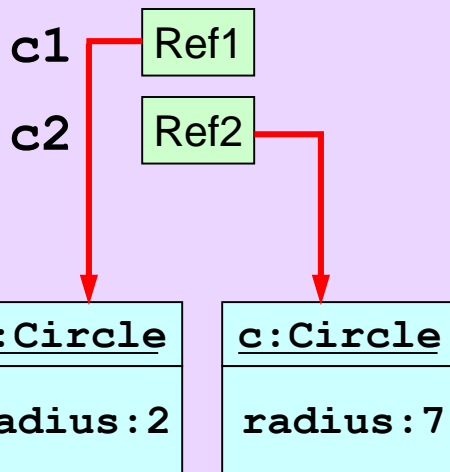


赋值之后

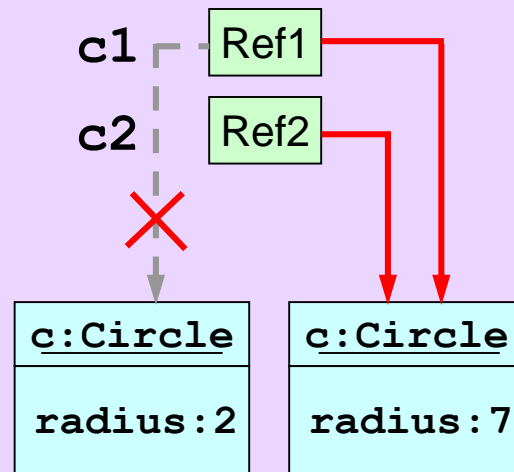


对象类型赋值:  $c1 = c2$

赋值之前



赋值之后



# 第三章 Java面向对象程序设计-1



## 3.1 面向对象程序设计方法概述

## 3.2 类与对象

## 3.3 对象初始化和回收

## 3.4 应用举例

## 3.3 对象初始化和回收



### ■ 对象初始化

- 系统在生成对象时，会为对象分配内存空间，并自动调用**构造方法**对实例变量进行初始化

### ■ 对象回收

- 对象不再使用时，系统会调用**垃圾回收程序**将其占用的内存回收

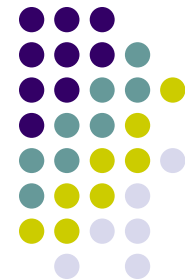
## 3.3.1 构造方法



### ■ 构造方法 —— 对象初始化

- 系统在生成对象时，会为对象分配内存空间，并自动调用构造方法对实例变量进行初始化
- 一种和类同名的特殊方法
- 用来初始化对象
- Java中的每个类都有构造方法，用来初始化该类的一个新的对象
- 没有定义构造方法的类，系统自动提供默认的构造方法

## 3.3.1 构造方法



### ■ 举例

```
import java.math.*;
class Circle
{
 double radius = 1.0;
 Circle()
 {
 }
 Circle(double rads)
 {
 radius = rads;
 }
 double getArea()
 {
 return radius * radius * Math.PI;
 }
}
```

## 3.3.1 构造方法



### ■ 构造方法的特点

- 方法名与类名相同
- 没有返回类型，修饰符void也不能有
- 通常被声明为公有的（public）
- 可以有任意多个参数
- 主要作用是完成对象的初始化工作
- 不能在程序中显式地调用
- 在创建一个对象时使用 new 操作符时，系统会自动调用该类的构造方法为新生成的对象初始化



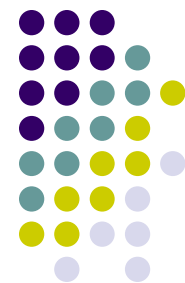
## 3.3.1 构造方法



### ■ 举例

```
import java.math.*;
public class Circle
{
 public double radius;
 public double getArea()
 {
 return radius * radius * Math.PI;
 }
}
```

## 3.3.1 构造方法



### ■ 系统提供的默认构造方法

- 如果在类的声明中没有声明构造方法，则Java编译器会提供一个默认的构造方法
- 默认的构造方法没有参数，其方法体为空
- 使用默认的构造方法初始化对象时，如果在类声明中没有给实例变量赋初值，则对象的属性值为零或空



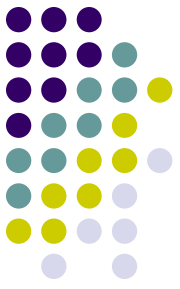
## ■ 使用默认构造方法：声明一个银行帐号类及测试代码

```
class BankAccount{
 String ownerName;
 int accountNumber;
 float balance;
}

public class BankTester{
 public static void main(String args[]){
 BankAccount myAccount = new BankAccount();
 System.out.println("ownerName=" + myAccount.ownerName);
 System.out.println("accountNumber=" + myAccount.accountNumber);
 System.out.println("balance=" + myAccount.balance);
 }
}
```

## ■ 运行结果

ownerName=null  
accountNumber=0  
balance=0.0



## ■ 自定义构造方法与方法重载

- 可在生成对象时给构造方法传送初始值，使用希望的值给对象初始化
- 构造方法可以被重载，构造方法的重载和方法的重载一致

```
public BankAccount(String initName, int initAccountNumber, float initBalance)
{
 ownerName = initName;
 accountNumber = initAccountNumber;
 balance = initBalance;
}
```

```
public BankAccount(String initName, int initAccountNumber) {
 ownerName = initName;
 accountNumber = initAccountNumber;
 balance = 0.0f;
}
```

## 3.3.1 构造方法



### ■ 自定义无参的构造方法

- 无参的构造方法对其子类的声明很重要。
- 如果在一个类中不存在无参的构造方法，则要求其子类声明时必须声明构造方法，否则在子类对象的初始化时会出错
- 在声明构造方法时，好的声明习惯是
  - 不声明构造方法
  - 如果声明，至少声明一个无参构造方法

## 再声明一个无参的构造方法

```
public BankAccount() {
 ownerName = "";
 accountNumber = 999999;
 balance = 0.0f;
}
```

### ■ 构建一个Circle类，有两个：

```
class Circle {
 Circle(int i) {.....
 Circle(double d)
}
```

- 如果写： `new Circle();`

编译器将要告诉你找不到对应的构造方法

### ■ 说明

- 用户在进行类声明时，如果没有声明任何构造方法，系统会赋给此类一个默认（无参）的构造方法。
- 但是，只要用户声明了构造方法，即使没有声明无参的构造方法，系统也不再赋默认的构造方法。

## 3.3.1 构造方法



```
class Circle
{
 double radius = 1.0;
 double getArea()
 {
 return radius*radius*Math.PI;
 }
}
```

声明了一个需要double参数构造方法  
系统不会隐含声明一个默认构造方法  
Circle c1 = new Circle(); // 错误  
Circle c2 = new Circle(5.0); // 正确

没有声明任何构造方法  
系统会隐含声明一个默认构造方法  
Circle c1 = new Circle(); // 正确  
Circle c2 = new Circle(5.0); // 错误

```
class Circle
{
 double radius = 1.0;
 Circle(double rads)
 {
 radius = rads;
 }
 double getArea()
 {
 return radius*radius*Math.PI;
 }
}
```

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

输出结果:

The area of the circle of radius 5.0 is 78.53981633974483  
The area of the circle of radius 1.0 is 3.141592653589793  
The area of the circle of radius 100.0 is 31415.926535897932



## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}
```

输出结果:

The area of the circle of radius 5.0 is 78.53981633974483

```
class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

程序  
执行  
过程



## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

输出结果:

The area of the circle of radius 5.0 is 78.53981633974483

The area of the circle of radius 1.0 is 3.141592653589793

程序  
执行  
过程



## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

跟踪  
程序  
执行  
过程

## ■ Demo

```
public class TestCircle{
 public static void main(String[] args){
 Circle myCircle = new Circle(5.0);
 System.out.println("The area of the circle of radius "
 + myCircle.radius + " is " + myCircle.getArea());

 Circle yourCircle = new Circle();
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());

 yourCircle.radius = 100;
 System.out.println("The area of the circle of radius "
 + yourCircle.radius + " is " + yourCircle.getArea());
 }
}

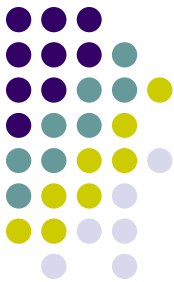
class Circle{
 double radius;
 Circle() {
 radius = 1.0;
 }
 Circle(double newRadius) {
 radius = newRadius;
 }
 double getArea() {
 return radius * radius * Math.PI;
 }
}
```

输出结果:

The area of the circle of radius 5.0 is 78.53981633974483  
The area of the circle of radius 1.0 is 3.141592653589793  
The area of the circle of radius 100.0 is 31415.926535897932

程序  
执行  
过程

## 3.3.1 构造方法



### ■ this关键字

- **this: 指向当前的对象**

类的成员变量名  
方法的参数名  
方法的局部变量

} 同名 → 清楚

- **类（静态）方法中不能使用**



## 3.3.1 构造方法



### ■ this关键字使用

- 使用 this 指向调用实例方法的对象。
- 使用 this 指向一个实例的对象域。
- 使用 this 调用同一类的重载的构造函数。

## 3.3.1 构造方法



### ■ this: 指向当前的对象

- 方法的参数名可以和类的成员变量同名

- class Circle

```
{ int x, y, radius;
 public Circle (int x, int y, int radius)
 { this.x = x;
 this.y = y;
 this.radius = radius;
 }
}
```

## 3.3.1 构造方法



### ■ 作为调用对象的代理，指向当前的对象

```
public class Circle
{
 private double radius = 1.0;
 private static int numberOfObjects;

 public void setRadius(double radius)
 {
 this.radius = radius;
 }
 public static void setNumberOfObjects(
 int numberOfObjects)
 {
 Circle.numberOfObjects = numberOfObjects;
 }
}
```

```
public class CircleTest
{
 public static void main(String[] args)
 {
 Circle c1 = new Circle();
 Circle c2 = new Circle();

 c1.setRadius(2.0);
 c2.setRadius(3.0);
 }
}
```

当调用 `c1.setRadius(2.0)` 时，执行了 `c1.radius=2.0`，此时 `this` 是 `c1`

当调用 `c2.setRadius(3.0)` 时，执行了 `c2.radius=3.0`，此时 `this` 是 `c2`

## 3.3.1 构造方法



### ■ 调用重载的构造函数

```
public class Circle
{
 private double radius;

 public Circle(double radius)
 {
 this.radius = radius; ← 传入的参数
 }
 public Circle()
 {
 this(1.0); → 本对象的 radius 属性。去掉 this, 就会有歧义
 }
 public double getArea()
 {
 return this.radius * this.radius * Math.PI;
 }
}
```

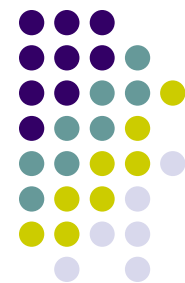
引用本对象的 radius 属性, 这里 this 可以省略

## 3.3.2 内存回收机制



- 当一个对象在程序中不再被使用时，就成为一个无用对象
  - 当前的代码段不属于对象的作用域
  - 把对象的引用赋值为空
- Java运行时系统通过垃圾收集器周期性地释放无用对象所使用的内存
- Java运行时系统在对对象进行自动垃圾回收前，自动调用对象的`finalize()`方法

## 3.3.2 内存回收机制



### ■ 垃圾收集器

- 自动扫描对象的动态内存区，对不再使用的对象做上标记以进行垃圾回收
- 作为一个线程运行
  - 通常在系统空闲时异步地执行
  - 当系统的内存用尽或程序中调用System.gc()要求进行垃圾收集时，与系统同步运行

## 3.3.2 内存回收机制



### ■ finalize()方法

- 在类java.lang.Object中声明，因此 **Java中的每一个类都有finalize()方法**
- 用于释放系统资源，如关闭打开的文件或socket等
- 声明格式

**protected void finalize() throws throwable**

- 如果一个类需要释放除内存以外的资源，则需在类中重写finalize()方法

## 3.3.2 内存回收机制



### ■ 同C和C++的区别

- C语言中通过`free`来释放内存
- C++中则通过`delete`来释放内存
- 在C和C++中，如果程序员忘记释放内存，则容易造成内存泄漏甚至导致内存耗尽
- 在Java中不会发生内存泄漏情况，但对于其它资源，则有产生泄漏的可能性



# 第三章 Java面向对象程序设计-1



## 3.1 面向对象程序设计方法概述

## 3.2 类与对象

## 3.3 对象初始化和回收

## 3.4 应用举例

# 内容小结



## ■ 内容

- 面向对象程序设计的基本概念和思想
- Java语言类与对象的基本概念和语法，包括类的声明、类成员的访问，方法的重载，以及对象的构造、初始化和回收

## ■ 要求

- 理解类和对象的概念
- 熟练使用类及其成员的访问控制方法
- 掌握方法重载的含义，并熟练应用
- 熟练掌握各种构造方法
- 了解Java的垃圾回收机制