



类和对象

李玮玮

讲授思路

- 类和对象概述
- 类的成员方法
- 垃圾回收机制
- 包的使用

类和对象概述

- 面向过程与面向对象的程序设计
- 类与对象概述
- 类的定义
- 对象的实例化
- 类成员的访问

面向过程的程序设计

- 举例：实现五子棋功能

- 面向过程的设计思路是首先分析问题的步骤：

- 开始游戏，
 - 黑子先走，
 - 绘制画面，
 - 判断输赢，
 - 轮到白子，
 - 绘制画面，
 - 判断输赢，
 - 返回步骤2，
 - 输出最后结果。



- 把上面每个步骤用分别的函数来实现，问题就解决了。

面向过程的程序设计

- 举例：实现五子棋功能

面向过程的程序设计存在的问题：

- 1.在解决问题时，从问题的细节入手。适用于解决简单问题，当程序大到一定程度的时候，其调试和维护就很困难。
- 2.面向过程的程序设计思维方式更贴近计算机，不利于大规模的程序设计、对代码重用支持的不够好。

面向对象的编程

- 面向对象编程的思想更接近于人的思维，程序用对象及对象间的相互作用来完成程序的功能，程序中的对象是对现实生活中存在的对象的抽象。
- 面向对象的程序设计可以很好的解决面向过程的程序设计出现的大规模编程，代码复用等问题。
- 面向对象程序设计三个重要特征是封装、继承、多态。

面向对象的设计思想（OOP）

- 面向对象的设计思想
 - 面向对象程序设计的基本思想是将现实世界中的事物抽象为对象，并给抽象出来的对象赋予相应的状态和行为，通过对消息的响应完成一定的任务。
 - 在现实世界中任何事物都可以被认为是对象，如：
 - 学生、教师
 - 课程、教室、班级
 - 计算机、电视机、空调等。

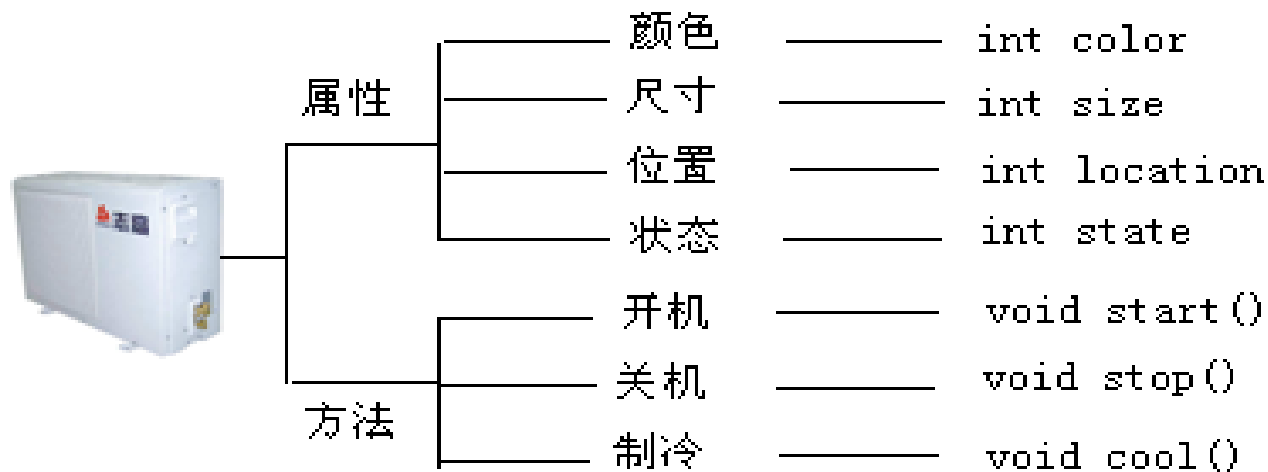


面向对象的设计思想（ OOP ）

- 思考在OOP设计中对象的共性是什么？
 - 具有一定的状态（数据）:静态属性
 - 具有一定的行为（功能）:动态属性
- 例如：一台空调，其状态包括颜色、尺寸、位置、当前运转状态（是否处于开机、关机、或制冷状态）等静态属性；行为包括开机、关机、制冷等动态属性。

面向对象的设计思想（OOP）

- 在面向对象的程序中，对象的状态称为对象的“属性”，对象的行为或功能称为对象的“方法”，一个对象的方法实现对象的一项功能。
- 面向对象程序设计方法就是把现实世界中对象的状态和行为抽象为程序设计语言中的对象，达到二者的统一。

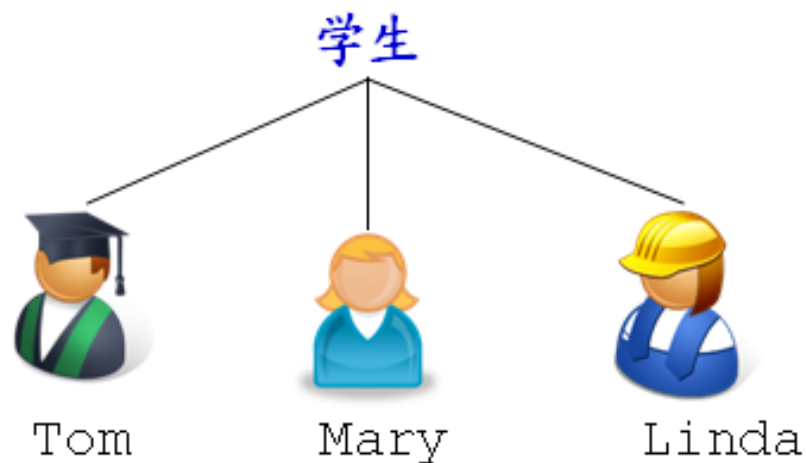


主要概念-抽象

- 现实世界中的对象很多，我们不可能为每一个对象都来定义一组属性和方法。而且这样做会产生大量重复性的工作，因为现实世界中的很多对象是有共性的。
- 例如：学生类，每个同学都是一个对象，各个对象之间有很多共性的。

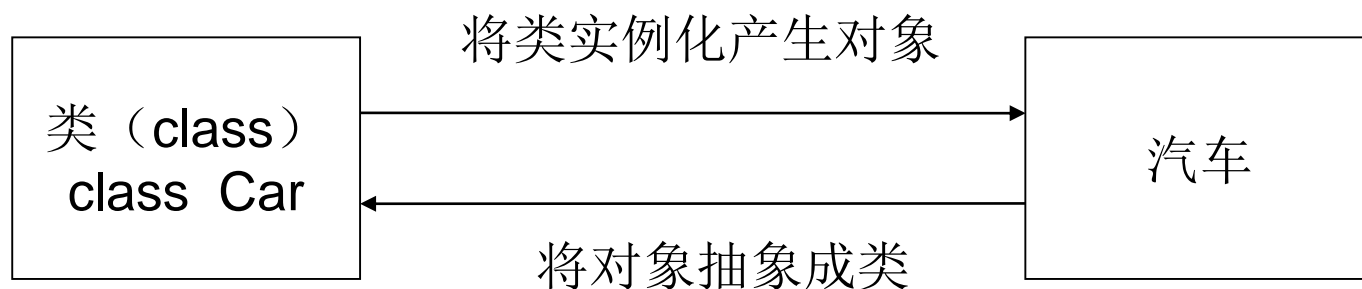
主要概念-抽象

- 我们需要对现实中的对象进行分类和抽象，将具有共性的一类对象（即具有相同的属性和方法）抽象为一个类。例如学生类、空调类、汽车类等。



主要概念-类与对象

- 类：是一种抽象的数据类型，是同种对象的集合与抽象，是具有共同行为和属性的若干对象的统一描述体。
- 对象：现实世界中某个存在的实体在计算机逻辑中的映射和体现。
- 类与对象的关系。



类的定义

- 类声明的语法:
 - [修饰符] class 类名 {
 - [private/protected/public] 成员的声明和定义 ;
 - }
 - 其中 , private、public及protected被称为访问修饰符。
 - 举例 : 定义一个学生类。

```
class Student {  
    public String name;  
    public String sex;  
    public int age;  
    public void study() {  
    }  
}
```

类的定义

- 类的成员
 - 属性：
 - name
 - sex
 - age
 - 方法：
 - void study()

对象的实例化

- 类只是抽象的数据类型，类对象才是具体可操作的实体。
利用对象使用类提供的功能：
- 对象的实例化格式：
 - <类名> <对象名> = new <类名> ([参数1, 参数2,...]);
 - 或
 - <类名> <对象名> ;
 - <对象名> = new <类名> ([参数1, 参数2,...]);
 - 举例：为学生类实例化对象。

```
Student mary = new Student();  
Student lily = new Student();
```

类成员的访问

- 访问属性的一般形式：
 - <对象名>.<属性名>
- 访问方法的一般形式：
 - <对象名>.<方法名>([<参数1> , <参数2> , ...]);
- 方法中参数的个数、数据类型与原方法中定义的要一致。
- 举例：访问学生类的成员。

```
Student mary = new Student();  
mary.name = "mary";  
mary.sex = "female";  
mary.age = 20;  
mary.study();
```


类体-成员方法

- Java中方法只能定义在类中，称为类的成员方法，基本的语法：
 - [方法修饰符] 方法返回值 方法名([<参数列表>]) {
 - 方法体;
 - }
- 成员方法修饰符：主要有public、private、protect、final、static、abstract和synchronized 7种, 用来说明方法的访问权限。

类体-成员方法

- 方法的返回值类型

- Java语言要求，在成员方法说明中必须指明返回值的类型，如不需要返回值，返回值类型被说明为void。
 - `void fun() {}`
- 返回值用return语句来实现，如有返回值，return语句要带参数，且参数必须与方法中的返回值类型一致。
 - `int fun() { return value(需要返回的值);}`

- 方法名

- 是Java语言合法的标识符。
- 成员方法名一般具有一定的含义。

类体-成员方法

- 参数列表
 - 由逗号分隔的类型及参数名组成，是可选项。类型是Java语言的任何数据类型。

```
void fun(int x , int y) {  
    int sum = 0;  
    sum = x + y;  
    System.out.println( "sum = " + sum);  
}
```

- 方法的调用中需要传递实际参数。

类体-成员方法

- 方法体
 - 包含了实现方法功能的Java语言程序代码。
 - 方法体中可以定义局部变量，它的作用域仅在方法体内。
 - 方法体用“{}”括起来。

主要概念-类与对象

```
public class Student
{
    //定义属性
    private String name ;
    private int age;
    private String stuld ;
    //定义方法
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getStuld() { return stuld; }
    public void setStuld(String stuld) { this.stuld = stuld; }
}
```

主要概念-类与对象

- 属性和方法组合在一起构成类，用来描述学生这类对象的共同特征。
- 可以通过下面的代码操作一个学生类的对象

```
//实例化一个对象
Student stu1= new Student ();
//设置学生对象姓名
stu1.setName( "zhang" );
Student stu2= new Student (); //实例化一个对象
```

类的成员方法

- 方法重载

方法重载

- 类中两个以上的同名方法，参数类型或个数不同，称为方法的重载。
- 方法重载的依据
 - 方法名相同
 - 参数列表必须不同
- 注意
 - 返回值可以不同(返回值不作为重载的依据)
 - 是否静态的也不作为重载依据
 - 重载的方法之间可以互相调用

方法重载

```
public class Dog {  
    Dog() {  
        this.bark();  
    }  
    void bark() { //bark()方法是重载方法  
        System.out.println("no barking!");  
    }  
    void bark(String m,double l) {  
        System.out.println("a barking dog!");  
    }  
    void bark(int a,String n) { //不能以返回值区分重载方法  
        System.out.println("a howling dog");  
    }  
}
```

类的成员方法

- 特殊方法——构造方法

一个完整的类

```
public class Student {  
    //属性定义  
    private String name;  
    //构造方法定义  
    public Student(){  
        System.out.println("this is constructor");  
    }  
    public Student(String name) {  
        this.name = name;  
        System.out.println("the student name is: " + this.name);  
    }  
    //成员方法定义  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
}
```

特殊成员方法-构造方法

- 对象的初始化，谁来完成？
 - 概念：在定义对象时为对象赋初值；
 - 注意：初始化就是开辟内存单元同时对数据成员给出明确的值。
- 构造方法
 - 功能：初始化对象
 - 方法名：与类名相同
 - 参数：可有，可无
 - 返回值：不指定返回值（不能写void）
 - 内容：任意，通常只包含成员赋值语句
 - 调用方法：创建对象时自动调用

特殊成员方法-构造方法

- 在对象的生命周期中构造方法只会调用一次。
- 一个类中如果没有定义构造方法，Java编译器会自动为类产生一个默认的构造方法。默认产生的构造方法是一个无参的，什么也不做的空方法，只要类中有显示声明的构造方法，Java编译就不产生默认的构造方法。
- 在一个类中可以定义多个构造方法，但构造方法的参数列表不能相同。

特殊成员方法-构造方法

```
public class Student {  
    //属性定义  
    private String name;  
    //构造方法定义  
    public Student() {  
        System.out.println("this is constructor");  
    }  
    public Student(String name) {  
        this.name = name;  
        System.out.println("the student name is: " + this.name);  
    }  
    //成员方法定义  
    public String getName() { return name; }  
    public void setName(String name) { this.name = name; }  
}
```

方法传参

- 首先，回顾一下在程序设计语言中有关将参数传递给方法（函数）的一些专业术语。
 - 值传递：表示方法接收的是调用者提供的变量的值。
 - 引用传递：表示方法接收的是调用者提供的变量地址。
 - 一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用所对应的变量值。

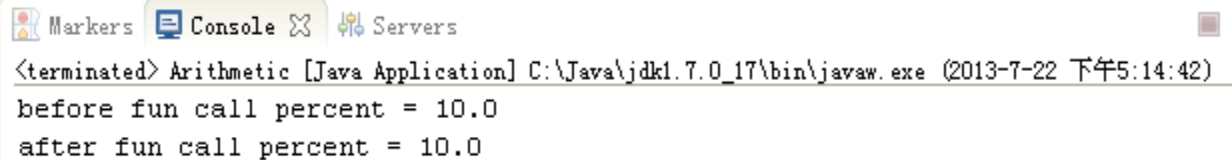
方法传参

- Java中方法参数共有两种类型：
 - 基本数据类型
 - 引用类型

方法传参（基本数据类型）

- 基本数据类型：
 - 方法不会改变实参的值。
- 举例：

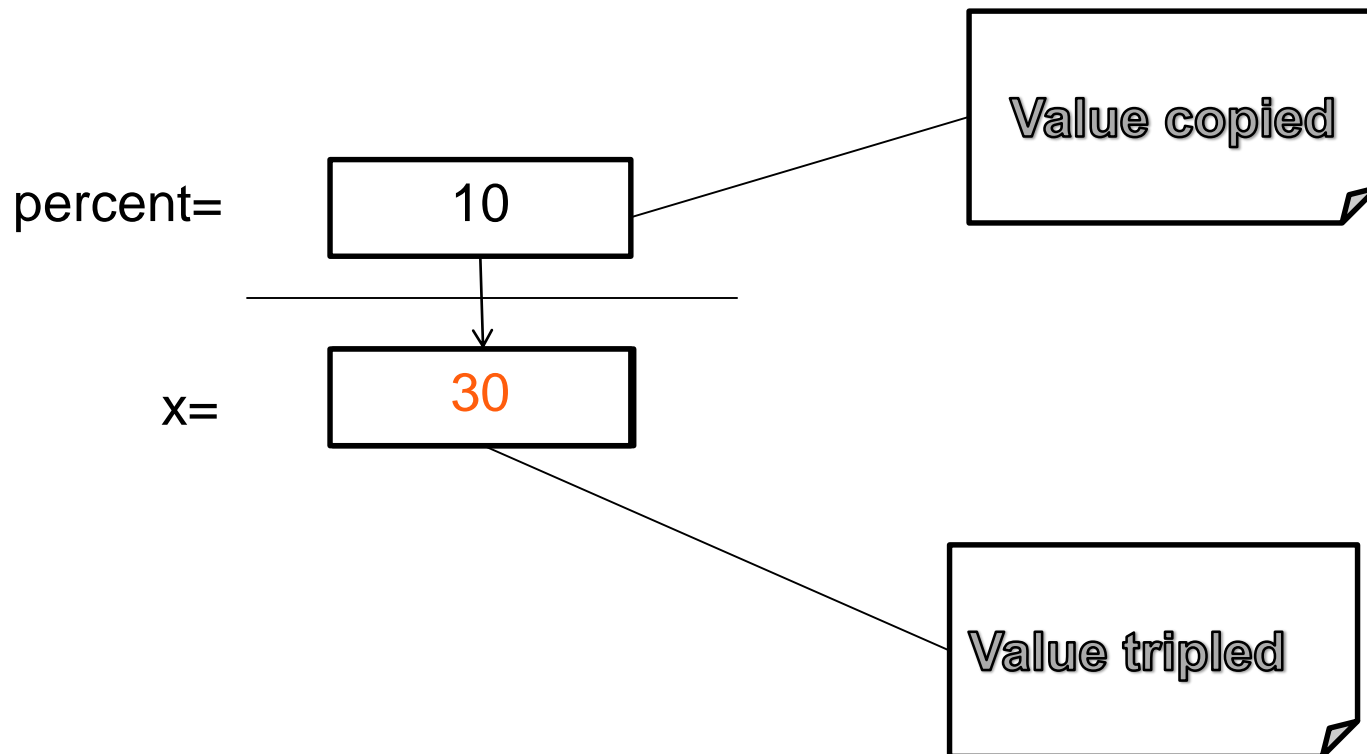
```
public class Arithmetic {  
    public static void main(String[] args) {  
        double percent = 10.0;  
        System.out.println("before fun call percent = "+percent);  
        tripleValue(percent);  
        System.out.println("after fun call percent = "+percent);  
    }  
    public static void tripleValue(double x){  
        x = 3 * x;  
    }  
}
```



```
Markers Console Servers  
<terminated> Arithmetic [Java Application] C:\Java\jdk1.7.0_17\bin\javaw.exe (2013-7-22 下午5:14:42)  
before fun call percent = 10.0  
after fun call percent = 10.0
```

方法传参（基本数据类型）

- 具体执行过程：



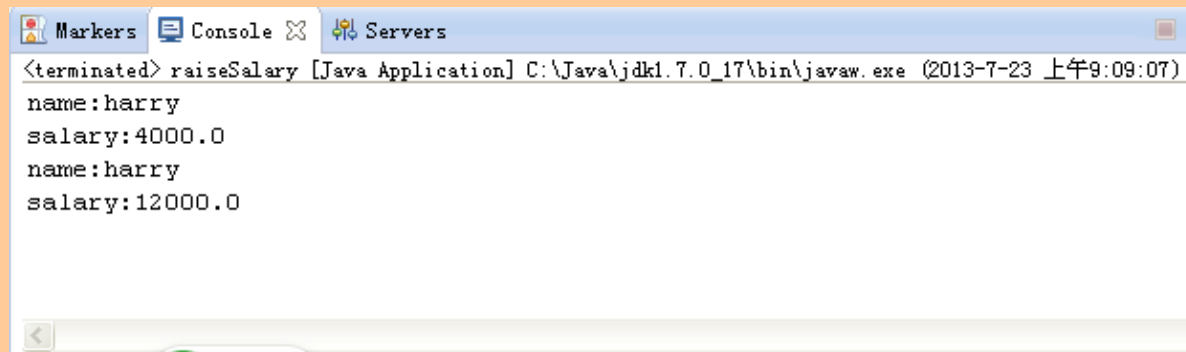
方法传参(引用类型)

- 引用类型：
 - 方法体内，通过引用改变了实际参数对象的内容。
- 举例：
 - 将一个雇员的薪金提高两倍。

```
class Employee{  
    public Employee(String name, double salary){  
        this.name = name;  
        this.salary =salary;  
    }  
    public void printMessage(){  
        System.out.println("name:"+name);  
        System.out.println("salary:"+salary);  
    }  
    public void raiseSalary(double percent){  
        salary = (percent * salary)/100;  
    }  
    private String name;  
    private double salary;  
}
```

方法传参(引用类型)

```
public class raiseSalary {  
    public static void main(String[] args) {  
        Employee harry = new Employee("harry",4000);  
        harry.printMessage();  
        raiseSalary.tripleSalary(harry);  
        harry.printMessage();  
    }  
    public static void tripleSalary(Employee x){  
        x.raiseSalary(300);  
    }  
}
```

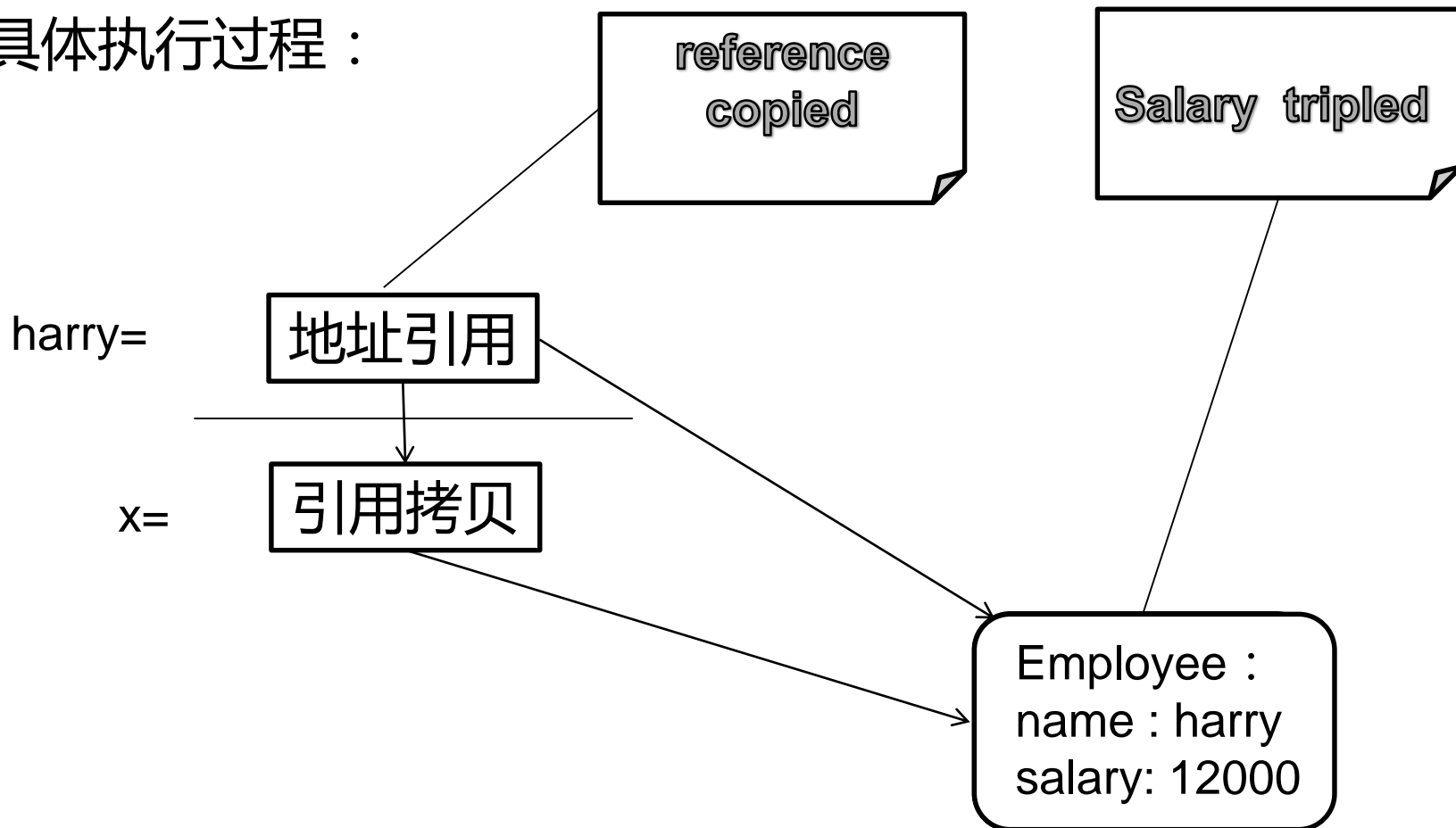


The screenshot shows a Java IDE console window with the following output:

```
<terminated> raiseSalary [Java Application] C:\Java\jdk1.7.0_17\bin\javaw.exe (2013-7-23 上午9:09:07)  
name:harry  
salary:4000.0  
name:harry  
salary:12000.0
```

方法传参（引用类型）

- 具体执行过程：



方法传参

- 总结：
 - 方法不能修改基本数据类型的参数（数值型和布尔型）。
 - 方法可以修改引用类型参数的状态。

课后阅读

- Java参数传递机制：
 - http://blog.sina.com.cn/s/blog_4b622a8e0100c1bo.html
- 对象和对象的引用
 - <http://jingyan.baidu.com/article/a501d80cf734c3ec630f5e25.html>

垃圾收集概述

- 在C或C++程序中，手工清理或删除缓存中的数据
 - 可以通过析构函数进行清理工作
- 手工内存管理缺点
 - 如果程序中存在错误或缺陷会导致内存泄露
 - 编写彻底手工内存管理的代码是一项重要而复杂的任务，所以会使复杂的程序的开发工作量加倍

Java的垃圾收集器

- Java的“垃圾收集器”为内存管理提供了一种自动解决方案
- 对于程序中不再使用的内存资源，“垃圾收集器”能自动将其释放
- 自动垃圾收集的缺点
 - 不能完全控制“垃圾收集器”什么时候执行或不执行

Java垃圾收集器何时运行

- 垃圾收集器受JVM的控制，JVM决定什么时候运行垃圾收集器

1

- 当JVM感到内存不足时会运行垃圾收集器

2

- 在Java程序中可以请求JVM运行垃圾收集器，但无法保证JVM会答应请求

垃圾收集器如何运行

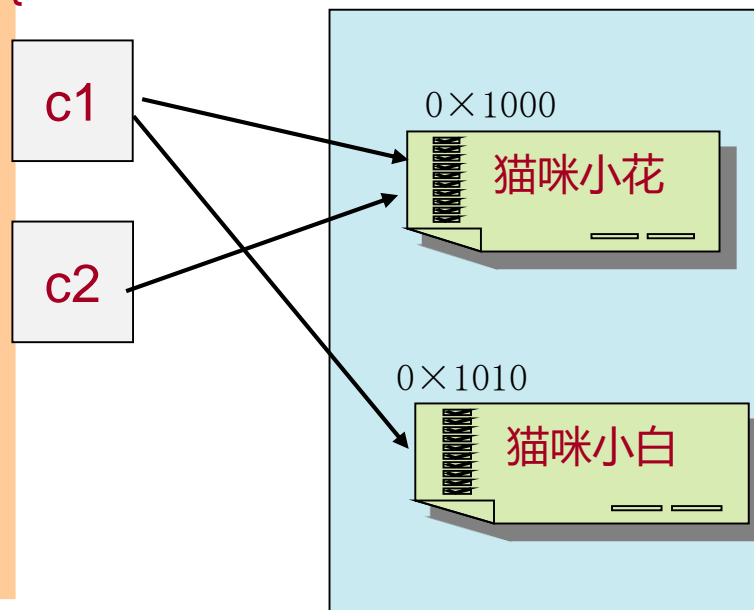
- 对象在何时符合垃圾收集条件
 - 当没有线程能够访问对象时，该对象就是适合进行垃圾收集的

垃圾回收情形一

- 空引用

```
class Cat{  
    public static void main(String[] args){  
        Cat c1;  
        Cat c2;  
        c1 = new Cat("小花");  
        c2 = c1;  
        c1 = new Cat("小白");  
        c2=null;  
    }  
}
```

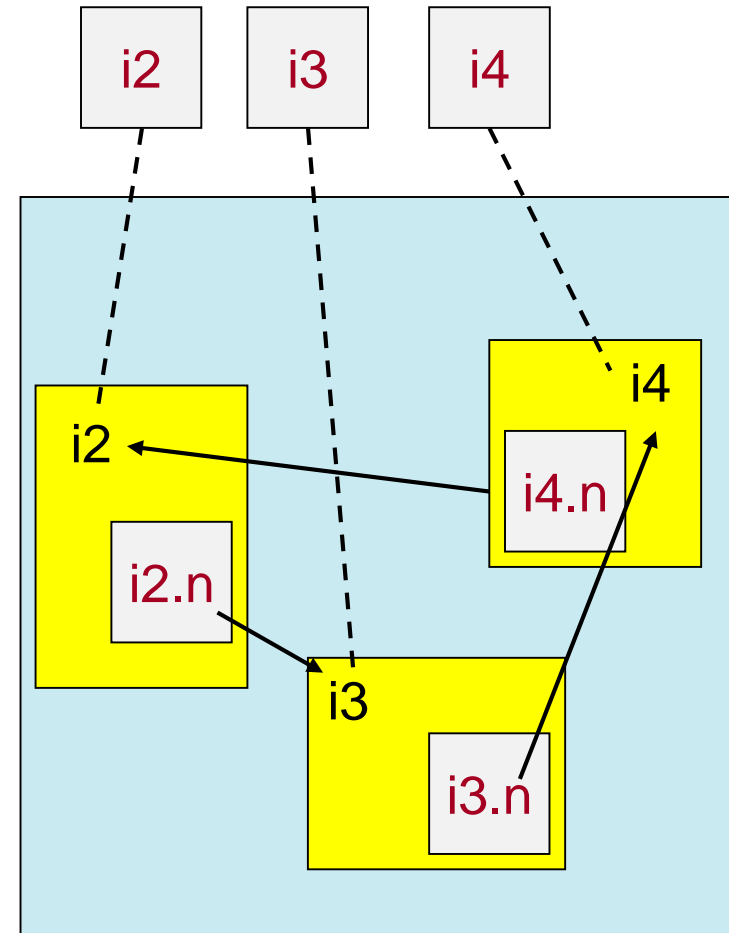
两个动作：创建对象和引用



垃圾回收情形二

- 隔离引用

```
class Island{  
    Island n;  
    public static void main(String[] args){  
        Island i2 = new Island();  
        Island i3 = new Island();  
        Island i4 = new Island();  
        i2.n=i3;  
        i3.n=i4;  
        i4.n=i2;  
        i2 = null;  
        i3 = null;  
        i4 = null;  
    }  
}
```



包的使用

- 包的概念
- 访问带包的类
- Java类库中几个重要的包

包的概念

- 包(package)：命名空间，避免命名冲突。
 - 必须放在源程序的除注释外的第一行。
 - 包的名称就像是我们的姓，而class 名称就像是我们的名字。
 - 如 java.lang.String
- 如果希望自己的类从属于一个包，可以使用package关键字。
- 使用方法：
 - 系统内置的包
 - 自定义包
 - 第三方提供的包

包的概念

- HelloWorld.java

```
package hello;  
public class HelloWorld{  
    public void hello(){  
        System.out.println("HelloWorld");  
    }  
}
```


访问带包的类

- 思考：如何访问另一个包中的公有类？
 - 第一种方式：在每个类名之前添加完整的包名。
 - 第二种方式：通过import关键字引入包。
 - import指定类
 - import整个包

访问带包的类

- 通过全名来访问某包中的类。
 - 如 : org.onest.edu2act.Student
 - 如 : java.lang.String

```
package main;  
public class Main{  
    public static void main(String[] args){  
        hello.HelloWorld h = new hello.HelloWorld();  
        h.hello();  
    }  
}
```

访问带包的类

- 通过import关键字引入包
 - import语句，必须位于package和类声明之间
 - 默认会引入java.lang包
 - import语名的唯一价值是减少键入
- import导入声明可分为两种：
 - 单类型导入(single-type-import)
 - 例:import java.util.ArrayList
 - 按需类型导入(type-import-on-demand)
 - 例:import java.util.*

访问带包的类

- HelloWorld.java

```
package hello;  
public class HelloWorld {  
    public void hello() {  
        System.out.println("HelloWorld");  
    }  
}
```

- Main.java

```
package main;  
public class Main {  
    public static void main(String[] args) {  
        hello.HelloWorld h = new hello.HelloWorld();  
        h.hello();  
    }  
}
```

Java类库中几个重要的包

- java.lang
 - 包含一些形成语言核心的类，如String、Math、Integer和Thread。
- java.awt
 - 包含了构成抽象窗口工具包（AWT）的类，这个包被用来构建和管理应用程序的图形用户界面。
- java.applet
 - 包含了可执行applet 特殊行为的类。

Java类库中几个重要的包

- java.net
 - 包含执行与网络相关的操作的类和处理接口及统一资源定位器 (URLs) 的类。
- java.io
 - 包含处理I/O 文件的类。
- java.util
 - 包含为任务设置的实用程序类，如随机数发生、定义系统特性和使用与日期日历相关的函数。

总结

- 面向过程的程序设计
- 面向对象的设计思想
- 面向对象程序设计的主要概念
- Java中方法的使用
- Java中的垃圾回收器
- Java中的包的概念



Thank You