

# JAVA程序设计

潘微科

感谢：教材《Java大学实用教程》的作者和其他老师提供PowerPoint讲义等资料！  
说明：本课程所使用的所有讲义，都是在以上资料上修改的。

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.1 面向对象编程

- 面向对象编程主要有三个特性
  - **封装（Encapsulation）**：将数据（属性）和对数据的操作（功能）封装在一起，例如“类（class）”的概念。
  - **继承（Inheritance）**：子类可以继承父类的属性和功能，同时可以增加子类独有的属性和功能。
  - **多态（Polymorphism）**：（1）操作名称的多态：多个操作具有相同的名字，但这些操作所接收的消息类型不同；（2）与继承相关的多态：同一操作被不同类型的对象调用时可能产生不同的行为。

## 4.1 面向对象编程

- Java程序设计的基本单位是类
  - 变量：属性、状态
  - 方法：功能

```
class Circle
{
    double radius;
    double getArea()
    {
        double area = 3.14*radius*radius;
        return area;
    }
}

public class Example4_2
{
    public static void main(String args[])
    {
        Circle circle;
        circle = new Circle();
        circle.radius = 1;
        double area = circle.getArea();
        System.out.println(area);
    }
}
```

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.2 类声明和类体

- 类（**class**）是组成Java程序的基本要素
- 类封装了一种类型的对象（**object**）的**变量**和**方法**
- 类是用来定义（**define**）对象的**模板**
- 可以用类创建对象，当使用一个类创建（**create**）一个**对象**时，我们也说给出了这个类的一个**实例**（**instance**）

## 4.2 类声明和类体

- 在语法上，类由两部分构成，**类声明**和**类体**。基本格式为：

```
class 类名  
{  
    类体  
}
```

- class是关键字，用来定义类
- “class 类名”是类的**声明部分**，类名必须是合法的Java标识符
- 两个大括号“{”、“}”以及之间的内容称作**类体**

## 4.2 类声明和类体

- 类的名字不能是Java中的关键字，要符合标识符规定，即名字可以由字母、下划线、数字或美元符号组成，并且第一个字符不能是数字。
- 但是，给类命名时，最好遵守下列习惯：
  - 如果类名使用拉丁字母，那么名字的首字母使用大写字母，如Hello、Time、People等。
  - 类名最好容易识别，见名知意。当类名由几个“单词”复合而成时，每个单词的首字母使用大写，例如，BeijingTime、AmericanGame、HelloChina，等。



# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.3 类体的构成

- 类体内容可以有两种类型的成员：
  - **成员变量（member variable）**：通过变量声明来定义的变量，称作成员变量或域（data field），用来刻画类创建的对象属性、状态
  - **方法（method）**：方法是类体的重要成员之一。其中的构造方法是具有特殊地位的方法，供类创建对象时使用，用来给出类所创建的对象初始状态；另一种方法，可以由类所创建的对象调用，对象调用这些方法来操作成员变量，进而形成一定的算法

## 4.3 类体的构成

- 【例子】

```
class Vehicle
{
    int speed;
    float weight,height;
    void changeSpeed(int newSpeed)
    {
        speed=newSpeed;
    }
    float getWeight()
    {
        return weight;
    }
    float getHeight()
    {
        return height;
    }
}
```

成员变量

方法

## 4.3 类体的构成

- **成员变量的类型：**可以是Java中的任何一种数据类型，包括前面学习过的整型、浮点型、字符型和数组，以及后面要学习的对象（`object`）及接口（`interface`）。
- 成员变量在整个类内都有效，与它在类体中书写的**先后位置无关**。

## 4.3 类体的构成

- 在定义类的成员变量时可以同时**赋予初值**，表明类所创建的对象**的初始状态**。
- 对成员变量的操作**只能放在方法中**。
- 类的成员类型中可以有数据和方法，即数据的定义和方法的定义，**但没有语句，语句必须放在方法中**。

```
class A
{
    int a=9;
    float b=12.6f;
    void f()
    {
        a=12;
        b=12.56f;
    }
}
```

```
class A
{
    int a;
    float b;
    a=12; //非法
    b=12.56f; //非法
    void f()
    {
    }
}
```

a=12是赋值语句，不是数据的声明

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.4 构造方法与对象的创建

- 类中有一部分方法称作**构造方法（constructor）**，类创建对象时需使用构造方法，以便给类所创建的对象一个合理的**初始状态**。
- 构造方法是一种特殊的方法
  - 它的名字必须**与它所在的类的名字完全相同**
  - **不返回任何数据类型**，即它是void型，但void必须省略不写
  - Java允许一个类中有**多个构造方法**，但这些构造方法的**参数必须不同**，即或者是参数的个数不同，或者是参数的类型不同

## 4.4 构造方法与对象的创建

- 【例子】

```
class Rect
{
    double sideA, sideB;
    Rect() //无参数构造方法
    {
    }
    Rect(double a, double b) //有参数构造方法
    {
        sideA=a;
        sideB=b;
    }
    double computeArea()
    {
        return sideA*sideB;
    }
    double computeGirth()
    {
        return (sideA+sideB)*2;
    }
}
```

构造方法



## 4.4 构造方法与对象的创建

- 当使用一个类创建（**create**）一个对象时，我们也说给出了这个类的一个实例（**instance**）。创建一个对象包括：
  - 对象的声明（**declare**）
  - 为对象分配成员变量

## 4.4 构造方法与对象的创建

- **1.对象的声明**
- 一般格式为:

类的名字 对象的名字;

- 例子:

```
Rect rectangleOne;
```

## 4.4 构造方法与对象的创建

- 2.为声明的对象分配成员变量
- 使用**new运算符**和**类的构造方法**为声明的对象分配成员变量，如果类中没有构造方法，系统会调用默认的构造方法（**默认的构造方法是无参数的**）。

- 例子：

```
rectangleOne = new Rect();
```

```
rectangleOne = new Rect(10,20);
```

严格来说：rectangleOne is  
**a variable** that contains a  
reference to a Rect object.

## 4.4 构造方法与对象的创建

- 如果类里定义了一个或多个构造方法，那么Java不提供默认的构造方法。
- 如果上述Rect类只提供一个带参数的构造方法，那么如下语句为非法

```
rectangleOne = new Rect();
```

非法

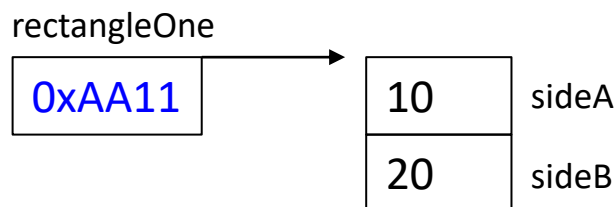
- 创建对象的代码：

```
rectangleOne = new Rect(10,20);
```

  - 为成员变量分配内存空间，然后执行构造方法中的语句
  - 给出一个信息，已确保这些成员变量是属于对象rectangleOne的

## 4.4 构造方法与对象的创建

- 创建对象就是指为它分配成员变量，并获得一个引用（reference），以确保这些成员变量由它来“操作管理”
- 为对象分配成员变量后，内存模型变成如下图所示，箭头示意对象可以操作这些属于自己的成员变量



## 4.4 构造方法与对象的创建

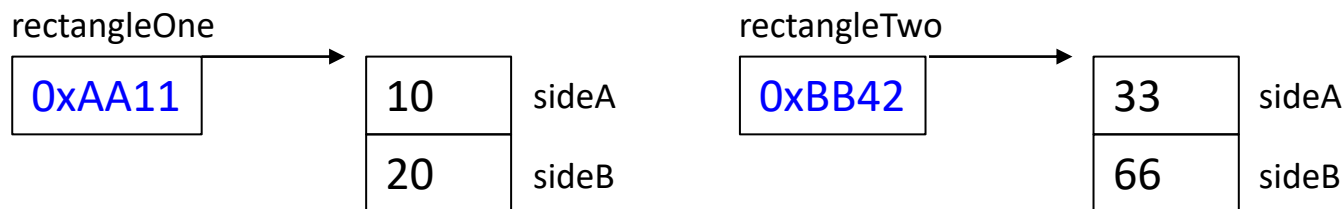
- **3.创建多个不同的对象**
- 一个类通过使用new运算符可以创建多个不同的对象，这些对象将被分配不同的内存空间，因此，改变其中一个对象的状态不会影响其它对象的状态。

- 例子：

```
rectangleOne=new Rect(10,20);
```

```
rectangleTwo=new Rect(33,66);
```

- 内存模型如图所示



## 4.4 构造方法与对象的创建

- **4.使用对象**
- 对象不仅可以操作自己的**变量**来改变状态，而且还拥有了使用创建它的那个类中的**方法**的能力，对象通过**使用这些方法**可以产生一定的行为（进而形成一个算法）
- 通过使用**运算符“.”**，对象可以实现对变量的访问（**access**）和方法的调用（**invoke**）
  - 对象操作自己的**变量**（对象的属性）
  - 对象调用类中的**方法**（对象的功能）

## 4.4 构造方法与对象的创建

- 【例子】

```
class Lader
{
    double above,bottom,height;
    Lader(){}
    Lader(double a,double b,double h)
    {
        above=a;
        bottom=b;
        height=h;
    }
    public void setAbove(double a)
    {
        above=a;
    }
    public void setBottom(double b)
    {
        bottom=b;
    }
    public void setHeight(double h)
    {
        height=h;
    }
    double computeArea()
    {
        return (above+bottom)*height/2.0;
    }
}
```

```
public class Example4_1
{
    public static void main(String args[])
    {
        double area1=0,area2=0;
        Lader laderOne,laderTwo;
        laderOne=new Lader();
        laderTwo=new Lader(10,88,20);
        laderOne.setAbove(16);
        laderOne.setBottom(26);
        laderOne.setHeight(100);
        laderTwo.setAbove(300);
        laderTwo.setBottom(500);
        area1=laderOne.computeArea();
        area2=laderTwo.computeArea();
        System.out.println(area1);
        System.out.println(area2);
    }
}
```

```
2100.0
8000.0
```



# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.5 对象的引用与实体

- 我们已经知道，当用类创建（create）一个对象时，成员变量被分配内存空间，这些内存空间称为该对象的实体（entity）或变量，而对象中存放着引用（reference），以确保这些变量由该对象操作使用。
- 因此，如果两个对象有相同的引用，那么就具有相同的实体。


## 4.5 对象的引用与实体

- Java具有“**垃圾收集**”（garbage collection）机制，Java的运行环境周期性地检测某个实体**是否已不再被任何对象所引用**，如果发现这样的实体，就释放该实体占有的内存。因此，Java编程人员不必像C++程序员那样，要自己时刻检查哪些对象应该释放内存。

```
rectangleOne=new Rect(10,20);
```

```
rectangleTwo=new Rect(33,66);
```

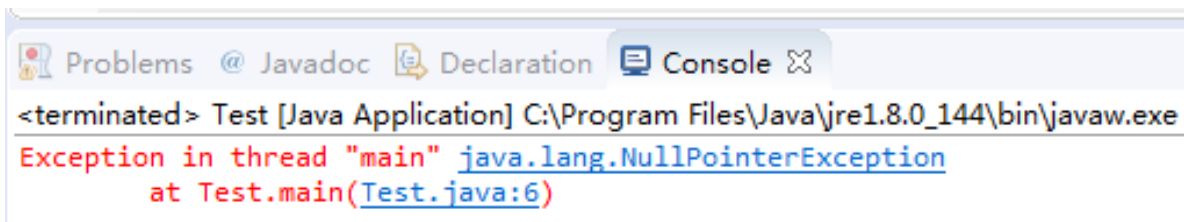
```
rectangleTwo=rectangleOne;
```



## 4.5 对象的引用与实体

- 没有实体的对象称作**空对象**。空对象不能使用，即不能让一个空对象去调用方法产生行为。假如程序中使用了空对象，程序在运行时会出现异常，即**NullPointerException**。由于对象是动态地分配实体，所以Java的编译器对空对象不做检查。因此，在编写程序时要避免使用空对象。

```
public class Test
{
    public static void main(String[] args)
    {
        String [] s = new String[10];
        System.out.println(s[0].length());
    }
}
```




# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.6 成员变量

- 用关键字**static**修饰的成员变量称作**静态变量**（**static variable**）或**类变量**（**class variable**），而没有使用**static**修饰的成员变量称作**实例变量**（**instance variable**）。

```
class A
{
    float x;
    static int y;
}
```



- 静态变量**是与类相关联的数据变量，也就是说，**静态变量**是和该类所创建的所有对象相关联的变量，改变其中一个对象的这个**静态变量**就同时改变了其它对象的这个**静态变量**。
- 因此，**静态变量**不仅可以**通过某个对象访问**也可以直接**通过类名访问**。  
注：通过类名访问静态变量是一个好的编程习惯。

## 4.6 成员变量

- **实例变量**仅仅是和相应的对象关联的变量，也就是说，不同对象的实例变量互不相同，即**分配了不同的内存空间**（注：从内存模型的角度来理解），改变其中一个对象的实例变量不会影响其它对象的这个实例变量。
- 实例变量必须通过对象访问（不能通过类名访问）。

## 4.6 成员变量

- 【例子】

```
class Lader
{
    double above,height; //实例变量
    static double bottom; //静态变量
    void setAbove(double a)
    {
        above=a;
    }
    void setBottom(double b)
    {
        bottom=b;
    }
    double getAbove()
    {
        return above;
    }
    double getBottom()
    {
        return bottom;
    }
}
```

```
public class Example4_2
{
    public static void main(String args[])
    {
        Lader.bottom=60;
        Lader laderOne,laderTwo;
        System.out.println(Lader.bottom);
        laderOne=new Lader();
        laderTwo=new Lader();
        System.out.println(laderOne.getBottom());
        System.out.println(laderTwo.getBottom());

        laderOne.setAbove(11);
        laderTwo.setAbove(22);
        laderTwo.setBottom(100);
        System.out.println(Lader.bottom);
        System.out.println(laderOne.getAbove());
        System.out.println(laderTwo.getAbove());
    }
}
```

```
60.0
60.0
60.0
100.0
11.0
22.0
```



## 4.6 成员变量

- 2.常量

- 如果一个成员变量修饰为**final**，就是常量，不能更改，常量的名字习惯用**大写字母**，例如：

```
final int MAX=100;
```

- **final**修饰的成员变量**不占用内存**，这意味着在声明**final**成员变量时，**必须要初始化**。
- 对于**final**修饰的成员变量，可以通过对象访问，但**不能通过类名访问**。
- 注：在实际开发中，常量通常是不同对象间共享的静态变量，因此会同时用**final** 和**static**来修饰（通常用类名来访问）。

## 4.6 成员变量

- 【例子】

```
class Tom
{
    final int MAX=100;
    final static int MIN=20;
}

public class Example4_3
{
    public static void main(String args[])
    {
        System.out.println(Tom.MIN);
        //System.out.println(Tom.MAX); // Error
        Tom cat = new Tom();
        System.out.println(cat.MAX);
    }
}
```

```
20
100
```

## 4.6 成员变量

- 成员变量的类型
  - **instance** variable
  - **static** variable (or class variable)
  - **final** variable (or constant)

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.7 方法

- 类体内容可以有两种类型的成员：成员变量和方法
- 方法
  - 实例方法（**instance method**）
  - 静态方法（**static method**），又称类方法
  - 构造方法（**constructor**）
- 方法的定义包括两部分：**方法声明**和**方法体**。一般格式为：

```
方法声明  
{  
    方法体  
}
```

## 4.7 方法

- 1.方法声明和方法体
- 最基本的方法声明包括方法名和方法的返回类型，返回类型也称为方法的类型。

```
float area()  
{  
    ...  
}
```

## 4.7 方法

- 方法的名字必须符合标识符规定。在给方法起名字时应遵守习惯。名字如果使用拉丁字母，**首字母要小写**。如果由多个单词组成，从第2个单词开始的首字母使用大写。例如：

```
float getTriangleArea()  
void setCircleRadius(double radius)
```

- 方法声明之后的一对大括号“{”、“}”以及之间的内容称作方法的方法体。
- 类中的方法必须要有方法体**，如果方法的类型是void类型，方法体中也可以不书写任何语句。

## 4.7 方法

- 2.方法体的构成
- 方法体的内容包括变量的定义和合法的Java语句，在方法体中声明的变量以及方法的参数称作局部变量，局部变量仅仅在该方法内有效。
- 方法的参数在整个方法内有效，方法内定义的局部变量从它定义的位置之后开始有效。
- 写一个方法和C语言中写一个函数完全类似，只不过在Java中一般称作方法（method）。
- 局部变量的名字必须符合标识符规定，遵守习惯。名字如果使用拉丁字母，首字母使用小写。如果由多个单词组成，从第2个单词开始的首字母使用大写。



## 4.7 方法

- 3.实例方法与类方法
- 实例方法（**instance method**）可以调用该类中的实例方法、静态方法
- 实例方法：不用static修饰的方法
- 静态方法（**static method**）只能调用该类的静态方法，不能调用实例方法
- 静态方法：方法声明中用关键字static修饰的方法
- 静态方法又称类方法
- All methods in the **Math** class are static methods.
- **main**方法也是静态方法

## 4.7 方法

- 【例子】

```
class A
{
    float a,b;
    void sum(float x,float y)
    {
        a=max(x,y);
        b=min(x,y);
    }
    static float getMaxSqrt(float x,float y)
    {
        float c;
        c=max(x,y)*max(x,y);
        return c;
    }
    static float max(float x,float y)
    {
        return x>y?x:y;
    }
    float min(float x,float y)
    {
        return x<y?x:y;
    }
}
```

实例方法

静态方法

静态方法

实例方法

## 4.7 方法

- 实例方法可以操作**实例变量**、**静态变量**
- 静态方法只能操作**静态变量**，不能操作**实例变量**
- 实例方法必须通过对象来调用
- **静态方法**可以通过类名调用，也可以通过对象来调用。
- **注：通过类名调用静态方法是一个好的编程习惯。**
- 无论静态方法或实例方法，当被调用执行时，方法中的**局部变量**才被分配内存空间，方法调用完毕，局部变量即刻释放所占的内存

## 4.7 方法

- 【例子】

```
class Computer
{
    double x,y;
    static double max(double a,double b)
    {
        return a>b?a:b;
    }
}
class Example4_4
{
    public static void main(String args[])
    {
        double max = Computer.max(12,45); //类名调用静态方法
        System.out.println(max);
    }
}
```

## 4.7 方法

- 4.参数传值
- 当方法被调用时，如果方法有参数，参数必须要实例化，即**参数变量必须有具体的值**。
- 在Java中，方法的所有参数**都是“传值”的（pass by value）**，也就是说，方法中参数变量的值是调用者指定的值的**拷贝**。方法如果改变参数的值，**不会影响向参数“传值”的变量的值**。
- (1) 基本数据类型参数的传值

## 4.7 方法

- 【例子】

```
class Tom
{
    void f(int x, double y)
    {
        x=x+1;
        y=y+1;
        System.out.printf("f: %d,%3.2f\n",x,y);
    }
}
public class Example4_5
{
    public static void main(String args[])
    {
        int x=10;
        double y=12.58;
        Tom cat=new Tom();
        cat.f(x,y);
        System.out.printf("main: %d,%3.2f\n",x,y);
    }
}
```

```
f: 11,13.58
main: 10,12.58
```

## 4.7 方法

- (2) 引用类型参数的传值
- Java的引用类型数据包括**对象（object）**、**数组（array）**、**接口（interface）**。当参数是引用类型时，“传值”传递的是**变量的引用（reference）**而不是变量所引用的实体（**entity**）。
- 如果改变参数变量所引用的**实体**，就会导致原变量的实体发生同样的变化，因为，两个引用型变量如果具有同样的引用（**reference**），就会用同样的实体（**entity**）。

## 4.7 方法

- 【例子】

```
class Tom
{
    void f(Jerry jerry)
    {
        jerry.setLeg(12);
        System.out.println("f: "
                           + jerry.getLeg());
    }
}
```

```
class Jerry
{
    int leg;
    Jerry(int n)
    {
        leg = n;
    }
    void setLeg(int n)
    {
        leg = n;
    }
    int getLeg()
    {
        return leg;
    }
}
```

```
public class Example4_6
{
    public static void main(String args[])
    {
        Tom tom = new Tom();
        Jerry jerry = new Jerry(2);
        System.out.println("before: " + jerry.getLeg());
        tom.f(jerry);
        System.out.println("after: " + jerry.getLeg());
    }
}
```

```
before: 2
f: 12
after: 12
```



## 4.7 方法

- 【例子】

```
class Circle
{
    double radius;
    Circle(double r)
    {
        radius=r;
    }
    double computeArea()
    {
        return 3.14*radius*radius;
    }
    void setRadius(double newRadius)
    {
        radius=newRadius;
    }
    double getRadius()
    {
        return radius;
    }
}
```

```
class Cone
{
    Circle bottom;
    double height;
    Cone(Circle c, double h)
    {
        bottom = c;
        height = h;
    }
    double computeVolume()
    {
        double volume;
        volume =
        bottom.computeArea()*height/3.0;
        return volume;
    }
    void setBottomRadius(double r)
    {
        bottom.setRadius(r);
    }
    double getBottomRadius()
    {
        return bottom.getRadius();
    }
}
```

```
public class Example4_7
```


```
{
    public static void main(String args[])
    {
        Circle circle = new Circle(1);
        Cone cone = new Cone(circle,1);
        System.out.println("radius: " + cone.getBottomRadius());
        System.out.println("volume:" + cone.computeVolume());
        cone.setBottomRadius(10);
        System.out.println("radius: " + cone.getBottomRadius());
        System.out.println("volume: " + cone.computeVolume());
    }
}
```

```
radius: 1.0
volume:1.0466666666666666
radius: 10.0
volume: 104.66666666666667
```

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.8 方法重载

- 方法重载（**overload**）是指一个类中可以有**多个方法具有相同的名字**，但这些方法的参数必须不同，即或者是**参数的个数不同**，或者是**参数的类型不同**。
  - **方法的返回类型**和参数的名字**不参与比较**，也就是说，如果两个方法的名字相同，即使返回类型不同，也必须保证参数不同。
- 

## 4.8 方法重载

- 【例子】

```
class People
{
    double getArea(double x, int y)
    {
        return x*y;
    }
    int getArea(int x, double y)
    {
        return (int)(x*y);
    }
    double getArea(float x, float y, float z)
    {
        return (x*x+y*y+z*z)*2.0;
    }
}
```

```
public class Example4_8
{
    public static void main(String args[])
    {
        People zhang = new People();
        System.out.println( "Area: " + zhang.getArea(10,8.0) );
        System.out.println( "Area: " + zhang.getArea(10.0,8) );
    }
}
```

```
Area: 80
Area: 80.0
```

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.9 关键字this

- this是Java的一个关键字，可以出现在实例方法（instance method）和构造方法（constructor）中，但不可以出现在静态方法（static method）中。
- 1.在构造方法中使用this
- this关键字可以出现在类的构造方法中，代表使用该构造方法所创建的对象。
- 也可以用来调用其他构造方法：If a class has multiple constructors, it is better to implement them using **this(arg-list)** as much as possible. In general, a constructor with no or fewer arguments can invoke a constructor with more arguments using **this(arg-list)**. This syntax often simplifies coding and makes the class easier to read and to maintain.

## 4.9 关键字this

- 【例子】

```
public class Tom
{
    int leg;
    Tom(int n)
    {
        this.cry(); // or cry();
        leg = n;
        this.cry(); // or cry();
    }
    void cry()
    {
        System.out.println(leg + " legs");
    }
    public static void main(String args[])
    {
        Tom cat = new Tom(4);
        //当调用构造方法Tom时，其中的this就是对象cat
    }
}
```

0 legs  
4 legs

## 4.9 关键字this

- 2.在实例方法中使用this
- this关键字可以出现在类的实例方法中，代表使用该方法的当前对象
- 实例方法可以操作成员变量。实际上，当成员变量在实例方法中出现时，默认的格式是：this.成员变量;

```
class A
{
    int x;
    void f()
    {
        this.x = 100; // or x=100
    }
}
```



## 4.9 关键字this

- 类的实例方法可以调用类的其它方法，调用的默认格式是：this.方法;

```
class B
{
    void f()
    {
        this.g(); // or g();
    }
    void g()
    {
        System.out.println("ok");
    }
}
```

## 4.9 关键字this

- 3.静态方法（static method）中不可以使用this
- 因为静态方法可以通过类名直接调用，**这时可能还没有创建任何对象。**

## 4.9 关键字this

- 4.使用this来区分成员变量和局部变量
- 如果局部变量的名字与成员变量的名字相同，则成员变量被隐藏，即这个成员变量在这个方法内暂时失效。
- 这时，如果想在该方法内使用成员变量，成员变量前面的“this.”就不可以省略。

```
class Triangle
{
    float sideA, sideB, sideC, lengthSum;
    void setSide(float sideA, float sideB, float sideC)
    {
        this.sideA=sideA;
        this.sideB=sideB;
        this.sideC=sideC;
    }
}
```

# Outline

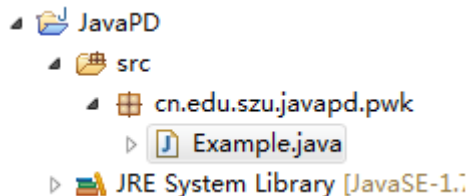
- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.10 包

- 通过关键字package声明包语句。package语句作为Java源文件中的**第一条语句**，指明该源文件定义的类所在的包。
- package语句的一般格式为：`package 包名;`
- 如果源程序中省略了package语句，源文件中所定义命名的类被隐含地认为是**无名包（default package）**的一部分，即源文件中定义命名的类在同一个包中，但该包没有名字。**注：不使用无名包是一个好的编程习惯。**
- 包名可以是一个合法的标识符，也可以是若干个标识符加“.”分割而成，如：
  - package pwk;
  - package cn.edu.szu.javapd.pwk;

## 4.10 包

- 【例子】



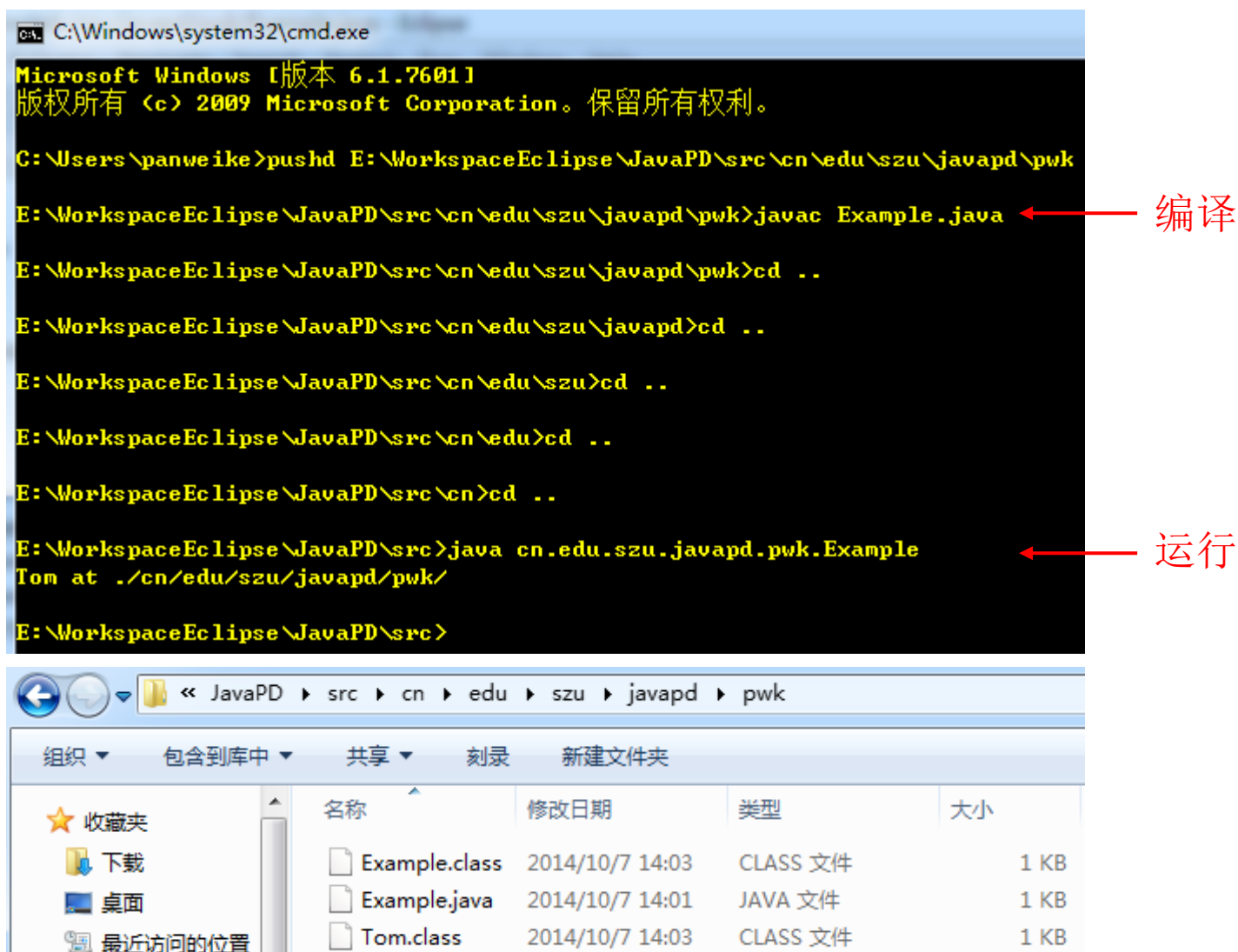
```
package cn.edu.szu.javapd.pwk;

class Tom
{
    void speak()
    {
        System.out.println("Tom
at ./cn/edu/szu/javapd/pwk/");
    }
}

public class Example
{
    public static void main(String args[])
    {
        Tom cat = new Tom();
        cat.speak();
    }
}
```

```
Tom at ./cn/edu/szu/javapd/pwk/
```

## 4.10 包



The screenshot shows a Windows command prompt window with the following commands and output:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk
E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk>javac Example.java
E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk>cd ..
E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd>cd ..
E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu>cd ..
E:\WorkspaceEclipse\JavaPD\src\cn\edu>cd ..
E:\WorkspaceEclipse\JavaPD\src\cn>cd ..
E:\WorkspaceEclipse\JavaPD\src>java cn.edu.szu.javapd.pwk.Example
Tom at ./cn/edu/szu/javapd/pwk/
E:\WorkspaceEclipse\JavaPD\src>
```

Red arrows point from the text "编译" (Compile) to the `javac Example.java` command and from "运行" (Run) to the `java cn.edu.szu.javapd.pwk.Example` command.

Below the command prompt is a file explorer window showing the directory `JavaPD > src > cn > edu > szu > javapd > pwk`. The file list is as follows:

名称	修改日期	类型	大小
Example.class	2014/10/7 14:03	CLASS 文件	1 KB
Example.java	2014/10/7 14:01	JAVA 文件	1 KB
Tom.class	2014/10/7 14:03	CLASS 文件	1 KB

类Example的全名: **cn.edu.szu.javapd.pwk.Example**

或

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src\

E:\WorkspaceEclipse\JavaPD\src>javac .\cn\edu\szu\javapd\pwk\Example.java

E:\WorkspaceEclipse\JavaPD\src>java cn.edu.szu.javapd.pwk.Example
Tom at ./cn/edu/szu/javapd/pwk/

E:\WorkspaceEclipse\JavaPD\src>
```

推荐的方法

← 编译

← 运行

或

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src

E:\WorkspaceEclipse\JavaPD\src>javac -d . .\cn\edu\szu\javapd\pwk\Example.java

E:\WorkspaceEclipse\JavaPD\src>java cn.edu.szu.javapd.pwk.Example
Tom at ./cn/edu/szu/javapd/pwk/

E:\WorkspaceEclipse\JavaPD\src>
```

“-d .” 在当前目录生成.class文件

← 编译

← 运行

或

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk

E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk>javac -d . .\Example.java

E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk>java cn.edu.szu.javapd.pwk.Example
Tom at ./cn/edu/szu/javapd/pwk/

E:\WorkspaceEclipse\JavaPD\src\cn\edu\szu\javapd\pwk>
```

“-d .” 在当前目录生成.class文件

← 编译

← 运行



# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.11 import语句

- 使用import 语句可以**引入包中的类**。在编写源文件时，除了自己编写类，我们经常需要使用Java提供的许多类，这些类可能在不同的包中。
- 在学习Java语言时，使用已经存在的类，**避免一切从头做起**，这是面向对象编程的一个重要方面。
- 1.使用类库中的类
- 在一个Java源程序中可以有多个import语句，它们必须写在package语句（假如有package语句）和源文件中类的定义之间。

## 4.11 import语句


- Java为我们提供了很多包
  - java.applet
  - java.awt
  - java.lang
  - java.io
  - java.net
  - java.util
  - ...
  - 更多资料
    - <https://docs.oracle.com/javase/> JDK 7, ..., JDK 17

## 4.11 import语句

- 如果使用import语句引入了整个包中的类，那么可能会增加编译时间，但不会影响程序运行的性能。
- Java运行平台由所需要的Java类库和虚拟机组成，这些类库被包含在C:\Program Files\Java\jdk1.8.0\_144\jre\lib\rt.jar中（注：不同版本的JDK路径名会有所不同），当程序执行时，Java运行平台从类库中加载程序真正使用的类字节码（bytecode）到内存中。

## 4.11 import语句

- 【例子】



```
import java.util.Date;
public class Example4_11
{
    public static void main(String args[])
    {
        Date date=new Date();
        System.out.printf("Local time: \n%s", date);
    }
}
```

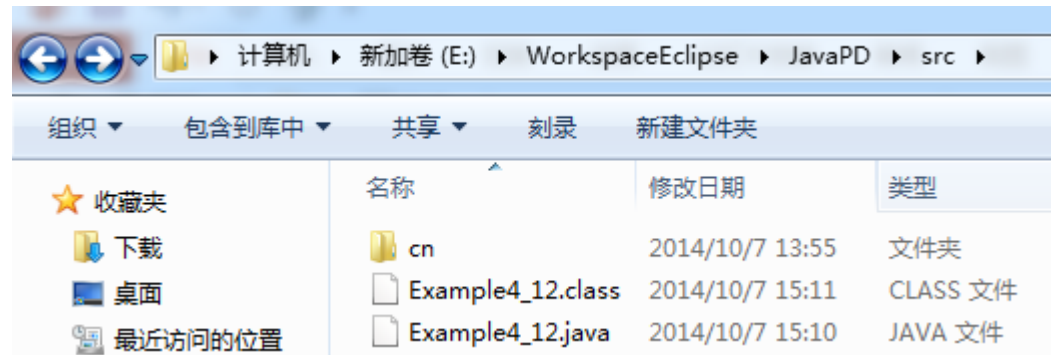
```
Local time:
Tue Oct 07 14:40:26 CST 2014
```

## 4.11 import语句

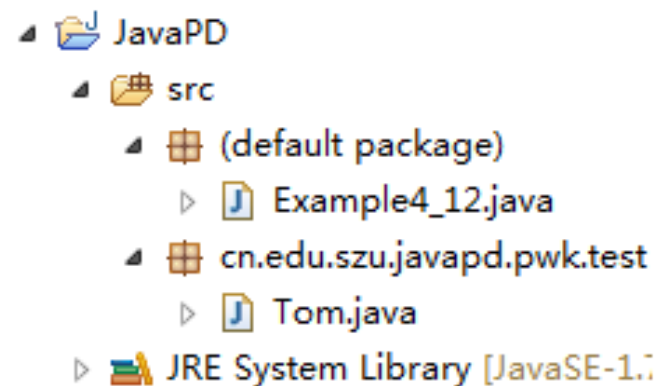
- 2.使用 **自定义包** 中的类
- 【例子】

```
package cn.edu.szu.javapd.pwk.test;  
public class Tom  
{  
    public void speak()  
    {  
        System.out.println("Hello");  
    }  
}
```

```
import cn.edu.szu.javapd.pwk.test.*;  
public class Example4_12  
{  
    public static void main(String args[])  
    {  
        Tom cat = new Tom();  
        cat.speak();  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
Microsoft Windows [版本 6.1.7601]  
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。  
C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src  
E:\WorkspaceEclipse\JavaPD\src>javac Example4_12.java  
E:\WorkspaceEclipse\JavaPD\src>java Example4_12  
Hello  
E:\WorkspaceEclipse\JavaPD\src>
```

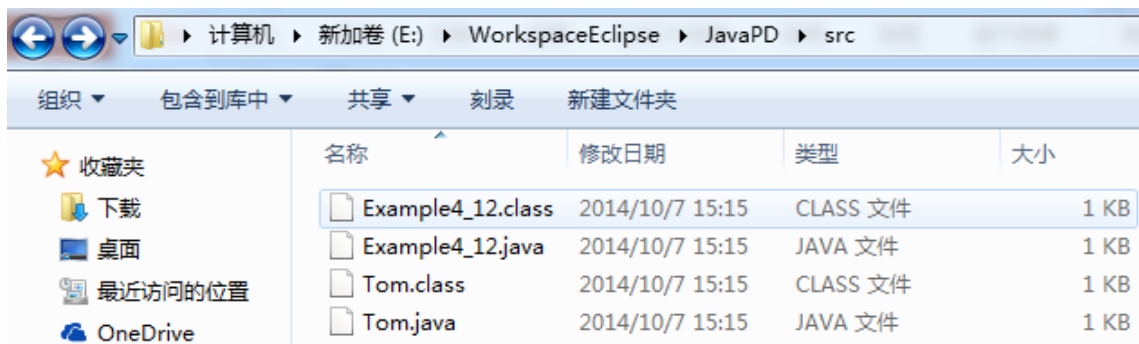


## 4.11 import语句

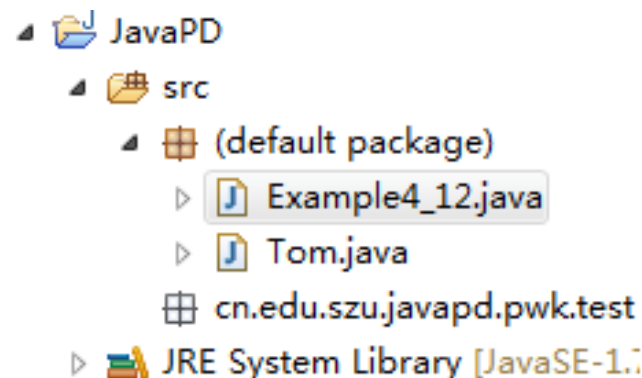
- 3.使用无名包中的类
- 【例子】

```
package cn.edu.szu.javapd.pwk.test;  
public class Tom  
{  
    public void speak()  
    {  
        System.out.println("Hello");  
    }  
}
```

```
import cn.edu.szu.javapd.pwk.test.*;  
public class Example4_12  
{  
    public static void main(String args[])  
    {  
        Tom cat = new Tom();  
        cat.speak();  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
Microsoft Windows [版本 6.1.7601]  
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。  
C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src  
E:\WorkspaceEclipse\JavaPD\src>javac Example4_12.java  
E:\WorkspaceEclipse\JavaPD\src>java Example4_12  
Hello  
E:\WorkspaceEclipse\JavaPD\src>
```



## 4.11 import语句

- 4.避免类名混淆
- Java运行环境总是先到程序所在的目录中寻找程序所使用的类，然后加载到内存
  - 如果在当前目录中找到了要加载的类，那么程序就不会再加载import语句引入的同名类
  - 如果在当前目录没有发现所需要的类，就到import语句所指的包中查找



# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.12 访问权限

- 类有两种重要的成员：成员变量和方法。
- 类创建的对象可以通过“.”运算符**访问**分配给自己的变量，也可以通过“.”运算符**调用**类中的**实例方法**和**静态方法**。
- 类在定义声明成员变量和方法时，可以用关键字private、protected和public来说明成员变量和方法的**访问权限（又称为可见性，visibility）**，使得对象访问自己的变量和使用方法受到一定的限制。
- 1.私有变量和私有方法
- 用关键字**private**修饰的成员变量和方法被称为**私有成员变量**和**私有方法**。
- 对于私有成员变量或私有方法，只有**在本类中创建该类的对象时**，这个对象才能访问自己的私有成员变量和类中的私有方法。

## 4.12 访问权限

- 【例子】

```
public class Employee
{
    private double salary=1800;
    public void setSalary(double salary)
    {
        this.salary=salary;
    }
    public double getSalary()
    {
        return salary;
    }

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

在本类中创建该类的对象

You can test a class by simply adding a main method in the same class.

```
        Employee zhang = new Employee();
        Employee wang = new Employee();
        zhang.setSalary(100);
        System.out.println("zhang's salary: "+zhang.getSalary());
        wang.salary=3888; // 合法
        System.out.println("wang's salary: "+wang.getSalary());
    }
}
```

## 4.12 访问权限

- 【例子】

```
class Employee
{
    private double salary=1800;
    public void setSalary(double salary)
    {
        this.salary=salary;
    }
    public double getSalary()
    {
        return salary;
    }
}
```

```
public class Example4_14
{
    public static void main(String args[])
    {
        Employee zhang = new Employee();
        Employee wang = new Employee();
        zhang.setSalary(100);
        System.out.println("zhang's salary: "+zhang.getSalary());
        wang.setSalary(3888);
        //wang.salary=88888; ERROR!!!
        System.out.println("wang's salary: "+wang.getSalary());
    }
}
```

## 4.12 访问权限

- 2.共有变量和共有方法
- 用**public**修饰的成员变量和方法被称为**共有成员变量**和**共有方法**。
- 当我们在任何一个类中用**类A**创建了一个对象a后，该对象a能访问自己的**public成员变量**和类中的**public方法**。

## 4.12 访问权限

- 3.友好变量和友好方法
- 不用**private, public, protected**修饰的成员变量和方法被称为友好成员变量和友好方法。
- 假如B与A是同一个包中的类，那么，下述B类中的a.weight, a.f(3,4)都是合法的

```
class B
{
    void g()
    {
        A a=new A();
        a.weight=23f; //合法
        a.f(3,4); //合法
    }
}
```

## 4.12 访问权限

- 4.受保护的成员变量和方法
- 用**protected**修饰的成员变量和方法被称为受保护的成员变量和受保护的方法（子类能访问）。

## 4.12 访问权限

- 5.public类与友好类
- 类声明时，如果关键字class前面加上public关键字，就称这样的类是一个public类。
- 不能用protected修饰类。
- 不能用private来修饰外部类，只能修饰内部类（这种情况也很少）。



## 4.12 访问权限

- 6.关于构造方法
- `private`, `public`, `protected`修饰符的意义也同样适合于构造方法（**constructor**）。
- 如果一个类没有明确地声明构造方法，那么**public**类的默认构造方法是**public**的，友好类的默认构造方法是友好的。
- 需要注意的是，如果一个**public**类定义声明的构造方法中没有**public**的构造方法，那么在另外一个类中使用该类创建对象时，使用的构造方法就不是 **public**的，**创建对象就受到一定的限制（例如，要求是否在同一个package中）**。
  - 更进一步，如果构造方法是**private**，则意味着**不允许用户创建对象**，例如**`java.lang.Math`**类的构造函数。注：`Math`类中的方法都是静态方法，因为需要通过类名来访问。

## 4.12 访问权限

Visibility increases  
→  
private, default (no modifier), protected, public

<i>Modifier on members in a class</i>	<i>Accessed from the same class</i>	<i>Accessed from the same package</i>	<i>Accessed from a subclass in a different package</i>	<i>Accessed from a different package</i>
public	✓	✓	✓	✓
protected	✓	✓	✓	—
default (no modifier)	✓	✓	—	—
private	✓	—	—	—

内部没有限制

默认在同一包里

protected为了继承

## 4.12 访问权限

- 记A的一个对象为a
- 在类A中，可以访问对象a的以下成员
  - **private**, friendly (or default), **protected**, **public**
- 在与类A同package的另外一个类B中，可以访问对象a的以下成员
  - Friendly (or default), **protected**, **public**
- 在类A的子类B中(不同package)，可以访问对象a的以下成员
  - **protected**, **public**
- 在与类A不同package的另外一个类C中，可以访问对象a的以下成员
  - **public**

## 4.12 访问权限

- **private**: make the members **private** if they are not intended for use from outside the class.
- **protected**: make the fields or methods **protected** if they are intended for the extenders of the class but not the users of the class.
- **public**: make the members **public** if they are intended for the users of the class (no limitation).

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- **4.13 对象的组合**
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.13 对象的组合


- 一个类可以把对象作为自己的成员变量，如果用这样的类创建对象，那么该对象中就会有其它对象，也就是说该对象将其他对象作为自己的组成部分（这就是人们常说的Has-A），或者说该对象是由几个对象组合而成。

## 4.13 对象的组合

- 【例子】

```
public class B
{
    private A a;
    B(A a)
    {
        this.a = a;
    }

    public void setAx(double x)
    {
        a.setX(x);
    }
}
```



```
public class A
{
    private double x;
    public void setX(double x)
    {
        this.x=x;
    }
    public double getX()
    {
        return x;
    }
}
```

```
public class MainClass
{
    public static void main(String args[])
    {
        A a = new A();
        a.setX(1);
        System.out.println(a.getX());

        B b = new B(a);
        b.setAx(2);
        System.out.println(a.getX());
    }
}
```

1.0  
2.0

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件



## 4.14 基本类型数据的类包装

- Java的基本数据类型
  - byte, short, int, long
  - float, double
  - char
- Java同时也提供了与基本类型数据相关的类，实现了对基本类型数据的封装。这些类在java.lang包中
  - Byte, Short, Integer, Long
  - Float, Double
  - Character

**java.lang.Number**

## 4.14 基本类型数据的类包装

- 【例子】

```
public class Example4_16
{
    public static void main(String args[])
    {
        char a[]={ 'a', 'b', 'c', 'D', 'E', 'F' };
        for(int i=0; i<a.length; i++)
        {
            if(Character.isLowerCase(a[i]))
                a[i]=Character.toUpperCase(a[i]);
            else if(Character.isUpperCase(a[i]))
                a[i]=Character.toLowerCase(a[i]);
        }
        for(int i=0; i<a.length; i++)
        {
            System.out.printf("%6c", a[i]);
        }
    }
}
```

A	B	C	d	e	f
---	---	---	---	---	---

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.15 对象数组

- 【例子】

```
public class Example4_19
{
    public static void main(String args[])
    {
        Integer m[] = new Integer[10];
        for(int i=0;i<10;i++)
        {
            m[i] = new Integer(101+i);
        }
        for(int i=0;i<10;i++)
        {
            System.out.println(m[i].intValue());
        }
    }
}
```

每个元素都是Integer类型的对象，目前还是空对象

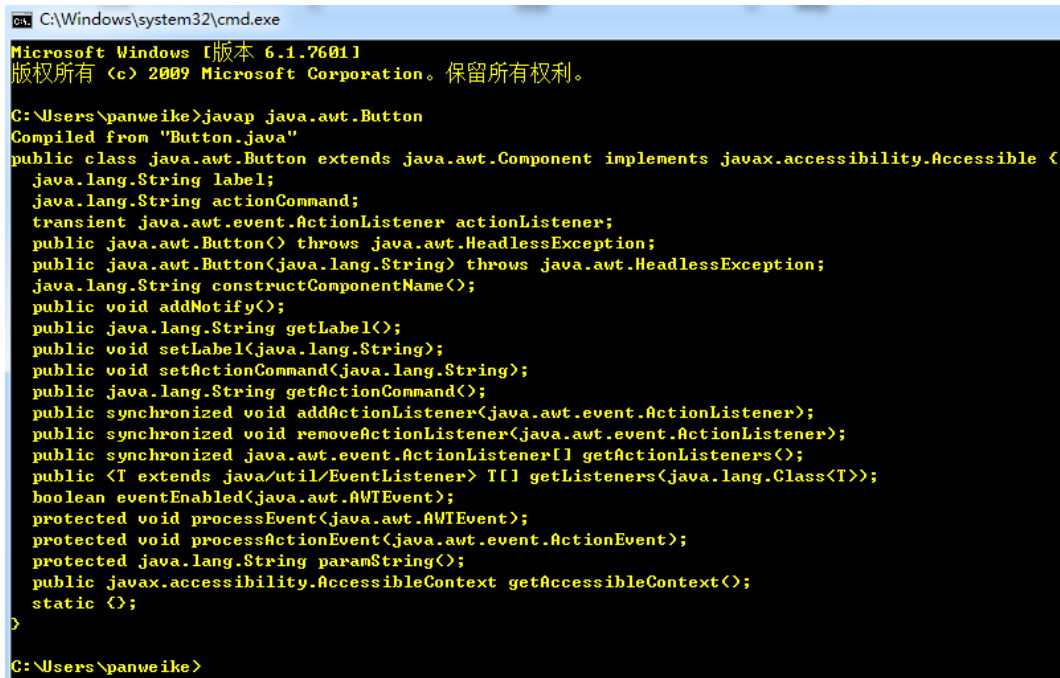
创建对象

# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.16 反编译和文档生成器

- 使用 **javap.exe** 可以将字节码反编译为源码，查看源码类中的方法的名字和成员变量的名字
- `javap java.awt.Button`



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation. 保留所有权利。

C:\Users\panweike>javap java.awt.Button
Compiled from "Button.java"
public class java.awt.Button extends java.awt.Component implements javax.accessibility.Accessible {
    java.lang.String label;
    java.lang.String actionCommand;
    transient java.awt.event.ActionListener actionListener;
    public java.awt.Button() throws java.awt.HeadlessException;
    public java.awt.Button(java.lang.String) throws java.awt.HeadlessException;
    java.lang.String constructComponentName();
    public void addNotify();
    public java.lang.String getLabel();
    public void setLabel(java.lang.String);
    public void setActionCommand(java.lang.String);
    public java.lang.String getActionCommand();
    public synchronized void addActionListener(java.awt.event.ActionListener);
    public synchronized void removeActionListener(java.awt.event.ActionListener);
    public synchronized java.awt.event.ActionListener[] getActionListeners();
    public <T extends java/util/EventListener> T[] getListeners(java.lang.Class<T>);
    boolean eventEnabled(java.awt.AWTEvent);
    protected void processEvent(java.awt.AWTEvent);
    protected void processActionEvent(java.awt.event.ActionEvent);
    protected java.lang.String paramString();
    public javax.accessibility.AccessibleContext getAccessibleContext();
    static {};
}
```

## 4.16 反编译和文档生成器

- javap -private java.awt.Button



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>javap -private java.awt.Button
Compiled from "Button.java"
public class java.awt.Button extends java.awt.Component implements javax.accessibility.Accessible <
    java.lang.String label;
    java.lang.String actionCommand;
    transient java.awt.event.ActionListener actionListener;
    private static final java.lang.String base;
    private static int nameCounter;
    private static final long serialVersionUID;
    private int buttonSerializedDataVersion;
    private static native void initIDs();
    public java.awt.Button() throws java.awt.HeadlessException;
    public java.awt.Button(java.lang.String) throws java.awt.HeadlessException;
    java.lang.String constructComponentName();
    public void addNotify();
    public java.lang.String getLabel();
    public void setLabel(java.lang.String);
    public void setActionCommand(java.lang.String);
    public java.lang.String getActionCommand();
    public synchronized void addActionListener(java.awt.event.ActionListener);
    public synchronized void removeActionListener(java.awt.event.ActionListener);
    public synchronized java.awt.event.ActionListener[] getActionListeners();
    public <T extends java.util.EventListener> T[] getListeners(java.lang.Class<T>);
    boolean eventEnabled(java.awt.AWTEvent);
    protected void processEvent(java.awt.AWTEvent);
    protected void processActionEvent(java.awt.event.ActionEvent);
    protected java.lang.String paramString();
    private void writeObject(java.io.ObjectOutputStream) throws java.io.IOException;
    private void readObject(java.io.ObjectInputStream) throws java.lang.ClassNotFoundException, java.io.IOException, java.awt.HeadlessException;
    public javax.accessibility.AccessibleContext getAccessibleContext();
    static <>
>
```

## 4.16 反编译和文档生成器

- 使用 **javadoc.exe** 可以制作源文件类结构的html格式文档

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src

E:\WorkspaceEclipse\JavaPD\src>javadoc *.java
正在加载源文件A.java...
正在加载源文件B.java...
正在加载源文件MainClass.java...
正在构造 Javadoc 信息...
标准 Doclet 版本 1.7.0_51
正在构建所有程序包和类的树...
正在生成 A.html...
正在生成 B.html...
正在生成 MainClass.html...
正在生成 package-frame.html...
正在生成 package-summary.html...
正在生成 package-tree.html...
正在生成 constant-values.html...
正在构建所有程序包和类的索引...
正在生成 overview-tree.html...
正在生成 index-all.html...
正在生成 deprecated-list.html...
正在构建所有类的索引...
正在生成 allclasses-frame.html...
正在生成 allclasses-noframe.html...
正在生成 index.html...
正在生成 help-doc.html...

E:\WorkspaceEclipse\JavaPD\src>
```

名称	修改日期	类型	大小
resources	2014/10/7 17:04	文件夹	
A.html	2014/10/7 17:06	Chrome HTML D...	8 KB
allclasses-frame.html	2014/10/7 17:06	Chrome HTML D...	1 KB
allclasses-noframe.html	2014/10/7 17:06	Chrome HTML D...	1 KB
B.html	2014/10/7 17:06	Chrome HTML D...	7 KB
constant-values.html	2014/10/7 17:06	Chrome HTML D...	4 KB
deprecated-list.html	2014/10/7 17:06	Chrome HTML D...	4 KB
help-doc.html	2014/10/7 17:06	Chrome HTML D...	7 KB
index.html	2014/10/7 17:06	Chrome HTML D...	3 KB
index-all.html	2014/10/7 17:06	Chrome HTML D...	6 KB
MainClass.html	2014/10/7 17:06	Chrome HTML D...	8 KB
overview-tree.html	2014/10/7 17:06	Chrome HTML D...	4 KB
package-frame.html	2014/10/7 17:06	Chrome HTML D...	1 KB
package-summary.html	2014/10/7 17:06	Chrome HTML D...	4 KB
package-tree.html	2014/10/7 17:06	Chrome HTML D...	4 KB
A.java	2014/10/7 16:28	JAVA 文件	1 KB
B.java	2014/10/7 16:35	JAVA 文件	1 KB
MainClass.java	2014/10/7 16:37	JAVA 文件	1 KB
stylesheet.css	2014/10/7 17:04	层叠样式表文档	12 KB
package-list	2014/10/7 17:06	文件	1 KB



## 4.16 反编译和文档生成器

The screenshot shows a web browser window with the address bar displaying `file:///E:/WorkspaceEclipse/JavaPD/src/index.html`. The browser's address bar includes navigation buttons (back, forward, refresh) and a list of bookmarks (Apps, Programming, 香港, Research, Faith, 深圳, 深大, MISC, People, 科研关注, 推荐). The browser's content area displays the Eclipse JavaDoc documentation for class `A`.

The documentation is organized into several sections:

- 所有类** (All Classes): A sidebar on the left lists classes `A`, `B`, and `MainClass`.
- 程序包** (Packages): A tabbed interface at the top shows `类` (Classes) as the active tab. Other tabs include `树` (Tree), `已过时` (Deprecated), `索引` (Index), and `帮助` (Help).
- 上一个类** (Previous Class) and **下一个类** (Next Class): Navigation links for navigating between classes.
- 框架** (Framework) and **无框架** (No Framework): Navigation links for navigating between frameworks.
- 根类: 嵌套 | 字段 | 构造器 | 方法** (Root Class: Nested | Fields | Constructors | Methods): Navigation links for navigating between different parts of the class.
- 详细资料: 字段 | 构造器 | 方法** (Detailed Information: Fields | Constructors | Methods): Navigation links for navigating between different parts of the class.

The main content area displays the documentation for class `A`:

- 类 A** (Class A): The class name.
- java.lang.Object**: The superclass.
- A**: The class name.
- public class A**: The class declaration.
- extends java.lang.Object**: The superclass.
- 构造器概要** (Constructor Summary): A section containing a table of constructors.
- 构造器** (Constructors): A table with one entry: `A()`.
- 方法概要** (Method Summary): A section containing a table of methods.
- 方法** (Methods): A table with two entries: `getX()` (returning `double`) and `setX(double x)` (returning `void`).
- 从类继承的方法 java.lang.Object** (Methods inherited from class java.lang.Object): A table listing methods inherited from `java.lang.Object`: `clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`.

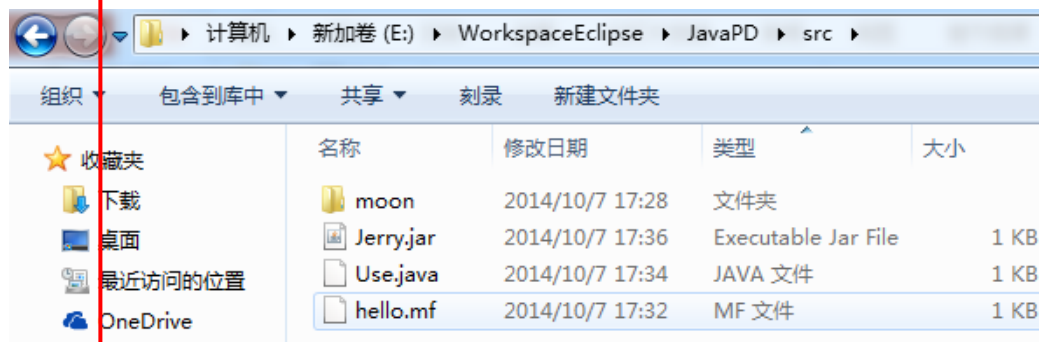
# Outline

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.17 jar文件

```
hello.mf
1 Manifest-Version: 1.0
2 Class: moon.star.TestOne moon.star.TestTwo
3 Created-By: 1.8
```

- Step 1: 编写TestOne.java, TestTwo.java
- Step 2: 编写hello.mf
- **Step 3: 生成jar文件**
- Step 4: 把jar文件移动到C:\Program Files\Java\jre1.8.0\_144\lib\ext\
- Step 5: 编写Use.java, 编译、运行



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\panweike>pushd E:\WorkspaceEclipse\JavaPD\src

E:\WorkspaceEclipse\JavaPD\src>jar cfm Jerry.jar hello.mf moon\star\TestOne.java moon\star\TestTwo.java

E:\WorkspaceEclipse\JavaPD\src>
```

## 4.17 jar文件

```
package moon.star;  
public class TestOne  
{  
    public void fTestOne()  
    {  
        System.out.println("I am a method in TestOne class");  
    }  
}
```

```
package moon.star;  
public class TestTwo  
{  
    public void fTestTwo()  
    {  
        System.out.println("I am a method in TestTwo class");  
    }  
}
```

```
import moon.star.*;  
public class Use  
{  
    public static void main(String args[])  
    {  
        TestOne a = new TestOne();  
        a.fTestOne();  
        TestTwo b = new TestTwo();  
        b.fTestTwo();  
    }  
}
```

```
I am a method in TestOne class  
I am a method in TestTwo class
```

# 小节

- 4.1 面向对象编程
- 4.2 类声明和类体
- 4.3 类体的构成
- 4.4 构造方法与对象的创建
- 4.5 对象的引用与实体
- 4.6 成员变量
- 4.7 方法
- 4.8 方法重载
- 4.9 关键字this
- 4.10 包
- 4.11 import语句
- 4.12 访问权限
- 4.13 对象的组合
- 4.14 基本类型数据的类包装
- 4.15 对象数组
- 4.16 反编译和文档生成器
- 4.17 jar文件

## 4.16 反编译和文档生成器

- 为何要学习**OOP**?
  - Object-oriented programming (OOP) enables you to develop **large-scale software** and **GUIs** effectively.
- 什么是面向过程?
  - The procedural paradigm focuses on **designing methods**.
- 什么是**OOP**? **OOP**有何优点?
  - The object-oriented paradigm **couples data and methods together into objects**. Software design using the object-oriented paradigm **focuses on objects and operations on objects**. The object-oriented approach combines the power of the procedural paradigm **with an added dimension that integrates data with operations into objects**.

# 问答题(1/2)



- 1. 请叙述在面向对象编程语言中，类和对象之间的关系。
- 2. 请写出三个合乎规范类名。
- 3. 请叙述构造方法和普通的方法之间的区别在哪里。
- 4. 请叙述类成员变量和对象的实例变量之间的区别在哪里。
- 5. 为什么修改一个对象的类成员变量，会影响其他由这个类创建的对象相应的类成员变量？
- 6. 请问如果在代码中试图为一个常量重新赋值，会出现什么错误？
- 7. 为什么类方法不允许访问一个对象的实例变量和其它的实例方法？
- 8. 请叙述在Java中“按值传递”基本数据类型参数和对象数据类型参数的区别在哪里？

## 问答题(2/2)

- 9. 请问如果通过对象数据类型的形参，在方法内部对形参所引用的实体进行修改。其改动在方法执行完毕后能保留下来吗？
- 10. 为什么`this`关键字不能出现在类方法中？
- 11. 请叙述`private`访问权限和`public`访问权限的区别。
- 12. 请问在内部类中，能定义静态成员变量吗？
- 13. 请叙述在Java中，包的命名惯例。
- 14. Tomcat是一款著名的Servlet容器和Web服务器。它的开发站点域名为tomcat.apache.org。按照JAVA包的命名惯例，存放tomcat源代码的包应该叫什么名字？
- 15. “import java.util.\*”和“import java.util.Scanner”有什么不同？