



Java程序设计

第4章 面向对象基础



第4章 面向对象基础

- 4.1 面向对象程序设计
- 4.2 类
- 4.3 构造方法与对象
- 4.4 类与程序的基本结构
- 4.5 参数传值
- 4.6 对象的组合
- 4.7 实例方法与类方法
- 4.8 this关键字
- 4.9 包
- 4.10 import语句
- 4.11 访问权限



4.2 类

Java语言与其他面向对象语言一样，引入了类和对象的概念，类是用来创建对象的模板，它包含被创建对象的属性和方法的定义。因此，要学习Java编程就必须学会怎样去编写类，即怎样用Java的语法去描述一类事物共有的属性和行为。



4.2.1 定义类

在Java语言中，类是基本的构成要素，是对象的模板，Java程序中所有的对象都是由类创建的。一个Java类主要包括以下两部分：

- 类的声明
- 类的主体

类的声明

在类声明中，需要定义类的名称、对该类的访问权限、该类与其他类的关系等。类声明的格式如下：

```
[修饰符] class <类名> [extends 父类名] [implements 接口列表]{ }
```

[修饰符]用于指定类的访问权限，可选值为public、abstract和final。

类名用于指定类的名称，类名必须是合法的Java标识符。一般情况下，要求首字母大写。

[extends 父类]名用于指定要继承参数。

[implements 接口列表]用于指定该类实现的所有接口。



类体

类声明部分大括号中的内容为类体。类体主要由以下两部分构成：

- (1) 成员变量的定义；
- (2) 成员方法的定义。

在程序设计过程中，编写一个能完全描述客观事物的类是不现实的。

比如，构建一个Apple类，该类可以拥有很多很多的属性（即成员变量），在定义该类时，选取程序需要的必要属性和行为就可以了。





4.2.2 成员变量和局部变量

在类体中所声明的变量称为类的成员变量，而在方法体中声明的变量和方法的参数则称为局部变量。

现在我们来来看一下，如何声明成员变量和局部变量，以及变量的有效范围。



声明成员变量

Java用成员变量来表示类的状态和属性，声明成员变量的基本语法格式如下：

```
[修饰符] [static] [final] <变量类型> <变量名>;
```

修饰符：可选参数，用于指定变量的被访问权限，可选值为public、protected和private。

static：可选，用于指定该成员变量为静态变量，可以直接通过类名访问。如果省略该关键字，则表示该成员变量为实例变量。

final：可选，用于指定该成员变量为取值不会改变的常量。

变量类型：必选：用于指定变量的数据类型，其值可以为Java中的任何一种数据类型。

变量名：必选，用于指定成员变量的名称，变量名必须是合法的Java标识符。



声明成员变量

例如，在类中声明3个成员变量。

```
public class Apple {  
    public String color; //声明公共变量color  
    public static int count; //声明静态变量count  
    public final boolean MATURE=true; //声明常量MATURE并赋值  
    public static void main(String[] args) {  
        System.out.println(Apple.count);  
        Apple apple=new Apple();  
        System.out.println(apple.color);  
        System.out.println(apple.MATURE);  
    }  
}
```

声明局部变量

定义局部变量的基本语法格式同定义成员变量类似，所不同的不能使用权限修是不和static关键字对局部变量进行修饰，但可以使用final关键字：

```
[final] <变量类型> <变量名>;
```

final: 可选，用于指定该局部变量为常量。

变量类型: 必选，用于指定变量的数据类型，其值可以为Java中的任何一种数据类型。

变量名: 必选，用于指定局部变量的名称，变量名必须是合法的Java标识符。





声明局部变量

例如，在grow()成员方法中声明两个局部变量。

```
public void grow() {  
    final boolean STATE; //声明常量STATE  
    int age; //声明局部变量age  
}
```



变量的有效范围

变量的有效范围是指该变量在程序代码中的作用区域，在该区域外不能直接访问变量。有效范围决定了变量的生命周期，变量的生命周期是指从声明一个变量并分配内存空间、使用变量，然后释放该变量并清除所占内存空间的一个过程。进行变量声明的位置，决定了变量的有效范围，根据有效范围的不同，可将变量分为以下两种。

(1) 成员变量：在类中声明，在整个类中有效。

(2) 局部变量：在方法内或方法内的复合代码块（“{”与“}”之间的代码）中声明的变量。在复合代码块声明的变量，只在当前复合代码块中有效；在复合代码块外方法内声明的变量在整个方法内都有效。





变量的有效范围

例如下面的实例:

```
public class Olympics {  
    private int medal-All=800; //成员变量  
    public void China() {  
        int medal-CN=100; //方法的局部变量  
        if(medal-CN<1000) { //代码块  
            int gold=50; //代码块的局部变量  
            medal-CN+=50; //允许访问  
            medal-All-=150; //允许访问  
        }  
    }  
}
```

4.2.3 了解成员方法

Java中类的行为由类的成员方法来实现。类的成员方法由方法声明和方法体两部分组成。其一般格式如下：

```
[修饰符] <方法返回值的类型> <方法名>([参数列表]) {  
    [方法体]  
}
```

[修饰符]用于指定方法的访问权限，可选值为public、protected和private。

方法返回值的类型用于指定该方法的返回值的类型，如果该方法没有返回值，必须使用关键字void进行标识。方法返回值的类型可以是任何Java数据类型。



了解成员方法

方法名用于指定成员方法的名称，方法名必须是合法的Java标识符。

[参数列表]用于指定方法中所需的参数。当存在多个参数时，各参数之间应使用逗号分隔。方法的参数可以是任何Java数据类型。

方法体是方法的实现部分，在方法体中可以完成指定的工作，可以只打印一句话，也可以省略方法体，使方法什么都不做。需要注意的是：当省略方法体时，其外面的大括号一定不能省略。

4.2.4 注意事项

- 上面说过，类体是由成员变量和成员方法组成。而对成员变量的操作只能放在方法中，方法使用各种语句对成员变量和方法体中声明的局部变量进行操作，声明成员变量是可以赋初值。

```
public class A {  
    int a = 12; // 声明变量的时候同时赋予初始值  
}
```

但是不能这样：

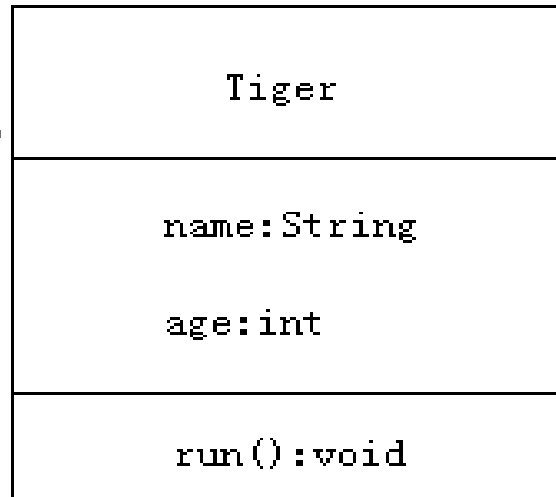
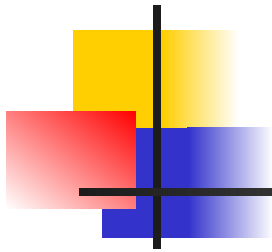
```
public class A {  
    int a ;  
    a = 12;    // 这样是非法的，此操作只能出现在方法体中  
}
```



4.2.5 类的UML图

- UML (Unified Modeling Language Diagram, UML) , 它是一个结构图, 用来描述一个系统的静态结构。一个UML中通常包含类 (class) 的UML图, 接口 (Interface) 的UML图以及泛化关系 (Generalization) 的UML图、关联关系 (Association) 的UML图、依赖关系 (Dependency) 的UML图和实现关系 (Realization) 的UML图。





Tiger类的UML图

- 第一层是名字层，如果类的名字是常规字形，表明该类是具体类，如果类的名字是斜体字形，表明该类是抽象类（后续会讲到抽象类）。
- 第二层是变量层，也称属性层，列出类的成员变量及类型。格式是“变量名：类型”。
- 第三层是方法层，列出类中的方法。格式是“方法名字：类型”。

4.3 构造方法与对象

- 构造方法用于对对象中的所有成员变量进行初始化。对象的属性通过变量来刻画，也就是类的成员变量，而对象的行为通过方法来体现，也就是类的成员方法。方法可以操作属性形成一定的算法来实现一个具体的功能。类把属性和方法封装成一个整体。

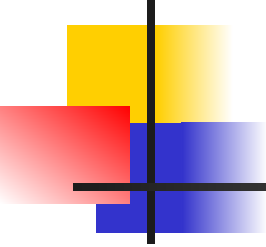


4.3.1 构造方法的概念及用途

构造方法是一种特殊的方法，它的名字必须与它所在类的名字完全相同，并且没有返回值，也不需要使用关键字void进行标识。

```
public class Apple {  
    public Apple() { // 构造方法  
    }  
}
```

构造方法用于对对象中的所有成员变量进行初始化，在创建对象时立即被调用。

- 
- 1. 默认构造方法和自定义构造方法
 - 如果类例定义了一个或多个构造方法，那么Java中不提供默认的构造方法。
 - 【例4-2】 定义Apple类，在该类的构造方法中初始化成员变量。



■ 2. 构造方法没有类型

表4-1 Java变量的初始值

类 型	初 值
byte	0
short	0
int	0
float	0.0F
long	0L
double	0.0D
char	'\u0000'
boolean	false
引用类型	null





4.3.2 对象的概述

- 在面向对象语言中，对象是对类的一个具体描述，是一个客观存在的实体。万物皆对象，也就是说任何事物都可看做对象，如一个人、一个动物，或者没有生命体的轮船、汽车、飞机，甚至概念性的抽象，如公司业绩等等。
- 一个对象在Java语言中的生命周期包括创建、使用和销毁3个阶段。

4.3.3 对象的创建

- 1. 对象的声明
- 声明对象的一般格式如下：

类名 对象名；

- 类名：必选，用于指定一个已经定义的类。
- 对象名：必选，用于指定对象名称，对象名必须是合法的Java标识符。
- 声明Apple类的一个对象redApple的代码如下：
- `Apple redApple;`



2. 实例化对象

- 在声明对象时，只是在内存中为其建立一个引用，并置初值为null，表示不指向任何内存空间。
- 声明对象以后，需要为对象分配内存，这个过程也称为实例化对象。在Java中使用关键字new来实例化对象，具体语法格式如下：

■ 对象名=new 构造方法名([参数列表])；

- 对象名：必选，用于指定已经声明的对象名。
- 类名：必选，用于指定构造方法名，即类名，因为构造方法与类名相同。
- 参数列表：可选参数，用于指定构造方法的入口参数。如果构造方法无参数，则可以省略。
- 在声明Apple类的一个对象redApple后，可以通过以下代码为对象redApple分配内存（即创建该对象）：

```
redApple=new Apple();
```

由于Apple类的构造方法无入口参数，所以省略了参数列表
- 在声明对象时，也可以直接实例化该对象：

```
Apple redApple=new Apple();
```



4.3.4 对象的使用

创建对象后，就可以访问对象的成员变量，并改变成员变量的值了，而且还可以调用对象的成员方法。通过使用运算符“.”实现对成员变量的访问和成员方法的调用。

语法格式为：

对象.成员变量

对象.成员方法()



4.3.5 对象的销毁

在许多程序设计语言中，需要手动释放对象所占用的内存，但是，在Java中则不需要手动完成这项工作。Java提供的垃圾回收机制可以自动判断对象是否还在使用，并能够自动销毁不再使用的对象，收回对象所占用的资源。

Java提供了一个名为`finalize()`的方法，用于在对象被垃圾回收机制销毁之前执行一些资源回收工作，由垃圾回收系统调用，可以重写该方法。但是垃圾回收系统的运行是不可预测的。`finalize()`方法没有任何参数和返回值，每个类有且只有一个`finalize()`方法。

4.4 类与程序的基本结构

- 一个Java应用程序是由若干个类组成，这些类可以在一个源文件中，也可以分布在若干个源文件中，如图所示。

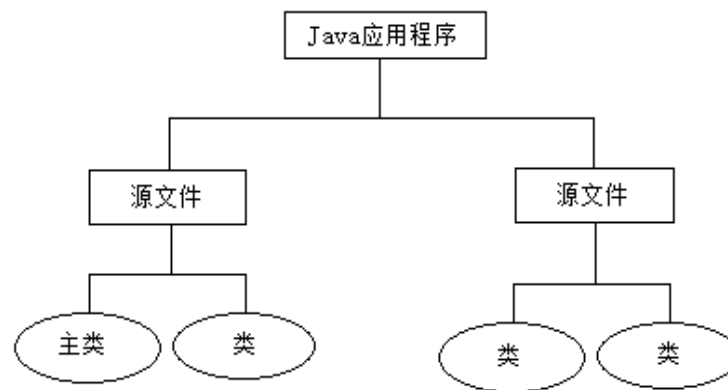
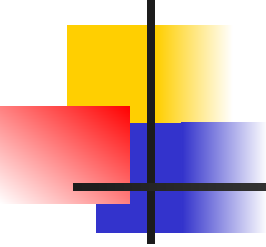


图4-8 Java应用程序结构

- 
- 在Java应用程序中有一个主类，即含有main方法的类，main方法是程序执行的入口，也就是说想要执行一个Java应用程序必须从main方法开始执行。在编写一个Java应用程序时，可以编写若干个Java源文件，每个源文件编译后产生若干个类的字节码文件。
 - 当解释器运行一个Java应用程序时，Java虚拟机将Java应用程序的字节码文件加载到内存中，然后再由Java的虚拟机解释执行。
 - Java程序以类为“基本单位”，从编译的角度看，每个源文件都是一个独立编译单位，当程序需要修改某个类时，只需要重新编译该类所在的源文件即可，不必重新编译其他类所在的源文件，这样非常有利于系统的维护。从软件设计角度看，Java语言中的类是可复用的，编写具有一定功能的可复用代码在软件设计中非常重要。





4.5 参数传值

- 在Java程序中，如果声明方法时包含了形参声明，则调用方法时必须给这些形参指定参数值，调用方法时实际传递给形参的参数值被称为实参。

4.5.1 传值机制

- Java方法中的参数传递方式只有一种，也就是值传递。所谓的值传递，就是将实际参数的副本传递到方法内，而参数本身不受任何影响。例如，去银行开户需要身份证原件和复印件，原件和复印件上的内容完全相同，当复印件上的内容改变的时候，原件上的内容不会受到影响。也就是说，方法中参数变量的值是调用者指定值的拷贝。

4.5.2 基本数据类型参数的传值

- 对于基本数据类型参数，向该参数传递值的级别不能高于该参数的级别，比如，不能想int型参数传递一个float值，但可以向double型参数传递一个float值。
- 【例】 在Point类中定义一个add方法，然后在Example类的主方法中创建Point类的对象，然后调用该对象的add(int x,int y)方法，当调用add方法的时候，必须向add方法中传递两个参数。



4.5.3 引用类型参数的传值

- 当参数是引用类型时，传递的值是变量中存放的“引用”，而不是变量所引用的实体。当两个相同类型的引用型变量，如果具有同样的引用，就会用同样的实体，因此，如果该表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；但是，改变参数中存放的“引用”不会影响向其传值的变量中存放的“引用”

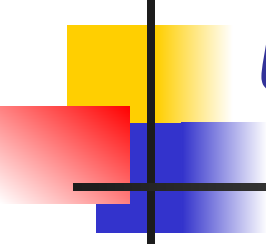
【例4-5】 Car类为汽车类，负责创建一个汽车类的对象，fuelTank类是一个油箱类负责创建油箱的对象。Car类创建的对象调用run(fuelTank ft)方法时需要将fuelTank类创建的油箱对象ft传递给run(fuelTank ft)，该方法消耗汽油，油箱中的油也会减少。



4.6 对象的组合

- 如果一个类把某个对象作为自己的一个成员变量，使用这样的类创建对象后，该对象中就会有其他对象，也就是该类对象将其他对象作为自己的一部分。





4.6.1 组合与复用

- 如果一个对象a组合了另一个对象b，那么对象a就可以委托对象b调用其方法，即对象a以组合的方式复用对象b的方法。

4.6.2 类的关联关系和依赖关系的UML图

■ 1. 关联关系

- 如果A类中成员变量是用B类声明的对象，那么A和B的关联是关联关系，称A类的对象关联于B类的对象或A类的对象组合了B类的对象。如果A关联于B，那么UML图通过一条实线连接A和B的UML图，实线的起始端是A的UML图，终点端是B的UML图，但终点端使用一个指向B的UML图的方向箭头表示实线的结束。

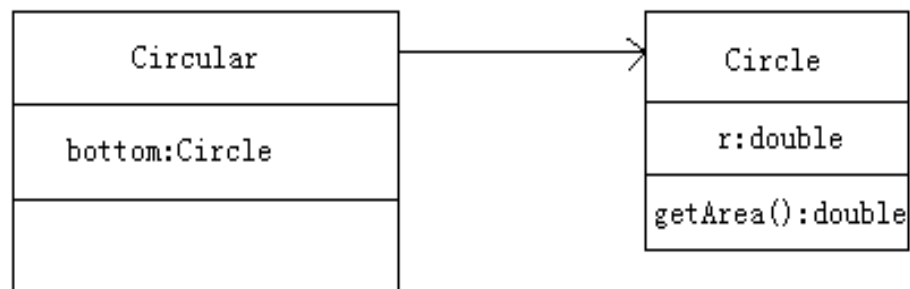


图4-9 关联关系UML图

2. 依赖关系

- 如果A类中某个方法的参数是用B类声明的对象或某个方法返回的数据类型是B类对象，那么A和B的关系是依赖关系，称A依赖于B。如果A依赖于B，那么UML通过使用一个虚线连A和B的UML图，虚线的起始端是A的UML图，终点端是B的UML图，但终点端使用一个指向B的UML图的方向箭头表示虚线的结束。

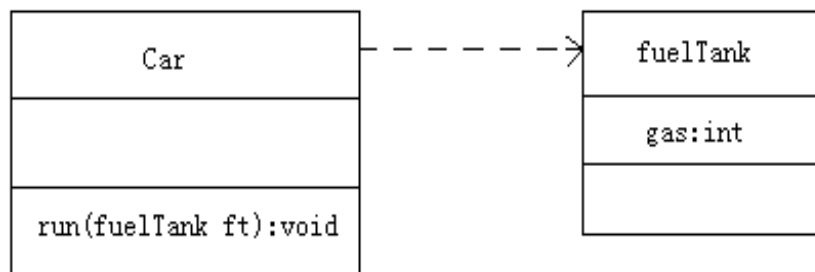


图4-10 依赖关系的UML图



4.7 实例方法与类方法

- 4.7.1 实例方法与类方法的定义
 - 声明方法时，方法类型前面不使用static修饰的是实例方法，使用static修饰的是类方法也称作静态方法。

4.7.2 实例方法和类方法的区别

- 1. 对象调用实例方法
 - 当字节码文件被分配到内存时，实例方法不会被分配入口地址，只有当该类创建对象后，类中的实例方法才会分配入口地址，这时实例方法才可被类创建的对象调用。
- 2. 使用类名调用类方法
 - 类中定义的方法，在该类被加载到内存时，就分配了相应的入口地址，这样类方法不仅可以被类创建的任何对象调用执行，也可以直接通过类名调用。类方法的入口地址直到程序退出时才被取消。但是需要注意，类方法不能直接操作实例变量，因为在类创建对象之前，实例成员变量还没有分配内存。实例方法只能使用对象调用，不能通过类名调用。



4.8 this 关键字

- this 关键字表示某个对象，this 关键字可以出现在实例方法和构造方法中，但不可以出现在类方法中。当局部变量和成员变量的名字相同时，成员变量就会被隐藏，这时如果想在成员方法中使用成员变量，则必须使用关键字 this。
- 语法格式为：

```
this.成员变量名  
this.成员方法名()
```

【例4-7】 创建一个类文件，该类中定义了 setName()，并将方法的参数值赋予类中的成员变量。





4.9 包

Java要求文件名和类名相同，所以如果将多个类放在一起时，很可能出现文件名冲突的情况，这时Java提供了一种解决该问题的方法，那就是使用包将类进行分组。下面将对Java中的包进行详细介绍。





4.9.1 包的概念

包 (package) 是Java提供的一种区别类的命名空间的机制，是类的组织方式，是一组相关类和接口的集合，它提供了访问权限和命名的管理机制。Java中提供的包主要有以下3种用途。

- (1) 将功能相近的类放在同一个包中，可以方便查找与使用。
- (2) 由于在不同包中可以存在同名类，所以使用包在一定程度上可以避免命名冲突。
- (3) 在Java中，某此访问权限是以包为单位的。



4.9.2 创建包

创建包可以通过在类或接口的源文件中使用 package 语句实现，package 语句的语法格式如下：

```
package 包名;
```

包名：必选，用于指定包的名称，包的名称必须为合法的Java标识符。当包中还有子包时，可以使用“包1.包2...包n”进行指定，其中，包1为最外层的包，而包n则为最内层的包。



4.9.3 使用包中的类

类可以访问其所在包中的所有类，还可以使用其他包中的所有public类。访问其他包中的public类可以有以下两种方法。

(1) 使用长名引用包中的类

使用长名引用包中的类比较简单，只需要在每个类名前面加上完整的包名即可。例如，创建Round类（保存在com.lzw包中）的对象并实例化该对象的代码如下：

```
com.lzw.Round round=new com.lzw.Round();
```

4.9.3 使用包中的类

(2) 使用import语句引入包中的类

由于采用使用长名引用包中的类的方法比较繁琐，所以Java提供了import语句来引入包中的类。import语句的基本语法格式如下：

```
import 包名1[.包名2.……].类名 [ * ];
```

当存在多个包名时，各个包名之间使用“.”分隔，同时包名与类名之间也使用“.”分隔。*：表示包中所有的类。

```
import com.lzw.Round;
```

例如，引入com.lzw包中的Round类的代码如下：

```
import com.lzw.*;
```

可以引入该包下的全部类：



4.10 import语句

- import关键字用于加载已定义好的类或包，导入支持类可供本类调用方法和属性。

4.10.1 类的两种访问方法

- 类可以访问其所在包中的所有类，还可以使用其他包中的所有public类。访问其他包中的public类可以有以下两种方法。
- (1) 使用长名引用包中的类。
- 使用长名引用包中的类比较简单，只需要在每个类名前面加上完整的包名即可。例如，创建Round类（保存在com.lzw包中）的对象并实例化该对象的代码如下：
`com.lzw.Round round=new com.lzw.Round();`
- (2) 使用import语句引入包中的类。
- 由于采用使用长名引用包中的类的方法比较烦琐，所以Java提供了import语句来引入包中的类。下面我们着重介绍使用import导入类。

4.10.2 引入类库中的类

- import语句的基本语法格式如下:
- `import 包名1[, 包名2.].类名|*;`
- 当存在多个包名时, 各个包名之间使用“.”分隔, 同时包名与类名之间也使用“.”分隔。
- *: 表示包中所有的类。
- 一个Java源程序中可以有多个import语句, 它们必须写在package语句和源文件中类的定义之间。下面列举部分Java类库中的包:
- `java.lang` 包含所有的基本语言类
- `javax.swing` 包含抽象窗口工具几种的图形、文本、窗口GUI类
- `java.io` 包含所有的输入输出类
- `java.util` 包含实用类
- `java.sql` 包含操作数据库的类
- 例如引入util包中的全部类:
- `import java.util.*;`
- 如果想要引入包中具体的类:
- `import java.util.Date;`
- 引入util包中的Date类。



4.11 访问权限

- 访问权限使用访问修饰符进行限制，访问修饰符有private、protected、public，它们都是Java中的关键字。

4.11.1 什么是访问权限

- 访问权限是指对象是否能够通过“.”运算符操作自己的变量或通过“.”运算符调用类中的方法。
- 在编写类的时候，类中的实例方法总是可以操作该类中的实例变量和类变量；类方法总是可以操作该类中的类变量，与访问限制符没有关系。

4.11.2 私有变量和私有方法

- 使用private修饰的成员变量和方法称为私有变量和私有方法。例如：
 - `public class A {`
 - `private int a; // 变量a 是私有的变量`
 - `private int sum (int m,int n) { // 方法sum是私有方法`
 - `return m - n;`
 - `}`
 - `}`
- 假如现在有个B类，在B类中创建一个A类的对象后，该对象不能访问自己的私有变量和方法。例如：
 - `public class B {`
 - `public static void main (String [] args) {`
 - `A ca = new A ();`
 - `ca.a = 18; // 编译错误，访问不到私有的变量a`
 - `}`
 - `}`

4.11.3 公有变量和公有方法

- 使用public修饰的变量和方法称为公有变量和公有方法。例如：
 - `public class A {`
 - `public int a; // 变量a 是公有的变量`
 - `public int sum (int m,int n) { // 方法sum是公有方法`
 - `return m - n;`
 - `}`
 - `}`
- 使用public访问修饰符修饰的变量和方法在任何一个类中创建对象后都会访问到。例如：
 - `public class B {`
 - `public static void main (String [] args) {`
 - `A ca = new A ();`
 - `ca.a = 18; // 可以访问, 编译通过`
 - `}`
 - `}`

4.11.4 友好变量和友好方法

- 不使用private、public、protected修饰符修饰的成员变量和方法被称为友好变量和友好方法，如：
 - `public class A {`
 - `int a; // 变量a 是友好的变量`
 - `int sum (int m,int n) { // 方法sum是友好方法`
 - `return m - n;`
 - `}`
 - `}`
- 同一包中的两个类，如果在一个类中创建了另外一个类的对象后，该对象能访问自己的友好变量和友好方法。
 - `public class B {`
 - `public static void main (String [] args) {`
 - `A ca = new A ();`
 - `ca.a = 18; // 可以访问，编译通过`
 - `}`
 - `}`

4.11.5 受保护的成员变量和方法

- 用protected访问修饰符修饰的成员变量和方法成为受保护的成员变量和受保护的方法，如：
 - `public class A {`
 - `protected int a; // 变量a 是受保护的变量`
 - `protected int sum (int m,int n) { // 方法sum是受保护的方法`
 - `return m - n;`
 - `}`
 - `}`
- 同一个包中的两个类，一个类在另一个类创建对象后可以通过该对象访问自己的protected变量和protected方法。
 - `public class B {`
 - `public static void main (String [] args) {`
 - `A ca = new A ();`
 - `ca.a = 18; // 可以访问，编译通过`
 - `}`
 - `}`



4.11.6 public类与友好类

- 在声明类的时候，如果在关键字class前面加上public关键字，那么这样的类就是公有的类。例如：
 - `public class A {`
 - `... ..`
 - `}`
- 可以在任何另外一个类中，使用public类创建对象。如果一个类不加public修饰，例如：
 - `class A {`
 - `... ..`
 - `}`
- 声明这个没有被public修饰的类就称为友好类，那么另外一个类中使用友好类创建对象时，必须保证它们是在同一个包中。

