



主讲教师 张 智
计算机学院软件工程系
课程群: 421694618

5 访问控制和内部类

5.1 Java包

5.2 访问控制

5.3 内部类

5.1 Java包 (package)

- 随着程序架构越来越大，类的个数也越来越多，解决类的命名冲突是一件很麻烦的事情。
- 有时，开发人员还可能需要将处理同一方面的问题的类放在同一个目录下，以便于管理。



Java 引入包(package)机制，提供了类的多层命名空间，用于解决类的命名冲突、类文件管理以及控制访问等问题



包允许在更广泛的范围内保护类、数据和方法。可以在包内定义类，而在包外的代码不能访问该类，这使得类之间有隐私性

包的定义：

```
package 包名[.子包名]... ;
```

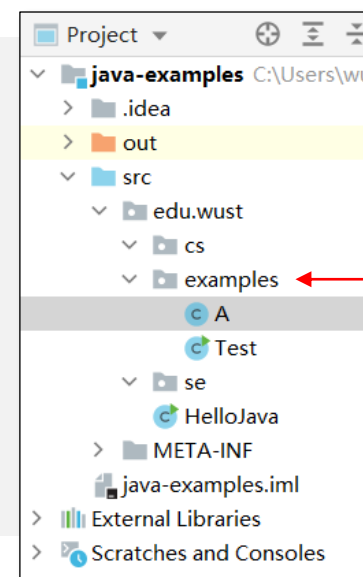


Java 包的命名规则：

- 包名全部由小写字母（多个单词也全部小写）
 - 如果包名包含多个层次，每个层次用 "." 分割
 - 包名一般由倒置的域名开头，比如 com.baidu，不要有 www
 - 自定义包不能用 java 开头（java.xxx.xx这种） ←—— Java安全机制
- 默认包：如果源文件没有定义包，那么类将会被放进一个无名的包中，称为默认包 (在实际开发中，通常不会把类定义在默认包下)

示例：

```
package edu.wust.examples;  
  
public class A {  
    public void f1() {  
        System.out.println("f1 called");  
    }  
}
```



创建edu.wust.examples包

- package命令必须放在源文件的最前面
- 一个源文件最多只能有一条package命令
- 如果没有package，类将会保存在当前默认包中（不推荐）
- 一条package命令对源文件中的所有类起作用
- 编译后的.class文件也按照包结构存放

导入包：

`import 包名[.子包名...].类名 ;`

如果是导入全部类就用 "*" 代替

- `import` 向某个Java文件中导入指定包层次下的某个类或全部类（不包含子包的类）
- `import` 语句位于 `package` 语句之后，类定义之前
- 一个Java源文件可以包含多个 `import` 语句
- 同一个包中类的无需 `import`

如果不使用 `import` 导入包，那么使用不同包中的其它类时，需要使用该类的全名(包名+类名)：

```
edu.wust.examples.A a = new edu.wust.examples.A(); //比较繁琐
```

其中，`edu.wust.examples`是包名，`A`是包中的类名，`a`是类的对象

注：Java默认已导入 `java.lang` 包下所有类，该包是java语言的核心，提供了很多基础类：如String、包装类、Math、Object、Class、Thread、Exception等类

了解：静态导入

如果是导入全部静态成员变量、方法就用 "*" 代替

```
import static 包名[.子包名...].类.静态成员(变量或方法);
```

```
import static java.lang.System.*; ← 导入System类中的全部静态成员变量和方法
```

```
import static java.lang.Math.*; ← 导入Math类中的全部静态成员变量和方法
```

```
public class Test {
```

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

```
        // out是java.lang.System类的静态成员变量，代表标准输出
```

```
        // PI是java.lang.Math类的静态成员变量，表示π常量
```

```
        out.println( PI ); ←
```

```
        // 直接调用Math类的sqrt静态方法
```

```
        out.println( sqrt(256) ); ←
```

```
    }
```

```
}
```

静态导入后代码中的类名都可省略了

部分系统包

包	说明
java.lang	Java的核心类库，包含运行 Java 程序必不可少的基础类，如String、包装类、Math、Object、Class、Thread线程、Exception异常处理等类，系统默认加载这个包
java.io	Java语言的标准输入/输出类库，如基本输入/输出流、文件输入/输出等
java.util	包含实用程序类，如Scanner、Date、Calender、Random、数据结构(如List/Map等集合类)等。
java.awt	构建图形用户界面(GUI)的类库，低级绘图操作Graphics类、图形界面组件和布局，以及用户界面交互控制和事件响应(如Event类)
java.net	包含执行与网络相关操作的类，如Socket、ServerSocket、URL类等
java.lang.reflect	提供用于反射对象的工具
java.sql	实现JDBC的类库
java.text	处理显示对象格式化，如SimpleDateFormat类等

[【返回】](#)

5.2 访问控制

- 信息隐藏是OOP最重要的功能之一。在编写程序时，有些数据可能不希望被用户调用，需要控制这些数据的访问
- 访问控制：就是限定类、属性或方法是否可以被程序里的其他部分访问



- 类的访问控制：只能是public或者default (不写出来，也称friendly)
- 属性和方法的访问控制有 4 个： private、default(不写)、protected 和 public

类的访问控制

- Java类常用访问控制有2种：public 和 default (不写出来，也称friendly)
 - **public类**：可以在**任何一个包中的任何一个类**中被访问和继承
 - **default类**：**只能在同一个包**中被其它类所访问和继承

成员的访问控制

包括属性和方法

作用域按大小排序：private < default < protected < public

	Private成员	默认的成员	Protected成员	Public成员
同一类中可见	是	是	是	是
同一个包中对子类可见	否	是	是	是
同一个包中对非子类可见	否	是	是	是
不同包中对子类可见	否	否	是	是
不同的包中对非子类可见	否	否	否	是

同一包

不同包

补充一张图

访问权限	类	包	子类	其他包	描述
public	✓	✓	✓	✓	对任何人都是可用的
protected	✓	✓	✓	✗	继承的类可以访问以及和 default 一样的权限
default	✓	✓	✗	✗	包访问权限，即在整个包内均可被访问
private	✓	✗	✗	✗	除类型创建者和类型的内部方法之外的任何人都不能访问的元素

访问控制说明

修饰符	说明
private	private成员，只能被该类自身的方法访问和修改，而不能被任何其他类(包括子类)访问和引用。因此，private 修饰符具有最高的保护级别
default	如果一个类没有访问控制符，说明它具有默认的控制访问特性。这种默认的控制访问规定，该类只能被同一个包中的类访问和引用，而不能被其他包中的类使用，即使其他包中有该类的子类
protected	protected成员可以被三种类所访问：该类自身、与它在同一个包中的其他类以及在其他包中的该类的子类。使用protected的主要作用，是允许其他包中它的子类来访问父类的特定属性和方法，否则可以使用默认访问控制
public	public成员具有被其任何类访问的可能性 (import引入public类)

访问控制测试：

■ 情况1：类在同一个包中

```
package edu.wust.cs;  ←-----类在同一个包中-----> package edu.wust.cs;
public class A {
    public void f1() {
        System.out.println("f1 called");
    }
    protected void f2() {
        System.out.println("f2 called");
    }
    void f3() {          ↙ 默认是default
        System.out.println("f3 called");
    }
    private void f4() {
        System.out.println("f4 called");
    }
}

public class Test {
    public static void main(String args[]) {
        A a = new A(); // public类可以被其他类访问
        a.f1();        // OK, 类对象可以访问public成员
        a.f2();        // OK, protected成员也可以访问
        a.f3();        // OK, default成员可以访问
        a.f4();        // 报错: private成员不能访问
    }
}
```

A类修改为default类（同一包）

```
package edu.wust.cs;  
public class A { 去掉public  
    public void f1() {  
        System.out.println("f1 called");  
    }  
    protected void f2() {  
        System.out.println("f2 called");  
    }  
    void f3() {  
        System.out.println("f3 called");  
    }  
    private void f4() {  
        System.out.println("f4 called");  
    }  
}
```

```
package edu.wust.cs;  
public class Test {  
    public static void main(String args[]) { 同一个包不用 import  
        A a = new A(); // 同一包中default类也可访问  
        a.f1(); // OK, 类对象可以访问public成员  
        a.f2(); // OK, protected成员也可以访问  
        a.f3(); // OK, default成员可以访问  
        a.f4(); // 报错: private成员不能访问  
    }  
}
```

对象依旧可以
访问非私有成员

■ 情况2：类在不同一个包中

```
package edu.wust.cs;  ←-----类在不同包中-----→ package edu.wust.se;
public class A {
    public void f1() {
        System.out.println("f1 called");
    }
    protected void f2() {
        System.out.println("f2 called");
    }
    void f3() {          ↙ 默认是default
        System.out.println("f3 called");
    }
    private void f4() {
        System.out.println("f4 called");
    }
}

import edu.wust.cs.A;  ←-----需要 import 其他包中的类-----

public class Test {
    public static void main(String args[]) {
        A a = new A(); // public类可以被其他类访问
        a.f1();        // OK, 类对象可以访问public成员
        a.f2();        // 报错, 不能访问protected成员
        a.f3();        // 报错, 不能访问default成员
        a.f4();        // 报错: private成员不能访问
    }
}
```


A类修改为default类（不同包）

```
package edu.wust.cs;  
public class A { 去掉public  
    public void f1() {  
        System.out.println("f1 called");  
    }  
    protected void f2() {  
        System.out.println("f2 called");  
    }  
    void f3() {  
        System.out.println("f3 called");  
    }  
    private void f4() {  
        System.out.println("f4 called");  
    }  
}
```

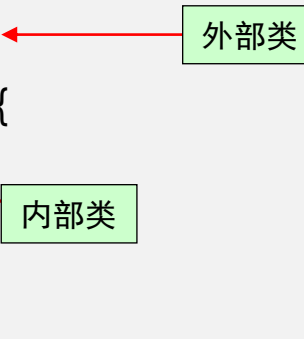
```
package edu.wust.se;  
import edu.wust.cs.A; 报错：default类不能被其他包访问  
  
public class Test {  
    public static void main(String args[]) {  
        A a = new A(); // 报错，A类无法导入使用  
    }  
}
```

[【返回】](#)

5.3 内部类

- 在类的内部可以定义另一个类。
- 如果在Outer类的内部再定义一个Inner类，此时Inner类就称为内部类(或嵌套类)，而类Outer则称为外部类(或宿主类)
- 如果有多层嵌套，最外层的类称为顶层类。

```
class Outer {  
    class Inner {  
        ...  
    }  
}
```

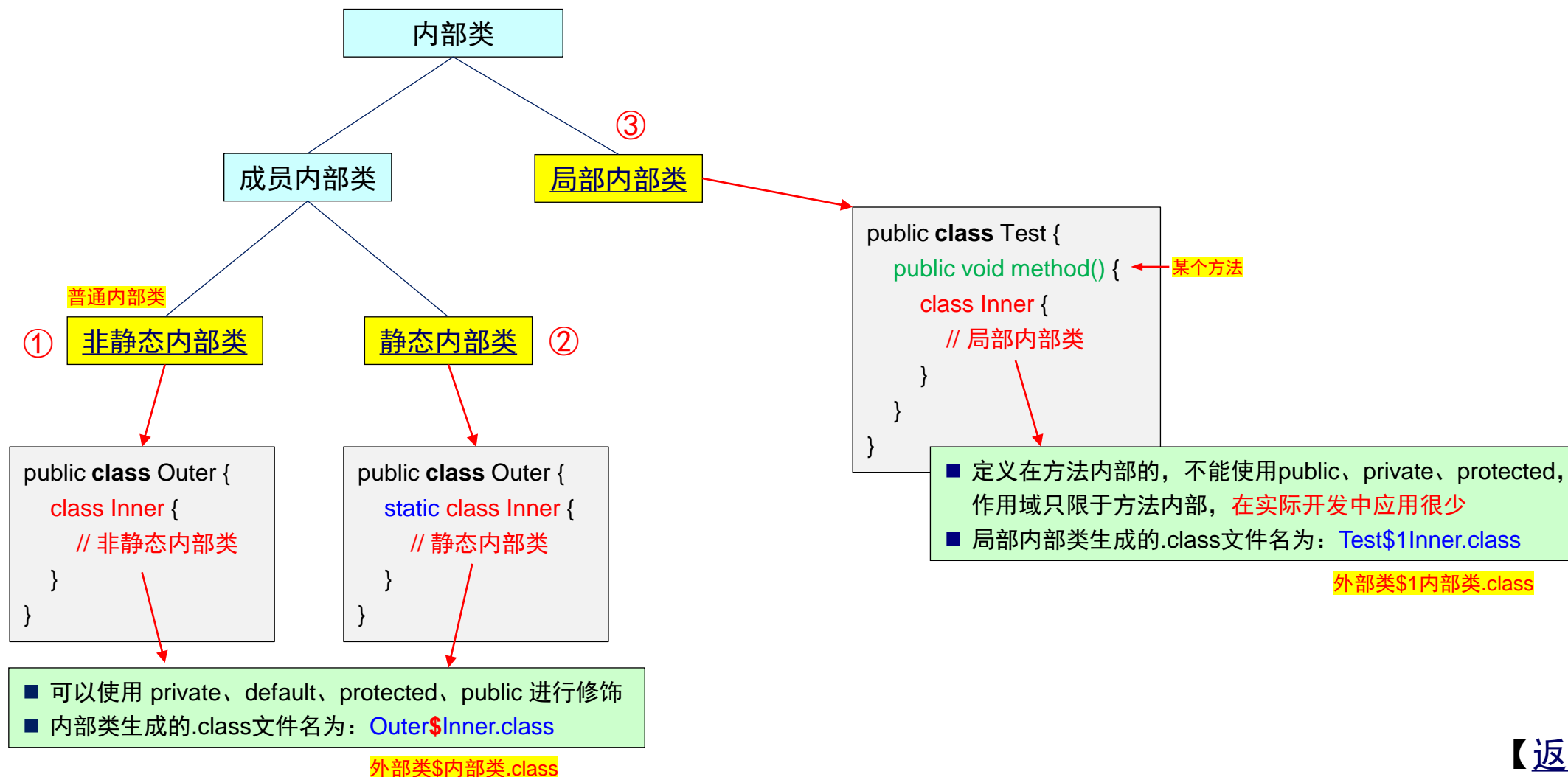


外部类

内部类

注：内部类与外部类不能重名

内部类分类

[【返回】](#)

1. 非静态内部类

- 在外部类的静态方法和外部类以外的其他类中，必须通过外部类的实例创建内部类的实例 ★
- 非静态内部类不能有静态方法、静态属性和静态代码块 ★ (以上两条见示例1)
- 非静态内部类可以直接访问外部类的成员，但是外部类不能直接访问非静态内部类成员 ★ (示例2、3)

Outer.java

```

public class Outer {
    class Inner {
        // static int a;
    }
    public void method1() {
        Inner i = new Inner();
    }
    public static void method2() {
        Outer.Inner i = new Outer().new Inner();
    }
    class Inner2 {
        Inner i = new Inner();
    }
}

class OtherClass {
    Outer.Inner i = new Outer().new Inner();
}

```

普通内部类(非静态内部类)

注: 普通的内部类不能有static成员 → 原因: 非static的内部类, 在外部类加载时, JVM并不会自动加载它, 定义static成员就无意义

// 实例方法中创建内部类对象: 直接new (不需要创建外部类实例)

// 静态方法中创建内部类对象: 需要创建外部类实例(new Outer())

此处可以省略, 但建议写全

另一个普通内部类

// 内部类中创建其他内部类对象: 直接new (不需要创建外部类实例)

其他类

正确用法: Outer.Inner i = new Outer().new Inner();
错误用法: Outer.Inner inner = new Outer.Inner();

// 其他类: 需要创建外部类实例(new Outer())

必须加上外部类

示例2 -- 在非静态内部类中，可以访问外部类的所有成员

注：本例两个类都在同一包下

```
package edu.wust.examples;

public class Outer { ← 外部类
    public int a = 100;
    static int b = 100;
    final int c = 100;
    private int d = 100;

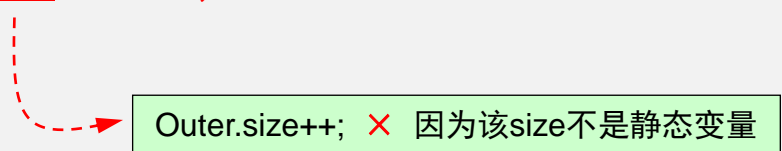
    class Inner { ← 非静态内部类能访问外部类的所有成员
        int a2 = a + 1; // 访问public的a
        int b2 = b + 1; // 访问static的b
        int c2 = c + 1; // 访问final的c
        int d2 = d + 1; // 访问private的d
    }
}
```

```
package edu.wust.examples;

public class Test {
    public static void main(String args[]) {
        Outer.Inner i = new Outer().new Inner(); // 创建内部类实例
        System.out.println( i.a2 ); // 输出101
        System.out.println( i.b2 ); // 输出101
        System.out.println( i.c2 ); // 输出101
        System.out.println( i.d2 ); // 输出101
    }
}
```

示例3 -- 非静态内部类同名变量问题

```
public class Outer {  
    private int size;  
    public class Inner { //内部类  
        private int size;  
        public void doStuff( int size ) {  
            size++;           // doStuff()方法自己的形参size  
            this.size++;       // 内部类自己的成员变量  
            Outer.this.size++; // 外部类成员变量  
        }  
    }  
}
```



Outer.size++; × 因为该size不是静态变量

内部类练习题：输出30， 20， 10， 请填空

```
class Outer {  
    private int num = 10;  
    class Inner {  
        public int num = 20;  
        public void show() {  
            int num = 30;  
            System.out.println(【1】);    // 答案: num  
            System.out.println(【2】);    // 答案: this.num  
            System.out.println(【3】);    // 答案: Outer.this.num  
        }  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        Outer.Inner oi = new Outer().new Inner();  
        oi.show();  
    }  
}
```

[【返回】](#)

2. 静态内部类

- 通常一个普通类不允许声明为静态的，只有一个内部类才可以
- 在创建静态内部类的实例时，不需要创建外部类的实例 ★ (示例1)
- 静态内部类中可以定义静态成员和实例成员 (示例2)
- 静态内部类可以直接访问外部类的静态成员，如果要访问外部类的实例成员，则需要通过外部类的实例去访问 (示例3)

静态内部类示例1 -- 不需外部类对象创建

```
public class Outer {  
    static class Inner {  
        //静态内部类  
    }  
}
```

```
class OtherClass {  
    Outer.Inner i = new Outer.Inner();  
}
```

静态内部类的实例化

静态内部类相当于把它放在外面
不再是嵌套的内部类，变成了顶层类

示例2：静态内部类中可以定义静态成员和实例成员

```
public class Outer {  
    static class Inner {  
        int a = 0;          // 实例变量a  
        static int b = 0;   // 静态变量b ← 补充：static内部类，在外部类加载时，JVM会自动加载它  
    }  
}
```

```
class OtherClass {  
    Outer.Inner i = new Outer.Inner();  
    int a2 = i.a;           // 访问实例成员  
    int b2 = Outer.Inner.b; // 通过类名访问静态成员  
}
```

↑
访问内部类静态成员

示例3：访问外部类的成员

```
public class Outer {  
    int a = 0;          // 实例变量a  
    static int b = 0;   // 静态变量 b  
  
    static class Inner {  
        Outer o = new Outer(); // 外部类实例  
        int a2 = o.a;   // 访问实例变量  
        int b2 = b;     // 访问静态变量  
    }  
}
```

访问外部类实例成员，需要外部类实例

可直接访问外部类的静态成员

[【返回】](#)

3. 局部内部类

- 局部内部类只在当前方法中有效
- 局部内部类不能使用public、private 和 protected修饰
- 局部内部类不能使用 static 修饰符
- 在局部内部类中可以访问外部类的所有成员
- 在局部内部类中只可以访问当前方法中 final 类型的参数与变量

```
public class Outer {  
    private int a = 5;  
    public Object foo(int localVar) { //在方法内部定义局部内部类  
        final int b = 6; //方法中的局部变量为final才能被内部类访问  
        class Inner{ // 内部类  
            public String toString() {  
                return ("Inner a=" + a + " b=" + b );  
            }  
        }  
        return new Inner();  
    }  
    public static void main(String[] args){  
        Outer outer = new Outer();  
        Object obj = outer.foo(66);  
        System.out.println("The object is " + obj);  
    }  
}
```

final量，不能修改

运行结果

The object is Inner a=5 b=6

注意局部内部类生成的.class名：Outer\$1Inner.class

\$1含义：在外部类方法中定义的一个局部内部类如果在外部类另一个方法中定义了另一个同名的局部内部类，则编号为\$2

【完】