Java6: 包、访问控制和接口

授课教师：邱元杰　电子邮箱：yuanjiq@126.com，微信电话: 13679081552

第6章 包、访问控制和接口

package：包语句

import：引入语句

成员访问控制

接口创建与使用

package-包语句

- 包：包实际上是一组类组成的集合，也称之为类库。
- 包的层次结构与文件系统的文件目录结构是相似的。包名是Java的合法标识符，一般都用小写的字母单词表示。
- Java语言提供了一些常用的基本类包，如java.io和java.lang。

包声明

- package语句作为Java源文件的第一条语句，指明该文件中定义的类所在的包，若缺省该语句，则指定为无名包。
- package pkgName1[.pkgName2[.pkgName3...]];
- 其中：pkgName1～pkgNameN表示包的目录层次。它对应于文件系统的目录结构。

包语句

- Java语言的JDK提供的包有：
- java.applet java.awt java.awt.image
- java.awt.peer java.io java.lang
- java.net java.util sun.tools.debug

- 在编译时，在javac上带-d选项。
- 例如：
  - javac -d . MyJavaPrg.java
  - 生成的MyJavaPrg.class存在user.java.sample包中。

包的作用

- 对类进行管理
  - 不同包里有相同的类不会发生冲突
  - 相同功能的类放在同一个包里
- 规定了类的使用范围
  - 同一个包里的类可以相互访问，不同包里的类不能直接相互访问。

第6章 包、访问控制和接口

package：包语句

import：引入语句

成员访问控制

接口创建与使用

Import 类引入语句

- 类引入语句
  - 引入语句提供了能使用Java中API或用户已创建的类。引入语句是在包语句(如果有的话)之后的任何条语句。
- import pack1[.pack2...].<className|*>;
  - pack1～packN为包的层次结构，它对应着要访问的类所在文件目录结构;
  - className则指明所要引入的类，如果要从一个包中引入多个类时，则可以用星号(*)来表示。
  - 使用“*”引入语句，只表示了源程序中所需要的类会在包中找到并引入，但是对包中其它的类或它下面的包中的类并不引入。

类引入语句

- 引入语句有两种形式：
  - 直接指明所要引入的类。

    import src.Point;
  - 使用“*”引入语句，指明类会在包。

    import src.*;
- 如果没有引入语句，而直接使用类时则必须显示其包。

  src.Point p = new src.Point(10,20);

第6章 包、访问控制和接口

package：包语句

import：引入语句

成员访问控制

接口创建与使用

访问控制

- 类的成员变量和方法都可以有自己的访问控制修饰符，来表示其访问控制的权限。
- 访问权限修饰符用于标明类、变量、方法的可访问程度。
- 在类中封装了数据和代码，在包中封装了类和其它的包。

访问控制

- Java中提供了对类的成员访问的四个范围:
  - 同一类中;
  - 同一包中;
  - 不同的包中的子类;
  - 不同包中的非子类。

- 四种访问权限修饰符:
  - public
  - protected
  - 缺省
  - private

访问控制

➢访问控制权限表

	类中	包中	子类	不同包中非子类
public	√	√	√	√
protected	√	√	√	
缺省	√	√		
private	√			

其中：“√”表示可访问，否则为不可访问。

访问控制

- public：可访问性最大修饰符，由public修饰的成员，则可以被任何范围中所访问。
- protected：允许类中、子类(包括在或不在同一包中)和它所在包中的类所访问。
- 缺省：可以被类自身和同一个包中的类访问。
- private：限制最强的修饰符。私有成员只能在它自身的类中访问。
- 可以最大限度地保持好类中敏感变量和方法，避免对象对这些类的成员访问时带来危害。

访问控制示例

```
package pack1;
class Original{
        private   int nPrivate   = 1;
        int nDefault   = 2;
        protected int nProtected = 3;
        public    int nPublic    = 4;
        public void access() {
          System.out.println("** 在类中, 可以访问的成员: **");
          System.out.println("Private member = "+nPrivate);
          System.out.println("Default member = "+nDefault);
          System.out.println("Protected member = "+nProtected);
          System.out.println("Public member = "+nPublic);
        }
}
```

访问控制示例

```
//在同一个包pack1中，也有继承关系的子类Derived。
package pack1;
class Derived extends Original{
        void access(){
          System.out.println("** 在子类中, 可以访问的成员: **");
// System.out.println("Private member = "+nPrivate);
          System.out.println("Default member = "+nDefault);
          System.out.println("Protected member = "+nProtected);
          System.out.println("Public member = "+nPublic);
        }
}
```

访问控制示例

```
//在同一个包pack1中，类SamePackage。
package pack1;
class SamePackage{
        void access(){
          Original o = new Original();
          System.out.println("**在同包中,其对象可访问的成员:**");
//  System.out.println("Private member = "+o.nPrivate);
          System.out.println("Default member = "+o.nDefault);
          System.out.println("Protected member="+o.nProtected);
          System.out.println("Public member = "+o.nPublic);
        }
}
```

访问控制示例

```
//在同一个包pack1中，类AccessControl1。
package pack1;
public class AccessControl1{
        public static void main(String[] args){
          Original o = new Original();
          o.access();
          Derived d = new Derived();
          d.access();
          SamePackage s = new SamePackage();
          s.access();
        }
      }
```

访问控制示例

```
//在不同包pack2中，也有继承关系的子类Derived。
package pack2;
import pack1.Original;
class Derived extends Original{
        void access(){
          System.out.println("**在不同包的子类中,可访问的成员:**");
//  System.out.println("Private member = "+nPrivate);
//  System.out.println("Default member = "+nDefault);
          System.out.println("Protected member = "+nProtected);
          System.out.println("Public member = "+nPublic);
        }
}
```

访问控制示例

```
//在不同包pack2中，类AnotherPackage。
package pack2;
import pack1.Original;
class AnotherPackage{
        void access(){
          Original o = new Original();
          System.out.println("**在不同包的类中,可访问的成员::**");
//  System.out.println("Private member = "+o.nPrivate);
//  System.out.println("Default member = "+o.nDefault);
//  System.out.println("Protected member="+o.nProtected);
          System.out.println("Public member = "+o.nPublic);
        }
}
```

访问控制示例

```
//在不同包pack2中，类AccessControl2。
package pack2;
import pack1.Original;
public class AccessControl2{
        public static void main(String[] args){
          Derived d = new Derived();
          d.access();
          AnotherPackage s = new AnotherPackage();
          s.access();
        }
      }
```

第6章 包、访问控制和接口

package：包语句

import：引入语句

成员访问控制

接口创建与使用

接口

- Java是通过接口使得处于不同类层次，甚至互不相关的类可以具有相同的行为。
- 接口是方法定义（没有实现）和常数变量的集合。
- 用接口，你可以指定一个类必须做什么，而不是规定它如何去做。
- 在类层次的任何地方都可以使用接口定义一个行为的协议实现它。

接口

- Java接口主要用于：
  - 通过接口可以指明多个类需要实现的方法。
  - 通过接口可以了解对象的交互界面，而不需要了解对象所对应的类。
  - 通过接口可以实现不相关类的相同行为，而不需要考虑这些类之间的层次关系。

接口的定义

- 接口的定义格式与类相似，具有成员变量和成员方法。但是接口中的所有方法都是abstract方法，这些方法是没有语句。

```
interfaceDeclaration{
  interfaceBody
}
```

其中：　interfaceDeclaration为接口声明部分；

interfaceBody为接口体部分。

接口的定义

- 接口体中包括常量定义和方法定义。

  type constantName = value;

  returnType methodName([paramList]);

- 其中：
  - type constantName=Value；语句为常量定义部分。在接口中定义的成员变量都是常量，具有pubilc，final和static属性，在创建这些变量时可以省略这些修饰符。
  - returnType methodName([parameterList])；为方法定义部分。接口中方法是抽象方法，只有方法声明，而无方法实现，所以它的方法定义是没有方法体，由；直接结束。接口中声明的方法具有public，abstract属性。

接口定义示例

```
interface Collection {
        int MAX_NUM=100;
        void add (Object objAdd);
        void delete (Object objDelet);
        Object find (Object objFind);
        int currentCount();
}
```

接口的实现

- 接口的方法必须由(非抽象的)类非抽象实现。
  - 可以由抽象类抽象的实现
- 在类的声明中，如果用implements子句就可以声明这个类对接口的实现。
- 关键字implements不同于extends，它表示类对接口的实现而不是继承，并且一个类可以实现多个接口。
- 类实现接口，则必须实现接口中的所有方法。

接口实现示例

```
class FIFOQueue implements Collection {
        public void add (Object objAdd){
          //add object code
        }
        public void delete (Object objDelet){
          //delete object code
        }
        public Object find (Object objFind){
          //find object code
        }
        public int currentCount(){
          //count object code
        }
}
```

接口的类型

- 接口可以作为一个引用类型来使用。任何实现该接口的类的实例都可以存储在该接口类型的变量中，通过这些变量可以访问类所实现接口中的方法。
- 在程序运行时，Java动态地确定需要使用那个类中的方法。

- 超类变量可以引用子类对象
- 接口变量可以引用子类对象

接口类型示例

```
{
  public static void main(String args[]){
        Collection cVar = new FIFOQueue();
        Object objAdd = new Object();
        …
        cVar.add(objAdd);
        …
  }
}
```

接口可以扩展

- 接口可以通过运用关键字extends被其他接口继承。语法与继承类是一样的。
- 当一个类实现一个继承了另一个接口的接口时，它必须实现接口继承链表中定义的所有方法。
- 接口中的变量和方法的被隐藏和覆盖：
  - 如果在子接口中定义了和超接口同名的常量或相同的方法，则超接口中的常量被隐藏，方法被覆盖。

IFExtend.java

接口的多重继承

接口A：Java代码

```
public interface InterfaceA {
    int len = 1 ;
    void output();
}
```

接口B：Java代码

```
public interface InterfaceB {
    int len = 2 ;
    void output();
}
```

接口Sub继承接口A和接口B：Java代码

```
public interface InterfaceSub extends InterfaceA, interfaceB { }
```

接口的多重继承

```
public class Xyz implements InterfaceSub {
    public void output() {  System.out.println(   “output in class Xyz.”   );  }
    public void outputLen( int type) {
        switch (type) {
            case InterfaceA.len:    System.out.println( "len of InterfaceA=." +type);
break ;
            case InterfaceB.len:  System.out.println(   “len of InterfaceB=.”   +type);
break ;  }
      }
      public static void main(String[] args) {
          Xyz xyz= new Xyz ();
          xyz .output();
          xyz .outputLen(1);
       }
}
```

接口中的变量

- 可以使用接口来引入多个类的共享常量，这样做只需要简单的声明包含变量初始化想要的值的接口就可以了。
- 如果一个类中包含那个接口（就是说当实现了接口时），所有的这些变量名都将作为常量看待。

常数分组

- 由于置入一个接口的所有字段都自动具有static 和final属性，所以接口是对常数值进行分组的一个好工具，它具有与C 或C++的enum 非常相似的效果

```
public interface Months {
    int JANUARY = 1, FEBRUARY = 2, MARCH = 3,
        APRIL = 4, MAY = 5, JUNE = 6, JULY = 7,
        AUGUST = 8, SEPTEMBER = 9, OCTOBER = 10,
        NOVEMBER = 11, DECEMBER = 12;
}
```

接口

- 接口与抽象类的比较：
  - 接口中的方法都是由public、abstract修饰的抽象方法，而抽象类中则即可以有抽象方法，也可以含有非抽象方法；
  - 接口中的变量都是由public、final和static修饰的常量，而抽象类中即可以有一般的成员变量，也可以自己声明的常量；
  - 接口可以用extends关键字实现多重继承，而抽象类继承性是类的单一继承，同时也可以实现接口；

接口

- 接口与抽象类的比较：
  - 接口实现的类由关键字implements声明，而抽象类的子类由关键字extends声明；
  - 实现接口的类必须实现接口中的所有方法，而抽象类的子类(非抽象类)只必须实现抽象类中的全部的抽象方法；
  - 接口中的变量(即常量)可以用接口名直接访问，而抽象类的变量则不完全可以用类名直接访问；
  - 接口不是类分级结构的一部分。而没有联系的类可以执行相同的接口。
  - 抽象类可以实现接口

内部类

- Java支持在一个类中声明另一个类，这样的类称作内部类，而包含内部类的类成为内部类的外嵌类。
- 内部类可以调用外嵌类的成员变量和方法
- 内部类不可以声明类变量和类方法

匿名类

- 和类有关的匿名类
  - Java允许我们直接使用一个类的子类的类体创建一个子类对象。
  - 创建子类对象时，除了使用父类的构造方法外还有类体，此类体被认为是一个子类去掉类声明后的类体，称作匿名类。
  - 假设People是类，那么下列代码就是用People的一个子类（匿名类）创建对象：

```
new People () {
    匿名类的类体
};
```

AnonymousClass1.java

匿名类

- 和接口有关的匿名类
  - 假设Computable是一个接口，那么，Java允许直接用接口名和一个类体创建一个匿名对象，此类体被认为是实现了Computable接口的类去掉类声明后的类体，称作匿名类。
  - 下列代码就是用实现了Computable接口的类（匿名类）创建对象：

```
new Computable() {
    实现接口的匿名类的类体
}
```

AnonymousClass2.java

完整的Java源文件

- Java源程序包括有：
  - 最多可以有一条package语句，并且放在除注解外的第一条语句的位置上；
  - 可以有任意条import语句，并处在package语句之后(如果有)；
  - 可以定义任意个类，如果没有接口时则至少有一个类的定义；
  - 可以定义任意个接口，如果没有类时则至少有一个接口的定义；

完整的Java源文件

- Java源程序包括有：
  - 如果在Applicatoin编程中，则把包括有main()方法的类声明为public；
  - 在一个源程序中，只能有一个类可以被声明为public；
  - 用public声明的类名作为源程序的文件名(注意大小写一致)且以.java作为后缀；
  - 如果源程序中只有接口定义，则用接口名作源文件名；
  - 在一个源程序中定义的所有类和接口，在成功编译后都将生成一个对应的字节码文件，这些字节码文件的名是类名或接口名，并以.class为扩展名。

思考

- Java语言中的包是什么含义，它如何使用?
- package语句和import语句的作用是什么？举例说明。
- Java源程序的组成中至少要有一个什么定义？举例说明。
- Java语言中对成员的访问有几种情况？举例说明。
- Java语言中有几种对成员访问控制的修饰符，它们是如何使用的?
- 接口与抽象类的主要区别在哪些方面?

课堂作业

- 设计一个接口Shape，包括2个抽象方法getPerimeter()和getArea()，分别是计算形状的周长和面积。设计实现该接口的具体类Rectangle、Triangle和Circle，分别表示矩形、三角形和圆，在三个子类中建立各自的构造方法并重写getPerimeter()和getArea()。在main()中声明Shape变量s，利用s（分别引用Rectangle、Triangle和Circle对象）输出某矩形、三角形和圆的周长和面积。