面向对象程序设计Java

江春华

电子科技大学信息与软件工程学院

内　容

第6章　包、访问控制和接口

1. package：包语句
2. import：引入语句
3. 成员访问控制
4. 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。
- 格式：

```
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**

例如：

```
package users.java.sample;
```

表示类在**users.java.sample**包中。

- 在编译时，在javac上带-d选项。

例如：

```
javac -d . MyJavaPrg.java
```

生成的**MyJavaPrg.class**存在**users.java.sample**包中。

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。

格式为：**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

- 如果没有引入语句，而直接使用类时则必须显示其包。

  例：

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

引入语句有两种形式：

- 直接指明所要引入的类。

  例：

```
import src.Point;
```

- 使用“*”引入语句，指明类会在包。

  例：

```
import src.*;
```

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

```
/**
   源程序: Original.java
*/
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。
/**
   源程序：Derived.java
*/
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。
/**
   源程序：SampePackage.java
*/
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。
/**
    源程序: AccessControl1.j
*/
package pack1;
public class AccessControl
    public static void mai
      Original o = new Or
      o.access();
      Derived d = new Deri
      d.access();
      SamePackage s = new SamePackage();
      s.access();
    }
  }
```

```
** 在类中，可以访问的成员： **
Private member = 1
Default member = 2
```

```
**在子类中，可以访问的成员:**
```

```
**在同包中,其对象可访问的成员:**
Default member = 2
Protected member = 3
Public member = 4
```

访问控制示例

```
//在不同包pack2中，也有继承关系的子类Derived。
/**
   源程序：Derived.java
*/
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。
/**
   源程序：AnotherPackage.java
*/
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。
/**
    源程序：AccessControl2.java
*/
package pack2;
import pack1.Original;
public class AccessControl2{
    public static void main(String[] args){
      Derived d = new Derived();
      d.access();
      AnotherPackage s
      s.access();
    }
  }
```

在不同包的子类中,可访问的成员:
Protected member = 3
Public member = 4

在不同包的类中,可访问的成员:
Public member = 4

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。
- 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。
  - 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。

接口

❖接口与抽象类的比较：

- 接口中的方法都是由**public**、**static**和**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)只必须覆盖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。

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。
- 格式：

```
interfaceDeclaration{
    interfaceBody
}
```

其中：

interfaceDeclaration为接口声明部分；

interfaceBody为接口体部分。

接口的定义

- 接口体的定义

  接口体中包括常量定义和方法定义。

- 其格式如下所示：

```
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，static，abstract**属性。

接口的定义

- 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。

接口定义示例

```
/**
    源程序：Collection.java
*/
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

- 接口的方法必须由类的非抽象实现。
- 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

```
/**
    源程序：FIFOQueue.java
*/
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。
- 基于接口的特性，可以把接口作为一个**数据类型**看待。在不需要明确对象所对应的具体的类，用接口表示对象所具有的行为。

接口类型示例

```
/**
    源程序：TestFIFOQueue.java
*/
class TestFIFOQueue{
  public static void main(String args[]){
    Collection cVar = new FIFOQueue();
    Object objAdd = new Object();
    ...
    cVar.add(objAdd);
    ...
  }
}
```

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。

接口应用示例

```
public class HuiDiao {
    public static void main(String args[]) {
        ShowBrand sb;              //接口变量
        sb=new TV();
        sb.show("ChangHong TV");
        sb=new PC();
        sb.show("IBM PC");
    }
}
```

运行结果:

```
*ChangHong TV
#IBM PC
```

思考问题

1. Java是如何应用包管理类的，以及如何引入包或类？

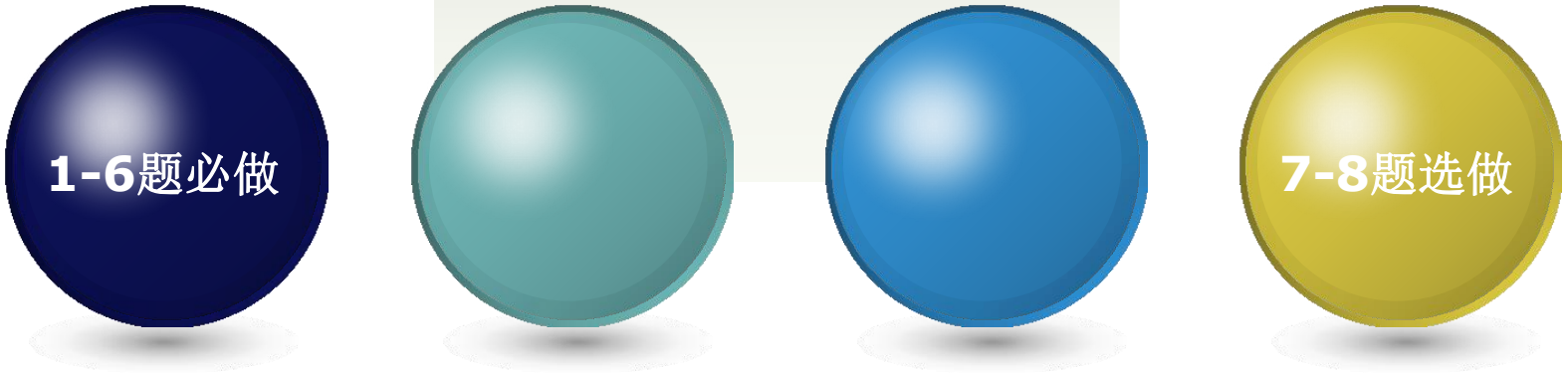
2. 类的成员的访问控制如何实现的？

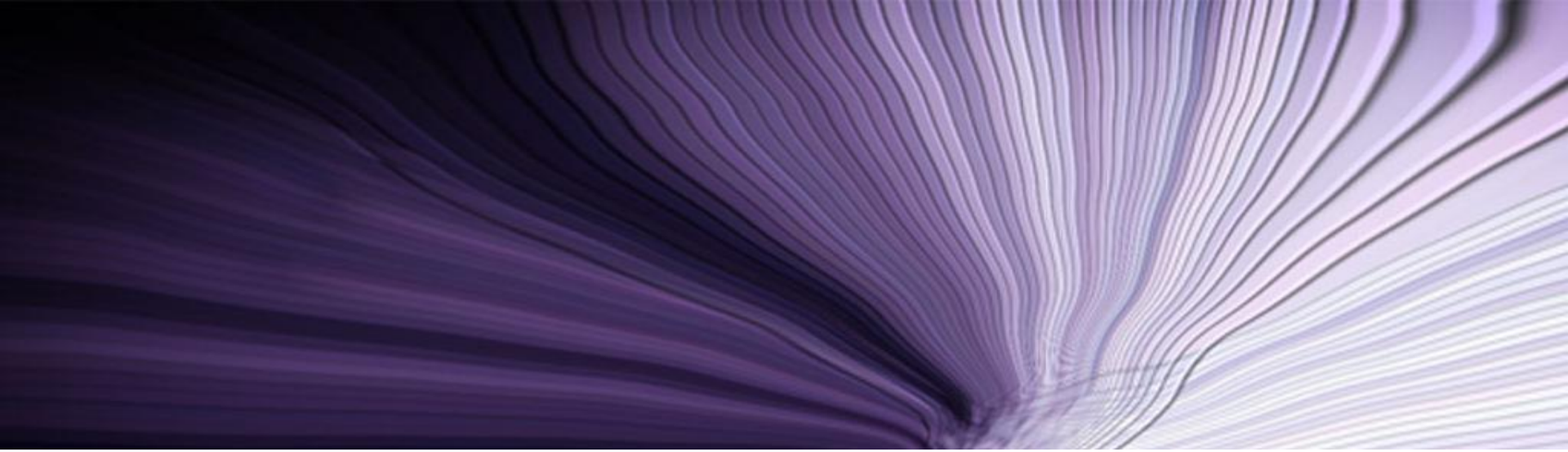
3. 接口有什么特点，它与abstract类有什么区别？

第6章作业

本章习题

习题1-8题





Q&A

电子科技大学信息与软件工程学院