# 1、课程名称：包及访问控制权限


# 2、知识点

## 2.1、上次课程的主要知识点

1、 异常处理的标准格式
2、 RuntimeException 和 Exception 的区别

## 2.2、本次预计讲解的知识点

1、 包的作用、定义、导入
2、 静态导入的语法
3、 常用的开发包
4、 访问控制权限

# 3、具体内容

## 3.1、包的定义及使用（重点）

### 3.1.1、包的定义

在同一个文件夹中是不可能同时存在同一个*.class 文件的，但是在不同的文件夹之中就可以存在，所以所谓的包实际上就属于一个文件夹，如果要定义一个包，可以使用 package 关键字完成。

```
package org.lxh.demo ;
public class Hello {
    public static void main(String args[]){
        System.out.println("Hello World!!!") ;
    }
};
```

此时通过 package 定义了一个包，所以现在一个类的完整名称是“包.类”名称。

**在实际的开发中没有包的类是不存在的。**

一旦程序中定义完了一个包之后就可以通过以下的命令进行编译：javac -d . Hello.java

- -d：表示将根据 package 的定义生成文件夹
- .：表示在当前所在的文件夹之中生成*.class

但是，访问的时候必须连包一起访问：java org.lxh.demo.Hello

### 3.1.2、包的导入

如果现在需要导入不同包的类的话，可以使用 import 语句。

```
import 包.类 ;
```

**范例：**定义一个简单类 —— Info.java

```
package org.demoa ;
class Info {
    public void print(){
        System.out.println("Hello World!!!") ;
    }
};
```

**范例：**在其他的包中使用此类 —— Hello.java

```
package org.demob ;
import org.demoa.Info ;      // 导入所需要的类
public class Hello {
    public static void main(String args[]){
        new Info().print() ;
    }
```

```
};
```

在 Hello.java 的类中导入了 Info 的操作类，并且通过 print()方法进行信息的输出，但是在编译的时候需要先编译 Info.java，再编译 Hello.java，但是在编译 Hello.java 的时候出现了错误：

```
Hello.java:2: org.demoa.Info is not public in org.demoa; cannot be accessed from outside package
import org.demoa.Info ; // 导入所需要的类
```

现在的提示是：由于 Info 不是 public class 声明，所以外包无法访问。

**关于 public class 和 class 声明的区别：**

- public class：类名称必须和文件名称一致，一个类要被外包所访问，需要声明成 public class
- class：类名称可以与文件名称不一致，但是执行的时候要执行类名称，而且不能被外包所访问

**范例：**修改 Info 类的定义

```
package org.demoa ;
public class Info {
    public void print(){
        System.out.println("Hello World!!!") ;
    }
};
```

这个时候程序可以正确的执行，但是在以上代码也可以发现一个问题，即：现在是通过包.类的形式导入的，如果现在一个包中有多个类的话，则采用这种方式就太麻烦了，如果现在要是导入一个包的多个类，使用如下的语法：

```
import org.demoa.* ;  // 导入所需要的类
```

使用*的导入与具体类的导入在性能上是完全一样的，因为即使使用了*，也会根据自己的需要加载所需要的类，不需要的类根本就不会被导入。

但是在进行导包的时候有一点也必须注意，如果现在同时导入了不同包的同名类的话呢？

```
package org.democ ;
public class Info {
    public void getHello(){
        System.out.println("Hello") ;
    }
};
```

之后在 Hello.java 中同时导入 demoa 和 democ 两个包。

```
package org.demob ;
import org.demoa.* ; // 导入所需要的类
import org.democ.* ; // 导入所需要的类
public class Hello {
    public static void main(String args[]){
        new Info().print() ;
    }
};
```

由于这两个包中都有 Info 的类，则此时的程序将无法分辨出来：

```
Hello.java:6: reference to Info is ambiguous, both class org.democ.Info in org.democ and class org.demoa.Info in org.demoa
match
```

那么在这种情况下，只能通过完整的“包.类”的方式进行对象的实例化。

```
package org.demob ;
import org.demoa.* ;  // 导入所需要的类
import org.democ.* ;  // 导入所需要的类
```

```
public class Hello {
    public static void main(String args[]){
        org.demoa.Info info = new org.demoa.Info() ;
        info.print() ;
    }
};
```

## 3.1.3、JDK 1.5 新特性 —— 静态导入（了解）

在 JDK 1.5 之后，如果一个类中的全部方法都是静态方法的话，则可以使用静态的导入方式，直接将方法导入进来。

```
package org.lxh.demo ;
public class MyMath {
    public static int add(int x,int y){
        return x + y ;
    }
    public static int sub(int x,int y){
        return x - y ;
    }
};
```

在最早的做法中，是将 MyMath 的类导入，所以语法如下：

```
package org.hello ;
import org.lxh.demo.* ;      // 导入所需要的类
public class MyMathDemo {
    public static void main(String args[]){
        System.out.println(MyMath.add(10,20)) ;
    }
};
```

但是，如果现在使用的是静态导入：

```
package org.hello ;
import static org.lxh.demo.MyMath.* ;      // 静态导入所需要的方法
public class MyMathDemo {
    public static void main(String args[]){
        System.out.println(add(10,20)) ;
    }
};
```

## 3.2、系统的常用包（了解）

在 Java 语言中为开发者准备了各种各样的开发包，常见的包有以下几种：

- java.lang：是一个基础的开发包，里面包含了一些常用类：String、Integer、Exception、Object
- java.lang.reflect：是反射操作包
- java.util：为开发的工具包
- java.io：完成 IO 操作的开发包

- java.sql：完成数据库的开发包
- java.text：格式化的开发包
- java.applet：Applet 程序的实现包
- java.awt、javax.swing：图形界面的开发包

对于 Applet 程序现在基本上已经是不再使用了，Applet 程序与 Application 程序的主要区别就是在于 Applet 没有主方法，而且必须编写 HTML 代码才可以执行。

**范例：**打印 Hello World

```
package org.lxh.demo ;
import java.applet.* ;
import java.awt.* ;
public class HelloApplet extends Applet {
    public void paint(Graphics g){
        g.drawString("Hello World",10,20) ;
    }
};
```

编译完成之后，就需要通过 HTML 代码访问此 Applet。

```
<applet code="org.lxh.demo.HelloApplet" height="300" width="300"/>
```

一般对于图形界面（GUI）开发中主要是依靠 java.awt 和 javax.swing。但是这两种包的开发只是适合于单机程序。

JAVA 2 的最大特点就是增加了 javax.swing 的轻量级的组件包。

## 3.3、JAR 命令（重点）

在实际的开发中，如果一个开发人员已经开发出了许多的类，这个时候如果要交付给客户使用的话，则一个个独立的*.class 文件会太多，维护起来也不方便，所以一般都会将其进行压缩，在 java 中就依靠 jar 进行压缩。

```
package org.democ ;
public class Info {
    public void getHello(){
        System.out.println("Hello") ;
    }
};
```

输入以下的命令进行压缩的操作：

```
D:\testjava>jar -cvf my.jar org
added manifest
adding: org/(in = 0) (out= 0)(stored 0%)
adding: org/democ/(in = 0) (out= 0)(stored 0%)
adding: org/democ/Info.class(in = 396) (out= 280)(deflated 29%)
```

**范例：**测试 Info 类

```
package test ;
import org.democ.* ;
public class TestInfo {
    public static void main(String args[]){
        new Info().getHello() ;
    }
```

```
};
```

但是如果要想让以上的程序可以正确的编译的话，则必须配置 classpath。

```
SET CLASSPATH=.;d:\testjava\my.jar
```

但是，这种配置只是针对于一个命令行窗口完成的，如果现在要针对于所有的命令行窗口，则必须在属性中增加 CLASSPATH 属性。


## 3.4、命名规范（重点）

在进行 Java 开发的时候一定要遵守以下的命名规范要求：

1、 定义类名称的时候，每个单词的首字母大写：HelloInfo
2、 定义方法名称的时候，第一个单词的首字母小写，之后每个单词的首字母大写：printInfo()
3、 定义属性的时候，第一个单词的首字母小写，之后每个单词的首字母大写：empName
4、 定义常量的时候，所有的单词字母都要大写：INFO
5、 定义包名称的时候，所有的单词的字母都要小写：org.demo

## 3.5、访问控制权限（重点）

在 Java 中一共规定出了四种访问控制权限：

| No. | 位置 | private | default | protected | public |
|---|---|---|---|---|---|
| 1 | 本类 | √ | √ | √ | √ |
| 2 | 本包中的其他类 | × | √ | √ | √ |
| 3 | 不同包的子类 | × | × | √ | √ |
| 4 | 不同包的非子类 | × | × | × | √ |

对于 private、default、public 基本上的使用都差不多了，就差一个 protected 权限。

**范例：**定义父类

```
package org.demoa ;
public class A {
    protected String name = "Hello World!!!" ;
};
```

**范例：**定义不同包的子类

```
package org.demob ;
import org.demoa.* ;
public class B extends A {
    public void print(){
        System.out.println(super.name) ;  // 访问 protected 属性
```

```
    }
};
```

之后通过程序进行测试。

```
package org.democ ;
import org.demob.* ;
public class C {
    public static void main(String args[]){
        new B().print() ;
    }
};
```

# 3.6、类图的表示（重点）

在进行程序的开发中，往往都会使用类图的形式表示出类之间的关系，或者是类的组成，类图的画法也是有严格要求的。

## 3.6.1、类

如果要想表示出一个类的关系，可以使用如下的一种图形：

| 类名称 |
| --- |
| 属性名称 |
| 方法名称 |

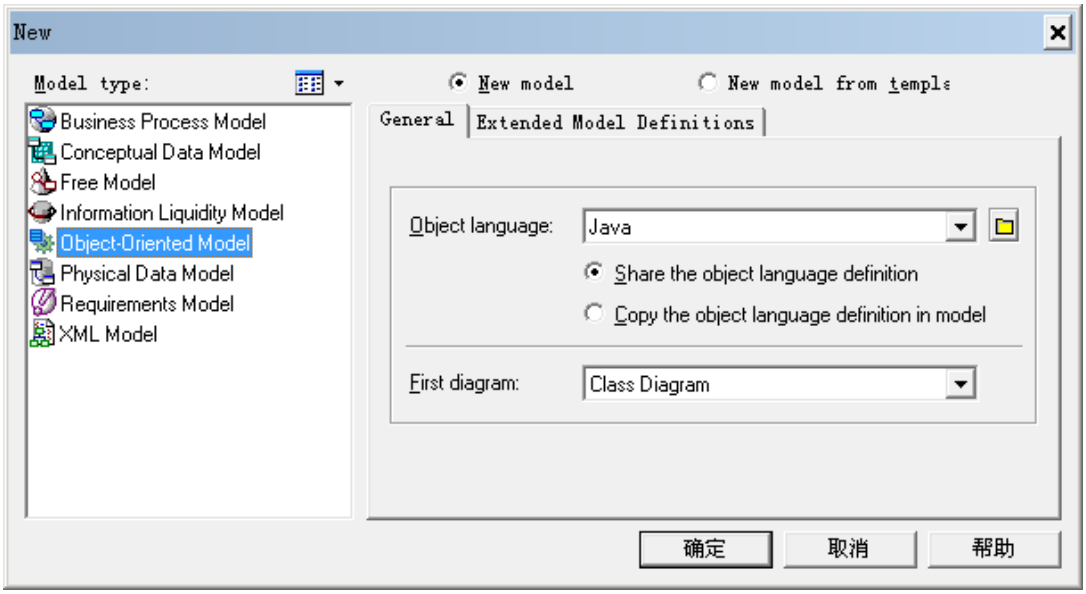
对于属性名称一定要按照此格式：访问权限 属性名称:属性类型

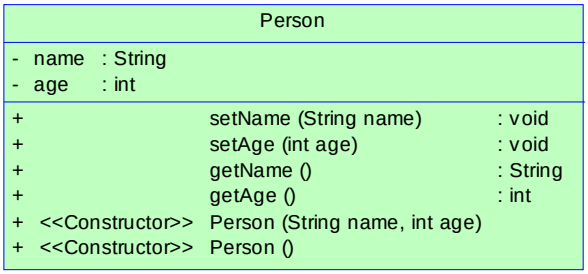
- 权限：private（-）、protected（#）、public（+）
- 例如：private String name à -name:String

方法名称本身也是有严格要求的，格式：访问权限 方法名称(参数类型 参数名称,..):返回值类型

- 例如：public String getName() à +getName():String

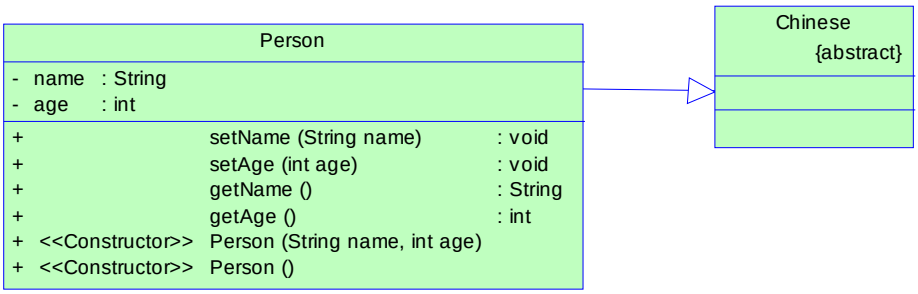
但是，如果所有的类都是这样手工画的话，那么基本上程序就不用写了，所以开发中都会通过设计工具完成。



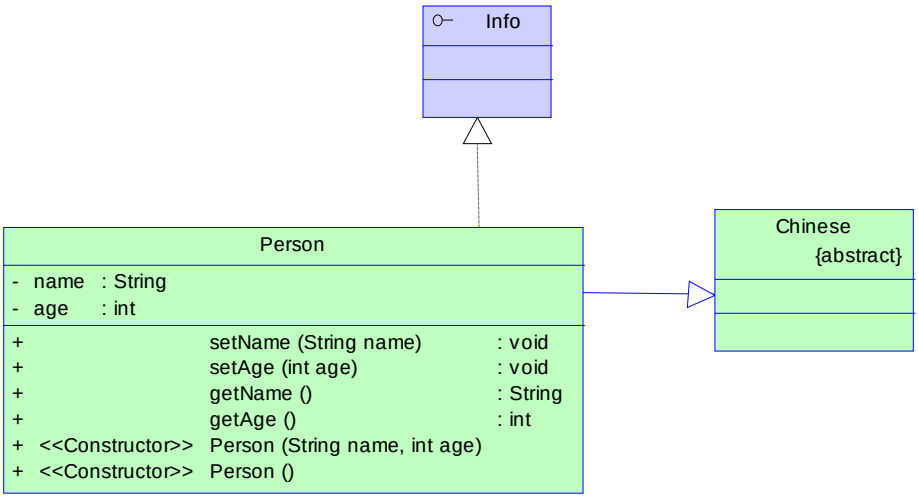



使用 PowerDesigner 工具最大的好处还再与可以自动生成代码。

在程序中，一个类也可以继承一个父类（父类一般为抽象类）

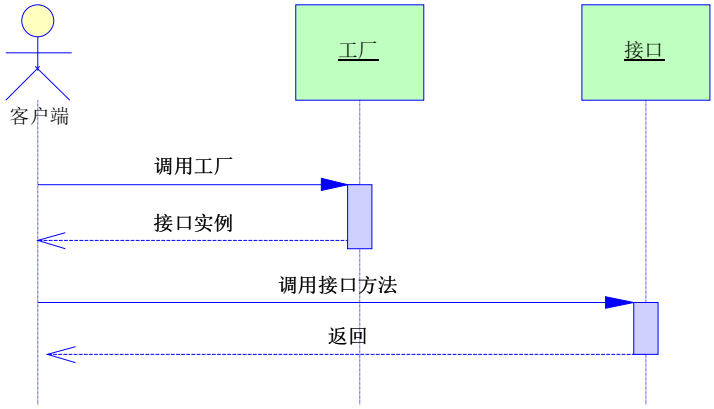



类也可以实现接口：



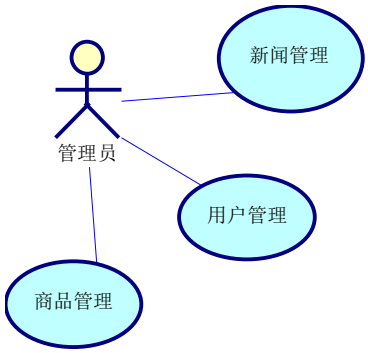

# 3.6.2、时序图

时序图表示的是一种操作的进行的状态，例如：以工厂设计为例。



### 3.6.3、用例图

用例图一般是在分析的时候使用，表示一个角色可以拥有的功能，例如：管理员，可以进行商品管理、新闻管理、用户管理。




# 4、总结

1、 面向对象部分就彻底完了，核心的部分就是接口，对象多态性，面向对象三大特征全部讲解完毕。
2、 包的定义和导入
3、 四种访问控制权限