Java程序设计

第4章 类、包和接口

第4章 类、包和接口

- 本章介绍Java中面向对象的程序设计的基本方法，包括类的定义、类的继承、包、访问控制、修饰符、接口等方面的内容。
- 4.1 类、字段、方法
- 4.2 类的继承
- 4.3 包
- 4.4 访问控制符
- 4.5 非访问控制符
- 4.6 接口
- 4.7 枚举

4.1 类、字段、方法

类、字段、方法

类

- 类是组成Java程序的基本要素
- 是一类对象的原型
- 它封装了一类对象的状态和方法
  - 它将变量与函数封装到一个类中

字段和方法

```
class Person {
        String name;
        int age;
        void sayHello(){
                System.out.println("Hello!  My name is" + name );
        }
}
```

- 字段（field）是类的属性，是用变量来表示的。
  - 字段又称为域、域变量、属性、成员变量等
- 方法（method）是类的功能和操作，是用函数来表示的

构造方法

- 构造方法（constructor）是一种特殊的方法
- 用来初始化（new）该类的一个新的对象
- 构造方法和类名同名，而且不写返回数据类型。

```
Person( String n, int a ){
        name = n;
        age = a;
}
```

默认构造方法

- 一般情况下，类都有一个至多个构造方法
- 如果没有定义任何构造方法，系统会自动产生一个构造方法，称为默认构造方法（default constructor）。
- 默认构造方法不带参数，并且方法体为空。

使用对象

- 访问对象的字段或方法，需要用算符“.”：
- Person p = new Person();
- System.out.println(p.name) ;
- p.sayHello();
- 这种使用方式的好处
  - 封装性
  - 安全性

方法重载（overload)

- 方法重载（overloading)：多个方法有相同的名字，编译时能识别出来。
- 这些方法的签名（signature)不同，或者是参数个数不同，或者是参数类型不同。
- 通过方法重载可以实现多态（polymorphism）。

- MethodOverloadingTest.java

this的使用

- 1．在方法及构造方法中，使用this来访问字段及方法
- 例如，方法sayHello中使用name和使用this.name是相同的。即：

```
void sayHello(){
        System.out.println("Hello!  My name is " + name );
    }
```

- 与

```
    void sayHello(){
        System.out.println("Hello!  My name is " + this.name );
    }
```

- 的含义是相同的。

this的使用（2）

- 2．使用this解决局部变量与域同名的问题
- 使用this还可以解决局部变量（方法中的变量）或参数变量与域变量同名的问题。如，在构造方法中，经常这样用：

```
Person( int age, String name ) {
        this.age = age;
        this.name = name;
}
```

- 这里，this.age表示域变量，而age表示的是参数变量。

this的使用（3）

- 3．构造方法中，用this调用另一构造方法
- 构造方法中，还可以用this来调用另一构造方法。如：

```
Person( )
{
        this( 0, "" );
        ……
}
```

- 在构造方法中调用另一构造方法，则这条调用语句必须放在第一句。

4.2 类的继承

类的继承

继承

- 继承(inheritance)是面向对象的程序设计中最为重要的特征之一
- 子类（subclass），父类或超类（superclass）
  - 父类包括所有直接或间接被继承的类
- Java支持单继承：一个类只能有一个直接父类。

继承的好处

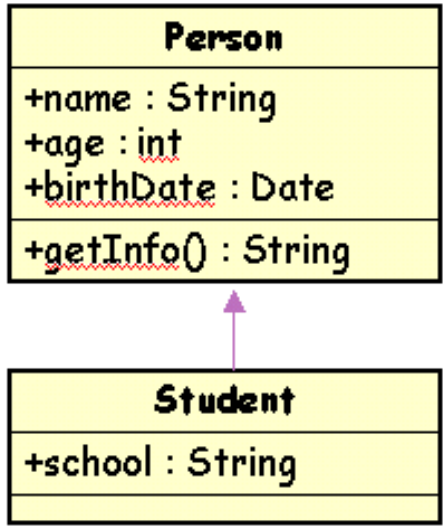
- 子类继承父类的状态和行为
  - 可以修改父类的状态或重载父类的行为
  - 可以添加新的状态和行为。
- 好处
  - 可以提高程序的抽象程度
  - 实现代码重用，提高开发效率和可维护性

派生子类

- Java中的继承是通过extends关键字来实现的
- class Student extends Person {
- ……
- }
- 如果没有extends子句，则该类默认为java.lang.Object的子类。
  - 所有的类都是通过直接或间接地继承java.lang.Object得到的。

继承关系

- 继承关系在UML图中，是用一个箭头来表示子类与父类的关系的。
- 相当于 is a



- 类Student从类Person继承，定义如下：

```
class  Student  extends Person {
    String school;
    int score;
    boolean  isGood(){ return score>80; }
    //...
}
```

字段

- 1．字段的继承
  - 子类可以继承父类的所有字段
  - Student自动具有Person的属性（name，age）
- 2. 字段的隐藏
  - 子类重新定义一个与从父类那里继承来的域变量完全相同的变量，称为域的隐藏。域的隐藏在实际编程中用得较少。
- 3．字段的添加
  - 在定义子类时，加上新的域变量，就可以使子类比父类多一些属性。如：
  - class Student extends Person
  - {
  - String school;
  - int score;
  - }

方法

- 1．方法的继承
  - 父类的非私有方法也可以被子类自动继承。如，Student自动继承Person的方法sayHello和isOlderThan。
- 2．方法的覆盖(Override)（修改）
  - 子类也可以重新定义与父类同名的方法，实现对父类方法的覆盖(Override)。

@Override

- @Override //JDK1.5以后可以用这个注记来表示(不用也是可以的)

```
void sayHello(){
        System.out.println("Hello!  My name is " + name + ". My school is " + school );
}
```

- 通过方法的覆盖，能够修改对象的同名方法的具体实现方法。

方法

- 3.方法的添加

- 子类可以新加一些方法，以针对子类实现相应的功能。
- 如，在类Student中，加入一个方法，对分数进行判断：

```
boolean isGoodStudent(){
    return score>=90;
}
```

方法

- 4. 方法的重载
- 一个类中可以有几个同名的方法，这称为方法的重载（Overload）。同时，还可以重载父类的同名方法。与方法覆盖不同的是，重载不要求参数类型列表相同。重载的方法实际是新加的方法。
- 如，在类Student中，重载一个名为sayHello的方法：

```
void sayHello( Student another ){
        System.out.println("Hi!");
        if( school .equals( another.school ))
           System.out.println(" Shoolmates ");
}
```

super的使用

- 1．使用super访问父类的域和方法
- 注意：正是由于继承，使用this可以访问父类的域和方法。但有时为了明确地指明父类的域和方法，就要用关键字super。
- 例如：父类Student有一个域age，在子类Student中用age, this.age, super.age来访问age是完全一样的：

```
void testThisSuper(){
    int a;
    a = age;
    a = this.age;
    a = super.age;
}
```

- 当然，使用super不能访问在子类中添加的域和方法。

- 有时需要使用super以区别同名的域与方法
  - 使用super可以访问被子类所隐藏了的同名变量。
  - 又如，当覆盖父类的同名方法的同时，又要调用父类的方法，就必须使用super。如：

```
void sayHello(){
    super.sayHello();
    System.out.println( "My school is " + school );
}
```

- 在覆盖父类的方法的同时，又利用已定义好的父类的方法。

super的使用(2)

- 2．使用父类的构造方法
- 构造方法是不能继承的
  - 比如，父类Person有一个构造方法Person(String, int)，不能说子类Student也自动有一个构造方法Student(String, int)。
- 但是，子类在构造方法中，可以用super来调用父类的构造方法。

```
Student(String name, int age, String school ){
        super( name, age );
        this.school = school;
}
```

- 使用时，super()必须放在第一句。
- 有关构造方法的更详细的讨论，参见第5章。

父类对象与子类对象的转换

- 类似于基本数据类型数据之间的强制类型转换，存在继承关系的父类对象和子类对象之间也可以在一定条件下相互转换。
- (1) 子类对象可以被视为其父类的一个对象
  - 如一个Student对象也是一个Person对象。
- (2) 父类对象不能被当做其某一个子类的对象。
- (3) 如果一个方法的形式参数定义的是父类对象，那么调用这个方法时，可以使用子类对象作为实际参数。
- (4) 如果父类对象引用指向的实际是一个子类对象，那么这个父类对象的引用可以用强制类型转换（casting)成子类对象的引用。
- 例： Student.java

4.3 包

包

package

- package pkg1[.pkg2[.pkg3...]];
- 包及子包的定义，实际上是为了解决名字空间、名字冲突
  - 它与类的继承没有关系。事实上，一个子类与其父类可以位于不同的包中。
- 包有两方面的含义
  - 一是名字空间、存储路径（文件夹）、
  - 一是可访问性（同一包中的各个类，默认情况下可互相访问）

package语句

- 包层次的根目录是由环境变量CLASSPATH来确定的。
- 在简单情况下，没有package语句，这时称为无名包（unnamed package）
  - 在Eclipse中，也叫(default package)。
- Java的JDK提供了很多包
  - java.applet，java.awt，java.awt.image，java.awt.peer，java.io，java.lang，java.net，java.util，javax.swing，等。

import语句

- 为了能使用Java中已提供的类，需要用import语句来导入所需要的类。
- import语句的格式为：
  - import package1[.package2…]. (classname |*);
- 例如：
  - import java.util.Date;
    - 这样，程序中 java.util.Date可以简写为Date
  - import java.awt.*;
  - import java.awt.event.*;
  - 注意：使用星号(*)只能表示本层次的所有类，不包括子层次下的类。
- Java编译器自动导入包java.lang.*
- Eclipse等IDE可以方便地生成import语句

编译和运行包中的类

- 使用javac可以将.class文件放入到相应的目录，只需要使用一个命令选项-d来指明包的根目录即可。
- javac -d d:\tang\ch04 d:\tang\ch04\pk\TestPkg.java
- javac -d . pk*.java
- 其中，“.”表示当前目录
- 运行该程序，需要指明含有main的类名：
- java pk.TestPkg

CLASSPATH

- 在编译和运行程序中，经常要用到多个包，怎样指明这些包的根目录呢？简单地说，包层次的根目录是由环境变量CLASSPATH来确定的。具体操作有两种方法。
- 一是在java及javac命令行中，用-classpath（或-cp)选项来指明，如：
- java -classpath d:\tang\ch04;c:\java\classes;. pk.TestPkg
- 二是设定classpath环境变量，用命令行设定环境变量，如：
- set classpath= d:\tang\ch04;c:\java\classes;.
- 在Windows中还可以按第2章中的办法设定环境变量。

4.4 访问控制符

访问控制符

修饰符

- 修饰符（modifiers）分为两类
  - 访问修饰符（access modifiers）
    - 如public/private等
  - 其他修饰符
    - 如abstract等
- 可以修饰类、也可以修饰类的成员（字段、方法）

成员的访问控制符（权限修饰符）

	同一个类中	同一个包中	不同包中的子类	不同包中的非子类
private	Yes			
默认（包可访问）	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

类的访问控制符

- 在定义类时，也可以用访问控制符。
- 类的访问控制符或者为public，或者默认。
  - 若使用public，其格式为:

```
public class 类名{
    ……
}
```

  - 如果类用public修饰，则该类可以被其他类所访问；
  - 若类默认访问控制符，则该类只能被同包中的类访问。

setter与getter

- 将字段用private修饰，从而更好地将信息进行封装和隐藏。
- 用setXXXX和getXXXX方法对类的属性进行存取，分别称为setter与getter。
- 这种方法有以下优点
  - （1）属性用private更好地封装和隐藏，外部类不能随意存取和修改。
  - （2）提供方法来存取对象的属性，在方法中可以对给定的参数的合法性进行检验。
  - （3）方法可以用来给出计算后的值。
  - （4）方法可以完成其他必要的工作（如清理资源、设定状态，等等）。
  - （5）只提供getXXXX方法，而不提供setXXXX方法，可以保证属性是只读的。

Setter/getter示例

```
class  Person2
{
        private int age;
        public void setAge( int age ){
                if (age>0 && age<200) this.age = age;
        }
        public  int getAge(){
                return age;
        }
}
```

4.5 其他修饰符

其他修饰符

非访问控制符

	基本含义	修饰类	修饰成员	修饰局部变量
static	静态的、非实例的、类的	可以修饰内部类	Yes	
final	最终的、不可改变的	Yes	Yes	Yes
abstract	抽象的、不可实例化的	Yes	Yes	

static 字段

- 静态字段最本质的特点是：
  - 它们是类的字段，不属于任何一个对象实例。
- 它不保存在某个对象实例的内存区间中，而是保存在类的内存区域的公共存储单元。
- 类变量可以通过类名直接访问，也可以通过实例对象来访问，两种方法的结果是相同的。
- 如System类的in和out对象，就是属于类的域，直接用类名来访问，即System.in和System.out。

例如

- 在类Person中可以定义一个类域为totalNum：
- class Person {
- 　　static long totalNum;
- 　　int age;
- 　　String Name;
- }
- totalNum代表人类的总人数，它与具体对象实例无关。可以有两种方法来访问：Person.totalNum和p.totalNum (假定p是Person对象)。
- 在一定意义上，可以用来表示全局变量

static方法

- 用static修饰符修饰的方法仅属于类的静态方法，又称为类方法。
- 与此相对，不用static修饰的方法，则为实例方法。
- 类方法的本质是该方法是属于整个类的，不是属于某个实例的。
- 声明一个方法为static有以下几重含义。
- (1) 非static的方法是属于某个对象的方法，在这个对象创建时，对象的方法在内存中拥有自己专用的代码段。而static的方法是属于整个类的，它在内存中的代码段将随着类的定义而进行分配和装载，不被任何一个对象专有。

- (2) 由于static方法是属于整个类的，所以它不能操纵和处理属于某个对象的成员变量，而只能处理属于整个类的成员变量，即static方法只能处理本类中的static域或调用static方法。

- (3) static方法中，不能访问实例变量，不能使用this 或super。

- (4) 调用这个方法时，应该使用类名直接调用，也可以用某一个具体的对象名。
  - 例如：Math.random()，Integer.parseInt()等就是类方法，直接用类名进行访问。

import static

- import static java.lang.System.*;

- out.println();表示System.out.println();

final

- 1．final类
  - 如果一个类被final修饰符所修饰和限定，说明这个类不能被继承，即不可能有子类。
- 2．final方法
  - final修饰符所修饰的方法，是不能被子类所覆盖的方法。

final

- 3．final字段及final局部变量
- final字段、final局部变量(方法中的变量)
  - 它们的值一旦给定，就不能更改。
  - 是只读量，它们能且只能被赋值一次，而不能被赋值多次。
- 一个字段被static final两个修饰符所限定时，它可以表示常量，
  - 如Integer. MAX_VALUE(表示最大整数)、Math.PI(表示圆周率)就是这种常量。
- 关于赋值
  - 在定义static final域时，若不给定初始值，则按默认值进行初始化（数值为0，boolean型为false，引用型为null）。
  - 在定义final字段时，若不是static的域，则必须且只能赋值一次，不能缺省。
    - 这种域的赋值的方式有两种：一是在定义变量时赋初始值，二是在每一个构造函数中进行赋值。
  - 在定义final局部变量时，也必须且只能赋值一次。它的值可能不是常量，但它的取值在变量存在期间不会改变。

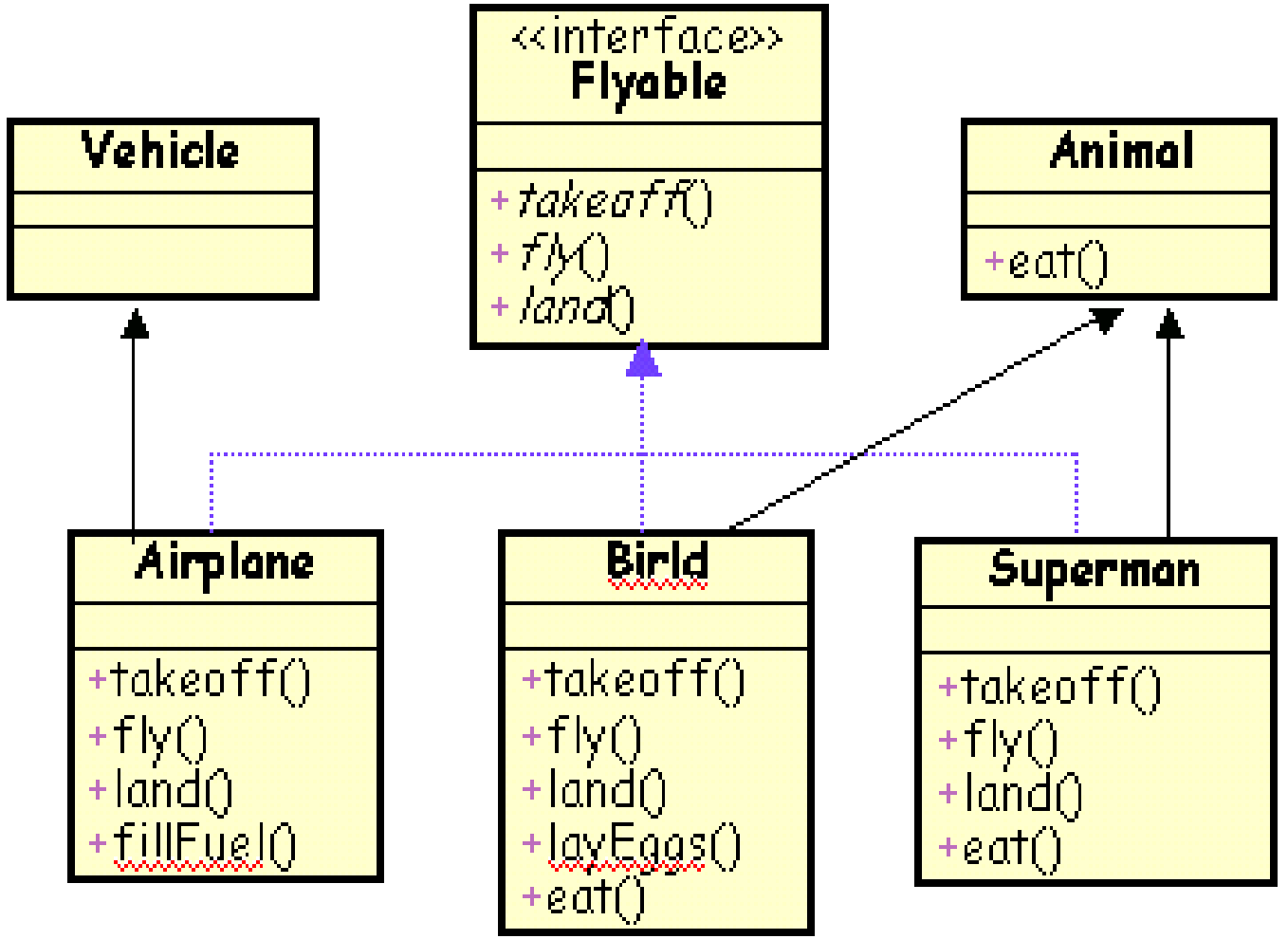
abstract

- 1．abstract类
  - 凡是用abstract修饰符修饰的类被称为抽象类。
  - 抽象类不能被实例化
- 2．abstract方法
  - 被abstract所修饰的方法叫抽象方法，抽象方法的作用在为所有子类定义一个统一的接口。对抽象方法只需声明，而不需实现，即用分号（；）而不是用{}，格式如下：
  - abstract returnType abstractMethod([paramlist]);
  - 抽象类中可以包含抽象方法，也可以不包含abstract方法。但是，一旦某个类中包含了abstract方法，则这个类必须声明为abstract类。
  - 抽象方法在子类中必须被实现，否则子类仍然是abstract的。

接口

接口（interface）

- 接口，某种特征的约定
  - 定义接口 interface
    - 所有方法都自动是public abstract
  - 实现接口 implements
    - 可以实现多继承
    - 与类的继承关系无关
- 面向接口编程，而不是面向实现
  - Flyable f = new Bird();
  - Java中有大量的接口



接口的作用

- 1. 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。从而在一定意义上实现了多重继承。
- 2. 通过接口可以指明多个类需要实现的方法。
- 3. 通过接口可以了解对象的交互界面，而不需了解对象所对应的类。

示例

- 下面我们给出一个接口的定义：
- interface Collection {
  - void add (Object obj);
  - void delete (Object obj);
  - Object find (Object obj);
  - int size ( );
- }

- 通常接口以able或ible结尾，表明接口能完成一定的行为。
- 接口声明中还可以包括对接口的访问权限以及它的父接口列表。完整的接口声明如下：
- [public] interface interfaceName [extends listOfSuperInterface]{
- ……
- }
- 其中public指明任意类均可以使用这个接口，缺省情况下，只有与该接口定义在同一个包中的类才可以访问这个接口。
- extends 子句与类声明中的extends子句基本相同，不同的是一个接口可以有多个父接口，用逗号隔开，而一个类只能有一个父类。子接口继承父接口中所有的常量和方法。

- 方法定义的格式为：
- returnType methodName ([paramlist]) ;
- 接口中只进行方法的声明，而不提供方法的实现，所以，方法定义没有方法体，且用分号(;)结尾。在接口中声明的方法具有public 和abstract属性。
  - 所以定义的时候这两个关键词是可以省略的
- 另外，如果在子接口中定义了和父接口同名的常量或相同的方法，则父接口中的常量被隐藏，方法被重载。

接口的实现

- 在类的声明中用implements子句来表示一个类使用某个接口，在类体中可以使用接口中定义的常量，而且必须实现接口中定义的所有方法。一个类可以实现多个接口。

- 下面我们在类FIFOQueue中实现上面所定义的接口collection：
- class FIFOQueue implements collection{
- public void add (Object obj){
- ……
- }
- public void delete(Object obj){
- ……
- }
- public Object find(Object obj){
- ……
- }
- public int currentCount{
- ……
- }
- 在类中实现接口所定义的方法时，方法的声明必须与接口中所定义的完全一致。

3. 接口类型

- 接口可以作为一种引用类型来使用。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的接口中的方法。Java运行时系统动态地确定该使用哪个类中的方法。
- 把接口作为一种数据类型可以不需要了解对象所对应的具体的类，以前面所定义的接口Collection和实现该接口的类FIFOQueue为例，下例中，我们以Collection作为引用类型来使用。

- public static void main(String args[]){
  - Collection c = new FIFOQueue();
  - ……
  - c.add(obj);
  - ……
- }

接口中的常量

- 接口体中可以包含常量定义
- 常量定义的格式为：
  - type NAME = value;
- 其中type可以是任意类型，NAME是常量名，通常用大写，value是常量值。
- 在接口中定义的常量可以被实现该接口的多个类共享，它与 C中用#define以及C++中用const定义的常量是相同的。
- 在接口中定义的常量具有public, static, final的属性。

枚举

- 从JDK1.5起，可以使用枚举
  - enum　Light {　Red,　Yellow, Green }
- 使用
  - Ligth light = Light.Red;
  - switch(light) {　　case Red: ..... Break; }
  - 注意：case后面不写为 Light.Red
- Java中的枚举是用class来实现的，可以复杂地使用

Java8中的接口

- Java8以上，接口成员还可以是：
  - static方法
  - 具有实现体的方法（default方法）
    - 默认方法的好处是：提供了一个默认实现，子类在implements可以不用再重新写了

语法小结

完整的类定义

```
[public] [abstact|final] class className [extends superclassName]
[implements InterfaceNameList]{ //类声明
    [public | protected | private] [static] [final] [transient] [volatile] type variableName;
    //成员变量声明，可为多个
    [public | protected | private] [static] [final | abstract] [native] [synchronized]
    returnType methodName ( [paramList] ) //方法定义及实现，可为多个
    [throws exceptionList]{
        statements
    }
}
```

完整的接口定义

- [public] interface InterfaceName [extends superInterfaceList]{ //接口声明
- type constantName = Value; //常量声明，可为多个
- retumType methodName ([paramList]); //方法声明，可为多个
- }

有三种方法要求固定的声明方式

- (1) 构造方法，声明为：

```
className( [paramlist] ){
    ......
}
```

- (2) main( )方法，声明为：

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

- 3. finalize( )方法，声明为：

```
protected void finalize( ) throws throwable{
    ......
}
```

-

完整的java源文件

- package packageName; //指定文件中的类所在的包，0个或1个
- import packageName.[className|*]; //指定引入的类，0个或多个
- public classDefinition //属性为public的类定义，0个或1个
- interfaceDefinition and classDefinition //接口或类定义，0个或多个。
-
- 源文件的名字必须与属性为public的类的类名完全相同
- 在一个.java文件中，package语句和public类最多只能有1个。