# 第13天面向对象

今日内容介绍

final

static

匿名对象

内部类

包的声明与访问

四种访问修饰符

代码块

# final关键字

## final的概念

继承的出现提高了代码的复用性，并方便开发。但随之也有问题，有些类在描述完之后，不想被继承，或者有些类中的部分方法功能是固定的，不想让子类重写。可是当子类继承了这些特殊类之后，就可以对其中的方法进行重写，那怎么解决呢？

要解决上述的这些问题，需要使用到一个关键字final，final的意思为最终，不可变。final是个修饰符，它可以用来修饰类，类的成员，以及局部变量。

## final的特点

final修饰类不可以被继承，但是可以继承其他类。

**class** Yy {}

**final** **class** Fu **extends** Yy{} //可以继承Yy类

**class** Zi **extends** Fu{} //不能继承Fu类

final修饰的方法不可以被覆盖,但父类中没有被final修饰方法，子类覆盖后可以加final。

**class** Fu {

// final修饰的方法，不可以被覆盖，但可以继承使用

**public** **final** **void** method1(){}

**public** **void** method2(){}

}

**class** Zi **extends** Fu {

//重写method2方法

**public** **final** **void** method2(){}

}

final修饰的变量称为常量，这些变量只能赋值一次。

**final** **int** i = 20;

i = 30; //赋值报错，final修饰的变量只能赋值一次

* 引用类型的变量值为对象地址值，地址值不能更改，但是地址内的对象属性值可以修改。

**final** Person p = **new** Person();

Person p2 = **new** Person();

p = p2; //final修饰的变量p，所记录的地址值不能改变

p.name = "小明";//可以更改p对象中name属性值

p不能为别的对象，而p对象中的name或age属性值可更改。

修饰成员变量，需要在创建对象前赋值，否则报错。(当没有显式赋值时，多个构造方法的均需要为其赋值。)

**class** Demo {

//直接赋值

**final** **int** m = 100;

//final修饰的成员变量，需要在创建对象前赋值，否则报错。

**final** **int** n;

**public** Demo(){

//可以在创建对象时所调用的构造方法中，为变量n赋值

n = 2016;

}

}

# static关键字

## static概念

当在定义类的时候，类中都会有相应的属性和方法。而属性和方法都是通过创建本类对象调用的。当在调用对象的某个方法时，这个方法没有访问到对象的特有数据时，方法创建这个对象有些多余。可是不创建对象，方法又调用不了，这时就会想，那么我们能不能不创建对象，就可以调用方法呢？

可以的，我们可以通过static关键字来实现。static它是静态修饰符，一般用来修饰类中的成员。

## static特点

被static修饰的成员变量属于类，不属于这个类的某个对象。（也就是说，多个对象在访问或修改static修饰的成员变量时，其中一个对象将static成员变量值进行了修改，其他对象中的static成员变量值跟着改变，即多个对象共享同一个static成员变量）

代码演示：

**class** Demo {

**public** **static** **int** *num* = 100;

}

**class** Test {

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

Demo d1 = **new** Demo();

Demo d2 = **new** Demo();

d1.*num* = 200;

System.***out***.println(d1.*num*); //结果为200

System.***out***.println(d2.*num*); //结果为200

}

}

被static修饰的成员可以并且建议通过类名直接访问。

访问静态成员的格式：

类名.静态成员变量名

类名.静态成员方法名(参数)

对象名.静态成员变量名 ------不建议使用该方式，会出现警告

对象名.静态成员方法名(参数) ------不建议使用该方式，会出现警告

代码演示：

**class** Demo {

//静态成员变量

**public** **static** **int** *num* = 100;

//静态方法

**public** **static** **void** method(){

System.***out***.println("静态方法");

}

}

**class** Test {

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

System.***out***.println(Demo.*num*);

Demo.*method*();

}

}

## static注意事项

* 静态内容是优先于对象存在，只能访问静态，不能使用this/super。静态修饰的内容存于静态区。

**class** Demo {

//成员变量

**public** **int** num = 100;

//静态方法

**public** **static** **void** method(){

//this.num; 不能使用this/super。

System.***out***.println(**this**.num);

}

}

* 同一个类中，静态成员只能访问静态成员

**class** Demo {

//成员变量

**public** **int** num = 100;

//静态成员变量

**public** **static** **int** *count* = 200;

//静态方法

**public** **static** **void** method(){

//System.out.println(num); 静态方法中，只能访问静态成员变量或静态成员方法

System.***out***.println(*count*);

}

}

* main方法为静态方法仅仅为程序执行入口，它不属于任何一个对象，可以定义在任意类中。

## 定义静态常量

开发中，我们想在类中定义一个静态常量，通常使用public static final修饰的变量来完成定义。此时变量名用全部大写，多个单词使用下划线连接。

定义格式：

public static final 数据类型 变量名 = 值;

如下演示：

**class** Company {

**public** **static** **final** String ***COMPANY\_NAME*** = "传智播客";

**public** **static** **void** method(){

System.***out***.println("一个静态方法");

}

}

当我们想使用类的静态成员时，不需要创建对象，直接使用类名来访问即可。

System.***out***.println(Company.***COMPANY\_NAME***); //打印传智播客

Company.*method*(); // 调用一个静态方法

注意：

接口中的每个成员变量都默认使用public static final修饰。

所有接口中的成员变量已是静态常量，由于接口没有构造方法，所以必须显示赋值。可以直接用接口名访问。

interface Inter {

public static final int *COUNT* = 100;

}

访问接口中的静态变量

Inter.COUNT

# 匿名对象

## 匿名对象的概念

匿名对象是指创建对象时，只有创建对象的语句，却没有把对象地址值赋值给某个变量。

如：已经存在的类：

public class Person{

public void eat(){

System.out.println();

}

}

创建一个普通对象

Person p = new Person();

创建一个匿名对象

new Person();

## 匿名对象的特点

创建匿名对象直接使用，没有变量名。

new Person().eat() //eat方法被一个没有名字的Person对象调用了。

匿名对象在没有指定其引用变量时，只能使用一次。

new Person().eat(); 创建一个匿名对象，调用eat方法

new Person().eat(); 想再次调用eat方法，重新创建了一个匿名对象

匿名对象可以作为方法接收的参数、方法返回值使用

**class** Demo {

**public** **static** Person getPerson(){

//普通方式

//Person p = new Person();

//return p;

//匿名对象作为方法返回值

**return** **new** Person();

}

**public** **static** **void** method(Person p){}

}

**class** Test {

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

//调用getPerson方法，得到一个Person对象

Person person = Demo.*getPerson*();

//调用method方法

Demo.*method*(person);

//匿名对象作为方法接收的参数

Demo.*method*(**new** Person());

}

}

# 内部类

## 内部类概念

什么是内部类

将类写在其他类的内部，可以写在其他类的成员位置和局部位置，这时写在其他类内部的类就称为内部类。其他类也称为外部类。

什么时候使用内部类

在描述事物时，若一个事物内部还包含其他可能包含的事物，比如在描述汽车时，汽车中还包含这发动机，这时发动机就可以使用内部类来描述。

class 汽车 { //外部类

class 发动机 { //内部类

}

}

内部类的分类

内部类分为成员内部类与局部内部类。

我们定义内部类时，就是一个正常定义类的过程，同样包含各种修饰符、继承与实现关系等。在内部类中可以直接访问外部类的所有成员。

## 成员内部类

成员内部类，定义在外部类中的成员位置。与类中的成员变量相似，可通过外部类对象进行访问

* 定义格式

class 外部类 {

修饰符 class 内部类 {

//其他代码

}

}

* 访问方式

外部类名.内部类名 变量名 = new 外部类名().new 内部类名();

成员内部类代码演示

定义类

class Body {//外部类，身体

private boolean life= true; //生命状态

public class Heart { //内部类，心脏

public void jump() {

System.out.println("心脏噗通噗通的跳")

System.out.println("生命状态" + life); //访问外部类成员变量

}

}

}

访问内部类

public static void main(String[] args) {

//创建内部类对象

Body.Heart bh = new Body().new Heart();

//调用内部类中的方法

bh.jump();

}

## 局部内部类

局部内部类，定义在外部类方法中的局部位置。与访问方法中的局部变量相似，可通过调用方法进行访问

* 定义格式

class 外部类 {

修饰符 返回值类型 方法名(参数) {

class 内部类 {

//其他代码

}

}

}

* 访问方式

在外部类方法中，创建内部类对象，进行访问

局部内部类代码演示

定义类

class Party {//外部类，聚会

public void puffBall(){// 吹气球方法

class Ball {// 内部类，气球

public void puff(){

System.out.println("气球膨胀了");

}

}

//创建内部类对象，调用puff方法

new Ball().puff();

}

}

访问内部类

public static void main(String[] args) {

//创建外部类对象

Party p = new Party();

//调用外部类中的puffBall方法

p.puffBall();

}

## 内部类的实际使用——匿名内部类

### 匿名内部类概念

内部类是为了应对更为复杂的类间关系。查看源代码中会涉及到，而在日常业务中很难遇到，这里不做赘述。

最常用到的内部类就是匿名内部类，它是局部内部类的一种。

定义的匿名内部类有两个含义：

* + 临时定义某一指定类型的子类
  + 定义后即刻创建刚刚定义的这个子类的对象

### 定义匿名内部类的作用与格式

**作用：**匿名内部类是创建某个类型子类对象的快捷方式。

**格式：**

new 父类或接口(){

//进行方法重写

};

代码演示

//已经存在的父类：

public abstract class Person{

public abstract void eat();

}

//定义并创建该父类的子类对象，并用多态的方式赋值给父类引用变量

Person p = new Person(){

public void eat() {

System.out.println(“我吃了”);

}

};

//调用eat方法

p.eat();

使用匿名对象的方式，将定义子类与创建子类对象两个步骤由一个格式一次完成，。虽然是两个步骤，但是两个步骤是连在一起完成的。

匿名内部类如果不定义变量引用，则也是匿名对象。代码如下：

new Person(){

public void eat() {

System.out.println(“我吃了”);

}

}.eat();

# 包的声明与访问

## 包的概念

java的包，其实就是我们电脑系统中的文件夹，包里存放的是类文件。

当类文件很多的时候，通常我们会采用多个包进行存放管理他们，这种方式称为分包管理。

在项目中，我们将相同功能的类放到一个包中，方便管理。并且日常项目的分工也是以包作为边界。

类中声明的包必须与实际class文件所在的文件夹情况相一致，即类声明在a包下，则生成的.class文件必须在a文件夹下，否则，程序运行时会找不到类。

## 包的声明格式

通常使用公司网址反写，可以有多层包，包名采用全部小写字母，多层包之间用”.”连接

类中包的声明格式：

package 包名.包名.包名…;

如：黑马程序员网址itheima.com那么网址反写就为com.itheima

传智播客 itcast.cn 那么网址反写就为 cn.itcast

注意：声明包的语句，必须写在程序有效代码的第一行（注释不算）

代码演示：

package cn.itcast; //包的声明，必须在有效代码的第一行

import java.util.Scanner;

import java.util.Random;

public class Demo {}

## 包的访问

在访问类时，为了能够找到该类，必须使用含有包名的类全名（包名.类名）。

包名.包名….类名

如： java.util.Scanner

java.util.Random

cn.itcast.Demo

带有包的类，创建对象格式：包名.类名 变量名 = new包名.类名();

cn.itcast.Demo d = new cn.itcast.Demo();

前提：包的访问与访问权限密切相关，这里以一般情况来说，即类用public修饰的情况。

类的简化访问

当我们要使用一个类时，这个类与当前程序在同一个包中（即同一个文件夹中），或者这个类是java.lang包中的类时通常可以省略掉包名，直接使用该类。

如：cn.itcast包中有两个类，PersonTest类，与Person类。我们在PersonTest类中，访问Person类时，由于是同一个包下，访问时可以省略包名，即直接通过类名访问 Person。

类名 变量名 = new类名();

Person p = new Person();

当我们要使用的类，与当前程序不在同一个包中（即不同文件夹中），要访问的类必须用public修饰才可访问。

package cn.itcst02;

public class Person {}

## import导包

我们每次使用类时，都需要写很长的包名。很麻烦，我们可以通过import导包的方式来简化。

可以通过导包的方式使用该类，可以避免使用全类名编写（即，包类.类名）。

导包的格式：

import 包名.类名;

当程序导入指定的包后，使用类时，就可以简化了。演示如下

//导入包前的方式

//创建对象

java.util.Random r1 = new java.util.Random();

java.util.Random r2 = new java.util.Random();

java.util.Scanner sc1 = new java.util.Scanner(System.in);

java.util.Scanner sc2 = new java.util.Scanner(System.in);

//导入包后的方式

import java.util.Random;

import java.util.Scanner;

//创建对象

Random r1 = new Random();

Random r2 = new Random();

Scanner sc1 = new Scanner(System.in);

Scanner sc2 = new Scanner(System.in);

import导包代码书写的位置：在声明包package后，定义所有类class前，使用导包import包名.包名.类名;

# 访问修饰符

在Java中提供了四种访问权限，使用不同的访问权限时，被修饰的内容会有不同的访问权限，以下表来说明不同权限的访问能力：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | public | protected | default | private |
| 同一类中 | √ | √ | √ | √ |
| 同一包中(子类与无关类) | √ | √ | √ |  |
| 不同包的子类 | √ | √ |  |  |
| 不同包中的无关类 | √ |  |  |  |

归纳一下：在日常开发过程中，编写的类、方法、成员变量的访问

要想仅能在本类中访问使用private修饰；

要想本包中的类都可以访问不加修饰符即可；

要想本包中的类与其他包中的子类可以访问使用protected修饰

要想所有包中的所有类都可以访问使用public修饰。

注意：如果类用public修饰，则类名必须与文件名相同。一个文件中只能有一个public修饰的类。

# 代码块

## 局部代码块

局部代码块是定义在方法或语句中

特点：

以”{}”划定的代码区域，此时只需要关注作用域的不同即可

方法和类都是以代码块的方式划定边界的

class Demo{

public static void main(String[] args) {

{

int x = 1;

System.out.println("普通代码块" + x);

}

int x = 99;

System.out.println("代码块之外" + x);

}

}

结果：

普通代码块1

代码块之外99

## 构造代码块

构造代码块是定义在类中成员位置的代码块

特点：

优先于构造方法执行，构造代码块用于执行所有对象均需要的初始化动作

每创建一个对象均会执行一次构造代码块。

public class Person {

private String name;

private int age;

//构造代码块

**{**

**System.out.println("**构造代码块执行了**");**

**}**

Person(){

System.out.println("Person无参数的构造函数执行");

}

Person(int age){

this.age = age;

System.out.println("Person（age）参数的构造函数执行");

}

}

class PersonDemo{

public static void main(String[] args) {

Person p = new Person();

Person p1 = new Person(23);

}

}

## 静态代码块

静态代码块是定义在成员位置，使用static修饰的代码块。

特点：

它优先于主方法执行、优先于构造代码块执行，当以任意形式第一次使用到该类时执行。

该类不管创建多少对象，静态代码块只执行一次。

可用于给静态变量赋值，用来给类进行初始化。

public class Person {

private String name;

private int age;

//静态代码块

static{

System.out.println("静态代码块执行了");

}

}

# 总结

## 知识点总结

final：关键字，最终的意思

final修饰的类：最终的类，不能被继承

final修饰的变量： 相当于是一个常量, 在编译生产.class文件后，该变量变为常量值

final修饰的方法： 最终的方法，子类不能重写，可以继承过来使用

static : 关键字， 静态的意思

可以用来修饰类中的成员(成员变量，成员方法)

注意： 也可以用来修饰成员内部类

* + 特点：

被静态所修饰的成员，会被所有的对象所共享

被静态所修饰的成员，可以通过类名直接调用，方便

Person.country = "中国";

Person.method();

* + 注意事项：

静态的成员，随着类的加载而加载，优先于对象存在

在静态方法中，没有this关键字

静态方法中，只能调用静态的成员(静态成员变量，静态成员方法

匿名对象：一个没有名字的对象

* + 特点：

创建匿名对象直接使用，没有变量名

匿名对象在没有指定其引用变量时，只能使用一次

匿名对象可以作为方法接收的参数、方法返回值使用

内部类：在一个类中，定义了一个新类，这个新的类就是内部类

class A {//外部类

class B{// 内部类

}

}

* + 特点：

内部类可以直接访问外部类的成员，包含私有的成员

包的声明与访问

* + 类中包的声明格式：

package 包名.包名.包名…;

* + 带有包的类，创建对象格式：包名.类名 变量名 = new包名.类名();

cn.itcast.Demo d = new cn.itcast.Demo();

* + 导包的格式：

import 包名.类名;

权限修饰符

public : 公共的

protected: 受保护的

private : 私有的

public protected 默认的 private

在当前类中 Y Y Y Y

同一包中的其他类 Y Y Y

不同包中的子类 Y Y

不同包中的其他类 Y

代码块：

局部代码块：定义在方法中的，用来限制变量的作用范围

构造代码块：定义在类中方法外，用来给对象中的成员初始化赋值

静态代码块：定义在类中方法外，用来给类的静态成员初始化赋值