* + - * 1. final关键字概念与四种用法

final关键字代表最终、不可改变。

常见的四种用法：

1. 可以用来修饰一个类。
2. 可以用来修饰一个方法。
3. 可以用来修饰一个局部变量
4. 可以用来修饰一个成员变量

**第一种 final关键字用来修饰类**

/\*  
当final关键字用来修饰一个类的时候，格式：  
public final class 类名称 {  
 // ...  
}  
含义：当前这个类不能有任何的子类。（太监类：即没有子类）  
注意：一个类如果是final的，那么其中所有的成员方法都无法进行覆盖重写。（因为没有儿子）

注意：final修饰的类虽然没有子，但是有父类，可以对父类的方法进行覆盖重写。  
 \*/

父类：  
public final class MyClass /\*extends Object\*/{  
  
 public void method(){  
 System.out.println("方法执行");  
 }  
}

子类：(不能进行继承，因为父类用了final修饰)

public class MySubClass /\*extends MyClass\*/ {

}

**第二种 final关键字用于修饰成员方法**

/\*  
当final关键字用来修饰一个方法的时候，这个方法就是最终方法，也就是不能被覆盖重写。  
格式：  
修饰符 final 返回值类型 方法名称（参数列表）{  
 // 方法体  
}  
注意事项：  
对于类、方法来说，abstract 关键字和 final 关键字不能同时使用，因为矛盾。  
  
 \*/

父类：  
public abstract class Fu { // 因为下边定义了一个抽象方法，所有此处定义为抽象类。  
 public final void method(){  
 System.out.println("父类方法执行");  
 }  
 public abstract /\*final\*/ void methodAbs() ; // 这里的 abstract 与 final 关键字冲突  
}

子类：

public class Zi extends Fu{  
 //父类中这个方法用了 final 关键字，所以子类不能在进行覆盖重写。  
 /\* @Override  
 public void method() {  
 System.out.println("子类覆盖重写父类的方法");  
 }\*/  
  
 //父类中这个方法没有使用 final 关键字，所以子类可以进行覆盖重写。  
 @Override  
 public void methodAbs() {  
 }  
}

第三种 final关键字修饰局部变量

定义 Student 为一个标准类：（无参、全参、Get/Set）省略；

主函数：

public class DemoFinal {  
 public static void main(String[] args) {  
 // 定义一个num变量，数值可以进行改变。  
 int num =10;  
 System.out.println(num); // 10  
 num = 20;  
 System.out.println(num); // 20  
  
 // 一旦使用 final 进行修饰局部变量，那么这个变量就不能在进行改变。  
 // 一次赋值，众生不变。  
 final int num1 = 30; // num1 永远为30  
  
 final int num2;  
 num2 = 40; // 正确写法。永远为40 类似 const 修饰。  
  
 // 对于基本类型来说，不可变说的是变量当中的数据不可以改变。  
 // 对于引用类型来说，不可变说的是变量当中的地址值不可改变，而非内容。  
  
 Student stu1 = new Student("张");  
 System.out.println(stu1); // 这串代码输出的是此时 stu1 的地址值。  
 System.out.println(stu1.getName()); // 张  
  
 stu1 = new Student("程");  
 System.out.println(stu1); // 这个输出的也为地址值  
 System.out.print(stu1.getName()); // 程  
 // 注意：此时 stu1 的内容不仅可以改变，还有地址值也发生了改变  
  
 final Student stu2 = new Student("云");  
 // 错误写法！final 的引用类型变量，其中的地址不可以改变。  
 // stu2 =new Student("王");  
 System.out.print(stu2.getName()); // 云  
 stu2.setName("博");  
 System.out.print(stu2.getName()); // 博   
 }  
}

第四种 final关键字用于修饰成员变量

不论以后怎么使用这个 name ，其地址值永远不变，但是可以使用

标准类中不可以使用 全参、setName两个方法。只可以使用无参和 getName 方法。

public class Student {  
 private final String name= "张";  
 public Student() {  
 }  
 public String getName() {  
 return name;  
 }   
}

主函数：

public class zhu {  
 public static void main(String[] args) {  
  
 Student stu1 = new Student();  
 System.out.println(stu1.getName()); // 张  
 // 注意：此时使用 setName 也不可以改变其中的内容。  
 }  
}

* + 1. 四种权限修饰符

Java中有四种权限修饰符：

public > protected > (default) > private

用一个类（我自己） YES YES YES YES

同一个包（我邻居） YES YES YES NO

不同包子类（我儿子） YES YES NO NO

不同包非子类（陌生人） YES NO NO NO

注意事项：(default)并不是关键字：default“，而是根本不写，空白。

* + 1. 内部类的概念与分类

概念：

如果一个事物的内部包含另一个事物，那么这就是一个类内部包含另一个类。

例如：身体和心脏的关系，有如：汽车和发动机的关系。

分类：

1. 成员内部类
2. 局部内部类（包含匿名内部类）
   * 1. 成员内部类的定义

/\*  
成员内部类的定义格式：  
修饰符 class 外部类名称 {  
 修饰符 class 内部类名称 {  
 // ...  
 }  
 // ...  
}  
  
注意：内用外，随意访问，外用内，炫耀内部类对象。  
 \*/

package demo12;  
  
public class Body { // 外部类  
  
 public class Heart { // 成员内部类  
  
 // 内部类的方法  
 public void beat(){  
 System.out.println("心脏跳动，蹦蹦蹦！");  
 System.out.println("我叫" + name); // 正确写法  
 }  
 }  
 // 外部类的成员变量  
 private String name;  
  
 // 外部类的成员变量  
 public void methodBody() {  
 System.out.println("外部类的方法");  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

* + 1. 成员内部类的使用

/\*

如何使用成员内部类：两种方式：  
1. 间接方式：在外部类的方法当中，使用内部类；然后main只是调用外部类的方法。  
2. 直接方法：公式：  
类名称 对象名 = new 类名称();  
【 外部类名称.内部类名称 对象名 = new 外部类名称().new 内部类名称(); 】  
 \*/

类：

public class Body { // 外部类  
  
 public class Heart { // 成员内部类  
  
 // 内部类的方法  
 public void beat(){  
 System.out.println("心脏跳动，蹦蹦蹦！");  
 System.out.println("我叫" + name); // 正确写法  
 }  
 }  
 // 外部类的成员变量  
 final String name = "王";  
  
 // 外部类的成员变量  
 public void methodBody() {  
 System.out.println("外部类的方法");  
  
 // 使用成员内部类 格式 1  
 Heart heart = new Heart();  
 heart.beat();  
 // 匿名写法 格式 2  
 new Heart().beat();  
 }  
}

主函数：

public class DemoInterClass {  
 public static void main(String[] args) {  
 // new 新的对象  
 Body obj = new Body();  
 obj.methodBody();  
 // 匿名方式，使用匿名对象，省略了对象名称。

new Body().methodBody();  
 System.out.println("上边为间接方式");  
  
 // 按照公式写  
 Body.Heart heart = new Body().new Heart();  
 heart.beat();  
 }  
}

* + 1. 内部类的同名变量访问

类函数：

// 如果出现了重名现象，那么格式是： 外部类名称.this.外部类成员变量名  
public class Out {  
  
 int num = 10; // 外部类的成员变量  
  
 public class Inner /\*extends Object\*/ {  
  
 int num = 20 ; // 内部类的成员变量  
  
 public void methodInner(){  
 int num = 30; // 内部类方法的局部变量  
 System.out.println(num); // 局部变量，就近原则 30  
 System.out.println(this.num); // 内部类的成员变量 20  
 System.out.println(Out.this.num); // 外部类的成员变量  
 }  
 }  
}

主函数：

public class DemoInterClass{  
  
 public static void main(String[] args) {  
 // 直接访问  
 Out.Inner obj = new Out().new Inner();  
 obj.methodInner(); // 30 20 10  
 }  
}

* + 1. 局部内部类的定义

/\*  
如果一个类是定义在一个方法内部的，那么这就是一个局部内部类。  
“局部”，只是当前所属的方法才能使用，出了这个方法外面就不能用了。  
  
定义格式：  
修饰符 class 外部类名称 {  
 修饰符 返回子类型 外部类方法名称（参数列表）{  
 class 局部内部类名称 {  
 // ...  
 }  
 }  
}  
 \*/

类：

public class Out {  
  
 public void methodOut(){  
 class Inner { // 局部内部类  
 int num = 10;  
 public void methodInner(){  
 System.out.println(num); // 10;  
 }  
 }  
  
 Inner inner = new Inner();  
 inner.methodInner();  
 }  
  
}

主函数：

public class zhu {  
  
 public static void main(String[] args) {  
 Out obj = new Out();  
 obj.methodOut();  
 }  
}

* + 1. 局部内部类的final问题

/\*  
局部内部类：如果希望访问所在方法的局部变量，那么这个局部变量必须是【有效final的】  
  
备注：从Java8 + 开始，只是局部变量事实不变，那么final关键字可以省略。  
  
原因：  
1. new出来的对象在堆内存当中。  
2. 局部变量是跟着方法走的，在栈内存当中。  
3. 方法运行结束之后，立刻出栈，局部变量就会立刻消失。  
4. 但是new出来的对象会在堆当中持续存在，直到垃圾回收消失。  
 \*/

类：

public class MyOut {  
  
 public void methodOuter(){  
 final int num = 10;  
 // 从 Java 8+ 开始 可以这么写 int num = 10;  
 class MyInner{  
 public void methodInner(){  
 System.out.println(num);  
 }  
 }  
 }  
}

* + 1. 匿名内部类  
       如果接口的实现类（或者是父类的子类）只需要使用唯一一次。  
       那么这中情况下就可以省略掉该类的定义，而改为使用【匿名内部类】。  
         
       内名内部类的定义格式：  
       接口名称 对象名 = new 接口名称();  
        // 覆盖重写所有抽象方法  
       }; 注意：分号不要丢

接口：

public interface MyInterface {  
 // 定义一个抽象方法。  
 public abstract void method(); // public abstract 可以省略。  
}

接口的一个实现类：

public class MyInterfaceImpl implements MyInterface{  
 @Override  
 public void method() {  
 System.out.println("实现类覆盖重写了方法");  
 }  
}

主函数（实现这个方法）：

public class DemoMain {  
 public static void main(String[] args) {  
  
 // 方法 1 new抽象方法的实现类  
 MyInterface impl = new MyInterfaceImpl();  
 impl.method();  
  
 // 方法 2 使用多态方法  
 MyInterface obj = new MyInterfaceImpl();  
 obj.method();  
// MyInterface some = new MyInterface(); // 错误写法  
  
 // 方法3 使用匿名内部类  
 MyInterface obj1 = new MyInterface() {  
 @Override  
 public void method() {  
 System.out.println(":匿名内部类实现了方法");  
 }  
 }; // 注意：这个括号中的内容就是匿名内部类

obj1.method();  
 }  
}

* + 1. 匿名内部类的注意事项

对格式“new 接口名称() {...}” 进行解析：  
1. new 代表常见对象的动作。  
2. 接口名称就是匿名内部类需要实现哪个接口。  
3. {...} 这才是匿名内部类的内容  
  
另外注意：  
1. 匿名内部类在【创建对象】的时候只能使用一次。一个对只能new一次，如果定义另一个，还需在写一遍大括号。  
如果希望多次创建对象，而且类的内容一样的话，那么就必须使用单独定义的实现类。  
2. 匿名对象，在【调用方法】的时候，只能使用唯一一次。  
如果希望同一个对象，调用多次方法，那么必须给对象起个名字。  
3. 匿名内部类是省略了【实现类/子类名称】，但是匿名对象是省略了【对象名称】  
强调：匿名内部类和匿名对象不是一回事！！！

* + 1. 类作为成员变量类型

**定义一个武器的标准类：**

public class Weapon {  
 private String code; // 武器的代号。

// 省略了 全参、无参、Get / Set 方法。  
}

**定义一个英雄类：**

// 游戏当中的英雄角色类  
public class Hero {  
  
 private String name; // 英雄名字，Ctrl 点击 String，发现他就是一个类。  
 private int age; // 英雄的年龄  
 private Weapon weapon; // 英雄的武器

// 定义的一个输出格式

public void attack(){  
 System.out.println("年龄为" + age + "的" + name + "用" + weapon.getCode() + "攻击敌人。");

// 省略了 全参、无参、Get / Set 方法。  
}

}

**主函数：**

public class DemoMain {  
  
 public static void main(String[] args) {  
 // 创建一个英雄角色  
 Hero hero = new Hero();  
 // 为英雄起一个名字，并且设置年龄  
 hero.setName("盖伦");  
 hero.setAge(20);  
 // hero.setWeapon("多兰剑");  
 // 错误写法，这个类在JDK中没有声明，而 String 在JDK中存在，不用在new对象，直接使用即可。   
  
 // 创建一个武器对象  
 Weapon weapon = new Weapon("多兰剑");  
 // 为英雄配备武器  
 hero.setWeapon(weapon);

hero.attack(); // 年龄为20的盖伦用多兰剑攻击敌人。  
 }  
}

* + 1. 接口作为成员变量类型

接口：

public interface Skill {  
  
 void use(); // 释放技能的抽象方法  
}

定义一个接口的实现类：

public class SkillImpl implements Skill {  
 @Override  
 public void use() {  
 System.out.println("Biu~Biu~Biu");  
 }  
}

定义一个英雄类：

public class Hero {  
// 省略了 全参、无参、Get / Set 方法。  
 private String name; // 英雄的名字  
 private Skill skill; // 英雄的技能

// 定义一个打印方法

public void attack(){  
 System.out.println("我叫" + name + ",开始施放技能");  
 skill.use(); // 调用接口的抽象方法。  
 System.out.println("施放技能完成");  
 }

}

主函数：【重点】

public class DemoMain {  
  
 public static void main(String[] args) {  
 Hero hero = new Hero();  
 hero.setName("艾希"); // 设置英雄的名称。  
  
 // 设置英雄技能  
 // 方法 1 使用单独定义的实现类 （SkillImpl）  
 hero.setSkill(new SkillImpl());  
 hero.attack();  
  
 // 方法 2 使用匿名内部类  
 Skill skill = new Skill() {  
 @Override  
 public void use() {  
 System.out.println("Pia~Pia~Pia~");  
 }  
 };  
 hero.setSkill(skill);  
 hero.attack();  
  
 // 方法 3 同时使用匿名内部类和匿名对象。  
 hero.setSkill(new Skill() {  
 @Override  
 public void use() {  
 System.out.println("Biu~Pia~Biu~Pia~");  
 }  
 });  
 hero.attack();  
 }  
}

* + 1. 接口作为方法的参数或者返回值

import java.util.ArrayList;  
import java.util.List;  
  
/\*  
java.util.List 正是ArrayList所实现的接口。  
 \*/  
public class DemoInterface {  
  
 public static void main(String[] args) {  
 // 右边是接口名称、右边是实现类名称，这个是多态写法。  
 List<String> list = new ArrayList<>();  
  
 List<String> result = addNames(list);  
 for (int i = 0; i < result.size(); i++) {  
 System.out.println(result.get(i));  
 }  
 }  
  
 public static List<String> addNames(List<String> list){  
 list.add("程");  
 list.add("王");  
 return list;  
 }  
}