2 面向对象

1. ~~定义类的语法~~

~~包名：~~

~~class 类名 父类 接口名~~

~~{~~

~~成员变量；（一般是基本数据类型，也可以是引用类型）（属性）~~

~~构造方法；~~

~~成员方法；（行为）~~

~~}·~~

二．基本注意点

1. 类名是一个数据类型
2. 公共类（public）的名要和Java文件名一致

编译时，输入java文件名 运行时，输入java文件里面的主类名

1. ~~类是抽象，对象是类的实例化~~
2. 对象总在内存中 Cat cat2; 声明,未创建，没有开辟内存空间

Cat cat2 = new Cat(); 声明并且创建

1. 对象名和对象不同，对象名相当于一个地址，指向内存中的对象
2. 类有两种，实体类（用来描述一类事物）和工具类
3. 在类中的构造方法或构造代码块尽量不要写方法，可以在类中写方法，在构造方法中引用

~~三．类的成员方法~~

~~1.public static void main(参数列表)~~

~~2.参数列表：表示成员函数输入，从外界传入的变量和值~~

~~3.方法名重复：方法的重载~~

~~注意点：1.关于引用变量 引用同一个作用域的变量无需加类名，在一个类中引用另一个类的变量时要通过创建对象去访问（静态变量除外）~~

2.定义一个方法时，注意返回值类型和未知的参数（思考这个方法的作用）

~~四．类的构造方法~~

~~1.作用：在创建新对象时初始化属性~~

~~2.特点：~~~~可以在多个对象中使用~~

（1）方法名和类名相同

(2) 没有返回值（void也不能写）

~~(3) 创建该类的对象时，自动调用~~

3.注意：

（1）如果一个类没有显式写上一个构造方法，编译器为该类添加一个无参的构造函数

（2）无参构造函数的权限修饰符和类的权限修饰符一致

（3）一个类可以定义多个构造方法，创建对象时，根据对象后面的参数，选择类中的构造方法

~~4．this~~

~~注意：（1）this代表了所属函数的调用者对象~~

~~（2）this 属于一个对象，不属于一个类，在类定义中使用，不能在类外面使用~~

~~（3）~~~~如果在一个类方法中，访问了一个变量，该变量只存在成员变量的情况下，那么java编译器会在该变量前添加this关键字~~

~~this的应用场景：（1）在构造函数中使用~~

~~作用：~~~~存在同名的成员变量与局部变量时，在方法的内部访问的是局部变量（java采取的是就近原则的机制访问的），可以通过this指定访问成员变量~~

~~（2）在一个构造函数中会调用另外一个构造函数初始化对象~~

~~this(一个参数的构造方法中的那个参数)~~

~~作用：避免两个构造方法中重复的代码~~

~~注意：this 关键字调用其他的构造函数时，this语句必须放在构造函数的第一个语句~~

~~this关键字在构造函数中不能出现相互调用的情况~~

~~（3）在类中的方法中使用~~

~~作用：this代表引用该方法的对象，所以不需要在参数列表中添加该对象的参数~~

五．构造代码块

1.作用：给对应的对象统一进行初始化

~~2.注意：（1）构造代码块的大括弧必须位于成员位置上，在类之内，方法外~~

~~（2）赋值先后顺序~~

~~java编译器编译一个java源文件的时候，会把成员变量的声明语句提前至一个类的最前端~~

~~成员变量的初始化工作其实都在构造方法中完成的~~

~~构造代码块中的代码块会移至构造方法中，构造方法中的代码是最后执行的~~

~~成员变量的显式初始化和构造代码块的代码是按照当前代码的顺序执行的~~

3.执行时间：创建一个对象的时候

4.代码块的类别：（1）构造代码块

（2）局部代码块：缩短局部变量的生命周期，节省内存

（3）静态代码块：类定义之后，它的一些信息已经写进代码区了，静态区域块就会自动执行

~~六．static修饰符~~

~~应用：static修饰成员变量~~

~~静态变量或类变量~~

~~1.~~~~作用：可以在多个对象中使用~~

~~减少内存的使用，因为如果是实例对象，则每创建一个新对象，就会多开辟一个新的堆内存空间~~

~~2.注意：~~

~~（1）访问方式既可以通过对象访问，也可以通过类访问~~

~~（2）使用static修饰变量一定要考虑变量是否真的需要共享~~

~~静态的成员变量和非静态的成员变量的区别~~

1. ~~作用上的区别~~
2. ~~静态的成员变量的作用共享一个数据给所有的对象使用~~
3. ~~非静态的成员变量的作用是描述一类事物的公众属性~~
4. ~~数量与存储位置上的区别~~
5. ~~静态的成员变量是存储在方法区内存中的，而且只会存在一份数据~~
6. ~~非静态的成员变量是存储在堆内存中的，有n个对象就有n个数据~~
7. ~~生命周期的区别~~
8. ~~静态的成员变量数据是随着类的加载而存在，随着类文件的消失而消失~~
9. ~~非静态的成员数据是随着对象的创建而存在，随着对象被垃圾回收器回收而消失~~

~~应用：static修饰成员方法~~

~~静态成员方法~~

~~1.作用：如果一个函数没有直接访问到非静态的成员时，那么就可以使用static修饰，一般用于工具类型的方法~~

~~2注意~~

~~（1）非静态函数只能由对象调用，静态函数可以由类名或对象进行调用~~

~~（2）静态函数中不能直接（静态函数只要存在对象，那么也可以访问非静态的数据，只是不能直接访问而已）访问非静态成员（包括成员变量和成员方法）~~

~~（3）静态函数中不能出现this或者super关键字~~

~~原因：因为静态的函数是可以使用类名调用的，一旦使用类名调用这时候不存在对象~~

~~2.~~~~静态的数据的生命周期：~~

~~静态方法和非静态方法是同时存在在内存中的~~

~~静态代码块是在Student.class文件加载到内存的时候就马上执行的，不需要等到创建对象的时候才执行~~

~~静态的成员变量数据优先于对象的数据而存在的~~

~~类文件 先存在于硬盘 然后在要使用到的时候才会加载到内存中，而静态代码块是在类文件加载的时候执行的~~

~~类文件加载到内存中后，解剖类文件，提取出方法和静态变量，放在方法区中，静态变量放在静态数据共享区~~

~~当用到某一个类文件中的方法时，在栈内存里开辟该方法的内存，在该内存里存储变量名~~

~~创建一个对象，则在堆内存中开辟内存储存数据~~

~~类文件属于懒惰型，加载后直到该程序执行结束才卸载~~

七.单例设计模式

保证一个类在内存中只有一个对象

饿汉单例设计模式

步骤：

1. 私有化构造函数
2. 声明本类的引用类型变量，并且使用该变量指向本类对象
3. 提供一个公共的方法获取本类的对象

懒汉单例设计模式

步骤：

1. 私有化构造函数
2. 声明本类的引用类型对象，不创建本类的对象
3. 提供公共静态的方法获取本类的对象，获取之前先判断是否已经创建了本类的对象，如果已经创建了，那么直接返回对象即可，如果还没有创建，那么先创建再返回

推荐使用：饿汉单例设计模式 因为懒汉单例设计模式会存在线程安全问题，目前还不能保证一类在内存中只有一个对象

八．成员变量和局部变量的区别

1.定义的位置上区别：成员变量在方法之外，类之内

局部变量是定义在方法之内

2.作用上的区别：成员变量的作用是用于描述一类事物的公共属性的

局部变量的作用是提供一个变量给方法内部使用

3.生命周期的区别：成员变量随着对象的创建而存在，随着对象的消失而消失

局部变量在调用对应的方法时，执行到创建该变量的语句时存在，局部变量一旦出了自己的作用域（创建该变量上一个括号内）那么马上从内存中消失

4.初始值的区别：成员变量是有默认的初始值的（包括引用变量）

局部变量是没有默认的初始值的，必须要先初始化才能使用（使用的意思是访问该变量所赋的值）

九．匿名对象

1.定义：没有引用类型变量指向的对象称为匿名对象

2.注意：一般不给匿名对象赋予属性值，因为永远无法获取到(当使用匿名对象作用实参调用函数时，可以通过构造方法去赋予匿名方法属性) 5

两个匿名对象永远不可能是同一个对象

3.应用场景：

(1)如果一个对象要调用一个方法，而调用这个方法之后，该对象不再使用了，这个时候使用匿名对象(对象调用该类的中的方法)

( 2)可以作为一个实参去调用函数(对象调用别的类中的方法)

4.好处：

(1)简化书写

(2)可以尽快地释放内存空间（如果按原方式调用对象，则对象所占的内存需要在它的作用域中所有的程序执行完毕后才回收对象的内存）

十.封装

一般实体类的所有属性都要封装起来！

权限修饰符：权限修饰符 控制变量可见范围

public:公共的，public修饰的成员变量或方法 任何人都可以访问

private：私有的，private修饰的成员变量或者方法 只能在本类中进行访问

封装的步骤：

1. 使用private修饰要封装的属性
2. 提供一个公共的方法设置或者获取该私有的成员属性

命名规范: set属性名（）//限定用户输入的内容

get属性名（）

如果比较两个字符串的内容是否一致，不要使用==比较，使用equals方法

封装的好处：（1）提高数据的安全性

1. 操作简单
2. 隐藏了实现

~~十一.继承~~

~~1. class 类名1 extends 类名2~~

~~{~~

~~}~~

~~1为2的子类,2为1的父类~~

~~2.注意点：~~

1. ~~父类私有的成员不能被继承~~
2. ~~父类的构造函数不能被继承~~
3. ~~创建子类对象时，默认会先调用父类的无参构造函数，然后再调用子类的构造函数~~

~~指定调用父类的构造方法~~

~~作用：调用父类的构造方法可以初始化从父类继承下去的属性（即子类没有的属性）~~

~~3.super关键字：super关键字代表的是父类的引用空间~~

~~super关键字的作用：~~

1. ~~如果子父类存在同名的成员（变量和方法）时，在子类中默认是访问子类的成员。可以通过super关键字指定访问父类的成员。~~
2. ~~创建子类对象时，默认会先调用父类的无参构造函数，可以使用super关键字指定调用父类的构造函数 super(name);//指定调用父类一个参数的构造函数，必须出现在子类的构造方法里面~~

~~super注意点：~~

1. ~~如果在子类的构造函数中没有指定具体父类构造函数，那么java编译器会在子类的构造函数中添加super（）语句~~
2. ~~必须出现在子类的构造方法里面~~
3. ~~super关键字调用构造函数时，必须出现构造函数中第一个语句~~
4. ~~this与super调用构造函数的时候不能同时出现在一个构造函数中，因为都需要是第一个语句~~
5. ~~一定要在子类的构造函数中指向父类的一个构造函数（无参或有参）~~

~~super和this的区别：~~

~~1.代表的事物不一致：super代表的是父类的内存空间~~

4.方法重写：子父类存在着同名的函数，必须要有继承

方法重载：在一个类中，存在同名的两个或两个以上的同名函数

方法重写的需求：如果父类的功能无法满足子类的需求，那么就进行重写

方法重写的要素：

1. 函数名和形参列表必须一致
2. 子类的权限修饰符必须大于或等于父类的权限修饰符
3. 子类的返回值类型必须小于或者等于父类的返回值类型
4. 子类抛出的异常类型必须要小于或者等于父类抛出的异常类型

5.instanceof：判断一个对象是否属于某种类型

instanceof关键字用法：

对象 instanceof 类

instanceof的使用前提：判断的对象所属的类与要判断的类型必须存在继承或者实现的关系

对象和对象不可放一起

做强制转换之前都会使用该关键字判断，然后再进行转换

6.final关键字：最终的

final关键字用法：

1. final关键字修饰一个基本类型的变量时，该变量不能重新赋值，第一次的值为最终的（必须要赋初始值）

即常量，常量的修饰符一般为public static final

1. final关键字修饰一个引用类型变量时，该变量不能重新指向新的对象
2. final关键字修饰一个函数的时候，该函数不能被重写
3. final关键字修饰一个类时，该类不能被继承

方法里面的变量问题：

主函数中方法的实参传递给方法形参的是实参变量存储的值

不同方法的局部变量是相互独立的，没有任何的关系

~~十二.抽象类~~

~~在描述一类事物的时候，发现该种事物存在某种行为，但是这种行为目前是不具体的，那么我们可以抽取这种行为的声明，但是不去实现该种行为，这时候这种行为我们称作为抽象的行为，我们就需要使用抽象类~~

~~注意：~~

1. ~~如果一个函数没有方法体，那么该函数必须要使用abstract修饰，把该函数修饰成抽象的函数~~
2. ~~如果一个类出现了抽象的函数，该类也必须使用abstract修饰~~
3. ~~如果一个非抽象类继承了抽象类，那么必须要把抽象类的所有抽象方法实现~~
4. ~~抽象类可以存在非抽象方法，可以不存在抽象方法~~
5. ~~抽象类不能创建对象~~
6. ~~抽象类可以有构造方法~~
7. ~~abstract 不能与 private、static、final共同修饰一个方法（非静态的方法必须通过对象去调用，静态方法可以通过类调用，因为抽象类中的方法为非静态方法，所以必须通过对象去调用，抽象方法无法创建对象，所以必须通过子类的对象去调用（或者运用多态，声明父类的变量名，创建子类类型的对象））~~

~~作用：~~

1. ~~可以针对每种不同的子类，写出符合子类的方法~~
2. ~~强制子类一定要重写父类中的方法~~

~~十三.接口~~

~~作用：~~

1. ~~程序的解耦~~
2. ~~定义约束规范~~
3. ~~拓展功能~~

~~定义格式：~~

~~interface 接口名{~~

~~成员变量；~~

~~成员函数；~~

~~}~~

~~实现格式：~~

~~class a implements 接口名~~

~~注意：~~

1. ~~接口是一个特殊的类~~
2. ~~接口的成员变量都是常量，默认修饰符为public static final~~
3. ~~接口的成员方法都为抽象方法，没有方法主体~~
4. ~~接口不能创建对象~~
5. ~~没有构造方法（因为构造方法的作用是用来初始化变量的，接口都是常量）~~
6. ~~接口是给类去实现使用的，非抽象类实现一个接口的时候，必须要把接口中所有方法全部实现~~

~~类与接口：实现关系~~

1. ~~非抽象类实现一个接口时，实现全部的抽象方法~~
2. ~~抽象类实现一个接口时，可以实现也可以不实现接口中的方法~~
3. ~~一个类可以实现多个接口~~

~~接口与接口：继承关系~~

~~（1）一个接口可以继承多个接口（接口的方法是抽象的）~~

~~十四.多态~~

~~一个对象具备多种形态（父类的引用类型变量指向子类对象或者接口的引用类型变量指向接口实现类的对象）~~

~~1.注意：~~

1. ~~多态情况下，子父类存在同名的成员变量时，访问的是父类的成员变量~~
2. ~~多态情况下，子父类存在同名的非静态成员函数时，访问的是子类的成员函数~~
3. ~~多态情况下，子父类存在同名的静态函数时，访问的是父类的成员函数~~

~~总结：多态情况下，子父类存在同名的成~~~~员时，只有在同名的非静态函数时，访问子类的成员~~

1. ~~多态情况下，不能访问子类特有的成员~~

~~编译看左边，运行不一定看右边~~

~~编译看左边：java编译器在编译的时候，会检查引用类型变量所属的类是否具备指定的成员，~~

~~如果不具备马上编译报错。~~

~~2.多态的应用场景：~~

1. ~~方法的参数列表使用多态，一个方法可以用于不同的子类~~
2. ~~方法的返回值使用多态，返回更多的数据类型（注意接收的类型与返回值类型一致）~~

3.强制数据类型转换

格式： 小数据类型 变量名 = （小数据类型）大数据变量名

十五.内部类

1.一个类定义在一个类的内部

成员内部类 局部内部类

2.访问方法：

1. 在外部类中创建一个方法，声明创建内部类的对象和调用内部类的方法
2. 在主函数中直接访问 外部类.内部类 变量名 = new外部类（）.new内部类（）；

注意：如果是一个静态内部类，在其他类创建的格式是 外部类.内部类 变量名 = new 外部类.内部类（）

3.内部类的应用场景：在描述A事物的时候，发现A事物内部还存在另外一个比较复杂的事物B的时候，而且这个比较复杂的事物B还需要访问A事物的属性等数据，那么这个时候我们就可以使用内部类描述B事物

4.内部类的好处：内部类可以直接访问外部类的所有成员

5.内部类要注意的细节：

1. 如果外部类与内部类存在同名的成员变量时，在内部类中默认情况下访问内部类的成员变量
2. 如果外部类与内部类存在同名的成员变量时，在内部类中可以通过 外部类.this.成员变量名 访问外部类的变量 this相当于一个对象
3. 如果内部类私有，只能通过在外部类中创建方法进行访问
4. 成员内部类一旦出现了静态的成员，该类也必须使用static修饰

（原因：如果内部类不用static修饰，则违背了访问被static修饰的成员，无需创建对象的原则。因为如果不用static修饰该类，访问static修饰的变量无需创建对象，但是访问该内部类时，需要创建外部类的对象）

6.局部内部类的访问方法及其注意的细节

（1）局部内部类访问局部变量，该变量必须被final修饰

7.匿名内部类：没有类名的内部类

好处：简化书写

使用前提：必须存在继承或者实现关系才能使用（它本身没有类名，必须要借用他父类或接口的名字）

格式： new 父类类名（）{ 匿名内部类是父类的子类（没有类名）的对象

}.匿名内部类的方法()；

当父类和匿名内部类里有两个方法，第一个方法的返回值为父类名，return this

调用时，第一个方法（）.第二个方法（）；

利用多态

父类名 a = new 父类名()

{

};

方法名（）；

方法名（）；

子类里面有特有的方法，只能用局部内部类

其余可以用匿名内部类

匿名内部类一般用于实参

局部内部类是在方法中使用的

内部类：成员内部类和局部内部类

局部内部类：局部内部类和匿名内部类

如果子类有特有的方法，使用局部内部类

十六. 异常

1.Throwable常用的方法:

toString() 返回当前异常对象的完整类名+病态方法

getMessage() 返回的是创建Throwable传入的字符串信息

printStrackTrace() 打印异常的栈信息

当编译器检测到语句的异常，会创建一个对应的异常对象，并且会调用这个异常对象的printStackTrace的方法来处理

异常的处理：

方式一：捕获处理

格式：

try(

可能发生异常的代码；

)catch（捕获的异常类型 变量名）（

处理异常的代码

）

注意：

1. 如果try块中代码出了异常，经过了处理之后，那么try—catch块外面的代码可以正常执行
2. 如果try块中代码出了异常，不会继续执行try中剩余的语句
3. 一个try块可以捕获多种异常类型的，但是捕获的异常类型必须从小到大捕获，否则编译报错

方式二：抛出处理

throw throws

注意：

1. 一个方法的内部抛出一个编译时异常对象，必须要在方法上声明抛出
2. 如果调用了一个声明抛出编译时异常的方法，那么调用者必须要处理异常
3. 如果一个方法内部抛出一个异常对象，throw语句houmian的代码都不会再执行了
4. 在一种情况下，只能抛出一种类型的异常对象

比较throw和throws两个关键字

1. throw关键字用于方法内部，throws用于方法声明上的
2. throw关键字是用于方法内部抛出一个异常对象的，throws关键字是用于在方法sha那个声明抛出异常类型的
3. throw关键字后面只能有一个异常对象，throws后面一次可以抛出多个异常类型

如果需要通知到调用者代码出现问题，使用抛出处理

如果代码直接与用户打交道，使用捕获处理

运行时异常（可以通过代码去避免，可以没有throws，在主函数中也可以不处理）

编译时异常（必须使用throws，必须去处理）

2.finally块

使用前提：必须要存在try块才能使用

除了退出jvm，其他情况都执行

适合做资源释放的工作

try块的三种组合方式

十七.导包语句

1. 包的作用：
2. 解决类名重复产生冲突的问题
3. 便于版本软件的发布
4. 注意：
5. 只能有一个包语句，package 语句必须位于java文件中的第一个语句
6. 如果一个类加上包语句，那么该类的完整类名就是：包名.类名
7. 编译时输入：javac -d . Demo.java
8. 运行时输入：java 包名.Demo
9. 导包语句作用：简化书写
10. 导包语句格式：import 包名.类名；

import 包名.\*；

1. 导包语句：
2. 一个java文件中可以出现多句导包语句
3. \*对应任何类名，包名.\*不能对应子包

十八.权限修饰符

public protected default(缺省，不写的时候默认) private

同一个类

同一个包

子父类

不同包

在不同包中，只有public 与protected 可以访问，protected必须在继承关系下才能够访问

十九.模板模式

1. 写出一类相同事件的某一个事情的方法（非静态，final类型）
2. 抽取出该具体事件，在类中写成抽象方法，将该类该成抽象类
3. 子类继承该类
4. 在main函数中，创建主类对象，调用该模板方法
5. 重写父类的抽象方法(非静态方法)

二十.Object类

1. toString()

子类重写该方法,主函数中System.out.println(对象名);直接调用

1. equals（） 比较对象的地址是否一致
2. hashCode（） 返回对象的哈希码,理解成对象的内存地址