* 类和对象

1.什么是类？什么是对象？

1)现实世界是由很多很多对象组成的

基于对象抽出了类

2)对象:真实存在的单个的个体

类:类型/类别，代表一类个体

3)类中可以包含:

3.1)所有对象所共有的属性/特征(静)-----成员变量

3.2)所有对象所共有的行为(动)----------方法

4)一个类可以创建多个对象

同一类型的多个对象，结构相同，数据不同

5)类是对象的模板，对象是类的具体的实例

2.如何创建类？如何创建对象？如何访问成员？

(1)定义类的方法:

class 类名 {

返回值类型 方法名称(参数列表) {

方法体………

}

… … …

}

(2)使用new关键字创建对象

类定义完成后，可以使用new关键字来创建对象。

new运算的语法为：Cell c = new Cell();

new 类名（）；

此创建对象的过程也通常称为实例化。

通过引用来访问对象的成员变量，以及调用方法:

Cell c = new Cell();

c.row = 2;

c.col = 3; //访问成员变量

c.drop();

c.moveLeft(2);

String str = c.getCellInfo(); //调用方法

引用

数据类型 引用类型变量 指向 对象

Cell c = new Cell();

引用

数据类型 引用类型变量 指向 对象

Student zs = new Student();

(3)引用类型变量的赋值:

引用类型变量存储的是对象的地址信息，

对引用类型变量的赋值， 除了使用上面的new关键字以外，还可以有另外一种赋值方式，

即：相同类型的引用类型变量之间相互赋值。

需要注意的是：引用类型变量之间的赋值不会创建新的对象，但有可能会使两个以上的引用指向同一个对象

Emp e1 = new Emp();

Emp e2 = e1; //将e1的值（对象的地址信息）赋给e2，e2和e1指向相同的对象。

e1.name =“黄河大虾”;

e2.name = “白发馍女”;

System.out.println(e1.name);

3.引用类型之间画等号:

1)指向同一个对象

2)对其中一个引用的修改会影响另一个引用

eg:房子钥匙

基本类型之间画等号:

1)赋值

2)对其中一个变量的修改不会影响另一个变量

eg:身份证复印件

4.null:空，没有指向任何对象

若引用的值为null，则不能再进行任何操作了

若操作则NullPointerException空指针异常

* 构造方法

1.方法的签名:方法名+参数列表

2.方法的重载(Overload):

1)发生在一个类中，方法名称相同，参数列表不同

2)编译器在编译时会根据签名自动绑定调用的方法

3.构造方法:

1)给成员变量赋初值

2)与类同名，没有返回值类型

3)在创建对象时被自动调用

4)若自己不写构造方法，则编译器默认一个无参构造方法

若自己写了构造，则不再默认提供

5)构造方法可以重载

4.this:指代当前对象，哪个对象调方法指的就是哪个对象

只能用在方法中，方法中访问成员变量之前默认有个this.

this的用法:

1)this.成员变量名--------访问成员变量

2)this.方法名()----------调用方法(不常用)

3)this()-----------------调用构造方法

5.构造方法的语法：

访问修饰符 类名( ) {

//构造方法体

}

6. 通过构造方法初始化成员变量

Java语言中的构造方法常常用于实现对对象成员变量的初始化，如下代码展示了构造方法的使用。

class Cell {

int row ；

int col ;

public Cell (int row1 , int col1){

row = row1;//this.row=row

col = col1;//this.col=col

}

}

class TestCell {

public static void main(String args[ ] ){

Cell c1 = new Cell( 15 , 6 );

printCell(c1);

}

* 引用类型数组:

//声明Student数组stus，包含10个元素

//每个元素都是Student型，默认值为null

Student[] stus = new Student[10];

stus[0] = new Student("zhangsan",25,"廊坊");

1.引用类型数组:

1)Cell[] cells = new Cell[4];

cells[0] = new Cell(2,5);

cells[1] = new Cell(2,6);

cells[2] = new Cell(2,7);

cells[3] = new Cell(2,8);

2)Cell[] cells = new Cell[]{

new Cell(2,5),

new Cell(2,6),

new Cell(2,7),

new Cell(3,6)

};

3)int[][] arr = new int[3][];

arr[0] = new int[2];

arr[1] = new int[3];

arr[2] = new int[2];

arr[1][0] = 100; //给arr中第2个元素中的第1个元素赋值为100

4)int[][] arr = new int[3][4];

for(int i=0;i<arr.length;i++){ //行

for(int j=0;j<arr[i].length;j++){ //列

arr[i][j] = 100;

* 关于继承

1.继承:

1)作用:实现代码的复用、避免代码重复

2)通过extends来实现继承

3)父类/基类:所有子类所共有的属性和行为

子类/派生类:子类所特有的属性和行为

4)子类继承父类后，子类具有:父类的+子类的

5)一个父类可以有多个子类

一个子类只能继承一个父类----单一继承

6)继承具有传递性

子类可以重写(覆盖)继承自父类的方法，即方法名和参数列表与父类的方法相同；

子类利用重写修改父类的方法。当子类对象的重写方法被调用时（无论是通过子类

的引用调用还是通过父类的引用调用），运行的是子类的重写修改后的版本

7)java规定:构造子类之前必须先构造父类

若子类自己不调父类构造，则编译器默认在子类构造的第一句通过super()调父类无参构造

若子类自己调了父类构造，则不再默认提供

super()调父类构造必须位于子类构造的第一行

2.super:指代当前对象的父类对象

super的用法:

super.成员变量名------------访问父类的成员变量

super.方法名()--------------调用父类的方法

super()---------------------调用父类的构造方法

3.向上造型:

1)父类型的引用指向子类的对象

2)能点出来什么，看引用的类型

class Animal{ //动物

}

class Tiger extends Animal{ //老虎

}

//动物是动物

Animal o1 = new Animal();

//老虎是老虎

Tiger o2 = new Tiger();

//老虎是动物

Animal o3 = new Tiger(); //向上造型

//动物是老虎---------------语义不通

Tiger o4 = new Animal(); //编译错误

* 关于内存管理

1.内存管理:由JVM来管理-----了解

1)堆:

1.1)存储new出来的对象(包括实例变量)

1.2)成员变量的生命周期:

创建对象时存在堆中，对象被回收时一并消失

1.3)垃圾:没有任何引用指向的对象

垃圾回收器不定时到内存中清扫垃圾，

回收过程是透明的，不一定发现垃圾就立刻回收，

调用System.gc()可以建议JVM尽快调度GC来回收

1.4)内存泄漏:不再使用的内存没有被及时的回收

建议:不再使用的对象需及时将引用设置为null

2)栈:

2.1)存储正在调用中的方法中的所有局部变量(包括参数)

2.2)调用方法时，会在栈中为该方法分配一块对应的栈帧，

栈帧中存储方法中所有局部变量(包括参数)

方法调用完后，栈帧被清除，局部变量一并失效

2.3)局部变量的生命周期:

调用方法时存在栈中，方法完之后与栈帧一并消失

3)方法区:

3.1)存储.class字节码文件(包括方法)

3.2)方法只有一份，能过this来区分具体的调用对象

* 重写与重载

1.方法的重写(Override):

1)发生在父子类中，方法名称相同，参数列表相同，方法体不同

2)重写方法被调用时，看对象的类型

子类可以重写(覆盖)继承自父类的方法，即方法名和参数列表与父类的方法相同；

子类利用重写修改父类的方法。当子类对象的重写方法被调用时（无论是通过子类

的引用调用还是通过父类的引用调用），运行的是子类的重写修改后的版本

2.重写与重载的区别:-----常见面试题

1)重写(Override):

1.1)发生在父子类中，方法名相同，参数列表相同，方法体不同

1.2)重写遵循"运行期"绑定，看对象的类型来调用方法

2)重载(Overload):

2.1)发生在一个类中，方法名相同，参数列表不同，方法体不同

2.2)重载遵循"编译期"绑定，看引用的类型来绑定方法

* package:

1)避免类名冲突

2)包名可以有层次结构，同包中的类不能同名

3)类的完全限定名: 包名.类名

4)建议:包名所有字母都小写

import:

1)同包中的类可以直接访问，

不同包中的类不能直接访问，只有如下两种方式:

1.1)先import声明类，再访问类----建议

1.2)类的完全限定名-------太繁琐，不建议

* 访问控制修饰符:

1)public:公开的，任何类

2)private:私有的，本类

3)protected:受保护的，本类、子类、同包类

4）x默认的:什么也不写，本类、同包类

类的访问修饰: public和默认的

类中成员的访问修饰: 如上4种都可以

* static:静态的

1)静态变量:

1.1)由static修饰

1.2)属于类，存在方法区中，只有一份

1.3)常常通过类名.来访问

1.4)何时用:所有对象共享的数据(图片、音频、视频等)

2)静态方法:

2.1)由static修饰

2.2)属于类，存在方法区中，只有一份

2.3)常常通过类名.来访问

2.4)静态方法没有隐式的this传递，

静态方法中不能直接访问实例成员

2.5)何时用:方法的操作仅与参数相关而与对象无关

3)静态块:

3.1)由static修饰

3.2)类被加载期间自动执行，因类只被加载一次，

所以静态块也只执行一次

3.3)何时用:常常用于初始化静态资源(图片、音频、视频等)

static修饰的方法不需要针对某些对象进行操作，其运行结果仅仅与输入的参数有关。

调用时直接用类名引用。由于static在调用时没有当前的对象this，因此在static方

法中不能对非static成员(当前对象的属性和方法)进行访问，但是非static方法可以访问static成员。

class Moo{

int a; //对象点来访问

static int b; //类名为来访问

void show(){ //有this

System.out.println(this.a);

System.out.println(Moo.b);

}

static void test(){ //没有this

System.out.println(a); //编译错误

System.out.println(Moo.b);

}

}

* 抽象类/接口

1.final:最终的、不可变的------单独应用几率不高

1)修饰变量:变量不可被改变

2)修饰方法:方法不可被重写

3)修饰类:类不可被继承

final关键字修饰成员变量，意为初始化后不可改变。该成员变量必需在初始化时赋值，

对象一旦创建即不可改变。可以在声明时初始化或在构造方法中进行初始化。

2.static final常量: 应用率高

1)常量必须声明同时初始化

2)通过类名点来访问，不可被改变

3)建议:常量名所有字母都大写，多个单词用\_分隔

4)编译器在编译时将常量直接替换为具体的值，效率高

3.抽象方法:

1)由abstract修饰

2)只有方法的定义，没有方法的具体实现(连{}都没有)

4.抽象类:

1)由abstract修饰

2)包含抽象方法的类必须是抽象类

不包含抽象方法的类也可以声明为抽象类----我乐意

3)抽象类不能被实例化

4)抽象类是需要被继承的，子类:

4.1)重写所有抽象方法------常用

4.2)也声明为抽象类--------不常用

5)抽象类的意义:

5.1)封装子类共有的属性和行为-------代码复用

5.2)为所有子类提供一种统一的类型---向上造型

5.3)可以包含抽象方法，为所有子类提供了统一的入口

每个子类的实现不同，但入口是一致的

5.接口:

1)是一个标准、规范----制定方

遵守了这个标准就能干某件事---------API

2)是一种数据类型(引用类型)

3)由interface定义

只能包含常量和抽象方法

4)接口不能被实例化

5)接口是需要被实现/继承的，实现类/子类:

必须重写接口中的所有抽象方法

6)“实现接口”与“继承父类”不同，一个类可以实现多个接口，实现的接口直接用逗号分隔。

若又继承又实现时，应先继承后实现

7)接口可以继承接口

类和类---------------继承extends

接口和接口-----------继承extends

类和接口-------------实现implements

Inter2 o1 = new Aoo(); //向上造型(直接造型)

Inter1 o2 = new Aoo(); //向上造型(间接造型)

interface Inter1{

void show();

}

interface Inter2 extends Inter1{

void test();

}

class Aoo implements Inter2{

public void test(){}

public void show()

}

* 多态/内部类

1.多态:多种形态

1)意义:

1.1)同一个类型的引用指向不同的对象时，有不同的实现

-----行为的多态:cut()、run()、study()......

1.2)同一个对象被造型为不同的类型时，有不同的功能

-----对象的多态:我、水、你......

2)向上造型(自动类型转换):

2.1)父类型的引用指向子类的对象

2.2)能造型成为的类型有: 父类、所实现的接口

2.3)能点出来什么，看引用的类型

3)强制类型转换，成功的条件有两点:

3.1)引用所指向的对象，就是该类型

3.2)引用所指向的对象，实现了该接口

4)强转时若不符合如下两个条件，则发生ClassCastException类型转换异常

建议在强转之前先通过instanceof判断引用指向的对象是否是该类型

2.成员内部类:应用率不高

1)类中套类，外面的称为外部类，里面的称为内部类

2)内部类通常只服务于外部类，对外不具备可见性

3)内部类通常只在外部类中被创建

4)内部类中可以直接访问外部类的成员(包括私有的)

内部类中有个隐式的引用指向了创建它的外部类对象

语法: 外部类名.this.

一般情况下，内部类对象会在其所在的外部类对象中创建（构造方法或其他方法）;

内部类对象中会有一个隐式的引用指向创建它的外部类对象。 成员内部类, 必须使

用“外部类的实例.new” 来创建对象。 本题中，在外部类的内部来创建成员内部

类的对象可以省略this，即将 “this.new Point()” 省略为 “new Point()”

class Test{

Aoo o1 = new Aoo(); //正确

Boo o2 = new Boo(); //编译错误

}

class Aoo{ //外部类

private int num = 5;

void test(){

Boo o = new Boo(); //正确

}

class Boo{ //内部类

void show(){

System.out.println(num); //正确

System.out.println(Aoo.this.num); //正确

System.out.println(this.num); //编译错误

}

}

3.匿名内部类:应用率比较高

1)若想创建一个类的对象，并且对象只被创建一次，

此时该类不必命名，称之为匿名内部类对象

2)匿名内部类中访问外部的变量，该变量必须是final的

* 面向对象三大特征:

1.封装:

1)类:封装的是对象的属性和行为

2)方法:封装的是具体的业务逻辑功能

3)访问控制修饰符:封装的是访问的权限

2.继承:

1)作用:代码的复用

2)父类:所有子类共有的属性和行为

子类:子类所特有的属性和行为

3)子继承父后，子具有:父类+子类

4)单一继承、多接口实现，传递性

3.多态:

1)意义:行为的多态、对象的多态

2)向上造型、强制类型转换、instanceof

3)多态的表现形式:

3.1)重写:主要是依据对象

3.2)重载:主要是依据参数

* 设计规则:

1)所有子类共有的属性和行为，抽象到父类中

2)所有子类行为都一样-----设计为普通方法

所有子类行为不一样-----设计为抽象方法

3)部分子类共有的行为，抽象到接口中

符合既是也是原则时，使用接口

接口是对继承的单根性的扩展----多继承