一、对象内存管理(面试点)---百度整理内存的面试题

>.编译好的程序运行在JVM中

>.程序，无论是代码还是数据，都存储于内存(JVM)中

>.JVM暂时分为三个区域：栈、堆、方法区。

1.堆内存

- 对象存储在堆中

>.主要用于存储new关键字创建的对象

- 成员变量的生命周期

>.访问对象一般靠--引用

>.当一个对象没有任何引用指向时，被视为废弃对象，属于被回收的范围，成员变量也会被回收

>.从对象在堆中被创建到对象从堆中被回收

Person p = new Person();创建对象 p = null; 废弃被回收

- 垃圾回收机制

>.垃圾回收器(Garbage Collection,GC) 是JVM自带的线程(独自运行着的一条程序)，用于回收没有任何引用指向的对象。

>.java程序不用担心内存管理，因为垃圾回收器是自动进行回收的。

- java程序内存泄漏问题(面试)

>.不再使用的内存没有被及时回收。

>.严重的内存泄露会因为占用太多内存而导致程序崩溃。

>.GC机制判断对象是否被回收是该对象是否有引用指向，因此，不用的对象一般应将其值设置为null。

- System.gc()方法

>. GC并不一定一发现有对象无引用，就立即回收。

>. 当我们需要立即处理可以调用system.gc();

>. 当前的gc();方法只是建议马上调度GC线程回收资源，但是具体的实现策略取决于不同的JVM系统

2.非堆-栈

- 栈用于存放方法中的局部变量

>.存储程序运行时在方法中声明的所有局部变量。

>.int a = 0 , int b = 1;则a、b变量存储在栈中，且存储的是a、b的值

- 成员变量和局部变量

>.成员变量和局部变量的区别如下：

局部变量：

定义在方法中

没有默认值，必须自行设定初始值

方法被调用中，存在栈中，方法调用结束后，从栈中清楚。

成员变量：

定义在类中，方法之外；

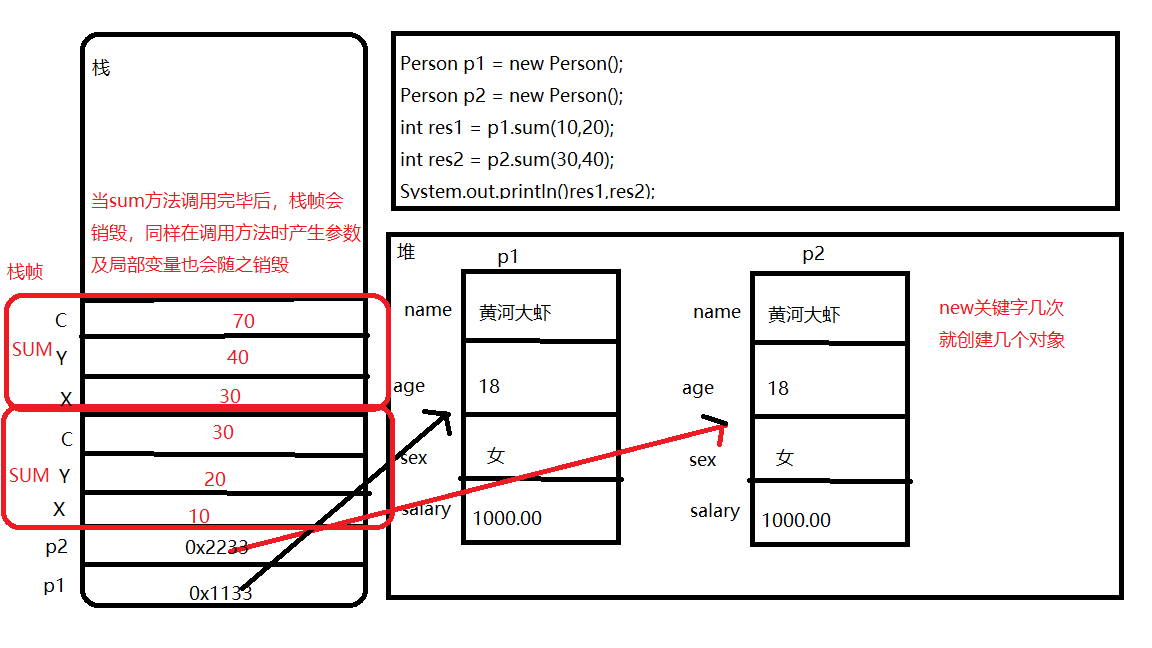
有默认的初始值，可以不显式的初始化

所在的类被实例化后，存在于堆中，对象被回收时，成员变量失效。

- 局部变量的生命周期

>一个运行着的程序,会有多次方法的调用,在每次调用时都会在栈中,开辟一块空间,这块空间叫栈帧。

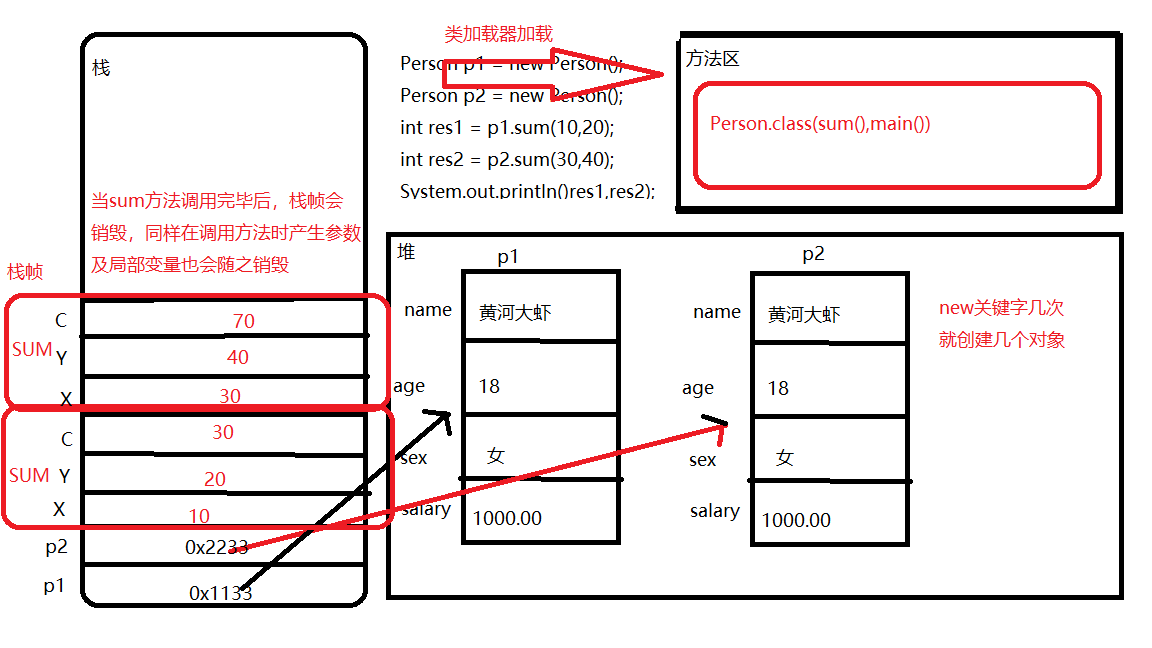
>.一个栈帧对应一个正在调用中的方法，栈帧中存储了该方法的参数、局部变量等数据,当某个方法调用完毕后，其对应的栈帧清除。局部变量失效。



3.非堆- 方法区

- 方法区用于存放类的信息

>.方法区用于存放类的信息(类的名称，路径，类的方法个数， 等等。)在java程序运行时，首先会经过类装载器(ClassLoader)载入文件的字节码信息，经过解析后再将其装入方法区，类的各种信息(包括方法)都在方法区中存储。



- 方法只有一份

>.当类的信加载到方法区时，除了类的基本信息外，同时还用类的方法。

>.类在实例化时，多个对象会拥有各自在堆中的空间，但所有实例对象共用一个方法区中的一份方法定义。

JFrame f1 = new JFrame();

JFrame f2 = new JFrame();

f1.setSize(200,300);

f2.setSize(300,400);//f1对象和f2对象共用一个setSize()方法

作业：

1. 新建图像类Shape类，定义周长和面积属性，定义计算周长及面积的方法
2. 新建子类圆类Cir类，计算圆面积周长并打印
   1. 新建子类正方形类Square类，计算正方形面积及周长打印

二、继承的意义

1.继承

-泛化的过程

>. 找一个类将公共属性进行封装，在派生子类中直接可以使用的过程。

- extends 关键字

>. 通过extends关键字可以实现类的继承。

Class A extends B {}

>. 子类(Sub Class)可以继承父类(Super Class)的所有属性和方法，并且可以 定义自身特有的属性和方法。

>. java语言不支持多继承，一个类只能有一个父类，但一个父类可以有多个子类。

--单一性

- 继承中的构造方法

>. 子类的构造方法中必须通过super关键字调用父类的构造方法，通过这种方法可以初始化继承自父类的属性。

>. 如果子类的构造方法没有调用父类的构造方法，java编译器会自动调用父类的无参构造方法(如果父类没有无参构造方法，必须有一个有参构造方法，否则会报错)

>. super语句必须放在构造方法的第一行

- 父类的引用指向子类对象(难点)

>. 一个子类对象可以向上造型为父类类型，即，父类引用指向子类对象

>. 父类类型引用只能调用父类的方法和属性，不能调用子类的方法和属性。

public static void main(String[] args) {

//子类引用指向子类对象，且子类型引用只能调用子类的属性和方法。

SubClass sc = new SubClass();

//父类引用指向了子类对象，且父类型引用只能调用父类的属性和方法

SuperClass sp = new SubClass();

sc.info();

sp.info();

}

2.重写

- 方法的重写

>. 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，但方法体(方法的实现不同)。

>. 当子类对象的重写方法被调用时(无论是子类的引用还是父类的引用)，都指向子类重写后的方法。

//子类引用指向子类对象

TestCirle c = new TestCirle();

//父类引用指向父类对象

TestSharp s = new TestSharp();

//父类对象指向子类对象

TestSharp s1 = new TestCirle();

c.zhouChang(); // 子

s.zhouChang();// 父

s1.zhouChang(); // 子---- 执行重写后的方法

- 重写中使用super关键字

>. 子类重写父类方法时，可以使用super. 父类方法名()这样的格式调用父类的方法。

- 重写和重载的区别(面试)

>. 重载：

方法名相同，参数列表不同构成重载，在编译时，根据参数而绑定调用的方法。

重写：

在子类中定义与父类完全相同(除修饰词外)的方法，在运行时，根据对象类型的不同而调用不同的版本。

1. 访问控制

1.包的概念

- package语句

>. 定义类时，需要指定类的名称，但如果仅仅将类名作为类的唯一标识，则会出现命名冲突的问题。

>. 在java语言中，采用包(package)的概念来解决命名冲突的问题

>. package 包名;

package com.tedu.oop.method; 在文件中是以(.)分割创建的多级文件目录。

>. 一个类的全类名格式为：包名 + 类名;

java.util.Scanner scan = new java.util.Scanner();

>. 包的命名遵循如下规则(程序员为合理解决命名冲突的问题而定义规则)：

com: 公司所属

tedu: 公司名称

oop: 项目名称

method: 项目模块

- import语句

>. 访问一个类时，需要使用类的全称，但这样的书写过于繁琐。

>. 因此，使用import 包名.类名; 这样的结构去声明包，则该源文件就可以直接写类名

2.访问控制修饰符

- 封装的意义

>. 对外提供可调用的，稳定的功能。

>. 意义：外界不能访问。--- 属性私有化，提供公共的set和get方法。

>. 封装降低了代码出错的可能性，便于维护。

>. 当内部的实现细节改变时，只要保证对外的功能定义不变，其他模块就不会受到牵连。

- public和private

>. private修饰的成员变量和方法仅仅只能在本类中调用，

>. public 修饰的成员变量和方法可以在任何地方调用，需要相对稳定。

- protected和默认的

>. protected修饰的可以在同包和子类中访问。

>. 默认的修饰符可以在同包下访问

- 访问控制符修饰类

>. 对于子类，可以使用public和默认方式。

>. 一个类中，只能有一个public 类。

>. public类必须与文件名称相同。

>. protected和private可以修饰内部类。

- 访问控制符修饰成员

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | 本类 | 同包中的类 | 子类 | 其他类 |
| public | 可以 | 可以 | 可以 | 可以 |
| protected | 可以 | 可以 | 可以 | 不可以 |
| 默认的 | 可以 | 可以 | 不可以 | 不可以 |
| private | 可以 | 不可以 | 不可以 | 不可以 |

1. static和final

1.static关键字

- static修饰成员变量

>. static修饰的变量属于类的基本信息，不属于对象的范畴。

>. static修饰的变量可以直接使用类名.成员的格式进行调用。

JFrame.EXIT\_ON\_CLOSE;

>. staric修饰的成员变量和类的信息一起存储在方法区，一个类static修饰的的成员变量

只有“一份”

- static修饰方法

>. 通常的方法都会涉及到具体对象的操作，这些方法在调用时，

需要隐式的传递对象的引用(this)。

计算两点之间的距离---int d = p1.distance(p2);

调用distance方法时，除了传递p2对象外，还隐式的传递了p1，而this就表示该参数。

>.static修饰的方法则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，调用时可直接使用 类名.方法 的格式。

Math.random(); //不需要隐式传递对象，所以在static 方法中不能使用this

>.在static 方法中不能使用this关键字。

>. static方法在调用时没有具体的对象，因此，在static方法中不能调用非static成员(属性和方法)。 static作用与一些”工厂方法”或”工具方法”等等。

- static块

>. 属于类的代码块，在类加载期间执行一次，且只执行一次，一般用来加载项目中的静态 资源。

总结:

1. 静态方法不能调用非静态成员。
2. 静态方法中不能使用this 关键字
3. 静态成员可以直接使用类名调用
4. 静态成员独一份，存储在方法区
5. 静态成员(静态代码块)都是在类加载期间加载，执行一次
6. 执行顺序：static块(只执行一次) > main > 非static块 > 构造方法

2.final 关键字

- final修饰成员变量

>. final关键字修饰成员变量，意为不可改变。

>. final修饰成员变量的时候，两种初始化方式：

- 声明时初始化

final int age = 18;

- 构造方法中初始化

final int age ;

public TestFinal() {

age = 16;

}

- final修饰方法

>. final修饰的方法不能被重写

>. 意义在于：防止“不经意”的重写

- final修饰类

>. final修饰的类不能被继承

>. 意义在于：保护类不能被继承，控制滥用继承对程序造成的危害。

String类不能被继承(面试) --- final修饰String类。

- static final变量(常用)

>. static final修饰的成员变量称为常量，声明时必须初始化，不能被改变。

public static final int AGE = 18; // (常量命名一般采用全大写，多个单词之间用下划线隔开)

public static final int EXIT\_ON\_CLOSE = 3;

>. 常量在编译时会被替换为实际的常量值

System.out.println(JFrame.EXIT\_ON\_CLOSE);

<--在编译时会被替换为实际值3 --> System.out.println(3);JFrame类不会被加载。

总结：1. final关键字修饰成员变量，意为不可改变

2. final修饰的方法不能被重写

3. final修饰的类不能被继承

1.20作业：

1. 新建Person类，定义四个私有属性，2 个公共方法，并在TestPerson测试访问属性和方法，且观察其保存信息。
2. 测试protected及默认的访问修饰符的访问权限范类(本类，子类，本包，异包)
3. 抽象类、接口、内部类

1.使用抽象类

- 抽象方法和抽象类

>. 抽象方法由abstract修饰的方法，且抽象方法只有方法的定义，没有方法的实现，即没有方法体，以分号(;)结尾。

>. 抽象类：由abstract修饰的类

>. 包含抽象方法的类，一定为抽象类，但抽象类不一定包含抽象方法。

>. 当一个非抽象类继承抽象类，就必须重写其定义的所有抽象方法。

- 抽象类不可以实例化

>. Shape s = new Shape(); 是错误的，抽象类不能被实例化

>. final和abstract不能同时修饰一个类，因为抽象类不继承将毫无意义。

- 继承抽象类

>.继承抽象类后必须实现其定义的所有抽象方法，不同子类有不同的实现。

- 抽象类的意义

>. 为其子类提供了一个公共的类型；

>. 封装子类中公共属性的方法；

>. 定义抽象方法，子类有不同的实现，但方法的定义是一定的

作业:

1. 根据周长计算圆，正方形、三角形的面积
2. 使用接口完成上面案例

2.使用接口(重)

注：JDK8之前，接口只能是抽象方法。

JDK8之后，静态方法可以直接定义方法体，可以直接用接口名调用，实现类和实现不可以调用。

public interface TestInterface {

int a = 1; //默认为public static final int a = 1

public static void A() {

System.out.println("A");

}

public static void main(String[] args) {

TestInterface.A();

}

}

- 定义一个接口

>. 接口时特殊的抽象类，只包含抽象方法的类。

>. 接口的关键字为interface 格式： interface 接口名 {}

- 实现接口

>. 与继承不同，一个类可以实现多个接口，实现的接口直接用逗号隔开。

>. 实现接口的类，叫实现类。格式为：class A implements B {}

>. 当实现类实现接口时，必须实现所有的接口中定义的抽象方法，同样，如果实现多个接口，则需要实现多个接口中所有的抽象方法。

>. 接口的类型引用可以指向实现类的对象

class A implements B{} B b = new A();

- 接口的继承

>. 接口之间可以有继承关系，一个接口可以通过extends继承另一个接口。

>. 实现类实现接口时，如果接口有继承关系，则实现类需要实现继承关系中所有的抽象方法

- 接口和抽象类的区别(面试)

>. 接口：

1. 接口的修饰词为interface。
2. 接口中的属性都为常量，默认为public static final修饰
3. 方法都为抽象方法，默认为 public abstract修饰
4. 在JDK1.8之后，可以有static修饰的方法，且有方法体，直接使用接口名调用。
5. 接口的实现需要依靠实现类，采用implements关键字实现。
6. 实现类实现接口时，应必须实现所有的抽象方法。
7. 接口可以继承接口，且实现时实现所有继承关系中的抽象方法。

>. 抽象类：

1. 抽象类的修饰词是abstract修饰的类。
2. 抽象类中可以有成员变量和方法。
3. 只要有一个方法为抽象方法，则这个类必须用abstract修饰。
4. 抽象类可以采用继承的方式实现，且子类必须重写继承自抽象类的所有抽象方法，除非子类也为抽象类，则不需重写。

3.多态

- 多态的意义

>. 一个类型的引用在指向不同的对象时会有不同的实现。

Shape s1 = new Circle(); //接口类型(父类)引用指向实现类(子类)对象

Shape s2 = new Square(); //接口类型(父类)引用指向实现类(子类)对象

>. 同一个对象，造型成不同类型时，会有不同的功能。

Shape s1 = new Circle(); //接口类型(父类)引用指向实现类(子类)对象

Circle c = new Circle(); // 子类引用指向子类对象

- 向上造型

>. 一个类对象可以向上造型的类型有：

父类类型

其定义的接口类型

- 向下造型

>. 其本质就是强制转换，就是将父类类型转换为子类类型，前提是该变量指向的对象确实为子类类型。

>. 也可以通过强制类型转换将接口类型转换为实现类 类型，前提是该变量指向的对象确实实现了该接口。

>. 如果在转换过程中违背了上述两个前提，则会抛出类型转换异常，ClassCastException。

- instanceof关键字

>. 在强制类型转换中，为了避免出现类型转换异常(ClassCastException),可以通过instanceof关键字判断某个引用指向的对象是否为当前类型。

if(s4 instanceof Square){

Square s6 = (Square)s4;

}

4.内部类

- 定义成员内部类

>. 定义在类内部的类。内部的类Inner，外部的类Outer

>. Inner只服务于Outer，对外不可见，Inner类中的方法可以直接调用Outer的属性和方法(包括私有的)。

- 创建内部类对象

>. 一般情况下，内部类对象会在外部类中创建(构造方法或其他方法);Inner对象中会有一个隐式的引用指向创建它的Outer类对象

- 定义匿名内部类

public class TestInner {

public static void main(String[] args) {

Shape1 s1 = new Shape1() { //匿名内部类

@Override

public void area() {

System.out.println("抽象类面积方法");

}

};

s1.area();

Shape2 s2 = new Shape2() {

@Override

public void area() {

System.out.println("接口面积方法");

}

};

s2.area();

}

}

面向对象项目案例-满天星

1. JFrame -- 窗体类 设计窗体的基本属性控制项目的启动
2. JPanerl -- 画板类 核心逻辑类主要用于绘制加载窗体中的画板
3. Factory -- 工具类 封装公共的属性及方法
4. 代码编写步骤：
5. 创建窗体类ManystarJFrame，继承JDK提供的JFrame
6. 构造方法设置窗体参数

- 设置标题

- 设置默认的关闭方式

- 设置位置及大小

- 将画板添加在窗体容器中

- 设置窗体可见

1. 定义main函数，实例化当前对象，项目启动
2. 创建Factory工具类

- 获取屏幕的宽--Toolkit

- 获取屏幕的高

- 设置窗体的宽

- 设置窗体的高

- 设置窗体的水平居中X坐标

- 设置窗体的垂直居中Y坐标

- 自定义随机颜色方法

- 自定义随机字体方法

- 自定义随机x及y坐标点方法

1. 创建类ManyStarJPanel,继承JPanel
2. 构造方法，初始化画板背景
3. 重写paint方法

- 设置画笔颜色

- 设置画笔字体

- 绘制