* 面向对象的理解并举例？

卖东西:

面向过程：到商店-->找商户-->找商品-->买商品

面向过程：找老妈-->老妈买商品

* 类与对象之间的关系？

类是一类具有相同特性事物的抽象集合，是对象的模版与蓝图。

* 如何对类进行分析，如果创建自定义类对象，并如何指挥对象做事情？

类具有公有的属性及公共的方法，如果创建自定义对象，可以创建该类公共的方法

* 对象的内存分布图？

创建对象时首先会在方法区加载该类class文件，解析该类具有的公有属性及公共行为，之后在栈内存中创建该类的引用类型变量，同时会在堆内存中创建该类的一个对象，栈内存中的引用类型变量指向堆内存中的对象。

* 成员变量和局部变量的区别？

|  |
| --- |
| 1.定义位置不同  成员变量：类之内，方法之外  局部变量：类之内，方法之内 |
| 2.作用范围不同  成员变量:在该类内部任何位置可以随意引用  局部变量:只在其方法定义的有效作用域内有效，一旦超出其作用域立即失效 |
| 3.生命周期不同  成员变量：随着对象的产生而产生，随着对象的消亡而消亡  局部变量：只在其方法定义的有效作用域内有效，一旦超出其作用域立即失效 |
| 4.初始值不同  成员变量：具有初始值，int 0，Boolean false  局部变量：没有初始值，必须初始化赋值才能使用 |
| 5.储存位置不同  成员变量:存储在堆内存中(对象存在的空间)  局部变量:存储在栈内存中 |

* 私有的使用。

private关键字的使用：一般情况下对于一个类中的公有属性来讲，只要是实体类就应该进行private封装，并按需求创建公有的访问器及修改器。而对于该类的方法来说，如果只是给该类内部进行使用可以使用private关键字进行封装。对于一个类来说，如果只是为同一个包中使用，可以使用private进行封装。

* 构造函数和一般函数的区别？

|  |
| --- |
| 1.构造函数没有返回值类型 |
| 2.构造函数的函数名与该类的类名相同 |
| 3.构造函数不是手动调用的，是创建对象值JVM主动调用的，每创建一个对象就会调用一次构造函数 |
| 4.构造函数可以通过重载的方式多个出现 |
| 5.当一个类没有创建造函数是编译器会自动给其添加一个无参的构造函数(继承自object类) |
| 6.当一个类显现创建一个构造方法时编译器不会个其添加无参的构造方法。 |

* 构造函数什么时候用？

创建对象时调用该类的构造方法，对成员变量进行初始化赋值。

* 构造代码块的作用？

对该类的对象进行统一的初始化赋值

* this关键字的特点和使用以及应用场景？

this关键字指向本类对象

|  |
| --- |
| 1.this关键字引用本类的构造方法时必须出现在第一句。 |
| 2.使用this关键字引用本类构造方法时要避免各自调用 |
| 3.this关键字代表本类对象，只能在非静态成员中使用 |
| 4.如果在一个函数中访问一个成员变量，而且没有存在同名的局部变量 时，那么 java编译器会默认在变量的前面加上 this关键字的。 |

* 静态和非静态的区别。说一下内存

|  |
| --- |
| 数量上的区别 |
| 存储位置上的区别 |
| 生命周期的区别 |
| 作用域与作用上的区别 |
| 访问方式上的区别 |

|  |
| --- |
| 调用方式：  静态成员：可以通过类名.静态成员及对象.静态成员访问  非静态成员：可以通过对象.非静态成员访问 |
| 访问限制：  静态：静态成员函数不能够直接使用非静态成员变量，能够直接使用静态成员变量  非静态：非静态成员函数能够直接使用非静态成员及静态成员。 |
| This，super关键字的使用：  静态：静态函数不能够使用this与super关键字  非静态：非静态函数能够调用this与super关键字 |

* 一个类文件加载到内存中，JVM会对该类文件进行解析,并将分析出该类文件具有哪些成员及哪些静态数据。 一个方法在运行时JVM会在栈内存中分配对应的内存空间

|  |
| --- |
| 1.当前内存由栈内存、堆内存、方法区(java总共有5片内存) |
| 2.类加载时先加载class文件进入共享区  其中该类的main方法加载进方法栈内存  .class 加载进方法区  堆内存开辟空间，实例变量进行默认初始化，显示初始化。  内存地址传给变量 p1，栈和堆建立连接 |
| 3.创建该类对象  1：堆内存开辟空间，实例变量进行默认初始化，显示初始化。  2：内存地址传给变量 p2，栈和堆建立连接 |
| 4.建立多个该类对象  每一个该类对象都可以访问Static中的静态变量 |

* 成员变量和静态变量的区别？

|  |
| --- |
| 访问方式：  静态成员变量：可以通过类名.静态成员及对象.静态成员访问  非静态成员变量：可以通过对象.非静态成员访问 |
| 数量差距：  静态成员变量：只会在方法区中存在一份，同类对象共同引用相同的静态成员变量  非静态成员变量：每个对象中都会存在一份自己的非静态成员变量 |
| 储存位置:  静态成员变量：在方法区中有且仅存在一份  非静态成员变量：在堆内存中的每一个对象空间中存在一份 |
| 作用区别：  静态成员变量：静态成员变量的作用是提供一个共享数据给所有对象使用的。  非静态成员变量：非静态成员变量的作用是用于描述一类事物的属性 |
| 生命周期：  静态成员变量：随着该类的加载而产生，随着类的消失而消失  非静态成员变量：随着对象的产生而产生，随着对象的消失而消失 |

* 静态的特点以及注意事项？
* 什么时候使用静态？

对于成员变量来说，如果该成员变量对于该类的所有对象来说只需要存在一份即可时(共享给该类使用对象共同使用)可以使用static进行修饰

对于成员函数来说，如果该成员函数没有直接使用该类的非静态成员变量时可以使用static进行修饰。

* 继承的好处？

|  |
| --- |
| 1.减少重复代码 |
| 2.描述类与类之间的相互关系 |
|  |

* java改良多继承的原因？

如果一个类能够继承多个父类会造成继承关系复杂，当一个类能够继承多个父类时构造函数的调用难以确定。

* 当使用一个已存在的继承体系时，该如何更快应用

直接使用子类，如果子类的方法不能够满足需求，可以对其非静态函数进行重写

* 什么时候用继承？

当类与类之间真实存在继承关系时使用继承

* super 和 this 的特点？

super指向父类的引用空间

super引用父类构造函数时需要出现在构造函数的第一句。

super引用父类构造函数时不能够与this关键字一起使用。

this与super关键字都不能与static共同使用，this代表该类对象

this关键字调用本类构造函数时需要出现在构造函数的第一句

this关键字调用本类构造函数时不能够与super共同使用

this关键调用本类构造函数时要避免重复调用

* 覆盖的特点，何时应用，注意事项？

当父类的方法不能够满足需求是，需要在子类中重写父类的方法

前提：类与类之间存在继承或者实现的关系

|  |
| --- |
| 1.函数名一致 |
| 2.形参列表一致 |
| 3.子类方法的返回值类型小与或者等于父类方法的返回值 |
| 4.子类方法的访问权限修饰符要大于或者等于父类方法的访问权限修饰符 |
| 5.子类的异常报错类型要小与父类方法的异常报错类型 |

* 子类的实例化过程？为什么是这样的实例化过程？

|  |
| --- |
| 1.子类class对象加载到方法区 |
| 2.栈内存创建子类的引用类型变量 |
| 3.堆内存创建子类对象的内存空间 |
| 4.父类class对象加载到方法区 |
| 5.解析父类有哪些成员变量及公共静态方法 |
| 6.子类使用super方法引用父类的空间 |

* super语句，和 this语句为什么不能同时存在，super为什么要定义在第一行？

因为this与super都要出现在构造函数的第一句，这要就相互矛盾了。

* 为什么要将一个类定义成内部类？

内部类能够减少代码量，简化书写

* 匿名内部类的使用和细节(面试题)

|  |
| --- |
| public void 方法(){  class Inner(){  内部类属性;  内部类方法;  }.内部类方法;  };  public void 方法(){  Father f =new Father(){  内部类属性;  内部类方法一;  内部类方法二;  };  f.内部类方法一;  f.内部类方法二;  };  public void 方法(){  new Father(){  内部类属性;  重写父类方法;  内部类方法(return this);  }.内部类方法.重写父类方法;  }; |

* 异常的思想和体系特点？

|throwable

|error(JVM或者硬件导致的错误，不能够进行处理只能够尽量避免)

|exception(可以通过java代码进行处理)

|运行时异常

|编译时异常

* throws和 throw的如何使用？

throws在方法声明使用，用于声明抛出异常

throw在方法中使用，用于抛出具体的异常

* 什么时候 try什么时候 throws？

需要捕获异常，对异常进行处理时使用try-catch关键字对异常进行捕获处理；如果不处理异常，需要将异常抛出是可以使用throws抛出异常。

* 编译时被检测异常和运行时异常的区别？

编译时异常，编译不能同过，体现在程序中就是由红色感叹号。

运行时异常，编译可以通过，程序没有红色感叹号，但是在运行时会有错误发生。

* 异常的所有细节？
* finally的应用？

try-catch捕获异常的最后步骤，通常用于关闭资源。

* 包的作用，名称空间的定义和理解？

|  |
| --- |
| 1 把功能相似或相关的类或接口组织在同一个包中，方便类的查找和使用。 |
| 2 如同文件夹一样，包也采用了树形目录的存储方式。同一个包中的类名字是不同的，不同的包中的类的名字是可以相同的，当同时调用两个不同包中相同类名的类时，应该加上包名加以区别。因此，包可以避免名字冲突。 |
| 3 包也限定了访问权限，拥有包访问权限的类才能访问某个包中的类。 |
|  |

* 在一个.java文件中可以一个public类和多个非public类，如果要将这些类组织在一个包当中，则在.java文件中除注释以外的第一行使用关键字package即可实现。当需要调用此包中的类时，就可以使用关键字import进行导入。在定义包的时候，应该注意几点:

|  |
| --- |
| 1 为了尽量使包名保持唯一性，包名通常采用小写、按倒写互联网址的形式进行定义。 |
| 2 在进行命名包时，应该避免使用与系统发生冲突的名字。 |

* jar包的基本使用

只要将类和包都存储到jar中，方便于使用。只要将jar配置到classpath路径下

* 抽象类的特点，以及细节？

|  |
| --- |
| 1. 如果一个方法没有方法体，那么该方法必须使用 abstract 修饰。 |
| 2. 如果一个类有抽象的方法，那么该类也必须使用 abstract 修饰。 |
| 3. 非抽象类继承抽象类的时候，必须要把抽象类中的所有抽象方法全部实现。 |
| 4. 抽象类可以定义抽象方法以及非抽象方法。 |
| 5. 抽象类是可以不存在抽象方法的。 |
| 6. 抽象类不能创建对象。为什么？疑问： 为什么抽象类不能创建对象呢？因为抽象类一旦创建了对象就可以使用对象去调用方法了，一旦调用了抽象方法就没有任何的意义了。java是不支持废话的。 |
| 7. 抽象类是存在构造方法的。疑问：既然抽象类不能创建对象，那么存在构造方法的意义在那？抽象类的构造方法是留给子类调用初始化从父类继续下去的成员变量的 |
| 8．抽象类不能创建对象，那么抽象类中是否有构造函数？抽象类中一定有构造函数。主要为了初始化抽象类中的属性。通常由子类实现。 |
| 9．final 和 abstract 是否可以同时修饰一个类？一定不能同时修饰。原因：final修饰的对象成员不能够被重写，重新赋值、继承，abstract修饰的对象，子类一定要实现父类的abstract修饰的方法。 |

|  |
| --- |
| 1. abstract 不能与 private 配合使用。private修饰的父类方法不能够被子类继承，只能够在本类中使用 |
| 2. abstract 不能与 final 关键字配合使用。final修饰的类不能够被继承  final:它的作用修饰类代表不可以继承 修饰方法不可重写  abstract 修饰类就是用来被继承的，修饰方法就是用来被重写的 |
| 3. abstract 不能与 static 关键字配合使用。 |

* 接口的表现形式的特点。

|  |
| --- |
| 接口的定义  interface 接口{  public abstract void 方法();  } |
| 接口的实现  class 实现类 implement 接口{  } |

|  |
| --- |
| 1.接口类没有构造方法 |
| 2.接口类并创建对象 |
| 3.接口类实现时需要使用implement关键字，一个类可以实现多个接口 |
| 4.接口中的常量属性默认是public static final，即所有常量都是静态常量，定义时必须进行初始化赋值 |
| 5.有抽象函数的类可以是抽象类也可以是抽象方法，抽象类不一定有抽象方法。 |
| 6.由于接口中的方法默认都是抽象的，所以不能被实例化。  static修饰的方法可以被子类继承，但是在继承时进行了隐藏，不能够被子类重新。  abstract修饰的方法需要被被子类重写以实现该方法。 |
| 7.对于接口而言，可以使用子类来实现接口中未被实现的功能函数。 |
| 8.如果实现类中要访问接口中的成员，不能使用super关键字。因为两者之间没有显示的继承关系，况且接口中的成员属性是静态的。可以使用接口名直接访问。 |

* 6接口的思想特点，要举例。

车具有空调的功能

* 多实现和多继承的区别？

多实现：实现多个接口

多继承：继承多个父类

* 抽象类和接口的区别？

抽象类是一类事物的抽象，是从类中抽取出来的

接口是功能方法的抽象

* 多态的体现，前提，好处，弊端。

|  |
| --- |
| 体现：父类的引用类型变量指向子类对象，父类引用也可以接受自己的子类对象 |
| 前提：必须存在继承或者实现关系 |
| 好处：提高扩展性，方法可以接收及返回更多的对象类型  1. 应用程序不必为每一个派生类编写功能调用，只需要对抽象基类进行处理即可。大大提高程序的可复用性。//继承  2. 派生类的功能可以被基类的方法或引用变量所调用，这叫向后兼容，可以提高可扩充性和可维护性。 //多态的真正作用 |
| 弊端：父类的引用类型变量不能够使用子类特有的方法，如果需要使用需要进行数据类型转化 |

* 我们现在知道创建对象的时候，jvm 需要调用到对应的构造方法，那么以前我们没有学构造方法，那么 jvm 还会调用到构造方法吗？而且构造方法存在吗？

存在构造方法的，但是不是 java 虚拟机添加的，是 java 编译器添加的。仅当类没有提供任何构造器的时候，系统才会提供一个默认的构造器

java 编译器添加的无参构造函数的修饰符是什么？无参的构造函数的权限修饰符是与类名一致的。

如何反编译：反编译可以使用 jdk 给我们提供的开发工具 javap

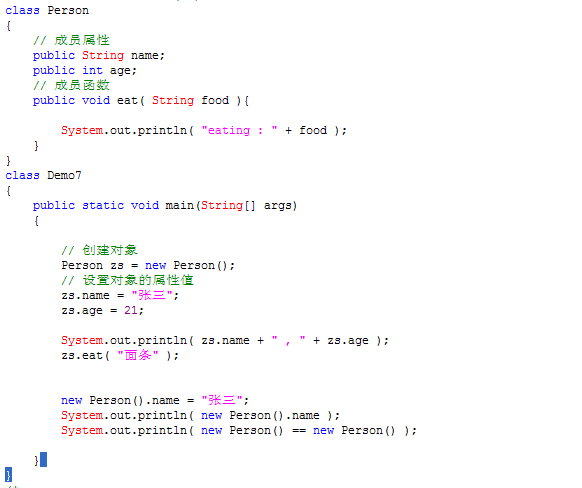
格式：javap -c -l -private class 文件

* 思考，如下两种写法，哪种可以通过编译？

|  |
| --- |
| // 写法1：  final int c = 0;  for (int i = 0; i < 10; i++) {  c = i;  System.out.println(c);  } |
| // 写法2：  for (int i = 0; i < 10; i++) {  final int c = i;  System.out.println(c);  } |

根据 final 的定义，写法1报错！写法2，为什么通过编译呢？因为每次循环，都是一次新的变量c。这也是大家需要注意的地方。

* 请问输出的name属性是什么? new Perosn == new Person()相等吗？



* 我要开车去丽江，这句话包含的类有什么？
* 体会以下几个经典案例涉及到的类。

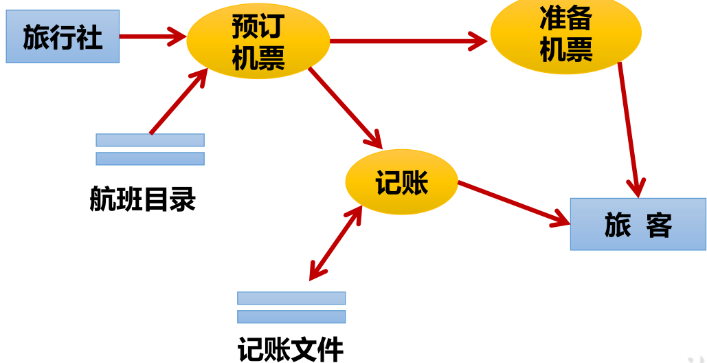
人在黑板上画圆

列车司机紧急刹车

售货员统计收获小票的金额

你把门关上了

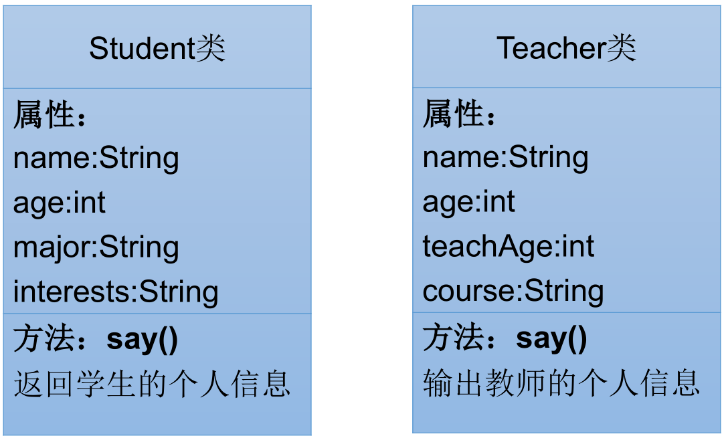
* 抽象出下面系统中的“类”及其关系。



* 练习：定义Person、Animal、ClassRoom、Zoo等类，加以体会。

|  |
| --- |
|  |
|  |
|  |
|  |

* 编写教师类和学生类，并通过测试类创建对象进行测试



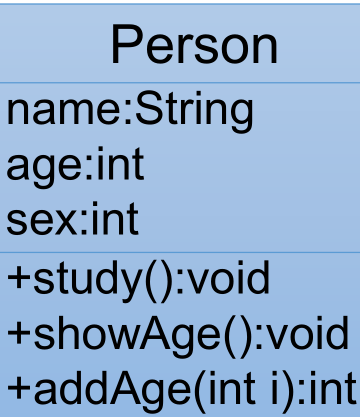
* 如何理解方法返回值类型为void 的情况 ?



* 创建一个Person类，其定义如下

要求：(1) 创建Person 类的对象，设置该对象的name、age 和sex 属性，调用study 方法，输出字符串“studying” ，调用showAge() 方法显示age 值，调用addAge() 方法给对象的age 属性值增加2 岁。

(2) 创建第二个对象，执行上述操作，体会同一个类的不同对象之间的关系。



|  |
| --- |
|  |

* 编写程序，声明一个method方法，在方法中打印一个10\*8 的\*型矩形，在main方法中调用该方法。

|  |
| --- |
|  |

修改上一个程序，在method方法中，除打印一个10\*8的\*型矩形外，再计算该矩形的面积，并将其作为方法返回值。在main方法中调用该方法，接收返回的面积值并打印。

|  |
| --- |
|  |

修改上一个程序，在method方法提供m和n两个参数，方法中打印一个m\*n的\*型矩形，并计算该矩形的面积， 将其作为方法返回值。在main方法中调用该方法，接收返回的面积值并打印。

|  |
| --- |
|  |

* 对象数组题目：定义类Student，包含三个属性：学号number(int)，年级state(int)，成绩score(int)。 创建20个学生对象，学号为1到20，年级和成绩都由随机数确定。

问题一：打印出3年级(state值为3）的学生信息。

问题二：使用冒泡排序按学生成绩排序，并遍历所有学生信息

提示：1) 生成随机数：Math.random()，返回值类型double;

2) 四舍五入取整：Math.round(double d)，返回值类型long。

|  |
| --- |
|  |

* 声明一个日期类型MyDate：有属性：年year,月month，日day。创建2个日期对象，分别赋值为：你的出生日期，你对象的出生日期，并显示信息。

|  |
| --- |
|  |

* 判断与void show(int a,char b,double c){}构成重载的有：

a) void show(int x,char y,double z){} // no

b) int show(int a,double c,char b){} // yes

c) void show(int a,double c,char b){} // yes

d) boolean show(int c,char b){} // yes

e) void show(double c){} // yes

f) double show(int x,char y,double z){} // no

g) void shows(){double c} // no

* 编写程序，定义三个重载方法并调用。方法名为mOL。

三个方法分别接收一个int参数、两个int参数、一个字符串参数。分别执行平方运算并输出结果，相乘并输出结果，输出字符串信息。

在主类的 main () 方法中分别用参数区别调用三个方法

|  |
| --- |
|  |

* 定义三个重载方法max()，第一个方法求两个int值中的最大值，第二个方法求两个double值中的最大值，第三个方法求三个double值中的最大值，并分别调用三个方法。

|  |
| --- |
|  |

* 请输出结果

|  |
| --- |
| class DataSwap {  public int a;  public int b;  } |
| public class TransferTest2 {  public static void swap(DataSwap ds) {  int temp = ds.a;  ds.a = ds.b;  ds.b = temp;  System.out.println("swap方法里，a Field的值是" + ds.a + ";b Field的值是" + ds.b);  }  public static void main(String[] args) {  DataSwap ds = new DataSwap();  ds.a = 5;  ds.b = 10;  swap(ds);  System.out.println("交换结束后，a Field的值是" + ds.a + ";b Field的值是" + ds.b);  }  } |



* 输出结果

|  |
| --- |
| package com.hfm.valuepass;  /\*\*  \* @author hfm  \* @version 1.01 2020-04-03 12:26  \* @date 2020/4/3  \*/  public class ObjectExchangeValueTest2 {  public static void main(String args[]) {  ObjectExchangeValueTest2 test = new ObjectExchangeValueTest2();  test.first();  }  public void first() {  int i = 5;  Value v = new Value();  v.i = 25;  second(v, i);  System.out.println(v.i);  }  public void second(Value v, int i) {  i = 0;  v.i = 20;  Value val = new Value();  v = val;  System.out.println(v.i + " " + i);  }  }  class Value {  int i = 15;  } |

* 定义一个int型的数组：int[] arr = new int[]{12,3,3,34,56,77,432};让数组的每个位置上的值去除以首位置的元素，得到的结果，作为该位置上的新值。遍历新的数组。

|  |
| --- |
|  |

* 判断

|  |
| --- |
| int[] arr = new int[10];  System.out.println(arr);//地址值?  char[] arr1 = new char[10];  System.out.println(arr1); //地址值? |

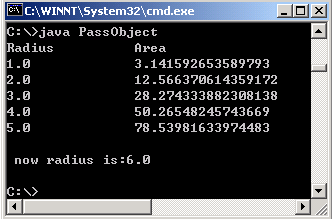
* 定义一个Circle类，包含一个double型的radius属性代表圆的半径，一个findArea()方法返回圆的面积。

|  |
| --- |
|  |

* 定义一个类PassObject，在类中定义一个方法printAreas()，该方法的定义如下：public void printAreas(Circle c, int time)在printAreas方法中打印输出1到time之间的每个整数半径值，以及对应的面积。例如，times为5，则输出半径1，2，3，4，5，以及对应的圆面积。

|  |
| --- |
|  |

* 在main方法中调用printAreas()方法，调用完毕后输出当前半径值。程序运行结果如图所示。



|  |
| --- |
|  |

* 请用Java写出递归求阶乘(n!)的算法

|  |
| --- |
|  |

* 已知有一个数列：f(0) = 1,f(1) = 4,f(n+2)=2\*f(n+1) + f(n),其中n是大于0的整数，求f(10)的值。

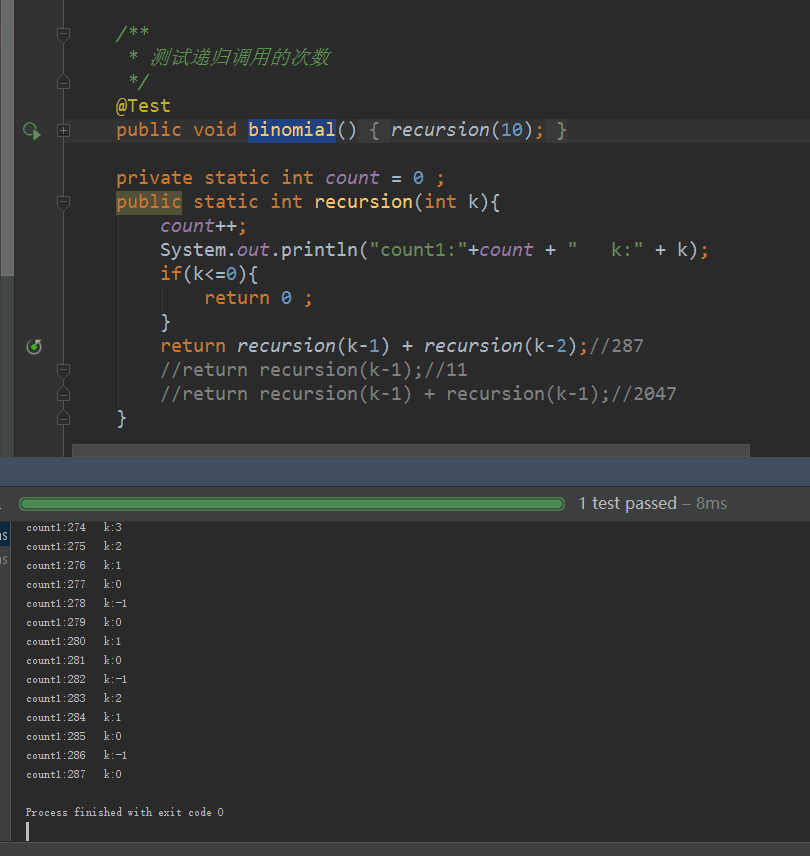
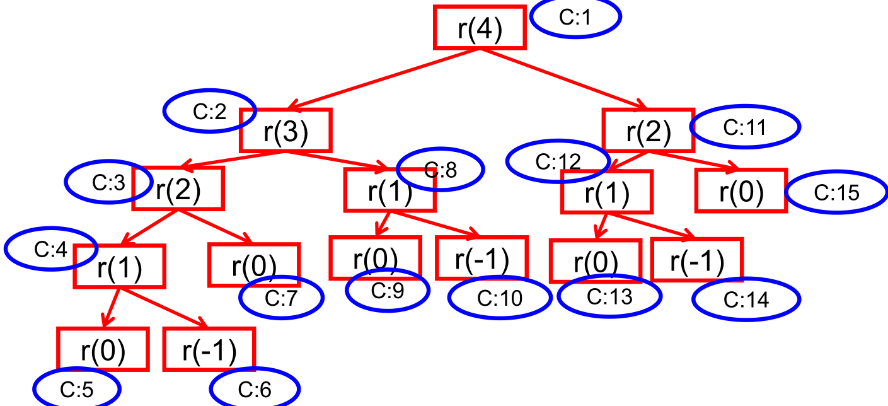
|  |
| --- |
|  |

* 已知一个数列：f(20) = 1,f(21) = 4,f(n+2) = 2\*f(n+1)+f(n),其中n是大于0的整数，求f(10)的值。

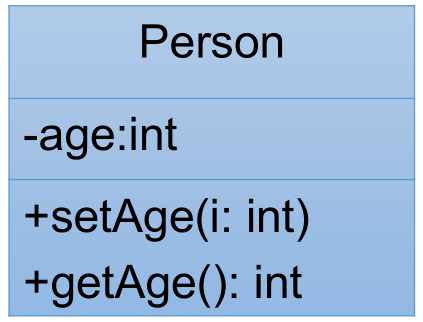
|  |
| --- |
|  |

* 输入一个数据n，计算斐波那契数列(Fibonacci)的第n个值，1 1 2 3 5 8 13 21 34 55，规律：一个数等于前两个数之和，要求：计算斐波那契数列(Fibonacci)的第n个值，并将整个数列打印出来

|  |
| --- |
|  |

* 创建程序,在其中定义两个类：Person和PersonTest类。定义如下：用setAge()设置人的合法年龄(0~130)，用getAge()返回人的年龄。在PersonTest 类中实例化 Person类 的 对 象 b， 调 用 setAge()和getAge()方法，体会Java的封装性。

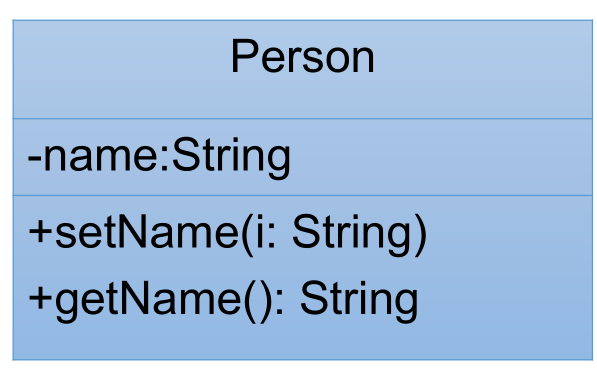


|  |
| --- |
|  |

在前面定义的Person类中添加构造器，利用构造器设置所有人的age属性初始值都为18

|  |
| --- |
|  |

* 修改上题中类和构造器，增加name属性,使得每次创建Person对象的同时初始化对象的age属性值和name属性值。



|  |
| --- |
|  |

* 编写两个类，TriAngle和TriAngleTest，其中TriAngle类中声明私有的底边长base和高height，同时声明公共方法访问私有变量。此外，提供类必要的构造器。另一个类中使用这些公共方法，计算三角形的面积。

|  |
| --- |
|  |

* 定义Student类,有4个属性：

String name;

int age;

String school;

String major;

|  |
| --- |
|  |

定义Student类的3个构造器:

第一个构造器Student(String n, int a)设置类的name和age属性；

第二个构造器Student(String n, int a, String s)设置类的name, age 和school属性；

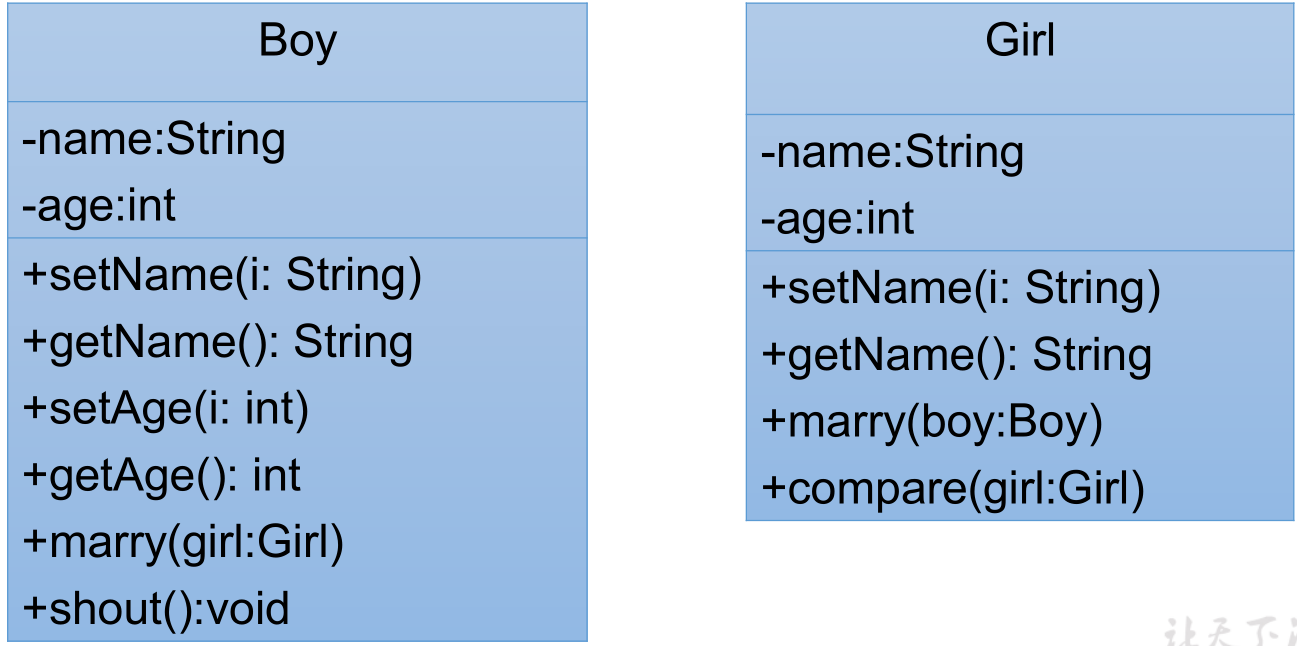
第三个构造器Student(String n, int a, String s, String m)设置类的name, age ,school和major属性；

|  |
| --- |
|  |

在main方法中分别调用不同的构造器创建的对象，并输出其属性值。

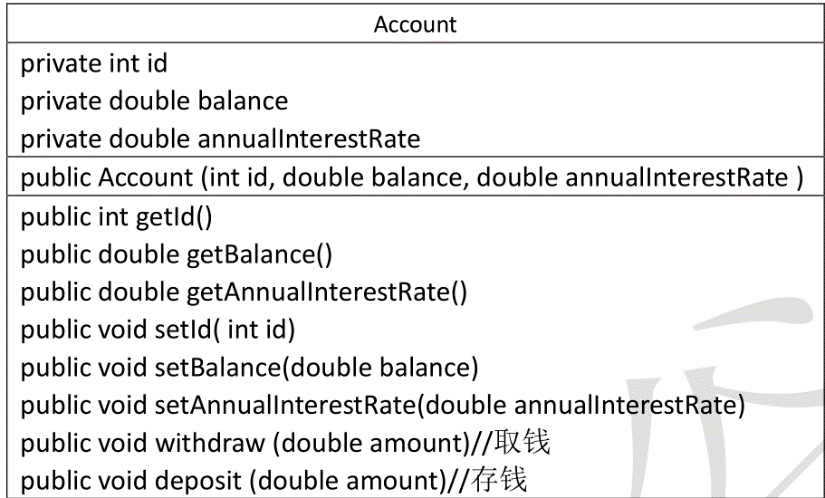
|  |
| --- |
|  |

* 添加必要的构造器，综合应用构造器的重载，this关键字。



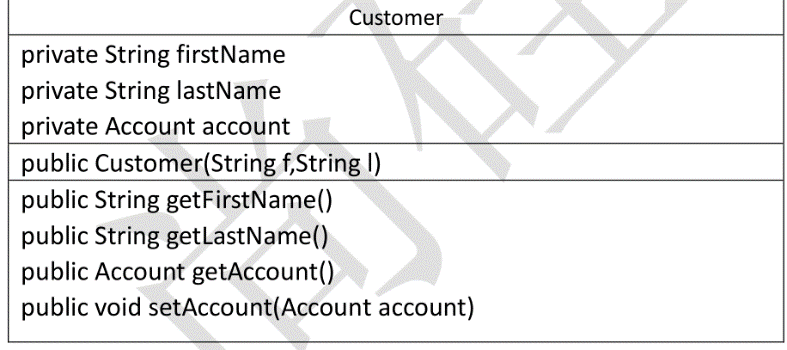
|  |
| --- |
|  |
|  |
|  |

* 写一个名为 Account 的类模拟账户。该类的属性和方法如下图所示。该类包括的属性：账号 id，余额 balance，年利率 annualInterestRate；包含的方法：访问器方法（getter 和 setter方法），取款方法 withdraw()，存款方法 deposit()。



提示：在提款方法 withdraw 中，需要判断用户余额是否能够满足提款数额的要求，如果不能，应给出提示。

* 创建 Customer 类。



a. 声明三个私有对象属性：firstName、lastName 和 account。

b. 声明一个公有构造器，这个构造器带有两个代表对象属性的参数（f 和 l）

c. 声明两个公有存取器来访问该对象属性，方法 getFirstName 和 getLastName 返回相应的属性。

d. 声明 setAccount 方法来对 account 属性赋值。

e. 声明 getAccount 方法以获取 account 属性。

|  |
| --- |
|  |

* 写一个测试程序。

（1） 创建一个 Customer ，名字叫 Jane Smith, 他有一个账号为 1000，余额为 2000 元，年利率为 1.23％ 的账户。

（2） 对 Jane Smith 操作。存入 100 元，再取出 960 元。再取出 2000 元。打印出 Jane Smith 的基本信息

成功存入 ：100.0

成功取出：960.0

余额不足，取款失败

Customer [Smith, Jane] has a account: id is 1000, annualInterestRate is 1.23％ , balance is1140.0

|  |
| --- |
|  |

* 编写一个Student类，包含name、gender、age、id、score属性，分别为String、String、int、int、double类型。

|  |
| --- |
|  |

类中声明一个say方法，返回String类型，方法返回信息中包含所有属性值。

|  |
| --- |
|  |

在另一个StudentTest类中的main方法中，创建Student对象，并访问say方法和所有属性，并将调用结果打印输出。

|  |
| --- |
|  |

* 定义一个丈夫Husband类，有姓名、年龄、妻子属性

|  |
| --- |
|  |

定义一个妻子Wife类，有姓名、年龄、丈夫属性

|  |
| --- |
|  |

丈夫类中有一个getInfo方法，其中，能显示自己的姓名，年龄，和他的妻子的姓名，年龄

|  |
| --- |
|  |

妻子类中有一个getInfo方法，其中，能显示自己的姓名，年龄，和她的丈夫的姓名，年龄

|  |
| --- |
|  |

定义一个测试类，创建妻子和丈夫对象，然后测试

|  |
| --- |
|  |

* 定义银行账户类Account，有属性：卡号cid，余额balance，所属用户Customer 银行账户类Account有方法：

（1）getInfo()，返回String类型，返回卡的详细信息

（2）取钱方法withdraw()，参数自行设计，如果取钱成功返回true，失败返回false

1. 存钱方法save()，参数自行设计，如果存钱成功返回true，失败返回false

其中Customer类有姓名、身份证号、联系电话、家庭地址等属性，Customer类有方法say()，返回String类型，返回他的个人信息。在测试类Bank中创建银行账户类对象和用户类对象，并设置信息，与显示信息

|  |
| --- |
|  |
|  |

* 哪个选项和show()方法重载

|  |
| --- |
| class Demo{  void show(int a,int b,float c){}  } |
| A.void show(int a,float c,int b){}//yes |
| B,void show(int a,int b,float c){}//一模一样。不可以出现在同一个类中。 |
| C.int show(int a,float c,int b){return a;}//yes。 |
| D.int show(int a,float c){return a;}//yes |

* 方法的声明与调用

（1）声明一个圆柱体类型，

（2）声明属性：底边的半径，和高

（3）声明方法：

A：方法的功能：在方法中打印圆柱体的详细信息，圆柱体的底边的半径是xxx，高是xxx，底面积是xxx，体积是xxx。

B：方法的功能：返回底面积

C：方法的功能：返回体积

D：方法的功能：为圆柱体的底边的半径，和高赋值

E：方法的功能：为圆柱体的底边的半径，和高赋值，并返回赋值的结果，如果底边的半径或高为<=0，赋值失败，返回false，否则返回true

1. 并测试

|  |
| --- |
| //声明圆柱体  class Cylinder{  double radius;//底边半径  double height;//高  /\*  A：方法的功能：在方法中打印圆柱体的详细信息  圆柱体的底边的半径是xxx，高是xxx，底面积是xxx，体积是xxx。  \*/  void printDetails(){  //double area = Math.PI \* radius \* radius;//底面积  //double volume = area \* height;//体积  //System.out.println("圆柱体的底边的半径是" + radius +" ，高是" + height + "，底面积是"+ area +"，体积是"+volume +"。");  //调用本类的方法  System.out.println("圆柱体的底边的半径是" + radius +" ，高是" + height + "，底面积是"+ getArea() +"，体积是"+getVolume() +"。")  }  //B：方法的功能：返回底面积  double getArea(){  double area = Math.PI \* radius \* radius;//底面积  return area;  }  //C：方法的功能：返回体积  double getVolume(){  //double area = Math.PI \* radius \* radius;//底面积  //double volume = area \* height;//体积  //return volume;  double volume = getArea() \* height;//体积  return volume;  }  //D：方法的功能：为圆柱体的底边的半径，和高赋值  void setValue(double r, double h){  radius = r;  height = h;  }  /\*  E：方法的功能：为圆柱体的底边的半径，和高赋值，并返回赋值的结果  如果底边的半径或高为<=0，赋值失败，返回false，否则返回true  \*/  boolean setRadiusAndHeight(double r, double h){  if(r<=0 || h<=0){  return false;  }  //radius = r;  //height = h;  setValue(r,h);  return true;  }  } |
| class TestMethodExer{  public static void main(String[] args){  //1、创建对象  Cylinder c = new Cylinder();  //c.radius = 2.0;  //c.height = 2;  c.setValue(2.0,2);  c.printDetails();  System.out.println("底面积： " + c.getArea());  System.out.println("体积： " + c.getVolume());  boolean flag = c.setRadiusAndHeight(3.0, 5);  if(!flag){// 如果flag = false, !flag结果就是true,条件成立  System.out.println("赋值失败");  }else{  c.printDetails();  }  }  } |

* 方法重载(overload)必须满足\_\_\_\_\_\_\_\_

A. 在不同class中定义的方法 B.在同一类型中定义的方法

C. 方法名必须相同 D.返回类型必须相同

E. 参数一定不同 F.参数可以相同

答案：BCE

* 写出输出结果

|  |
| --- |
| class Demo{  public static void main(String[] args){  show(0);  show(1);  }  public static void show(int i){  switch(i){  default:  i+=2;  case 1:  i+=1;  case 4:  i+=8;  case 2:  i+=4;  }  System.out.println("i="+i);  }  } |

* 写出输出结果

|  |
| --- |
| class Demo{  public static void main(String[] args){  int x = 1;  for(show('a'); show('b') && x<3; show('c')){  show('d');  x++;  }  }  public static boolean show(char ch){  System.out.print(ch);  return true;  }  }  //答案：abdcbdcb |

* 以下代码的运行结果是什么？

|  |
| --- |
| **public** **class** Test1 {  **public** **static** **boolean** foo(**char** c) {  System.***out***.print(c);  **return** **true**;  }  **public** **static** **void** main(String[] args) {  **int** i = 0;  **for** (*foo*('A'); *foo*('B') && (i < 2); *foo*('C')) {  i++;// 1 2  *foo*('D');  }  }  } |
| 答案：ABDCBDCB |

* 面向对象三大特征的说明

答：面向对象有三大特点：封装、继承、多态。(如果要回答四个，可加上 抽象性 这一特点)

1.继承性：继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。

2.封装性：封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封装的对象，这些对象通过一个受保护的接口访问其他对象。

3. 多态性：多态性是指允许不同类的对象对同一消息作出响应。多态性包括参数化多态性和包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好的解决了应用程序函数同名问题。

4.抽象性：抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象，二是数据抽象。

* 作用域public,private,protected,以及默认不写时的区别
* 找错

|  |
| --- |
| public class Something {  void doSomething () {  private String s = "";  int l = s.length();  }  } |

有错吗?

答案: 错。局部变量前不能放置任何访问修饰符 (private，public，和protected)。

* Java的内存管理之垃圾回收

分配：由JVM自动为其分配相应的内存空间

释放：由JVM提供垃圾回收机制自动的释放内存空间

垃圾回收机制（GC：Garbage Collection）：将垃圾对象所占用的堆内存进行回收。Java的垃圾回收机制是JVM提供的能力，由单独的系统级垃圾回收线程在空闲时间以不定时的方式动态回收。

垃圾对象：不再被任何引用指向的对象。

* 在程序中是否可以通知垃圾回收机制过来回收垃圾？

能，通过调用System.gc();或Runtime.getRuntime().gc();

* 调用了System.gc();或Runtime.getRuntime().gc();后是立刻执行垃圾回收吗？

不是，该调用并不会立刻启动垃圾回收机制开始回收，但会加快垃圾回收机制的运行。

|  |
| --- |
| public class TestGC{  public static void main(String[] args)throws Exception{  for(int i=0; i<10; i++){  MyClass m = new MyClass();//这里本次循环完，本次创建的对象就成为垃圾了  System.out.println("创建第" + (i+1) + "的对象：" + m);  }    //通知垃圾回收机制来收集垃圾  System.gc();    //为了延缓程序结束  for(int i=0; i<10; i++){  Thread.sleep(1);  System.out.println("程序在继续....");  }  }  }  class MyClass{  //这个方法是垃圾回收机制在回收它的对象时，自动调用，理解成对象留临终遗言的方法  public void finalize(){  System.out.println("轻轻的我走了.....");  }  } |

* 构造器Constructor是否可被override

构造器Constructor不能被继承，因此不能重写Override，但可以被重载Overload

* 编程创建一个Box类，在其中定义三个变量表示一个立方体的长、宽和高，定义一个方法求立方体的体积。创建一个对象，求给定尺寸的立方体的体积。（提供无参的构造器和一个有参的构造器）

|  |
| --- |
|  |

* 定义一个圆类型

提供显示圆周长功能的方法

提供显示圆面积的方法

提供无参的构造器和一个有参的构造器

|  |
| --- |
|  |

* 设计一个Dog类，有名字、颜色和年龄属性，定义构造器初始化这些属性，定义输出方法show()显示其信息。提供无参的构造器和一个有参的构造器

|  |
| --- |
|  |

* 定义一个类，用于描述坐标点

0——————>X

|

|

| P(X,Y)

|

|

Y

（1）具有计算当前点到原点距离的功能

（2）求到任意一点（m，n）的距离

（3）求到任意一点（Point p）的距离

（4）具有坐标点显示功能，显示格式（x，y）

（5）提供无参的构造器和一个有参的构造器

|  |
| --- |
|  |

* 写一个人的类

属性：名字，性别，年龄；提供无参的构造器和一个有参的构造器

方法：（1）自我介绍的方法（2）吃饭的方法

创建一个对象“张三”

|  |
| --- |
|  |

* 写一个汽车类

属性：品牌；车长；颜色；价格；

创建五个对象：“捷达”，“宝马”，“劳斯莱斯”，“科鲁兹”，“迈锐宝”

提供无参的构造器和一个有参的构造器

|  |
| --- |
|  |

* 写一个课程类

属性：课程名；学时；任课老师；

创建五个对象：“c语言”，“java编程”，“php网络编程”，“c++”，“数据结构”

提供无参的构造器和一个有参的构造器

|  |
| --- |
|  |

* 以下程序的运行结果是

|  |
| --- |
| public class Test1 {  public static void main(String[] args) {  new A(new B());  }  }  class A{  public A(){  System.out.println("A");  }  public A(B b){  this();  System.out.println("AB");  }  }  class B{  public B(){  System.out.println("B");  }  } |
| 答案：  B  A  AB |

* 写出结果。

|  |
| --- |
| public class Test{  public static void leftshift(int i, int j){  i+=j;  }  public static void main(String args[]){  int i = 4, j = 2;  leftshift(i, j);  System.out.println(i);  }  } |
| 答案：4． 和leftShift函数没关系。 |

* 写出结果

|  |
| --- |
| public class Demo{  public static void main(String[] args){  int[] a=new int[1];  modify(a);  System.out.println(a[0]); //  }  public static void modify(int[] a){  a[0]++;  }  } |
| 答案： 1 |

* 写出结果

|  |
| --- |
| public class TestA {  int i ;  void change(int i){  i++;  System.out.println(i);  }  void change1(TestA t){  t.i++;  System.out.println(t.i);  }  public static void main(String[] args) {  TestA ta = new TestA();  System.out.println(ta.i); //  ta.change(ta.i);//  System.out.println(ta.i); //  ta.change1(ta); //  System.out.println(ta.i);//  }  } |

* 写出结果

|  |
| --- |
| class Value{  int i = 15;  }  class Test{  public static void main(String argv[]) {  Test t = new Test();  t.first();  }  public void first() {  int i = 5;  Value v = new Value();  v.i = 25;  second(v, i);  System.out.println(v.i);  }  public void second(Value v, int i) {  i = 0;  v.i = 20;  Value val = new Value();  v = val;  System.out.print(v.i + " " + i);  }  } |

A.15 0 20

B.15 0 15

C.20 0 20

D.0 15 20

A is correct.



|  |
| --- |
| 1. public class Test {  2. int x= 12;  3. public void method(int x) {  4. x+=x;  5. System.out.println(x);  6. }  7. }  Given:  34. Test t = new Test();  35. t.method(5); |
| What is the output from line 5 of the Test class?  A. 5  B. 10  C. 12  D. 17  E. 24 |



|  |
| --- |
| import java.util.Arrays;  public class PassValueExer2{  public static void main(String[] args){  int[] array = {3,2,5,1,7};  //调用sort方法，实现从大到小排序  //在此处补充代码  ....  //显示结果  System.out.println("排序后的结果是：" + Arrays.toString(array));  }  //要求使用冒泡排序完成  public void sort(//形参？){  }  } |
| /\*  考点：  1、方法的参数传递机制  2、冒泡排序  \*/  import java.util.Arrays;  public class PassValueExer2{  public static void main(String[] args){  int[] array = {3,2,5,1,7};  PassValueExer2 exer = new PassValueExer2();  //调用sort方法，实现排序  exer.sort(array);//实参给形参的是地址，数组的首地址    //遍历结果  System.out.println("排序后的结果是：" + Arrays.toString(array));  }    //功能：用冒泡排序，实现为数组排序，而且从大到小  //形参的类型？我要把什么传过来  //传递数组  //接收的类型也肯定是数组，即形参接收实参，即形参的类型是int[]  public void sort(int[] arr){  //冒泡排序  //在这里对谁排，对arr进行排序  for(int i=1; i<arr.length; i++){//多少轮  //每一轮，从左往后--》for(int j=0;...)  //要实现从大到小-->前面的元素比后面的元素小，就交换  //每一轮几次，  //假设，数组的长度为5  //第一轮：4次，i=1, j=0,1,2,3 j<4 j< arr.length-i  //第二轮：3次，i=2, j=0,1,2  for(int j=0; j<arr.length-i; j++){  if(arr[j] < arr[j+1]){  int temp = arr[j];  arr[j] = arr[j+1];  arr[j+1] = temp;  }  }  }  }  } |

* 以下代码的执行结果是什么

|  |
| --- |
| public static void main(String[] args) {  int i = 0;  change(i);  i = i++;  System.out.println("i = " + i);  }  public static void change(int i){  i++;  } |
| 答案：i = 0 |

* 以下程序的运行结果

|  |
| --- |
| public static void main(String[] args) {  String str = new String("world");  char[] ch = new char[]{'h','e','l','l','o'};  change(str,ch);  System.out.println(str);  System.out.println(String.valueOf(ch));  }  public static void change(String str, char[] arr){  str = "change";  arr[0] = 'a';  arr[1] = 'b';  arr[2] = 'c';  arr[3] = 'd';  arr[4] = 'e';  } |
| 答案：  world  abcde |

* 以下代码的运行结果是？

|  |
| --- |
| public class Test {  int a;  int b;  public void f(){  a = 0;  b = 0;  int[] c = {0};  g(b,c);  System.out.println(a + " " + b + " " + c[0]);  }  public void g(int b, int[] c){  a = 1;  b = 1;  c[0] = 1;  }  public static void main(String[] args) {  Test t = new Test();  t.f();  }  } |
| 答案：1 0 1 |

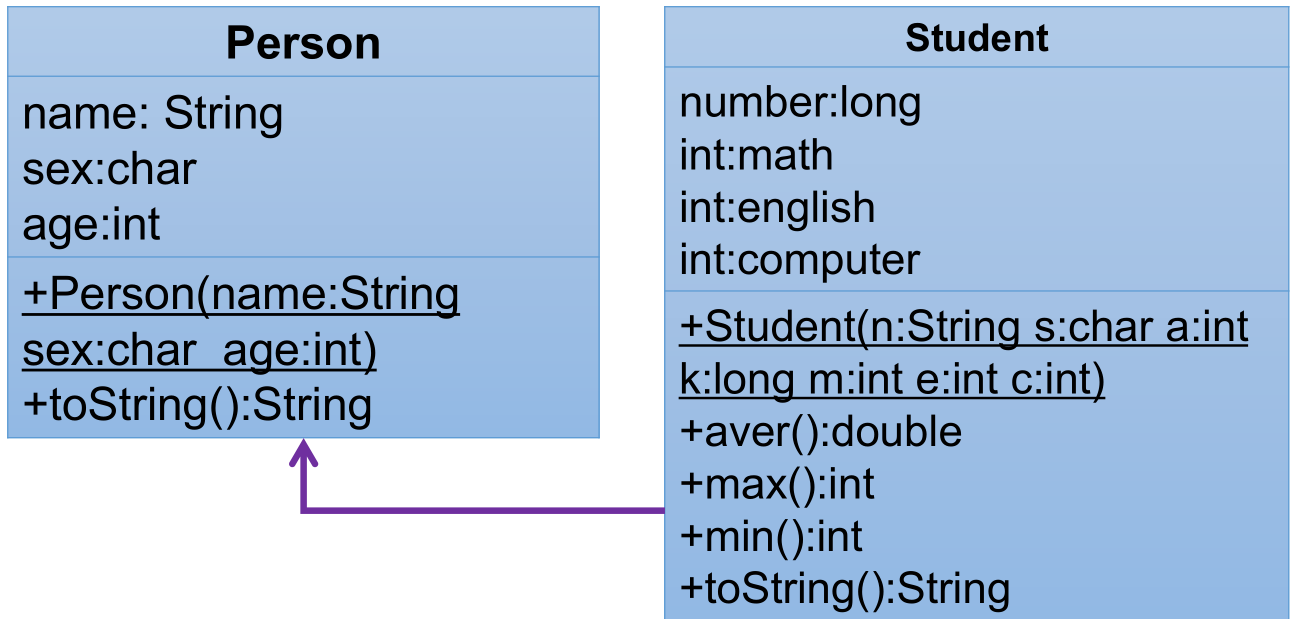
* 当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

答：是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的

* 补足compare函数内的代码，不许添加其他函数

|  |
| --- |
| class Circle {  private double radius;  public Circle(double r) {  radius = r;  }  public Circle compare(Circle cir) {  // 程序代码  /\*  \* if(this.radius>cir.radius) return this; return cir;  \*/  // return (this.radius>cir.radius)?this: cir;  }  } |
| class TC {  public static void main(String[] args) {  Circle cir1 = new Circle(1.0);  Circle cir2 = new Circle(2.0);  Circle cir;  cir = cir1.compare(cir2);  if (cir1 == cir)  System.out.println("圆1的半径比较大");  else  System.out.println("圆2的半径比较大");  }  } |

* 定义类一个学生类Student ，它继承自Person类



|  |
| --- |
|  |

* (1)定义一个ManKind类，包括

成员变量int sex和int salary；

方法void manOrWoman()：根据sex的值显示“man”(sex==1)或者“woman”(sex==0)；

方法void employeed()：根据salary的值显示“no job”(salary==0)或者“ job”(salary!=0)。

|  |
| --- |
|  |

(2)定义类Kids继承ManKind，并包括

成员变量int yearsOld；

方法printAge()打印yearsOld的值。

|  |
| --- |
|  |

(3)定义类KidsTest，在类的main方法中实例化Kids的对象someKid，用该对象访问其父类的成员变量及方法。

|  |
| --- |
|  |

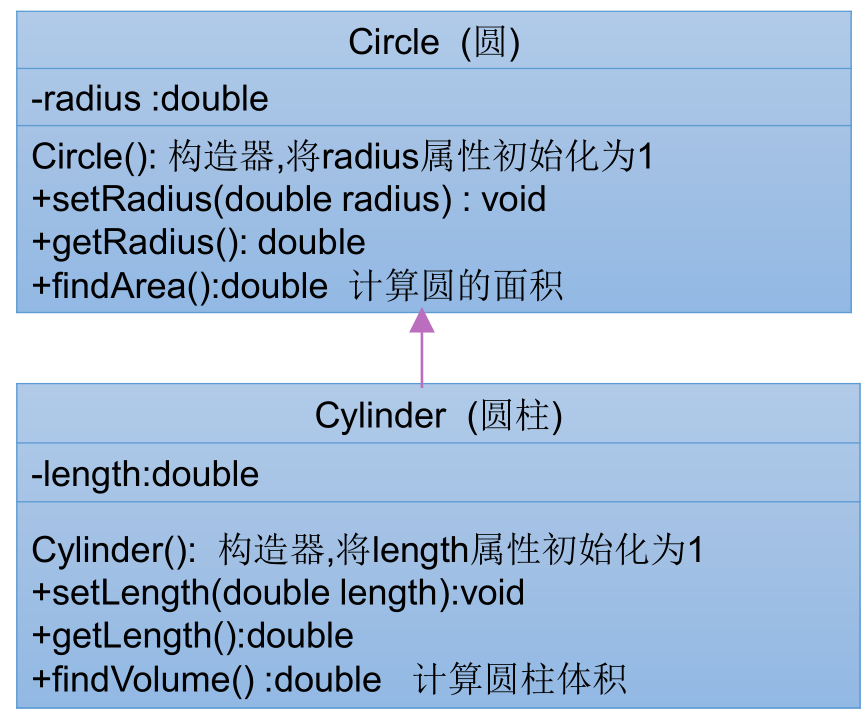
* 修改练习1.2中定义的类Kids，在Kids中重新定义employeed()方法，覆盖父类ManKind中定义的employeed()方法，输出“Kids should study and no job.”

|  |
| --- |
|  |

* 修改练习1.2中定义的类Kids中employeed()方法，在该方法中调用父类ManKind的employeed()方法，然后再输出“but Kids should study and no job.”

|  |
| --- |
|  |

* 根据下图实现类在 。在CylinderTest类 类建 中创建Cylinder 类的对象，设置圆柱的底面半径和高，并输出圆柱的体积。



|  |
| --- |
|  |

修改练习1.3中定义的Cylinder类，在Cylinder类中覆盖findArea()方法，计算圆柱的表面积。考虑：findVolume方法怎样做相应的修改？

|  |
| --- |
|  |

在CylinderTest类中创建Cylinder类的对象，设置圆柱的底面半径和高，并输出圆柱的表面积和体积。

|  |
| --- |
|  |

在CylinderTest类中创建一个Circle类的对象，设置圆的半径，计算输出圆的面积。体会父类和子类成员的分别调用。

|  |
| --- |
|  |

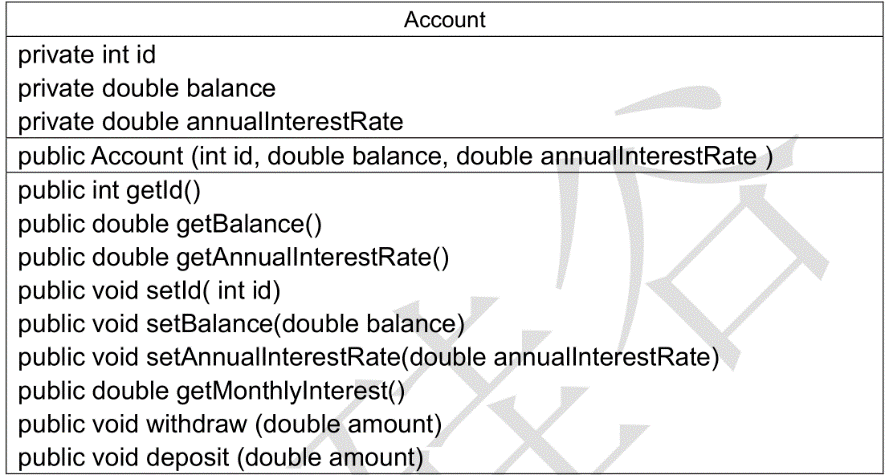
修改练习1.3中定义的Circle类和Cylinder类的构造器，利用构造器参数为对象的所有属性赋初值。

|  |
| --- |
|  |

* 继承的发展

|  |
| --- |
| class Person {  public String name;  public int age;  public Date birthDate;  public String getInfo() {  //...  }  } |
| class Student {  public String name;  public int age;  public Date birthDate;  public String school;  public String getInfo() {  // ...  }  } |
| class Person {  public String name;  public int age;  public Date birthDate;  public String getInfo() {  // ...  }  }  class Student extends Person {  public String school;  } |

* 如果现在父类的一个方法定义成private访问权限，在子类中将此方法声明为default访问权限，那么这样还叫重写吗？(NO)
* 写一个名为 Account 的类模拟账户。该类的属性和方法如下图所示。该类包括的属性：账号 id，余额 balance，年利率 annualInterestRate；包含的方法：访问器方法（getter 和setter 方法），返回月利率的方法 getMonthlyInterest()，取款方法 withdraw()，存款方法deposit()。



|  |
| --- |
|  |

写一个用户程序测试 Account 类。在用户程序中，创建一个账号为 1122、余额为 20000、年利率 4.5%的 Account 对象。使用 withdraw 方法提款 30000 元，并打印余额。再使用 withdraw 方法提款 2500 元，使用 deposit 方法存款 3000 元，然后打印余额和月利率。

提示：在提款方法 withdraw 中，需要判断用户余额是否能够满足提款数额的要求，如果不能，应给出提示。

运行结果如图所示：



|  |
| --- |
|  |

创建 Account 类的一个子类 CheckAccount 代表可透支的账户，该账户中定义一个属性overdraft 代表可透支限额。在 CheckAccount 类中重写 withdraw 方法，其算法如下：

如果（取款金额<账户余额），可直接取款；如果（取款金额>账户余额），计算需要透支的额度，判断可透支额 overdraft 是否足够支付本次透支需要，如果可以将账户余额修改为 0，冲减可透支金额；如果不可以提示用户超过可透支额的限额。

要求：写一个用户程序测试 CheckAccount 类。在用户程序中，创建一个账号为 1122、余额为 20000、年利率 4.5%，可透支限额为 5000 元的 CheckAccount 对象。

使用 withdraw 方法提款 5000 元，并打印账户余额和可透支额。

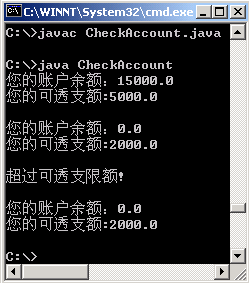
再使用 withdraw 方法提款 18000 元，并打印账户余额和可透支额。

再使用 withdraw 方法提款 3000 元，并打印账户余额和可透支额。

提示：（1）子类 CheckAccount 的构造方法需要将从父类继承的 3 个属性和子类自己的属性全部初始化。

（2）父类Account的属性balance被设置为private，但在子类CheckAccount的withdraw方法中需要修改它的值，因此应修改父类的 balance 属性，定义其为 protected。

运行结果如下图所示：



|  |
| --- |
|  |

* 为什么super(…)和this(…)调用语句不能同时在一个构造器中出现？
* 为什么super(…)或this(…)调用语句只能作为构造器中的第一句出现？

也就是说你必须在构造器的第一行放置super或者this构造器，否则编译器会自动地放一个空参数的super构造器的，其他的构造器也可以调用super或者this，调用成一个递归构造链，最后的结果是父类的构造器（可能有多级父类构造器）始终在子类的构造器之前执行，递归的调用父类构造器。无法执行当前的类的构造器。也就不能实例化任何对象，这个类就成为一个无为类。从另外一面说，子类是从父类继承而来，继承了父类的属性和方法，如果在子类中先不完成父类的成员的初始化，则子类无法使用，应为在java中不允许调用没初始化的成员。在构造器中是顺序执行的，也就是说必须在第一行进行父类的初始化。而super能直接完成这个功能。This()通过调用本类中的其他构造器也能完成这个功能。 因此，this()或者super()必须放在第一行。

* 继承成员变量和继承方法的区别

|  |
| --- |
| class Base {  int count = 10;  public void display() {  System.out.println(this.count);  }  } |
| class Sub extends Base {  int count = 20;  public void display() {  System.out.println(this.count);  }  } |
| public class FieldMethodTest {  public static void main(String[] args){  Sub s = new Sub();  System.out.println(s.count);  s.display();  Base b = s;  System.out.println(b == s);  System.out.println(b.count);  b.display();  }  } |

* 建立InstanceTest 类，在类中定义方法method(Person e);在method 中:

(1) 根据e 的类型调用相应类的getInfo() 方法。

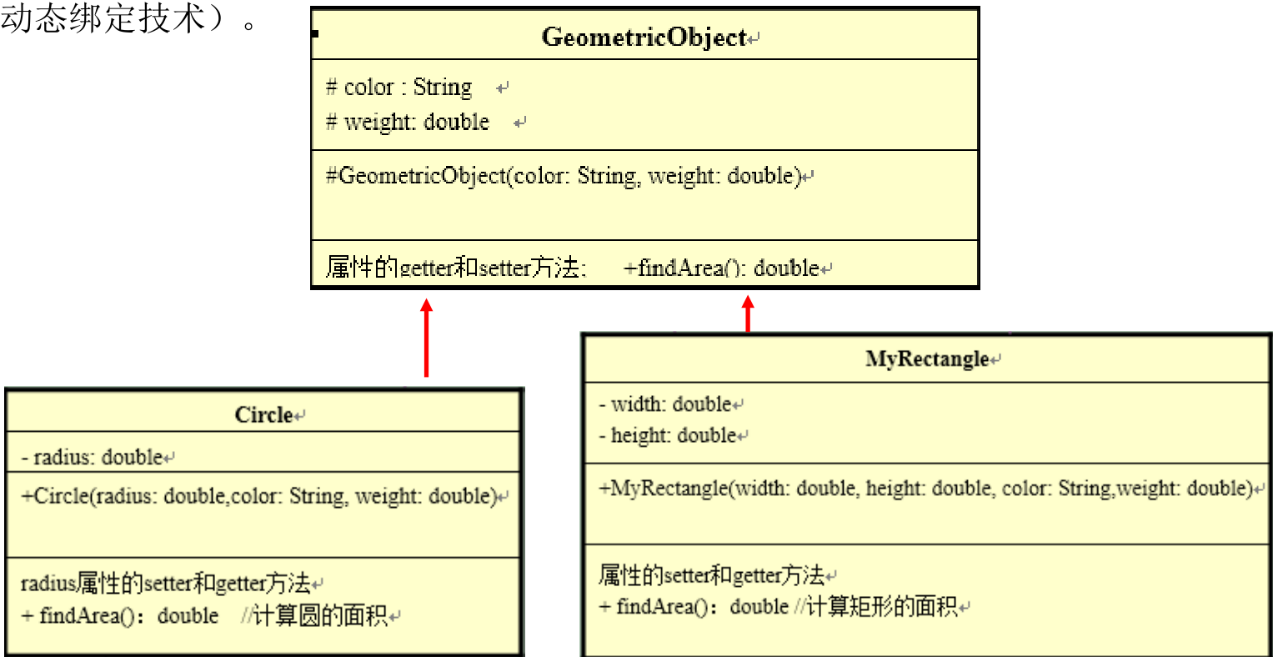
(2) 根据e 的类型执行：如果e 为Person，类的对象，输出“a person”;如果e 为Student象 类的对象，输出：“a student”;如果e 为Graduate 类的对象，输出：“a graduated student”“a student”“a person”

|  |
| --- |
| class Person {  protected String name="person";  protected int age=50;  public String getInfo() {  return "Name: "+ name + "\n" +"age: "+ age;  }  } |
| class Student extends Person {  protected String school="pku";  public String getInfo() {  return "Name: "+ name + "\nage: "+ age+ "\nschool: "+ school;  }  } |
| class Graduate extends Student{  public String major="IT";  public String getInfo(){  return "Name: "+ name + "\nage: "+ age+ "\nschool: "+ school+"\nmajor:"+major;  }  } |

* 定义三个类，父类GeometricObject代表几何形状，子类Circle代表圆形，MyRectangle代表矩形。

|  |
| --- |
|  |

定义一个测试类GeometricTest，编写equalsArea方法测试两个对象的面积是否相等（注意方法的参数类型，利用动态绑定技术），编写displayGeometricObject方法显示对象的面积（注意方法的参数类型，利用动态绑定技术）



|  |
| --- |
|  |

* 多态是编译时行为还是运行时行为？如何证明？

|  |
| --- |
| //面试题：多态是编译时行为还是运行时行为？  //证明如下：  class Animal {  protected void eat() {  System.out.println("animal eat food");  }  } |
| class Cat extends Animal {  protected void eat() {  System.out.println("cat eat fish");  }  } |
| class Dog extends Animal {  public void eat() {  System.out.println("Dog eat bone");  }  } |
| class Sheep extends Animal {  public void eat() {  System.out.println("Sheep eat grass");  }  } |
| public class InterviewTest {  public static Animal getInstance(int key) {  switch (key) {  case 0:  return new Cat ();  case 1:  return new Dog ();  default:  return new Sheep ();  }  }  public static void main(String[] args) {  int key = new Random().nextInt(3);  System.out.println(key);  Animal animal = getInstance(key);  animal.eat();  }  } |

|  |
| --- |
| //考查多态的笔试题目：  class Base {  public void add(int a, int... arr) {  System.out.println("base");  }  } |
| class Sub extends Base {  public void add(int a, int[] arr) {  System.out.println("sub\_1");  }  // public void add(int a, int b, int c) {  // System.out.println("sub\_2");  // }  } |
| public class InterviewTest1 {  public static void main(String[] args) {  Base base = new Sub();  base.add(1, 2, 3);  // Sub s = (Sub)base;  // s.add(1,2,3);  }  } |

* == 号与equals()

|  |
| --- |
| int it = 65;  float fl = 65.0f;  System.out.println(“65和65.0f是否相等？” + (it == fl)); //true  char ch1 = 'A'; char ch2 = 12;  System.out.println("65和'A'是否相等？" + (it == ch1));//true  System.out.println(“12和ch2是否相等？" + (12 == ch2));//true  String str1 = new String("hello");  String str2 = new String("hello");  System.out.println("str1和str2是否相等？"+ (str1 == str2));//false  System.out.println("str1是否equals str2？"+(str1.equals(str2)));//true  System.out.println(“hello” == new java.util.Date()); //编译不通过 |

* 编写Order类，有int型的orderId，String型的orderName，相应的getter()和setter()方法，两个参数的构造器，重写父类的equals()方法：public boolean equals(Object obj)，并判断测试类中创建的两个对象是否相等

|  |
| --- |
|  |

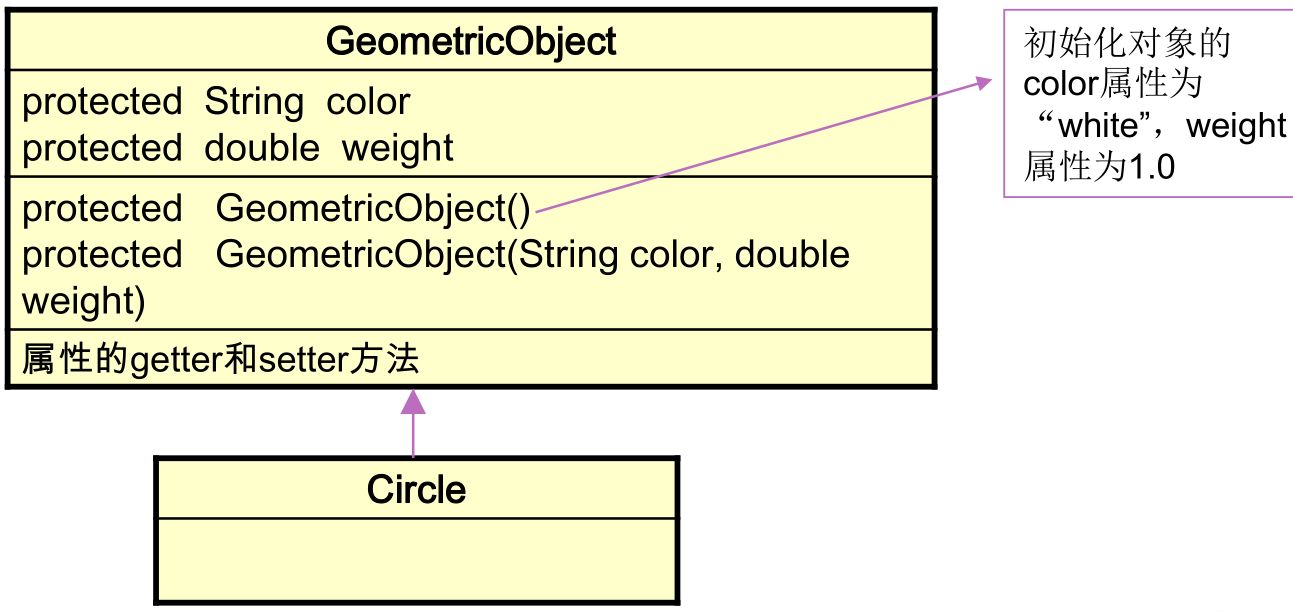
* 请根据以下代码自行定义能满足需要的MyDate类,在MyDate类中覆盖equals方法，使其判断当两个MyDate类型对象的年月日都相同时，结果为true，否则为false。 public boolean equals(Object o)

|  |
| --- |
| public class EqualsTest {  public static void main(String[] args) {  MyDate m1 = new MyDate(14, 3, 1976);  MyDate m2 = new MyDate(14, 3, 1976);  if (m1 == m2) {  System.out.println("m1==m2");  } else {  System.out.println("m1!=m2"); // m1 != m2  }  if (m1.equals(m2)) {  System.out.println("m1 is equal to m2");// m1 is equal to m2  } else {  System.out.println("m1 is not equal to m2");  }  }  } |

* 输出结果

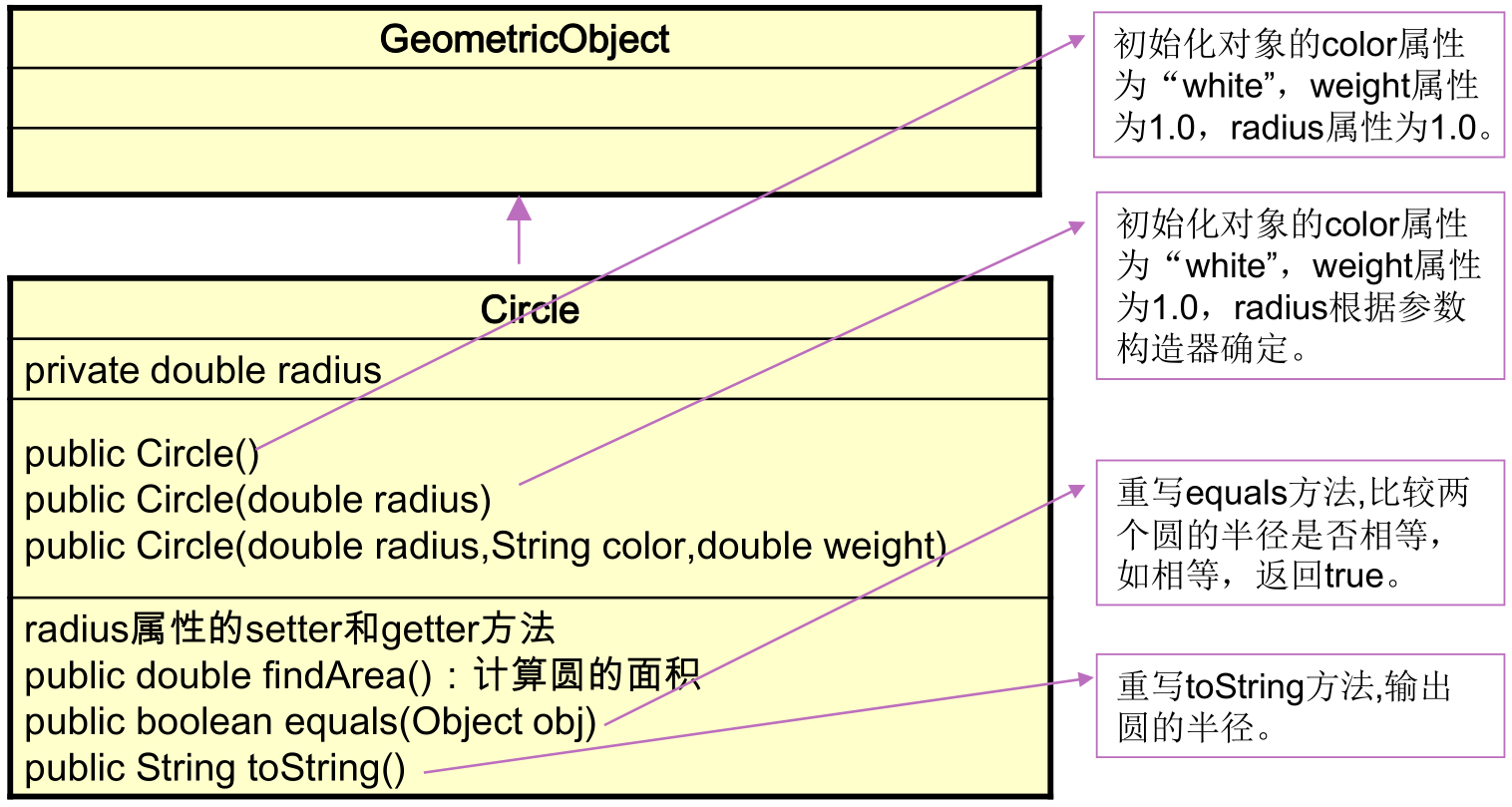
|  |
| --- |
| public void test() {  char[] arr = new char[] { 'a', 'b', 'c' };  System.out.println(arr);//  int[] arr1 = new int[] { 1, 2, 3 };  System.out.println(arr1);//  double[] arr2 = new double[] { 1.1, 2.2, 3.3 };  System.out.println(arr2);//  } |

* 定义两个类，父类GeometricObject代表几何形状，子类Circle代表圆形。



|  |
| --- |
|  |

写一个测试类，创建两个Circle对象，判断其颜色是否相等；利用equals方法判断其半径是否相等；利用toString()方法输出其半径。



|  |
| --- |
|  |

* 如下两个题目输出结果相同吗？各是什么

|  |
| --- |
| Object o1 = true ? new Integer(1) : new Double(2.0);  System.out.println(o1);// |
| Object o2;  if (true)  o2 = new Integer(1);  else  o2 = new Double(2.0);  System.out.println(o2);// |

* 输出结果

|  |
| --- |
| public void method1() {  Integer i = new Integer(1);  Integer j = new Integer(1);  System.out.println(i == j);  Integer m = 1;  Integer n = 1;  System.out.println(m == n);//  Integer x = 128;  Integer y = 128;  System.out.println(x == y);//  } |

* 利用Vector代替数组处理：从键盘读入学生成绩（以负数代表输入结束），找出最高分，并输出学生成绩等级。

提示：数组一旦创建，长度就固定不变，所以在创建数组前就需要知道它的长度。而向量类java.util.Vector可以根据需要动态伸缩。

创建Vector对象：Vector v=new Vector();

给向量添加元素：v.addElement(Object obj); //obj必须是对象

取出向量中的元素：Object obj=v.elementAt(0);

注意第一个元素的下标是0，返回值是Object类型的。

计算向量的长度：v.size();

若与最高分相差10分内：A等；20分内：B等；30分内：C等；其它：D等

|  |
| --- |
|  |

* java类是否可以多继承，怎么实现多继承？

答：java没有多继承，但可以通过接口的形式来达到多继承的目地。

* 定义类A 和类B 如下

|  |
| --- |
| class A {  int a = 1;  double d = 2.0;  void show() {  System.out.println("Class A: a=" + a + "\td=" + d);  }  } |
| class B extends A {  float a = 3.0f;  String d = "Java program.";  void show() {  super.show();  System.out.println("Class B: a=" + a + "\td=" + d);  }  } |

(1) 若在应用程序的main 方法中有以下语句：

A a=new A();

a.show();

则输出的结果如何？

(2) 若在应用程序的main 方法中定义类B 的对象b：

A b=new B();

b.show();

则输出的结果如何？

答：输出结果为：

1）Class A: a=1 d=2.0

2）Class A: a=1 d=2.0；Class B: a=3.0 d=Java program。

* 写出重载与重写的区别。
* 重载（overload)和重写(overried，有的书也叫做“覆盖”）的区别？

答：方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被"屏蔽"了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。

* Overload的方法是否可以改变返回值的类型？

Overloaded的方法是可以改变返回值的类型

* 写出错误答案错误的原因

|  |
| --- |
| class Demo{  int show(int a,int b){return 0;}  } |

下面那些函数可以存在于Demo的子类中。

A.public int show(int a,int b){return 0;}//可以，覆盖。

B.private int show(int a,int b){return 0;}//不可以，权限不够。

C.private int show(int a,long b){return 0;}//可以，和父类不是一个函数。没有覆盖，相当于重载。

D.public short show(int a,int b){return 0;}//不可以，因为该函数不可以和给定函数出现在同一类中，或者子父类中。

E.static int show(int a,int b){return 0;}//不可以，静态只能覆盖静态。

* 写出程序结果

|  |
| --- |
| class Super {  public int get() {  return 4;  }  } |
| class Demo15 extends Super {  public long get() {  return 5;  }  public static void main(String[] args) {  Super s = new Demo15();  System.out.println(s.get());  }  } |

编译失败，因为子类父类中的get方法没有覆盖。但是子类调用时候不能明确返回的值是什么类型。所以这样的函数不可以存在子父类中。

* 按要求实现下列问题：实现一个名为Person的类和它的子类Employee，Employee有两个子类Faculty和Staff。具体要求如下：

1） Person类中的属性有：姓名name（String类型），地址address（String类型），电话号码telphone（String类型）和电子邮件地址email（String类型）；

2） Employee类中的属性有：办公室office（String类型），工资wage（double类型），受雇日期hiredate（String类型）；

3） Faculty类中的属性有：学位degree（String类型），级别level（String类型）；

4） Staff类中的属性有：职务称号duty（String类型）。

5） 现有对象Person p1 =new Faculty()和Person p2 =new Staff ()，请分别为p1的属性赋值“本科”和Staff类的duty赋值“职员”

|  |
| --- |
|  |

* 比较两个String总是false，但是它们明明都是"abc" ！

答：比较String一定要使用equals或equalsIgnoreCase方法，不要使用 == ==比较的是两个引用（变量）是否指向了同一个对象，而不是比较其内容。

* 为什么要重写toString()方法？
* 为什么要重写equals()方法？
* 定义一个person类，属性如下：

（1）身份证号，性别，姓名，年龄，户籍，出生日期（Data类型，需要引用java.uitl.Data）功能：

（2）自我介绍：介绍格式：（toString）身份证号+姓名+户籍

（3）提供对象比较equals方法，只要身份证号+姓名相同就认为对象相等

|  |
| --- |
|  |

* 写出“==”和equals的区别
* 类似：两个对象a和b，请问a==b和a.equals(b)有什么区别？
* 以下代码如何优化

|  |
| --- |
| if(username.equals(“admin”){  ....  } |
| 答案：  if(“admin”.equals(username)){  } |

* 创建一个汽车类（Car）

（1）为其定义两个属性：颜色和型号。每个属性都使用private进行封装，为每个属性设置set、get方法。

（2）为该类创建两个构造方法。第一个为无参的构造方法。第二个为带参构造方法

（3）重写toString方法，通过toString输出汽车信息。

（4）重写equals方法，如果汽车的颜色和型号相同就认为是同一辆车。

（5）实例化两个对象，输出两个对象的信息，比较两个对象是否是同一个对象。

|  |
| --- |
|  |

* 编写一个Java应用程序，设计一个汽车类Vehicle，包含的属性有车轮个数wheels和车重weight。小车类Car是Vehicle的子类，其中包含的属性有载人数loader。卡车类Truck是Car类的子类，其中包含的属性有载重量payload。

（1）每个类都有构造方法进行属性初识化

（2）每个类都输出相关数据的toString方法

（3）使用Test类中的main方法定义各类初始化数据后台打印相关数据

|  |
| --- |
|  |

* 定义员工类Employee，包含姓名、工号和工资，包含计算奖金方法bonus，普通员工和经理都是员工，计算奖金的方法为工资\*奖金系数，普通员工的奖金系数为1.5（常量），经理为2（常量），分别实现bonus方法，创建对象测试。

|  |
| --- |
|  |

* 定义一个交通工具类，并定义抽象方法，说明交通工具可以移动。继承交通工具而产生汽车、飞机、轮船类，并实现其移动方法。

|  |
| --- |
|  |

* equals()编程

（1）编写一个人类Person，其中包含姓名、性别和年龄的属性，包含构造方法以及显示姓名、性别和年龄的方法。

|  |
| --- |
|  |

（2）再编写一个学生类Student，它继承Person类，其中包含学号属性，包含构造方法以及显示学号的方法。

|  |
| --- |
|  |

（3）在Student类里面重写equals方法，只要是学号和姓名一致就认为是同一个学生。

|  |
| --- |
|  |

（4）最后编写一个测试类，此类里面包含main()方法，在main()方法中定义两个学生s1和s2并给他们赋值，最后显示他们的学号、姓名、性别以及年龄，以及判断两个学生是否是同一个学生。

|  |
| --- |
|  |

* 按要求实现下列问题

1）封装一个西游记人物类Person：包含私有属性：姓名（name）、身高（height）、武器（weapon）、种族（race），并提供访问器（get方法）和修改器（set方法）

|  |
| --- |
|  |

2） 在测试类中实例化三个对象：tangseng（唐僧）、sunwukong（孙悟空）、baigujing（白骨精），分别设置他们的种族为：人族、仙族、妖族

|  |
| --- |
|  |

3） 在打印对象时，按照例如：“姓名：孙悟空；种族：仙族”的格式在控制台输出信息。

|  |
| --- |
|  |

4） 提供一个技能方法void skill()，这个方法根据当前对象name属性判断技能，如果是孙悟空，就打印“七十二变”，如果是唐僧，就打印“紧箍咒”，如果是白骨精，就打印“九阴白骨爪”，在测试类中调用这个方法。

|  |
| --- |
|  |

重写equals方法进行判断，如果种族属性是人族或者仙族，则返回true，如果是妖族，则返回false，在测试类中对孙悟空和唐僧进行比较，打印结果，对唐僧和白骨金进行比较，并打印结果。

|  |
| --- |
|  |

* 写出8中基本数据类型的包装类
* int 和 Integer 有什么区别

答：Java 提供两种不同的类型：引用类型和原始类型（或内置类型）。int是java的原始数据类型，Integer是java为int提供的封装类。Java为每个原始类型提供了封装类。原始类型封装类型及其对应的包装类：boolean Boolean,char Character,byte Byte,short Short,int Integer,long Long,float Float,doubl Double。引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关。

* 请写出java基本数据类型及对应的封装类型，并说明封装类的作用。
* 以下代码的运行结果是

|  |  |
| --- | --- |
| **public** **static** **void** main(String[] args) {  Integer i1 = 128;  Integer i2 = 128;  **int** i3 = 128;  **int** i4 = 128;  System.***out***.println(i1 == i2);  System.***out***.println(i3 == i4);  System.***out***.println(i1 == i3);  } | |
| 答案：  false  true  true | Integer的i1和i2是对象，他们==比较的是地址。如果-128~127范围，那么使用缓存的常量对象，如果超过这个范围，是新new的对象，不是常量对象 |

* 以下代码的运行结果

|  |
| --- |
| **public** **static** **void** main(String[] args) {  **double** a = 2.0;  **double** b = 2.0;  Double c = 2.0;  Double d = 2.0;  System.***out***.println(a == b);  System.***out***.println(c == d);  System.***out***.println(a == d);  } |
| 答案：  true  false  true |

* 写出程序结果

|  |
| --- |
| **class** Test {  Test() {  System.***out***.println("Test");  }  } |
| **class** Demo **extends** Test {  Demo() {  // super();  System.***out***.println("Demo");  }  **public** **static** **void** main(String[] args) {  **new** Demo();  **new** Test();  }  } |

Test

Demo

Test

* 写出程序结果

|  |
| --- |
| class Super {  int i = 0;  public Super(String a) {  System.out.println("A");  i = 1;  }  public Super() {  System.out.println("B");  i += 2;  }  } |
| class Demo extends Super {  public Demo(String a) {  // super();  System.out.println("C");  i = 5;  }  public static void main(String[] args) {  int i = 4;  Super d = new Demo("A");  System.out.println(d.i);  }  } |

B C 5

* 写出程序结果

|  |
| --- |
| class Super {  int i = 0;  public Super(String s) {  i = 1;  }  } |
| class Demo extends Super {  public Demo(String s) {  i = 2;  }  public static void main(String[] args) {  Demo d = new Demo("yes");  System.out.println(d.i);  }  } |

// 编译失败，因为父类中缺少空参数的构造器。

// 或者子类应该通过super语句指定要调用的父类中的构造器。

* 写出程序结果

|  |
| --- |
| public class FatherClass{  public FatherClass(){  System.out.println("FatherClass Create");  }  } |
| public class ChildClass extends FatherClass{  public ChildClass(){  System.out.println("ChildClass Create");  }  public static void main(String[] args) {  FatherClass fc = new FatherClass();  ChildClass cc = new ChildClass();  }  } |
| 当执行如下操作  C:\>java test.ChildClass |

输出结果为：

FatherClass Create

FatherClass Create

ChildClass Create

* 以下代码的运行结果

|  |
| --- |
| public class Test {  public static void main(String[] args) {  Base b1 = new Base();  Base b2 = new Sub();  }  } |
| class Base{  Base(){  method(100);  }  public void method(int i){  System.out.println("base : " + i);  }  } |
| class Sub extends Base{  Sub(){  super.method(70);  }  public void method(int j){  System.out.println("sub : " + j);  }  } |
| 答案：  base : 100  sub : 100  base : 70 |

* 编写一个Person类，包括属性（name、age），构造器、方法say(返回自我介绍的字符串）。

|  |
| --- |
|  |

* 编写一个Student类，继承Person类，增加ID、score属性，以及构造器，并覆盖say方法。

|  |
| --- |
|  |

* 编写StudentTest类，创建分别Person和Student对象，调用say方法输出自我介绍。

|  |
| --- |
|  |

* 写出程序结果

|  |
| --- |
| class A {  private int a;  public void setA(int a) {  this.a = a;  }  public int getA() {  return a;  }  } |
| class B extends A {  private int a;  public void setA(int a) {  this.a = a;  }  // public int getA(){  // return a;  // }  } |
| public class PersonTest {  public static void main(String[] args) {  A c = new B();  c.setA(5);  System.out.println(c.getA());  }  } |

* 写出程序结果

|  |
| --- |
| class Fu {  boolean show(char a) {  System.out.println(a);  return true;  }  } |
| class Demo extends Fu {  public static void main(String[] args) {  int i = 0;  Fu f = new Demo();  Demo d = new Demo();  for (f.show('A'); f.show('B') && (i < 2); f.show('C')) {  i++;  d.show('D');  }  }  boolean show(char a) {  System.out.println(a);  return false;  }  } |

A B

* 写出程序结果

|  |
| --- |
| class Fu {  int num = 4;  void show() {  System.out.println("showFu");  }  } |
| class Zi extends Fu {  int num = 5;  void show() {  System.out.println("showZi");  }  } |
| class T {  public static void main(String[] args) {  Fu f = new Zi();  Zi z = new Zi();  System.out.println(f.num);  System.out.println(z.num);  f.show();  z.show();  }  } |

4

5

showZi

showZi

* 判断错误的地方

|  |
| --- |
| class Person {  private int id;  private static int total = 0;  public static int getTotalPerson() {  //id++; //非法  return total;  }  public Person() {  total++;  id = total;  }  } |
| public class PersonTest {  public static void main(String[] args) {  System.out.println("Number of total is " + Person.getTotalPerson());  //没有创建对象也可以访问静态方法  Person p1 = new Person();  System.out.println( "Number of total is "+ Person.getTotalPerson());  }  } |

* 为什么父类 static 方法不能被重写
* 编写一个类实现银行账户的概念，包含的属性有“帐号”、“密码”、“存款余额”、“利率”、“最小余额”，定义封装这些属性的方法。账号要自动生成。

|  |
| --- |
|  |

编写主类，使用银行账户类，输入、输出3个储户的上述信息。考虑：哪些属性可以设计成static属性

|  |
| --- |
|  |

* 此处，Something类的文件名叫OtherThing.java

|  |
| --- |
| class Something {  public static void main(String[] something\_to\_do) {  System.out.println("Do something ...");  }  } |

上述程序是否可以正常编译、运行？

* 指出错误

|  |
| --- |
| public final class Test {  public static int totalNumber = 5;  public final int ID;  public Test() {  ID = ++totalNumber; // 可在构造器中给final修饰的“变量”赋值  }  public static void main(String[] args) {  Test t = new Test();  System.out.println(t.ID);  final int I = 10;  final int J;  J = 20;  J = 30; // 非法  }  } |

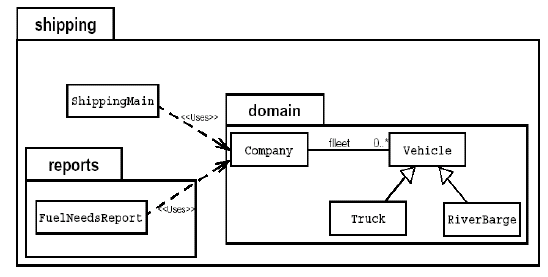
* 排错

|  |
| --- |
| public class Something {  public int addOne(final int x) {  return ++x;  // return x + 1;  }  } |

* 排错

|  |
| --- |
| public class Something {  public static void main(String[] args) {  Other o = new Other();  new Something().addOne(o);  }  public void addOne(final Other o) {  // o = new Other();  o.i++;  }  } |
| class Other {  public int i;  } |

* 卡车(Truck)和驳船(RiverBarge)的燃料效率和行驶距离的计算方法完全不同。Vehicle类不能提供计算方法，但子类可以。在航运公司系统中，Vehicle类需要定义两个方法分别计算运输工具的燃料效率和行驶距离。



|  |
| --- |
| public abstract class Vehicle{  public abstract double calcFuelEfficiency();//计算燃料效率的抽象方法  public abstract double calcTripDistance(); //计算行驶距离的抽象方法  } |
| public class Truck extends Vehicle{  public double calcFuelEfficiency( ) { //写出计算卡车的燃料效率的具体方法 }  public double calcTripDistance( ) { //写出计算卡车行驶距离的具体方法 }  } |
| public class RiverBarge extends Vehicle{  public double calcFuelEfficiency( ) { //写出计算驳船的燃料效率的具体方法 }  public double calcTripDistance( ) { //写出计算驳船行驶距离的具体方法}  } |

* 为什么抽象类不可以使用final关键字声明？
* 一个抽象类中可以定义构造器吗？
* 是否可以这样理解：抽象类就是比普通类多定义了抽象方法，除了不能直接进行类的实例化操作之外，并没有任何的不同？
* 编写一个Employee类，声明为抽象类，包含如下三个属性：name，id，salary。提供必要的构造器和抽象方法：work()。对于Manager类来说，他既是员工，还具有奖金(bonus)的属性。请使用继承的思想，设计CommonEmployee类和Manager类，要求类中提供必要的方法进行属性访问。
* 编写工资系统，实现不同类型员工(多态)的按月发放工资。如果当月出现某个Employee对象的生日，则将该雇员的工资增加100元。实验说明：

（1）定义一个Employee类，该类包含：private成员变量name,number,birthday，其中birthday 为MyDate类的对象；abstract方法earnings()；toString()方法输出对象的name,number和birthday。

|  |
| --- |
|  |

（2）MyDate类包含:private成员变量year,month,day ；toDateString()方法返回日期对应的字符串：xxxx年xx月xx日

|  |
| --- |
|  |

（3）定义SalariedEmployee类继承Employee类，实现按月计算工资的员工处理。该类包括：private成员变量monthlySalary；实现父类的抽象方法earnings(),该方法返回monthlySalary值；toString()方法输出员工类型信息及员工的name，number,birthday

|  |
| --- |
|  |

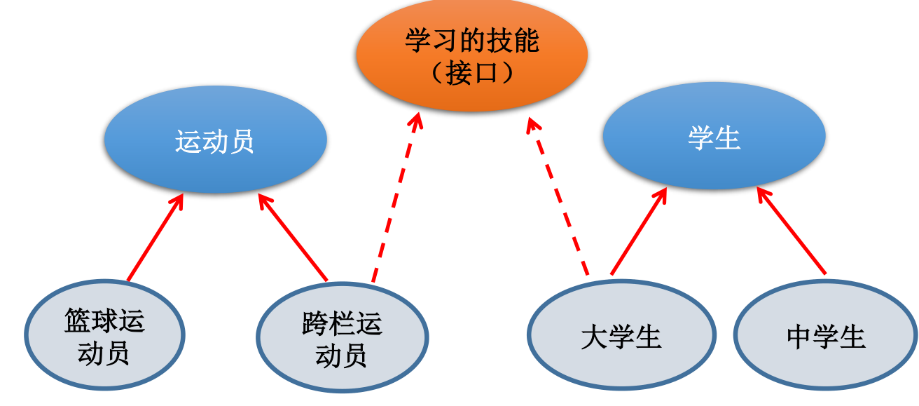
（4）参照SalariedEmployee类定义HourlyEmployee类，实现按小时计算工资的员工处理。该类包括：private成员变量wage和hour；实现父类的抽象方法earnings(),该方法返回wage\*hour值；toString()方法输出员工类型信息及员工的name，number,birthday。

|  |
| --- |
|  |

（5）定义PayrollSystem类，创建Employee变量数组并初始化，该数组存放各类雇员对象的引用。利用循环结构遍历数组元素，输出各个对象的类型,name,number,birthday,以及该对象生日。当键盘输入本月月份值时，如果本月是某个Employee对象的生日，还要输出增加工资信息。

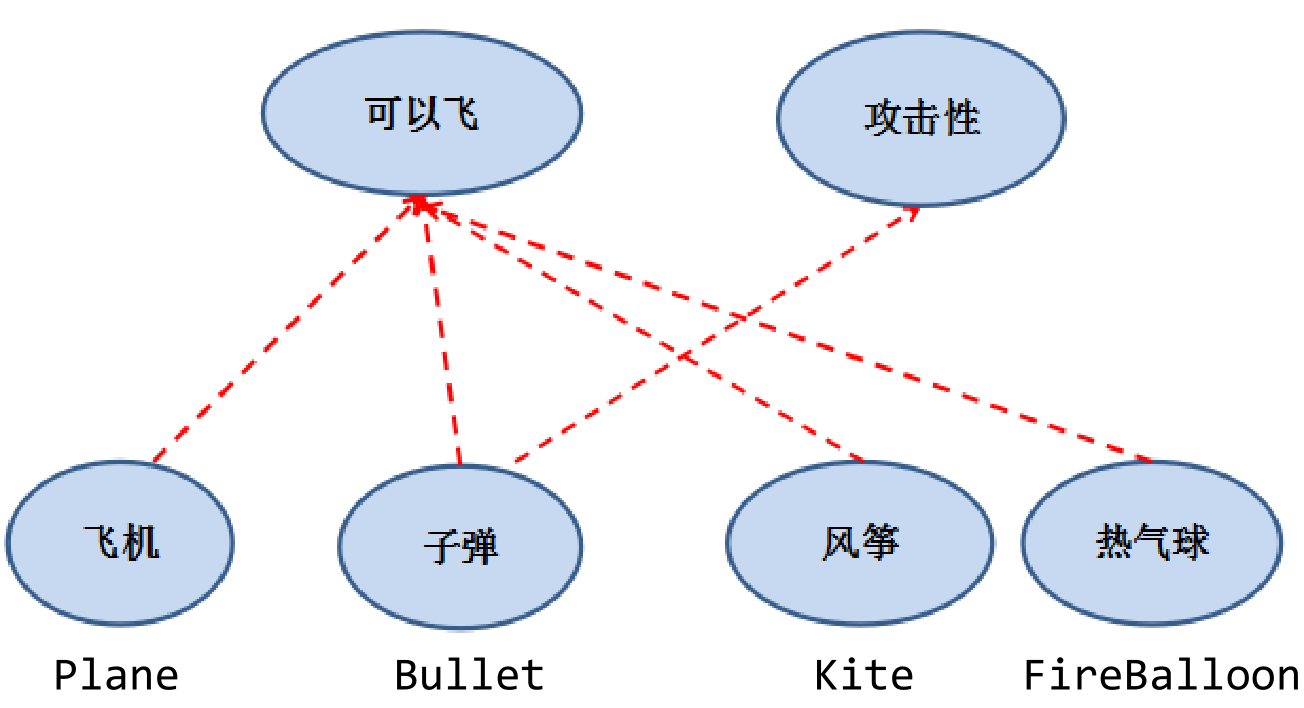
|  |
| --- |
| 提示：  //定义People类型的数组People c1[]=new People[10];  //数组元素赋值  c1[0]=new People("John","0001",20);  c1[1]=new People("Bob","0002",19);  //若People有两个子类Student和Officer，则数组元素赋值时，可以使父类类型的数组元素指向子类。  c1[0]=new Student("John","0001",20,85.0);  c1[1]=new Officer("Bob","0002",19,90.5); |
|  |

* 举例



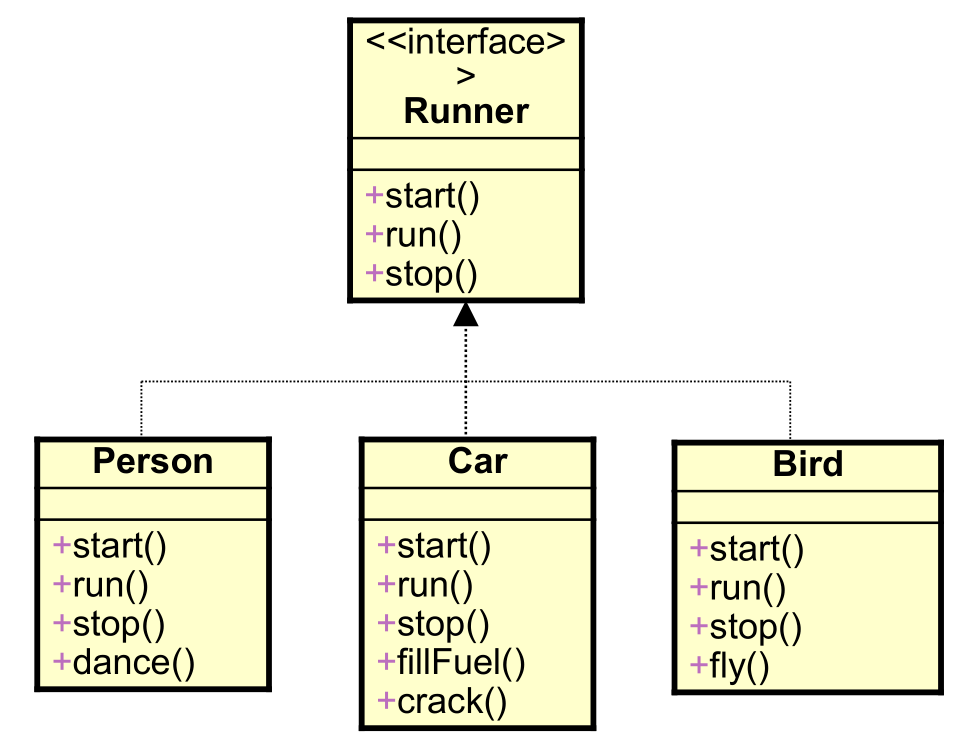
|  |
| --- |
|  |
|  |
|  |
|  |

* 举例



|  |
| --- |
|  |
|  |
|  |
|  |
|  |

* 举例



|  |
| --- |
|  |
|  |
|  |
|  |

* 排错

|  |
| --- |
| interface A {  int x = 0;  } |
| class B {  int x = 1;  } |
| class C extends B implements A {  public void pX() {  System.out.println(x);  }  public static void main(String[] args) {  new C().pX();  }  } |

* 排错

|  |
| --- |
| interface Playable {  void play();  } |
| interface Bounceable {  void play();  } |
| interface Rollable extends Playable,  Bounceable {  Ball ball = new Ball("PingPang");  } |
| class Ball implements Rollable {  private String name;  public String getName() {  return name;  }  public Ball(String name) {  this.name = name;  }  public void play() {  ball = new Ball("Football");  System.out.println(ball.getName());  }  } |

* 定义一个接口用来实现两个对象的比较。

|  |
| --- |
| interface CompareObject{  public int compareTo(Object o); //若返回值是 0 , 代表相等; 若为正数，代表当前对象大；负数代表当前对象小  } |

|  |
| --- |
|  |

* 定义一个Circle类，声明redius属性，提供getter和setter方法

|  |
| --- |
|  |

* 定义一个ComparableCircle类，继承Circle类并且实现CompareObject接口。在ComparableCircle类中给出接口中方法compareTo的实现体，用来比较两个圆的半径大小。

|  |
| --- |
|  |

* 定义一个测试类InterfaceTest，创建两个ComparableCircle对象，调用compareTo方法比较两个类的半径大小。

|  |
| --- |
|  |

* 思考 ： 参照上述做法定义矩形类 Rectangle 和 ComparableRectangle 类， 在ComparableRectangle类中给出compareTo方法的实现，比较两个矩形的面积大小。

|  |
| --- |
|  |

* 判断输出结果为何

|  |
| --- |
| public class Test {  public Test() {  Inner s1 = new Inner();  s1.a = 10;  Inner s2 = new Inner();  s2.a = 20;  Test.Inner s3 = new Test.Inner();  System.out.println(s3.a);  }  class Inner {  public int a = 5;  }  public static void main(String[] args) {  Test t = new Test();  Inner r = t.new Inner();  System.out.println(r.a);  }  } |

* 选择题

|  |
| --- |
| class Test {  public static void main(String[] args) {  String foo = args[1];  String bar = args[2];  String baz = args[3];  }  } |

d:\>java Test Red Green Blue

what is the value of baz?

A. baz has value of ""

B. baz has value of null

C. baz has value of "Red"

D. baz has value of "Blue"

E. baz has value of "Green"

F. the code does not compile

G. the program throw an exception

* 1)下面关于接口的说法中不正确的是（）。

A．接口中所有的方法都是抽象的

B．接口中所有的方法都是public访问权限

C．子接口继承父接口所用的关键字是implements

D．接口是Java中的特殊类，包含常量和抽象方法

* 2)Java语言接口间的继承关系是（）。

A．单继承 B．多重继承 C．不能继承 D．不一定

* 3)一个类实现接口的情况是（）。

A．一次可以实现多个接口 B．一次只能实现一个接口

C．不能实现接口 D．不一定

* 1)\_\_\_\_\_是声明接口的关键字，可以把它看成一个特殊类。接口中的数据成员默认的修饰符是\_\_\_\_\_，接口中的成员方法默认的修饰符是\_\_\_\_\_。
* 2)如果实现某接口的类不是abstract的抽象类，则在类的定义部分必须该接口的所有抽象方法；如果实现某接口的类是abstract的抽象类，则它可以该接口所有的方法。但是对于这个抽象类任何一个非抽象的子类而言，它们父类所实现的接口中的所有抽象方法以及自身所实现接口中的抽象方法都必须有实在的.
* 补全代码

|  |
| --- |
| interface Inter {  void show(int a, int b);  void func();  } |
| class Demo {  public static void main(String[] args) {  // 补足代码；调用两个函数，要求用匿名内部类  Inter in = new Inter() {  public void show(int a, int b) {  }  public void func() {  }  };  in.show(4, 5);  in.func();  }  } |

* abstract class 和interface 有什么区别?

答：声明方法的存在而不去实现它的类被叫做抽象类（abstract class），它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建abstract 类的实例。然而可以创建一个变量，其类型是一个抽象类，并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法

接口（interface）是抽象类的变体。在接口中，所有方法都是抽象的。多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，没有一个有程序体。接口只可以定义static final成员变量。接口的实现与子类相似，除了该实现类不能从接口定义中继承行为。当类实现特殊接口时，它定义（即将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口.

* 接口是否可继承接口? 抽象类是否可实现(implements)接口? 抽象类是否可继承实体类(concrete class)?

答案是：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承实体类，但实体类必须不能是如下两种情况之一：

1，final修饰符修饰的类是不能的

2，如果此实体类有且仅有私有的构造函数也是不能的。

* 是否能通过编译？

|  |
| --- |
| interface A{  int x = 0;  } |
| class B{  int x =1;  } |
| class C extends B implements A {  public void pX(){  System.out.println(x); //super.x A.x  }  public static void main(String[] args) {  new C().pX();  }  } |

答案：错误。在编译时会发生错误(错误描述不同的JVM有不同的信息，意思就是未明确的x调用，两个x都匹配（就象在同时import java.util和java.sql两个包时直接声明Date一样）。对于父类的变量,可以用super.x来明确，而接口的属性默认隐含为 public static final.所以可以通过A.x来明确。

* 写出程序结果

|  |
| --- |
| interface A{  } |
| class B implements A{  public String func(){  return "func";  }  } |
| class Demo{  public static void main(String[] args){  A a=new B();  System.out.println(a.func());  }  } |

编译失败：因为A接口中并未定义func方法。本题主要考察接口、包、继承、封装等问题。

* 编一个程序，包含以下文件

（1）Shape.java文件，在该文件中定义接口类Shape，该接口在shape包中。属性：PI。接口：求面积的方法area()。

|  |
| --- |
|  |

（2）Circle.java文件，在该文件中定义圆类Circle，该类在circle包中，实现Shape接口类。属性：圆半径radius。方法：构造器；实现求面积方法area()；求周长方法perimeter()。

|  |
| --- |
|  |

（3）Cylinder.java文件，在该文件中定义圆柱体类Cylinder，该类在cylinder包中，继承圆类。属性：圆柱体高度height。方法：构造器；求表面积方法area()；求体积方法volume()。

|  |
| --- |
|  |

（4）X5\_3\_6.java文件，在该文件中定义主类X5\_3\_6，该类在默认包中，其中包含主方法main()，在主方法中创建两个圆类对象cir1和cir2，具体尺寸自己确定，并显示圆的面积和周长；再创建两个圆柱体类的对象cy1和cy2，具体尺寸自己确定，然后分别显示圆柱体cy1和cy2的底圆的面积和周长以及它们各自的体积和表面积。

|  |
| --- |
|  |

* 鸭嘴兽属于脊椎动物中的哺乳动物，太阳花属于植物中的种子植物，而鸭嘴兽和太阳花都属于生物。完成如下要求

1）设计一系列的接口来表示这些规范：生物动物脊椎动物哺乳动物；生物植物种子植物，并且按照箭头要求来继承；

|  |
| --- |
|  |

2）生物都有呼吸方式，只有一个方法声明：void respirations();设计两个类，分别是鸭嘴兽和太阳花，分别实现生物接口的respirations方法，执行方法可以分别输出：“鸭嘴兽是动物，需要呼吸氧气”，“太阳花是植物，可以吸收二氧化碳，呼出氧气”；

|  |
| --- |
|  |

3）动物接口拥有方法声明：void run(),鸭嘴兽类可以实现方法，输出信息：“鸭嘴兽只有两只脚，它可以用两只脚奔跑”

|  |
| --- |
|  |

4）植物接口都有繁殖方法声明 void reproduction(),太阳花在实现reproduction方法时输出信息“太阳花可以用种子繁殖，也可以用枝条繁殖”

|  |
| --- |
|  |

5)声明一个Test类作为测试，执行上述声明的所有方法；

|  |
| --- |
|  |

* 请使用接口编码实现如下需求：乐器（Instrument）分为：钢琴(Piano)、小提琴(Violin).各种乐器的弹奏（ play ）方法各不相同。

|  |
| --- |
|  |

编写一个测试类InstrumentTest，要求：编写方法testPlay，对各种乐器进行弹奏测试。要依据乐器的不同，进行相应的弹奏。在main方法中创建不同的乐器对象，通过testPlay的弹奏测试方法进行测试。

|  |
| --- |
|  |

* 按要求实现下列问题：

动物类Animal包含了抽象方法 abstract void shout()；

|  |
| --- |
|  |

Cat类继承了Animal，并实现方法shout，打印“猫会喵喵叫”

|  |
| --- |
|  |

Dog类继承了Animal，并实现方法shout，打印“狗会汪汪叫”

|  |
| --- |
|  |

在测试类中实例化对象Animal a1 =new Cat(),并调用a1的shout方法

|  |
| --- |
|  |

在测试类中实例化对象Animal a2 =new Dog(),并调用a2的shout方法

|  |
| --- |
|  |

* 如下代码是否可以编译通过，如果可以，运行结果是什么？

|  |
| --- |
| interface A{  int x = 0;  } |
| class B{  int x = 1;  } |
| class C extends B implements A{  public void printX(){  System.out.println(x);  }  public static void main(String[] args) {  new C().printX();  }  } |
| 答案：编译错误  System.out.println(x);报错，x有歧义 |

* 选择题

|  |
| --- |
| public class OuterClass {  private double d1 = 1.0;  //insert code here  } |

You need to insert an inner class declaration at line 3. Which two inner class declarations are valid?(Choose two.)

|  |
| --- |
| A. class InnerOne{  public static double methoda() {return d1;}  } |
| B. public class InnerOne{  static double methoda() {return d1;}  } |
| C. private class InnerOne{  double methoda() {return d1;}  } |
| D. static class InnerOne{  protected double methoda() {return d1;}  } |
| E. abstract class InnerOne{  public abstract double methoda();  } |

说明如下：

一.静态内部类可以有静态成员，而非静态内部类则不能有静态成员。 故 A、B 错

二.静态内部类的非静态成员可以访问外部类的静态变量，而不可访问外部类的非静态变量；return d1 出错。 故 D 错

三.非静态内部类的非静态成员可以访问外部类的非静态变量。故 C 正确

四.答案为C、E

* Anonymous Inner Class (匿名内部类) 是否可以extends(继承)其它类，是否可以implements(实现)interface(接口)

答：匿名的内部类是没有名字的内部类。不能extends(继承) 其它类，但一个内部类可以作为一个接口，由另一个内部类实现

* Static Nested Class 和 Inner Class的不同

答：Nested Class （一般是C++的说法），Inner Class (一般是JAVA的说法)。Java内部类与C++嵌套类最大的不同就在于是否有指向外部的引用上。

注：静态内部类（Inner Class）意味着创建一个static内部类的对象，不需要一个外部类对象，2不能从一个static内部类的一个对象访问一个外部类对象

* 写出程序结果

|  |
| --- |
| class TD {  int y = 6;  class Inner {  static int y = 3;  void show() {  System.out.println(y);  }  }  } |
| class TC {  public static void main(String[] args) {  TD.Inner ti = new TD().new Inner();  ti.show();  }  } |

编译失败，非静态内部类中不可以定义静态成员。

内部类中如果定义了静态成员，该内部类必须被静态修饰。

* 补足代码

|  |
| --- |
| interface Test {  void func();  } |
| class Demo {  public static void main(String[] args) {  // 补足代码；(匿名内部类)  // new Demo().show(new Test() {  // public void func() {  //  // }  // });  }  void show(Test t) {  t.func();  }  } |

* 判断题

1)内部类（可以/不可以）了解外部类

2)静态内部类用修饰。

3)外部类（可以/不可以）通过类名访问静态内部类里的静态成员（变量和方法）

4)外部类（可以/不可以）通过类名访问非静态内部类里的成员（变量和方法）

5)非静态内部类可以访问外部类的成员，也能访问外部类的成员

6)静态内部类可以访问外部类的成员，不能访问外部类的成员

7)非静态内部类（可以/不可以）用static修饰她的成员（变量和方法）

* 纠错

|  |
| --- |
| public class First {  Privateinta;  static int b = 5;  public class Contents{  publicintc;  Staticintd;  publicvoidf(){  System.out.println(a);  System.out.println(b);  }  }  staticclassTitles{  publicinte;  Staticintf;  publicvoidshow(){  System.out.println(a);  System.out.println(b);  }  }  publicvoidgetInnerClassDate(){  System.out.println(c);//  System.out.println(d);//  System.out.println(e);//  System.out.println(f);//  }  publicstaticvoidmain(String[] args) {    }  } |

1. 这个类定义有问题吗？如果有请找出。
2. 在main方法中如何获得两个内部类的实例对象？并且访问内部类的方法，请写出代码

* 有一个Car类，有属性temperature（温度），车内有Air（空调），有吹风的功能flow，Air会监视车内的温度，如果温度超过40度则吹冷气。如果温度低于0度则吹暖气，如果在这之间则关掉空调。实例化具有不同温度的Car对象，调用空调的flow方法，测试空调吹的风是否正确

|  |
| --- |
|  |

* 编一个类A，在类中定义局部类B，B中有一个常量name，有一个方法show()打印常量name。进行测试

|  |
| --- |
|  |

* 设计公司类（Lenovo）,有一个接口类Works,里面有接口work()，下面有2个部门（开发部和销售部）,使用内部类分别实现开发部和销售部的工作。

|  |
| --- |
|  |

* 计算器具有work方法，功能是运算，时钟同样具有work方法，功能是计时，有一个手机类Cellphone，同时需要拥有运算和计时功能，通过内部类实现多继承效果。

|  |
| --- |
|  |

* 有一个铃声接口类Bell，有一个接口sound，有一个手机类Cellphone，具有闹钟功能alarmclock，参数是Bell类型，测试手机类的闹钟功能，通过匿名内部类作为参数，打印：懒猪起床了。

|  |
| --- |
|  |

* abstract 的method 是否可同时是static，是否可同时是native，是否可同时是synchronized？

答：都不能

* 抽象类和抽象方法之间的关系

定义抽象类和抽象方法的关键字是（）。抽象类中（）（可以/不可以）有抽象方法，（）（可以/不可以）有普通方法（）（可以/不可以）有属性；一个类中定义了抽象方法，那这个类（）（必须/不必须）用abstract修饰，即抽象类。

* 判断题

|  |
| --- |
| abstract class Name {  private String name;  public abstract boolean isStupidName(String name) {}  } |

这有何错误?

答案: 错。abstract method必须以分号结尾，且不带花括号。

* 是否可以通过编译？

|  |
| --- |
| abstract class Something {  private abstract String doSomething ();  } |

答案: 错。abstract的methods不能以private修饰。abstract的methods就是让子类implement(实现)具体细节的，怎么可以用private把abstract method封锁起来呢? (同理，abstract method前不能加final)。

* 定义一个抽象类Person，有name,age,sex三个属性，创建构造函数，给这三个属性赋值，重写打印描述信息方法，打印三个属性值；这个类中有两个抽象方法work和hello；定义两个子类Teacher，Student；教师的work实现是：教书育人，hello实现是：“同学好”；学生的work实现是：认真学习，hello实现是“老师好”。

|  |
| --- |
|  |

* 定义Shape抽象类，包含私有属性color，创建构造器为color赋值；包含计算周长的方法celPerimeter();定义子类Triangle，包含三边；定义子类Circle，包含半径radius；子类分别实现父类的计算周长功能。

|  |
| --- |
|  |

* 叙述工厂模式的设计理念
* 有一个交通工具接口类Vehicles，有work接口，有Horse类和Boat类分别实现Vehicles，创建交通工具工厂类，有两个方法分别获得交通工具Horse和Boat；有Person类，有name和Vehicles属性，在构造器中赋值，实例化“唐僧”，一般情况下用Horse作为交通工具，遇到大河时用Boat作为交通工具。

|  |
| --- |
|  |
|  |
|  |

* 有一个农场公司，专门向市场销售各类水果，主要包括下列水果：葡萄　Grape，草莓　Stuawberry，苹果　Apple；有一个Fruit接口，包含grow生长，harvest收获和plant方法，葡萄、草莓和苹果都实现了这个接口；有一个农场园丁类FruitGardener，具有采摘水果的方法getFruit(String fruitName)；这时有人来果园玩，和园丁说介绍一下你的各种水果(创建一个People类进行测试，分别通过农场园丁类得到各种水果)。

|  |
| --- |
|  |
|  |
|  |

* 有一个显示器接口Graphoscope，具有display方法，有两个类：台式显示器和液晶显示器都实现显示器接口，有一个显示器生产厂家能够生产这两种显示器；有Computer类，具有Graphoscope属性，生产两台电脑，分别配置台式显示器和液晶显示器。

|  |
| --- |
|  |
|  |
|  |

* 是否可以从一个static方法内部发出对非static方法的调用？

答：不可以,如果其中包含对象的method()；不能保证对象初始化.

* 写一个Singleton出来

|  |
| --- |
|  |
|  |

Singleton模式主要作用是保证在Java应用程序中，一个类Class只有一个实例存在。一般Singleton模式通常有几种形式:第一种形式: 定义一个类，它的构造函数为private的，它有一个static的private的该类变量，在类初始化时实例化，通过一个public的getInstance方法获取对它的引用,继而调用其中的方法。定义一个类，它的构造函数为private的，所有方法为static的。一般认为第一种形式要更加安全些

* 编译是否通过？

|  |
| --- |
| public class Something {  public static void main(String[] args) {  Something s = new Something();  System.out.println("s.doSomething() returns " + doSomething());  }  public String doSomething() {  return "Do something ...";  }  } |

看上去很完美。

答案: 错。看上去在main里call doSomething没有什么问题，毕竟两个methods都在同一个class里。但仔细看，main是static的。static method不能直接call non-static methods。可改成"System.out.println("s.doSomething() returns " + s.doSomething());"。同理，static method不能访问non-static instant variable。

* 写出程序结果

|  |
| --- |
| public class Demo {  private static int j = 0;  private static boolean methodB(int k) {  j += k;  return true;  }  public static void methodA(int i) {  boolean b;  b = i < 10 | methodB(4);  b = i < 10 || methodB(8);  }  public static void main(String args[]) {  methodA(0);  System.out.println(j);  }  } |

* 谈谈final, finally, finalize的区别(后面异常中讲finally)

答：final—修饰符（关键字）如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 abstract的，又被声明为final的。将变量或方法声明为final，可以保证它们在使用中不被改变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重载

finally—再异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）

finalize—方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的

* 如下程序是否可通过编译？

|  |
| --- |
| public class Something {  public int addOne(final int x) {  return ++x;  }  } |

这个比较明显。

答案: 错。int x被修饰成final，意味着x不能在addOne method中被修改。

* 如下程序是否可通过编译？

|  |
| --- |
| public class Something {  public static void main(String[] args) {  Other o = new Other();  new Something().addOne(o);  }  public void addOne(final Other o) {  o.i++;  }  } |
| class Other {  public int i;  } |

和上面的很相似，都是关于final的问题，这有错吗?

答案: 正确。在addOne method中，参数o被修饰成final。如果在addOne method里我们修改了o的reference(比如: o = new Other();)，那么如同上例这题也是错的。但这里修改的是o的member vairable(成员变量)，而o的reference并没有改变。

* 是否可以通过编译？

|  |
| --- |
| class Something {  int i;  public void doSomething() {  System.out.println("i = " + i);  }  } |

答案: 正确。输出的是"i = 0"。int i属於instant variable (实例变量，或叫成员变量)。instant variable有default value。

int的default value是0。

* 是否可以通过编译？

|  |
| --- |
| class Something {  final int i;  public void doSomething() {  System.out.println("i = " + i);  }  } |

接上题

和上面一题只有一个地方不同，就是多了一个final。这难道就错了吗?

答案: 错。final int i是个final的instant variable (实例变量，或叫成员变量)。final的instant variable没有default value，必须在constructor (构造器)结束之前被赋予一个明确的值。可以修改为"final int i = 0;"。

* 以下代码的运行结果是？

|  |
| --- |
| public class Test {  static int x, y, z;  static {  int x = 5;  x--;  }  static {  x--;  }  public static void main(String[] args) {  System.out.println("x=" + x);  z--;  method();  System.out.println("result:" + (z + y + ++z));  }  public static void method() {  y = z++ + ++z;  }  } |
| 答案：  x=-1  result:3 |