**类和对象**

A. 成员变量可以用 public default protected private final static修 饰

B. 构造器的返回值是实例，是隐式的，无需定义返回值类型，也不用 return，如果给构造器加上返回类型，不会编译错误，但Java会将 它当方法，它不再是构造器

C. 关于this

a.this关键字总是指向当前调用该方法的对象，一个方法访问本类 其它变量和方法，是可以不加this的，但this依然存在 b.因为在Java中，调用成员变量、方法时，主调是必需的 c.当在static方法中使用this，则无法指向合适的对象，因为static 方法不属于实例，属于类本身，所以static方法中不使用this， d.所以static方法也不能访问非static成员，因为非static成员需 要实例来调用，static方法无法给它提供一个该类的实例 tips：关于static方法不能访问非static成员： 非static成员需要一个实例来调用，如果在非static方法中，这 个实例就是调用该方法的当前实例，可以省略this；但在static 方法中，static方法不能属于某个实例，所以static方法中不存 在一个实例能用来访问非static成员 参照上面cd点

**构造器**

A. 构造器是创建对象的重要途径，创建对象时，系统为实例变量初 始化，默认基本类型设为0，0.0，\u0000(空格)，false，引用 型设为null，可以通过构造器来改变默认的初始化值

B. 当系统执行构造器执行体之前，已经创建了一个对象并进行默认 初始化，但还不能被外部程序访问，执行构造器之后会返回其引 用

C. 通常都把构造器设为public访问权限，除非在某些情况下需要限 制创建该类的对象

D.使用this调用重载构造器

当B类的代码完全包含了A类的代码时，可在B类中用this来 调用A类的代码，避免相同的代码重复书写，仅限重载构造器

class A(int a, int b){….}

class B(int a, int b, int c){ this(a,b); xxx = c }

tips：用this调用重载构造器只能在构造器中使用，而且必须作为 构造器的第一条语句

**方法**

A. 重载方法只跟参数类型与个数有关，与返回值类型，修饰符等无 关

B. 不推荐重载参数可变的方法

**变量**

A. 成员变量

a. 内存分配机制：

当一个类首次被使用来创建对象的时候，第一步系统会为该类的 类变量在堆内存中分配空间并赋初始值，接着第二步创建一个对 象，为该对象的实例变量在堆内存中分配空间，并把对象引用赋 给栈内存中的指定变量。 创建第二个对象的时候，就不用为类变量分配空间了，直接给实 例变量分配空间之后返回引用，也就是直接执行第二步

b. 类变量是一个类自己的，是这个类的所有实例共享的，可以直接 通过类名来访问，也可以通过实例来访问，最好用类名访问

B. 局部变量

局部变量放在栈内存中，当被赋初值时系统才给它分配内存，随着 方法体或代码块(例如for循环)的结束而消失，不用系统回收

C. 变量使用规范

大部分情况下可以使用成员变量，但这使变量的作用域扩大到类或 对象的存在范围，这将增大与扩大变量的生存时间与作用域，导致 更大的内存消耗，不利于提高程序的内聚性

所以如果只需要一个变量在方法体或代码块里起作用，就在它里面 定义，在有需要的情况下才定义为成员变量，

**隐藏与封装**

进行程序设计时。尽量避免一个模块直接操作和访问另一个模块的 数据，模块设计追求高内聚(尽可能把模块的内部数据，和功能实现 细节隐藏在模块内部独立完成，不允许外部直接干预)，低耦合(仅暴 露少量方法给外部使用)

private -> default -> protecte d -> public 级别由小到大

A. 控制符使用原则

a. 绝大多数成员变量都应该用private修饰，只有一些static， 或类似全局变量的成员变量才可考虑用public

b. 如果父类中有方法仅希望给子类重写，不希望被外界直接用 protectd修饰

c. 希望暴露给其他类自由调用的方法应该用public修饰，例 如构造器

d. 外部类不能用private跟protected修饰，因为它不在任何类 内部，也就不存在其所在类的子类，可用public或default

B. package, import, import static

a. 父包要使用子包中的类，需用子包的全名，不能省略父包名。

例如：父包名.子包名.类名 = new 父包名.子包名.类名()

b. 一个java源文件只能有一个package语句，但可以有多个 import语句，多个import语句用来导入多个包层次下的类。

1. import 包名1.\* 只能导入包名1的所有类，但包名1的子 包里的类不会被导入。

2. 同一个包里的类可以直接用不需加包前缀

如果要使用不同包的其他类，应该用类的全名，如果已经用 import语句导入，用的时候可以直接写类名而不用写该类 的全名(即加包前缀) 这两点可以用文件夹目录树状结构来帮助理解

c. import static可以导入某一个包中某一个类的单个或全部静态 变量和静态方法(仅限静态)，与普通import语句一样可以用\* 导入全部类成员，例如导入java.lang.System类中的全部静态 成员变量

import static java.lang.System.\* //可以简化代码：

out.println(….);

tips：一句话概括import跟import static的作用：import 可以省略包名，import static连类名都可以省略

**继承**

A. 每个类最多只能有一个直接父类，但可以有无限多个间接父类

B. 如果定义一个类时未显式指定其直接父类，则默认扩展

java.lang.Object类

C. 子类重写父类方法，子类方法访问权限要比父类大或相等，这是 为了确保可以使用父类实例的地方都可以使用子类实例,两种方法 必须都是类方法或实例方法。重写方法的类型要与父类的类型一 致或是其子类型

D. 子类重写了父类的方法后，无法直接访问父类被覆盖的方法，但 可以在子类方法中调用父类中被覆盖的方法，可用super(被覆盖 的是实例方法)，或父类名(被覆盖的是类方法)作为主调

E. 子类无法重写父类带private权限的方法，若子类中定义了一个 与父类private方法名，参数，返回值相同的方法，也不是重写

而是定义了一个新方法

F. super

a. super用来限定当前对象调用它从父类继承的实例成员(类成员 用父类名调用)，因此在static方法中也像this一样不能用 super

b. 若子类定义了与父类同名的实例变量，则称父类实例变量被隐 藏，也要用super来调用父类实例变量

c. 如果在某个方法中访问变量a，而没有显示指定调用者，则系 统查找a的顺序为：

1. 在当前方法中查找是否有名为a的局部变量

2. 在当前类中查找是否有名为a的成员变量

3. 查找a的直接父类中是否有名为a的成员变量，依次上溯到 java.lang.Object类，若最终找不到，则编译错误

G. 当创建一个子类对象时，系统会为它和它所有直接跟间接父类的 实例变量分配内存

H. 调用父类构造器

a. 子类不能获得父类构造器，但可以调用父类构造器中的代码， 方法类似于前面调用重载构造器，那个用this，这个用super 由于super调用要出现在子类构造器执行体第一行，所以this 调用跟super调用不能同时出现

b. 不管子类是否使用super来调用父类构造器代码，子类构造器 都会调用父类构造器一次，分这几种情况：

1. 子类构造器执行体中没有super，也没有this，系统将会在 执行子类构造器之前，隐式调用父类无参数的构造器

2. 子类构造器执行体第一行代码使用this显示调用本类中重 载的构造器，系统将根据this调用里的实参列表调用本类 中的另一个构造器，执行本类中的另一个构造器时就会调用 父类构造器

3. 子类构造器执行体第一行代码使用super显式调用父类构造 器，系统将根据super调用里的实参列表调用父类对应的 构造器

例子如下：

public class Father {

Father(){

out.println("i am father class");

}

}

class Son extends Father{

Son(int a){

out.println("i am son1 " + a);

}

Son(int a, int b){

this(a);

out.println(a + " i am son2 " + b);

}

}

class Grandson extends Son{

Grandson(){

super(1, 2);

out.println("i am Grandson");

}

}

public class M {

public static void main(String[] args){

Grandson gs = new Grandson();

}

}

这里输出的结果是：

i am father class

i am son1 1

1 i am son2 2

i am Grandson

I. 方法重写发生在通过继承而相关的不同类中，方法重载可以发生 在同一个类中，也可以发生在通过继承而相关的不同类中

**多态**

A. 声明类型实例在写代码时只能调用声明类里的变量、方法，不能 调用实际类里的方法，但运行时实例调用的则是父类被实际类覆 盖的方法，即是实际类(子类)的方法

B. 虽然调用的这个方法名都是父类里的方法名，但实际上运行时用 的是具体实际类(子类)对应的方法，而不是父类中被覆盖的方法

C. 因为一个父类的方法可能被多个子类重写，所以不同实例调用的 同一个方法可能对应多个子类的方法。即相同类型的变量，调用 同一个方法时呈现出多种不同的行为特征，这就体现了多态

D. 变量没有多态，通过对象来访问其包含的实例变量时，系统总是 试图访问其声明时类型所定义的成员变量，而不是运行时类型所 定义的成员变量

E. 关于子类父类对象相互转换 假设A类是B类和C类的父类

a.将子类对象赋给父类引用变量时被称为上转型，这总可以成功

例如: A f1 = new A(); B s1 = new B(); A f2; f1 = s1; f2 =s1; 这种转型只是表明这个引用变量编译时的类型是父类，但实际执 行的是子类的方法

b.而将父类对象赋给子类引用变量时需要强制类型转换，而且这 个对象必须实际上是子类对象才行，即声明类型是父类，实际类 型是子类

例如：A f1 = new B(); B s1 = (B)f1;

如果实际类型是父类对象或者是父类的另一个子类则不行

例如：A f1 = new A(); B s1 = (B)f1; || A f1 = new C(); B s1 = (B)f1;

Father f1 = new Father();

Father f2 = new Dau();

Father f3 = new Son();

Son s1 = (Son)f1;

Son s2 = (Son)f2;

Son s3 = (Son)f3;

Son s4 = new Son();

f1 = s4;

以上8行代码体现了a, b中的几种情况，4, 5行会出现异常

总的来说就是：

1) 实际类型是子类的对象总是可以赋值给父类型的变量(上转型)

2) 只有实际类型是子类的对象(尽管其声明类型为父类型)才可以 赋值给子类型的变量

再来个例子：

public class Test {

void testVoice(Animal a){

a.voice();

if(a instanceof Cat ){

Cat a1 = (Cat)a;

a1.catchMouses();

}

}

public static void main(String[] args){

Test te = new Test();

Animal c = new Cat();

Animal d = new Dog();

Animal t = new Tiger();

te.testVoice(c);

te.testVoice(d);

te.testVoice(t);

}

}

理解了E点就能更容易地理解F点(instanceof)

F. instanceof

a.instanceof前一个操作数通常是引用类型变量，后一个操作数 通常是类或接口，前面的变量的编译类型与后面的类必须相同或 有继承关系，否则会引起编译错误

b.若前面的对象是后面的类或是其子类，则返回true，否则false

c.区分三种情况

使用instanceof在写代码(编译)时对前操作数是用其声明类型， 而在运行时(即返回true or false) 是用其实际类型

1)前后操作数不符合要求引起的编译错误

上面说到编译时是用前操作数的声明类型，所以当前操作数的 声明类型与后操作数类型不符合要求(即不相同或没有继承关 系)，它会出现编译错误，此时无关返回值true or false

只有通过编译，才会有下面两种情况

2)返回true

若前操作数的实际类型与后面的类相同或是其子类，返回true

3)否则返回false

Father f1 = new Son();

Father f2 = new Father();

Dau d1 = new Dau();

System.out.println(d1 instanceof Son);

System.out.println(f1 instanceof Son);

System.out.println(f1 instanceof Father);

System.out.println(f1 instanceof Dau);

System.out.println(f2 instanceof Son);

第4行编译错误，第5,6行返回true，第7,8行返回false