# 面向对象的特征

**1>**抽象：抽象是将一类对象的共同特征总结出来构造类的过程，包括数据抽象以及行为抽象。抽象只关注对象有哪些属性和行为，并不关注行为如何实现，以及行为的细节是什么

2>继承：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类(超类、基类)，得到继承信息的类被称为子类(派生类)。继承让变化的软件有了一定的延续性，同时继承也是封装程序中可变因素的重要手段。

3>封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的借口。面向对象的本质就是将现实世界描绘成一幅完全自制、封闭的对象。我们在类中写的方法就是对实现细节的一种封装。我们编写一个类就是对数据和数据操作的封装。可以说封装就是隐藏一切可以隐藏的东西，向外界提供最简单的编程接口。

4>多态性：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同一个引用对象调用相同的方法但是做了不同的事情。多态性分为运行时和编译时多态。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态可以理解为：当A系统访问B系统的服务时，B系统有多种提供服务的方式，但一切对于A系统来说都是透明的。方法重载(reload)实现的是编译时的多态性(也成为前绑定)，而方法重写(override)实现的是运行时的多态性(也成为后绑定)。运行时的多态是面向对象最精髓的东西，要实现多态需要两件事情：**1)** 方法重写(子类继承父类并重写父类中已有的或抽象的方法) **2)** 对象造型(用父类引用引用子类对象，这样同样的引用调用同样的方法就会根据子类型的不同而表现出不同的行为)

# 访问权限修饰符public private default protected的区别？

| **修饰符** | **当前类** | **同 包** | **子 类** | **其他包** |
| --- | --- | --- | --- | --- |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ | × |
| default | √ | √ | × | × |
| private | √ | × | × | × |

外部类的修饰符只能是public或者default,类的成员(包括内部类)的修饰符

可以是以上四种。

# String是基本的数据类型么？

不是.String是引用数据类型。java中的基本数据类型有：byte short int long double float boolean char，除了基本数据类型(primitive type)其他的都是引用数据类型(reference type),枚举类型也算是特殊的引用类型

# float f=3.4;是否正确？

数据类型等级：byte—short—char—int—long—float--double

不正确。3.4是双精度数，将双精度型(double)赋值给单精度(float)属于下转型(down-casting，也称窄化)会造成精度丢失，因此需要强制转换类型float=(float)3.4;或者写成float f=3.4F;

在定义double或者float时后面加F或者D是为了区分数据类型。

# **5、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？**

Short s1=1; s1=s1+1有错。因为1是int类型的数据,此时做运算会将s1 short类型隐式转换为int类型，而当把值赋值给s1的时候，int的等级比short高，需要强制类型转换。

而short s1=1; s1+=1;没错。因为它相当于s1=(short)(s1+1);已经做了隐式的强制类型转换

# **7、int和Integer有什么区别？**

Integer是int的包装类型(wrapper class) 自动装箱/拆箱操作

原始类型的包装类有

byte short char int long float double

Byte Short Character Integer Long Float Double

**public** **static** Integer valueOf(**int** i) {

**final** **int** offset = 128;

**if** (i >= -128 && i <= 127) { // must cache

**return** IntegerCache.cache[i + offset];

}

**return** **new** Integer(i);

}

# **8、&和&&的区别？**

这两个都是逻辑运算符,其中&还可以用作按位与，&&有段路功能,当条件表达式1不成立是就不在判断表达式2。

# 9:、javascript短路运算

var a = 5 && 6;

console.log(a); //返回的结果为 6 如果逻辑与运算符左边的布尔值转换后位true,那么返回右边的值(不管右边的值是真是假)

var a = false && 6;

console.log(a); //返回的结果为 false 如果逻辑与运算符左边的布尔值转换后位false,那么返回左边的值, 但是当逻辑与的左边为 null/NaN/undefined ，结果就会得到null/NaN/undefined。

var a = false || 6;

console.log(a); //返回的结果为 6 如果逻辑或运算符左边的值布尔转换后为false，那么返回右边的值（不管右边的值是真还是假）

var a = true || 6;

console.log(a); //返回的结果为 true 如果逻辑或运算符左边的值布尔转换后为true，那么返回左边的值，如果两个操作数都是是null（NaN/undefined），返回null（NaN/undefined）

# **10、解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法。**