# 2020 最新《BAT Java 必考面试题集》

## 1、面向对象的特征有哪些方面?

答：面向对象的特征主要有以下几个方面：

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，包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属性和行为，并不关注这些行为的细节是什么。

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）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段（如果不能理解请阅读阎宏博士的《Java 与模式》或《设计模式精解》中关于桥梁模式的部分）。

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接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编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隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

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。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编

译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当 A 系统访问 B 系统提供的服务时，B 系统有多种提供服务的方式，但一切对 A 系统来说都是透明的（就像电动剃须刀是 A 系统，它的供电系统是 B 系统，B 系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A 系统只会通过 B 类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：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（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

**2、访问修饰符 public,private,protected,以及不写（默认）时的区别?**

答：区别如下：

| 作用域 | 当前类 | 同包 | 子类 | 其他 |
| --- | --- | --- | --- | --- |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ | × |
| default | √ | √ | × | × |
| private | √ | × | × | × |

类的成员不写访问修饰时默认为 default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。

**3、String 是最基本的数据类型吗?**

答：不是。Java 中的基本数据类型只有 8 个：byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type）和枚举类型（enumeration type），剩下的都是引用类型（reference type）。

**4、float f=3.4;是否正确?**

答:不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换 float f =(float)3.4; 或者写成 **float f =3.4F**;。

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

答：对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1 += 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

**6、Java 有没有 goto?**

答：goto 是 Java 中的保留字，在目前版本的 Java 中没有使用。（根据 James Gosling（Java 之父）编写的《The Java Programming Language》一书的附录中给出了一个 Java 关键字列表，其中有 goto 和 const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉 C 语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

## 7、int 和 Integer 有什么区别?

答：Java 是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入不是对象的基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java 为每一个基本数据类型都引入了对应的包装类型（wrapper class），int 的包装类就是 Integer，从 JDK 1.5 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

Java 为每个原始类型提供了包装类型：

原始类型: boolean，char，byte，short，int，long，float，double

包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

```java
package com.lovo;

public class AutoUnboxingTest {

	public static void main(String[] args) {
		Integer a = new Integer(3);
		Integer b = 3;                          // 将3自动装箱成Integer类型
		int c = 3;
		System.out.println(a == b);     // false 两个引用没有引用同一对象
		System.out.println(a == c);     // true a自动拆箱成int类型再和c比较
	}
}
```

补充：最近还遇到一个面试题，也是和自动装箱和拆箱相关的，代码如下所示：

```java
public class Test03 {

	public static void main(String[] args) {
		Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;

		System.out.println(f1 == f2);
		System.out.println(f3 == f4);
	}
}
```

如果不明就里很容易认为两个输出要么都是 true 要么都是 false。首先需要注意的是 f1、f2、f3、f4 四个变量都是 Integer 对象，所以下面的==运算比较的不是值而是引用。装箱的本质是什么呢？当我们给一个 Integer 对象赋一个 int 值的时候，会调用 Integer 类的静态方法 valueOf，如果看看 valueOf 的源代码就知道发生了什么。

```java
public static Integer valueOf(int i) {
	if (i >= IntegerCache.low && i <= IntegerCache.high)
	    return IntegerCache.cache[i + (-IntegerCache.low)];
	return new Integer(i);
    }
```

IntegerCache 是 Integer 的内部类，其代码如下所示：

```
    private static class IntegerCache {
        static final int low = -128;
        static final int high;
        static final Integer cache[];

        static {
            // high value may be configured by property
            int h = 127;
            String integerCacheHighPropValue =
                sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");
            if (integerCacheHighPropValue != null) {
                try {
                    int i = parseInt(integerCacheHighPropValue);
                    i = Math.max(i, 127);
                    // Maximum array size is Integer.MAX_VALUE
                    h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
                } catch( NumberFormatException nfe) {
                    // If the property cannot be parsed into an int, ignore it.
                }
            }
            high = h;

            cache = new Integer[(high - low) + 1];
            int j = low;
            for(int k = 0; k < cache.length; k++)
                cache[k] = new Integer(j++);

            // range [-128, 127] must be interned (JLS7 5.1.7)
            assert IntegerCache.high >= 127;
        }

        private IntegerCache() {}
    }
```

简单的说，如果字面量的值在-128 到 127 之间，那么不会 new 新的 Integer 对象，而是直接引用常量池中的 Integer 对象，所以上面的面试题中 f1==f2 的结果是 true，而 f3==f4 的结果是 false。越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。

## 8、&和&&的区别?

答：&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是 true 整个表达式的值才是 true。&&之所以称为短路运算是因为，如果&

&左边的表达式的值是 false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是 null 而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的 equals 比较，否则会产生 NullPointerException 异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

补充：如果你熟悉 JavaScript，那你可能更能感受到短路运算的强大，想成为 JavaScript 的高手就先从玩转短路运算开始吧。

**9、解释内存中的栈（stack）、堆(heap)和静态存储区的用法。**

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的栈空间；而通过 new 关键字和构造器创建的对象放在堆空间；程序中的字面量（literal）如直接书写的 100、“hello”和常量都是放在静态存储区中。栈空间操作最快但是也很小，通常大量的对象都是放在堆空间，整个内存包括硬盘上的虚拟内存都可以被当成堆空间来使用。

String str = new String(“hello”);

上面的语句中 str 放在栈上，用 new 创建出来的字符串对象放在堆上，而“hello”这个字面量放在静态存储区。

补充：较新版本的 Java 中使用了一项叫“逃逸分析“的技术，可以将一些局部对象放在栈上以提升对象的操作性能。

**10、Math.round(11.5) 等于多少？Math.round(-11.5)等于多少?**

答：Math.round(11.5)的返回值是 12，Math.round(-11.5)的返回值是-11。四舍五入的原理是在参数上加 0.5 然后进行下取整。

**11、swtich 是否能作用在 byte 上，是否能作用在 long 上，是否能作用在 String 上?**

答：早期的 JDK 中，switch（expr）中，expr 可以是 byte、short、char、int。从 1.5 版开始，Java 中引入了枚举类型（enum），expr 也可以是枚举，从 JDK 1.7 版开始，还可以是字符串（String）。长整型（long）是不可以的。

**12、用最有效率的方法计算 2 乘以 8?**

答： 2 << 3（左移 3 位相当于乘以 2 的 3 次方，右移 3 位相当于除以 2 的 3 次方）。

补充：我们为编写的类重写 hashCode 方法时，可能会看到如下所示的代码，其实我们不太理解为什么要使用这样的乘法运算来产生哈希码（散列码），而且为什么这个数是个素数，为什么通常选择 31 这个数？前两个问题的答案你可以自己百度一下，选择 31 是因为可以用移位和减法运算来代替乘法，从而得到更好的性能。说到这里你可能已经想到了：31 * num <==> (num << 5) - num，左移 5 位相当于乘以 2 的 5 次方（32）再减去自身就相当于乘以 31。现在的 VM 都能自动完成这个优化。

```java
package com.loonstudio;

public class PhoneNumber {
	private int areaCode;
	private String prefix;
	private String lineNumber;

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + areaCode;
		result = prime * result
				+ ((lineNumber == null) ? 0 : lineNumber.hashCode());
		result = prime * result + ((prefix == null) ? 0 : prefix.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		PhoneNumber other = (PhoneNumber) obj;
		if (areaCode != other.areaCode)
			return false;
		if (lineNumber == null) {
			if (other.lineNumber != null)
				return false;
		} else if (!lineNumber.equals(other.lineNumber))
			return false;
		if (prefix == null) {
			if (other.prefix != null)
				return false;
		} else if (!prefix.equals(other.prefix))
			return false;
		return true;
	}

}
```

## 13、数组有没有 length()方法?String 有没有 length()方法?

答：数组没有 length()方法，有 length 的属性。String 有 length()方法。JavaScript 中，获得字符串的长度是通过 length 属性得到的，这一点容易和 Java 混淆。

**14、在 Java 中，如何跳出当前的多重嵌套循环?**

答：在最外层循环前加一个标记如 A，然后用 break A;可以跳出多重循环。（Java 中支持带标签的 break 和 continue 语句，作用有点类似于 C 和 C++中的 goto 语句，但是就像要避免使用 goto 一样，应该避免使用带标签的 break 和 continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

**15、构造器（constructor）是否可被重写（override）？**

答：构造器不能被继承，因此不能被重写，但可以被重载。

**16、两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对?**

答：不对，如果两个对象 x 和 y 满足 x.equals(y) == true，它们的哈希码（hash code）应当相同。Java 对于 eqauls 方法和 hashCode 方法是这样规定的：(1)如果两个对象相同（equals 方法返回 true），那么它们的 hashCode 值一定要相同；(2)如果两个对象的 hashCode 相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在 Set 集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

补充：关于 equals 和 hashCode 方法，很多 Java 程序都知道，但很多人也就是仅仅知道而已，在 Joshua Bloch 的大作《Effective Java》（很多软件公司，《Effective Java》、《Java 编程思想》以及《重构：改善既有代码质量》是 Java 程序员必看书籍）中是这样介绍 equals 方法的：首先 equals 方法必须满足自反性（x.equals(x)必须返回 true）、对称性（x.equals(y)返回 true 时，y.equals(x)也必须返回 true）、传递性（x.equals(y)和 y.equals(z)都返回 true 时，x.equals(z)也必须返回 true）和一致性（当 x 和 y 引用的对象信息没有被修改时，多次调用 x.equals(y)应该得到同样的返回值），而且对于任何非 null 值的引用 x，x.equals(null)必须返回 false。实现高质量的 equals 方法的诀窍包括：1. 使用==操作符检查“参数是否为这个对象的引用”；2. 使用 instanceof 操作符检查“参数是否为正确的类型”；3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；4. 编写完 equals 方法后，问自己它是否满足对称性、传递性、一致性；5. 重写 equals 时总是要重写 hashCode；6. 不要将 equals 方法参数中的 Object 对象替换为其他的类型，在重写时不要忘掉@Override 注解。

## 17、是否可以继承 String 类?

答：String 类是 final 类，不可以被继承。

补充：继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关联（HAS-A）而不是继承（IS-A）。

**18、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。对象的属性可以在被调用过程中被改变，但对象的引用是永远不会改变的。C++和 C#中可以通过传引用或传输出参数来改变传入的参数的值。

补充：Java 中没有传引用实在是非常的不方便，这一点在 Java 8 中仍然没有得到改进，正是如此在 Java 编写的代码中才会出现大量的 Wrapper 类（将需要通过方法调用修改的引用置于一个 Wrapper 类中，再将 Wrapper 对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从 C 和 C++转型为 Java 程序员的开发者无法容忍。

**19、String 和 StringBuilder、StringBuffer 的区别?**

答：Java 平台提供了两种类型的字符串：String 和 StringBuffer / StringBuilder，它们可以储存和操作字符串。其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的。而 StringBuffer 和 StringBuilder 类表示的字符串对象可以直接进行修改。StringBuilder 是 JDK 1.5 中引入的，它和 StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被 synchronized 修饰，因此它的效率也比 StringBuffer 略高。

补充 1：有一个面试题问：有没有哪种情况用+做字符串连接比调用 StringBuffer / StringBuilder 对象的 append 方法性能更好？如果连接后得到的字符串

在静态存储区中是早已存在的，那么用+做字符串连接是优于 StringBuffer / StringBuilder 的 append 方法的。

补充 2：下面也是一个面试题，问程序的输出，看看自己能不能说出正确答案。

```
package com.lovo;

public class StringEqualTest {

	public static void main(String[] args) {
		String a = "Programming";
		String b = new String("Programming");
		String c = "Program" + "ming";

		System.out.println(a == b);
		System.out.println(a == c);
		System.out.println(a.equals(b));
		System.out.println(a.equals(c));
		System.out.println(a.intern() == b.intern());
	}
}
```

**20、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分?**

答：方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

补充：华为的面试题中曾经问过这样一个问题：为什么不能根据返回类型来区分重载，说出你的答案吧！

## 21、描述一下 JVM 加载 class 文件的原理机制?

答：JVM 中类的装载是由类加载器（ClassLoader） 和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

补充：

1.由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接(验证、准备和解析)和初始化。类的加载是指把类的.class 文件中的数据读入到内存中，通常是创建一个字节数组读入.class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备(为静态变量分配内存并设置默认的初始值)和解析(将符号引用替换为直接引用)三个步骤。最后 JVM 对类进行初始化，包括：1 如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2 如果类中存在初始化语句，就依次执行这些初始化语句。

2.类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader 的子类）。从 JDK 1.2 开始，类加载过程采取了父亲委托机制(PDM)。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加

载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

a)Bootstrap：一般用本地代码实现，负责加载 JVM 基础核心类库（rt.jar）；

b)Extension：从 java.ext.dirs 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

c)System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

**22、char 型变量中能不能存贮一个中文汉字?为什么?**

答：char 类型可以存储一个中文汉字，因为 Java 中使用的编码是 Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个 char 类型占 2 个字节（16bit），所以放一个中文是没问题的。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM 内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和 OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务；对于 C 程序员来说，要完成这样的编码转换恐怕要依赖于 union（联合体/共用体）共享内存的特征来实现了。

## 23、抽象类（abstract class）和接口（interface）有什么异同?

答：抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是 private、默认、protected、public 的，而接口中的成员全都是 public 的。抽象类中可以定义成员变量，而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象类，而抽象类未必要有抽象方法。

## 24、静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同?

答：Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化，其语法看起来挺诡异的，如下所示。

```java
package com.lovo;

/**
 * 扑克类（一副扑克）
 * @author 貉昊
 *
 */
public class Poker {
	private static String[] suites = {"黑桃", "红桃", "草花", "方块"};
	private static int[] faces = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};

	private Card[] cards;

	/**
	 * 构造器
	 *
	 */
	public Poker() {
		cards = new Card[52];
		for(int i = 0; i < suites.length; i++) {
			for(int j = 0; j < faces.length; j++) {
				cards[i * 13 + j] = new Card(suites[i], faces[j]);
			}
		}
	}

	/**
	 * 洗牌 （随机乱序）
	 *
	 */
	public void shuffle() {
		for(int i = 0, len = cards.length; i < len; i++) {
			int index = (int) (Math.random() * len);
			Card temp = cards[index];
			cards[index] = cards[i];
			cards[i] = temp;
		}
	}

	/**
	 * 发牌
	 * @param index 发牌的位置
	 *
	 */
	public Card deal(int index) {
		return cards[index];
	}

	/**
	 * 卡片类（一张扑克）
	 * [内部类]
	 * @author 貉昊
	 *
	 */
	public class Card {
		private String suite;   // 花色
		private int face;              // 点数

		public Card(String suite, int face) {
			this.suite = suite;
			this.face = face;
		}

```

```
64              @Override
65              public String toString() {
66                      String faceStr = "";
67                      switch(face) {
68                      case 1: faceStr = "A"; break;
69                      case 11: faceStr = "J"; break;
70                      case 12: faceStr = "Q"; break;
71                      case 13: faceStr = "K"; break;
72                      default: faceStr = String.valueOf(face);
73                      }
74                      return suite + faceStr;
75              }
76      }
77  }
```

```
1   package com.lovo;
2
3   class PokerTest {
4
5           public static void main(String[] args) {
6                   Poker poker = new Poker();
7                   poker.shuffle();                        // 洗牌
8                   Poker.Card c1 = poker.deal(0);  // 发第一张牌
9                   // 对于非静态内部类Card
10                  // 只有通过其外部类Poker对象才能创建Card对象
11                  Poker.Card c2 = poker.new Card("红心", 1);      // 自己创建一张牌
12
13                  System.out.println(c1);         // 洗牌后的第一张
14                  System.out.println(c2);         // 打印：红心A
15          }
16  }
```

## 25、Java 中会存在内存泄漏吗，请简单描述。

答：理论上 Java 因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是 Java 被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被 GC 回收也会发生内存泄露。一个例子就是 Hibernate 的 Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象。下面的例子也展示了 Java 中发生内存泄露的情况：

```java
package com.lovo;

import java.util.Arrays;
import java.util.EmptyStackException;

public class MyStack<T> {
	private T[] elements;
	private int size = 0;

	private static final int INIT_CAPACITY = 16;

	public MyStack() {
		elements = (T[]) new Object[INIT_CAPACITY];
	}

	public void push(T elem) {
		ensureCapacity();
		elements[size++] = elem;
	}

	public T pop() {
		if(size == 0)
			throw new EmptyStackException();
		return elements[--size];
	}

	private void ensureCapacity() {
		if(elements.length == size) {
			elements = Arrays.copyOf(elements, 2 * size + 1);
		}
	}
}
```

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的 pop 方法却存在内存泄露的问题，当我们用 pop 方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象的过期引用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发 Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成 OutOfMemoryError。

**26、抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native），是否可同时被 synchronized 修饰?**

答：都不能。抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如 C 代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized 和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

**27、静态变量和实例变量的区别?**

答：静态变量是被 static 修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。在 Java 开发中，上下文类和工具类中通常会有大量的静态成员。

**28、是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用?**

答：不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，因此在调用静态方法时可能对象并没有被初始化。

**29、如何实现对象克隆?**

答：有两种方式：

1.实现 Cloneable 接口并重写 Object 类中的 clone()方法；

2.实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆，代码如下。

```java
package com.lovo;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class MyUtil {

	private MyUtil() {
		throw new AssertionError();
	}

	public static <T> T clone(T obj) throws Exception {
		ByteArrayOutputStream bout = new ByteArrayOutputStream();
		ObjectOutputStream oos = new ObjectOutputStream(bout);
		oos.writeObject(obj);

		ByteArrayInputStream bin = new ByteArrayInputStream(bout.toByteArray());
		ObjectInputStream ois = new ObjectInputStream(bin);
		return (T) ois.readObject();

		// 说明：调用ByteArrayInputStream或ByteArrayOutputStream对象的close方法没有任何意义
		// 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源
	}
}
```

注意：基于序列化和反序列化实现的克隆不仅仅是深度克隆，更重要的是通过泛型限定，可以检查出要克隆的对象是否支持序列化，这项检查是编译器完成的，不是在运行时抛出异常，这种是方案明显优于使用 Object 类的 clone 方法克隆对象。

## 30、GC 是什么？为什么要有 GC?

答：GC 是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：Sys

tem.gc() 或 Runtime.getRuntime().gc() ，但 JVM 可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在 Java 诞生初期，垃圾回收是 Java 最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今 Java 的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得 iOS 的系统比 Android 系统有更好的用户体验，其中一个深层次的原因就在于 Android 系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的 Java 进程既有栈又有堆。栈保存了原始型局部变量，堆保存了要创建的对象。Java 平台对堆内存回收和再利用的基本算法被称为标记和清除，但是 Java 对其进行了改进，采用“分代式垃圾收集”。这种方法会跟 Java 对象的生命周期将堆内存划分为不同的区域，在垃圾收集过程中，可能会将对象移动到不同区域：

- 伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。
- 幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。
- 终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐养园时，就会触

发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。

与垃圾回收相关的 JVM 参数：

- -Xms / -Xmx --- 堆的初始大小 / 堆的最大大小
- -Xmn --- 堆中年轻代的大小
- -XX:-DisableExplicitGC --- 让 System.gc()不产生任何作用
- -XX:+PrintGCDetail --- 打印 GC 的细节
- -XX:+PrintGCDateStamps --- 打印 GC 操作的时间戳

**31、String s=new String(“xyz”);创建了几个字符串对象?**

答：两个对象，一个是静态存储区的"xyz",一个是用 new 创建在堆上的对象。

**32、接口是否可继承（extends）接口? 抽象类是否可实现（implements）接口? 抽象类是否可继承具体类（concrete class）?**

答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承具体类，但前提是具体类必须有明确的构造函数。

**33、一个“.java”源文件中是否可以包含多个类（不是内部类）? 有什么限制?**

答：可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

**34、Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口?**

答：可以继承其他类或实现其他接口，在 Swing 编程中常用此方式来实现事件监听和回调。

**35、内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制?**

答：一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

**36、Java 中的 final 关键字有哪些用法?**

答：(1)修饰类：表示该类不能被继承；(2)修饰方法：表示方法不能被重写；(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

**37、指出下面程序的运行结果:**

```
class A{

    static{
        System.out.print("1");
    }

    public A(){
        System.out.print("2");
    }
}

class B extends A{

    static{
        System.out.print("a");
    }

    public B(){
        System.out.print("b");
    }
}

public class Hello{

    public static void main(String[] args){
        A ab = new B();
        ab = new B();
    }

}
```

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

## 38、数据类型之间的转换:

1)如何将字符串转换为基本数据类型?

2)如何将基本数据类型转换为字符串?

答:

1)调用基本数据类型对应的包装类中的方法 parseXXX(String)或 valueOf(String)即可返回相应基本类型;

2)一种方法是将基本数据类型与空字符串（”“）连接（+）即可获得其所对应的字符串；另一种方法是调用 String 类中的 valueOf(...)方法返回相应字符串

## 39、如何实现字符串的反转及替换?

答：方法很多，可以自己写实现也可以使用 String 或 StringBuffer / StringBuilder 中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

```
public static String reverse(String originStr) {
                if(originStr == null || originStr.length() <= 1)
                        return originStr;
                return reverse(originStr.substring(1)) + originStr.charAt(0);
        }
```

## 40、怎样将 GB2312 编码的字符串转换为 ISO-8859-1 编码的字符串?

答：代码如下所示:

String s1 = "你好";

String s2 = newString(s1.getBytes("GB2312"), "ISO-8859-1");

## 41、日期和时间：

1)如何取得年月日、小时分钟秒?

2)如何取得从 1970 年 1 月 1 日 0 时 0 分 0 秒到现在的毫秒数?

3)如何取得某月的最后一天?

4)如何格式化日期?

答：操作方法如下所示：

1)创建 java.util.Calendar 实例，调用其 get()方法传入不同的参数即可获得参数所对应的值

2)以下方法均可获得该毫秒数:

```
Calendar.getInstance().getTimeInMillis();
System.currentTimeMillis();
```

3)示例代码如下:

```
Calendar time = Calendar.getInstance();
time.getActualMaximum(Calendar.DAY_OF_MONTH);
```

4)利用 java.text.DataFormat 的子类 (如 SimpleDateFormat 类) 中的 format(Date)方法可将日期格式化。

## 42、打印昨天的当前时刻。

答:

```
public class YesterdayCurrent {
    public static void main(String[] args){
        Calendar cal = Calendar.getInstance();
        cal.add(Calendar.DATE, -1);
        System.out.println(cal.getTime());
    }
}
```

## 43、比较一下 Java 和 JavaSciprt。

答：JavaScript 与 Java 是两个公司开发的不同的两个产品。Java 是原 Sun 公司推出的面向对象的程序设计语言，特别适合于互联网应用程序开发；而 JavaScript 是 Netscape 公司的产品，为了扩展 Netscape 浏览器的功能而开发的

一种可以嵌入 Web 页面中运行的基于对象和事件驱动的解释性语言，它的前身是 LiveScript；而 Java 的前身是 Oak 语言。

下面对两种语言间的异同作如下比较：

1）基于对象和面向对象：Java 是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object-Based）和事件驱动（Event-Driven）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用；

2）解释和编译：Java 的源代码在执行之前，必须经过编译；JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行；

3）强类型变量和类型弱变量：Java 采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript 中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型；

4）代码格式不一样。

补充：上面列出的四点是原来所谓的标准答案中给出的。其实 Java 和 JavaScript 最重要的区别是一个是静态语言，一个是动态语言。目前的编程语言的发展趋势是函数式语言和动态语言。在 Java 中类（class）是一等公民，而 JavaScript 中函数（function）是一等公民。对于这种问题，在面试时还是用自己的语言回答会更加靠谱。 

## 44、什么时候用 assert?

答：assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后， assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统会报告一个 AssertionError。

断言用于调试目的：

```
assert(a > 0); // throws an AssertionError if a <= 0
```

断言可以有两种形式：

```
assert Expression1;

assert Expression1 : Expression2 ;
```

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。

断言在默认情况下是禁用的，要在编译时启用断言，需使用 source 1.4 标记：

```
javac -source 1.4 Test.java
```

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用-da 或者-disableassertions 标记。

要在系统类中启用断言，可使用-esa 或者-dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

## 45、Error 和 Exception 有什么区别?

答：Error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；Exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

补充：2005 年摩托罗拉的面试中曾经问过这么一个问题“If a process reports a stack overflow run-time error, what’s the most possible cause?”，给了四个选项 a. lack of memory; b. write on an invalid memory space; c. recursive function calling; d. array index out of boundary. Java 程序在运行时也可能会遭遇 StackOverflowError，这是一个错误无法恢复，只能重新修改代码了，这个面试题的答案是 c。如果写了不能迅速收敛的递归，则很有可能引发栈溢出的错误，如下所示：

```java
package com.lovo;

public class StackOverflowErrorTest {

	public static void main(String[] args) {
		main(null);
	}
}
```

因此，用递归编写程序时一定要牢记两点：1. 递归公式；2. 收敛条件（什么时候就不再递归而是回溯了）。

**46、try{}里有一个 return 语句，那么紧跟在这个 try 后的 finally{}里的 code 会不会被执行，什么时候被执行，在 return 前还是后?**

答：会执行，在方法返回调用者前执行。Java 允许在 finally 中改变返回值的做法是不好的，因为如果存在 finally 代码块，try 中的 return 语句不会立马返回调用者，而是记录下返回值待 finally 代码块执行完毕之后再向调用者返回其值，然后如果在 finally 中修改了返回值，这会对程序造成很大的困扰，C#中就从语法上规定不能做这样的事。

**47、Java 语言如何进行异常处理，关键字：throws、throw、try、catch、finally 分别如何使用?**

答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常处理是通过 5 个关键词来实现的：try、catch、throw、throws 和 finally。一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throw）一个

异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；try 用来指定一块预防所有“异常”的程序；catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws 用来标明一个成员函数可能抛出的各种“异常”；finally 为确保一段代码不管发生什么“异常”都被执行一段代码；可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写另一个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常”进行处理，栈就会展开，直到遇到有处理这种“异常”的 try 语句。

## 48、运行时异常与受检异常有何异同?

答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序设计无误，仍然可能因使用的问题而引发。Java 编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，神作《Effective Java》中对异常的使用给出了以下指导原则：

- 不要将异常处理用于正常的控制流（设计良好的 API 不应该强迫它的调用者为了正常的控制流而使用异常）
- 对可以恢复的情况使用受检异常，对编程错误使用运行时异常
- 避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）

- 优先使用标准的异常
- 每个方法抛出的异常都要有文档
- 保持异常的原子性
- 不要在 catch 中忽略掉捕获到的异常

**49、列出一些你常见的运行时异常?**

答:

ArithmeticException（算术异常）

ClassCastException （类转换异常）

IllegalArgumentException （非法参数异常）

IndexOutOfBoundsException （下表越界异常）

NullPointerException （空指针异常）

SecurityException （安全异常）

**50、final，finally，finalize 的区别?**

答：final：修饰符（关键字）有三种用法：如果一个类被声明为 final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 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，不能在子类中被重写。finally：通常放在 try...catch 的后面构

造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中。finalize：Object 类中定义的方法，Java 中允许使用 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销毁对象时调用的，通过重写 finalize() 方法可以整理系统资源或者执行其他清理工作。

VM 内存可简单分为三个区：

1、堆区（heap）：用于存放所有对象，是线程共享的（注：数组也属于对象）

2、栈区（stack）：用于存放基本数据类型的数据和对象的引用，是线程私有的（分为：虚拟机栈和本地方法栈）

3、方法区（method）：用于存放类信息、常量、静态变量、编译后的字节码等，是线程共享的（也被称为非堆，即 None-Heap）

Java 的垃圾回收器（GC）主要针对堆区