String 的值是不可变的，每次对String的操作都会生成新的String对象，不仅效率低，而且耗费大量内存空间

StringBuffer类和String类一样，也用来表示字符串，但是StringBuffer的内部实现方式和String不同，在进行字符串处理时，不生成新的对象，在内存使用上要优于String。  
  
StringBuffer 默认分配16字节长度的缓冲区，当字符串超过该大小时，会自动增加缓冲区长度，而不是生成新的对象。

StringBuffer类中的方法主要偏重于对于字符串的操作，例如追加、插入和删除等，这个也是StringBuffer类和String类的主要区别。实际开发中，如果需要对一个字符串进行频繁的修改，建议使用 StringBuffer。

字符串的”+“操作实际上也是先创建一个StringBuffer对象，然后调用append()方法将字符串片段拼接起来，最后调用toString()方法转换为字符串。  
  
这样看来，String的连接操作就比StringBuffer多出了一些附加操作，效率上必然会打折扣。  
  
但是，对于长度较小的字符串，”+“操作更加直观，更具可读性，有些时候可以稍微牺牲一下效率

StringBuffer的执行效率比String快上千倍，这个差异随着叠加次数的增加越来越明显所以，强烈建议在涉及大量字符串操作时使用StringBuffer。

StringBuilder类和StringBuffer类功能基本相似，方法也差不多，主要区别在于StringBuffer类的方法是多线程安全的，而StringBuilder不是线程安全的，相比而言，StringBuilder类会略微快一点。线程安全就是说多线程访问同一代码，不会产生不确定的结果

## 总结

**线程安全：**

* StringBuffer：线程安全
* StringBuilder：线程不安全

**速度：**  
一般情况下，速度从快到慢为 StringBuilder > StringBuffer > String，当然这是相对的，不是绝对的。  
  
**使用环境：**

* 操作少量的数据使用 String；
* 单线程操作大量数据使用 StringBuilder；
* 多线程操作大量数据使用 StringBuffer。

几点说明：

* 上面讲的是静态数组。静态数组一旦被声明，它的容量就固定了，不容改变。所以在声明数组时，一定要考虑数组的最大容量，防止容量不够的现象。
* 如果想在运行程序时改变容量，就需要用到数组列表(ArrayList，也称动态数组)或向量(Vector)。
* 正是由于静态数组容量固定的缺点，实际开发中使用频率不高，被 ArrayList 或 Vector 代替，因为实际开发中经常需要向数组中添加或删除元素，而它的容量不好预估。

一个类可以包含以下类型变量：

* 局部变量：在方法或者语句块中定义的变量被称为局部变量。变量声明和初始化都是在方法中，方法结束后，变量就会自动销毁。
* 成员变量：成员变量是定义在类中、方法体之外的变量。这种变量在创建对象的时候实例化（分配内存）。成员变量可以被类中的方法和特定类的语句访问。
* 类变量：类变量也声明在类中，方法体之外，但必须声明为static类型。static 也是修饰符的一种

说明：

* 构造方法不能被显示调用。
* 构造方法不能有返回值，因为没有变量来接收返回值。

在Java中，使用new关键字来创建对象，一般有以下三个步骤：

* 声明：声明一个对象，包括对象名称和对象类型。
* 实例化：使用关键字new来创建一个对象。
* 初始化：使用new创建对象时，会调用构造方法初始化对象。

例如：

1. Dog myDog; // 声明一个对象
2. myDog = **new** Dog("花花", 3); // 实例化

也可以在声明的同时进行初始化：

1. Dog myDog = **new** Dog("花花", 3);

声明为私有访问类型的变量只能通过类中公共的Getter/Setter方法被外部类访问。

## ****访问控制和继承****

请注意以下方法继承的规则：

父类中声明为public的方法在子类中也必须为public。

父类中声明为protected的方法在子类中要么声明为protected，要么声明为public。不能声明为private。

父类中默认修饰符声明的方法，能够在子类中声明为private。

父类中声明为private的方法，不能够被继承。

## 如何使用访问控制符

访问控制符可以让我们很方便的控制代码的权限：

* 当需要让自己编写的类被所有的其他类访问时，就可以将类的访问控制符声明为 public。
* 当需要让自己的类只能被自己的包中的类访问时，就可以省略访问控制符。
* 当需要控制一个类中的成员数据时，可以将这个类中的成员数据访问控制符设置为 public、protected，或者省略。

this 关键字用来表示当前对象本身，或当前类的一个实例，通过 this 可以调用本对象的所有方法和属性。

override是方法的重写，通常发生在子类与父类之中，指的是子类中定义了一个与父类返回值类型，参数类型完全相同的方法  
overload是方法的重载，通常在同一个类中，定义了一堆方法名相同，但返回值可能不同，参数也可能不同的方法，在Java中，同一个类中的多个方法可以有相同的名字，只要它们的参数列表不同就可以，这被称为方法重载

****override（重写，覆盖）****  
1、**方法名、参数、返回值相同。**  
2、子类方法不能缩小父类方法的访问权限。  
3、子类方法不能抛出比父类方法更多的异常(但子类方法可以不抛出异常)。  
4、**存在于父类和子类之间。**  
5、方法被定义为final不能被重写。  
  
****overload（重载，过载）****1、**参数类型、个数、顺序至少有一个不相同。**   
**2、不能重载只有返回值不同的方法名。  
3、存在于父类和子类、同类中。**

* 声明为final的方法不能被重写。
* 声明为static的方法不能被重载，但是能够被再次声明。

ava 1.5 之后可以自动拆箱装箱，也就是在进行基本数据类型和对应的包装类转换时，系统将自动进行

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，不能重写。  
  
  finally—在异常处理时提供 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）。finally创建一个代码块。该代码块在一个try/catch 块完成之后另一个try/catch出现之前执行。finally块无论有没有异常抛出都会执行。  
  
  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。

* 构造方法不能被继承。

super 关键字的功能：

* 调用父类中声明为 private 的变量。
* 点取已经覆盖了的方法。
* 作为方法名表示父类构造方法。

值得注意的是：

* 在构造方法中调用另一个构造方法，调用动作必须置于最起始的位置。
* 不能在构造方法以外的任何方法内调用构造方法。
* 在一个构造方法内只能调用一个构造方法。
* 最后注意 super 与 this 的区别：super 不是一个对象的引用，不能将 super 赋值给另一个对象变量，它只是一个指示编译器调用父类方法的特殊关键字。

多态存在的三个必要条件：要有继承、要有重写、父类变量引用子类对象

instanceof 运算符用来判断一个变量所引用的对象的实际类型，注意是它引用的对象的类型，不是变量的类型

可以看出，如果变量引用的是当前类或它的子类的实例，instanceof 返回 true，否则返回 false

静态变量和静态方法能够通过类名来访问，不需要创建一个类的对象来访问该类的静态成员，所以static修饰的成员又称作类变量和类方法。静态变量与实例变量不同，实例变量总是通过对象来访问，因为它们的值在对象和对象之间有所不同。

静态变量属于类，不属于任何独立的对象，所以无需创建类的实例就可以访问静态变量。之所以会产生这样的结果，是因为编译器只为整个类创建了一个静态变量的副本，也就是只分配一个内存空间，虽然有多个实例，但这些实例共享该内存。任何一个对象对类变量的修改，都会影响其它对象。外部有两种访问方式：通过对象来访问或通过类名来访问。实例变量则不同，每创建一个对象，都会分配一次内存空间，不同变量的内存相互独立，互不影响，改变 a 对象的实例变量不会影响 b 对象。

读者肯定注意到，main() 也是一个静态方法，不对任何对象进行操作。实际上，在程序启动时还没有任何对象，main() 方法是程序的入口，将被执行并创建程序所需的对象。  
  
关于静态变量和静态方法的总结：

* 一个类的静态方法只能访问静态变量；
* 一个类的静态方法不能够直接调用非静态方法；
* 如访问控制权限允许，静态变量和静态方法也可以通过对象来访问，但是不被推荐；
* 静态方法中不存在当前对象，因而不能使用 this，当然也不能使用 super；
* 静态方法不能被非静态方法覆盖；
* 构造方法不允许声明为 static 的；
* 局部变量不能使用static修饰。

定义Java类时如果没有显示的指明父类，那么就默认继承了 Object 类

使用内部类的主要原因有：

* 内部类可以访问外部类中的数据，包括私有的数据。
* 内部类可以对同一个包中的其他类隐藏起来。
* 当想要定义一个回调函数且不想编写大量代码时，使用匿名(anonymous)内部类比较便捷。
* 减少类的命名冲突。

类是对象的模板，抽象类和接口可以看做是具体的类的模板。由于从某种角度讲，接口是一种特殊的抽象类我们首先分析它们具有的相同点。

* 都代表类树形结构的抽象层。在使用引用变量时，尽量使用类结构的抽象层，使方法的定义和实现分离，这样做对于代码有松散耦合的好处。
* 都不能被实例化。
* 都能包含抽象方法。抽象方法用来描述系统提供哪些功能，而不必关心具体的实现。，必须重载抽象方法，实现各自功能

下面说一下抽象类和接口的主要区别。

1. 抽象类可以为部分方法提供实现，避免了在子类中重复实现这些方法，提高了代码的可重用性，这是抽象类的优势；而接口中只能包含抽象方法，不能包含任何实现。
2. 一个类只能继承一个直接的父类（可能是抽象类），但一个类可以实现多个接口，这个就是接口的优势。

综上所述，接口和抽象类各有优缺点，在接口和抽象类的选择上，必须遵守这样一个原则：

* 行为模型应该总是通过接口而不是抽象类定义，所以通常是优先选用接口，尽量少用抽象类。
* 选择抽象类的时候通常是如下情况：需要定义子类的行为，又要为子类提供通用的功能。

异常的分类：exception和error

一个try和它的catch语句形成了一个单元。catch子句的范围限制于try语句前面所定义的语句。一个catch语句不能捕获另一个try声明所引发的异常（除非是嵌套的try语句情况）。  
  
被try保护的语句声明必须在一个大括号之内（也就是说，它们必须在一个块中）。你不能单独使用try。构造catch子句的目的是解决异常情况并且像错误没有发生一样继续运行

当你用多catch语句时，记住异常子类必须在它们任何父类之前使用是很重要的。这是因为运用父类的catch语句将捕获该类型及其所有子类类型的异常。这样，如果子类在父类后面，子类将永远不会到达。而且，Java中不能到达的代码是一个错误

1. 用户程序自定义的异常和应用程序特定的异常,必须借助于 throws 和 throw 语句来定义抛出异常。  
     
   1.1   throw是语句抛出一个异常。  
   语法：throw (异常对象);  
            throw e;  
     
   1.2   throws是方法可能抛出异常的声明。(用在声明方法时，表示该方法可能要抛出异常)  
   语法：[(修饰符)](返回值类型)(方法名)([参数列表])[throws(异常类)]{......}  
               public void doA(int a) throws Exception1,Exception3{......}

throws E1,E2,E3只是告诉程序这个方法可能会抛出这些异常，方法的调用者可能要处理这些异常，而这些异常E1，E2，E3可能是该函数体产生的。  
throw则是明确了这个地方要抛出这个异常。

throw语句用在方法体内，表示抛出异常，由方法体内的语句处理。  
throws语句用在方法声明后面，表示再抛出异常，由该方法的调用者来处理。  
  
throws主要是声明这个方法会抛出这种类型的异常，使它的调用者知道要捕获这个异常。  
throw是具体向外抛异常的动作，所以它是抛出一个异常实例。  
  
throws说明你有那个可能，倾向。  
throw的话，那就是你把那个倾向变成真实的了。  
  
同时：  
1、throws出现在方法函数头；而throw出现在函数体。  
2、throws表示出现异常的一种可能性，并不一定会发生这些异常；throw则是抛出了异常，执行throw则一定抛出了某种异常。  
3、两者都是消极处理异常的方式（这里的消极并不是说这种方式不好），只是抛出或者可能抛出异常，但是不会由函数去处理异常，真正的处理异常由函数的上层调用处理。

断言用于证明和测试程序的假设，比如“这里的值大于 5”。  
断言可以在运行时从代码中完全删除，所以对代码的运行速度没有影响。

断言有两种方法：

* 一种是 assert<<布尔表达式>> ；
* 另一种是 assert<<布尔表达式>> ：<<细节描述>>。

如果布尔表达式的值为false ， 将抛出AssertionError 异常

用于验证方法中的内部逻辑，包括：

* 内在不变式
* 控制流程不变式
* 后置条件和类不变式

字节流：由于InputStream和OutputStream都是抽象类，所以在程序中创建的输入流对象一般是它们某个子类的对象，通过调用对象继承的read()和write()方法就可实现对相应外设的输入输出操作。

字符流是针对字符数据的特点进行过优化的，因而提供一些面向字符的有用特性，字符流的源或目标通常是文本文件。 Reader和Writer是java.io包中所有字符流的父类。由于它们都是抽象类，所以应使用它们的子类来创建实体对象，利用对象来处理相关的读写操作

1. Vector和ArrayList、LinkedList区别   
   (1)、LinkedList内部以链表形式存储数据。ArrayList内部以数组形式存储数据。   
   (2)、Vector同ArrayList，不过它与ArrayList比较起来是线程安全(thread-safe)的。   
   3、Hashtable 和 HashMap之间的区别   
   (1)、Hashtable是继承了Dictionary，是线程安全的。   
   (2)、HashMap实现了Map接口，不是线程安全的。

switch语句的表达式可以是byte吗?可以是long吗?可以是String吗?

可以,不可以,JDK7以后可以

A:case后面只能是常量，不能是变量，而且，多个case后面的值不能出现相同的

Java的内存分配

A:栈 存储局部变量

B:堆 存储所有new出来的

C:方法区(面向对象部分详细讲解)

D:本地方法区(系统相关)

E:寄存器(CPU使用)

注意：

a:局部变量 在方法定义中或者方法声明上定义的变量。

b:栈内存和堆内存的区别

栈：数据使用完毕，就消失。

堆：每一个new出来的东西都有地址

每一个变量都有默认值

byte,short,int,long 0

float,double 0.0

char '\u0000'

boolean false

引用类型 null

数据使用完毕后，在垃圾回收器空闲的时候回收。

成员变量和局部变量的区别(理解)

(1)在类中的位置不同

成员变量：类中方法外

局部变量：方法定义中或者方法声明上

(2)在内存中的位置不同

成员变量：在堆中

局部变量：在栈中

(3)生命周期不同

成员变量：随着对象的创建而存在，随着对象的消失而消失

局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

(4)初始化值不同

成员变量：有默认值

局部变量：没有默认值，必须定义，赋值，然后才能使用

6:this关键字(掌握)

(1)代表当前类的引用对象

记住：哪个对象调用方法，该方法内部的this就代表那个对象

(2)this的应用场景：

A:解决了局部变量隐藏成员变量的问题

B:其实this还有其他的应用，明天讲解。

思考题：构造方法中可不可以有return语句呢?

可以。而是我们写成这个样子就OK了：return;

其实，在任何的void类型的方法的最后你都可以写上：return;

代码：Student s = new Student();做了哪些事情?(理解)

(1)把Student.class文件加载到内存

(2)在栈内存为s开辟空间

(3)在堆内存为学生对象申请空间

(4)给学生的成员变量进行默认初始化。null,0

(5)给学生的成员变量进行显示初始化。林青霞,27

(6)通过构造方法给成员变量进行初始化。刘意,30

(7)对象构造完毕，把地址赋值给s变量

(5)静态变量和成员变量的区别

A:所属不同

静态变量：属于类，类变量

成员变量：属于对象，对象变量，实例变量

B:内存位置不同

静态变量：方法区的静态区

成员变量：堆内存

C:生命周期不同

静态变量：静态变量是随着类的加载而加载，随着类的消失而消失

成员变量：成员变量是随着对象的创建而存在，随着对象的消失而消失

D:调用不同

静态变量：可以通过对象名调用，也可以通过类名调用

成员变量：只能通过对象名调用

B:this和super的区别和各自的作用?

(3)多态中的成员访问特点

A:成员变量

编译看左边，运行看左边

B:构造方法

子类的构造都会默认访问父类构造

C:成员方法

编译看左边，运行看右边

D:静态方法

编译看左边，运行看左边

为什么?

因为成员方法有重写。

3:抽象类(掌握)

(1)把多个共性的东西提取到一个类中，这是继承的做法。

但是呢，这多个共性的东西，在有些时候，方法声明一样，但是方法体。

也就是说，方法声明一样，但是每个具体的对象在具体实现的时候内容不一样。

所以，我们在定义这些共性的方法的时候，就不能给出具体的方法体。

而一个没有具体的方法体的方法是抽象的方法。

在一个类中如果有抽象方法，该类必须定义为抽象类。

(2)抽象类的特点

A:抽象类和抽象方法必须用关键字abstract修饰

B:抽象类中不一定有抽象方法,但是有抽象方法的类一定是抽象类

C:抽象类不能实例化

D:抽象类的子类

a:是一个抽象类。

b:是一个具体类。这个类必须重写抽象类中的所有抽象方法。

(3)抽象类的成员特点：

A:成员变量

有变量，有常量

B:构造方法

有构造方法

C:成员方法

有抽象，有非抽象

(4)抽象类的练习

A:猫狗案例练习

B:老师案例练习

C:学生案例练习

D:员工案例练习

(5)抽象类的几个小问题

A:抽象类有构造方法，不能实例化，那么构造方法有什么用?

用于子类访问父类数据的初始化

B:一个类如果没有抽象方法,却定义为了抽象类，有什么用?

为了不让创建对象

C:abstract不能和哪些关键字共存

a:final 冲突

b:private 冲突

c:static 无意义

4:接口(掌握)

(1)回顾猫狗案例，它们仅仅提供一些基本功能。

比如：猫钻火圈，狗跳高等功能，不是动物本身就具备的，

是在后面的培养中训练出来的，这种额外的功能，java提供了接口表示。

(2)接口的特点：

A:接口用关键字interface修饰

interface 接口名 {}

B:类实现接口用implements修饰

class 类名 implements 接口名 {}

C:接口不能实例化

D:接口的实现类

a:是一个抽象类。

b:是一个具体类，这个类必须重写接口中的所有抽象方法。

(3)接口的成员特点：

A:成员变量

只能是常量

默认修饰符：public static final

B:构造方法

没有构造方法

C:成员方法

只能是抽象的

默认修饰符：public abstract

(4)类与类,类与接口,接口与接口

A:类与类

继承关系，只能单继承，可以多层继承

B:类与接口

实现关系，可以单实现，也可以多实现。

还可以在继承一个类的同时，实现多个接口

C:接口与接口

继承关系，可以单继承，也可以多继承

(5)抽象类和接口的区别(自己补齐)?

A:成员区别

抽象类：

接口：

B:关系区别:

类与类：

类与接口：

接口与接口：

C:设计理念不同

抽象类：is a，抽象类中定义的是共性功能。

接口：like a，接口中定义的是扩展功能。

字符串的面试题(看程序写结果)

A:==和equals()

String s1 = new String("hello");

String s2 = new String("hello");

System.out.println(s1 == s2);// false

System.out.println(s1.equals(s2));// true

String s3 = new String("hello");

String s4 = "hello";

System.out.println(s3 == s4);// false

System.out.println(s3.equals(s4));// true

String s5 = "hello";

String s6 = "hello";

System.out.println(s5 == s6);// true

System.out.println(s5.equals(s6));// true

B:字符串的拼接

String s1 = "hello";

String s2 = "world";

String s3 = "helloworld";

System.out.println(s3 == s1 + s2);// false

System.out.println(s3.equals((s1 + s2)));// true

System.out.println(s3 == "hello" + "world");// false 这个我们错了，应该是true

System.out.println(s3.equals("hello" + "world"));// true

小细节：

StringBuffer：同步的，数据安全，效率低。

StringBuilder：不同步的，数据不安全，效率高。

集合和数组的区别?

A:长度区别

数组固定

集合可变

B:内容区别

数组可以是基本类型，也可以是引用类型

集合只能是引用类型

C:元素内容

数组只能存储同一种类型

集合可以存储不同类型(其实集合一般存储的也是同一种类型)

(7)List的子类特点(面试题)

ArrayList

底层数据结构是数组，查询快，增删慢。

线程不安全，效率高。

Vector

底层数据结构是数组，查询快，增删慢。

线程安全，效率低。

LinkedList

底层数据结构是链表，查询慢，增删快。

线程不安全，效率高。

到底使用谁呢?看需求?

分析：

要安全吗?

要：Vector(即使要，也不使用这个，后面再说)

不要：ArrayList或者LinkedList

查询多；ArrayList

增删多：LinkedList

什么都不知道，就用ArrayList。

在集合中常见的数据结构(掌握)

ArrayXxx:底层数据结构是数组，查询快，增删慢

LinkedXxx:底层数据结构是链表，查询慢，增删快

HashXxx:底层数据结构是哈希表。依赖两个方法：hashCode()和equals()

TreeXxx:底层数据结构是二叉树。两种方式排序：自然排序和比较器排序

|--Set 无序,唯一

|--HashSet

底层数据结构是哈希表。

如何保证元素唯一性的呢?

依赖两个方法：hashCode()和equals()

开发中自动生成这两个方法即可

|--LinkedHashSet

底层数据结构是链表和哈希表

由链表保证元素有序

由哈希表保证元素唯一

|--TreeSet

底层数据结构是红黑树。

如何保证元素排序的呢?

自然排序

比较器排序

如何保证元素唯一性的呢?

根据比较的返回值是否是0来决定

4:针对Collection集合我们到底使用谁呢?(掌握)

唯一吗?

是：Set

排序吗?

是：TreeSet

否：HashSet

如果你知道是Set，但是不知道是哪个Set，就用HashSet。

否：List

要安全吗?

是：Vector

否：ArrayList或者LinkedList

查询多：ArrayList

增删多：LinkedList

如果你知道是List，但是不知道是哪个List，就用ArrayList。

如果你知道是Collection集合，但是不知道使用谁，就用ArrayList。

如果你知道用集合，就用ArrayList。

(2)Map和Collection的区别?

A:Map 存储的是键值对形式的元素，键唯一，值可以重复。夫妻对

B:Collection 口存储的是单独出现的元素，子接Set元素唯一，子接口List元素可重复。光棍

2)面试题：Collection和Collections的区别

A:Collection 是单列集合的顶层接口，有两个子接口List和Set

B:Collections 是针对集合进行操作的工具类，可以对集合进行排序和查找等

(4)面试题

A:编译期异常和运行期异常的区别?

编译期异常 必须要处理的，否则编译不通过

运行期异常 可以不处理，也可以处理

B:throw和throws是的区别

throw:

在方法体中,后面跟的是异常对象名,并且只能是一个

throw抛出的是一个异常对象，说明这里肯定有一个异常产生了

throws:

在方法声明上,后面跟的是异常的类名,可以是多个

throws是声明方法有异常，是一种可能性，这个异常并不一定会产生

(5)finally关键字及其面试题

A:finally用于释放资源，它的代码永远会执行。特殊情况：在执行到finally之前jvm退出了

B:面试题

a:final,finally,finalize的区别?

b:如果在catch里面有return,请问finally还执行吗?如果执行,在return前还是后

会，前。

实际上在中间。这个上课我们讲过

IO流分类

字节流：

InputStream

FileInputStream

BufferedInputStream

OutputStream

FileOutputStream

BufferedOutputStream

字符流：

Reader

FileReader

BufferedReader

Writer

FileWriter

BufferedWriter

(1)多线程：一个应用程序有多条执行路径

进程：正在执行的应用程序

线程：进程的执行单元，执行路径

单线程：一个应用程序只有一条执行路径

多线程：一个应用程序有多条执行路径

多进程的意义?

提高CPU的使用率

多线程的意义?

提高应用程序的使用率

(2)Java程序的运行原理及JVM的启动是多线程的吗?

A:Java命令去启动JVM，JVM会启动一个进程，该进程会启动一个主线程。

B:JVM的启动是多线程的，因为它最低有两个线程启动了，主线程和垃圾回收线程。

(3)多线程的实现方案(自己补齐步骤及代码 掌握)

A:继承Thread类

B:实现Runnable接口

(4)线程的调度和优先级问题

A:线程的调度

a:分时调度

b:抢占式调度 (Java采用的是该调度方式)

B:获取和设置线程优先级

a:默认是5

b:范围是1-10

(5)线程的控制(常见方法)

A:休眠线程

B:加入线程

C:礼让线程

D:后台线程

E:终止线程(掌握)

(6)线程的生命周期(参照 线程生命周期图解.bmp)

A:新建

B:就绪

C:运行

D:阻塞

E:死亡

9)多线程安全问题的原因(也是我们以后判断一个程序是否有线程安全问题的依据)

A:是否有多线程环境

B:是否有共享数据

C:是否有多条语句操作共享数据

(10)同步解决线程安全问题

A:同步代码块

synchronized(对象) {

需要被同步的代码;

}

这里的锁对象可以是任意对象。

B:同步方法

把同步加在方法上。

这里的锁对象是this

C:静态同步方法

把同步加在方法上。

这里的锁对象是当前类的字节码文件对象(反射再讲字节码文件对象)

(11)回顾以前的线程安全的类

A:StringBuffer

B:Vector

C:Hashtable

D:如何把一个线程不安全的集合类变成一个线程安全的集合类

用Collections工具类的方法即可。

# **Java中HashMap和TreeMap的区别深入理解**

首先介绍一下什么是Map。在数组中我们是通过数组下标来对其内容索引的，而在Map中我们通过对象来对对象进行索引，用来索引的对象叫做key，其对应的对象叫做value。这就是我们平时说的键值对。  
  
HashMap通过hashcode对其内容进行快速查找，而 TreeMap中所有的元素都保持着某种固定的顺序，如果你需要得到一个有序的结果你就应该使用TreeMap（HashMap中元素的排列顺序是不固定的）。   
HashMap 非线程安全 TreeMap 非线程安全   
  
**线程安全**   
在Java里，线程安全一般体现在两个方面：   
1、多个thread对同一个java实例的访问（read和modify）不会相互干扰，它主要体现在关键字synchronized。如ArrayList和Vector，HashMap和Hashtable   
（后者每个方法前都有synchronized关键字）。如果你在interator一个List对象时，其它线程remove一个element，问题就出现了。   
  
2、每个线程都有自己的字段，而不会在多个线程之间共享。它主要体现在java.lang.ThreadLocal类，而没有Java关键字支持，如像static、transient那样。   
1.AbstractMap抽象类和SortedMap接口   
AbstractMap抽象类：(HashMap继承AbstractMap)覆盖了equals()和hashCode()方法以确保两个相等映射返回相同的哈希码。如果两个映射大小相等、包含同样的键且每个键在这两个映射中对应的值都相同，则这两个映射相等。映射的哈希码是映射元素哈希码的总和，其中每个元素是Map.Entry接口的一个实现。因此，不论映射内部顺序如何，两个相等映射会报告相同的哈希码。   
SortedMap接口：（TreeMap继承自SortedMap）它用来保持键的有序顺序。SortedMap接口为映像的视图(子集)，包括两个端点提供了访问方法。除了排序是作用于映射的键以外，处理SortedMap和处理SortedSet一样。添加到SortedMap实现类的元素必须实现Comparable接口，否则您必须给它的构造函数提供一个Comparator接口的实现。TreeMap类是它的唯一一份实现。   
  
2.两种常规Map实现   
HashMap：基于哈希表实现。使用HashMap要求添加的键类明确定义了hashCode()和equals()[可以重写hashCode()和equals()]，为了优化HashMap空间的使用，您可以调优初始容量和负载因子。   
(1)HashMap(): 构建一个空的哈希映像   
(2)HashMap(Map m): 构建一个哈希映像，并且添加映像m的所有映射   
(3)HashMap(int initialCapacity): 构建一个拥有特定容量的空的哈希映像   
(4)HashMap(int initialCapacity, float loadFactor): 构建一个拥有特定容量和加载因子的空的哈希映像   
TreeMap：基于红黑树实现。TreeMap没有调优选项，因为该树总处于平衡状态。   
(1)TreeMap():构建一个空的映像树   
(2)TreeMap(Map m): 构建一个映像树，并且添加映像m中所有元素   
(3)TreeMap(Comparator c): 构建一个映像树，并且使用特定的比较器对关键字进行排序   
(4)TreeMap(SortedMap s): 构建一个映像树，添加映像树s中所有映射，并且使用与有序映像s相同的比较器排序   
  
3.两种常规Map性能   
HashMap：适用于在Map中插入、删除和定位元素。   
Treemap：适用于按自然顺序或自定义顺序遍历键(key)。   
  
4.总结   
HashMap通常比TreeMap快一点(树和哈希表的数据结构使然)，建议多使用HashMap，在需要排序的Map时候才用TreeMap。