1.List和Map底层用的是什么，分别适用于什么场景

List和Map的底层分别用的是数组和链表，

数组的应用场景：数据量比较少，经常做的是按序号访问元素，数组更容易实现，任何高级语言都支持，构建的线性表较稳定。

链表的应用场景：对线性表的长度或者规模难以估计，频繁的做插入删除操作，构建动态性比较强的线性表。

2.ArrayList为什么增删效率低，查询速率高

因为在ArrayList的中间插入或删除一个元素意味着这个列表中剩余的元素都会被移动，所以ArrayList的增删速度比较低，

ArrayList的内部实现是基于基础的对象数组的，所以他在使用get方法访问列表中的任意一个元素时，速度要快。

linkedList为什么增删效率高，查询速率低：

linkedList的底层是基于链表结构的，所以在LinkedList的中插入或者删除一个元素的开销是固定的，

LinkedList中的get方法是按照顺序从列表的一端开始检查，直到另外一端。所以查询速率低。

线程安全就是多线程访问时，采用了加锁机制，当一个线程访问该类的某个数据时，进行保护，其他线程不能进行访问直到该线程读取完，其他线程才可使用。不会出现数据不一致或者数据污染。

线程不安全就是不提供数据访问保护，有可能出现多个线程先后更改数据造成所得到的数据是脏数据

3.HashMap的键是如何保证唯一性的

通过hashcode（）和equals（）方法保证键的唯一性；

Equals方法用于比较是否重复，hashcode用于减少比较的次数；

#### 如何理解HashCode的作用

从Object角度看，JVM每new一个Object，它就会将这个Object丢到一个Hash表中去，这样的话，下次做Object的比较或者取这个对象的时候（读取过程），它会根据对象的HashCode再从Hash表中取这个对象。这样做的目的是提高取对象的效率。若HashCode相同再去调用equal

4.堆栈结构的特点

先进后出，后进先出；

5.多线程的创建方式以及不同

1.继承 Thread 类：但 Thread 本质上也是实现了 Runnable 接口的一个实例，它代表一个线程的实例，并且，启动线程的唯一方法就是通过 Thread 类的 start()实例方法。start()方法是一个 native 方法，它将启动一个新线程，并执行 run()方法。这种方式实现多线程很简单，通过自己的类直接 extend Thread，并复写 run()方法，就可以启动新线程并执行自己定义的 run()方法。例如：继承 Thread 类实现多线程，并在合适的地方启动线程

用法简单，但是不能继承多个父类

2．实现Runnable接口的方式实现多线程，并且实例化Thread，传入自己的Thread实例，调用run（）方法；

3.使用ExecutorService .Callable .Future 实现有返回结果的多线程：

ExecutorService Callable Future 这些对象实际上都属于Executor框架中的功能类；

线程类只是实现了Runnable接口或Callable接口，还可以继承其他类。

在这种方式下，多个线程可以共享同一个target对象，所以非常适合多个相同线程来处理同一份资源的情况，从而可以将CPU、代码和数据分开，形成清晰的模型，较好地体现了面向对象的思想。

6.在java中wait和sleep方法有什么不同：

最大的不同是等待时wait方法会释放锁，而sleep方法一直持有锁，wait方法通常被用于线程间的交互，sleep方法通常被用于暂停执行。

7.说说多线程的几种状态(共6种):

New（新建）:线程刚刚被创建，但是并未启动，还没有调用start方法

Runnable（可运行）：线程可以在java虚拟机中运行的状态，可能正在运行自己的代码，也可能没有，这取决与操作系统处理器

Blocked（锁阻塞）：当一个线程试图获取一个对象锁，而该对象锁被其它线程持有，则该线程进入Blocked状态，当该线程持有锁的时候，该线程变成Runnable状态。

Waiting（无线等待）：一个线程在等待另一个线程执行一个（唤醒）动作，该线程进入Waiting状态，进入这个状态后是不能自动唤醒的，必须等待另一个线程调用notify或者notifyAll方法才能够唤醒。

Timed Waiting（计时等待）：同waiting状态，有几个方法有超时参数，调用他们进入Timed Waiting状态，这一状态将一直保持到朝时期满或者接收到唤醒通知。带有超时参数的常用方法有Thread.sleep Object.wait。

Teminated（被终止）：因为run方法正常退出而死亡，或者因为没有捕获的异常终止了run方法而死亡。

8.java中异常的分类：

按照异常异常需要处理的时机分为编译时异常和运行时异常

对于编译时异常的处理方式有两种一种是try（）catch，另一种就是直接抛出。

而对于运行是异常只有当代码在运行的时候才会发生。

说出5个runtimeException：

NullPointerException：空指针异常

IllegalArgumentException：不合法参数异常

NumberFormatException：字符串转换为数字异常

IndexOutOfBoundException：数组索引越界异常

ClassCastException：数据类型转换异常

SQLException：SQL异常

9.Throw和throws的区别：

Throw语句语句用在方法体内，表示抛出异常，由方法体内的语句处理；

Throw是具体向外抛出异常的动作，所以他抛出的是一个异常实例，执行throw一定是抛出某种异常。

Throws语句是用在方法声明后面，表示如果抛出异常由该方法的调用者来进行处理；

Throws主要是声明这个方法会抛出某种类型的异常，让他的使用者要知道需要捕获的异常的类型；

Throws表示出现异常额一种可能性，并不是一定会发生这种异常

10.String. StringBuild. StringBuffer的区别

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；

StringBuffer/StringBuilder表示的字符串对象可以直接进行修改。

StringBuilder是java5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，因为它的所有方法都没有被synchronize修饰，因此它的效率理论上也比StringBuffer要高；

11.Java的基本数据类型都有那些各占了几个字节



12.java中有几种类型的流

按照流的方向可分为：输入流和输出流

按照处理数据的单位可分为：字节流和字符流

按照实现功能可分为：节点流（可以从或者向一个特定的节点读取数据。如FileRearder）和处理流（对一个与存在的流的连接和封装，通过所封装的流的功能实现数据读写。如BufferedReader）

13.什么是java的序列化，如何实现java序列化：

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可以将流化后的对象传输于网络之间。序列化是为了解决在对 对象流进行读写操作是所引发的问题。

序列化的实现：将需要被序列化的类 实现Serializable接口，该接口没有需要实现的方法，implements Serializable 只是为了标注该对象是可以被序列化的，然后使用一个输出流来构造一个ObjectOutPutStream对象，接着使用ObjectOutPutStream对象的writeObject（Object obj）方法就可以将参数为obj的对象进行写出；

14.List 和Map Set的区别：

List和Set是存储单列数据的集合，Map是存储键和值这样双列数据的集合；

List中存储的数据是有顺序的，并且允许重复；

Map中存储的数据是没有顺序的，其键是不能重复的，

Set中存储的数据无序且不能重复，但元素在集合中的位置由元素的hashcode决定，

15.抽象类和接口的相同点和不同点：

一、不同点：

抽象类：

抽象类可以定义构造器

可以有抽象方法和具体方法

抽象类中的成员可以是private，默认，protected Public

抽象类中可以定义成员方法

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

抽象类中可以包含静态方法static

一个类只能继承一个抽象类

接口：

接口中不能定义构造器

方法全部都是抽象方法

接口中定义的成员变量实际上都是常量

接口中的成员全都是public的

接口中不能有静态方法

一个类可以实现多个接口

二、相同：

不能够实例化

可以将抽象类和接口作为引用类型

一个类如果继承了某个抽象类或者实现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。

16.==和equals的区别：

Equals和==最大的区别就是一个是方法一个是运算符。

==：如果比较的对象是基本数据类型，则比较的是数值是否相等；如果比较的是引用类型，则比较的是对象的地址值是否相等。

Equals（）：用来比较方法两个对象的内容是否相等。

注意：equals方法不能用于基本数据类型变量，如果没有对equals方法进行重写，则比较的是引用类型的变量所指向的对象地址。

17.Break和continue的区别：

Break和continue都是用来控制循环的语句

Break用于完全结束一个循环，跳出循环执行体-执行循环后面的语句。

Continue用于跳过本次循环，执行下次循环。

&和&&的区别：

&运算符有两种用法：（1）按位与；（2）逻辑与；

&&运算符是短路与：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整的表达式的值才是true。

&&之所以被成为短路运算符是因为，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。很多时候我们使用的都是&&而不是&。

18.数据库事务的四个特性及含义

数据库事务transanction正确执行的四个基本要素。ACID,原子性(Atomicity)、一致性(Correspondence)、隔离性(Isolation)、持久性(Durability)。

原子性:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

一致性:在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。

隔离性:隔离状态执行事务，使它们好像是系统在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行 相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请 求，使得在同一时间仅有一个请求用于同一数据。

持久性:在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

19.数据库优化的思路

1.SQL语句优化(至少说出3点,可以百度查)

1）应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。

2）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：

select id from t where num is null

可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

select id from t where num=0

3）很多时候用 exists 代替 in 是一个好的选择

4）用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤

20.什么是数据库的三范式?

第一范式，又称1NF，它指的是在一个应用中的数据都可以组织成由行和列的表格形式，且表格的任意一个行列交叉点即单元格，都不可再划分为行和列的形式，实际上任意一张表格都满足1NF；

第二范式，又称2NF，它指的是在满足1NF的基础上，一张数据表中的任何非主键字段都全部依赖于主键字段，没有任何非主键字段只依赖于主键字段的一部分。即，可以由主键字段来唯一的确定一条记录。比如学号+课程号的联合主键，可以唯一的确定某个成绩是哪个学员的哪门课的成绩，缺少学号或者缺少课程号，都不能确定成绩的意义。

第三范式，又称3NF，它是指在满足2NF的基础上，数据表的任何非主键字段之间都不产生函数依赖，即非主键字段之间没有依赖关系，全部只依赖于主键字段。例如将学员姓名和所属班级名称放在同一张表中是不科学的，因为学员依赖于班级，可将学员信息和班级信息单独存放，以满足3NF。

<https://wenku.baidu.com/view/85e160bd294ac850ad02de80d4d8d15abe230013.html>

21.SQL 之聚合函数

聚合函数是对一组值进行计算并返回单一的值的函数，它经常与 select 语句中的 group by 子句一同使 用。

1. avg():返回的是指定组中的平均值，空值被忽略。

b. count():返回的是指定组中的项目个数。

c. max():返回指定数据中的最大值。

d. min():返回指定数据中的最小值。

e. sum():返回指定数据的和，只能用于数字列，空值忽略。

f. group by():对数据进行分组，对执行完 group by 之后的组进行聚合函数的运算，计算每一组的值。 最后用 having 去掉不符合条件的组，having 子句中的每一个元素必须出现在 select 列表中(只针对于 mysql)。

**22.字节通向字符的桥梁是谁？**

回答：InputStreamReader

备注：字符是人认识的，字节是计算机认识的，将计算机认识的变成人认识的，这是一个读的过程，类似于将磁盘里面的内容读取到内存中给我们看。

**23.字符通向字节的桥梁是谁?**

回答:OutputStreamWriter

备注:字符是人认识的，字节是计算机认识的，将人认识的变成计算机认识的，这是一个写的过程，类似于将内存中的内容写入到磁盘中进行保存。

**24.List集合和Set集合的区别?**

相同点:

1. 都可以存储元素
2. 都是Collection接口的子接口

不同点:

1. List集合可以存储重复的元素
2. Set集合不可以存储重复的元素

**25. ArrayList和LinkedList的选用场景?**

ArrayList：底层数组实现，有索引，查找快，增删慢

LinkedList:底层是链表实现，增删快，查找慢

备注:参考资料https://blog.csdn.net/bjzhuhehe/article/details/72230559

**26.线程有几种状态？**

回答:线程共有6种状态。

分别为:

NEW(新建)、

RUNNABLE(运行)、

BLOCKED(阻塞)、

WAITING(等待)、

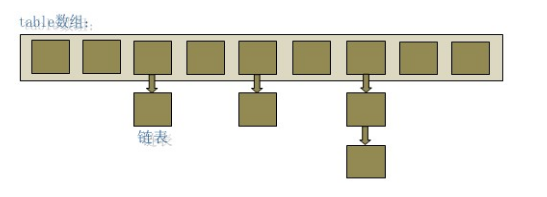
TIMED\_WAITING(带超时的等待)、

TERMINATED(终止)

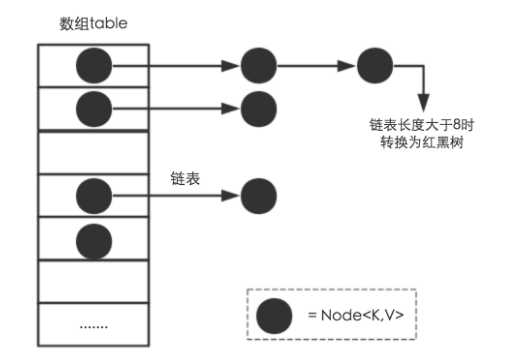
**27.HashMap的数据结构?**

回答:

JDK1.7中的HashMap的数据结构:（数组+链表）(https://www.cnblogs.com/my-king/p/7806571.html)



JDK1.8中的HashMap的数据结构:(HashMap是数组+链表+红黑树)(http://www.importnew.com/20386.html)



**28.Java中的数据类型有几种？**

两种。

第一种:基本数据类型.

四类八种:

数值类型:int、byte、short、long

浮点类型:float、double

字符类型:char

布尔类型:true、false

第二种:引用数据类型.

数组、类、接口

# **29.String是引用数据类型还是基本数据类型？**

回答:String是引用数据类型。

# **30.数组中能存储引用数据类型吗？**

回答:可以。

# **31.什么叫自动装箱？什么叫自动拆箱？自动拆箱和自动装箱是从jdk几版本出现的？**

回答:

自动装箱:基本数据类型->引用数据类型

自动拆箱:引用数据类型->基本数据类型

自动拆箱和自动装箱是从jdk1.5开始的。

# **32.什么叫Overriding?什么叫Overloading?**

回答:

Overriding(重写):

条件:

1.发生在两个类中

2.两个类之间是继承的关系

3.子类的方法的名字、返回值类型、参数和父类的方法的名字、返回值类型、参数一致。

Overloading(重载):

条件:

1. 发生在一个类中
2. 方法的名字一样，但是参数列表不一样，参数列表包括参数的个数、参数的数据类型、参数的顺序

注意:方法的重载和返回值类型无关

# **33.Jdk1.5的特性有哪些？**

## 自动装箱和拆箱

自动装箱的过程：每当需要一种类型的对象时，这种基本类型就自动地封装到与它相同类型的包装中。

自动拆箱的过程：每当需要一个值时，被装箱对象中的值就被自动地提取出来，没必要再去调用intValue()和doubleValue()方法。

自动装箱，只需将该值赋给一个类型包装器引用，java会自动创建一个对象。

自动拆箱，只需将该对象值赋给一个基本类型即可。

java——类的包装器

类型包装器有：Double,Float,Long,Integer,Short,Character和Boolean

## 枚举

把集合里的对象元素一个一个提取出来。枚举类型使代码更具可读性，理解清晰，易于维护。枚举类型是强类型的，从而保证了系统安全性。而以类的静态字段实现的类似替代模型，不具有枚举的简单性和类型安全性。

简单的用法：JavaEnum简单的用法一般用于代表一组常用常量，可用来代表一类相同类型的常量值。

复杂用法：Java为枚举类型提供了一些内置的方法，同事枚举常量还可以有自己的方法。可以很方便的遍历枚举对象。

## 静态导入

通过使用 import static，就可以不用指定 Constants 类名而直接使用静态成员，包括静态方法。

import xxxx 和 import static xxxx的区别是前者一般导入的是类文件如import java.util.Scanner;后者一般是导入静态的方法，import static java.lang.System.out。

## 可变参数（Varargs）

可变参数的简单语法格式为：

methodName([argumentList], dataType...argumentName);

## 内省（Introspector）

是 Java语言对Bean类属性、事件的一种缺省处理方法。例如类A中有属性name,那我们可以通过getName,setName来得到其值或者设置新 的值。通过getName/setName来访问name属性，这就是默认的规则。Java中提供了一套API用来访问某个属性的getter /setter方法，通过这些API可以使你不需要了解这个规则（但你最好还是要搞清楚），这些API存放于包java.beans中。

一 般的做法是通过类Introspector来获取某个对象的BeanInfo信息，然后通过BeanInfo来获取属性的描述器 （PropertyDescriptor），通过这个属性描述器就可以获取某个属性对应的getter/setter方法，然后我们就可以通过反射机制来 调用这些方法。

## 泛型(Generic)

C++ 通过模板技术可以指定集合的元素类型，而Java在1.5之前一直没有相对应的功能。一个集合可以放任何类型的对象，相应地从集合里面拿对象的时候我们也 不得不对他们进行强制得类型转换。引入了泛型，它允许指定集合里元素的类型，这样你可以得到强类型在编译时刻进行类型检查的好处。

## For-Each循环

For-Each循环得加入简化了集合的遍历。假设我们要遍历一个集合对其中的元素进行一些处理。

备注:参考地址https://www.cnblogs.com/langtianya/p/3757993.html(本文章仅供参考)

# **34.Jdk1.7的特性有哪些？**

## switch中可以使用字符串了

|  |
| --- |
| String s = "test"; switch (s) {  case "test" :    System.out.println("test");  case "test1" :    System.out.println("test1");  break ;  default :   System.out.println("break");  break ;  } |

## "<>"这个玩意儿的运用List<String> tempList = new ArrayList<>(); 即泛型实例化类型自动推断。

|  |
| --- |
| public class JDK7GenericTest {  public static void main(String[] args) {  // JDK 7以前需要在后面的<>中写上数据类型  List<String> lst1 = new ArrayList<String>();  // JDK 7 supports limited type inference for generic instance creation  // jdk 7 以后在后面的<>中就可以不用写上数据类型了  List<String> lst2 = new ArrayList<>();    lst1.add("Mon");  lst1.add("Tue");  lst2.add("Wed");  lst2.add("Thu");    for (String item: lst1) {  System.out.println(item);  }    for (String item: lst2) {  System.out.println(item);  }  }  } |

## 3. 自定义自动关闭类

以下是jdk7 api中的接口，（不过注释太长，删掉了close()方法的一部分注释）

|  |
| --- |
| /\*\*  \* A resource that must be closed when it is no longer needed.  \*  \* @author Josh Bloch  \* @since 1.7  \*/public interface AutoCloseable {  /\*\*  \* Closes this resource, relinquishing any underlying resources.  \* This method is invoked automatically on objects managed by the  \* {@code try}-with-resources statement.  \*  \*/  void close() throws Exception;  } |

只要实现该接口，在该类对象销毁时自动调用close方法，你可以在close方法关闭你想关闭的资源，例子如下

|  |
| --- |
| class TryClose implements AutoCloseable {  @Override  public void close() throw Exception {  System.out.println(" Custom close method …  close resources "); }  } //请看jdk自带类BufferedReader如何实现close方法（当然还有很多类似类型的类）    public void close() throws IOException {         synchronized (lock) {             if (in == null)                 return;             in.close();             in = null;             cb = null;         }     } |

## 数值可加下划线

|  |
| --- |
| 例如：int one\_million = 1\_000\_000; |

## 支持二进制文字

|  |
| --- |
| 例如：int binary = 0b1001\_1001; |

## jdk7之前，你必须用try{}finally{}在try内使用资源，在finally中关闭资源，不管try中的代码是否正常退出或者异常退出。jdk7之后，你可以不必要写finally语句来关闭资源，只要你在try()的括号内部定义要使用的资源。请看例子：

jdk7之前

|  |
| --- |
| import java.io.\*;// Copy from one file to another file character by character.// Pre-JDK 7 requires you to close the resources using a finally block.public class FileCopyPreJDK7 {  public static void main(String[] args) {  BufferedReader in = null;  BufferedWriter out = null;  try {  in = new BufferedReader(new FileReader("in.txt"));  out = new BufferedWriter(new FileWriter("out.txt"));  int charRead;  while ((charRead = in.read()) != -1) {  System.out.printf("%c ", (char)charRead);  out.write(charRead);  }  } catch (IOException ex) {  ex.printStackTrace();  } finally { // always close the streams  try {  if (in != null) in.close();  if (out != null) out.close();  } catch (IOException ex) {  ex.printStackTrace();  }  }    try {  in.read(); // Trigger IOException: Stream closed  } catch (IOException ex) {  ex.printStackTrace();  }  }  } |

jdk7之后

|  |
| --- |
| import java.io.\*;// Copy from one file to another file character by character.// JDK 7 has a try-with-resources statement, which ensures that// each resource opened in try() is closed at the end of the statement.public class FileCopyJDK7 {  public static void main(String[] args) {  try (BufferedReader in = new BufferedReader(new FileReader("in.txt"));  BufferedWriter out = new BufferedWriter(new FileWriter("out.txt"))) {  int charRead;  while ((charRead = in.read()) != -1) {  System.out.printf("%c ", (char)charRead);  out.write(charRead);  }  } catch (IOException ex) {  ex.printStackTrace();  }  }  } |

备注:参考资料https://www.cnblogs.com/langtianya/p/3757993.html

# **35.Jdk1.8的特性有哪些？**

## ****接口的默认方法****

Java 8允许我们给接口添加一个非抽象的方法实现，只需要使用 default关键字即可，这个特征又叫做扩展方法，示例如下：

|  |
| --- |
| 代码如下:  interface Formula {     double calculate(int a);      default double sqrt(int a) {         return Math.sqrt(a);     } } |

## ****Lambda 表达式****

首先看看在老版本的Java中是如何排列字符串的：

|  |
| --- |
| List<String> names = Arrays.asList("peter", "anna", "mike", "xenia");  Collections.sort(names, new Comparator<String>() {     @Override     public int compare(String a, String b) {         return b.compareTo(a);     } }); |

在Java 8 中你就没必要使用这种传统的匿名对象的方式了，Java 8提供了更简洁的语法，lambda表达式：

|  |
| --- |
| Collections.sort(names, (String a, String b) -> {     return b.compareTo(a); }); |

## ****函数式接口****

备注:参考资料https://www.cnblogs.com/langtianya/p/3757993.html

# **36.Jdk1.9的特性有哪些？**

## 回答:JShell : 交互式 Java REPL

参考资料:https://blog.csdn.net/dghafq/article/details/78131252

# **37.栈结构的特点？**

回答:后进先出，冬天穿十件衣服，脱衣服的顺序。

# **38.队列结构的特点？**

回答:先进先出，公交车排队。

# **39.数组结构的特点是什么？**

回答:有索引，所以查找元素快，增删慢。

# **40.链表结构的特点是什么？**

回答:增删快，查找慢。

# **41.多态的前提有哪些？**

* 1. 继承关系
     1. 子类继承了父类
     2. 子类继承了抽象类
  2. 实现关系
     1. 接口和接口的实现类是实现的关系

# **42.抽象类和接口的区别？**

* 1. 相同点
     1. 都有抽象方法
     2. 在jdk1.8以上，接口中可以定义非抽象方法，方法被default修饰

注意:

1. 当一个接口添加新方法时，需要所有的实现类都重写新方法，影响到了已有的实现类，可能导致应用崩溃

2. 默认方法可以不强制重写，也不会影响到已有的实现类

* 1. 不同点
     1. 抽象类的修饰符是abstract,接口的修饰符是interface
     2. 如果一个类继承了抽象类，那么这个类可以重写抽象类中的方法，也可以不重写抽象类中的方法，但是如果一个类实现了接口，那么它必须实现接口中的全部方法

# **43.什么是进程?什么是线程?**

* 1. 进程:指一个内存中运行的应用程序,每个进程都有一个独立的内存空间
  2. 线程:进程内部一个独立的执行单元

# **44.线程的创建方式有几种?**

回答:线程的创建方式有很多种，常见的有如下三种:

第一种:继承Thread类创建线程

第二种:实现Runnable接口创建线程

第三种:使用Callable和Future创建线程

# **45.Lambda表达式是jdk几开始出现的？**

回答:jdk1.8

# **46.集合和数组的区别？**

* 1. 相同点:
     1. 都是容器，都是用来存储元素的
  2. 不同点:
     1. 获取集合元素的个数的方式是通过size()方法
     2. 获取数组元素的个数的方式是通过length属性
     3. 数组中可以存储任何数据类型，但是同一个数组只能存储同一种数据类型，而集合中只能存储引用数据类型

转换:

数组转换集合 Arrays.asList(数组) 没有返回值 直接转换为集合

集合转换为数组 集合.toArray() 直接就是一个 数组了

# **47.集合获取集合元素个数和数组获取数组中的元素的个数分别怎么获取？**

* 1. 数组中元素个数的获取方式:通过length属性
  2. 集合中元素个数的获取方式:通过size()方法

# **48.成员变量和局部变量的区别？**

1. 在类中的位置不同

成员变量:在类中方法外面

局部变量:在方法或者代代码块中,或者方法的声明上.在参数列表中

1. 在内存中的位置不同

成员变量:在堆中

局部变量:在栈中

1. 生命周期不同

成员变量:随着对象的创建为存在,随着对象的消失二消失

局部变量:随着方法的调用或者代码块的执行而存在,随着方法的调用完毕或者代码块的执行完毕二消失

1. 初始值

成员变量 有默认的初始值

局部变量 没有默认的初始值,使用之前需要赋值,否则会报错

# **49.如果一个类中的方法是被static关键字修饰的，那么这个方法在其他类中怎么调用?**

回答:通过类名.方法名进行调用

# **50.如果一个类中的变量是被static关键字修饰的，那么这个变量在其他类中怎么调用?**

回答:通过类名.变量名进行调用