抽象方法需要子类重写，而静态的方法是无法被重写的

1. ****阐述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。
2. 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，在调用静态方法时可能对象并没有被初始化
3. 内部类必须在外部类实例化后才能实例化，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
4. 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）。
5. 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。相关代码见mavenTest工程com.yx.mavenTest包下A,B,C类；

32、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：

|  |  |
| --- | --- |
| 1  2 | String s1 = "你好";  String s2 = new String(s1.getBytes("GB2312"), "ISO-8859-1"); |
|  |  |

1. 一般来说，断言用于保证程序最基本、关键的正确性。断言检查通常在开发和测试时开启。为了保证程序的执行效率，在软件发布后断言检查通常是关闭的。断言不应该以任何方式改变程序的状态。简单的说，如果希望在不满足某些条件时阻止代码的执行，就可以考虑用断言来阻止它。
2. 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。
3. 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（什么时候就不再继续递归）。
4. try{}里有一个return语句，那么紧跟在这个try后的finally{}里的代码会不会被执行，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。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，就会返回修改后的值。显然，在finally中返回或者修改返回值会对程序造成很大的困扰*，*
5. finally为确保一段代码不管发生什么异常状况都要被执行；try语句可以嵌套，每当遇到一个try语句，异常的结构就会被放入异常栈中，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，异常栈就会执行出栈操作，直到遇到有处理这种异常的try语句或者最终将异常抛给JVM。
6. 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 （下标越界异常）  
   - NullPointerException （空指针异常）  
   - SecurityException （安全异常）
7. ArrayList 和Vector都是使用数组方式存储数据，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector中的方法由于添加了synchronized修饰，因此Vector是线程安全的容器，但性能上较ArrayList差，因此已经是Java中的遗留容器；LinkedList使用双向链表实现存储，形成一个可以按序号索引的线性结构，与数组的连续存储方式相比，内存的利用率更高，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快，由于ArrayList和LinkedListed都是非线程安全的，如果遇到多个线程操作同一个容器的场景，则可以通过工具类Collections中的synchronizedList方法将其转换成线程安全的容器后再使用。
8. Collection和Collections的区别？  
   答：Collection是一个接口，它是Set、List等容器的父接口；Collections是个一个工具类，提供了一系列的静态方法来辅助容器操作，这些方法包括对容器的搜索、排序、线程安全化等等。
9. TreeMap和TreeSet在排序时如何比较元素？Collections工具类中的sort()方法如何比较元素？  
   答：TreeSet要求存放的对象所属的类必须实现Comparable接口，该接口提供了比较元素的compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap要求存放的键值对映射的键必须实现Comparable接口从而根据键对元素进行排序。Collections工具类的sort方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现Comparable接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是Comparator接口的子类型（需要重写compare方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用。前者在定义对象属性时就重写compareTo方法，short方法则需要传入要比较的元素容器和要比较的对象Comparator<>,之后重写compare方法。
10. sleep()方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复，线程回到就绪状态；wait()是Object类的方法，调用对象的wait()方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool），只有调用对象的notify()方法（或notifyAll()方法）时才能唤醒等待池中的线程进入等锁池（lock pool），如果线程重新获得对象的锁就可以进入就绪状态。线程是进程的一个实体，是CPU调度和分派的基本单位，
11. 线程的sleep()方法和yield()方法有什么区别？  
    答：① sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；  
    ② 线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；  
    ③ sleep()方法声明抛出InterruptedException，而yield()方法没有声明任何异常；  
    ④ sleep()方法比yield()方法（跟操作系统CPU调度相关）具有更好的可移植性。
12. 当一个线程进入一个对象的synchronized方法A之后，其它线程是否可进入此对象的synchronized方法B？synchronized关键字可以将对象或者方法标记为同步，  
    答：不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。因为非静态方法上的synchronized修饰符要求执行方法时要获得对象的锁，如果已经进入A方法说明对象锁已经被取走，那么试图进入B方法的线程就只能在等锁池（注意不是等待池哦）中等待对象的锁。
13. 请说出与线程同步以及线程调度相关的方法。  
    答：- wait()：使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；  
    - sleep()：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要处理InterruptedException异常；  
    - notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；  
    - notityAll()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；
14. 编写多线程程序有几种实现方式？一种是继承Thread类；另一种是实现Runnable接口。两种方式都要通过重写run()方法来定义线程的行为，推荐使用后者，因为Java中的继承是单继承，实现Callable接口，该接口中的call方法可以在线程执行结束时产生一个返回值。
15. 所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作.
16. 启动一个线程是调用run()还是start()方法？  
    答：启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM 调度并执行，这并不意味着线程就会立即运行。run()方法是线程启动后要进行回调（callback）的方法.
17. newSingleThreadExecutor：创建一个单线程的线程池,如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。newCachedThreadPool：创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。 newScheduledThreadPool：创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。newSingleThreadExecutor：创建一个单线程的线程池。此线程池支持定时以及周期性执行任务的需求。
18. 简述synchronized 和java.util.concurrent.locks.Lock的异同？Lock 能完成synchronized所实现的所有功能；主要不同点：Lock有比synchronized更精确的线程语义和更好的性能，而且不强制性的要求一定要获得锁。synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且最好在finally 块中释放（这是释放外部资源的最好的地方）。
19. Blocked表示阻塞状态，阻塞状态又有多种情况，可能是因为调用wait()方法进入等待池，也可能是执行同步方法或同步代码块进入等锁池，或者是调用了sleep()方法或join()方法等待休眠或其他线程结束，或是因为发生了I/O中断。
20. Java中如何实现序列化，有什么意义？  
    答：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，也可将流化后的对象传输于网络之间。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会存在数据乱序的问题）。要实现序列化，需要让一个类实现Serializable接口，该接口是一个标识性接口，标注该类对象是可被序列化的，然后使用一个输出流来构造一个对象输出流并通过writeObject(Object)方法就可以将实现对象写出（即保存其状态）；如果需要反序列化则可以用一个输入流建立对象输入流，然后通过readObject方法从流中读取对象。序列化除了能够实现对象的持久化之外，还能够用于对象的深度克隆.

53、编程实现文件拷贝**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**import** java.nio.ByteBuffer;

**import** java.nio.channels.FileChannel;

**public** **final** **class** MyUtil {

**private** MyUtil() {

**throw** **new** AssertionError();

    }

**public** **static** **void** fileCopy(String source, String target) **throws** IOException {

**try** (InputStream in = **new** FileInputStream(source)) {

**try** (OutputStream out = **new** FileOutputStream(target)) {

**byte**[] buffer = **new** **byte**[4096];

**int** bytesToRead;

**while**((bytesToRead = in.read(buffer)) != -1) {

                    out.write(buffer, 0, bytesToRead);

                }

            }

        }

    }

**public** **static** **void** fileCopyNIO(String source, String target) **throws** IOException {

**try** (FileInputStream in = **new** FileInputStream(source)) {

**try** (FileOutputStream out = **new** FileOutputStream(target)) {

                FileChannel inChannel = in.getChannel();

                FileChannel outChannel = out.getChannel();

                ByteBuffer buffer = ByteBuffer.allocate(4096);

**while**(inChannel.read(buffer) != -1) {

                    buffer.flip();

                    outChannel.write(buffer);

                    buffer.clear();

                }

            }

        }

    }

}

54、****写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。**import** java.io.BufferedReader;

**import** java.io.FileReader;

**public** **final** **class** MyUtil {

    // 工具类中的方法都是静态方式访问的因此将构造器私有不允许创建对象(绝对好习惯)

**private** MyUtil() {

**throw** **new** AssertionError();

    }

  /\*\*

     \* 统计给定文件中给定字符串的出现次数

     \*

     \* @param filename  文件名

     \* @param word 字符串

     \* @return 字符串在文件中出现的次数

     \*/

**public** **static** **int** countWordInFile(String filename, String word) {

**int** counter = 0;

**try** (FileReader fr = **new** FileReader(filename)) {

**try** (BufferedReader br = **new** BufferedReader(fr)) {

                String line = **null**;

**while** ((line = br.readLine()) != **null**) {

**int** index = -1;

**while** (line.length() >= word.length() && (index = line.indexOf(word)) >= 0) {

                        counter++;

                        line = line.substring(index + word.length());//截取剩下的内容

                    }

                }

            }

        } **catch** (Exception ex) {

            ex.printStackTrace();

        }

**return** counter;

    }

 }

1. 使用JDBC操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能？  
   答：要提升读取数据的性能，可以指定通过结果集（ResultSet）对象的setFetchSize()方法指定每次抓取的记录数（典型的空间换时间策略）；要提升更新数据的性能可以使用PreparedStatement语句构建批处理，将若干SQL语句置于一个批处理中执行。
2. 什么是DAO模式？DAO（Data Access Object）顾名思义是一个为数据库或其他持久化机制提供了抽象接口的对象，在不暴露底层持久化方案实现细节的前提下提供了各种数据访问操作。在实际的开发中，应该将所有对数据源的访问操作进行抽象化后封装在一个公共API中。用程序设计语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有事务方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口，在逻辑上该类对应一个特定的数据存储。
3. JDBC能否处理Blob和Clob？  
   答： Blob是指二进制大对象（Binary Large Object），而Clob是指大字符对象（Character Large Objec），因此其中Blob是为存储大的二进制数据而设计的，而Clob是为存储大的文本数据而设计的。JDBC的PreparedStatement和ResultSet都提供了相应的方法来支持Blob和Clob操作。
4. 获得一个类的类对象有哪些方式？方法1：类型.class，例如：String.class  
   - 方法2：对象.getClass()，例如：”hello”.getClass()  
   - 方法3：Class.forName()，例如：Class.forName(“java.lang.String”)
5. 如何通过反射创建对象？  
   答：- 方法1：通过类对象调用newInstance()方法，例如：String.class.newInstance()  
   - 方法2：通过类对象的getConstructor()或getDeclaredConstructor()方法获得构造器（Constructor）对象并调用其newInstance()方法创建对象，例如：String.class.getConstructor(String.class).newInstance(“Hello”);
6. JDBC中如何进行事务处理？  
   答：Connection提供了事务处理的方法，通过调用setAutoCommit(false)可以设置手动提交事务；当事务完成后用commit()显式提交事务；如果在事务处理过程中发生异常则通过rollback()进行事务回滚。除此之外，从JDBC 3.0中还引入了Savepoint（保存点）的概念，允许通过代码设置保存点并让事务回滚到指定的保存点。
7. 如何通过反射获取和设置对象私有字段的值？答：可以通过类对象的getDeclaredField()方法字段（Field）对象，然后再通过字段对象的setAccessible(true)将其设置为可以访问，接下来就可以通过get/set方法来获取/设置字段的值了。
8. 简述一下你了解的设计模式：工厂模式，代理模式，适配器模式：模板方法模式。
9. 用Java写一个冒泡排序。**import** java.util.Comparator;

**public** **interface** Sorter {

**public** <T **extends** Comparable<T>> **void** sort(T[] list);

**public** <T> **void** sort(T[] list, Comparator<T> comp);

}

**import** java.util.Comparator;

**public** **class** BubbleSorter **implements** Sorter {

  @Override

**public** <T **extends** Comparable<T>> **void** sort(T[] list) {

**boolean** swapped = **true**;

**for** (**int** i = 1, len = list.length; i < len && swapped; ++i) {

            swapped = **false**;

**for** (**int** j = 0; j < len - i; ++j) {

**if** (list[j].compareTo(list[j + 1]) > 0) {

                    T temp = list[j];

                    list[j] = list[j + 1];

                    list[j + 1] = temp;

                    swapped = **true**;

                }

            }

        }

    }

    @Override

**public** <T> **void** sort(T[] list, Comparator<T> comp) {

**boolean** swapped = **true**;

**for** (**int** i = 1, len = list.length; i < len && swapped; ++i) {

            swapped = **false**;

**for** (**int** j = 0; j < len - i; ++j) {

**if** (comp.compare(list[j], list[j + 1]) > 0) {

                    T temp = list[j];

                    list[j] = list[j + 1];

                    list[j + 1] = temp;

                    swapped = **true**;

                }

            }

        }

    }

}

64、用Java写一个折半查找。**import** java.util.Comparator;

**public** **class** MyUtil {

**public** **static** <T **extends** Comparable<T>> **int** binarySearch(T[] x, T key) {

**return** binarySearch(x, 0, x.length- 1, key);

   }

  // 使用循环实现的二分查找

**public** **static** <T> **int** binarySearch(T[] x, T key, Comparator<T> comp) {

**int** low = 0;

**int** high = x.length - 1;

**while** (low <= high) {

**int** mid = (low + high) >>> 1;

**int** cmp = comp.compare(x[mid], key);

**if** (cmp < 0) {

            low= mid + 1;

          }

**else** **if** (cmp > 0) {

            high= mid - 1;

          }

**else** {

**return** mid;

          }

      }

**return** -1;

   }

  // 使用递归实现的二分查找

**private** **static**<T **extends** Comparable<T>> **int** binarySearch(T[] x, **int** low, **int** high, T key) {

**if**(low <= high) {

**int** mid = low + ((high -low) >> 1);

**if**(key.compareTo(x[mid])== 0) {

**return** mid;

        }

**else** **if**(key.compareTo(x[mid])< 0) {

**return** binarySearch(x,low, mid - 1, key);

        }

**else** {

**return** binarySearch(x,mid + 1, high, key);

        }

      }

**return** -1;

   }

}计算中间位置时不应该使用(high+ low) / 2的方式，因为加法运算可能导致整数越界，这里应该使用以下三种方式之一：low + (high – low) / 2或low + (high – low) >> 1或(low + high) >>> 1（>>>是逻辑右移，是不带符号位的右移）

1. 什么是ORM？答：对象关系映射（Object-Relational Mapping，简称ORM）是一种为了解决程序的面向对象模型与数据库的关系模型互不匹配问题的技术；简单的说，ORM是通过使用描述对象和数据库之间映射的元数据（在Java中可以用XML或者是注解），将程序中的对象自动持久化到关系数据库中或者将关系数据库表中的行转换成Java对象，其本质上就是将数据从一种形式转换到另外一种形式。
2. UML是统一建模语言（Unified Modeling Language）的缩写,是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持。使用UML可以帮助沟通与交流，辅助应用设计和文档的生成，还能够阐释系统的结构和行为。用例图（用来捕获需求，描述系统的功能，通过该图可以迅速的了解系统的功能模块及其关系）、类图（描述类以及类与类之间的关系，通过该图可以快速了解系统）、时序图（描述执行特定任务时对象之间的交互关系以及执行顺序，通过该图可以了解对象能接收的消息也就是说对象能够向外界提供的服务）。
3. 持久层设计要考虑的问题有哪些？你用过的持久层框架有哪些？
4. 持久,简单的说，就是将内存中的数据保存到关系型数据库、文件系统、消息队列等提供持久化支持的设备中。持久层就是系统中专注于实现数据持久化的相对独立的层面。

持久层设计的目标包括： 数据存储逻辑的分离，提供抽象化的数据访问接口。数据访问底层实现的分离，可以在不修改代码的情况下切换底层实现。资源管理和调度的分离，在数据访问层实现统一的资源调度（如缓存机制）。数据抽象，提供更面向对象的数据操作。持久层框架有：[Hibernate](http://hibernate.org/" \t "http://www.importnew.com/_blank),[MyBatis](http://blog.mybatis.org/" \t "http://www.importnew.com/_blank), [TopLink](http://www.oracle.com/technetwork/cn/middleware/toplink/overview/index.html" \t "http://www.importnew.com/_blank),[Guzz](https://code.google.com/p/guzz/" \t "http://www.importnew.com/_blank),[jOOQ](http://www.jooq.org/" \t "http://www.importnew.com/_blank),[Spring Data](http://projects.spring.io/spring-data/" \t "http://www.importnew.com/_blank),[ActiveJDBC](https://code.google.com/p/activejdbc/" \t "http://www.importnew.com/_blank)

1. 举一个多对多关联的例子，并说明如何实现多对多关联映射。  
   答：例如：商品和订单、学生和课程都是典型的多对多关系。可以在实体类上通过@ManyToMany注解配置多对多关联或者通过映射文件中的和标签配置多对多关联，但是实际项目开发中，很多时候都是将多对多关联映射转换成两个多对一关联映射来实现的。
2. 谈一下你对继承映射的理解。  
   答：继承关系的映射策略有三种：① 每个继承结构一张表（table per class hierarchy），不管多少个子类都用一张表。② 每个子类一张表（table per subclass），公共信息放一张表，特有信息放单独的表。③ 每个具体类一张表（table per concrete class），有多少个子类就有多少张表。第一种方式属于单表策略，其优点在于查询子类对象的时候无需表连接，查询速度快，适合多态查询；缺点是可能导致表很大。后两种方式属于多表策略，其优点在于数据存储紧凑，其缺点是需要进行连接查询，不适合多态查询。
3. @OneToMany注解的mappedBy属性有什么作用？@OneToMany用来配置一对多关联映射，但通常情况下，一对多关联映射都由多的一方来维护关联关系，例如学生和班级，应该在学生类中添加班级属性来维持学生和班级的关联关系（在数据库中是由学生表中的外键班级编号来维护学生表和班级表的多对一关系），如果要使用双向关联，在班级类中添加一个容器属性来存放学生，并使用@OneToMany注解进行映射，此时mappedBy属性就非常重要。
4. MyBatis中使用#和$书写占位符有什么区别？  
   答：#将传入的数据都当成一个字符串，会对传入的数据自动加上引号；$将传入的数据直接显示生成在SQL中。注意：使用$占位符可能会导致SQL注入，能用#的地方就不要使用$，写order by子句的时候应该用$而不是#。$符是直接拼成sql的 ，#符则会以字符串的形式 与sql进行拼接。
5. 解释一下MyBatis中命名空间（namespace）的作用。答：在项目中，可能存在大量的SQL语句，这时候为每个SQL语句起一个唯一的标识（ID）就变得并不容易了。为了解决这个问题，在MyBatis中，可以为每个映射文件起一个唯一的命名空间，这样定义在这个映射文件中的每个SQL语句就成了定义在这个命名空间中的一个ID。只要我们能够保证每个命名空间中这个ID是唯一的，即使在不同映射文件中的语句ID相同，也不会再产生冲突了。

74、ThreadLocal为解决多线程程序的并发问题提供了一种新的思路。ThreadLocal，顾名思义是线程的一个本地化对象，当工作于多线程中的对象使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程分配一个独立的变量副本，所以每一个线程都可以独立的改变自己的副本，而不影响其他线程所对应的副本。从线程的角度看，这个变量就像是线程的本地变量。ThreadLocal类非常简单好用，只有四个方法，能用上的也就是下面三个方法：- void set(T value)：设置当前线程的线程局部变量的值。- T get()：获得当前线程所对应的线程局部变量的值。- void remove()：删除当前线程中线程局部变量的值。