**Queue**:常用的LinkedList, PriorityQueue,

offer，add区别：

一些队列有大小限制，因此如果想在一个满的队列中加入一个新项，多出的项就会被拒绝。

这时新的 offer 方法就可以起作用了。它不是对调用 add() 方法抛出一个 unchecked 异常，而只是得到由 offer() 返回的 false。

poll，remove区别：

remove() 和 poll() 方法都是从队列中删除第一个元素。remove() 的行为与 Collection 接口的版本相似，

但是新的 poll() 方法在用空集合调用时不是抛出异常，只是返回 null。因此新的方法更适合容易出现异常条件的情况。

peek，element区别：

element() 和 peek() 用于在队列的头部查询元素。与 remove() 方法类似，在队列为空时， element() 抛出一个异常，而 peek() 返回 null

|  |
| --- |
| **Java中的linkedlist是双向链表** |

Heap: PriorityQueue

一个例子：

|  |
| --- |
| //define my comparator  **Comparator<Person> myComparator = new** **Comparator<PriorityQueueTest.Person>()** {    public int compare(Person o1, Person o2)  {  return o1.getAge() - o2.getAge();  }  };  PriorityQueue<Person> heap = new PriorityQueue<Person>(10,**myComparator**); |

**Stack**:

|  |
| --- |
| /\*\*java中stack的使用：继承与Vector, 因此底层也是由array实现的  \* Stack只有一个无参的构造函数  \* 几个重要的方法：peek, pop, push, empty()(注意不是isEmpty())而且peek和pop在对空stack时是会抛出异常的，因此在操作次两种方法时，要empty()检查一下是否时空stack.  \* search(Object o): 返回出现object位置到最上面元素的距离 |

**java中复制数组和排序**:

|  |
| --- |
| int[] myNum = new int[nums.length];  System.arraycopy(nums,0,myNum,0,nums.length);  Arrays.sort(myNum); |

**Java中的几个最大值和最小值**：

|  |
| --- |
| Integer.MAX\_VALUE, Integer.MIN\_VALUE  Double.NEGATIVE\_INFINITY, Double.POSITIVE\_INFINITY |

**Java中substring:**

|  |
| --- |
| Start: inclusive  End: exclusive |

**Java中定义数组**：

|  |
| --- |
| Int[] res = new int[10];  Int[] res = {1,2,3,4}; |

**Java 中的array 的copy的问题**：

|  |
| --- |
| Arrays.copyOf (int[] original, int newLength)  Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length  Arrays.copyOfRange (int[] original, int from,int to) |

**Java 中的collections包**：

|  |
| --- |
| Max(Collection<> coll,[Comparator<> comp]);  Reverse(List<> list)  Shuffle(List<> list)  List(Enumeration<> e)  Copy(List<> dest, List<> src) |

**Java 中的bit manipulation**：

|  |
| --- |
| **& (bitwise and) | (bitwise or)**  **^ (bitwise XOR) ~ (bitwise compliment)**  **<< (left shift) >> (right shift) ==🡺 signed shift(保存和传递符号位)**  **>>> (zero fill right shift) ==🡺unsigned shift(补0)**  **<<=**  Left shift AND assignment operator.**Example**C <<= 2 is same as C = C << 2  **^=**  bitwise exclusive OR and assignment operator.**Example:** C ^= 2 is same as C = C ^ 2 |

**Java 中的StringBuilder 和String**：StringBuffer一样，只是线程安全

|  |
| --- |
| Append 可以加string,int,float,StringBuffer等等数据结构  Reverse()  toString()  subString(int start,[int end])  StringBuilder 的构造函数的输入参数可以是string或者是空。注意string可以用’+’的操作符，例如”” + ‘a’ +’b’,甚至也可以”” +’a’+”ter”  String 的**substring(int startindex)**，不是subString(); |

**Java 中的HashSet**：

|  |
| --- |
| Clear(), remove(Object o), add(E e), isEmpty()  Iterator<E> iter = set.iterator() 🡺遍历整个set |

**Java 中的static 方法和成员变量**：

|  |
| --- |
| 在static方法中不能使用类的非static 变量，因此在做题的时候最好将helper方法都声明成非static方法；  类不能被static 修饰，但是一个静态类代表其方法或者字段都是静态的 |

**Java 中的HashMap**：

|  |
| --- |
| 当value是数组时，先定义好数组，再加入元素，即：  Map<Character,char[]> stringMap =new HashMap<Character, char[]>();  char[] aa ={'a','A','%'};  stringMap.put('a', aa); |

**Java 中的几个final类型**：

|  |
| --- |
| String , int , Boolean 等原始类型数据都是final的，如果传递给另一个函数，如果再函数体内改变了这些值，则会另外创建，而并不是再原变量上进行操作。这点和python是一样的，需要特别注意。  但是数组如果作为参数传递给函数，则会直接左右与原数组，不管是String数组还是int数组等等。  这点有时候c++的指针还是挺方便的，直接修改其值。 |

**Java 中的几个默认值**：

|  |
| --- |
| boolean -> false , 对于一个boolean数组同样初试默认值是false  int byte long short -> 0 float ->0.0f double->0.0d  char -> ‘u000’ String -> null |

**Java 中的隐士转换**：

|  |
| --- |
| a = a + b 与 a += b 的区别(答案) **+= 隐式的将加操作的结果类型强制转换为持有结果的类型**。如果两这个整型相加，如 byte、short 或者 int，首先会将它们提升到 int 类型，然后在执行加法操作。如果加法操作的结果比 a 的最大值要大，则 a+b 会出现编译错误，但是 a += b 没问题，如下： byte a = 127; byte b = 127; b = a + b; // error : cannot convert from int to byte b += a; // ok （译者注：这个地方应该表述的有误，其实无论 a+b 的值为多少，编译器都会报错，因为 a+b 操作会将 a、b 提升为 int 类型，所以将 int 类型赋值给 byte 就会编译出错）  from <http://www.importnew.com/17232.html> |

**Java 中的堆和栈**：

|  |
| --- |
| JVM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享，而堆被整个 JVM 的所有线程共享 |

**a == b 和 a.equals(b)的区别：**

|  |
| --- |
| 如果 a 和 b 都是对象，则 a==b 是比较两个对象的引用，只有当 a 和 b 指向的是堆中的同一个对象才会返回 true，而 a.equals(b) 是进行逻辑比较，所以通常需要重写该方法来提供逻辑一致性的比较。例如，String 类重写 equals() 方法，所以可以用于两个不同对象，但是包含的字母相同的比较。 |

**Final,finally,finalize:**

|  |
| --- |
| final 是一个修饰符，可以修饰变量、方法和类。如果 final 修饰变量，意味着该变量的值在初始化后不能被改变。finalize 方法是在对象被回收之前调用的方法，给对象自己最后一个复活的机会，但是什么时候调用 finalize 没有保证。finally 是一个关键字，与 try 和 catch 一起用于异常的处理。finally 块一定会被执行，无论在 try 块中是否有发生异常。 |

**Java 怎么打印数组**：

|  |
| --- |
| 你可以使用 Arrays.toString() 和 Arrays.deepToString() 方法来打印数组。由于数组没有实现 toString() 方法，所以如果将数组传递给 System.out.println() 方法，将无法打印出数组的内容，但是 Arrays.toString() 可以打印每个元素。 |

**Java 中的Math.sqrt()**：

|  |
| --- |
| 注意输入后返回值都是double,因此要判断一个树是否是完全平方数,需要强制转换：  public boolean isValid(int num)  {  if(num <= 0) return false;  int tmp = (int)Math.sqrt(num);  return tmp\*tmp==num?true:false;  } |

**Java 中的怎么初始化特定值的数组**：

|  |
| --- |
| 需要借用Arrays.fill(array,value)函数,例如：  int[] res = new int[10];  Arrays.fill(res,100);//初始化了每个值都是100的一个数组 |

**Java 中操作字符**：

|  |
| --- |
| 第一反应是字符相减就是对应的数字，例如：  ‘9’ – ‘0’ = 9  ‘b’ – ‘a’ =1,  对于字符串相关的问题通用的。 |

**Java list中null的问题**：

|  |
| --- |
| 在list中，可以插入null值，而且是计入总长度的，例如  ArrayList<Integer> cur = new ArrayList<>();  cur.add(null);  cur.add(null);  最后得到总长度是2； |

**Java 中HashMap 中的put方法**：

|  |
| --- |
| 当hashmap 中put方法中key存在于原来的map中，则会覆写原来的value，如：  HashMap<Integer,Integer> dict = new HashMap<>();  dict.put(1,1);  dict.put(1,2);//最后1->2 |

**Java 中范型的类型问题**：

|  |
| --- |
| List<Object> list=new ArrayList<String>();  List<String> list2=new ArrayList<Object>();  两种方法都不能通过编译，因为范型只针对于编译的过程，编译器发现无法convert 相关类型时候（can't convert ArrayList<String> to List<Object>）,就抱错了。 |

**Java 中object的方法**：

|  |
| --- |
| getClass(),hashCode(),equals(),clone(),toString(),notify(),notifyAll(),wait(),finalize(),  上述的hashCode()方法，如果不重写会怎样：因为object的hashcode方法是根据内存地址来hash的，如果你需要根据对象的某个属性来存取，如果不重写hashcode()方法，会无法找到对象，重写hashcode方法一般都要重写equals（）方法 |

**Java 中string 的trim()方法**：

|  |
| --- |
| str.trim()方法是将字符串的头和尾部的空格去掉，注意其有返回值的，因此如果要处理愿字符串需要：str = str.trim(); |

**Java 中初始化的问题**：

|  |
| --- |
| Int -> 0  Boolean -> false  Double -> 0.0 |

**对邻接链表的初始化问题**：不管那种发放，都需要先定义，再初始化每个元素

|  |
| --- |
| 1: 采用数组：先定义，再初始化  ArrayList<Integer>[] res = **new ArrayList[5];**  for(int i=0;i<res.length;i++)  {res[i] = new ArrayList<Integer>();}  2:采用hashmap来存取：  Map<Integer, List<Integer>> Adjacency\_List;  Adjacency\_List = new HashMap<Integer, List<Integer>>();  for (int i = 1 ; i <= number\_of\_vertices ; i++)  {  Adjacency\_List.put(i, new LinkedList<Integer>());  }  3:采用ArrayList<ArrayList<Integer>>来存储：  List<ArrayList<Integer>> res2 = new ArrayList<ArrayList<Integer>>();  for(int i=0; i<5;i++)  {  res2.add(new ArrayList<Integer>());  }  } |

**Java 中ArrayList 构造函数中带参数的构造方法**：

|  |
| --- |
| ArrayList<Integer> cur = new ArrayList<>(5);  注意此时的5是实现ArrayList底层数组数组的**初始容量**，但是此时的  cur.size()仍然是0；而且尽是初始容量，对于load factor大于某一thresh值，仍然要扩充容量。 |

**Java 中NIO 和IO的区别**：

|  |
| --- |
| 详细的解读：http://blog.csdn.net/shimiso/article/details/24990499  Java NIO是在jdk1.4开始使用的，它既可以说成“新I/O”，也可以说成非阻塞式I/O。下面是java NIO的工作原理： 1. 由一个专门的线程来处理所有的 IO 事件，并负责分发。  2. 事件驱动机制：事件到的时候触发，而不是同步的去监视事件。  3. 线程通讯：线程之间通过 wait,notify 等方式通讯。保证每次上下文切换都是有意义的。减少无谓的线程切换。  典型的例子：饭店招待服务员的例子 |

**Java 中怎样得到特定的输出形式**：例如(小数点后保留两位有效数字)

|  |
| --- |
| **java.text.DecimalFormat**类：  int test1 = 10;  long test2 = 7;  float test3 = 15.4f;  DecimalFormat formatTest = new DecimalFormat("#.00");  System.out.println(formatTest.format(test1)); //10.00  System.out.println(formatTest.format(test2));//7.00  System.out.println(formatTest.format(test3));//15.40 |

**Java 中怎样得某个范围内的随机数**：例如产生10-30之间的随机数

|  |
| --- |
| 使用Random类的nextInt(int bound)的变形：  // generate the random int value for the given bound  public static int generateRandom(int left,int right)  {  Random random = new Random();  int gap = right -left + 1;  return random.nextInt(Integer.MAX\_VALUE)%gap + left;  } |

**Java 中自定义一个可以比较的类**：

|  |
| --- |
| 例如设计一个演员类，其中包括身高，体重，编号，并且要求能够按照演员的体重排序：  1:Comparable接口：compareto(T t)      private static class Actor implements Comparable<Actor>{          int height;          int weight;          Actor(int w,int h){              weight=w;              height=h;          }          public int compareTo(Actor o) {              if(weight>o.weight)                  return 1;              if (weight<o.weight) {                  return -1;              }else {                  return o.height-height;              }          }  }  1:Comparator接口: compare(T t1,T t2)    public  static class people implements Comparator<people>{//体重\身高从小到大      int num;      int height;      int weight;      @Override      public **int compare(people o1, people o2)** {          // TODO Auto-generated method stub          if(o1.weight!=o2.weight)              return o1.weight-o2.weight;          else              return o2.height-o1.height;      } |