3 cubic metres 立方米

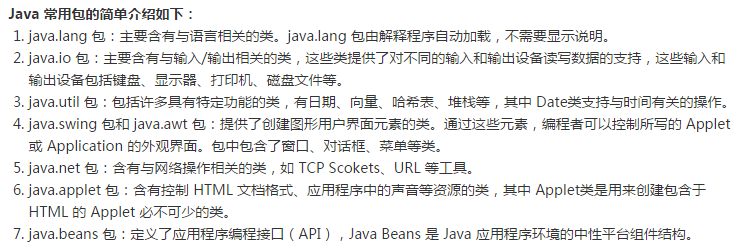
diameter 直径

centimeters 厘米

volume 体积

multiply 乘

常用：



定义一个数组：

int[] demoArray;

int[] demoArray = new int[10];

int len = demoArray.length;

for (int tmp:demoArray)

{

.....

}

String str="Hello";

str+="world";

String str = new String("Hello"); 实际创建了2个对象，一个"Hello" 一个是new申请的空间；

int len = str.length();

char a = str.charAt(2);

boolean b = str.contains("llo");

String str2 = str.replace("llo", "llo!!!");

String strArr[] = str.split("\_");

StringBuffer 不重新new对象，是线程安全的

StringBuffer str = new StringBuffer("dfafdf");

str.append("hhhh");

str.deleteAt(3);

str.delete(1,4);

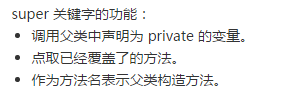
str.insert(3, "kkk");

StringBuilder 不重新new对象，不是线程安全的

速度：StringBuilder > StringBuffer > String

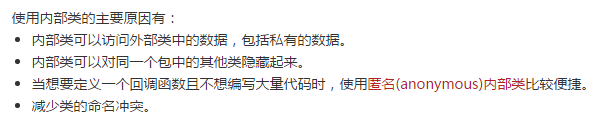
自动拆箱、装箱：

可以通过 Integer 类的构造方法将 int 装箱，通过 Integer 类的 intValue 方法将 Integer 拆箱



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

内部类：

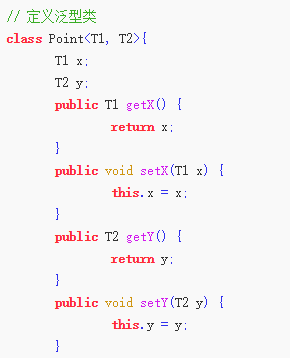


注意：必须先有外部类的对象才能生成内部类的对象，因为内部类需要访问外部类中的成员变量，成员变量必须实例化才有意义。

接口、抽象类：

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。



Point<Integer, Integer> p1 = **new** Point<Integer, Integer>();

Point<Double, String> p2 = **new** Point<Double, String>();

异常：

所有异常类型都是内置类Throwable的子类

紧接着Throwable下面的是两个把异常分成两个不同分支的子类。

Exception：

该类用于用户程序可能捕捉的异常情况。它也是你可以用来创建你自己用户异常类型子类的类。在Exception分支中有一个重要子类RuntimeException。该类型的异常自动为你所编写的程序定义并且包括被零除和非法数组索引这样的错误。(可检测的，不可检测的)

Error:

另一类分支由Error作为顶层，Error定义了在通常环境下不希望被程序捕获的异常。Error类型的异常用于Java运行时系统来显示与运行时系统本身有关的错误。堆栈溢出是这种错误的一例。本章将不讨论关于Error类型的异常处理，因为它们通常是灾难性的致命错误，不是你的程序可以控制的

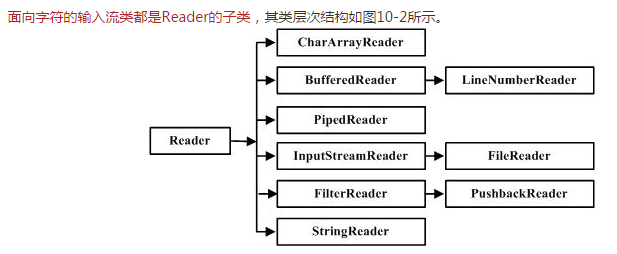
未检测异常：

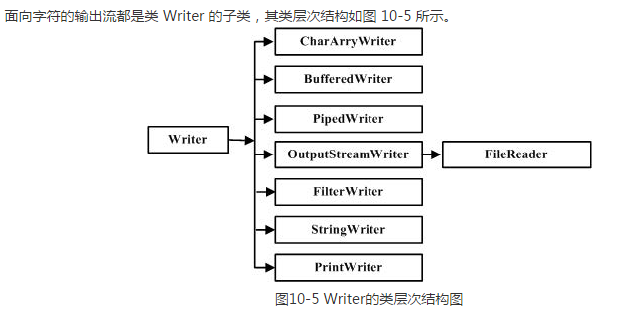


可检测异常：

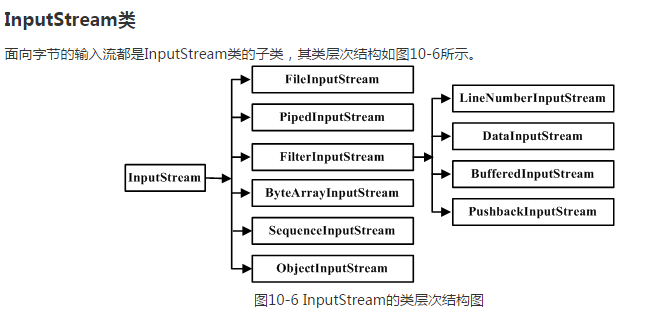


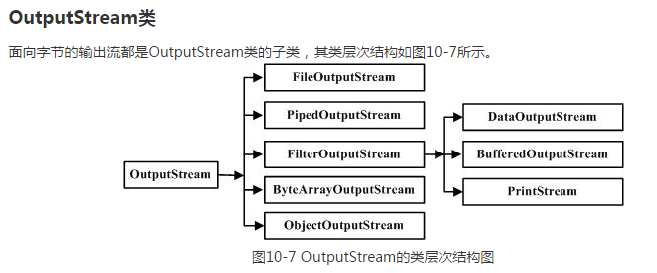
character stream: 文本文件 Reader/Writer：





byte stream: 文本和二进制文件 InputStream/OutPutStream：





1、读文件

FileWriter fout = new FileWriter("test.txt");

fout.write("2,3,4,5,done");

fout.close();

FileReader fin = new FileReader("test.txt");

Scanner scanner = new Scanner(fin);

while(scanner.hasNext())

{

if(scanner.hasNextDouble())

{

......

}

}

算法：

**1、Reverse a singly lined list**

1）Collections.reverse()

public static void main(Strings[] args)

{

String[] a = {"q","w","e","r","t"};

System.out.println("original:"+Arrays.asList(arrays));

List<String> tmp = Arrays.asList(arrays);

Collections.reverse(tmp);

Sysout(tmp);

}

2）traverse from tail to head: O(n)

public static void main(Strings[] args)

{

String[] a = {"q","w","e","r","t"};

System.out.println("original:"+Arrays.asList(arrays));

reverse(a);

}

public static String[] reverse(String[] a)

{

if(a==null||a.length<=1)

{

return a;

}

String[] reversedArray = new String[a.length];

for(int i=a.length-1,j=0; i>=0;i--,j++)

{

reversedArray[j] = a[i];

}

return reversedArray;

}

3) change the head and tail, change the next of head and the previous of tailO(n/2)

public static String[] reverse(String[] a)

{

if(a==null||a.length<=1)

{

return a;

}

int i=0;

int j=a.length-1;

for(int i=0,j=a.length-1;i<=a.length/2&&j>=a.length;i++,j--)

{

swap(a, i, j);

}

return reversedArray;

}

public static void swap(String[] a, int i, int j)

{

String tmp = a[i];

a[i] = a[j];

a[j] = tmp;

}

**2、Print a binary search tree level by level**

public class BsTreeNode

{

BsTreeNode left;

BsTreeNode right;

BsTreeNode parent;

int value;

//pre\_order:

public void pre\_order(BsTreeNode node)

{

if(node==null)

{

return node;

}else{

sysout(node.value)

pre\_order(node.left);

pre\_order(node.right);

}

}

}

**3、Max sub sequence problem**

只求最大和：

public static void main(Strings[] args)

{

int[] n;

int maxSum=0;

for(int i=0;i<n.length;i++)

{

int currentSum = 0;

for(int j=i;j<n.length;j++)

{

currentSum+=n[j];

if(currentSum>maxSum)

{

maxSum=currentSum;

}

}

}

sysout(maxSum);

}

//线性的算法O(N)

long maxSubSum4(const vector<int>& a)

{

       long maxSum = 0, thisSum = 0;

       for (int j = 0; j < a.size(); j++)

       {

              thisSum += a[j];

              if (thisSum > maxSum)

                     maxSum = thisSum;

              else if (thisSum < 0)

                     thisSum = 0;

       }

       return maxSum;

}

**3、Find the nth smallest/largest number in a unsorted array**

**4、Two sum, three sun problem**

**5、Java pow**

**double** pow(**int** a,**int** b){

**double** yourNum =1;

**for**(**int** i=0; i< b; i++)

 **if**(b>=0)

   yourNum\*=a;

  **else**

    yourNum/=a;

 **return** yourNum;

 }

**6、if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.**

1、冒泡

**public** **static** **void** bubbleRank2(**int** n[]){

               **for** (**int** i =0; i <n .length ;i ++)

                      **for**(**int** j =1;j <n .length -i;j ++)

                     {

                            **if**(n [j-1 ]>n [j ]){

swap(n[j-1], n[j]);

                           }

                     }

       }

**public** **static** **void** bubbleRank(**int** n[]){

               **for** (**int** i =0; i <n .length ;i ++)

                      **for**(**int** j =0;j <n .length -i -1;j ++)

                     {

                            **if**(n [j ]>n [j +1]){

                                   **int** tmp = n [j ];

                                   n[ j]= n[ j+1];

                                   n[ j+1]= tmp;

                           }

                     }

       }

2、快排

**public** **static** **void** quicksort(**int** n[], **int** left, **int** right) {

               **int** dp ;

               **if** (left < right ) {

                      dp = *partition*(n, left, right);

                      *quicksort*(n, left, dp - 1);

                      *quicksort*(n, dp + 1, right);

              }

       }

        **static** **int** partition (**int** n [], **int** left, **int** right) {

               **int** pivot = n [left ];

               **while** (left < right ) {

                      **while** (left < right && n[right] >= pivot)

                            right--;

                      **if** (left < right )

                            n[ left++] = n[ right];

                      **while** (left < right && n[left] <= pivot)

                            left++;

                      **if** (left < right )

                            n[ right--] = n[ left];

              }

               n[ left] = pivot;

               **return** left ;

       }

3、直接插入排序straight insertion sort

increase:

**public** **static** **void** main(**int** n[], **int** left, **int** right) {

int[] n=[1,5,2,6...];

straightSort(n);

}

public static straightSort(n)

{

for(int i=2;i<n.length;i++)

{

if(n[i]<n[i-1])

{

n[0]=n[i];

n[i]=n[i-1];

for(j=i-2;n[0]<n[j];j--)

{

n[j+1]=j[j];

n[j+1]=n[0];

}

}

}

}