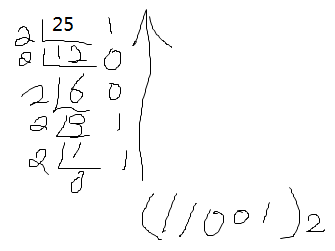
1. **Java语言释型还是编译型?还是两者都是?**

先编译再解释

编译型语言在程序执行之前，有一个单独的编译过程，将程序翻译成机器语言就不用再进行翻译了。解释型语言，是在运行的时候将程序翻译成机器语言，所以运行速度相对于编C/C++ 等都是编译型语言，而Java，C#等都是解释型语言。

1. **如何将10进制转换为2进制数据**



1. **如下写法哪些是不对的：**

**a. byte b = 30;**

**b. byte c = 500;**

**c. long d = 2343223;**

**d. float f = 3.14;**

b:不对，超出了byte的范围

d:不对，缺少f,应该写为3.14f或3.14F

byte计算时自动转换成int，所以a对

1. **浮点数能用于比较吗?下面可能打印什么结果:**

|  |
| --- |
| **float d1 = 423432423f;**  **float d2 = d1+1;**  **if(d1==d2 ){**  **System.out.println("d1==d2");**  **}else{**  **System.out.println("d1!=d2");**  **}** |

打印结果为: d1 == d2

float类型的数据在内存中的存储形式为科学计数法, 且有效位数为7位, 则d1为4.2343242E7, d2用科学计数法表示同样为4.2343242E7, 因此d1 == d2

1. **使用Scanner接收键盘输入，是否一定要加import Java.util.\*; ?**

需要导包

也可以写为import java.util.Scanner;

1. **引用类型是占用几个字节?**

引用数据类型占4个字节，用于存储对象的引用

1. **关系运算符中，能不能这么写："1<a<3"?**

不能，

应该写为a>1&&a<3

1. **运算符||和|的异同之处**

相同:

|| 与 | 都是逻辑运算符, 运算规则都为只要有一个为true,则直接返回true

不同:

|| 短路或 : 从左到右计算，如果符号左边的操作数为true,则不会继续计算,直接返回 true; 如果符号左边的操作数为false,则需要继续计算

| 逻辑或：操作数左右两边的操作数都需要计算,不会发生短路

|| 只是逻辑运算符

| 还可以是位运算符中的按位或符号

1. **5\*4最快的运算方式是?**

位运算5<<2

1. **4&5，4|5的结果分别是多少? 4&&5这个操作可行吗?为什么?**

4&5的结果为4

4｜5的结果为5

4&&5这个操作不可行，因为&&左右要求为boolean类型表达式,if后面的括号里面也只能是false或者true，不能是别的数字

1. **布尔类型能否自动转换为int? 如果不能，简述理由。**

不能

因为类型不匹配

1. **下面两种写法，哪个较好：**

**a. 70L\*60\*24\*365\*70\*20**

**b. 70\*60\*24\*365\*70\*20L**

第一种写法好;

第二种表达式的前半部分70\*60\*24\*365\*70的乘积有可能会超过int的取值范围, 导致精度损失

1. **Math.random()是什么意思?如果想获得15-20之间的随机数，怎么办?**

产生一个0-1之间的随机数，包含0但不包含1

(int)(Math.random()\*6)+15;

1. **switch中的表达式的结果需要是什么类型?**

byte,char,short,int

jdk1.5 加入了枚举

jdk1.7加入了String

1. **Java中，参数的传递使用值传递还是引用传递?**

值传递

1. **什么是方法的重载，有什么作用？**

方法重载:

[1]在同一个类中

[2]方法的名称相同

[3]参数列表不同

[4]与访问修饰符和返回值类型无关

[5]与异常无关

作用:

传递不同的参数实现相同的效果

1. **两同三不同指的是?返回值不同构成重载吗?形参名称不同构成重载吗?**

两同：同一个类中，方法名称相同

三不同：参数列表的参数的类型，个数，顺序不同

1. **【上机】使用递归算法完成阶乘算法**

public static int jiCheng(int number){

if (number==1) {

return 1;

}else{

return number\*jiCheng(number-1);

}

}

1. **栈的特点是?存放什么内容?**

栈的特点：先进后出

在函数中定义的一些基本类型的变量和对象的引用变量都在函数的栈内存中分配。

1. **堆得特点是?存放什么内容?**

**堆内存的特点：**

堆内存有内存地址，都是将内存的地址赋值给引用变量

堆内存变量无用后由垃圾回收机制不定时回收

堆内存会自动初始化

堆内存存放引用数据类，及new出来的都在堆内存里面

1. **【上机】画出如下程序的内存结构（课堂上的代码）：**

|  |
| --- |
| **//测试类和对象**  **public class TestObject{**  **public static void main(String[] args){**  **Car c1 = new Car();**  **c1.changeColor("红色");**  **c1.showColor();**  **System.out.println(Car.tyreNum);**  **System.out.println(c1.tyreNum);**    **Car c2 = new Car();**  **Engine e = new Engine();**  **e.speed = 1000;**  **e.weight = 10;**  **c2.engine = e;**  **c2.color="黑色";**  **c2.tyreNum = 10;**  **System.out.println(c1.tyreNum);**  **}**  **}**  **class Car {**  **static int tyreNum=4;**  **Engine engine;**  **String color; //char sequence :字符序列**    **void changeColor(String c){**  **color = c;**  **}**    **void showColor(){**  **System.out.println("我的颜色是："+color);**  **}**  **}**  **class Engine{**  **int speed;**  **int weight;**  **}** |

|  |
| --- |
|  |

1. **import Java.util.\*; 会不会降低程序运行速度?为什么?**

不会降低程序的运行速度

因为是导入该包下的所有的类，会降低编译速度

1. **import static 静态导入的作用是导入类还是导入类的静态属性和静态方法**

导入类的静态属性和静态方法

1. **多态的三个必要条件是什么?**

继承

方法重写

父类引用指向子类对象

1. **Java的方法绑定采用动态绑定还是静态绑定?**

静态方法，构造器，private方法，用关键字super调用的方法是静态绑定

通过对象调用的方法，采用动态绑定

1. **String类能不能被继承?为什么?**

String类不能被继承

因为String类是使用final修饰的类

1. **抽象类中能不能有普通方法？能不能定义构造方法?**

抽象类中可以有普通方法

抽象类中可以定义构造方法

1. **接口中只能定义常量和抽象方法吗?**

JDK1.7之前,接口中只能定义静态常量和公共的抽象方法

JDK1.8中,接口中可以定义静态方法

1. **接口描述了现实世界中什么逻辑?**

接口描述了现实世界是“如果你是…则必须能..”的思想

如果你是天使，则必须能飞;如果你是汽车，则必须能跑

1. **接口中的常量一定是public static final吗？抽象方法一定是public abstract 吗?**

是的

1. **如何对一个数组的多个对象按照不同的依据进行排序；**

对象实现Comparable接口

或者定义比较规则的类实现Comparator接口

1. **模拟实现Comparable和Comparator接口；**

**实现Comparable接口**

public class Student implements Comparable<Student>{

private int age;

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public int compareTo(Student o) {

return this.age-o.getAge();

}

}

**实现Comparator接口**

class AgeComparator implements Comparator<Student>{

@Override

public int compare(Student o1, Student o2) {

return o1.getAge()-o2.getAge();

}

}

1. **GC指的是什么?写出英文全称。垃圾回收机制中，程序员能不能调用垃圾回收器?**

GC:是指垃圾回收

英文:garbage collection

程序员无权调用垃圾回收器

1. **如果catch两个异常，一个是父类，一个是子类。这两个异常的catch顺序如何  
    确定?**

子类异常在前父类异常在后

1. **Throwable是一个类还是接口?**

Throwwable是一个类

1. **方法重写时，子类声明异常能否超出父类的范围?**

子类声明异常不能超出父类的范围

[1]父类没有声明异常，子类也不能

[2]不可抛出原有方法抛出异常类的父类或上层类

[3]抛出的异常类型的数目不可以比原有的方法抛出的还多(不是指个数)

1. **为什么需要包装类?包装类的作用是?**

因为java语言是面向对象的语言，但是java中的基本数据类型却不是面向对象的，而在实际的使用中经常需要将基本数据转化成对象，便于操作。比如说在集合中存储数据时，只能存储对象

作用：

[1]作为和基本数据类型对应的类类型存在，方便涉及到对象的操作

[2]包含每种基本数据类型相关的属性以及相关的操作方法

1. **为什么String类被称为不可变字符序列?从String类的源代码分析，给出解释。**

字符串是常量，它们的值在创建之后不能更改，String对象是不可变的，所以可以共享

String类的底层结构是char类型的数组value,而这个数组使用final进行修饰

1. **【上机】String类的equals方法跟Object的equals方法什么关系? 并详细阅 读String类的equals方法源代码，分析它的内部流程。**

String类的equals方法重写了Object类的equals方法，用于比较两个String对象的内容是否相同

String类的equals方法的源码分析:

如果两个String对象的内存地址（引用）相同，那么return true

如果两个String对象的内存地址（引用）不相同，那么进行类型判断,如果不是String类型，那么直接返回false,如果是String类型则进行向下类型转换，转换成String类型，然后变改char类型数组，比较两个数组对应位置上的内容是否相同，如果相同返回true,如果不同返回false

1. **"hamburger".substring(4, 8) 返回的结果是?**

urge

1. **【上机】分析下面代码的结果，并画出内存结构图，针对每个打印的结果给出文字  
    解释。**

|  |
| --- |
| **String s = "abc";**  **String ss = "abc";**  **String s3 = "abc"+"def"; //此处编译器做了优化!**  **String s4 = "abcdef";**  **String s5 = ss+"def";**  **String s2 = new String("abc");**  **System.out.println(s==ss);**  **System.out.println(s3==s4);**  **System.out.println(s4==s5);**  **System.out.println(s4.equals(s5));** |

System.out.println(s==ss);结果为true

System.out.println(s3==s4);结果为true

System.out.println(s4==s5);结果为false

System.out.println(s4.equals(s5));结果为true

|  |
| --- |
|  |

1. **StringBuffer和StringBuilder的联系是？区别是？**

联系：

StringBuffer和StringBuilder都是可变字符序列，底层数组结构都是char类型的数组

区别：

StringBuffer：jdk1.0版,线程安全，但是效率低

1. **如下的代码会造成什么后果?运行期间会产生多少个对象? 使用StringBuilder 修改这段代码。**

|  |
| --- |
| **String s = "";**  **for(int i=0;i<10000;i++){**  **s +=i;**  **}** |

会导致大量副本字符串对象存留在内存中，降低效率

运行期间会产生10001个对象

StringBuffer sb=new StringBuffer();

for(int i=0;i<10000;i++){

sb.append(i);

}

StringBuilder:jdk1.5版，线程不安全，但是效率高(自己看看源码)

1. **计算机中的时间是如何表示的？**

获取从1970-1-1 0:0:0到当前时间所经历的毫秒数，然后转换为日期或者时间253

1. **File类的方法mkdir跟mkdirs，有什么区别?**

mkdir:只能在已经存的目录中创建文件夹

mkdirs:可以在不存的目录中创建文件夹

1. **File类能代表一个目录吗? （自己手写一个遍历文件目录的代码）**

能

public static void main(String[] args) {

File file = new File("e:/教学1");

showTree(file, 1);

}

public static void showTree(File file, int level){

File[] files = file.listFiles();

for (File f : files) {

for(int i = 0; i < level; i++){

System.out.print("-");

}

if (f.isDirectory()) {

System.out.println(f.getName());

showTree(f, level+1);

}else{

System.out.println(f.getName());

}

}

}

1. **想取两个容器中元素的交集，使用哪个方法?**

retainAll(Collection c): 只保留当前集合（调用此方法的集合）与集合c（此方法的参数）中相同的元素。

1. **说出ArrayLIst、LinkedList、Vector的区别。**

ArrayLIst、LinkedList两者都实现了List接口，都具有List中元素有序、不唯一的特点。

ArrayList实现了长度可变的数组，在内存中分配连续空间。遍历元素和随机访问元素的效率比较高；

Snap4

LinkedList采用链表存储方式。插入、删除元素时效率比较高

Snap7

1. **定义Computer类，使用价格排序。(使用Comparable接口)**
2. 定义Computer类，实现Comparable接口：

**publicclass** Computer **implements**Comparable {

**privatedouble** price;//私有属性；

//构造方法；

**public** Computer(**double** price) {

**super**();

**this**.price = price;

}

//实现Comparable接口中的compareTo方法；

@Override

**publicint** compareTo(Object o) {

Computer c=(Computer)o;

**if**(**this**.price>c.price){

**return** 1;

}**elseif**(**this**.price<c.price){

**return** -1;

}**else**{

**return** 0;

}

}

//重写toString()方法；

@Override

**public** String toString() {

**return**"Computer [price=" + price + "]";

}

}

1. 加入TreeSet;

**import** java.util.Iterator;

**import** java.util.TreeSet;

**publicclass** Test {

**publicstaticvoid** main(String[] args) {

//创建TreeSet;

TreeSet<Computer> treeSet=**new** TreeSet<Computer>();

//创建Computer对象；

Computer computer1=**new** Computer(3000);

Computer computer2=**new** Computer(2650);

Computer computer3=**new** Computer(5878.8);

Computer computer4=**new** Computer(6000.78);

//将Computer对象加入到treeSet中；

treeSet.add(computer1);

treeSet.add(computer2);

treeSet.add(computer3);

treeSet.add(computer4);

//为treeSet创建迭代器；

Iterator<Computer> it=treeSet.iterator();

//遍历treeSet

**while**(it.hasNext()){

System.***out***.println(it.next());

}

}

}

结果：实现价格排序；

Computer [price=2650.0]

Computer [price=3000.0]

Computer [price=5878.8]

Computer [price=6000.78]

1. **equals返回true,hashcode一定相等吗?**

是的。

1. **HashSet和TreeSet的区别**

HashSet:

1. 存储结构：采用Hashtable哈希表存储结构
2. 优缺点：

优点：添加速度快，查询速度快，删除速度快

缺点：无序

TreeSet

1. 存储结构： 采用二叉树的存储结构

（2） 优缺点：

优点：有序（排序后的升序）查询速度比List快

（按照内容查询）

缺点：查询速度没有HashSet快

1. **使用HashSet存储自定义对象，为什么需要重写hashCode()和equals()？**

HashSet存储用的哈希表结构，哈希表需要用到hashCode()和equals()方法：

hashCode()产生hash值以计算内存位置；

当hash值相同时要调用equals()方法进行比较。

如果不重写，调用的是Object的hashcode，而Object的hashCode实际上是地址。系统类已经覆盖了hashCode方法。

所以HashSet存储自定义对象的化要重写hashCode()和equals()方法，目的是告诉程序去除重复元素的策略。

1. **使用TreeSet存储多个学生数据，实现按照不同属性值进行排序？**
2. 创建一个Student类, 实现Comparable接口；

**publicclass** Student **implements**Comparable {

//私有属性；

**privateint** id;

**privateint** score;

//getter和setter方法；

**publicint** getScore() {

**return** score;

}

**publicvoid** setScore(**int** score) {

**this**.score = score;

}

**publicint** getId() {

**return** id;

}

**publicvoid** setId(**int** id) {

**this**.id = id;

}

//构造方法；

**public** Student(){

}

**public** Student(**int** id, **int** score) {

**super**();

**this**.id = id;

**this**.score = score;

}

@Override

**public** String toString() {

**return**"Person [id=" + id + ", score=" + score + "]";

}

//实现compareTo方法，这里是用id比较；

**publicint** compareTo(Object obj){

Student other=(Student)obj;

**int** result=**this**.id-other.id;

**return** result;

}

}

1. 加入TreeSet;

**publicclass** Test2 {

**publicstaticvoid** main(String[] args) {

/\*

参看下面创建TreeSet时有两种方法：

**代码1：**使用无参构造方法TreeSet()，当 Student对象添加进去后，排序时的比较策略用的是Student内部实现的compareTo方法（内部比较器，此例中按照学生id排序）。

**代码2：**使用有参构造方法**new** TreeSet(scoreComp)；那么参数scoreComp是什么呢？它表示的是一个外部比较器的对象。在什么情形下使用呢？当用TreeSet实现排序时，我们不想用Student的内部比较器（也就是说不想用id排序），想用学生score排序？那怎么办呢？有同学说，把内部比较器改一下呗，改成用score排序的。但改来改去是不是不够灵活呢？我们有另外一种方法。还记得外部比较器吗？Student的内部比较器我们可以不改，再为Student类定义一个外部比较器（见以下代码中“定义外部比较器”的部分），定义外部比较器要实现Comparator接口中的compare(Object obj1,Object obj2)（如果忘记了就往前翻翻吧☺），在这个方法中实现用分数(score)比较。

以此类推，如果你想用其它属性比较（如年龄，姓名等），可以继续定义相应的外部比较器，使用方法参见代码2.

\*/

/\*

**代码1：**

Set treeSet=new TreeSet();

\*/

/\*

**代码2：**

ScoreComp scoreComp=**new** ScoreComp();//定义一个外部比较器的对象；

Set treeSet=**new** TreeSet(scoreComp);//把外部比较器对象作为TreeSet构造方法的参数；

\*/

Student p1=**new** Student(1,78);

Student p2=**new** Student(2,67);

Student p3=**new** Student(3,96);

Student p4=**new** Student(4,87);

treeSet.add(p3);

treeSet.add(p4);

treeSet.add(p1);

treeSet.add(p2);

Iterator it=treeSet.iterator();

**while**(it.hasNext()){

System.***out***.println(it.next());

}

}

}

//定义外部比较器；

**class** ScoreComp **implements**Comparator{

**publicint** compare(Object obj1,Object obj2){

Student p1=(Student)obj1;

Student p2=(Student)obj2;

**int** result=p1.getScore()-p2.getScore();//用学生分数排序；

**return** result;

}

}278