## 一、简答题

1. 基本数据类型的包装类有哪些

Byte Short Integer Long Character Boolean Float Double

1. String和StringBuffer有什么相同和不同点

String是字符串，定义后不可变

StringBuffer是字符串缓冲区，定义后可变

1. 详细描述==和equals的区别

==比较基本数据类型时比较值是否相等，比较引用类型时比较地址是否相等

Equals比较引用类型，内容是否相等（默认地址）

1. SimpleDateFormat的作用是什么，怎么使用？

规范日期格式，SimpleDateFormat s = new SimpleDateFormat（yyyy-MM-dd-HH-mm-ss）

1. List和Set有什么相同和不同

List为有序可重复集合

Set为无序不可重复集合

1. 异常的处理方式有哪些,写出几个常见的异常类？

系统自动抛、手动抛异常 ArraysIndexOutOfBoundsException、classCastException 、 IoException

1. 简述异常传递的过程

沿着方法调用方向反向传递

1. 如何自定义、使用一个异常

自定义异常类继承RuntimeException，继承父类构造，在自定义类中针对错误情况抛出异常，并建立自定义异常类对象

1. Properties是什么，怎么使用？

属性集合

Properties p = new Properties();

P.setProperties(“A”,”a”);

1. try.catch.finally分别是什么作用？

try运行可能异常的代码

Catch捕捉异常

finally无论是否有异常都执行的内容

1. throw与throws区别

throw 手动抛出异常

Throws 向上抛出异常 谁调用该方法谁处理

1. error与exception区别？

Error是不可挽救的错误

Exception是可以处理的异常

1. Java中集合框架的有哪些，分别有什么内容和特点？

Collection：单列集合，包括List、Set

有序有下标可重复List:ArrayList、Vector 、LinkedList

无序无下标不可重复Set:HashSet 、TreeSet 、HashLinkedSet

Key不能重复Value可重复Map：双列集合.HashMap 、HashTable、 Properties 、TreeMap

1. ArrayList、Vector、LinkedList的存储特性和性能？

ArrayList：不同步效率快不安全基于数组实现的集合 增删慢查询快

Vector：同步效率慢安全数组实现的集合

LinkedList：不同步效率快不安全基于链表实现的集合 增删快查询慢

1. Array与ArrayList有什么不一样？

Array是数组，长度固定

ArrayList是集合，长度可变，是储存对象的对象

1. Map有什么特点

Map是双列集合，将key映射到value，内容可以是null，Key不可重复，Value可重复，

1. Final、finally、finalize分别是什么作用

Final是修饰符，表示最终定义不可修改； finally是异常中无论是否有异常都执行的语句 ；

Finalize是对象被gc回收时自动调用的方法，来自于Object类

1. 简述TreeSet去重原理

TreeSet通过HashCode（）与equals（）两个方法去重，先比较HashCode是否相同，相同则再调用equals（）比较内容是否相同

1. break和return和System.exit(0)有什么不同

Break是退出循环；return是退出方法；System.exit（）是退出虚拟机

1. 简述HashSet去重的原理

HashSet通过HashCode（）与equals（）两个方法去重，先比较HashCode是否相同，相同则再调用equals（）比较内容是否相同

## 二、编码题

1. 键盘录入一个你的生日，计算到现在你存活了多少天

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("请输入出生时间（yyyy-MM-dd-HH-mm-ss）");

String t = sc.next();

SimpleDateFormat s = **new** SimpleDateFormat("yyyy-MM-dd-HH-mm-ss");

Date d1 = s.parse(t);

Date d2 = **new** Date();

System.***out***.println("已经存活毫秒数："+(d2.getTime() - d1.getTime()));

1. 创建一个List集合，存放5个学生对象，用4种方式遍历

ArrayList a = **new** ArrayList();

a.add("张三");

a.add("李四");

a.add("王五");

a.add("赵六");

a.add("老七");

System.***out***.println("===========for==========");

**for**(**int** i = 0 ; i < a.size() ; i++) {

System.***out***.println(a.get(i));

}

System.***out***.println("===========while==========");

**int** i = 0;

**while**(i<a.size()) {

System.***out***.println(a.get(i));

i++;

}

System.***out***.println("===========Iterator==========");

Iterator it = a.iterator();

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

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

}

System.***out***.println("===========foreach==========");

**for** (Object l : a) {

System.***out***.println(l);

}

1. 编写代码，比较ArrayList和LinkedList增删和查询1000000条数据的速度

**long** start =System.*currentTimeMillis*();

ArrayList a = **new** ArrayList();

**for**(**int** i = 0; i<1000000 ; i++) {

a.add(i);

}

**long** t1 =System.*currentTimeMillis*();

**for**(**int** i = 0; i<1000000 ; i++) {

a.get(i);

}

**long** t2 =System.*currentTimeMillis*();

System.***out***.println("ArrayList增删时间："+(t1-start));

System.***out***.println("ArrayList查询时间："+(t2-t1));

**long** t3 =System.*currentTimeMillis*();

LinkedList l = **new** LinkedList();

**for**(**int** i = 0; i<1000000 ; i++) {

l.add(i);

}

**long** t4 =System.*currentTimeMillis*();

**for**(**int** i = 0; i<1000000 ; i++) {

l.get(i);

}

**long** t5 =System.*currentTimeMillis*();

System.***out***.println("LinkedList增删时间："+(t4-t3));

System.***out***.println("LinkedList查询时间："+(t5-t4));

1. 创建Set集合，存储Person对象，按照年龄排序，遍历这个集合

TreeSet<Student> stus = **new** TreeSet<Student>();

stus.add(**new** Student("zhangsan", 12));

stus.add(**new** Student("lisi", 32));

stus.add(**new** Student("wangwu", 42));

stus.add(**new** Student("zhaoliu", 23));

Iterator it = stus.iterator();

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

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

}

}

}

**class** Student **implements** Comparable<Student>{

String name;

**int** age;

**public** Student(String name, **int** age) {

**super**();

**this**.name = name;

**this**.age = age;

}

@Override

**public** String toString() {

**return** "Student [name=" + name + ", age=" + age + "]";

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + age;

result = prime \* result + ((name == **null**) ? 0 : name.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Student other = (Student) obj;

**if** (age != other.age)

**return** **false**;

**if** (name == **null**) {

**if** (other.name != **null**)

**return** **false**;

} **else** **if** (!name.equals(other.name))

**return** **false**;

**return** **true**;

}

@Override

**public** **int** compareTo(Student s) {

**int** ret = **this**.age - s.age;

**if** (ret == 0) {

ret = **this**.name.compareTo(s.name);

}

**return** ret;

}

}

1. 公司有若干个工人(Worker), 每个工人都必须有基本工资,和绩效系数两个属性,

实际工资=基本工资\*绩效系数（绩效系数范围0.8—1.2，超出报异常）

* 1. 创建Worker类；
  2. 创建5个工人对象存入Map<String, Worker >集合中;
  3. 遍历Map集合;
  4. 计算工人的平均收入;

HashMap<String, Worker > m =**new** HashMap<String, Worker>();

m.put("张1", **new** Worker(5000,0.8));

m.put("张2", **new** Worker(6000,0.8));

m.put("张3", **new** Worker(7020,0.9));

m.put("张4", **new** Worker(8000,1.0));

m.put("张5", **new** Worker(9000,1.1));

Set<Entry<String, Worker>> s =m.entrySet();

Iterator<Entry<String, Worker>> it = s.iterator();

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

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

}

Set<String> k =m.keySet();

**double** total = 0;

**for** (String string : k) {

total += m.get(string).getSalary();

}

System.***out***.println("平均工资为："+total/m.size());

}

}

**class** Worker{

**int** salary;

**double** factor;

**public** Worker(**int** salary,**double** factor) {

**super**();

**this**.salary = salary;

**if**(factor>=0.8&&factor<=1.2) {

**this**.factor = factor;

}**else** {

**throw** **new** FactorOutOfBounds("绩效系数范围0.8—1.2，超出异常");

}

}

**public** **double** getSalary() {

**return** salary\*factor;

}

}

**class** FactorOutOfBounds **extends** RuntimeException{

**public** FactorOutOfBounds(String message) {

**super**(message);

}

}

1. 统计字符串dhfahdfjL,444555H./;;;中字母、数字、符号出现的次数

String s = "dhfahdfjL,444555H./;;;";

**byte**[] bb = s.getBytes();

**int** count = 0;

**int** c = 0;

**int** v = 0;

**for** (**byte** b : bb) {

**if**(b >=65 && b<=90) {

count++;

}**else** **if**(b>=97&&b<=122) {

count++;

}**else** **if**(b >=48 && b <= 57){

c++;

}**else** **if**(b>= 32 && b<=47) {

v++;

}**else** **if**(b>= 58 && b<=63) {

v++;

}**else** **if**(b>= 91 && b<=95) {

v++;

}**else** **if**(b>= 123 && b<=127) {

v++;

}

}

System.***out***.println("字母数量:"+count);

System.***out***.println("数字数量:"+c);

System.***out***.println("符号数量:"+v);

1. 遍历D盘中所有的文件和文件夹，使用递归实现

File f = **new** File("D:/");

**if**(f.exists() && f.isDirectory()) {

*getAllFiles*(f);

}**else** {

System.***out***.println("地址有误");

**return**;

}

}

**public** **static** **void** getAllFiles(File f) {

File [] files = f.listFiles();

**for** (File ff : files) {

**if**(ff.isFile()) {

System.***out***.println("\t文件:"+ff);

}**else** **if**(ff.isDirectory()) {

*getAllFiles*(ff);

}

}

}

}

1. 编写代码，复制本地文件(文件>4G)

FileInputStream i = **null**;

FileOutputStream o = **null**;

**try** {

i = **new** FileInputStream(**new** File("G:\\第一行代码Java 9787115448156.rar"));

o = **new** FileOutputStream(**new** File("C:/Users/ALIENWARE/Desktop/JAVA资料.rar"));

**byte**[] b = **new** **byte**[1024\*1024];

// 每次读取数据的长度

**int** len = -1;

**while** ((len=i.read(b)) != -1) {

o.write(b, 0, len);

}

}**catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

**if** (o != **null**) {

o.close();

}

**if** (i != **null**) {

i.close();

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

System.***out***.println("复制文件结束...");

}

}