第一章:Java 运行平台: JavaSE(J2SE)标准版平台(桌面开发和低端商务应用、Applet); JavaEE(企业)API、开发网络、web 应用JavaEE(J2EE)企业版平台、JavaME(J2ME)(嵌入式设备)(微型)移动设备; Java CARD SIM、ATM卡。JRE(运行环境)JDK(开发工具包)javac.exe编译器 java.exe解释器 jdb.exe调试器 javap.exe反编译。Java特点: 简单、面向对象、分布式、解释型、健壮、安全、架构中立、可移植、高性能、多线程、动态。Java 虚拟机: 字节代码,垃圾回收有一定滞后性.命令行编译: javac Hello.java→生成Hello.class字节码→运行java Hello。如何搭建Java开发环境:首先下载安装JDK 然后配置环境(1)配置PATH,操作系统运行环境的路径(2)配置CLASSPATH JAVA 运行应用程序时所需要的类包的路径(3)配置JAVA_HOME供需要运行JAVA的程序使用.第二章:标识符:字母数字 _$不能数字开头3.关键字:enum、transient、strictfp、volatile、native、(goto,const保留字)4.数据范围:077(八)0x3ABC(十六)5.Byte 1字节8位−2^7 ~2^7 − 1(byte运算必须转化byte s = byte(a+b)) 6.short 2字节16位 int 4字节32位 7.long 8字节64位(后面加l或L) boolean单个32位、数组中8位8.char 2字节0~65535(0~2^16-1)('1'、转义字符、'\u????'(四位十六进制)使用时是一个数字)'t'水平制表符'n'换行'f'表格符'r'回车'\"'双引号'\''单引号'\\'反斜线9.float 4字节1.4E-45 ~ 3.4E+38(后面要加f) 10.double 8字节4.9E-324~1.7E308(后面加d/D 可缺省)Long.Size获得位数 11.默认值:数据的都是0,char是'\u0000',String是null,boolean是false精度级别:byte short int long float double(左到右自动转换,右到左显式转换12.System.out.printf("%5.2f")输出小数部分最多6位的float数据 小数点后2位(会四舍) 14.要使用数组要分配空间。数组下标访问ArrayIndexOutOfBoundsException 15.%可以对浮点数用,3.5%2 =1.5,++可以对浮点型用17.Instanceof 确定某个对象是否属于某个类18.<<左移右补0>>右移左按符号补0或1 >>>不带符号右移左补0 19.switch中表达式是整型/字符型; 20.A.基本(简单)数据(String不是)数值: 整型btye、short、int、long; 浮点型、double; 字符(char); 布尔(boolean)B.复合数据: 类接口数组 21数组拷贝是引用arraycopy方法arraycopy(Object src,int srcPos,Object dest,int destPos,int length)第三章: 1.面向对象特性:封装继承多态3.没有实体的对象使用会有NullPointerException(运行出错) 4.final修饰常量,不占内存,不能通过类名访问。用static修饰的成员变量称为静态变量(类变量) 5.重载:参数列表不同即可.(返回值可以不同13.java.lang.Number中有基本类型的类包装Byte Short Long Integer Float Double Character. 7.面向对象好处: 模块化、信息隐藏、代码重用、易调试.8.Java中只有按值传递,没有按引用传递9.this代表对当前对象的引用,不能出现在类方法中; 在构造方法中使用this调用其他构造方法时须放在方法第一11.用static修饰的成员方法称为静态方法(类方法),不使用则称为实例方法; 类方法可通过类名或对象名来调用继承与接口1.重写/覆盖: 名字返回类型参数个数和类型全相同。2.重载是编译时处理\覆盖是运行时处理.3.访问权限要大于等于被覆盖方法的权限,例外列表要小于等于被覆盖方法的例外列表。9.abstract类可有构造方法10.abstract类中有abstract方法(只声明不实现)和非abstract方法10. interface声明,只含常量、方法声明,不含方法体,可以声明标量,不可实例化11.子类不能继承父的初始化方法。当用子类的初始化方法创建一个子类的实例对象时,这个子类声明的所有成员变量都被分配了内存空间,直接父类和所有的祖先类的成员变量也都分配了内存空间。17.如果父类实现了某个接口,则其子类也就自然实现这个接口。18.非static内部类的不可以声明静态变量和静态方法,只可以申请静态常量 20.在类A中的内部类B,若B中要使用B创建对象,则需要B obj = new A().new B(); 21 匿名类一定是内部类,可以继承,匿名类不可以声明静态成员变量和静态方法。常用:方法参数(new 类名or 接口名{重写方法})22.异常类: Exception的相应子类23.上转型变量:不能通过上转型对象变量访问子类对象实体中的成员变量和成员方法; 通过上转型对象变量访问子类对象实体重写的父类成员方法,执行的代码是子类重写的方法体; 可通过上转型对象变量访问父类被隐藏的成员变量; 可通过强制类型转换将上转型对象变量转换为子类对象变量24.super:子类方法通过使用super 调用父类中被子类隐藏的成员变量和覆盖的成员方法/子类的初始化方法中使用super调用父类的初始化方法(必须在子类的初始化方法的第一)25.类方法的方法体中不能有与类的对象有关的内容1.类方法中不能引用对象变量不能调用类的对象方法不能调用super this关键字; 4.类方法不能被覆盖。String表示一个UTF-16格式的2byte字符串 2. java.lang.String字符串构造String (char a[]) String(char a[], int startIndex, int count) String类提供了将字符串存放到数组中的方法: public void getChars(int start, int end, char c[], int offset) **String类的实体不可修改,StringBuffer类可修改构造**: StringBuffer() StringBuffer(int size) StringBuffer(String s) append方法char charAt(int index): void setCharAt(int index, char ch): StringBuffer insert(int index, String str) public StringBuffer reverse() StringBuffer delete(int startIndex, int endIndex) StringBufferreplace(int startIndex, int endIndex, String str) **StringBuffer与StringBuilder**功能几乎完全相同 StringBuffer是线程安全的,StringBuilder不是线程安全的 如果字符串缓冲区被单个线程使用(这种情况很普遍),建议优先采用StringBuilder,因为效率高–如果需要多线程同步,则建议使用StringBuffer StringTokenizer类 **StringTokenizer(String s)**: 为字符串s构造一个分析器。使用默认的分隔符集合,即空格符(多个空格被看做一个空格)、换行符'\n'、回车符'\r'、tab 符'\t'、进纸符'\f'– **StringTokenizer(String s, String delim)**: 为字符串s构造一个分析器,参数delim中的字符被作为分隔符(Pattern p; Matcher m; String input ="0A1A2A3A4A5A6A7A8A9"; p=Pattern.compile("\\dA\\d"); m = p.matcher(input);)正则表达式: \d 0~9的任何一个数字;\D任何一个非数字字符;\s空格类字符,'\t', '\n', '\x0B', '\f', '\r' ;\S非空格类字符;\w可用于标识符的字符(不包括类元符号);\W 不能用于标识符的字符;? 0 次或1次; * 0 次或多次;+ 1次或多次;{n} 恰好出现n次;{n,}至少出现n次;{n, m}出现n次至m次正则表达式匹配while(m.find()) str = m.group()字符串的替换 public String replaceAll(String regex, String replacement)字符串的分解public String[] split(String regex)常见类集合类与泛型java.util.*;: 1.Java中集合分为两大类,实现了Collection接口(包括List:ArrayList,LinkedList,Vector和Set:HashSet,TreeSet,LinkedHashSet)和实现了Map接口(包括HashMap,TreeMap,HashTable) 2 ArrayList(int capaticy=16) 3.ArrayList<E> :add(E) add(index,E) get(index) Object clone()强转 remove(int i) bool contains(Object) indexOf(Object)(-1)isEmpty()lastIndexOf() set(index,value) removeRange(start,end) size() Object[] toArray() LinkedList: addFirst(),getFirst(),removeFirst()(or Last)4.Vector最安全。线程安全版List list =Collections.synchronizedList(new ArrayList<>()/或者为LinkedList<>())5.HashSet<E>(不保证元素顺序不变化)如果是E是自定义对象,则需要在对象中重写int hashCode{return 自定义hash编码;}boolean equals(Object obj){if(obj.hashCode()==this.hashCode())return true;return false;} 线程安全:**Set set**=Collections.synchronizedSet(new HashSet<>()); 操作:add,remove,contains,size,clone无get,用迭代器获得元素(Treeset<String>is=new TreeSet<String>();Iertator<String> it=is.iterator()返回迭代器while(it.hasNext){it.next();})集合操作:A.addAll(B)并retainAll 交removeAll()差结果在A 中6.TreeSet<E>是有序的集合若E为自定义类,需要实现Comparable<E>接口重写ComparTo方法public int compareTo(E obj) { returnobj.attr >this.attr?-1:(obj.attr==this.attr?0:1);} 升序.String的比较:标点最前,然后大写字母,然后小写。线程同步SortedSet s = Collections.synchronizedSortedSet(newTreeSet<>());7.HashMap无序快TreeMap有序慢LinkedHashMap有序快8.HashMap<T,E>注意若T为自定义类,要跟HashSet<>一样重写两个方法操作:put(Key,Value) get(Key)containsKey(Key) containsValue(Value) remove(Key) Set set = hashmap.KeySet() Collection values = hashmap.values()可用迭代器Iterator 或ArrayList包装强制转换values 9.TreeMap要在自定义的Key类中实现Comparable 接口(同上)或者在构造TreeMap时提供一个比较器对象参数,比较器为实现泛型接口Comparator<E>的类,方法:public int compare(E o1,E o2){ return o1.attr>o2.attr?1:(o1.attr==o2.attr?0:-1);} 线程安全: Map m=Collections.synchronizedMap(new HashMap(..))**Date类**: System 类的静态方法public long currentTimeMillis()获取系统当前时间SimpleDateFormat(String pattern) 常用时间元字符**Calendar类**: getInstance()可以初始化一个日历对象异常处理java.lang.*; java.io.*; java.util.*; java.net.*: 2.异常-生成异常对象-交给JVM(过程称为抛出异常)JVM寻找(遍历栈)能处理该异常的方法,交给这个方法(捕获异常)(默认处理是输出异常信息终止程序)3.标准异常类都是Exception的子类4.Throwable(java.lang 包)直接子类:Error:OutOfMemoryError, ThreadDeath(致命错误,与JVM 相关,由系统处理如线程死亡,内存溢出)Exception(异常) 5.Exception(String)Exception()6.异常的方法printStackTrace()输出异常的信息7. try- catch-finally语句块finally 块是个可选项(一定执行)8.常见异常:运行异常ArithmeticException--ArrayIndexOutOfBoundsException—NullPointerException--ClassCastException(类型转换异常) –NumberFormatException 非运行异常:IOException\FileNotFoundException\SQLException9.自定义异常类:继承Exception,构造函数(String str),并且要super(str) 10.Exception分为运行时异常RuntimeException(不检查异常)和非运行时异常(检查异常) 11.子类重写父类的带抛出异常的方法:1.不声明抛出异常2 声明抛出的异常必须是其父类方法声明抛出的那种异常或者其子类异常 线程程序是一段静态的代码,它是应用软件执行的蓝本。进程是程序的一次动态执行过程,它对应了从代码加载、执行至执行完毕的一个完整过程,这个过程也是进程本身从产生、发展至消亡的过程。线程是比进程更小的执行单位。一个进程在其执行过程中,可以产生多个线程,形成多条执行线索,每条线索,即每个线程也有它自身的产生、存在和消亡的过程,也是一个动态的概念。**线程的4种状态**: 新建、运行、中断、死亡 **线程优先级**: setPriority(int grade) 1-10 **Runnable接口**(String s1="treasurer"; // 会计String s2="cashier"; // 出纳Bank bank = new Bank(s1,s2);Thread zhang;Thread cheng;zhang = new Thread(bank); // 目标对象bank cheng = new Thread(bank); // 目标对象bank zhang.setName(s1);cheng.setName(s2); bank.setMoney(120); zhang.start(); cheng.start();)(public void run(){while(true) {money = money-10;if(Thread.currentThread(). getName().equals(name1)){System.out.println(name1 + ": " + money); if(money<=100) {System.out. println(name1 + ": Finished"); return; } }else if(Thread.currentThread(). getName().equals(name2)){ System.out.println(name2 + ": " + money);if(money<=60){ System.out.println(name2 + ": Finished"); return; } } try{ Thread.sleep(800); } catch(InterruptedException e) {}} } })**1.**Thread构造方法中的参数是一个Runnable类型的接口,因此,在创建线程对象时必须向构造方法的参数传递一个实现Runnable接口的类的实例,该实例对象称作所创建线程的目标对象,2.关于run()方法中的局部变量 对于具有相同目标对象的线程,当其中一个线程享用CPU资源时,目标对象自动调用接口中的run()方法,当轮到另一个线程享用CPU 资源时,目标对象会再次调用接口中的run()方法。不同线程的run()方法中的局部变量互不干扰,一个线程改变了自己的run()方法中局部变量的值不会影响其他线程的run()方法中的局部变量。**线程的常用方法**: start、run、sleep、isAlive、currentThread、interrupt **线程同步**是指多个线程要执行一个synchronized修饰的方法,如果一个线程A在占有CPU资源期间,使得synchronized方法被调用执行,那么在该synchronized方法返回之前(即synchronized方法调用执行完毕之前),其他占有CPU资源的线程一旦调用这个synchronized方法就会引起堵塞,堵塞的线程要一直等到堵塞的原因消除(即synchronized方法返回),再排队等待CPU 资源,以便使用这个同步方法。**使用wait(),notify(),notifyAll()协调同步线程**: 使用wait()方法可以中断方法的执行,使本线程等待用notifyAll()方法通知所有由于使用这个同步方法而处于等待的线程结束等待。**线程联合**:一个线程A在占有CPU资源期间,可以让线程B调用join()方法和本线程联合,如: B.join(); 我们称A在运行期间联合了B。如果线程A在占有CPU资源期间一旦联合B线程,那么A线程将立刻中断执行 **守护线程**: 一个线程调用void setDaemon(boolean on)方法可以将自己设置成一个守护(daemon)线程,例如: thread.setDaemon(true);当程序中的所有用户线程都已结束运行时,即使守护线程的run()方法中还有需要执行的语句,守护线程也立刻结束运行。文件及输入输出流**9.1** 文件构造方法文件的属性目录文件的创建与删除运行可执行文件(Runtime ec = Runtime.getRuntime(); File file = new File("C:\\windows", "Notepad.exe"); ec.exec(file.getAbsolutePath());) **9.12** 使用Scanner解析文件创建Scanner对象,并指向要解析的文件(File file = new File("hello.java"); Scanner scanner = new Scanner(file);)使用useDelimiter方法指定正则表达式作为分隔标记(while(scanner.hasNextDouble()){ fare = scanner.nextDouble(); sum = sum+fare;}) **9.3** 文件字符流 FileReader类–int read() - int read(char b[ ])–int read(char b[ ], int off, int len): 读取len个字符并存放到字符数组b中,返回实际读取的字符数目; 如果到达文件的末尾,则返回-1。其中,off参数指定read方法从字符数组b中的什么地方存放数据FileWriter类public void write(char b[]) -public void write(char b[], int off, int len) -void write(String str)- void write(String str,int off, int len) **9.5 缓冲流**创建一个BufferedReader对象然后,input 调用readLine()顺序读取文件的一行。(FileReader fr = new FileReader("Student.txt"); BufferedReader input = new BufferedReader(fr););write(String s) **9.2** 文件字节流FileInputStream类int read(byte b[ ]);FileOutputStream类void write(byte b[]) **9.8** 数据流DataInputStream类和DataOutputStream类DataInputStream(InputStream is)、DataOutputStream(OutputStream os) **9.9** 对象流 ObjectInputStream 类和ObjectOutputStream 类 **9.10** 序列化和对象克隆使用对象流很容易获取一个序列化对象的深度克隆(原对象有引用型变量的时候)•我们只需将该对象写入到对象输出流,后用对象输入流读回的对象就是原对象的一个深度克隆。**9.11** 随机读写流RandomAccessFile(String name, String mode) **9.13**文件锁 JDK1.4增加了一个FileLock类,该类的对象称做文件锁。RondomAccessFile创建的流在读写文件时可以使用文件锁,那么只要不解除该锁,其它线程无法操作被锁定的文件。1. RandomAccessFile input = new RandomAccessFile ("Example.java", "rw"); 2.FileChannel channel = input.getChannel(); 3. FileLock lock = channel.tryLock(); **9.6数组流字节输入流**: ByteArrayInputStream(byte[] buf, int offset, int length)字节输出流: ByteArrayOutputStream(int size) public byte[] toByteArray(): 返回输出流写入到缓冲区的全部字节 **9.7字符串流构造**方法: public StringReader(String s) public int read(char[] buf, int off, int len) StringWritter(int size); public void write(String str, int off, int len) GUI编程**1** AWT 组件与Swing 组件概述AWT是Abstract Window Toolkit(抽象窗口工具包)的缩写。javax.swing包提供了Swing组件,大部分组件是轻组件,没有同位体,把与显示组件有关的许多工作和处理组件事件的工作交给相应的UI代表来完成。**2** JFrame窗体 **5**中间容器JPanel面板JScrollPane滚动窗格JSplitPane拆分窗格JLayeredPane分层窗格 **对话框JDialog**类对话框分为无模式(modeless)和有模式(model)两种。无模式对话框处于激活状态时,程序仍能激活它所依赖的窗口或组件,它也不堵塞线程的执行。有模式对话框处于激活状态时,只让程序响应对话框内部的事件,程序不能再激活它所依赖的窗口或组件,而且它将堵塞当前线程的执行,直到该对话框消失不可见。JOptionPane类: 输入对话框showInputDialog消息对话框showMessageDialog确认对话框showConfirmDialog颜色对话框ColorshowDialog文件对话框showOpenDialog 、showSaveDialog **3**菜单组件JMenuBar菜单条JMenu菜单JMenuItem菜单项JTextField文本框JPasswordField密码框 发生ActionEvent事件的事件源对象获得监视器方法是: addActionListener(ActioListener listener); JTextArea文本区 **4**布局设计FlowLayout(水平垂直Jpanel默认)、BorderLayout(默认,东西南北)、CardLayout(选项卡)、GridLayout(网格区域)、BoxLayout(盒式布局) **14**窗口事件WindowListener接口WindowAdapter适配器 **15**鼠标事件MouseListener接口与MouseMotionListener接口网络编程**1** URL类 一个URL对象通常包含最基本的三部分信息: 协议、地址、资源。常用的http、ftp、file 协议都是JVM支持的协议–地址必须是能连接的有效的IP地址或域名(host name)资源可以是主机上的任何一个文件public URL(String spec) throws MalformedURLException **2**读取URL中的资源URL对象调用InputStream openStream() 该方法可以返回一个输入流,该输入流指向URL对象所包含的资源。通过该输入流可以将服务器上的资源信息读入到客户端。**3** 显示URL资源中的HTML文件JEditorPane类 **4** 处理超链接JEditorPane对象调addHyperlinkListener(HyperlinkListener listener)获得监视器。监视器需实现

HyperlinkListener 接口,该接口中的方法是void hyperlinkUpdate(HyperlinkEvent e) **5** InetAddress类获取Internet上主机的地址 InetAddress类的静态方法: getByName(String s); **6套接字**Socket套接字连接就是客户端的套接字对象和服务器端的套接字对象通过输入、输出流连接在一起(1) 服务器建立ServerSocket对象(2)客户端创建Socket 对象(3) 流连接 **7**使用多线程处理套接字连接 **8**UDP数据报 基于UDP 通信的基本模式是(1)将数据打包,称为数据包(好比将信件装入信封一样),然后将数据包发往目的地。(2)接收别人发来的数据包(好比接收信封一样),然后查看数据包中的内容。byte data[ ]="近来好吗".getByte(); InetAddtress address =InetAddtress.getName ("www.sian.com.cn"); DatagramPacket data_pack = new DatagramPacket(data, data.length, address, 5678); (2)发送数据 **9**广播数据报1.设置组播地址2.创建多点广播套接字3.设置广播的范围4.加入组播组5.广播数据和接收数据 **10**让一个虚拟机上的应用程序请求调用位于网络上另一处虚拟机上的对象 (1)Remote接口 必须扩展Remote接口 定义Remote的子接口是RemoteSubject-public interface RemoteSubject extends Remote (2)远程对象-public class RemoteConcreteSubject extends UnicastRemoteObject implements RemoteSubject 存根（Stub）的字节码是RemoteConcreteSubject_Stub.class (3)启动远程对象服务-RemoteConcreteSubject remoteObject = new RemoteConcreteSubject(); Naming.rebind("rmi://127.0.0.1/rect", remoteObject);

```
public class Server {
    public static void main(String args[]) {
        ServerSocket server = null;
        Socket socketAtServer = null;
        DataOutputStream out = null;
        DataInputStream in = null;
        try {
            server = new ServerSocket(4333);
        } catch (IOException e1) {
            System.out.println("" +e1);
        }
        try {
            socketAtServer = server.accept();
            in = new DataInputStream(socketAtServer.getInputStream());
            out = new DataOutputStream(socketAtServer.getOutputStream());
            while (true) {
                int m = 0;
                m = in.readInt();
                out.writeInt(m * 2);
                System.out.println("Server received: " + m);
                Thread.sleep(500);
            }
        } catch (IOException e) {
            System.out.println("" + e);
        } catch (InterruptedException e) {
        }
    }
}
```

```
public class Client {
    public static void main(String args[]) {
        Socket socketAtClient;
        DataInputStream in = null;
        DataOutputStream out = null;
        try {
            socketAtClient = new Socket("localhost", 4333);
            in = new DataInputStream(socketAtClient.getInputStream());
            out = new DataOutputStream(socketAtClient.getOutputStream());
            out.writeInt(1);
            while (true) {
                int m2 = 0;
                m2 = in.readInt();
                out.writeInt(m2);
                System.out.println("Client received: " + m2);
                Thread.sleep(500);
            }
        } catch (IOException e) {
            System.out.println("Unable to connect to the server");
        } catch (InterruptedException e) {
        }
    }
}
```

```
public class TCPNetTime {
    public static void main(String[] args) {
        Socket socket = null;
        try {
            socket = new Socket("time.nist.gov", 13);
            InputStream is = socket.getInputStream();
            Scanner in = new Scanner(is);

            while (in.hasNextLine()) {
                String line = in.nextLine();
                System.out.println(line);
            }
        } catch (Exception e) {
            System.out.println(e);
        }
        if (socket != null) {
            try {
                socket.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }
}
```

```
class Account {//银行取钱 多线程同步
    private int count = 0;
    public synchronized void save(int howMany) {
        while (count > 10000) {
            try {this.wait(); //release the lock
            } catch (Exception e) {}
        }
        count += howMany;
        System.out.println("save"+howMany+" total"+count);
        this.notifyAll();
    }
    public synchronized int take(int howMany) {
        while (count < howMany) {
            try {
                long id = Thread.currentThread().getId();
                System.out.println("no enough money");
                this.wait();   //release the lock
            } catch (Exception e) {}
        }
        count -= howMany;
        long id = Thread.currentThread().getId();
        System.out.println(id+"take"+howMany+"remained"+count);
        this.notifyAll();
        return howMany;
    }
}
```

```
class Father extends AbstractStoppable implements Runnable {
    private Account acc;//Son一样
    public Father(Account acc) {
        this.acc = acc;
    }
    public void run() {
        while (!isStop()) {
            try {Thread.sleep( millis: 1000);}
            catch (Exception e) {}
            acc.save( howMany: 5000);
        }
    }
}
```

```
public class BankApp {
    public static void main(String[] args) {
        Account acc = new Account();
        Father f = new Father(acc);
        Thread fT = new Thread(f);
        int NUM_SON_THREADS = 5;
        ArrayList<Thread> sThreads = new ArrayList<>();
        ArrayList<Son> sList = new ArrayList<>();
        for (int i = 0; i < NUM_SON_THREADS; ++i) {
            Son s = new Son(acc);
            sList.add(s);
            sThreads.add(new Thread(s));
        }
        fT.start();
        for (Thread t : sThreads) {t.start();}
        int c = 0;
        while (++c < 10) {
            try {Thread.sleep( millis: 1000);}
            catch (Exception e) {}
        }
        f.setStop();
        for (Son s : sList) {s.setStop();}
    }
}
```

```
public class HashMapTest {
    public void compute() {
        HashMap<String, Double> h = new HashMap<>();
        h.put("北京烤鸭", 189.0);
        h.put("西芹炒肉", 12.9);
        h.put("酸菜鱼", 69.0);
        h.put("铁板牛柳", 32.0);
        Set<Map.Entry<String,Double>> s = h.entrySet();
        ArrayList<String> menu = new ArrayList<>();
        Double total = 0.0;
        for(Map.Entry<String, Double> e: s) {
            menu.add(e.getKey());
            total += e.getValue();
        }
        for(String m: menu) {
            System.out.print(m + ", ");
        }
    }
    public static void main(String[] args) {
        Menu m = new Menu();
        m.compute();
    }
}
```

```
public class WordCount{
    public static void main(String[] args) {
        try{//统计给定文本文件中的单词出现频率，最后按照字典顺序将统计结果打印出来
            FileReader fr = new FileReader( fileName: "readme.txt");
            BufferedReader br = new BufferedReader(fr);
            TreeMap<String, Integer> counts = new TreeMap<String,Integer>();
            String line = null;
            while( (line=br.readLine()) != null ){
                String [] words = line.split( regex: " ");
                for(String word:words){
                    if( counts.containsKey(word) ){
                        counts.put(word, (counts.get(word)+1));
                    }
                    else{
                        counts.put(word,1);
                    }
                }
            }
            fr.close();
            br.close();
            System.out.println(counts);
        }
        catch(IOException e){
            System.err.println(e);
        }
    }
}
```

编写一个服务器端程序ServerDemo.java，它能在8001 端口响应客户端的请求。如果客户端发来内容是字符串“Hello”，服务器将回复字符串“welcome”给客户端。服务器还需要将所有请求的请求时间和请求内容写入日志文件。客户端会将收到的内容打印到屏幕。

```
public class ServerDemo {
    public static void main(String args[]) {
        ServerSocket server = null;
        Socket sk = null;
        DataOutputStream out = null;
        DataInputStream in = null;
        FileOutputStream logfile = null;
        try {
            server = new ServerSocket( port: 8001);
        } catch (IOException e1) {
            System.out.println("ERROR:" + e1);
        }
        try {
            sk = server.accept();
            in = new DataInputStream(sk.getInputStream());
            out = new DataOutputStream(sk.getOutputStream());

            logfile = new FileOutputStream( name: "log.txt");
            while (true) {
                String s = in.readUTF();
                if ("Hello".equals(s)) {
                    out.writeUTF( str: "welcome");
                }
                logfile.write(((new Date()) + ": " + s).getBytes());

            }
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

```
public class ClientDemo {
    public static void main(String args[]) {
        String s = null;
        Socket mysocket;
        DataInputStream in = null;
        DataOutputStream out = null;
        try {
            mysocket = new Socket( host: "127.0.0.1", port: 8001);
            in = new DataInputStream(mysocket.getInputStream());
            out = new DataOutputStream(mysocket.getOutputStream());
            out.writeUTF( str: "Hello");
            s = in.readUTF();
            System.out.println(s);
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

创建两个线程对象，分别在屏幕上打印1-100 之间的奇数和偶数。

```
public class ThreadDemo{
    public static void main(String args[ ]){
        SubThread t1 = new SubThread(1,100);
        SubThread t2 = new SubThread(2,100);
        t1.start();
        t2.start();
    }
}
class SubThread extends Thread{
    int start = 0;
    int end = 0;
    SubThread(int start, int end){
        this.start = start;
        this.end = end;
    }
    public void run(){
        for(int i=start;i<=end;i+=2){
            System.out.println(i);
        }
    }
}
```

对输入的一维整型有序数组data[](但不确定足开序还是降序)，求出其中位数

```
int n=data.length;
int pos=n/2;
double result;
if (n%2==0) result=(double)(data[pos]+data[pos-1])/2;
else result=data[pos];
```

小明打算开发一个简单的模拟计算机的程序，模拟CPU\内存、总线、从内存取数

```
public class CPU {
    private Bus bus;
    public CPU(){}
    public void connectBus(Bus aBus) {
        bus=aBus;
    }
    public void add(int addr1, int addr2, int addr3) {
        int a =bus.read(addr1);
        int b= bus.read(addr2);
        bus.write( value: a+b, addr3);
        return;
    }
    public void assign(int value, int addr) {
        bus.write(value, addr);
    }public void show(int addr) {
        System.out.println(bus.read(addr));
    }
```

```
public class Memory {
    private int data[];
    Memory() {
        data=new int[100];
    }
    public boolean write(int value, int addr) {
        data[addr]=value;
        return true;
    }
    public int read(int addr) {
        return data[addr];
    }
}
public class Bus {
    private Memory mem;
    private CPU cpu;
    public Bus() {};
    public void connectCPU(CPU acpu) {
        cpu=acpu;
    }
    public void connectMemory(Memory mem) {
        this.mem=mem;
    }
    public boolean write(int value, int addr) {
        return mem.write(value, addr);
    }
    public int read(int addr) {
        return mem.read(addr);
    }
}
```

```
public class Computer {
    public static void main(String args[]) {
        CPU aCPU=new CPU();
        Bus aBus=new Bus();
        Memory aMem=new Memory();
        aCPU.connectBus(aBus);
        aBus.connectCPU(aCPU);
        aBus.connectMemory(aMem);
        aCPU.assign( value: 5, addr: 1);
        aCPU.assign( value: 4, addr: 2);
        aCPU.add(1,2,3);
        aCPU.show( addr: 3);
    }
}
```

```
interface Listener {
    void onEvent(int eventId);
}
class EventList implements Listener {
    private ArrayList<Integer> lst = new ArrayList<>();
    public void onEvent(int eventId) { lst.add(eventId); }
    public String toString() {
        int size = lst.size();
        if (size == 0) return "";
        StringBuffer b = new StringBuffer();
        for(int i=0; i<(size-1); ++i) {
            b.append(lst.get(i)); //3 points
            b.append(",");
        }
        b.append(lst.get(size-1));
        return b.toString();
    }
}
```

↓

1、 给定一个Listener 接口，它有如下方法：1) void onEvent(int eventId)：接收一个事件，其编号为eventId。试编写一个EventList 类，实现Listener 接口:1) void onEvent(int eventId)：收集所有收到的事件eventId.2) 并且实现方法String toString()，将收集到的所有事件按先后次序，转换为字符串（ 以逗号分隔每个事件id ）编写一个类RandomEventGenerator，实现下述方法：1) RandomEventGenerator(Listener listener)：存放listener 到this.listerner 2) void generate(int stop)：每隔一秒，随机生成一个[0,100]内的整数x，当x 为奇数时，调用Listener 接口方法onEvent(x)，当x=stop 时，返回。

试编写一个设备类Equipment，它有下述属性：

1）String name:名称 2）int num：数量 3）float price

再编写一个EquipmentCollection 类，实现下述方法：

1） void add(Equipment e)：增加一个设备

2） List<Equipment> sortByTotalPrice()：以总价由小到大的顺序排序所有增加的设备

3） public static void main(String[] args)：随机生成一组设备，

并依次增加到EquipmentCollection 实例中，测试sortByTotalPrice 是否正常工作。

```
class Equipment {
    public String name;
    public int num;
    public float price;
    public Equipment(String name, int num, float price) {
        this.name = name;
        this.num = num;
        this.price = price; //1 points
    }
    public String toString() {
        return name + "t_" + num * price + "__" + num + "_" + price;
    }
}
```

实现一个Master 类和Slaver 线程类，以缓冲区通信方式对每一批次的整数求和，具体如下：

Master 类实现下述方法：

1) Master():创建一个线程类Slaver 实例self.slaver。

2) loop(int low, int high, int num, int seconds):

2.1) 每隔seconds秒，随机生成一个批次的整数集（共num个），每个整数在[low, high]区间，

并将每一批次的整数集发送给Slaver 实例self.slaver 以便求和.

2.2) 若Master 收到self.slaver 对某一批次的整数集的求和结果，则打印输出结果

Slaver 线程类在没有收到一个批次的整数集时，就选择睡眠1 秒钟，若收到

一个批次的整数集时，则求和，并将结果发送回Master。

Master 类和Slaver 类完成线程同步以便保证不能漏掉任何一批次的整数集及其求和结果。

```
class Piple {
    private LinkedList<ArrayList<Integer>> inputs = new LinkedList<>();
    private LinkedList<Integer> results = new LinkedList<>();
    synchronized public ArrayList<Integer> pollData() {return inputs.pollFirst();}
    synchronized public void addData(ArrayList<Integer> data) {inputs.add(data);}
    synchronized public Integer pollResult() {return results.pollFirst();}
    synchronized public void addResult(Integer r) {results.add(r);}
}
class Slaver extends Thread{
    private Piple pip;
    public Slaver(Piple pip) {this.pip = pip;}
    public void run() {
        while(true) {
            ArrayList<Integer> data = pip.pollData();
            if (data == null) {
                try { Thread.sleep( millis: 1000); }
                catch(Exception e) {System.out.println(e); }
                continue;
            }
            int sum = 0;
            for(Integer e: data) { sum += e; }
            pip.addResult(sum);
        }
    }
```

```
public class Master {
    private Slaver s;
    private Piple pip;
    private Random r = new Random();
    public Master() {
        pip = new Piple();
        s = new Slaver(pip);
    }
    private ArrayList<Integer> getBatch(int low, int high, int num) {
        ArrayList<Integer> res = new ArrayList<>();
        float gap;
        for(int i=0; i<num; ++i) {
            gap = r.nextFloat();
            gap = gap * (high - low); //1 points
            int v = (int)gap + low;
            if (v > high) v = high;
            res.add(v);
        }
        return res;
    }
    public void loop(int low, int high, int num, int seconds) {
        s.start(); //1 points
        while(true) {
            try { Thread.sleep(seconds); }
            catch(Exception e) {System.out.println(e); }
            ArrayList<Integer> batch = getBatch(low, high, num);
            pip.addData(batch); //1 points
            for(Integer d: batch) { System.out.print(d + ", "); }
            System.out.println();
            Integer res = pip.pollResult(); //1 points
            if (res == null) continue;
            System.out.println("res: " + res);
        }
    }
```

```
public static void main(String[] args) {
    Master m = new Master();
    m.loop( low: 1,  high: 10,  num: 5,  seconds: 5);
}
```

```
public class RandomEventGenerator {
    private Listener listener;
    public RandomEventGenerator(Listener listener) {
        this.listener = listener; //1 points
    }
    public void generate(int stop) {
        Random r = new Random();
        int x;
        do {
            x = r.nextInt() % 100;
            if (x < 0) x = -x;
            if (x % 2 == 1)
                listener.onEvent(x); //2 points
        }
        while(x != stop);
    }
    public String toString() { return listener.toString(); }
    public static void main(String[] args) {
        RandomEventGenerator g = new RandomEventGenerator(new EventList());
        g.generate( stop: 11);
        System.out.println(g); //2 points
    }
}
```

```
public class EquipmentCollection {
    private ArrayList<Equipment> arrs = new ArrayList<>();
    public void add(Equipment e) { arrs.add(e); }
    public List<Equipment> sortByTotalPrice() {
        arrs.sort(new Comparator<Equipment>() {
            public int compare(Equipment a, Equipment b) {
                float sp = a.num * a.price;
                float sp2 = b.num * b.price;
                if (sp > sp2) return 1;
                if (sp < sp2) return -1;
                return 0;
            }
        });
        return arrs;
    }
    public String toString() {
        StringBuffer b = new StringBuffer();
        for(Equipment e: arrs) { b.append(e.toString() + "\n"); }
        return b.toString();
    }
    public static void main(String[] args) {
        EquipmentCollection c = new EquipmentCollection();
        Random r = new Random();
        int num; float price;
        for(int i=0; i<10; ++i) {
            num = r.nextInt() % 10;
            if (num < 0) num = -num;
            price = r.nextFloat() * 100;
            c.add(new Equipment( name: "" +i, num , price));
        }
        System.out.println(c);
        List<Equipment> lst = c.sortByTotalPrice();
        for(Equipment e: lst) { System.out.println(e); }
        System.out.println();
    }
}
```