《Java程序设计教程》

图书基本信息

书名：《Java程序设计教程》

13位ISBN编号：9787302338949

出版时间：2014-3-1

作者：雍俊海

页数：630

《Java程序设计教程》

内容概要

本书讲解Java程序设计知识及其编程方法，包括Java语言的基础语法、结构化程序设计、面向对象程序设计、数组、字符串、向量、哈希表、泛型、枚举、异常处理、文件和数据流、图形用户界面设计、小应用程序、线程、编程规范、网络程序设计、多媒体和图形学程序设计以及数据库程序设计等。
本书的章节编排与内容以人们学习与认知过程为基础，与公司的实际需求相匹配。内容力求简明，每章都附有习题，而且在附录中包含了图、表、例程以及类和接口的页码索引，在正文中采用特殊字体突出中心词，希望读者在轻松和欢乐之中迅速地了解并掌握Java程序设计的知识和方法，能应用到实践中去。
本书内容丰富，结构合理，语言简练，而且提供了丰富的例程，既可以作为计算机专业和非计算机专业的基础教材以及Sun公司的SCJP（Java程序员认证）考试的辅导教材，也可以作为需要使用Java语言的工程人员和科技工作者的自学参考书。

《Java程序设计教程》

作者简介

1991年到2000年就读于清华大学计算机科学与技术系，获学士、硕士和博士学位，被评为清华大学优秀博士毕业生，博士论文被评为全国优秀博士论文。从2000年3月到2000年6月在香港科技大学计算机系担任访问研究员；从2000年11月到2002年9月在美国肯塔基大学计算机系做博士后；现为清华大学软件学院教师。
从2003年起在清华大学开设了《Java程序设计》、《计算机图形学》和《计算机动画的算法与技术》等课程，编写了教材《Java程序设计》，很受学生欢迎。根据清华大学2003－2004学年秋季学期教学评估结果，教学成绩在全校名列前茅，获得学生一致好评，在清华大学软件学院所有参评教师中取得教学成绩第一名。获得了清华大学软件学院教学工作优秀奖，大学生研究训练计划(SRT)优秀指导教师一等奖和清华之友——优秀教师奖。

《Java程序设计教程》

书籍目录

《Java程序设计教程》

章节试读

1、《Java程序设计教程》的笔记-第120页

Java 数组：http://justcoding.iteye.com/blog/1131078数组类型是引用数据类型，数组对象包含一系列具有相同类型的数据元素，还含有成员域length。数组元素的下标是从0开始的，即第一个元素的下标是0。当数组元素的数据类型是引用数据类型时，数组元素(数组变量名[数组元素下标])的值是引用。在没赋值之前其默认值为null。数组元素的类型可以是Java语言允许的任何数据类型。当数组元素的类型是数组类型时，就构成了多维数组。定义时方括号的个数即为数组的维数。

2、《Java程序设计教程》的笔记-第440页

URL书写格式：
Protocol://Host:Port/File#Reference
File:广义的文件，普通文件和路径；
Reference:指向文件内部的某一指针；
对应类：java.net.URL
通过构造函数创建实例对象：
public URL(String spec) throws MalformedURLException
例程：J_URL.javaimport java.net.MalformedURLException;
import java.net.URL;

```
public class J_URL {
    public static void main(String args[]){
        try{
            String string = "http://source.android.com/source/building-running.html#choose-a-target";
            URL mURL = new URL(string);
            System.out.println("In URL \"" + mURL + "\":");
            System.out.println("协议:  " + mURL.getProtocol());
            System.out.println("主机名: " + mURL.getHost());
            System.out.println("端口号: " + mURL.getPort());
            System.out.println("文件名: " + mURL.getFile());
            System.out.println("引用:  " + mURL.getRef());
        }catch(MalformedURLException e){
            System.out.println("发生异常: " + e);
            e.printStackTrace();
        }
    }
}
```

URL运行结果

3、《Java程序设计教程》的笔记-第157页

类java.util.WeakHashMap 会自动删除“ 不常用 ”的元素。例程：结果与教材结果不相差很大
J_WeakHashMap.java

```
import java.util.WeakHashMap;
public class J_WeakHashMap{
    public static void main(String args[]) throws Exception{
        int size = 800;
```

```
        WeakHashMap&lt;String, String&gt; mWeakHashMap =
            new WeakHashMap&lt;String, String&gt;(size*3/4, 0.75f);

        for(int i=0; i&lt;size; i++){
            mWeakHashMap.put(("key" + i),("value" + i));
        }
        System.out.println("After inserting values, the size of WeakHashMap is:" + mWeakHashMap.size());

        for(int i=0; i&lt;size; i++){
            if(mWeakHashMap.containsKey("key" + i)){
                System.out.print("key" + i + ", ");
            }
        }
        System.out.println("");
        System.out.println("After some time, the size of mWeakHashMap is: " + mWeakHashMap.size());

        for(int i=0; i&lt;size; ){
            if(!mWeakHashMap.containsKey("key" + i)){//输出已经不存在WeakHashMap中的元素
                System.out.print("key" + i + ", ");
            }
            if(mWeakHashMap.size()!=size) //等待弱哈希表删除元素
                i++;
        }

        System.out.println("");
        System.out.println("After some time, the size of mWeakHashMap is: " + mWeakHashMap.size());
    }
}三段输出结果：初始WeakHashMap的size为800依次输出800个hashtable中value依次全部自动删除从序列0开始到799结束的元素，最终弱哈希表中没有任何元素剩下，与教程例程结果
简化输出结果：J_WeakHashMap.java
import java.util.WeakHashMap;
public class J_WeakHashMap{
    public static void main(String args[]) throws Exception{
        int size = 800;
        int i = 0;
        WeakHashMap&lt;String, String&gt; mWeakHashMap =
            new WeakHashMap&lt;String, String&gt;(size*3/4, 0.75f);

        for(i=0; i&lt;size; i++){
            mWeakHashMap.put(("key" + i),("value" + i));
        }
        System.out.println("After inserting values, the size of WeakHashMap is:" + mWeakHashMap.size());

        for(i=0; i&lt;size; ){
            if(!mWeakHashMap.containsKey("key" + i)){
    //            System.out.print("key" + i + ", ");
            }
            if(mWeakHashMap.size()!=size) //等待弱哈希表删除元素
                i++;
```

```
        }

        System.out.println("");
        System.out.println("After some time, the size of mWeakHashMap is: " + mWeakHashMap.size());
    }
}
```

似乎弱哈希表直接自动清空，而非删除“不常用”元素

4、《Java程序设计教程》的笔记-第395页

线程分为后台线程(daemon thread，又称为守护线程)和用户线程(user thread)，两者的区别是当在一个程序中只有后台线程在运行时，程序会立即退出；如果一个程序还存在正在运行的用户线程，则该程序不会终止。后台线程通常用来为其他线程提供服务，经常用于任务结束时的善后处理；后台线程的优先级要比其他线程的优先级低。

通过类java.lang.Thread的成员方法：

public final boolean isDaemon() //判断一个线程是否为后台线程

public final void setDaemon(boolean on) //进行用户线程和后台线程直接设置

setDaemon方法必须在调用start方法之前，一旦线程已经启动，则不能再调用setDaemon方法，否则会出现java.lang.IllegalThreadStateException类型异常。

例程：后台线程 J_ThreadDeamon.javapublic class J_ThreadDeamon extends Thread{

```
    public void run() {
        // TODO Auto-generated method stub
        for(int i=0; true; i++ ){
            System.out.println("运行线程 : " + i);
            try{
                Thread.sleep((int)(Math.random() * 1000));
            } catch(InterruptedException e){
                System.err.println("发生异常: " + e);
                e.printStackTrace();
            }
        }
    }

    public static void main(String args[]){
        J_ThreadDeamon t = new J_ThreadDeamon();
        t.setDaemon(true); //唯一区别
        t.start();
        if(t.isDaemon()){
            System.out.println("创建一个后台线程.");
        }else{
            System.out.println("创建一个用户线程.");
        }
        System.out.println("The end of main.");
    }

}
```

运行结果：

创建一个后台线程.

运行线程 : 0
The end of main.

例程：用户线程 J_ThreadUser.javapublic class J_ThreadUser extends Thread{

```
    public void run() {
        // TODO Auto-generated method stub
        for(int i=0; true; i++ ){
            System.out.println("运行线程 : " + i);
            try{
                Thread.sleep((int)(Math.random() * 1000));
            } catch(InterruptedException e){
                System.err.println("发生异常: " + e);
                e.printStackTrace();
            }
        }
    }

    public static void main(String args[]){
        J_ThreadUser t = new J_ThreadUser();
        t.start();
        if(t.isDaemon()){
            System.out.println("创建一个后台线程.");
        }else{
            System.out.println("创建一个用户线程.");
        }
        System.out.println("The end of main.");
    }

}
```

运行结果：虽然main成员方法结束，但是用户线程仍然继续运行
创建一个用户线程.
The end of main.
运行线程 : 0
运行线程 : 1
运行线程 : 2
运行线程 : 3
运行线程 : 4
运行线程 : 5
运行线程 : 6
运行线程 : 7
运行线程 : 8
运行线程 : 9
...

5、《Java程序设计教程》的笔记-第374页

参考链接：

http://www.360doc.com/content/06/0901/14/5874_196090.shtml
http://kelaocai.iteye.com/blog/227822
1.单行注释 //注释内容2.多行注释 /*
这是一个
多行注释
*/3.文档注释 /**
* This is first line.

***** This is second line.
This is third line.
*/javadoc注释内容可以通过javadoc命令自动生成HTML格式的API文档，保证程序代码和技术文档的同步，并且很多用于HTML格式化的HTML标记也可以用于文档注释中。
javadoc工具从Java代码中提取注释生成API文档时，主要从以下几项内容中提取信息：
1.包：包注释无法放入Java文件中，通过在包对应的目录中添加一个“ package.html ”文件，在生成HTML文档时，package.html文件中<BODY> </BODY>部分内容将会被提取出来作为包的说明；
2.public 类和接口：类注释用于说明整个类的功能、特性，放在import之后，class定义之前；
3.public 和 protected方法：方法注释用于说明方法的定义，如参数、返回值、方法的作用，位于所描述方法定义之前；
4.public 和 protected属性：默认情况下javadoc只对public和protected属性产生文档--通常为静态常量。
5.概要注释：对所有类文件提供概要说明的文件，同包注释一样，需要为这类注释单独建名为“ overview.html ”的HTML文件。

javadoc 标记由"@"及其后所跟的标记类型和专用注释引用组成，javadoc 标记有如下一些：
@author 标明开发该类模块的作者
@version 标明该类模块的版本
@see 参考转向，也就是相关主题
@param 对方法中某参数的说明
@return 对方法返回值的说明
@exception 对方法可能抛出的异常进行说明

@author 作者名
@version 版本号
其中，@author 可以多次使用，以指明多个作者，生成的文档中每个作者之间使用逗号 (,) 隔开。@version 也可以使用多次，只有第一次有效

使用 @param、@return 和 @exception 说明方法
这三个标记都是只用于方法的。@param 描述方法的参数，@return 描述方法的返回值，@exception 描述方法可能抛出的异常。它们的句法如下：
@param 参数名参数说明
@return 返回值说明
@exception 异常类名说明javadoc标签
Javadoc命令：javadoc [options] [packagenames] [sourcefiles] [@files]
packagenames:javadoc不递归作用于子包，不允许使用通配符，必须显示地列出希望建立文档的每一个包；
sourcefiles：类源代码文件、包描述文件、总体概述文件(-overview选项指定路径和名称)、其它杂文件(必须放在 doc-files 目录下，在Javadoc注释中使用HTML标签引用，如<img src="doc-files/XXX.gif">)；
files:把需要建立文档的文件名和包名放在一个或多个文本文件中，简化javadoc命令；

例程：Javadoc.java

```
/**
 * javadoc演示示例--&lt;b&gt;Javadoc&lt;/b&gt;
 * @author Nick
 *
 */
public class Javadoc {
    /**
     * 在main()方法中使用的显示字符串
     *
     * @see #main(java.lang.String[])
     */
    static String SDisplay;

    static String 变量;

    /**
     * 显示JavaDoc
     *
     * @param args
     * 从命令行输入字符串
     */
    public static void main(String args[]){
        SDisplay = "Hello World, ";
        变量 = "test";
        System.out.println(SDisplay + 变量);
    }
}
```

6、《Java程序设计教程》的笔记-第390页

多线程是Java语言的一个重要特性。Java虚拟机正是通过多线程机制来提高程序运行效率的。合理地进行多线程程序设计，编写多线程程序，可以更加充分地利用各种计算机资源，提高程序的执行效率。
编写线程程序主要是构造线程类。2种方法：构造java.lang.Thread 或者 java.lang.Runnable 的子类，java.lang.Thread也是实现了接口 java.lang.Runnable 的类，本质上都是构造 java.lang.Runnable 的子类。

java.lang.Thread类及其子类的每个实例对象即为Java程序的一个线程。构造 java.lang.Thread子类的目的是为了让线程类实例对象能够完成线程程序所需要的功能。

要点：
1.复写 java.lang.Thread中的run方法；
2.调用 java.lang.Thread成员方法start启动线程。如果直接调用成员方法run，则一般来说会立即执行成员方法run，从而失去线程的特性。在调用成员方法start之后，Java虚拟机会自动启动线程，从而由Java虚拟机进一步统一调度线程，实现各个线程一起并发地运行。Java虚拟机决定是否开始以及何时开始运行该线程，而线程的运行实际上就是执行线程的成员方法run。run()方法又被称为线程体，调用start()方法，线程进入Runnable状态(非Running状态，因此不会马上执行)，它将向线程调度器注册这个线程。直接调用run()方法，在可能存在线程等待的情况下容易造成死锁。

例程：通过构造 java.lang.Thread子类创建线程 Thread.java

```
public class J_Thread extends Thread{
    private int m_threadId;

    public J_Thread(int i){
        m_threadId = i;
        System.out.println("创建线程: " + m_threadId);
    }

    public void run(){
        for(int i=0; i&lt;3; i++){
            System.out.println("运行线程: " + m_threadId);
            try{
                Thread.sleep((int)(Math.random() * 1000));
            }catch(InterruptedException e){
                System.err.println("InterruptedException: " + e.toString());
                e.printStackTrace();
            }
        }
    }

    public static void main(String args[]){
        new J_Thread(1).start();
        new J_Thread(2).start();
        System.out.println("The end of mian.");
    }
}
```

运行结果，创建的线程并发运行，而且在成员方法main结束之后仍然可以继续运行。PS:
java.lang.Thread 的成员方法：
  public static void sleep(long millis) throws InterruptedException
  public void start()
  public void run()
java.lang.Math的成员方法：
  public static double random() //返回一个随机数x(0<= x < 1)

7、《Java程序设计教程》的笔记-第141页

StringBuffer的使用：http://www.360doc.com/content/06/1214/11/15458_293495.shtml

构造方法	length	capacity
public StringBuffer()	0	16
public StringBuffer(int capacity)	0	capacity
public StringBuffer(String str)	0	str.length + 16

当调用 ensureCapacity(int newCapacity),setLength(int newLength),append(...),insert(...)等函数时，如果操作之后的字符串缓冲区新长度超过操作之前旧的容量，则：
newCapacity = max(newLength, oldCapacity*2+2).

如下示例程序：J_StringBuffer.java

```
public class J_StringBuffer{
   public static void main(String args[]){
      StringBuffer sb = new StringBuffer("123456789");
      System.out.println("字符串缓冲区的字符序列为\"" + sb + "\"");
      System.out.println("Length of the StringBuffer is " + sb.length());
      System.out.println("Capacity of the StringBuffer is " + sb.capacity());
      System.out.println();

      sb.insert(5,"abcdefghijklmnopqrstuvwxyz");
      System.out.println("After calling insert(5,\"abcdefghijklmnopqrstuvwxyz\")");
      System.out.println("字符串缓冲区的字符序列为\"" + sb + "\"");
      System.out.println("Length of the StringBuffer is " + sb.length());
      System.out.println("Capacity of the StringBuffer is " + sb.capacity());
      System.out.println();

      sb.ensureCapacity(54);
      System.out.println("After calling ensureCapacity(30)");
      System.out.println("字符串缓冲区的字符序列为\"" + sb + "\"");
      System.out.println("Length of the StringBuffer is " + sb.length());
      System.out.println("Capacity of the StringBuffer is " + sb.capacity());

   }
}StringBuffer示例结果，注意Capacity的变化
```

8、《Java程序设计教程》的笔记-第113页

结构化程序设计主要特点是采用自顶向下、逐步求精的程序设计方法，使用3种基本控制结构构造程序；结构化设计的根本目标是把复杂的系统分解成简单模块的层次结构。因为其本质是功能分解，围绕实现处理功能的“过程”来构造系统，其稳定性、可重用性和可修改性比较差。
然而，用户需求的变化大多数是针对功能的，用户需求的变化往往造成系统结构的较大变化，这种变化对基于过程的程序设计来说是灾难性的。这种局限性促使了面向对象(OO)的思想获得广泛接受。OO在1970年代即出现，由于受到软硬件限制，直至90年代才成为主流程序设计思想。
面向对象的方法把系统看成完成某项任务的一组对象集合，对象是对系统消息作出响应的事物，因此面向对象方法中最值得关注的不是它应该做什么，而是它如何作出反应，即消息。

9、《Java程序设计教程》的笔记-第438页

在网络程序中，可以用类java.net.InetAddress的实例对象来记录网络地址，并获取一些相关信息。因为类java.net.InetAddress的构造方法的访问属性是默认模式，所以通常不能通过类java.net.InetAddress的构造方法来创建其实例对象。通过静态成员函数创建类java.net.InetAddress的实例对象：

```
public static InetAddress[] getAllByName(String host) throws UnknownHostException
public static InetAddress getByAddress(byte[] addr) throws UnknownHostException
public static InetAddress getByAddress(String host,byte[] addr) throws UnknownHostException
public static InetAddress getByName(String host) throws UnknownHostException
public static InetAddress getLocalHost() throws UnknownHostException
```

网络地址例程：J_InetAddress.java

```
import java.net.InetAddress;
import java.net.UnknownHostException;


public class J_InetAddress {
    public static void main(String args[]){
        String string = "www.baidu.com";
        InetAddress baidu = null;
        InetAddress[] baidus=null;

        try{
            baidu = InetAddress.getByName(string);
        } catch(UnknownHostException e){
            System.err.println("发生异常: " + e);
            e.printStackTrace();
        }//end of try
        if(baidu != null){
            System.out.println("Baidu网络地址是: " + baidu.getHostAddress());
            System.out.println("Baidu的主机名是: " + baidu.getHostName());
        } else{
            System.out.println("无法访问网络地址: " + string);
        }//end of if
        System.out.println();

        try{
            baidus = InetAddress.getAllByName(string);
        } catch(UnknownHostException e){
            System.err.println("发生异常: " + e);
            e.printStackTrace();
        }//end of try
        if(baidus.length &gt; 0){
            System.out.println("Baidu所有网络地址是: ");
            for(InetAddress addr:baidus){
                System.out.println(addr.getHostAddress());
            }
            System.out.println("Baidu的主机名是: " + baidu.getHostName());
        } else{
            System.out.println("无法访问网络地址: " + string);
        }//end of if
        System.out.println();

        InetAddress local = null;
        try{
            local = InetAddress.getLocalHost();
        }catch(UnknownHostException e){
            System.err.println("发生异常: " + e);
            e.printStackTrace();
        }//end of try
        if(local != null){
```

```
            System.out.println("本机网络地址是: " + local.getHostAddress());
            System.out.println("本机的主机名是: " + local.getHostName());
        }else{
            System.out.println("无法访问网络地址: " + string);
        }
    }//end of main
}//end of class
```

10、《Java程序设计教程》的笔记-第392页

实现Runnable接口的类必须使用Thread类的实例才能创建线程：
1.将实现Runnable接口的类实例化；
2.建立一个Thread类，并将第一步实例化的对象作为参数传入Thread类的构造方法。 通过接口java.lang.Runnable构造线程有时是非常有必要的。以为Java语言的语法规定每个类只能有一个直接父类，所以通过接口java.lang.Runnable构造线程是在构造线程过程中可能出现的多重继承问题的一种解决方法。

```
public class A extends B implements Runnable{
   //
   public void run(){
   //
   }
   //
}
```

如果“ extends B ”是必须的，则通过接口java.lang.Runnable构造线程的方法是非常有必要的。接口java.lang.Runnable只声明了唯一的成员方法: void run()

例程：通过接口Runnable构造线程 J_Runnable.java

```
public class J_Runnable implements Runnable{

    private int m_threadId;
    public J_Runnable(int i){
        m_threadId = i;
        System.out.println("创建线程: " + m_threadId);
    }

    public void run() {
        // TODO Auto-generated method stub
        for(int i=0; i&amp;lt;3; i++ ){
            System.out.println("运行线程 : " + m_threadId);
            try{
                Thread.sleep((int)(Math.random() * 1000));
            } catch(InterruptedException e){
                System.err.println("发生异常: " + e);
                e.printStackTrace();
            }
        }
    }

    public static void main(String args[]){
```

```
        Thread t1 = new Thread(new J_Runnable(1));
        Thread t2 = new Thread(new J_Runnable(2));

        t1.start();
        t2.start();

        System.out.println("The end of main.");
    }

}
```

运行结果：

```
   创建线程: 1
   创建线程: 2
   运行线程 : 1
   The end of main.
   运行线程 : 2
   运行线程 : 1
   运行线程 : 2
   运行线程 : 1
   运行线程 : 2
```

11、《Java程序设计教程》的笔记-第14页

```
       HelloWorld.javapublic class HelloWorld{
    public static void main(String args[]){
        System.out.println("Hello, Java World!");
        System.out.println("I will be the best Java programmer.");
    }
}HelloWorld 执行结果
```

程序说明：　　一个源程序文本文件可以包含多个类，但是每个文件最多只能包含一个公共类，而且这个公共类必须与其所在的文件同名。

　　成员方法main是所有Java应用程序执行的入口，但不是Java小应用程序的入口。因此，可以运行的Java应用程序必须含有main成员方法。

　　成员方法main必须同时含有public、static和void的属性。这是Java语言所规定的。

System.out.println 输出字符串之后自动换行

System.out.print　输出字符串之后不换行

编译命令和执行命令

>javac CLASSNAME.java

>java CLASSNAME

包管理：为了避免不同开发组织开发的类名的重复，建议使用组织域名的倒写来当做包名。

当将开发的类放入包中时，必须将类的源文件放入与包名的元素相一致的目录结构中，如package com.example.xjtu，必须将HelloWorld类文件放入到com\example\xjtu目录下；在Java类文件中使用“.”分隔包的层次结构。package com.example.xjtu;

```
public class HelloWorld{
    public static void main(String args[]){
        System.out.println("Hello, Java World!");
```

```
        System.out.println("I will be the best Java programmer.");
    }
}
```
编译执行取消package，执行出错，无法找到类文件

12、《Java程序设计教程》的笔记-第15页

import语句在package语句之后，类的定义之前；package语句只能有一个，import语句可以有多个。

单类型导入(single-type-import),例如import java.io.File;
按需类型导入(type-import-on-demand),例如import java.io.*;

import不会递归地执行引入动作，如import java.io.*，只会引入java.io包中的所有类，去不会引入java.io.sub_pkg_name(如果存在)中的类。

使用import并不会将相应的类或者包加载到class文件，也不会包含到Java源文件中，它的作用仅仅是对需要用到的类进行定位。

Java编译器默认为所有的Java程序引入JDK的java.lang包中的所有类。

JDK中常用的包：
1.java.lang--Java核心类，如String、Math、Integer、System和Thread等，提供常用功能，默认自动引入，这个包中的类可以在程序中直接使用；
2.java.net--执行与网络相关操作的类；
3.java.io--提供多种输入/输出功能的类；
4.java.util--一些实用工具类和数据结构类，如定义系统特性、使用与日历相关的函数、集合、堆栈等；
5.java.sql--用于访问数据库的类

13、《Java程序设计教程》的笔记-第2页

The Java Tutorial：http://docs.oracle.com/javase/tutorial/index.html

Java 语言的特点：
1.简单：去掉C/C++的指针、多重继承，增加垃圾回收机制，统一数据类型；
2.网络特性：针对互联网的语言；
3.面向对象：新型语言，无兼容过程式语言的负担；
4.平台无关/可移植性：Java虚拟机是平台相关的，运行在Java虚拟机上的Java Application是平台无关的；
5.鲁棒性：编译和执行过程中进行严格的检查；自动垃圾回收机制和异常处理机制；语言简单性；
6.安全：Java程序执行过程中，Java虚拟机对程序进行安全性检测；
7.多线程：处理复杂和并行事务，提高程序运行效率但是增加程序设计难度；
8.解释语言：现在也兼有编译语言的特点，可选择编译然后执行，提高效率；

建立基于J2SE的Java开发环境：1.下载J2SE安装程序；
2.运行，安装J2SE；
3.设置环境变量运行路径(path)和类路径(classpath);
4.下载J2SE的在线帮助文档
Java程序分为两种类型：
应用程序(Application):可以独立运行的计算机程序

小应用程序(Applet):Java语言开发的嵌在网页中运行的程序

14、《Java程序设计教程》的笔记-第125页

数组应用例程--求解和为15的棋盘游戏问题使用穷举法求解

```
///和为15的棋盘游戏问题：将从1到9的九个数不重复地填入3*3的棋盘中，使得各行、各列
//以及两条对角线上的3个数之和均为15

public class J_Grid15{
    int [][] m_board;
    long stateNumber;
    int resultNumber;

    J_Grid15(){
        m_board = new int[3][3];
        stateNumber = 0;
        resultNumber = 0;
    }//构造函数
    private  void mb_dataInit(){
        int i,j,k;
        for(i = 0, k = 1; i &lt; m_board.length; i++){
            for(j = 0; j&lt; m_board[i].length; j++,k++){
                m_board[i][j] = k;
            }
        }
    }//棋盘初始化
    //检测棋盘结束状态
    private boolean mb_dataEnd(){
        int i,j,k;
        for(i = 0, k = 9; i &lt; m_board.length; i++){
            for(j = 0; j &lt; m_board[i].length; j++, k--){
                if(m_board[i][j] != k)
                    return false;
            }
        }
        return true;
    }
    //转换棋盘到下一个状态？？？
    private void mb_dataNext(){
        int i,j,k;
        stateNumber ++;
        for(i = m_board.length - 1; i &gt;= 0; i--){
            for(j = m_board[i].length - 1; j &gt;=0; j--){
                if(m_board[i][j] == 9)
                    m_board[i][j] =1;
                else{
                    m_board[i][j]++;
                    return;
```

```
            }
        }
    }
}
//检测是否存在同样的数字
private boolean mb_dataCheckDifferent(){
    int i,j;
    int []digit = new int[10];
    for(i = 0; i &lt; m_board.length; i++){
        for(j = 0; j &lt; m_board[i].length; j++){
            digit[m_board[i][j]] = 1;
        }
    }
    for(i = 1, j = 0; i &lt; digit.length; i++){
        j += digit[i];
    }
    if(j == 9)
        return true;
    return false;
}
//检测各行和是否为15
private  boolean mb_dataCheckSumRow(){
    int i,j,sum;
    for(i = 0; i &lt; m_board.length; i++){
        for(j = 0, sum  = 0; j &lt; m_board[i].length; j++){
            sum += m_board[i][j];
        }
        if(sum != 15) return false;
    }
    return true;
}
//检测各列和是否为15
private boolean mb_dataCheckSumCol(){
    int i,j,sum;
    for(i = 0; i &lt; m_board.length; i++){
        for(j = 0, sum  = 0; j &lt; m_board[i].length; j++){
            sum += m_board[j][i];
        }
        if(sum != 15) return false;
    }
    return true;
}
//检测对角线之和是否为15
private boolean mb_dataCheckDiagonal(){
    int i, sum1, sum2;
    for(i = 0, sum1 = 0, sum2 = 0; i &lt; m_board.length; i++){
        sum1 += m_board[i][i];
        sum2 += m_board[i][m_board.length - 1 - i];
    }
```

```
    if(sum1 == 15 &amp;amp;&amp;amp; sum2 == 15) return true;
    return false;
}

//检测是否符合状态要求
private boolean mb_dataCheck(){
    if(!mb_dataCheckDifferent()) return false;
    if(!mb_dataCheckSumCol()) return false;
    if(!mb_dataCheckSumRow()) return false;
    if(!mb_dataCheckDiagonal()) return false;
    return true;
}
private void mb_arrange(){
    for(mb_dataInit(); !mb_dataEnd(); mb_dataNext()){
        if(mb_dataCheck()){
            resultNumber ++;
            System.out.println("第 " + resultNumber + " 种解:");
            mb_printGrid();
        }
    }

}
private void mb_printGrid(){
    int i;
    mb_printGridRowBoard();
    for(i=0; i&amp;lt;m_board.length; i++){
        mb_printGridRowBoard(i);
        mb_printGridRowBoard();
    }
}
private void mb_printGridRowBoard(){
    int i;
    System.out.print("+");
    for(i = 0; i &amp;lt; 5; i++){
        System.out.print("-");
    }
    System.out.println("+");
}
private void mb_printGridRowBoard(int i){
    int j;
    for(j = 0; j &amp;lt; m_board[i].length; j++){
        System.out.print("|" + m_board[i][j]);
    }
    System.out.println("|");
}
public static void main(String [] args){
    J_Grid15 obj = new J_Grid15();
    obj.mb_arrange();
    obj.printResult();
```

```
    }
    private void printResult(){
        long result = 0;
        result = 9*9*9*9*9*9*9*9*9;
        System.out.println("9^9 = " + result);
        System.out.println("The number of state: " + stateNumber);
        System.out.println("The result number: " + resultNumber);
    }
}J_Grid15运行结果
```

《Java程序设计教程》

版权说明