Java语言基础 Day05

1 插入排序算法

1.1 问题

插入排序。一个整型数组大小是12，初始化后，对其中的数字按从小打到进行排序，输出，可以使用插入排序的算法。系统交互情况如图-50所示：

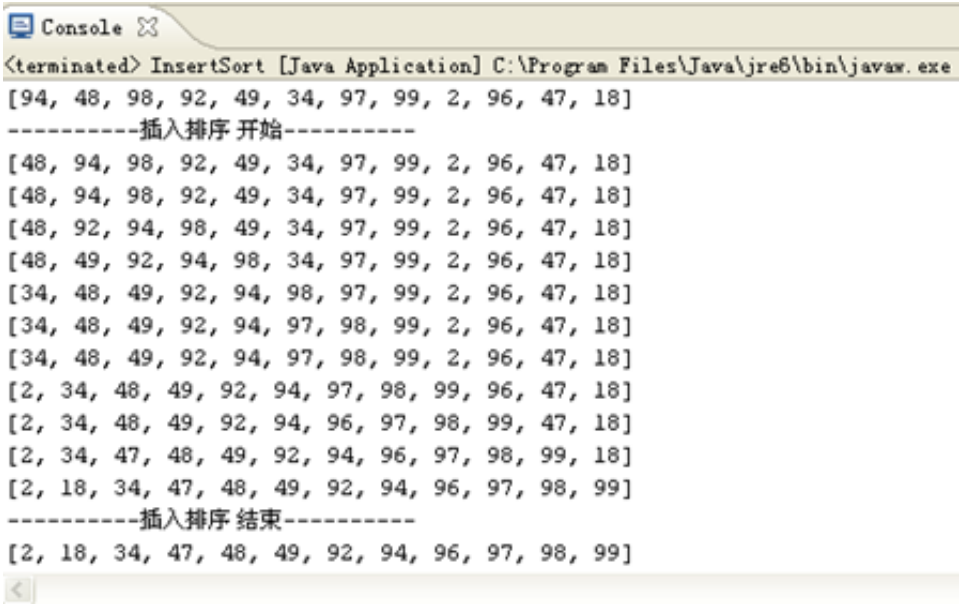

图-50

1.2 方案

将数组中每个元素与第一个元素比较，如果这个元素小于第一个元素，则交换这两个元素循环第1条规则，找出最小元素，放于第1个位置经过n-1轮比较完成排序。代码如下

```
01.     for (int i = 1; i < arr.length; i++) {
02.         int k = arr[i];// 取出待插入元素
03.         // 找到插入位置
04.         int j;
05.         for (j = i - 1; j >= 0 && k < arr[j]; j--) {
06.             arr[j + 1] = arr[j];// 移动元素
```

```
        }
                // 插入元素
        arr[j + 1] = k;
        System.out.println(Arrays.toString(arr));
    }
```

1.3 实现

系统代码实现如下：

```
import java.util.Arrays;
import org.apache.commons.lang.math.RandomUtils;
public class InsertSort {
    public static void main(String[] args) {
        int[] arr = new int[12];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = RandomUtils.nextInt(100);
        }
        // 插入排序
        System.out.println(Arrays.toString(arr));
        System.out.println("----------插入排序 开始----------");
        for (int i = 1; i < arr.length; i++) {
            int k = arr[i];// 取出待插入元素
            // 找到插入位置
            int j;
            for (j = i - 1; j >= 0 && k < arr[j]; j--) {
                arr[j + 1] = arr[j];// 移动元素
            }
            // 插入元素
            arr[j + 1] = k;
            System.out.println(Arrays.toString(arr));
        }
        System.out.println("----------插入排序 结束----------");
        System.out.println(Arrays.toString(arr));
    }
}
```

隐藏

1.4 扩展

熟练掌握插入排序的算法和代码实现。

2 冒泡排序算法

2.1 问题

冒泡排序。个整形数组大小是12，初始化后，对其中的数字按从小打到进行排序，输出，可以使用插入排序的算法。系统交互情况如图-51所示：

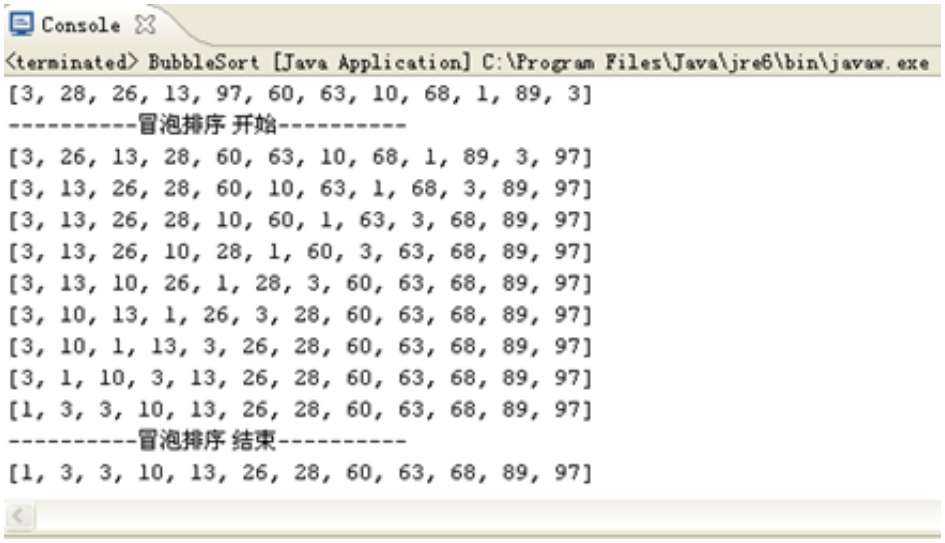

图-51

2.2 方案

比较相邻的元素，将小的放到前面。代码如下

```
01.     for (int i = 0; i < arr.length - 1; i++) {
02.         boolean isSwap = false;
03.         for (int j = 0; j < arr.length - i - 1; j++) {
04.             if (arr[j] > arr[j + 1]) {
05.                 int t = arr[j];
06.                 arr[j] = arr[j + 1];
07.                 arr[j + 1] = t;
08.                 isSwap = true;
09.             }
10.         }
11.         if (!isSwap)
12.             break;
13.         System.out.println(Arrays.toString(arr));
14.     }
```

2.3 实现

系统代码实现如下：

```
01.  import java.util.Arrays;
02.  import org.apache.commons.lang.math.RandomUtils;
03.  public class BubbleSort {
04.      public static void main(String[] args) {
05.
06.          int[] arr = new int[12];
07.          for (int i = 0; i < arr.length; i++) {
08.              arr[i] = RandomUtils.nextInt(100);
09.          }
10.          // 冒泡排序
11.          System.out.println(Arrays.toString(arr));
12.          System.out.println("----------冒泡排序 开始----------");
13.          for (int i = 0; i < arr.length - 1; i++) {
14.              boolean isSwap = false;
15.              for (int j = 0; j < arr.length - i - 1; j++) {
16.                  if (arr[j] > arr[j + 1]) {
17.                      int t = arr[j];
18.                      arr[j] = arr[j + 1];
19.                      arr[j + 1] = t;
20.                      isSwap = true;
21.                  }
22.              }
23.              if (!isSwap)
24.                  break;
25.              System.out.println(Arrays.toString(arr));
26.          }
27.          System.out.println("----------冒泡排序 结束----------");
28.          System.out.println(Arrays.toString(arr));
29.      }
30.  }
```

隐藏

2.4 扩展

冒泡排序法可以使用大气泡沉底的方式，也可以使用轻气泡上浮的方式实现。请使用轻气泡上浮的方式实现冒泡排序算法，系统交互过程如图 - 52所示。

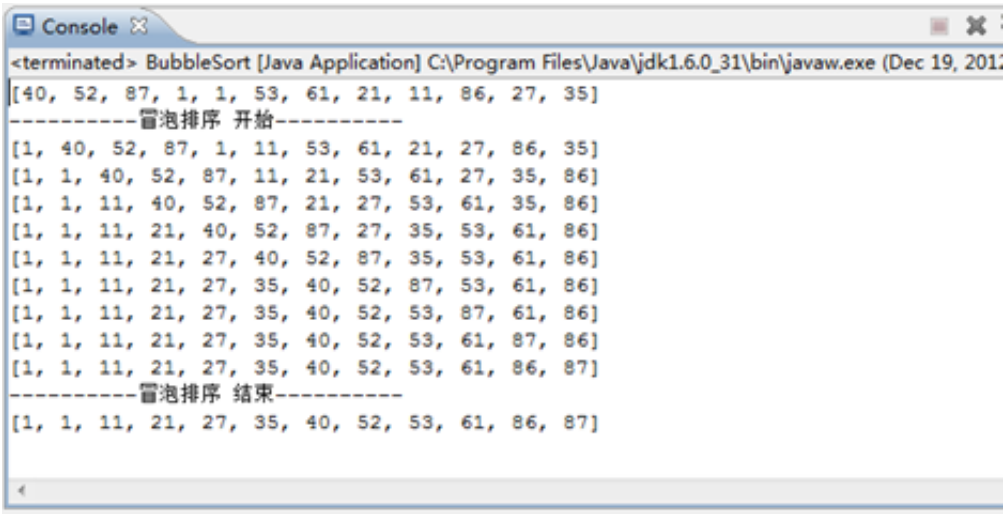

图-52

系统代码实现如下：

```
01.  import java.util.Arrays;
02.  import org.apache.commons.lang.math.RandomUtils;
03.  public class BubbleSort {
04.      public static void main(String[] args) {
05.
06.          int[] arr = new int[12];
07.          for (int i = 0; i < arr.length; i++) {
08.              arr[i] = RandomUtils.nextInt(100);
09.          }
10.
11.          // 冒泡排序
12.          System.out.println(Arrays.toString(arr));
13.          System.out.println("----------冒泡排序 开始----------");
14.          for (int i = 0; i < arr.length - 1; i++) {
15.              boolean isSwap = false;
16.              for (int j = arr.length - 1; j > i; j--) {
17.                  if (arr[j] < arr[j - 1]) {
18.                      int t = arr[j];
19.                      arr[j] = arr[j - 1];
20.                      arr[j - 1] = t;
21.                      isSwap = true;
22.                  }
23.              }
24.              if (!isSwap)
25.                  break;
```

```
26.                 System.out.println(Arrays.toString(arr));
27.             }
28.             System.out.println("----------冒泡排序 结束----------");
29.             System.out.println(Arrays.toString(arr));
30.         }
31.     }
```

隐藏

3 二分法查找(选做)

3.1 问题

二分查找算法。系统随机生成12个数，形成一个数组，并且对该数组排序，把排序后的数组打印到控制台，并且提示用户输入要查找的数字。系统使用二分查找算法，查找出用户要查找的数字在排序后的数组中的索引位置。系统交互情况如图-53所示：

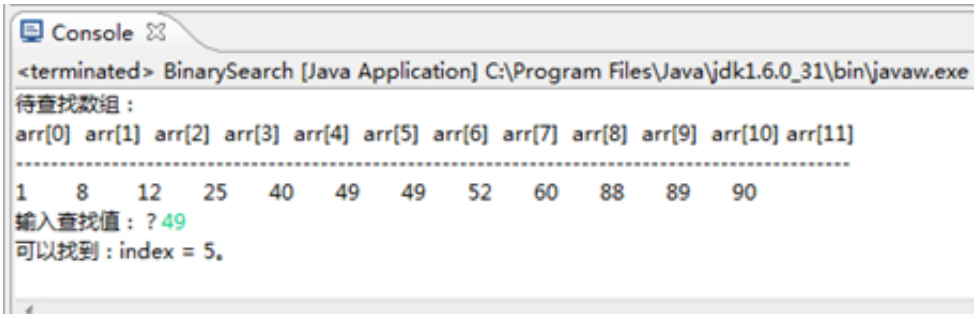

图-53

3.2 方案

系统使用commons包中的RandomUtils.nextInt()方法实现随机生成100以内的随机数，代码如下：

```
01.     int[] arr = new int[12];
02.     for (int i = 0; i < arr.length; i++) {
03.         arr[i] = RandomUtils.nextInt(100);
04.     }
```

系统使用Arrays.sort()方法实现对数组从小到大的排序，代码如下：

```
01.     Arrays.sort(arr);
```

二分法思想是取中，比较 ：

（1）求有序序列arr的中间位置mid

（2）k为要查找的数字，

若arr[mid] ==k，查找成功；

若arr[mid] >k，在前半段中继续进行二 分查找；

若arr[mid] <k，则在后半段中继续进行二分查找。

假如有一组数为3，12，24，36，55，68，75，88要查给定的值k=24.可设三个变量low，mid，high分别指向数据的上界，中间和下界，mid=（low+high）/2.

1.开始令low=0（指向3），high=7（指向88），则mid=3（指向36）。因为k<mid，故应在前半段中查找。

2.令新的high=mid-1=2（指向24），而low=0（指向3）不变，则新的mid=1（指向12）。此时k>mid，故确定应在后半段中查找。

3.令新的low=mid+1=2（指向24），而high=2（指向24）不变，则新的mid=2，此时k=arr[mid]，查找成功。

如果要查找的数不是数列中的数，例如k=25，当第四次判断时，k>mid[2]，在后边半段查找，令low=mid+1，即low=3(指向36)，high=2（指向24）出现low>high的情况，表示查找不成功。

二分查找算法代码如下：

```
    int low = 0;
    int high = arr.length - 1;
    int mid = -1;
    while (low <= high) {
        mid = (low + high) / 2;
        if (arr[mid] < value)
            low = mid + 1;
        else if (arr[mid] > value)
            high = mid - 1;
        else
            break;
        }
        if (low <= high) {
            System.out.println("可以找到: index = " + mid + "。");
        } else {
            System.out.println("无法找到！");
    }
```

3.3 实现

系统代码实现如下：

```java
import java.util.Arrays;
import java.util.Scanner;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.math.RandomUtils;
public class BinarySearch {
    public static void main(String[] args) {
        int[] arr = new int[12];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = RandomUtils.nextInt(100);
        }
        Arrays.sort(arr);
        Scanner scanner = new Scanner(System.in);
        System.out.println("待查找数组：");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(StringUtils.rightPad(
"arr[" + i + "]", 8, " "));
        }
        System.out.println(
"\n"+StringUtils.repeat("-", arr.length * 8));
        for (int i = 0; i < arr.length; i++) {
            System.out.print(StringUtils.rightPad("" + arr[i], 8, " "));
        }
        System.out.println();
        System.out.print("输入查找值：？");
        int value = scanner.nextInt();
        scanner.close();
        // 折半查找
        int low = 0;
        int high = arr.length - 1;
        int mid = -1;
        while (low <= high) {
            mid = (low + high) / 2;
            if (arr[mid] < value)
                low = mid + 1;
            else if (arr[mid] > value)
                high = mid - 1;
            else
                break;
        }
```

```
40.             if (low <= high) {
41.                 System.out.println("可以找到: index = " + mid + "。");
42.             } else {
43.                 System.out.println("无法找到! ");
44.             }
45.         }
46.     }
```

隐藏

3.4 扩展

熟练掌握二分法查找的算法和代码实现。

4 递归方式实现阶乘

4.1 问题

使用递归的方式计算某正整数的阶乘，并在控制台输出结果。系统交互情况如图-54所示：

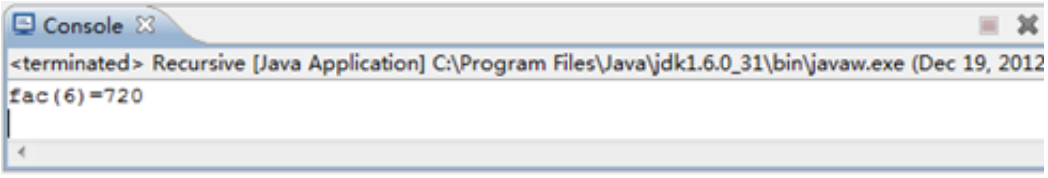

图-54

注：正整数阶乘指从1乘以2乘以3乘以4一直乘到所要求的数。 例如4的阶乘公式是1×2×3×4，得到的积是24，则24就是4的阶乘。

4.2 方案

首先找出求阶乘的规律：每个数字的阶乘都等于其本身乘以前一个数的阶乘，即：

F(n) = n * F(n-1)

然后找出递归结束的条件：1的阶乘为1，比1小的整数则不用计算阶乘。

因此，可以定义方法来使用递归算法计算某正整数的阶乘。代码如下：

```
01.     public static long fac(int n) {
02.         if (n == 1)
03.             return 1;
04.         return n * fac(n - 1);
05.     }
```

注意：因为阶乘的数值可能较大，因此需要定义 long 类型作为返回值的类型。

4.3 实现

系统代码实现如下：

```
01.     public class Recursive {
02.         public static void main(String[] args) {
03.             int n = 6;
04.             long r1 = fac(n);
05.             System.out.println("fac(" + n + ")=" + r1);
06.         }
07.
08.         public static long fac(int n) {
09.             if (n == 1)
10.                 return 1;
11.             return n * fac(n - 1);
12.         }
13.     }
```

隐藏

4.4 扩展

熟练掌握递归方式实现阶乘的思路和代码实现。

5 递归实现费氏序列（反例）

5.1 问题

使用递归的方式计算费氏序列中某项的数值，并在控制台输出结果。系统交互情况如图-55所示：

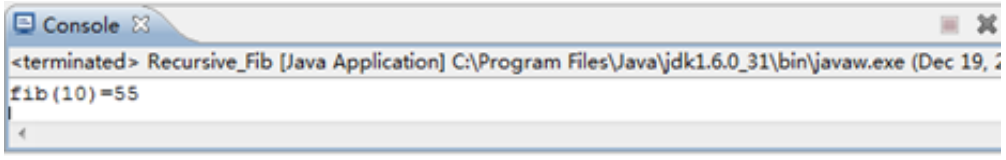

图-55

注：费氏数列指的是这样一个数列：1、1、2、3、5、8、13、21、34…… 。这个数列的第一项和第二项均为1，从第三项开始，每一项都等于前两项之和。

5.2 方案

在数学上，费氏数列以如下递归的方法定义：

F0=0，F1=1，Fn=F(n-1)+F(n-2)（n>=2，n∈N*）

因此，可以定义方法来使用递归算法计算某费氏序列。代码如下：

```
01.         public static long fib(int n) {
```

```
02.             if (n == 1 || n == 2)
03.                 return 1;
04.             return fib(n - 1) + fib(n - 2);
05.         }
```

注意：因为所计算的数值可能较大，因此需要定义 long 类型作为返回值的类型。

5.3 实现

系统代码实现如下：

```
01.     public class Recursive {
02.             public static void main(String[] args) {
03.             int n = 10;
04.             System.out.println("fib(" + n + ")=" + fib(n));
05.         }
06.
07.         public static long fib(int n) {
08.             if (n == 1 || n == 2)
09.                 return 1;
10.             return fib(n - 1) + fib(n - 2);
11.         }
12.     }
```

隐藏

5.4 扩展

递归算法的效率通常比较差，尤其是在递归次数较多的情况下。

比如，计算费氏序列中第50项的数值，可以使用循环计算，也可以使用递归算法进行计算，二者有着很大的性能差异。

查看如下代码（注意性能和时间的比较）：

```
01.     public class Recursive {
02.
03.         public static void main(String[] args) {
04.             int n = 50;
05.             long time = System.currentTimeMillis();
06.             System.out.println("fibByFor(" + n + ")=" + fibByFor(n));
07.             System.out.println("使用for循环计算，用时："
```

```
08.                         + (System.currentTimeMillis() - time) + "毫秒");
09.
10.             time = System.currentTimeMillis();
11.             System.out.println("fibByRecursive(" + n + ")="
12.                         + fibByRecursive(n));
13.             System.out.println("使用递归计算，用时："
14.                         + (System.currentTimeMillis() - time) + "毫秒");
15.         }
16.         //循环方式计算
17.         public static long fibByFor(int n) {
18.             if (n == 1 || n == 2)
19.                 return 1;
20.             long f1 = 1;
21.             long f2 = 1;
22.             for (int i = 3; i <= n; i++) {
23.                 f2 = f1 + f2;
24.                 f1 = f2 - f1;
25.             }
26.             return f2;
27.         }
28.         //递归方式计算
29.         public static long fibByRecursive(int n) {
30.             if (n == 1 || n == 2)
31.                 return 1;
32.             return fibByRecursive(n - 1) + fibByRecursive(n - 2);
33.         }
34.     }
```

隐藏

程序运行的结果如图 - 56所示。

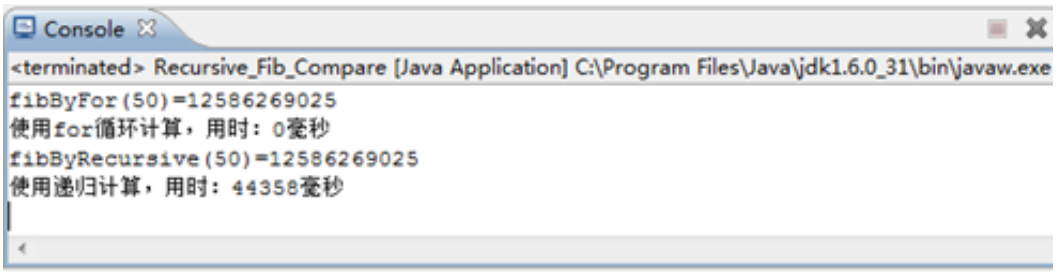

图-56

6 基本类型解析

6.1 问题

基本数据类型的解析。例如：将一个int类型的随机数存入tmp文件，从文件中，按字节读取出该随机数后进行解析，解析后形成32位的整数输出到控制台，最后将tmp文件删除，控制台输出情况如图-24所示：

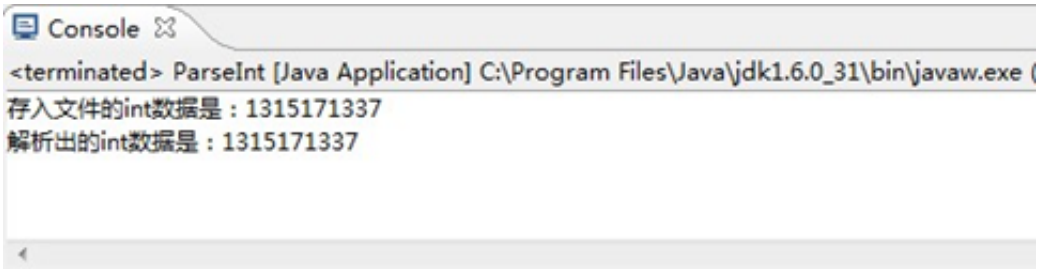

图-24

6.2 方案

系统使用before方法，实现将一个int类型的随机数存入tmp文件，代码如下所示：

```
01.     public static void before() throws Exception {
02.         DataOutputStream dos =
03.     new DataOutputStream(new FileOutputStream("tmp"));
04.         Random ran = new Random();
05.         int value = ran.nextInt(Integer.MAX_VALUE);
06.         System.out.println("存入文件的int数据是：" + value);
07.         dos.writeInt(value);
08.         dos.close();
09.     }
```

系统使用RandomAccessFile类的readByte()方法逐个字节的读取tmp文件中的整数，readByte()方法读取一个有符号的八位值，代码如下所示：

```
01.     RandomAccessFile raf = new RandomAccessFile("tmp", "r");
02.     byte b1 = raf.readByte();
03.     byte b2 = raf.readByte();
04.     byte b3 = raf.readByte();
05.     byte b4 = raf.readByte();
06.     raf.close();
```

系统使用0xff清理int数据: 将高24位清理为0, 低8位是从文件中读取的byte数据，代码如下所示：

```
int d1 = b1 & 0xff;
int d2 = b2 & 0xff;
int d3 = b3 & 0xff;
int d4 = b4 & 0xff;
```

向左移位int数据: d1移动24位, d2移动16位,d3移动8位, d4移动0位, 将4个整数相加使四个整数数据拼接为一个整数，代码如下所示：

```
int value = (d1 << 24) + (d2 << 16) + (d3 << 8) + d4;
```

系统使用after方法，将tmp文件删除，代码如下所示：

```
public static void after() throws Exception {
    File file = new File("tmp");
    if (file.exists() && file.isFile()) {
        file.delete();
    }
}
```

6.3 实现

系统代码实现如下：

```
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.RandomAccessFile;
import java.util.Random;
public class ParseInt {
    public static void main(String[] args) throws Exception {
        before();
        RandomAccessFile raf = new RandomAccessFile("tmp", "r");
        byte b1 = raf.readByte();
        byte b2 = raf.readByte();
        byte b3 = raf.readByte();
```

```
            byte b4 = raf.readByte();
            raf.close();
            // ----------------------------------------
            int d1 = b1 & 0xff;
            int d2 = b2 & 0xff;
            int d3 = b3 & 0xff;
            int d4 = b4 & 0xff;
            int value = (d1 << 24) + (d2 << 16) + (d3 << 8) + d4;
            System.out.println("解析出的int数据是：" + value);
            // ----------------------------------------
            after();
        }
        public static void before() throws Exception {
            DataOutputStream dos =
    new DataOutputStream(new FileOutputStream("tmp"));
            Random ran = new Random();
            int value = ran.nextInt(Integer.MAX_VALUE);
            System.out.println("存入文件的int数据是：" + value);
            dos.writeInt(value);
            dos.close();
        }
        public static void after() throws Exception {
            File file = new File("tmp");
            if (file.exists() && file.isFile()) {
                file.delete();
            }
        }
    }
```

隐藏

6.4 扩展

实现 short 数据类型的解析。例如：将一个short类型的随机数存入tmp文件，从文件中，按字节读取出该随机数后进行解析，解析后形成16位的整数输出到控制台，最后将tmp文件删除，控制台输出情况如图-25所示：

/

图-25

系统实现的代码如下：

```java
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.RandomAccessFile;
import java.util.Random;
public class ParseShort {
    public static void main(String[] args) throws Exception {
        before();
        RandomAccessFile raf = new RandomAccessFile("tmp", "r");
        byte b1 = raf.readByte();
        byte b2 = raf.readByte();
        raf.close();
        // ----------------------------------------
        int d1 = b1;
        int d2 = b2;
        d1 &= 0xff;
        d2 &= 0xff;
        d1 <<= 8;
        short value = (short) (d1 + d2);
        System.out.println("解析出的short数据是：" + value);
        // ----------------------------------------
        after();
    }
    public static void before() throws Exception {
        DataOutputStream dos =
new DataOutputStream(new FileOutputStream("tmp"));
        Random ran = new Random();
        int value = ran.nextInt(Short.MAX_VALUE);
        System.out.println("存入文件的short数据是：" + value);
        dos.writeShort(value);
        dos.close();
    }
    public static void after() throws Exception {
        File file = new File("tmp");
        if (file.exists() && file.isFile()) {
            file.delete();
        }
    }
```

```
39.     }
```

隐藏

7 彩票双色球生成器

7.1 问题

系统作为彩票双色球生成器，模拟机选一注双色球的彩票号码。需要从“01”到“32”中随机选择出6个数字作为红色球且这6个数字不能重复，并从”01”到”07”中随机选择一个数字作为蓝色球；7个数字合到一起作为一注双色球彩票的号码。系统交互情况如图-44所示：

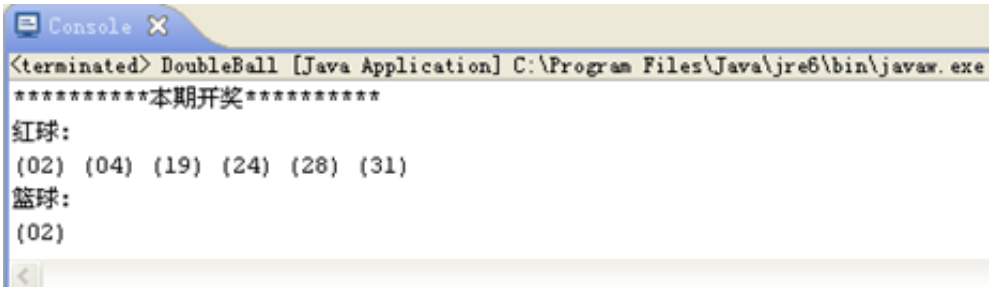

图-44

7.2 方案

系统使用一个字符串数组存放所有红色球，通过另外一个字符串数组存放所有蓝色球，并定义一个与红色球数组大小一样的boolean类型的数组，标识红色球是否已经被选中，并定义一个数组放被选中的红色球，代码如下：

```
01.     String[] RED_BALLS = { "01", "02", "03", "04", "05", "06", "07", "08",
02.                     "09", "10", "11", "12", "13", "14", "15", "16", "17",
03.      "18","19", "20", "21", "22", "23", "24", "25", "26", "27",
04.      "28","29", "30", "31", "32" };
05.     String[] BLUE_BALLS = { "01", "02", "03", "04", "05", "06", "07" };
06.     boolean[] redFlags = new boolean[RED_BALLS.length];
07.     String[] redBalls = new String[6];
```

通过循环随机选择红色球，选中的把标识设置为true,此球不能再被选，一直到6个全部选中为止：代码如下

```
01.     for (int i = 0; i < redBalls.length; i++) {
02.         int index;
03.         do {
04.             index = ran.nextInt(RED_BALLS.length);
05.         } while (redFlags[index]);
06.         redBalls[i] = RED_BALLS[index];
```

```
            redFlags[i] = true;
        }
```

7.3 实现

系统代码实现如下：

```
import java.util.Arrays;
import java.util.Random;
public class DoubleBall {
public static void main(String[] args) {
String[] RED_BALLS = { "01", "02", "03", "04", "05", "06", "07", "08",
            "09", "10", "11", "12", "13", "14", "15", "16", "17", "18",
            "19", "20", "21", "22", "23", "24", "25", "26", "27", "28",
            "29", "30", "31", "32" };
String[] BLUE_BALLS = { "01", "02", "03", "04", "05", "06", "07" };
boolean[] redFlags = new boolean[RED_BALLS.length];
String[] redBalls = new String[6];
String blueBall;
    Random ran = new Random();
        // red
    for (int i = 0; i < redBalls.length; i++) {
            int index;
            do {
                index = ran.nextInt(RED_BALLS.length);
            } while (redFlags[index]);
            redBalls[i] = RED_BALLS[index];
            redFlags[index] = true;
        }
        // blue
        blueBall = BLUE_BALLS[ran.nextInt(BLUE_BALLS.length)];
        Arrays.sort(redBalls);
        System.out.println("**********本期开奖**********");
        System.out.println("红球: ");
        for (int i = 0; i < redBalls.length; i++) {
            System.out.print("(" + redBalls[i] + ") ");
        }
        System.out.println();
```

```
32.             System.out.println("篮球: ");
33.             System.out.print("(" + blueBall + ") ");
34.         }
35.     }
```

隐藏

7.4 扩展（选做）

生成验证码。验证码要求如下：

1.6位长度的大写字母和数字的组合；

2.6个字符不能重复；

3.为避免阅读困难，不能包含以下字符：数字0、数字1、数字2、字母I、字母O、字母Z；

系统交互情况如图-45所示：

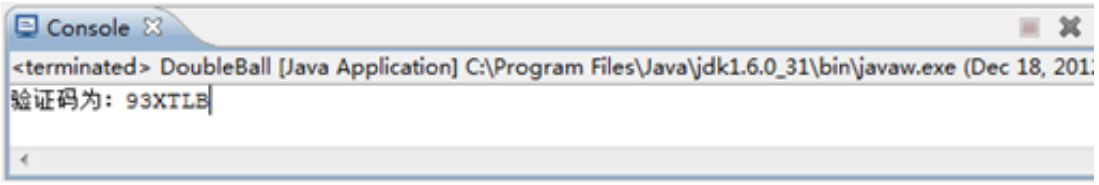

图-45

系统代码实现如下：

```
01.     import java.util.Arrays;
02.     import java.util.Random;
03.     public class VerificationCode {
04.     public static void main(String[] args) {
05.             String[] CHARS = { "3", "4", "5", "6", "7", "8", "9", "A", "B", "
06.                     "D", "E", "F", "G", "H", "J", "K", "L", "M", "N", "P", "Q"
07.                     "R", "S", "T", "U", "V", "W", "X", "Y" };
08.             boolean[] charFlags = new boolean[CHARS.length];
09.             String[] verifyCodes = new String[6];
10.
11.             Random ran = new Random();
12.             for (int i = 0; i < verifyCodes.length; i++) {
13.                 int index;
14.                 do {
15.                     index = ran.nextInt(CHARS.length);
16.                 } while (charFlags[index]);
17.                 verifyCodes[i] = CHARS[index];
18.                 charFlags[index] = true;
19.             }
```

Top

```
20.             System.out.print("验证码为：");
21.             for (int i = 0; i < verifyCodes.length; i++) {
22.                 System.out.print(verifyCodes[i]);
23.             }
24.     }
```

隐藏