1. 可变参数

在**JDK1.5**之后，如果我们定义一个方法需要接受多个参数，并且多个参数类型一致，我们可以对其简化.

格式：

```
修饰符 返回值类型 方法名(参数类型... 形参名){  }
```

底层：

其实就是一个数组

好处：

在传递数据的时候，省的我们自己创建数组并添加元素了，JDK底层帮我们自动创建数组并添加元素了

代码演示：

```
public class ChangeArgs {
  public static void main(String[] args) {
      int sum = getSum(6, 7, 2, 12, 2121);
      System.out.println(sum);
  }

  public static int getSum(int... arr) {
      int sum = 0;
       for (int a : arr) {
       sum += a;
      }
       return sum;
  }
}
```

注意：

1.一个方法只能有一个可变参数

2.如果方法中有多个参数，可变参数要放到最后。

应用场景：Collections

在Collections中也提供了添加一些元素方法：

`public static <T> boolean addAll(Collection<T> c, T... elements)`:往集合中添加一些元素。

代码演示：

```
public class CollectionsDemo {
    public static void main(String[] args) {
      ArrayList<Integer> list = new ArrayList<Integer>();
      //原来写法
      //list.add(12);
      //list.add(14);
      //list.add(15);
      //list.add(1000);
      //采用工具类 完成 往集合中添加元素
      Collections.addAll(list, 5, 222, 1, 2);
      System.out.println(list);
}
```

2. Collections类

2.1 Collections常用功能

- `java.utils.Collections`是集合工具类，用来对集合进行操作。

  常用方法如下：

- `public static void shuffle(List<?> list)`:打乱集合顺序。
- `public static <T> void sort(List<T> list)`:将集合中元素按照默认规则排序。
- `public static <T> void sort(List<T> list, Comparator<? super T> )`:将集合中元素按照指定规则排序。

Collections常用的API

方法名称	说明
public static <T> boolean addAll(Collection<T> c, T... elements)	批量添加元素
public static void shuffle(List<?> list)	打乱List集合元素的顺序
public static <T> void sort(List<T> list)	排序
public static <T> void sort(List<T> list, Comparator<T> c)	根据指定的规则进行排序
public static <T> int binarySearch (List<T> list, T key)	以二分查找法查找元素
public static <T> void copy(List<T> dest, List<T> src)	拷贝集合中的元素
public static <T> int fill (List<T> list, T obj)	使用指定的元素填充集合
public static <T> void max/min(Collection<T> coll)	根据默认的自然排序获取最大/小值
public static <T> void swap(List<?> list, int i, int j)	交换集合中指定位置的元素

代码演示：

```
public class CollectionsDemo {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();

        list.add(100);
        list.add(300);
        list.add(200);
        list.add(50);
        //排序方法
        Collections.sort(list);
        System.out.println(list);
    }
}
结果:
[50,100, 200, 300]
```

我们的集合按照默认的自然顺序进行了排列，如果想要指定顺序那该怎么办呢？

2.2 Comparator比较器

创建一个学生类，存储到ArrayList集合中完成指定排序操作。

Student 类

```java
public class Student{
    private String name;
    private int age;
    //构造方法
    //get/set
    //toString
}
```

测试类：

```java
public class Demo {
    public static void main(String[] args) {
        // 创建四个学生对象 存储到集合中
        ArrayList<Student> list = new ArrayList<Student>();

        list.add(new Student("rose",18));
        list.add(new Student("jack",16));
        list.add(new Student("abc",20));
        Collections.sort(list, new Comparator<Student>() {
          @Override
            public int compare(Student o1, Student o2) {
            return o1.getAge()-o2.getAge();//以学生的年龄升序
         }
        });


        for (Student student : list) {
            System.out.println(student);
        }
    }
}
Student{name='jack', age=16}
Student{name='rose', age=18}
Student{name='abc', age=20}
```

3. 综合练习

练习1：随机点名器

需求：班级里有N个学生，实现随机点名器

代码实现：

```
public class Test1 {
    public static void main(String[] args) {
        /* 班级里有N个学生，学生属性:姓名，年龄，性别。
        实现随机点名器。*/


        //1.定义集合
        ArrayList<String> list = new ArrayList<>();
        //2.添加数据
        Collections.addAll(list,"范闲","范建","范统","杜子腾","杜琦燕","宋合泛","侯笼藤","朱益群","朱穆朗玛峰","袁明媛");
        //3.随机点名
        /* Random r = new Random();
        int index = r.nextInt(list.size());
        String name = list.get(index);
        System.out.println(name);*/

        //打乱
        Collections.shuffle(list);

        String name = list.get(0);
        System.out.println(name);


    }
}
```

练习2：带概率的随机

需求：

班级里有N个学生

要求在随机的时候，70%的概率随机到男生，30%的概率随机到女生

代码实现：

```
public class Test2 {
    public static void main(String[] args) {
        /* 班级里有N个学生
        要求:
        70%的概率随机到男生
        30%的概率随机到女生

        "范闲","范建","范统","杜子腾","宋合泛","侯笼藤","朱益群","朱穆朗玛峰",
        "杜琦燕","袁明媛","李猜","田蜜蜜",
        */
        //1.创建集合
        ArrayList<Integer> list = new ArrayList<>();
        //2.添加数据
        Collections.addAll(list,1,1,1,1,1,1,1);
        Collections.addAll(list,0,0,0);
        //3.打乱集合中的数据
        Collections.shuffle(list);
        //4.从list集合中随机抽取0或者1
        Random r = new Random();
        int index = r.nextInt(list.size());
        int number = list.get(index);
        System.out.println(number);
        //5.创建两个集合分别存储男生和女生的名字
        ArrayList<String> boyList = new ArrayList<>();
        ArrayList<String> girlList = new ArrayList<>();

        Collections.addAll(boyList,"范闲","范建","范统","杜子腾","宋合泛","侯笼藤","朱益群","朱穆朗玛峰");
        Collections.addAll(girlList,"杜琦燕","袁明媛","李猜","田蜜蜜");

        //6.判断此时是从boyList里面抽取还是从girlList里面抽取
```

```
        if(number == 1){
            //boyList
            int boyIndex = r.nextInt(boyList.size());
            String name = boyList.get(boyIndex);
            System.out.println(name);
        }else{
            //girlList
            int girlIndex = r.nextInt(girlList.size());
            String name = girlList.get(girlIndex);
            System.out.println(name);
        }


    }
}
```

练习3：随机不重复

需求：

班级里有N个学生，被点到的学生不会再被点到。但是如果班级中所有的学生都点完了，需要重新开启第二轮点名。

代码实现：

```
public class Test3 {
    public static void main(String[] args) {
       /* 班级里有5个学生
        要求：
        被点到的学生不会再被点到。
        但是如果班级中所有的学生都点完了，需要重新开启第二轮点名。*/


        //1.定义集合
        ArrayList<String> list1 = new ArrayList<>();
        //2.添加数据
        Collections.addAll(list1, "范闲", "范建", "范统", "杜子腾", "杜琦燕", "宋合泛", "侯笼藤", "朱益群", "朱穆朗玛峰", "袁明媛");
        //创建一个临时的集合，用来存已经被点到学生的名字
        ArrayList<String> list2 = new ArrayList<>();
        //外循环：表示轮数
        for (int i = 1; i <= 10; i++) {
```

```
            System.out.println("=========第" + i + "轮点名开始了
=====================");
            //3.获取集合的长度
            int count = list1.size();
            //4.随机点名
            Random r = new Random();
            //内循环：每一轮中随机循环抽取的过程
            for (int j = 0; j < count; j++) {
                int index = r.nextInt(list1.size());
                String name = list1.remove(index);
                list2.add(name);
                System.out.println(name);
            }
            //此时表示一轮点名结束
            //list1 空了 list2 10个学生的名字
            list1.addAll(list2);
            list2.clear();

        }
    }
}
```

练习4：集合的嵌套

需求：

定义一个Map集合，键用表示省份名称province，值表示市city，但是市会有多个。

添加完毕后，遍历结果格式如下：

江苏省 = 南京市，扬州市，苏州市，无锡市，常州市

```
湖北省 = 武汉市，孝感市，十堰市，宜昌市，鄂州市

河北省 = 石家庄市，唐山市，邢台市，保定市，张家口市
```

代码实现：

```
public class Test4 {
    public static void main(String[] args) {
        /* 需求
```

```
        定义一个Map集合，键用表示省份名称province，值表示市city，但是市会有多个。
        添加完毕后，遍历结果格式如下：
                江苏省 = 南京市，扬州市，苏州市，无锡市，常州市
                湖北省 = 武汉市，孝感市，十堰市，宜昌市，鄂州市
                河北省 = 石家庄市，唐山市，邢台市，保定市，张家口市*/


        //1.创建Map集合
        HashMap<String, ArrayList<String>> hm = new HashMap<>();

        //2.创建单列集合存储市
        ArrayList<String> city1 = new ArrayList<>();
        city1.add("南京市");
        city1.add("扬州市");
        city1.add("苏州市");
        city1.add("无锡市");
        city1.add("常州市");

        ArrayList<String> city2 = new ArrayList<>();
        city2.add("武汉市");
        city2.add("孝感市");
        city2.add("十堰市");
        city2.add("宜昌市");
        city2.add("鄂州市");

        ArrayList<String> city3 = new ArrayList<>();
        city3.add("石家庄市");
        city3.add("唐山市");
        city3.add("邢台市");
        city3.add("保定市");
        city3.add("张家口市");

        //3.把省份和多个市添加到map集合
        hm.put("江苏省",city1);
        hm.put("湖北省",city2);
        hm.put("河北省",city3);

        Set<Map.Entry<String, ArrayList<String>>> entries = hm.entrySet();
        for (Map.Entry<String, ArrayList<String>> entry : entries) {
```

```
            //entry依次表示每一个键值对对象
            String key = entry.getKey();
            ArrayList<String> value = entry.getValue();
            StringJoiner sj = new StringJoiner(", ","","");
            for (String city : value) {
                sj.add(city);
            }
            System.out.println(key + " = " + sj);

        }
    }
}
```

4. 斗地主发牌

4.1 案例介绍

按照斗地主的规则，完成洗牌发牌的动作。
具体规则：

使用54张牌打乱顺序,三个玩家参与游戏，三人交替摸牌，每人17张牌，最后三张留作底牌。

4.2 案例分析

- 准备牌：

  牌可以设计为一个ArrayList,每个字符串为一张牌。
  每张牌由花色数字两部分组成，我们可以使用花色集合与数字集合嵌套迭代完成每张牌的组装。
  牌由Collections类的shuffle方法进行随机排序。

- 发牌

  将每个人以及底牌设计为ArrayList,将最后3张牌直接存放于底牌，剩余牌通过对3取模依次发牌。

- 看牌

  直接打印每个集合。

4.3 代码实现

```
public class App {
    public static void main(String[] args) {
      /*
        完成控制台版的三步：
          准备牌
          洗牌
          发牌
       */
        //从程序的主入口开启斗地主游戏
        new PokerGame();
    }
}

public class PokerGame {
    //牌盒
    //♥3 ♣3
    static ArrayList<String> list = new ArrayList<>();

    //静态代码块
    //特点：随着类的加载而在加载的，而且只执行一次。
    static {
        //准备牌
        // "♦", "♣", "♥", "♠"
        // "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A", "2"
        String[] color = {"♦", "♣", "♥", "♠" };
        String[] number = {"3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A", "2"};

        for (String c : color) {
            //c依次表示每一种花色
            for (String n : number) {
                //n 依次表示每一个数字
                list.add(c + n);
            }
        }
        list.add("小王");
        list.add("大王");
    }

```

```
39      public PokerGame(){
40          //洗牌
41          Collections.shuffle(list);
42
43          //发牌
44          ArrayList<String> lord = new ArrayList<>();
45          ArrayList<String> player1 = new ArrayList<>();
46          ArrayList<String> player2 = new ArrayList<>();
47          ArrayList<String> player3 = new ArrayList<>();
48
49          //遍历牌盒得到每一张牌
50          for (int i = 0; i < list.size(); i++) {
51              //i：索引
52              String poker = list.get(i);
53              if(i <= 2){
54                  lord.add(poker);
55                  continue;
56              }
57
58              //给三个玩家轮流发牌
59              if(i % 3 == 0){
60                  player1.add(poker);
61              }else if(i % 3 == 1){
62                  player2.add(poker);
63              }else{
64                  player3.add(poker);
65              }
66          }
67          //看牌
68          lookPoker("底牌",lord);
69          lookPoker("钢脑壳",player1);
70          lookPoker("大帅比",player2);
71          lookPoker("蛋筒",player3);
72
73      }
74
75      /*
76      * 参数一：玩家的名字
77      * 参数二：每位玩家的牌
78      * */
79      public void lookPoker(String name, ArrayList<String> list){
80          System.out.print(name + ": ");
```

```
        for (String poker : list) {
            System.out.print(poker + " ");
        }
        System.out.println();
    }
}
```

4.4 排序（第一种排序方式）

```
public class App {
    public static void main(String[] args) {
      /*
        完成控制台版的四步：
          准备牌
          洗牌
          发牌
          排序

       */

        //从程序的主入口开启斗地主游戏
        new PokerGame();
    }
}


public class PokerGame {
    //牌盒 Map
    //此时我们只要把牌跟序号产生对应关系就可以了，不需要按照序号进行排序，所以只要HashMap就可以了
    static HashMap<Integer, String> hm = new HashMap<>();
    static ArrayList<Integer> list = new ArrayList<>();

    static {
        String[] color = {"♦", "♣", "♥", "♠"};
        String[] number = {"3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A", "2"};

        //序号
        int serialNumber = 1;
        //细节
```

```
        for (String n : number) {
            //依次表示每一个数字
            for (String c : color) {
                //依次表示每一个花色
                hm.put(serialNumber, c + n);
                list.add(serialNumber);
                serialNumber++;
            }
        }

        hm.put(serialNumber, "小王");
        list.add(serialNumber);
        serialNumber++;
        hm.put(serialNumber, "大王");
        list.add(serialNumber);

    }

    public PokerGame() {
        //洗牌
        Collections.shuffle(list);

        //发牌
        TreeSet<Integer> lord = new TreeSet<>();
        TreeSet<Integer> player1 = new TreeSet<>();
        TreeSet<Integer> player2 = new TreeSet<>();
        TreeSet<Integer> player3 = new TreeSet<>();

        for (int i = 0; i < list.size(); i++) {
            //i :依次表示集合中的每一个索引
            //list.get(i)元素：牌的序号
            int serialNumber = list.get(i);

            if(i <= 2){
                lord.add(serialNumber);
                continue;
            }

            if(i % 3 == 0){
                player1.add(serialNumber);
            }else if(i % 3 == 1){
                player2.add(serialNumber);
```

```
            }else{
                player3.add(serialNumber);
            }
        }


        //看牌
        lookPoker("底牌",lord);
        lookPoker("钢脑壳",player1);
        lookPoker("大帅比",player2);
        lookPoker("蛋筒",player3);

    }

    /*
    * 参数一：玩家的名字
    * 参数二：牌的序号
    * */
    public void lookPoker(String name, TreeSet<Integer> ts){
        System.out.print(name + ": ");
        //遍历TreeSet集合得到每一个序号，再拿着序号到Map集合中去找真正的牌
        for (int serialNumber : ts) {
            String poker = hm.get(serialNumber);
            System.out.print(poker + " ");
        }
        System.out.println();
    }
}
```

4.5 排序（第二种排序方式）

```
public class App {
    public static void main(String[] args) {
        new PokerGame();
    }
}


public class PokerGame {
    //牌盒
```

```
10      static ArrayList<String> list = new ArrayList<>();
11
12      //创建一个集合，用来添加牌的价值
13      static HashMap<String, Integer> hm = new HashMap<>();
14
15      static {
16          //准备牌
17          String[] color = {"♦", "♣", "♥", "♠"};
18          String[] number = {"3", "4", "5", "6", "7", "8", "9",
    "10", "J", "Q", "K", "A", "2"};
19
20          for (String c : color) {
21              for (String n : number) {
22                  list.add(c + n);
23              }
24          }
25          list.add(" 小王");
26          list.add(" 大王");
27
28
29          //指定牌的价值
30          //牌上的数字到Map集合中判断是否存在
31          //存在，获取价值
32          //不存在，本身的数字就是价值
33          hm.put("J", 11);
34          hm.put("Q", 12);
35          hm.put("K", 13);
36          hm.put("A", 14);
37          hm.put("2", 15);
38          hm.put("小王", 50);
39          hm.put("大王", 100);
40
41
42      }
43
44      public PokerGame() {
45          //洗牌
46          Collections.shuffle(list);
47
48          //发牌
49          ArrayList<String> lord = new ArrayList<>();
50          ArrayList<String> player1 = new ArrayList<>();
```

```
        ArrayList<String> player2 = new ArrayList<>();
        ArrayList<String> player3 = new ArrayList<>();

        for (int i = 0; i < list.size(); i++) {
            String poker = list.get(i);
            //发底牌
            if (i <= 2) {
                lord.add(poker);
                continue;
            }

            //给三个玩家轮流发牌
            if (i % 3 == 0) {
                player1.add(poker);
            } else if (i % 3 == 1) {
                player2.add(poker);
            } else {
                player3.add(poker);
            }
        }


        //排序
        order(lord);
        order(player1);
        order(player2);
        order(player3);


        //看牌
        lookPoker("底牌",lord);
        lookPoker("钢脑壳",player1);
        lookPoker("大帅比",player2);
        lookPoker("蛋筒",player3);
    }


    /*
     * 参数一：玩家的名字
     * 参数二：每位玩家的牌
     * */
    public void lookPoker(String name, ArrayList<String> list){
```

```
93          System.out.print(name + ": ");
94          for (String poker : list) {
95              System.out.print(poker + " ");
96          }
97          System.out.println();
98
99      }
100
101
102     //利用牌的价值进行排序
103     //参数：集合
104     //♥5 ♥3 ♥6 ♥7 ♥9
105     public void order(ArrayList<String> list){
106         Collections.sort(list, new Comparator<String>() {
107             //Array.sort （插入排序 + 二分查找）
108             @Override
109             public int compare(String o1, String o2) {
110                 //o1：表示当前要插入到有序序列中的牌
111                 //o2：表示已经在有序序列中存在的牌
112
113                 //负数：o1小 插入到前面
114                 //正数：o1大 插入到后面
115                 //0：o1的数字跟o2的数字是一样的，需要按照花色再次排序
116
117                 //1.计算o1的花色和价值    大王
118                 String color1 = o1.substring(0, 1);
119                 int value1 = getValue(o1);
120
121                 //2.计算o2的花色和价值
122                 String color2 = o2.substring(0, 1);
123                 int value2 = getValue(o2);
124
125                 //3.比较o1和o2的价值     ♥3  ♠3
126                 int i = value1 - value2;
127                 return i == 0 ? color1.compareTo(color2) : i;
128
129             }
130         });
131     }
132
133     //计算牌的价值
134     //参数：牌
```

```
135     //返回值：价值
136     public int getValue(String poker){//♥3
137         //获取牌上的数字
138         String number = poker.substring(1);//把这里截取出来的结果，
    让这个结果再Map集合中存在 “ 大王”
139         //拿着数字到map集合中判断是否存在
140         if(hm.containsKey(number)){
141             //存在，获取价值
142             return hm.get(number);
143         }else{
144             //不存在，类型转换
145             return Integer.parseInt(number);
146         }
147     }
148 }
```