

面向葫芦娃编程：

一、主要类说明：

- 【主类】 class GourdSort:

```
public class GourdSort{
    static int babynum = 7;
    static GourdBaby[] brothers = new GourdBaby[babynum];
    static Cards cards = new Cards(babynum);
    public static void main(String[] args){
    public static void initialize()
    public static void orchestrationSort()
    public static void choreographySort()
    public static void printResult()
}
```

成员简介：

数据成员包括【葫芦娃的个数】babynum，【葫芦娃类数组】brothers，以及【卡牌数组】cards；

方法成员包括【主方法】main，【初始化方法】initialize，【Orchestration排序算法】orchestrationSort，【choreography排序算法】choreographySort，以及【排序结果打印方法】printResult；

部分方法功能简介：

main:

```
public static void main(String[] args)
{
    System.out.println("-----The first way to sort as orchestration-----");
    initialize();
    orchestrationSort();
    printResult();

    System.out.println("-----The second way to sort as choreography-----");
    initialize();
    choreographySort();
    printResult();
}
```

=> 初始化葫芦娃数组，并依次调用两种形式的排序算法对数组进行排序，最后打印排序结果

initialize:

```

public static void initialize()
{
    cards.shuffle();
    for(int i=0;i<babynum;i++)
    {
        brothers[i] = new GourdBaby(cards.deal());
        brothers[i].setpos(i);
    }
    System.out.println("The original order is as below:");
    for(int i=0;i<babynum;i++)
    {
        brothers[i].numberOff();
    }
}

```

=> 调用Cards类的成员cards的【洗牌方法】shuffle随机打乱次序，再依次通过【发牌方法】deal将初始的战队序号分给葫芦娃

orchestrationSort: => 详见“二、Orchestration排序”

choreographySort: => 详见“三、Choreography排序”

- 【葫芦娃类】 class GourdBaby:

```

class GourdBaby extends Thread
{
    private Color color;
    private int pos;
    private static GourdBaby[] brothers;
    GourdBaby(int v){}
    public void run(){}
    public void setpos(int p){}
    public int getRank(){}
    public void setRank(int v){}
    public void numberOff(){}
    public boolean cmp(GourdBaby other){}
    public void swap(GourdBaby other){}
    private synchronized void lookAround(){}
    private synchronized boolean doneAll(){}
    public static void queue(int idx,GourdBaby[] bros){}
    private static void threadSort(){}
    private static void bubbleSort(){}
    private static void quickSort(){}
    private static void mergeSort(){}
    public void printResult(){}
    class Color{}
}

```

成员简介:

数据成员包括【葫芦娃颜色属性】color，【葫芦娃位序】pos，以及【静态数组】brothers；

方法成员主要包括【线程执行方法】run，【排行相关方法】setpos,getRank,setRank，【报数方法】numberoff，【比较排行方法】cmp，【交换次序方法】swap，【观察左右调整次序方法】lookAround，【判断是否排队完成方法】doneAll，【排序算法调用方法】queue，【一系列排序算法】threadSort,bubbleSort,quickSort等；

部分方法功能简介:

numberoff:

```

public void numberOff()
{
    System.out.println("I am the "+color.getColor()+" Baby!");
}

```

=> 我采用的是输出“I am the Color Baby!”来报数，例如大娃会输出“I am the Red Baby!”，二娃会输出“I am the Orange Baby!”，以此类推；

cmp 和 swap:

```

public boolean cmp(GourdBaby other)
{
    if(this.getRank()<other.getRank())
        return true;
    return false;
}
public void swap(GourdBaby other)
{
    int rk1 = this.getRank();
    int rk2 = other.getRank();
    this.setRank(rk2);
    other.setRank(rk1);
}

```

=> cmp会比较【自己】this和【兄弟】other的排行，若自己排行更小则返回true，反之返回false；

=> swap则交换【自己】this和【兄弟】other的排行，以达到间接交换次序的目的；

run,lookAround,queue等: => 详见“三、choreography排序”

■ 【爷爷类】class GrandFather:

```

class GrandFather
{
    private GourdBaby[] brothers;
    public boolean cmp(GourdBaby gb1,GourdBaby gb2)
    public void swap(GourdBaby gb1,GourdBaby gb2)
    public void command(int idx,GourdBaby[] bros)
    private void bubbleSort()
    private void quickSort()
    private void mergeSort()
}

```

成员简介:

数据成员包括【静态葫芦娃数组】brothers;

方法成员主要包括【比较两葫芦娃排行方法】cmp，【交换两葫芦娃次序方法】swap，【指挥方法/排序算法调用方法】command，【一系列排序算法】bubbleSort,quickSort等；

部分方法功能简介:

cmp 和 swap:

```

public boolean cmp(GourdBaby gb1,GourdBaby gb2)
{
    if(gb1.cmp(gb2))
        return true;
    return false;
}
public void swap(GourdBaby gb1,GourdBaby gb2)
{
    gb1.swap(gb2);
}

```

=> cmp会比较【两个葫芦娃】gb1和gb2的排行，若前者排行更小则返回true，反之返回false；

=> swap则交换【两个葫芦娃】gb1和gb2的排行，以达到间接交换次序的目的；

command: => 详见“二、Orchestration排序”

- 【卡牌类】class Cards:

```
class Cards
{
    int num = 0;
    int next = 0;
    boolean[] visited;
    int[] cards;
    Cards(int n){}
    public void shuffle(){}
    public int deal(){}
    public void clear(){}
}
```

成员简介:

=> 数据成员包括【牌数】num，【当前发牌序号】next，【访问互斥数组】visited，以及【卡牌数组】cards

=> 方法成员包括【洗牌方法】shuffle，【发牌方法】deal，【重发方法】clear

部分方法简介:

=> 略

二、Orchestration排序:

- 【算法思想】

让【爷爷】GrandFather类对象gf，通过【比较两个葫芦娃的排行】cmp，和【交换两个葫芦娃的次序】swap两个基本操作，【指挥葫芦娃】command，【调用一系列的基于比较的排序算法】bubbleSort,quickSort等，对【葫芦娃数组】brothers进行排序

- 【具体实现】

orchestrationSort:

```
public static void orchestrationSort()
{
    int choice = 0;

    GrandFather gf = new GrandFather();
    gf.command(choice,brothers);
}
```

=> 在【主类】GourdSort中，通过调用【orchestration排序算法】orchestrationSort，【选择排序算法的序号】choice，然后调用【爷爷类的指挥方法】gf.command，就可以通过choice选择的排序算法对【葫芦娃数组】brothers进行排序了

=> 排序算法只需要通过更改choice的值即可替换

command:

```

public void command(int idx, GourdBaby[] bros)
{
    brothers = bros;
    switch(idx)
    {
        case 0: bubbleSort(); break;
        case 1: quickSort(); break;
        case 2: mergeSort(); break;
        default: bubbleSort(); break;
    }
}

```

=> 赋值【葫芦娃数组的引用】brothers

=> 通过switch语句，调用相应的排序算法，此处默认为冒泡算法

bubbleSort:

```

private void bubbleSort()
{
    int n = brothers.length;
    for(int i=0; i<n-1; i++)
        for(int j=0; j<n-i-1; j++)
        {
            if(!cmp(brothers[j], brothers[j+1]))
                swap(brothers[j], brothers[j+1]);
        }
}

```

=> 爷爷通过cmp和swap两个基本操作，利用冒泡的思想，对葫芦娃数组完成了排序

=> 此处仅写了冒泡算法为例，其他的排序算法暂未实现（但只要是基于比较的排序算法，通过cmp和swap两个操作一定都可以实现）

三、Choreography排序:

▪ 【算法思想】

让【葫芦娃】brothers[i]通过【比较操作】cmp，前后比较【相邻葫芦娃】brother[i-1]或brother[i+1]的排行，相应地进行【交换】swap，即可协作以完成排序，

但此处不同于Orchestration排序的是，除了【调用一系列基于比较的排序算法】bubbleSort, quickSort之外，由于葫芦娃的运动的“相似性”以及“可并发性”，可以利用【多线程编程】的技术，实现一个基于比较的【多线程排序算法】threadSort，这样可能使得“协作效率”大大提高；

▪ 【具体实现】

choreographySort:

```

public static void choreographySort()
{
    int choice = 0;

    GourdBaby.queue(choice, brothers);
    try{Thread.sleep(500);}
    catch(InterruptedException e) {System.out.println(e.toString());}
}

```

=> 在【主类】GourdSort中，通过调用【choreography排序算法】choreographySort，【选择排序算法的序号】choice，然后调用【葫芦娃类的静态排队方法】GourdbBaby.queue，就可以通过choice选择的排序算法对【葫芦娃数组】brothers进行排序了

=> 排序算法只需要通过更改choice的值即可替换

queue:

```
public static void queue(int idx,GourdBaby[] bros)
{
    brothers = bros;
    switch(idx)
    {
        case 0: threadSort(); break;
        case 1: bubbleSort(); break;
        case 2: quickSort(); break;
        case 3: mergeSort(); break;
        default: threadSort(); break;
    }
}
```

=> 赋值【葫芦娃数组的引用】brothers

=> 通过switch语句，调用相应的排序算法，此处默认为多线程算法

threadSort和bubbleSort:

```
private static void threadSort()
{
    for(int i=0;i<brothers.length;i++)
    {
        brothers[i].start();
    }
}
private static void bubbleSort()
{
    int n = brothers.length;
    for(int i=0;i<n-1;i++)
        for(int j=0;j<n-i-1;j++)
        {
            if(!brothers[j].cmp(brothers[j+1]))
                brothers[j].swap(brothers[j+1]);
        }
}
```

=> threadSort直接通过调用每个【葫芦娃的线程启动函数】start，即可开启线程

=> bubbleSort原理同“二、Orchestration排序”，不再赘述

run:

```
public void run()
{
    while(true)
    {
        lookAround();
        if(doneAll())
            break;
    }
}
```

=> 线程执行函数，每个葫芦娃不断【观察周围】lookAround，并相应地调整次序

=> 直到【判断所有的人都站好了】即可退出线程

lookAroud和doneAll:

```
private synchronized void lookAround()
{
    if(pos>0)
    {
        if(cmp(brothers[pos-1]))
            swap(brothers[pos-1]);
    }
    if(pos<brothers.length-1)
    {
        if(!cmp(brothers[pos+1]))
            swap(brothers[pos+1]);
    }
}
private synchronized boolean doneAll()
{
    boolean doneall = true;
    for(int i=0;i<brothers.length;i++)
    {
        if(brothers[i].getRank()!=i)
        {
            doneall = false;
            break;
        }
    }
    return doneall;
}
```

=> lookAround即观察相邻左右兄弟的次序，并相应地进行交换调整（当然最左边和最右边的只能观察一边）

=> 若干次迭代后，如果【所有葫芦娃都站好队了】doneAll，则运行当前线程退出；

四、运行示例及说明：

- 【终端运行示例】

```
-----The first way to sort as orchestration-----
The original order is as below:
I am the Purple Baby!
I am the Orange Baby!
I am the Green Baby!
I am the Yellow Baby!
I am the Cyan Baby!
I am the Red Baby!
I am the Blue Baby!
The result order is as below:
I am the Red Baby!
I am the Orange Baby!
I am the Yellow Baby!
I am the Green Baby!
I am the Cyan Baby!
I am the Blue Baby!
I am the Purple Baby!
-----The second way to sort as choreography-----
The original order is as below:
I am the Yellow Baby!
I am the Orange Baby!
I am the Purple Baby!
I am the Blue Baby!
I am the Cyan Baby!
I am the Red Baby!
I am the Green Baby!
The result order is as below:
I am the Red Baby!
I am the Orange Baby!
I am the Yellow Baby!
I am the Green Baby!
I am the Cyan Baby!
I am the Blue Baby!
I am the Purple Baby!
```

■ 【运行说明】

=> 两种不同形式的排序过程依次打印

=> 排序过程首先打印【初始序列报数】original order，再打印【排好序报数】result order

=> 另外，值得说明的是，本人在测试过程中，发现偶尔【多线程算法】会打印出错，经过调试后发现是由于主线程的【结果打印函数】`printResult`先于若干葫芦娃子线程执行导致的，即使本人在主线程中添加了【线程睡眠】`Thread.sleep(500)`，也只是降低了发生打印错误的概率，仍然有可能发生打印错误

=> 因此，希望老师或助教在测试代码时，如果遇到打印错误，请再测试1次或n次，勿轻易判定为代码不合格，谢谢！