Java应用技术-HW2

一、实验内容

- 实现一个火柴棒游戏，满足如下要求
  - 用户从命令行输入最大数字的位数（如1位数、2位数、3位数）；
  - 用户从命令行输入提示数（2或3），表示等号左式数字的个数；
  - 用户从命令行输入题目类型编号（移动、移除、添加），以及火柴棒根数；
  - 系统随机自动生成火柴棒游戏，并展示（直接用数字展示）；
  - 用户输入答案，系统验证是否正确；若正确，则提示正确；若错误，则让用户继续输入；
  - 若用户直接回车，则显示正确答案。

二、实验环境

- 操作系统：Windows 10 发行版
- JDK版本：1.8
- IDE：IntelliJ IDEA 2020版

三、实验具体实现过程

3.1 程序代码文件

- 本次作业的代码量约为660行左右，工作量比较大，程序的源代码分为如下几个java文件：

   image-20200930104825411
  - 其中JavaHW2是入口类，包含main函数
  - game文件夹中的丝格蕾分别包含如下功能：
    - checkGame包含检查用户输入答案，判断游戏是否结束等功能
    - showGame包含的主要功能是将生成的火柴棒游戏绘制到控制台的字符界面上
    - staticData包含一系列静态数据和与之相关的操作：找到使得火柴棒数目变动最大的改变方式
    - matchGame是本程序的核心类，用于产生一个火柴棒游戏，接受用户的输入输出，并生成游戏的答案，调用其他几个类生成游戏的题目，展示题目和检测用户的输入是否正确

3.2 核心类的实现

3.2.1 matchGame类

- 该类中用到了**随机算法**等算法来帮助我们生成合适的题目，详细的会在下面进行介绍
- 这是本程序的核心类，可以接受四个参数，分别代表作业要求中的四个参数并随机生成一个**正确的表达式**和**满足游戏需求的题目**
  - 该类有如下成员变量

```
public class matchGame {
    public int maxDigit, number, mode, count;
    public int[] operateNumber;
    public int result = 0;
    public char[] operator;
    public char[] expr; // 作为题目的表达式，先保存正确答案然后再根据一定的算法更改
    public char[] trueExpr; // 作为答案的表达式
};
```

- 该类的构造方法如下，接受的四个参数就是题目所要求的，并对重要数组进行初始化

```
public matchGame(int maxDigit, int maxNum, int gameMode, int match) {
    this.maxDigit = maxDigit;
    this.number = maxNum;
    this.mode = gameMode;
    this.count = match;
    operateNumber = new int[number];
    operator = new char[number];
}
```

- 之后主程序main函数会调用这个类的runGame方法，而runGame方法的代码如下：

```
public void runGame() {
        System.out.println("----------------------游戏开始----------------------");
        boolean ok = false;
        // 成功生成题目的条件还要再考虑：当选择了移动的时候，不能使得题目和答案相同
        do {
            this.generateGame();
            this.trueExpr = this.generateExpr(operateNumber, operator, result);
            ok = this.gameShuffle();
        } while(!ok || checkGame.compare(this.expr, this.trueExpr));
        showGame.showMatchGame(expr);
        checkGame.checkResult(trueExpr, this.mode, this.count);
    }
```

- 这个类会进行如下操作
  - 首先调用`generateGame()`随机地生成一个正确的火柴棒表达式(这一部分代码比较长就不贴了)，采用java的Random类生成若干满足位数要求的数和运算符，并计算结果得到一个正确的火柴棒表达式，如果表达式不满足条件就会重新生成
  - 之后会将表达式扩展成为一个15位的**标准表达式**(15位是因为三个运算数字最多三位，加上结果和运算符，就是3*3+3+3=15)一个表达式最多只有15个字符，因此多的位数用*来填充，这一部分的代码如下

```
public char[] generateExpr(int[] num, char[] op, int res) {
        StringBuilder temp1 = new StringBuilder();
        StringBuilder temp2 = new StringBuilder();
        this.expr = new char[15];
        for(int i = 0; i < num.length; i++) {
            // 为完整的表达式加上空位
```

```
            if(num[i] <= 9)
                temp2.append("**");
            else if(num[i] <= 99)
                temp2.append("*");
            temp1.append(num[i]);
            temp1.append(op[i]);
            temp2.append(num[i]);
            temp2.append(op[i]);
        }
        temp1.append(res);
        // System.out.println(expression);
        //生成有多余位*的表达式
        if(res <= 9)
            temp2.append("**");
        else if(res <= 99)
            temp2.append("*");
        temp2.append(res);
        this.expr = temp2.toString().toCharArray();

        // 将整个表达式转化为一个char[]类型的数组
        return temp1.toString().toCharArray();
    }
```

- 然后调用`gameShuffle`方法随机生成一个作为题目的表达式
  - 首先要明确，对于题目的需求，如果是玩家需要删除2根的题目，那我们就需要在答案的基础上添加2根火柴棒生成一个游戏，供玩家挑战，添加和移动也是相同的道理
  - gameShuffle采用随机算法，每次对15位的标准表达式，随机取出一位下标，并尝试对其对应的数字的火柴棒进行修改，修改的具体算法会在另一个类中具体实现，这里先不详细介绍，并且设置了一定的退出机制，如果尝试多次不成功就会返回上一步重新生成，代码如下

```
/**
* 按照输入的要求随机生成一个新的表达式
*/
public boolean gameShuffle() {
        System.out.println("改变之前的表达式为:");
        System.out.println(this.expr);
        int loop = 0;
        if(mode == 1) {
            // 直到生成合适的表达式之前一直尝试生成合适的表达式
            while(true) {
                loop++;
                if(matchAdd()) {
                    break;
                }
                if(loop >= 10) {
                    return false;
                }
            }
            while(true) {
                loop++;
                if(matchMinus()) {
                    break;
```

```
                }
                if(loop >= 10) {
                    return false;
                }
            }
        } else if(mode == 2) {
            while(true) {
                loop++;
                if(matchAdd()) {
                    break;
                }
                if(loop >= 10) {
                    return false;
                }
            }
        } else {
            while(true) {
                loop++;
                if(matchMinus()) {
                    break;
                }
                if(loop >= 10) {
                    return false;
                }
            }
        }
        return true;
    }
```

- 其中matchMinus和matchAdd就是实现了随机算法的方法，代码如下

```
public boolean matchAdd() {
        int target = this.count;
        int len = this.expr.length;
        int[] visited = new int[len];
        int loop = 0;
        // 每次随机选取一个下标来增加若干根火柴棒，增加的时候采用贪心的方法
        while(target != 0) {
            Random star = new Random();
            int index = star.nextInt(len);
            if(visited[index] >= 3 || this.expr[index] == '=') {
                continue;
            } else {
                int res = staticData.findMaxChange(this.expr, index, target);
                target -= res;
                visited[index]++;
                loop++;
            }
            // 设置一个循环次数的上限，超过这个上限就表示生成失败，需要重新生成一次
            if(loop >= 10) {
```

```
                return false;
            }
        }
        return true;
    }
```

    - 这里将“移动n根”类型的题目的生成转化成了先添加n根再删除n根，因此可能会造成题目和答案一样，这时我们**设立了检测机制**，如果生成的题目和答案是一样的们就会重新生成
  - 之后runGame就调用showGame类来在控制台中展示出这个题目，并调用checkGame类让用户输入答案并进行判断，这一部分类的实现在下面介绍

3.2.2 staticData类

- 该类主要存储了一些跟火柴棒自身性质有关的静态数据和对单个操作数/运算符的增加/删除的操作
  - 该类包含这样一个成员变量，`diffrence[x][y]` 表示x直接转换成y需要变动的火柴棒的数目，其中0-9表示数字0-9，而10，11，12，13的下标分别表示+，-，=这几个符号

```
public static int[][] diffrence = {
    {0, -4, 0, 0, 0, 0, 0, -3, 1, 0, 0, 0, 0}, // 0 - 9
            {4, 0, 0, 3, 2, 0, 0, 1, 5, 4, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0},
            {0, -3, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0},
            {0, -2, 0, 0, 0, 0, 0, 0, 3, 2, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 1, 0, 2, 1 , 0, 0, 0},
            {0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0, 0, 0},
            {3, -1, 0, 2, 0, 0, 0, 0, 4, 3, 0, 0, 0},
            {-1, -5, -2, -2, -3, -2, -1, -4, 0,-1, 0, 0, 0},
            {0, -4, 0, -1, -2, -1, 0, -3, 1, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0}, // +
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0}, // -
            {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
```

  - 另外该类的方法`numberOfMatch`可以返回单个字符需要的火柴数，`matchOfExpr` 可以计算一个表达式用到的火柴数，这两个方法的实现比较简单，这里仅简要提一下
  - 该类最重要的方法是`findMaxChange(char[] x, int n, int remain)` 和 `findMinChange(char[] x, int n, int remain)` 分别表示在增加，删除火柴棒的时候，对其中一个数字/符号进行操作的策略
    - 采用了贪心算法，每次找到**使得被操作的数字的火柴棒数目变化最大**的数字/符号作为改变的结果，并返回火柴棒改变的根数
    - 转换的规则是：数字只能转换成数字，+和-可以互相转换，=符号不能改变，这是为了保证生成表达式的合法性，并且每次找到变化最大的转换
    - 方法的实现代码如下

```
public static int findMaxChange(char[] x, int n, int remain) {
        int index = getIndex(x, n);
        // 对于数字的增加火柴棒的操作
        if(index <= 9) {
            int maxi = 0, maxChange = 0;
            for(int i = 0; i < 13; i++) {
                if(difference[index][i] > maxChange &&
difference[index][i] <= remain) {
```

```
                    maxChange = difference[index][i];
                    maxi = i;
                }
            }
            x[n] = (char)('0' + maxi);
            return maxChange;
        }
        // 我们规定加号不会变化，方便生成游戏
        else if(index == 10) {
            return 0;
        } else if (index == 11) {
            x[n] = '+';
            return 1;
        } else {
            //碰到的是空白位置*，此时只要添加就可以
            //采取的策略是尽可能多的消耗remain，即剩余的火柴棒数量，如果剩下超过了7根，就只能变成8然后用掉7根
            if(remain <= 1) {
                return 0;
            } else if(remain == 2) {
                x[n] = '1';
            } else if(remain == 3) {
                x[n] = '7';
            } else if(remain == 4) {
                x[n] = '4';
            } else if(remain == 5) {
                x[n] = '2';
            } else if(remain == 6) {
                x[n] = '6';
            } else {
                x[n] = '8';
                return 7;
            }
            return remain;
        }
    }
```

3.2.3 showGame类

- 这个类通过二维字符串数组来存储所有的数字/运算符，并用方法`showMatchGame(char[] game)`对游戏中的表达式进行打印
  - 二维字符串数字的定义如下：

```
static String[][] list = new String[][] {
        num0, num1, num2, num3, num4, num5, num6, num7, num8, num9, plus, minus, equal
};
```

  - 其中每个数字/符号用如下方法表示

```
static String [] num6 = new String[] {
        " * * * * ",
        " *     ",
        " * * * * ",
        " *   * ",
        " * * * * "
};
```

- 打印方法的实现如下：通过二层循环进行打印

```
public static void showMatchGame(char[] game) {
        System.out.println("请挑战下面的火柴棒游戏:");
        System.out.println(game);
        for(int i = 0; i < 5; i++) {
            for (char c : game) {
                int index = getIndex(c);
                if(index != -1) {
                    System.out.print(list[getIndex(c)][i]);
                }
            }
            System.out.print('\n');
        }
        System.out.println("\n");
    }
```

3.2.4 checkGame类

- 该类包括了检测用户输入答案，判断游戏是否结束等功能
  - 核心方法是`checkResult(char[] origin, int mode, int match)`开启输入答案的循环`gameLoop(char[] origin, int mode, int match)`，直到输入正确答案或者玩家按下回车键要求显示正确答案

```
public static void checkResult(char[] origin, int mode, int match) {
        // System.out.println(origin);
        while(true) {
            if(!gameLoop(origin, mode, match))
                break;
        }
    }

    public static boolean gameLoop(char[] origin, int mode, int match)
{
        System.out.println("请输入你的答案:");
        Scanner input = new Scanner(System.in);
        String inputResult = input.nextLine();
        System.out.println("输入的答案是:" + inputResult);
        if(inputResult.isEmpty()) {
            System.out.println("正确答案是: ");
            showTrueAnswer(origin);
            return false;
        }
        char[] inputExpression = new char[origin.length];
        int j = 0;
```

```
        for(int i = 0; i < inputResult.length(); i++) {
            if(j >= origin.length) {
                System.out.println("答案错误！请重试！");
                return false;
            }
            if(inputResult.charAt(i) == ' ') {
                continue;
            } else if(inputResult.charAt(i) >= '0' &&
inputResult.charAt(i) <= '9') {
                inputExpression[j++] = inputResult.charAt(i);
            }
            else if(inputResult.charAt(i) == '+' ||
inputResult.charAt(i) == '-' || inputResult.charAt(i) == '=') {
                inputExpression[j++] = inputResult.charAt(i);
            }
            else {
                System.out.println("答案错误！请重试！");
                return false;
            }
        }
        System.out.println(inputExpression);
        int resultNum = staticData.matchOfExpr(inputExpression);
        if(mode == 2) {
            resultNum -= match;
        } else if(mode == 3) {
            resultNum += match;
        }
        if(compare(origin, inputExpression) ||
                (resultNum == staticData.matchOfExpr(origin) &&
isExprRight(inputExpression))) {
            System.out.println("答案正确！恭喜您完成了游戏！");
            return true;
        } else {
            System.out.println("答案错误！请重试！");
            return false;
        }
    }
```

- 其中检测答案是否正确的算法描述如下：
  - 首先读入输入的答案，如果用户输入的是回车键，就直接显示正确的答案并返回，游戏结束
  - 否则就开始判断结果是否正确，首先将原答案的标准表达式形式转换成一般形式(去掉所有表示空的*)，并将输入的内容叶转换成一般形式(去掉空格之类的多余符号)，然后调用compare方法进行判断两个表达式是否一致
  - 但是答案往往不止一种，因此我还设置了另一种检查方式`isExprRight(char[] expr)`通过检查两个表达式的**火柴棒的数量是否相等**(并且还要考虑变动的量)和**表达式结果计算**是否正确，如果都正确，那么这也是一个正确答案

四、程序测试

- 关于测试的几点说明
  - 为了方便输入正确/错误的测试答案，我们在测试的时候会将正确答案先打印出来，这样可以加快测试的速度，否则开发人员可能也做不出随机生成的题目

    - 程序开始运行的时候需要根据提示输入对应的参数，游戏结束的时候也会有相应的提示
    - 因为程序中存在一些事务回滚的机制，所以可能会多次生成表达式，但是除了最后生成的表达式以外其他的都因为无法生成合适的题目而被回滚了，因此出现的题目是最后一次生成的表达式
- 测试1：普通的游戏：删除两根火柴棒

   image-20200930134039169
- 测试2：边界测试样例，所有值取最小的

   image-20200930134113878
- 测试3：边界测试样例，所有参数取最大的
- 测试4：按回车键观察是否给出正确答案

   image-20200930134228634
- 测试5：不合法的初始条件，观察是否会接受新的输入再生成游戏
- 测试6：输入的答案有空格，观察是否会被过滤

   image-20200930134442948
- 测试7：输入错误答案观察是否继续游戏并重新输入答案

   image-20200930134543535

五、心得体会

- 这是Java应用技术课程的第一个正式的编程作业，代码量比较大，当然也有可能是我想复杂了的原因，花了比较长的时间来编写代码和进行测试以及撰写报告
- 整个程序的构思总体上来说是比较自然的，主要的难点在于如何生成一个合适的表达式和如何检测答案是否正确
    - 在生成表达式这个问题上，我结合了随机算法和贪心算法来生成合适的题目，同时用一系列适当的回滚机制来防止随机算法和贪心方法无法求解的情况出现
    - 事实上如果采用遍历等暴力方法来搜索整个表达式来生成题目的话算法的时间复杂度会非常高，代码也会比现在更加难写
    - 而随机算法的目的就在于用随机来使得大部分情况下决策的时间复杂度下降，而会导致少数情况变复杂，因此设置适当的回滚机制就可以取长补短解决这个问题
    - 对于答案正确性的检测我设置了多种检测方法来满足题目的多解性
- 总的来说第一次作业比较有难度，我也从中收获了不少新的知识和技巧，锻炼了代码能力