Lab4（2018-10-16）

董依菡 15302010054@fudan.edu.cn

耿同欣 15302010048@fudan.edu.cn

张星宇 15307110273@fudan.edu.cn

Part1. 日历

任务

接下来的任务中，大家需要实现一个日历，这个日历完成如下的功能

- **提示并接收**用户的输入的年份与月份，限制年份大于等于2000
- 打印该年份与月份对应的日历

```
Please input the year(>=2000)
2018
Please input the month
10
Sun Mon Tue Wed Thu Fri Sat
    1   2   3   4   5   6
7   8   9   10  11  12  13
14  15  16  17  18  19  20
21  22  23  24  25  26  27
28  29  30  31
```

其中，2018和10来自于用户输入，日历的部分是程序根据用户的输入情况进行打印的结果

提示

这次的lab希望大家能够体验到在程序中使用方法的好处，使用方法能够大大简化你的代码，提高开发效率，最重要的是使程序的逻辑清晰易懂。希望同学们仔细阅读提示部分。以下内容只是给大家的建议，程序可以不按照下面的方式进行设计，只要完成了上述任务即可。

- 对需求进行分析：

  在我们写程序之前呢，我们要思考，我们的程序需要实现一个什么样的功能呢？这就是需求分析，在这里，我们需要将用户输入的year和month拿到，放到我们的变量里面，通过对这2个变量进行各种复杂的计算，打印我们的日历。

- 设计我们的程序

试想一下，如果这个需求交给我们人去做，那最简单的方式就是，打开我们电脑中的日历，然后翻到某一年某一月，我们就看到了这个日历。那么整个过程分为三步

- 打开电脑中的日历
- 翻到日历的对应的月份
- 我们得到了这个日历。

对于程序而言，我们是不是也可以制定出具体的步骤，让他来完成这个任务呢？当然可以

- 提示用户输入年份和月份，并将他保存到year,month变量里(1)
- 打印出year,month对应的日历(2)

  好了，我们已经成功地将这个程序分成了两个部分，而且我们发现，(2)的过程仅仅依赖于(1)输出的两个变量，即year和month，这样的设计我们把他叫作“低耦合的”，(1)和(2)之间的关系越小，相互影响也就越小，程序的耦合度越低。低耦合的设计，使得程序更加模块化，部分与部分之间分工明确，当程序出了问题，我们能够很快定位该问题并解决。比如我们发现year,month的值根本不是我们输入的值，那问题一定出现在(1)，否则问题出现在(2)。

  对于步骤(1)的实现这里就不再分析，着重分析下(2)的实现

  打印一个日历，我们需要哪些东西呢，year和month，缺一不可。拿到了year和month，就一定能打印日历嘛？是的，的确是这样。所以，这个方法我们一定能实现出来。我们再细致看这个问题，日历的第一排，打印"Sun Mon Tue Wed Thu Fri Sat "， 这个可以直接写出来。从第二排开始，我们发现可能我们需要一些信息了，首先，每个月的第一天是星期几呢？知道了这个信息，我们才能知道"1"的前面有多少空格对吧。再往后看，我们发现，我们是不是必须还要知道这个月有多少天呢，知道了天数，我们才能知道打印到哪个数字才结束对吧。于是我们把打印过程分为几个部分。

- 打印出year,month对应的日历(2)
  - 计算year,month对应的月份第一天是星期几 (3)
  - 计算year,month月份下有多少天 (4)
  - 根据上面的信息打印地图 (5)

  对于(5)，只要知道了有多少天以及第一天是星期几，剩下的就是字符的处理问题，技巧问题，就交给同学们去实现啦。

  对于(4)，哪些月是31天，哪些月是30天相信同学们都还记得，对于2月份到底是28天还是29天呢？当然就看我们的year是不是闰年啦。对于(4)我们又可以分解为

  - 计算year,month下有多少天(4)
    - 如果month是二月份，我们判断year是否为闰年，决定结果是28还是29 (6)
    - 如果month不是二月份，我们直接得到结果并返回 (7)

  判断year是否是闰年怎么判断呢？同学们自行思考。

  对于(3)，这个问题比较麻烦。仔细想想，假设我们知道了2018年9月份的第一天是星期几，那么我们想求出2018年10月份第一天是星期几是不是就变得容易了呢？所以(3)可以由递推求得，递推在函数的实现就是函数递归调用。由于我们输入的year>=2000,那么我们可以将2000年1月设为初始值。我们查阅日历，得到为星期六，那么我们设为6。注意，星期天我这里用0表示。

  - 计算year,month对应的月份第一天是星期几(3)
    - 如果year等于2000，month等于1，返回6 (8)

        - 否则，计算得到上个月的第一天是星期几，用这个值加上上个月的天数，然后对7取模，得到这个月的第一天是星期几，然后返回。(9)

整个代码的结构为

- 提示用户输入年份和月份，并将他保存到year,month变量里(1)
- 打印出year,month对应的日历(2)
    - 计算year,month对应的月份第一天是星期几 (3)
        - 如果year等于2000，month等于1，返回6 (8)
        - 否则，计算得到上个月的第一天是星期几，用这个值加上上个月的天数，然后对7取模，得到这个月的第一天是星期几，然后返回。(9)
    - 计算year,month月份下有多少天 (4)
        - 如果month是二月份，我们判断year是否为闰年，决定结果是28还是29 (6)
        - 如果month不是二月份，我们直接得到结果并返回 (7)
    - 根据上面的信息打印地图 (5)

```
class CalendarUtil {
    public static void main(String[] args) {
        /* 提示用户输入年份和月份，保存到year和month中 参考(1)*/
        // Your code here
        ... (建议在10行以内)
        printCalendar(year, month); // (2)
    }
    public static void printCalendar(int year, int month) {
        int firstDayOfWeek = getFirstDayOfWeek(year, month); // (3)
        int daysOfMonth = getDaysOfMonth(year, month); // (4)
        /* 根据上面两个的信息打印你的地图 参考(5) */
        // Your code here
        ... (建议在30行以内)
    }
    public static int getDaysOfMonth(int year, int month) {
        boolean leapYear = isLeapYear(year); // (6)
        /* 根据leapYear及month的值返回你的天数 参考 (7) */
        // Your code here
        ... (建议在30行以内)
    }
    public static int getFirstDayOfWeek(int year, int month) {
        /* 参考(8)(9) */
        // Your code here
        ... (建议15行以内)
    }
    public static boolean isLeapYear(int year) {
        // Your code here
       ... (建议5行以内)
    }
}
```

- 测试我们的程序

如果程序出了bug，如何快速定位你的bug出在哪里呢？我们只需要找到出问题的方法就可以了，比如我们可以在main里面输出System.out.println(getDaysOfMonth(2018, 2))，看结果是否为28。类似于这样的方法。同学们也可以用intellij中的debug程序来进行调试，但思路是一样的，我们只需要找到出问题的方法即可。

Part2. 重构Lab3

任务

请大家在lab3中使用静态方法将之前的代码进行封装，考虑哪些代码可以封装为静态方法，哪些静态方法之间可以重载。写一个文档(doc,pdf,txt,pages,md均可)，在文档中你需要描述

- 方法骨架：罗列出你使用的方法，描述每个静态方法接收什么样的参数，完成什么功能，返回什么样的值。

该任务只需要提交一个文档即可

提示

学习了函数，完成了part1，是否对自己lab3的代码有所思考呢，希望大家可以回到lab3，使用函数将一些代码进行封装，比如每次进行移动和旋转时，我们会判断下一个状态是否与容器发生碰撞，这些过程接收方块的下一个状态(包括坐标，每个小块的相对位置，或者旋转状态等等)作为参数，返回一个boolean值。在旋转和移动时都会进行调用。这样的过程是否可以写成一个方法呢。再比如，打印地图，也是可以封装的。

参考的代码框架

```
class Lab3 {
    static int x, y          // 方块中心坐标
    static int rotateState; // 方块旋转状态

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        /* 对x, y, rotateState进行初始化 */
        ...

        while(true) {                          // 循环进行游戏
            printMap(x, y, rotateState);     // 打印整个地图
            System.out.println("w:rotate s:down a:left d:right");// 提示输入
            char nextCommand = scanner.next().charAt(0);          // 得到输入
            boolean gameOver = executeCommand(command);           // 执行指令
            if (gameOver) {
                break;
            }
        }
    }
```

```
/** 打印地图
 *
 * @param x 中心方块横坐标
 * @param y 中心方块纵坐标
 * @param rotateState 方块当前旋转状态
 */
public static void printMap(int x, int y, int rotateState) {
    /* 根据rotateState确定你的4个小方块的相对坐标 */
    int x1,x2,x3,x4,y1,y2,y3,y4;
    switch(rotateState) {
        case 0: // 旋转状态为0度时为x1,x2...y3,y4赋值
            ...
        case 1: // 旋转状态为90度...
            ...
        case 2: // 180度...
            ...
        case 3: // 270度...
            ...

    }

    /* 根据x,y,x1,x2....y3,y4打印地图 */
    ...
}

/** 执行指令
 *
 * @param command 下一条指令
 * @return 游戏是否结束
 */
public static void executeCommand(char command) {
    boolean gameOver = false        // 用一个变量表示游戏是否结束
    switch(nextCommand) {
        case 'w':
            if (!collide(x, y, (rotateState + 1) % 4)) {
                rotate(x, y, rotateState);
            }
            break;
        case 'a':
            if (!collide(x - 1, y, rotateState)) {
                x--;
            }
            break;
        case 'd':
            if (!collide(x + 1, y, rotateState)) {
                x++;
            }
            break;
```

```
            case 's':
                if (!collide(x, y + 1, rotateState)) {
                    y++;
                    gameOver = true;
                }
                break;
        }
        return gameOver;
    }


    /** 判断碰撞
     *
     * @param x 中心方块横坐标
     * @param y 中心方块纵坐标
     * @param rotateState 方块当前旋转状态
     * @return 当前状态下的方块是否与容器底部或旁边碰撞
     */
    public static boolean collide(x, y, rotateState) {
        /* 判断x,y,rotateState下的方块是否超过边界(即碰撞) */
        ...
    }
}
```

- 定义相关状态变量，记录方块旋转状态，方块总体坐标
- 循环进行游戏
  - 打印整个地图 (根据方块总体坐标，方块旋转状态)
    - 根据旋转状态得到子方块的相对坐标
    - 打印容器
    - 打印方块(根据方块总体坐标，子方块的相对坐标)
  - 提示并接收用户下一条指令的输入
    - 提示用户输入下一条指令
    - 得到用户输入的指令
  - 根据指令进行方块操作
    - 预判断该操作是否会碰撞
      - 若不碰撞，允许这次操作
      - 若碰撞，碰撞底部则break循环，游戏结束，碰撞左右则跳过这个操作。

Part3. 提交

- 提交地址：ftp://10.132.141.33/classes/18/181 程序设计A（戴开宇）/WORK_UPLOAD/Lab4
- 提交内容：将Part1中的java程序(java格式)和Part2中的文档打包到一个文件夹里进行压缩，压缩成zip格式，zip压缩包的名称为:学号_姓名_lab4.zip，如:18302010090_我的名字_lab4.zip，将该zip文件提交到上述地址上。

- 截止日期：2018/10/21 23:59:59