检查流程优化导医系统

需求

1.患者；检查项目；检查机器

1.1患者：

1.1.1不断地有患者进入

可以令每x分钟进入一个患者（如果是一个比较优秀的系统，时间就应该随机）

1.1.2患者序列

患者1：项目1，项目2，项目3（每一个患者会有一些要做的项目，这些项目是医院有的项目，已经排好序）

患者2：项目1，项目3，项目6

患者3：项目2，项目6，项目7

等等

1.1.3患者信息

ID（可能是字符串string）

姓名（字符串string）

性别（male/female 可用bool或者直接用字符来标识）

1.2项目

1.2.1检查条件

是否为空腹检查（如果为非空腹，需要给患者一个信号下次再检查）

1.2.2项目的之间的先后关系

e.g.先做项目1，之后才能做项目3（先后关系/条件关系等）

有些项目没有先后优先顺序

1.2.3机器效率

机器1：项目1（y分钟），项目3（z分钟）

机器2：项目1（y分钟），项目6（z分钟），项目7（x分钟）

等等

2.简单设计流程

1.患者进行空腹检查

患者信息导入（ID，姓名，性别，项目序列<项目1，项目2等等>，空腹情况）

若非空腹，则建议明日检查，患者队列剔除该患者，下一名患者接上。

若空腹，则继续

2.项目分配

将按顺序把患者的项目分配到不同的机器，准备检查

3.完成

机器完成项目，将该项目剔除，在患者信息中将该项目标记为已做，当所有项目都完成，该患者剔除

3.限定条件

1.空腹条件是加给项目的，作为判断优先级的

2.记录每个患者的信息，项目次序，最后的项目运行时间

3.机器工作效率优先级（比如1>2那么2的分配的项目就要多）

实现

结果

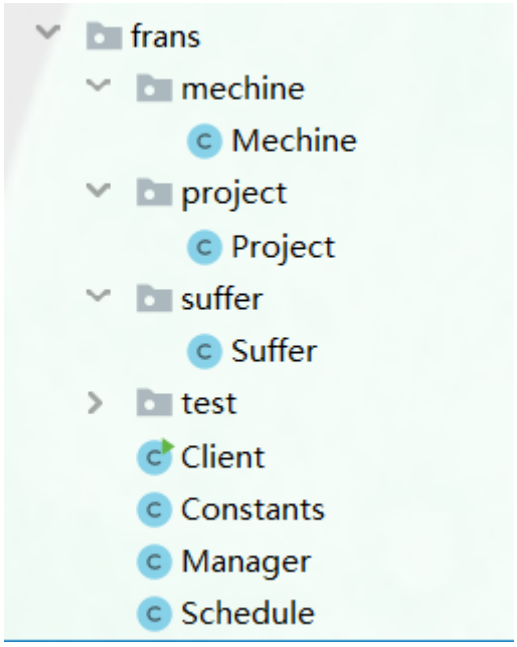
```
====================>Mechines Info
Mechine{mid=0, isIdle=false, project2time={0=7, 1=4, 2=3}, suffers=[], queue={}}
Mechine{mid=1, isIdle=false, project2time={1=5, 2=8}, suffers=[], queue={}}
Mechine{mid=2, isIdle=false, project2time={2=2}, suffers=[], queue={}}

====================>Projects Info
Project{pid=0, isEmpty=true, mechines=[0, 2], preprojects=[2], queue={}}
Project{pid=1, isEmpty=false, mechines=[1], preprojects=[], queue={}}
Project{pid=2, isEmpty=true, mechines=[0, 1, 2], preprojects=[], queue={}}

请输入个人信息：
姓名：
1
性别：
1
年龄：
1
是否空腹：
true
请选择项目：（填项目编号，q退出）
0.项目1
1.项目2
2.项目3

0 1 2 q
==============>Suffer Info
Suffer{sid=1, sname='1', ssex='1', isFast=true, stime=8, projects=[0, 1, 2], ptimes=[]}
==========================>topo Projects
1 2 0
序号    项目编号    时间     机器    所需时间
1       1       8       1       5
2       2       13      2       2
3       0       15      0       7
请输入个人信息：
```

项目结构



过程

对患者的项目使用topo排序

```
private void topo(List<Integer> projects) {

        List<Project> projects1 = new Manager().getProjects();

        int[][] prerequisites = new int[Constants.PROJECTS_NUMS * 10][2];

        boolean[] vis = new boolean[Constants.PROJECTS_NUMS];
        for (boolean vi : vis) {
            vi = false;
        }
        for (int i = 0; i < projects.size(); i++) {
            vis[projects.get(i)] = true;
        }
        int count = 0;
        for (Integer project : projects) {
            List<Integer> preprojects =
projects1.get(project).getPreprojects();
            if(preprojects.size() == 0) {
                int[] p = new int[2];
                p[1] = project;
                p[0] = project;
                prerequisites[count++] = p;
                continue;
            }
            for (Integer preproject : preprojects) {
                if (vis[preproject]) {
                    int[] p = new int[2];
                    p[1] = preproject;
                    p[0] = project;
                    prerequisites[count++] = p;
                }
            }
        }
```

```
        /*
        for (int i = 0; i < count; i++) {
            System.out.println(prerequisites[i][0] + "  " + prerequisites[i][1]);
        }
        */
        //topo
        if (count == 0) return;
        int[] inDegrees = new int[count];
        for (int i = 0; i < count; i++) {
            int[] p = prerequisites[i];
            if(p[0] == p[1]) inDegrees[p[0]] = 0;
            else inDegrees[p[0]]++;
        }

        Queue<Integer> queue = new LinkedList<>();
        for (int i = 0; i < inDegrees.length; i++) {
            if (inDegrees[i] == 0) queue.offer(i);
        }
        int[] res = new int[count];
        count = 0;
        while (!queue.isEmpty()){
            int curr = queue.poll();
            res[count++] = curr;
            for (int[] p : prerequisites) {
                if(p[1] == p[0]) continue;
                if (p[1] == curr){
                    inDegrees[p[0]]--;
                    if (inDegrees[p[0]] == 0) queue.offer(p[0]);
                }
            }
        }
        List<Integer> ans = new ArrayList<>();
        for (int i = 0; i < count; i++) {
            ans.add(res[i]);
        }
        this.projects = ans;

    }
```

对机器和项目的管理已经定义好的

```
private static void setMechines() {
        //map -> <project, time>
        mechines.get(0).getProject2time().put(0, 7);
        mechines.get(0).getProject2time().put(1, 4);
        mechines.get(0).getProject2time().put(2, 3);

        mechines.get(1).getProject2time().put(1, 5);
        mechines.get(1).getProject2time().put(2, 8);

        mechines.get(2).getProject2time().put(2, 2);

        System.out.println("====================>Mechines Info");
        for (Mechine mechine : mechines) {
```

```
            System.out.println(mechine.toString());
        }
        System.out.println();
}
private static void setProjects() {
        //Fast
        projects.get(0).setEmpty(true);
        projects.get(1).setEmpty(false);
        projects.get(2).setEmpty(true);

        //Mechine
        //p[0]
        projects.get(0).getMechines().add(0);
        projects.get(0).getMechines().add(2);
        //p[1]
        projects.get(1).getMechines().add(1);
        //p[2]
        projects.get(2).getMechines().add(0);
        projects.get(2).getMechines().add(1);
        projects.get(2).getMechines().add(2);


        //preprojects
        //pre[0] = 2;
        projects.get(0).getPreprojects().add(2);


        System.out.println("====================>Projects Info");
        for (Project project : projects) {
            System.out.println(project.toString());
        }
        System.out.println();
}
```

调度类使用线程模拟患者进入情况

```
@Override
    public void run() {

        // List<Suffer> suffers = new ArrayList<>();

        while (true) {
            // 锁
            synchronized (Schedule.class) {
                Suffer suffer = createSuffer();
                if (suffer == null) {
                    continue;
                }
                System.out.println("==============>Suffer Info");
                System.out.println(suffer.toString());
                // suffers.add(suffer);
                suffer.sortProjects();
                schedule(suffer);
                print(suffer);

```

```
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {

                }
            }
        }
    }
```

调度实现

```
/**
     * @param suffer
     */
    public static void schedule(Suffer suffer) {
        List<Integer> projects = suffer.getProjects();

        List<Project> allprojects = manager.getProjects();
        List<Mechine> allmechines = manager.getMechines();

        Integer time = suffer.getStime();

        /*
         * 1. 对于每个项目，找到可以做的机器
         * 2. 对于每个机器的项目队列，找到最小的最长时间
         *       如果当前时间减去项目在机器上的检查时间大于项目进队列的时间，出队列。
         */
        for (Integer project : projects) {
            List<Integer> mechines = allprojects.get(project).getMechines();
            int temp = 0, min_t = 0x3f;
            for (Integer _mechine : mechines) {
                //项目名称和执行时刻执行队列
                Mechine mechine = allmechines.get(_mechine);
                Map<Integer, Integer> projectQueue = mechine.getQueue();
                Iterator<Map.Entry<Integer, Integer>> it =
projectQueue.entrySet().iterator();
                int t = 0;
                while (it.hasNext()) {
                    Map.Entry<Integer, Integer> entry = it.next();
                    Integer key = entry.getKey();
                    int p = mechine.getProject2time().get(key);
                    Integer value = entry.getValue();
                    if (p <= time - value) {
                        it.remove();
                        continue;
                    }
                    t = Math.max(t, time - value);
                }
                if (min_t > t) {
                    temp = _mechine;
                    min_t = t;
                }
            }
            if (min_t == 0) {
```

```
                int tmin_t = 0x3f;
                for (Integer _mechine : mechines) {
                    Integer integer =
allmechines.get(_mechine).getProject2time().get(project);
                    if (integer == null) continue;
                    if (tmin_t > integer) {
                        tmin_t = integer;
                        temp = _mechine;
                    }
                }
            }
            int t = min_t == 0 ? time : time + min_t;
            Mechine mechine = allmechines.get(temp);
            mechine.getQueue().put(project, t);
            suffer.getPtimes().add(t);
            suffer.getMechines().add(temp);
            time = t + allmechines.get(temp).getProject2time().get(project);
        }
    }
```