**项目说明文档**

**科目：操作系统**

**报告名称：电梯调度**

**学号：2253406**

**姓名：李跃跃**

**完成日期：2024年5月7日**

# 项目目的

1. 通过控制电梯调度，实现操作系统调度过程；
2. 学习特定环境下的多线程编程方法；
3. 进一步加深对调度算法的理解，学会对调度算法的应用。

# 开发环境

## 操作系统平台

Windows 11+x64处理器

## 开发研究环境

编译器及版本：IntelliJ IDEA 2024.1.1

编程语言：Java

# 基本需求

## 基本任务

某一楼一共有20层，有5部互联的电梯。基于线程思想，编写一个电梯调度程序。

## 功能描述

1. 电梯上应有一些控制按键，如：数字键、关门键、开门键、报警键等；
2. 每层楼电梯门口有上、下行按键以及当前电梯位于几楼的数码显示；
3. 5部电梯互联，即当一个按钮按下去时，其他电梯的相应按钮同时点亮；
4. 电梯调度算法
5. 所有电梯初始时都处于第一层；
6. 每个电梯在没有相应请求的情况下，应该在原地保持不动；
7. 电梯调度算法自行设计。

# 实现的基本思路

## 对数据结构进行初始化

1. **brick**

类型为数组，大小初始化为100，该数组用于存放界面中每部电梯运行的通道，便于在电梯运行时进行可视化演示。存放的顺序为：从第20楼到第1楼依次存放通道；同一楼层的不同电梯按电梯编号从小到大依次存放。

1. **numberBrick**

类型为数组，大小初始化为100，该数组用于存放每部电梯内部的数字按键，便于在按键被按下时进行可视化演示。存放的顺序为：从第20楼到第1楼依次存放按键，即高楼层的按键索引小，低楼层的按键索引大；同一楼层的不同电梯按电梯编号从小到大依次存放按键。

1. **controlBrick**

类型为数组，大小初始化为40，该数组用于存放每层楼电梯门口的上、下行按键，便于在按键被按下时进行可视化演示。存放的顺序为：从第20楼到第1楼依次存放按键；同一楼层的“上”索引小于“下”。

1. **doorBrick**

类型为数组，大小初始化为10，该数组用于存放每部电梯内部的开、关门按键，便于对电梯线程以及开关门进程的并发进行可视化演示。存放的顺序为：按照电梯编号从小到大依次存放每一步电梯的“开”、“关”键，且同一部电梯的“开”索引小于“关”。

1. **panel**

类型为JPanel，用于容纳与电梯有关的按键以及文本框，便于构建用户界面。

1. **display**

类型为TextField数组，大小初始化为5，存放电梯上部显示运行楼层的GUI组件，用于在电梯运行时对电梯当前所在楼层进行可视化。存放的顺序为：按照电梯编号从小到大依次存放，且初始时均显示为1（因为初始时所有电梯都位于第一层）。

1. **controlPanel**

类型为JPanel，用于容纳电梯门口的控制按键，便于构建用户界面。

1. **table**

类型为int的二维数组，大小初始化为20×5，table[i][j]标识编号为j的电梯的第20-i楼是否有来自电梯内部数字按键的任务，如果有则值为1，否则值为0，初始化时全部为0。

1. **controlTable**

类型为int的二维数组，大小初始化为20×2，controlTable[i][0]标识第20-i楼的“上”按键是否被按下，若被按下则值为1，否则值为0；controlTable[i][1]标识第20-i楼的“下”按键是否被按下，若被按下则值为1，否则值为0。初始化时数组中的所有元素全部为0，表示当前没有任何按键被按下。

1. **upSignalTable**

类型为int的一维数组，大小初始化为20，upSignalTable[i]标识第i+1楼的“上”按键是否被按下，若被按下则值为1，否则值为0，初始化时全部为0，表示当前没有任何按键被按下。该数组和controlTable数组的区别在于：该数组用于在按键被按下时标识任务，并由任务分配监听器读取该数组中的值并根据五部电梯当前的状态进行任务分配，当任务结束后数组元素会由1修改为0；而controlTable数组主要用于读取界面中按键的状态并据此修改界面中按键的颜色。

1. **downSignalTable**

类型为int的一维数组，大小初始化为20，downSignalTable[i]标识第i+1楼的“下”按键是否被按下，若被按下则值为1，否则值为0，初始化时全部为0，表示当前没有任何按键被按下。

## 实现基本的功能函数

1. **startGame()**

用于开始时定义五个电梯线程，并启动任务分配监听器，设置任务分配监听器执行的延时为800ms，即每隔0.8s执行一次监听动作。

1. **drawBrick()**

用于刷新五个电梯的运行通道，若电梯未到达当前楼层，则通道块为黑，蓝色通道块标识电梯当前所在楼层。

1. **drawNumberBrick()**

用于刷新电梯内部的数字按键，若按键被按下，即table数组中对应电梯的对应楼层为1，则将按键颜色置为黄色，否则置为白色。

1. **drawControlBrick()**

用于刷新每个楼层电梯门口的控制按键，若上、下行按键被按下，即controlTable数组中对应楼层的对应位置为1，则将按键置为绿色，否则置为灰色。

1. **drawDoorBrick()**

用于刷新电梯内部开、关门按键，若“开”按键被按下，且当前电梯处于静止状态，则电梯线程中的变量door值为0，且“开”按键被置为红色，“关”按键被置为灰色，表示电梯门打开；若“关”按键被按下，即电梯线程中的door值为1，则相“关”按键被置为红色，“开”按键被置为灰色，表示电梯门关闭。

1. **workStatus()**

接受一个int类型参数liftNumber并检查编号为liftNumber的电梯是否有来自电梯内部按键的任务，即遍历table[i][liftNumber](1<=i<=20)，若至少有一个i满足table[i][liftNumber]=1，则返回true，表示该电梯至少有一个来自内部数字按键的任务；否则返回false。

## 定义电梯进程的监听器

### 成员变量

该监听器类中的变量只有一个liftNumber，数据类型为int，用于将电梯线程的监听器与电梯进行绑定。

### 方法

1. **构造方法**

该方法接受一个类型为int的参数n，并用该参数对成员变量liftNumber进行初始化，将新定义的电梯线程监听器与相应的电梯进行绑定。

1. **actionPerformed()**

该方法中实现了电梯的运行，具体实现过程见第6部分核心算法。

## 定义任务分配的监听器

该监听器类中的方法只有一个actionPerformed()，用于在接收到电梯门口按键的请求后进行任务分配，具体实现过程见第6部分核心算法。

## 定义电梯门口上、下行按键的监听器

以上行按键的监听器UpListener的实现为例进行思路说明，下行按键的监听器DownListener的实现同理。

### 成员变量

该监听器类中的变量只有一个floor，数据类型为int，用于将电梯门口上行按键与所在楼层进行对应。

### 方法

1. **构造方法**

该方法接受一个类型为int的参数f，并用该参数对成员变量floor进行初始化，将按键与所在的楼层进行对应。

1. **actionPerformed()**

该方法定义了当上行按键被按下时的动作：首先修改controlTable[20-floor][0]为1，代表上行按键被按下，然后将upSignalTable[floor-1]修改为1，便于任务分配监听器读取该数组并将任务分配给电梯线程，最后调用drawControlBrick()函数刷新电梯门口的上、下行控制按键在界面上进行可视化演示。

上、下行按键监听器的和对应按键的绑定是在定义按键时进行的，随后再将按键加入controlBrick数组进行统一管理。

## 定义电梯内部数字键的监听器

### 成员变量

包括int型变量liftNumber和int型变量floor，前者用于与电梯对应，后者用于与楼层对应。

### 方法

1. **构造方法**

该方法接受两个类型为int的参数l和f，分别用于初始化成员变量liftNumber和floor。

1. **actionPerformed()**

该方法定义了当数字按键被按下时的动作：将table[20-floor][liftNumber]修改为1，代表编号为liftNumber的电梯接受到来自第floor层的任务，同时若电梯之前为静止状态，则还应检查liftNumber号电梯当前所在楼层currentFloor并据此修改电梯状态，若currentFloor<floor，则电梯此后应该上行，修改lift[liftNumber].status=1，否则电梯此后应该下行，修改lift[liftNumber].status=2。最后调用drawNumberBrick()函数刷新电梯内数字按键的颜色在界面上进行可视化演示。

数字按键监听器和对应按键的绑定是在定义按键时进行的，随后再将按键加入numberBrick数组进行统一管理。

## 定义电梯内开、关门按键的监听器

以开门按键的监听器OpenListener的实现为例进行思路说明，关门按键的监听器CloseListener的实现同理。

### 成员变量

该监听器类中的变量只有一个liftNumber，数据类型为int，用于将电梯内的开、关门按键与所在电梯进行对应。

### 方法

1. **构造方法**

该方法接受一个类型为int的参数f，用于初始化成员变量liftNumber。

1. **actionPerformed()**

该方法定义了当开门按键被按下时的动作：首先检查lift[liftNumber].status，若为0，则代表电梯当前处于静止状态，可以开门，则修改lift[liftNumber].door为0，代表门打开；然后调用DrawDoorBrick()函数刷新电梯中的开、关门按键在界面上进行可视化演示。

开、关门按键监听器和对应按键的绑定是在定义按键时进行的，随后再将按键加入DoorBrick数组进行统一管理。

## 定义电梯线程类

### 成员变量

1. **liftNumber**

记录电梯编号，取值范围为0~4对应5部电梯。

1. **currentFloor**

记录电梯当前所在楼层，取值范围为1~20。

1. **Status**

记录电梯状态，值为0代表电梯处于静止状态，值为1代表电梯正在上行，值为2代表电梯正在下行。

1. **Door**

记录电梯门的状态，值为0代表门处于打开状态，值为1代表门处于关闭状态。

1. **aim**

记录电梯当前正要去往的目标楼层，即外部上、下行按键被按下后分配给电梯的任务。

### 方法

1. **构造方法**

该方法接受一个类型为int的参数l，用于初始化成员变量liftNumber。

1. **run()**

该方法定义电梯线程被创建后的动作：首先将本电梯线程与一个电梯线程监听器liftThreadListener进行绑定，并设置延时为800ms，即每隔0.8s执行一次监听动作，然后启动该定时器。

# 数据结构

数组，用于保存按键、按键状态以及记录任务。

# 核心算法

## 电梯运行

首先检查电梯在当前的楼层是否有任务，即table[currentFloor][liftNumber]是否等于1，如果等于1，则代表编号为liftNumber的电梯内该楼层的按键被按下，将table[currentFloor][liftNumber]置为0，并调用drawNumberBrick()函数刷新电梯内按键的颜色。

然后读取电梯的状态，如果status不为0，即电梯当前处于运行状态，但是电梯当前已经到达目标楼层，即currentFloor=aim，则检查电梯当前运行方向与目标楼层上、下行按键的方向是否一致。

若电梯运行方向与目标楼层上、下行按键方向相反，即：若电梯为下行且该楼层的按键方向为上行，则该楼层的按键任务完成，将controlTable[currentFloor][0]修改为0，upSignalTable[currentFloor]修改为0，并调用drawControlBrick()函数刷新电梯门口上、下行按键的颜色；若电梯为上行且该楼层的按键方向为下行，则该楼层的按键任务完成，将controlTable[currentFloor][1]修改为0，downSignalTable[currentFloor]修改为0，并调用drawControlBrick()函数刷新电梯门口上、下行按键的颜色。

若电梯运行方向与目标楼层上、下行按键方向相同，即：若电梯为上行且楼层的按键方向为上行，则该楼层的按键任务完成，将controlTable[currentFloor][0]修改为0，upSignalTable[currentFloor]修改为0，并调用drawControlBrick()函数刷新电梯门口上、下行按键的颜色；若电梯为下行且该楼层的按键方向为向下，则该楼层的按键任务完成，将controlTable[currentFloor][1]修改为0，downSignalTable[currentFloor]修改为0，并调用drawControlBrick()函数刷新电梯门口上、下行按键的颜色。

若电梯运行方向与目标楼层上、下按键方向相同，则还需要检查电梯是否需要沿原运行方向继续运行，即：若电梯原本向上运行，则检查该电梯之上是否还有楼层有来自电梯内部数字按键的任务（如果外部按键有任务，则该任务在任务调度器中被保存在电梯的aim变量中），即检查table[i][liftNumber](currentFloor<i<=20)是否为1，只要有至少一个i满足条件，电梯就还需要向上运行。如果该电梯之上已经没有楼层有来自电梯内部数字按键的任务，则检查该电梯是否已经完成所有外部按键的任务，即currentFloor是否大于等于aim，若是，则外部按键的任务也完成，修改电梯的状态变量status为0，表示电梯在原地静止不动，然后将电梯内部的按键全部刷新为0，并调用drawNumberBrick()函数进行可视化演示。

若电梯原本向下运行，则检查该电梯之下是否还有楼层有来自电梯内部数字按键的任务（如果外部按键有任务，则该任务在任务调度器中被保存在电梯的aim变量中），即检查table[i][liftNumber](1<=i<currentFloor)是否为1，只要有至少一个i满足条件，电梯就还需要向下运行。如果该电梯之下已经没有楼层有来自电梯内部数字按键的任务，则检查该电梯是否已经完成所有外部按键的任务，即currentFloor是否大于等于aim，若是，则外部按键的任务也完成，修改电梯的状态变量status为0，表示电梯在原地静止不动，然后将电梯内部的按键全部刷新为0，并调用drawNumberBrick()函数进行可视化演示。

最后根据电梯当前的状态修改currentFloor变量的值，即若status=1，则电梯继续向上运行，currentFloor+1；若status=2，则电梯继续向下运行，currentFloor-1，最后调用drawBrick()函数刷新电梯运行通道中电梯的位置进行可视化演示，并修改电梯上部显示楼层的GUI组件中的内容。

## 任务分配

遍历upSignalTable数组检查是否有来自外部上行按键的请求，若有，则为该任务分配一部合适的电梯，以下用floor代指上行按键被按下的楼层。用direction记录最终被分配任务的电梯要运行的方向，liftNumber记录被选中的电梯的编号，distance记录被选中的电梯当前的位置和任务楼层floor之间的距离

遍历lift数组，检查每一部电梯的运行状态，对于上行任务来讲，该任务可以分配给四种运行状态的电梯：当前位于floor之下且运行方向为向上的电梯、当前位于floor之下且静止的电梯、当前位于floor之上且静止的电梯以及当前位于floor之上且运行方向为向下但已经完成所有内部数字按键任务的电梯。若该电梯当前位于该楼层之下，即currentFloor<floor，且运行方向为向上，则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为上行、liftNumber、distance变量为该电梯所对应的值；若该电梯当前位于该楼层之下，即currentFloor<floor，且状态为静止，即status=0，则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为上行、liftNumber、distance变量为该电梯所对应的值；若该电梯当前位于该楼层之上，即curretnFloor>floor，且状态为静止，即status=0，则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为下行、liftNumber、distance变量为该电梯所对应的值；若该电梯当前位于该楼层之上，且运行方向为向下，但电梯内部数字按键的任务已完成（说明该电梯当前有下方楼层的外部按键任务），则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为下行、liftNumber、distance变量为该电梯所对应的值。最后保存在liftNumber中的值即为满足以上四种运行状态之一，且距离floor距离最近的电梯编号，修改lift[liftNumber]的status值为direction，aim值为floor，完成任务分配。

遍历downSignalTable数组检查是否有来自外部下行按键的请求，若有，则为该任务分配一部合适的电梯，以下用floor代指下行按键被按下的楼层。用direction记录最终被分配任务的电梯要运行的方向，liftNumber记录被选中的电梯的编号，distance记录被选中的电梯当前的位置和任务楼层floor之间的距离

遍历lift数组，检查每一部电梯的运行状态，对于下行任务来讲，该任务可以分配给四种运行状态的电梯：当前位于floor之上且运行方向为向下的电梯、当前位于floor之上且静止的电梯、当前位于floor之下且静止的电梯以及当前位于floor之下且运行方向为向上但已经完成所有内部数字按键任务的电梯。若该电梯当前位于该楼层之上，即currentFloor>floor，且运行方向为向下，则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为下行、liftNumber、distance变量为该电梯所对应的值；若该电梯当前位于该楼层之上，即currentFloor>floor，且状态为静止，即status=0，则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为下行、liftNumber、distance变量为该电梯所对应的值；若该电梯当前位于该楼层之下，即curretnFloor<floor，且状态为静止，即status=0，则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为上行、liftNumber、distance变量为该电梯所对应的值；若该电梯当前位于该楼层之下，且运行方向为向上，但电梯内部数字按键的任务已完成（说明该电梯当前有下方楼层的外部按键任务），则比较该电梯当前所处楼层与floor的距离是否小于distance，若是，则更新direction为上行、liftNumber、distance变量为该电梯所对应的值。最后保存在liftNumber中的值即为满足以上四种运行状态之一，且距离floor距离最近的电梯编号，修改lift[liftNumber]的status值为direction，aim值为floor，完成任务分配。

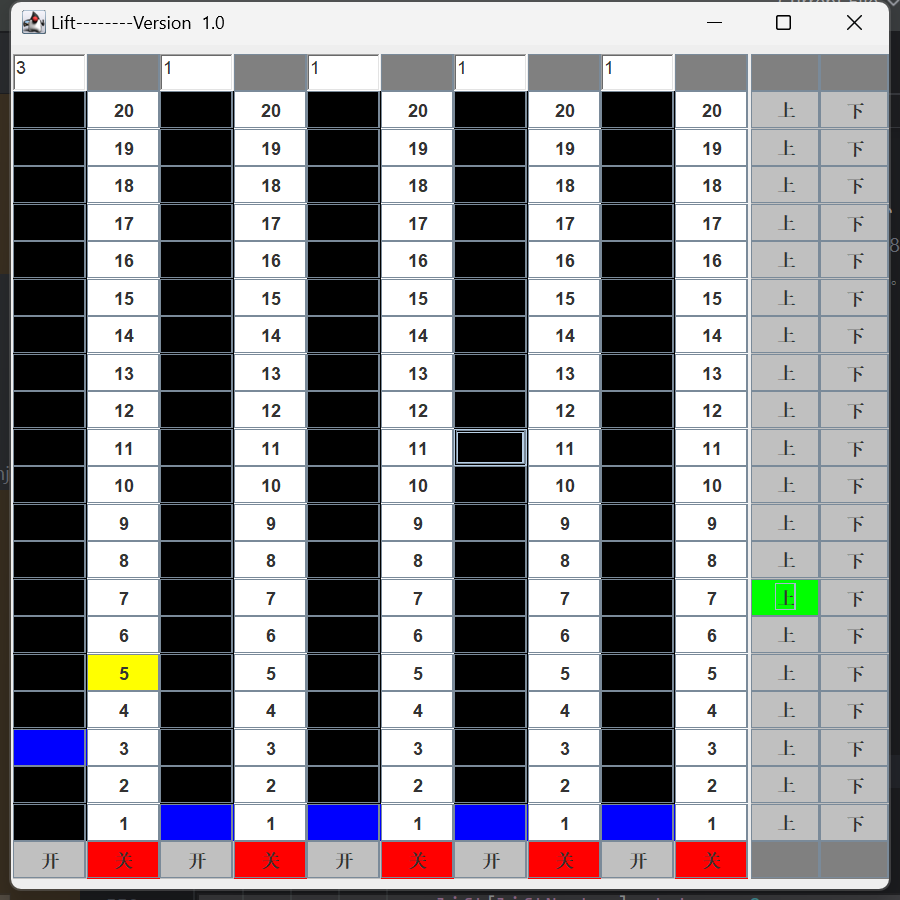
## 进程同步

电梯门的开关和电梯运行之间存在进程同步，进行进程同步处理的地方如下：

1. 关门按键的点击无限制，但在开门按键被按下时，必须先检查电梯的状态：若电梯当前处于静止状态，则将lift[liftNumber].door的值修改为1，否则不进行任何操作。这样可以保证当电梯处于运行状态是门是无法打开的。
2. 当电梯的状态由静止改为上行或下行时，必须首先修改lift[liftNumber].door的值为0。这样可以保证电梯运行之前先进行的关门的操作。

# 操作说明

显示电梯当前所处楼层



蓝色代表电梯当前所处楼层

上、下行按键被按下后变为绿色

红色代表电梯门当前状态

数字按键被按下后变为黄色

电梯内数字按键

电梯门口上、下行按键

# 心得体会

## 总结

通过本次对电梯调度的模拟，我学习到了java语言中用户交互界面的设计和布局方法，同时还对调度算法有了更加深刻的理解与认识；在修补程序漏洞的过程中，我对程序调试和debug的过程与方法也有了更深的体会。

## 反思

由于缺乏前端制作的经验，我在项目开始初期遇到了很多困难，比如尝试用在Qt软件中使用C++语言制作用户交互界面，但是由于依赖库缺失的问题得不到解决，最后无奈只能参考之前学长的开发经验使用java语言。并且由于这是我首次使用java语言开发项目、制作用户交互界面，因此在界面设计部分参考了往届学长的代码，但这次项目也给我积累了经验，相信下一次我一定能够独立制作用户交互界面。

同时我在调试程序的过程中遇到了很多的漏洞，虽然已经出现的漏洞我都已经修复了，但仍然有可能有考虑不周到的地方，如果后续想要对代码进行优化和改善可以从这方面进行考虑。