# Elevator Simulator 电梯模拟器

## 操作系统课程作业

同济大学软件学院

1652795 王陆洋

目录

# 项目概述

## 项目简介

本项目是一个大型建筑电梯模拟器系统，项目基于Java语言编写，使用了JavaFX框架以实现图形化用户界面（GUI）。

## 项目目的

本项目的编写目的在于通过对电梯系统的运行状况的调度实现对一个多线程系统的模拟，进而提升多线程的理解和运用。

## 项目主要运用到的技术

在项目整体的编写过程中，使用了Java语言的多种特性，同时结合了最新的Java标准（编写平台基于Oracle于2018年3月发布的Java 10[1]，要求运行平台至少具有Java 1.8以上的运行环境）。同时，在GUI编写工作进行的过程中采用了JavaFX模块+JavaFX Scene Builder的可视化编程技术，有效地减轻了GUI编制工作的难度，同时优化了项目的效果。

在对电梯系统的模拟中，采用了MVC（Model-View-Controller）模型，将View模型交给JavaFX制作，建筑的模拟则封装在Model中，同时为了准确，将每一个电梯模型封装为一个线程，通过一个独立的Controller线程加以控制，形成运行性能良好、模拟效果优秀的项目模型。

# 项目分析

## 类分析

### cn.leonwong.ElevatorSimulator.Model 包

cn.leonwong.ElevatorSimulator.Model包（下简称Model包）包含了所有建筑的相关信息构成的模型，包括Elevator（电梯）类、Building（建筑）类、Passenger（乘客）类和Message（消息）类

#### Building类

Building类的成员：

公有：  
/// Note for each elevator in this building  
public Vector<Elevator> elevatorList;  
/// Note for persons waiting for elevators in each level  
public Vector< Vector<Passenger> > levelList;  
/// A message center noting for events  
public Vector<Message> messageCenter;

*/\*\*  
 \* Creates a building  
 \** ***@param*** *levs \#levels in ths building  
 \** ***@param*** *elevs \#elevators in this building  
 \** ***@param*** *maxPass max \# of passengers an elevator can contain  
 \*/*public Building(int levs, int elevs, int maxPass)；

*/\*\*  
 \* getter for numbers of elevators  
 \** ***@return*** *numbers of elevators in this building  
 \*/*public int getElevators()；

*/\*\*  
 \* getter for numbers of levels  
 \** ***@return*** *numbers of levels in this building  
 \*/*public int getLevels();

私有成员：  
/// \#Levels of this building  
private int levels;  
/// \#Elevators of this building  
private int elevators;  
/// Note for each elevator in this building  
/// A lock used to help lock levelList  
private ReentrantLock lock;

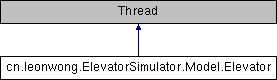
Building类的作用：

Building类模拟了一栋建筑，保存了建筑整体的有关信息（楼层数、电梯数、每一层楼的状况等等），并且利用了Java中Vector容器的线程安全性和ReentrantLock重入锁来保护楼层数据的安全。

#### Elevator类

继承关系：

Elevator类继承了Thread类。



Elevator类的成员：

公有成员：

/// time interval for going to another floor  
public static final int *FLOOR\_INTERVAL* = 500;  
/// time interval for a passenger entering/leaving  
public static final int *PASS\_INTERVAL* = 200;  
  
*/\*\*  
 \* A class contains to directions in order to indicate direction  
 \*/*public static final class Direction;  
*/\*\*  
 \* Start this Thread  
 \*/*@Override  
public void start();

*/\*\*  
 \* Run function  
 \*/*@Override  
public void run();  
@Override  
public String toString();

*/\*\*  
 \* create a new elevator  
 \** ***@param*** *name the in dex of this elevator  
 \** ***@param*** *max the max level of the building  
 \** ***@param*** *maxPass the capacity of this elevator  
 \** ***@param*** *levs a list for passengers in each level  
 \** ***@param*** *mess message center  
 \** ***@param*** *l a reentrantlock to lock level list  
 \*/*public Elevator(int name, int max, int maxPass, Vector< Vector<Passenger> > levs, Vector<Message> mess, ReentrantLock l);

*/\*\*  
 \* decide whether this elevator is full  
 \** ***@return*** *true if this elevator is full, false otherwise  
 \*/*public boolean isFull();

*/\*\*  
 \* decide whether this elevator is empty  
 \** ***@return*** *true if this elevator is empty, false otherwise  
 \*/*public boolean isEmpty();

*/\*\*  
 \* decide whether this elevator is idle  
 \** ***@return*** *true if this elevator is idle, false otherwise  
 \*/*public boolean isIdle();

*/\*\*  
 \* add a destination for this elevator  
 \** ***@param*** *dest the destination level to be added  
 \*/*public void addDestination(int dest);

*/\*\*  
 \* getter for the current level this elevator is at  
 \** ***@return*** *the current level this elevator is at  
 \*/*public int getLevel();

*/\*\*  
 \* getter for the current direction of this elevator  
 \** ***@return*** *the current direction of this elevator  
 \*/*public int getDirection();

*/\*\*  
 \* getter for the biggest level this elevator is heading for  
 \** ***@return*** *the biggest number of level in destination list  
 \*/*public int getMaxDestination();

*/\*\*  
 \* getter for the lowest level this elevator is heading for  
 \** ***@return*** *the smallest number of level in destination list  
 \*/*public int getMinDestination();

*/\*\*  
 \* getter for the numbers of passengers in this elevator  
 \** ***@return*** *the number of passengers in this elevator  
 \*/*public int getPassengers();

*/\*\*  
 \* directly move this elevator to a level  
 \** ***@param*** *l the destination level  
 \*/*public void setLevel(int l);

*/\*\*  
 \* used to stop this thread  
 \*/*public void stopThread();

*/\*\*  
 \* getter for a list of destinations of this elevator in a String  
 \** ***@return*** *the String of list of destinations  
 \*/*public String getDestinations();

*/\*\*  
 \* getter for the destination list's size  
 \** ***@return*** *this size of this destination list  
 \*/*public int getDestinationSize();

私有成员：

/// Note for the direction this elevator is heading  
private int direction;  
/// When changing direction is needed, add another flag  
private int nextDire;  
/// Thread for an elevator  
private Thread t;  
/// Note for the index of this elevator  
private int index;  
/// Note for person waiting for elevators in each level  
private Vector< Vector<Passenger> > levelList;  
/// Note for max level available  
private int maxLevel;  
/// Note for maximum numbers of passengers available in this elevator  
private int maxPassenger;  
/// Note for which level this elevator is at  
private int level;  
/// Note for messageCenter  
private Vector<Message> messageCenter;  
/// Note for each passenger in this elevator  
private Vector<Passenger> passengerList;  
/// Note for each level that this elevator should stop and open  
private TreeSet<Integer> destinations;  
/// Note for the lock used to lock levelList  
private ReentrantLock lock;

*/\*\*  
 \* add one passenger into this elevator  
 \** ***@param*** *lev where the passenger entered the elevator  
 \** ***@param*** *pass an object refering to the passenger  
 \** ***@return*** *true if the passenger entered this elevator successfully, false otherwise  
 \*/*private synchronized boolean passengerEnterElevetor(int lev, Passenger pass)；

*/\*\*  
 \* remove a passenger from this elevator  
 \** ***@param*** *lev the level that the passenger leaves at  
 \** ***@param*** *pass an object refering to the passenger  
 \** ***@return*** *true if the passenger removed successfully, false otherwise  
 \*/*private synchronized boolean passengerLeaveElevator(int lev, Passenger pass)；

*/\*\*  
 \* decide what direction this elevator should go after a level's all work done  
 \*/*private synchronized void decideDirection()；

*/\*\*  
 \* use this to ensure that every passenger's destination is in this destination list  
 \*/*private void decideDestinations()；

/// decide whether this thread has been stopped  
private boolean stop;

Elevator类的作用：

Elevator类采用线程机制，每一个线程模拟一台电梯的运行情况。

#### Message类：

Message类的成员：

公有成员：

/// used to indicate this message is describing a passenger leaving an elevator  
public static final int *passengerLeaveElevator* = 0;  
/// used to indicate this message is describing a passenger entering an elevator  
public static final int *passengerEnterElevator* = 1;  
/// used to indicate this message is describing an elevator has moved to another floor  
public static final int *elevatorChangeFloor* = 2;  
/// used to indicate this message is describing an elevator is idle  
public static final int *elevatorIsIdle* = 3;  
  
/// to denote which kind of message this one is  
public int mode;  
/// to denote the elevator that this message took place in  
public int destElevator;  
/// to denote the level that this message took place at  
public int destLevel;  
/// to denote the passenger related to this message  
public Passenger pass;  
  
*/\*\*  
 \* to build a new messafe  
 \** ***@param*** *modeOfMessage the kind of message  
 \** ***@param*** *elevatorSender the elevator this message took place in  
 \** ***@param*** *levelHappened the level this message took place at  
 \** ***@param*** *passenger the related passenger  
 \*/*public Message(int modeOfMessage, int elevatorSender, int levelHappened, Passenger passenger);

私有成员：

Message类没有私有成员。

Message类的作用：

每个Elevator线程触发一个电梯事件（进入电梯、离开电梯、电梯空闲、电梯移动）的时候都会构造一个新的Message类累计在MessageCenter里面，Controller类通过访问MessageCenter来进行界面的绘制，显示正确的动画。

#### Passenger类

Passenger类的成员：

公有成员：

/// the destination of this passenger  
public int destination;  
*/\*\*  
 \* create a new passenger heading for some floor  
 \** ***@param*** *dest the destination of this passenger  
 \*/*public Passenger(int dest);

私有成员：

Passenger类没有私有成员。

Passenger类的作用：

Passenger类模拟了一个前往某一层的乘客，这个乘客出现在某一层的层列表的时候会根据这个乘客的目的地的方向（向上还是向下）触发这一层的方向指示灯，乘客在某个电梯的时候会根据这个乘客的目的地楼层激活电梯内部的目的地列表。

### cn.leonwong.ElevatorSimultor中的其他类

#### Controller类：