Project 7

PROJECT 7

This project was a two-week and two-part project. The first week consisted of creating a functional PriorityQueue class as well as implementing a Passenger and a PassengerGroup class. The second week then consisted of using Elevator simulation code to run an Elevator simulation. Building off that, Project 7b also included adding other rules/conditionals to speed up the elevator simulation. Overall, the goal of this project was the create a functional Elevator simulation using the code provided and to speed up the simulation any way you possibly can.

Task 1 - 7a. In this task, we created a Passenger class that extends Cell. It required the following 5 fields:

- private boolean active whether the Passenger is active in the simulation.
- private boolean onboard whether the Passenger is on an Elevator.
- private final int startFloor the starting floor of the passenger.
- private final int targetFloor the target floor of the passenger.
- private int waitTime the time the Passenger has been waiting.

In addition, the Passenger class required a number of new methods like getTargetFloor(), getStartFloor(), getWaitTime(), isActive(), isOnboard(), and many more. In this first task, I implemented all of these methods and correctly created my Passenger class.

Task 2 - 7a. Here, I was asked to created two nested classes within Passenger called MaxFloor and MinFloor that use Comparator and implement one method each. Here is the snippet of code from this task:

```
    public static class MaxFloor implements Comparator<Passenger>{

        public int compare( Passenger A, Passenger B ){
            if(A.getTargetFloor() > B.getTargetFloor()){
                return 1;
            }
            else if(A.getTargetFloor() == B.getTargetFloor()){
                return 0;
            }
            else{
                return -1;
            }
        }
    }
    public static class MinFloor implements Comparator<Passenger>{

    public int compare( Passenger A, Passenger B ){
            if(A.getTargetFloor() > B.getTargetFloor()){
                return -1;
            }
            else if(A.getTargetFloor() == B.getTargetFloor()){
                return 0;
            }
            else{
                return 1;
            }
        }
    }
```

Task 3 - 7a. Next, i was asked to create my PassengerGroup class with 4 specific methods. Here is the code for my PassengerGroup class:

```
public class PassengerGroup extends MyPriorityQueue<Passenger> implements Iterable<Passenger> {

    public PassengerGroup(int max){
        super(max, new Passenger.MaxFloor());
    }

    public void useMaxFloors(){
        setComparator(new Passenger.MaxFloor());
    }

    public void useMinFloors(){
        setComparator(new Passenger.MinFloor());
    }

    public String toString(){
        return ("Passengers: " + this.getSize());
    }

    /*public static void main(String[] args){
```

Task 4 - 7a. Finally, we were told to download the Elevator code bundle and run the ElevatorSimulation and make sure it works. Also, we were instructed to observe the patterns and how the elevators worked.

Task 5 - 7a. This task prepared us for the second half of the project and asked us to think about ways to make the ElevatorSimulation strategy more efficient.

Task 1 - 7b. For this task, we had to set the ElevatorSimulation to have 8 elevators, 25 floors, and a 8 capacity passengers. To do this, I simply edited one line in the main code of ElevatorSimulation to say:

ElevatorSimulation sim = new ElevatorSimulation(8, 25, 8);

Task 2 - 7b. This was the meat of Project 7b. This task was basically to implement a few changes in the code to make the simulation run more quickly. I worked with CS231 students Steve Parrot and Brendan Doyle on this task. First, I wanted to make the simulation work more like a normal elevator situation would. So, the first change I made was in the emptyRule method. I wanted to add an else conditional that would check for waiting passengers and then change the direction of the Empty Elevator if it located a waiting passenger in the opposite direction. Instead of

just scanning up and down until it finds a waiting passenger, the empty Elevator will now switch directions if there is a waiting passenger above or below it. To do this, I take the difference between the StartFloor of the waiting passenger and the CurrentFloor of the Elevator and if the difference is less than 0, I move the Elevator down and if the difference is greater than 0, I move the Elevator up. I also removed a couple this.openDoors() calls at the top and the bottom of the Bank to speed up the simulation because if the Elevator is empty, there is no point in opening the doors at the top/bottom. Here is the snippet of my added else conditional in the emptyRule method:

```
    }
    //Task 2 Work
    //worked with Steve Parrot and Brendan Doyle
    else{
        //checks to see if someone is waiting
        if (nextWaiting != null){
        //if someone is waiting
            //then, take the start floor for the person waiting
            //and subtract the current floor and get difference
            int difference = nextWaiting.getStartFloor() - this.getCurrentFloor();
            //if difference is < 0 move down
            if(difference < 0){
                this.direction = Direction.DOWN;
            }
            //move up
            else{this.direction = Direction.UP;
            }
        }
    }
```

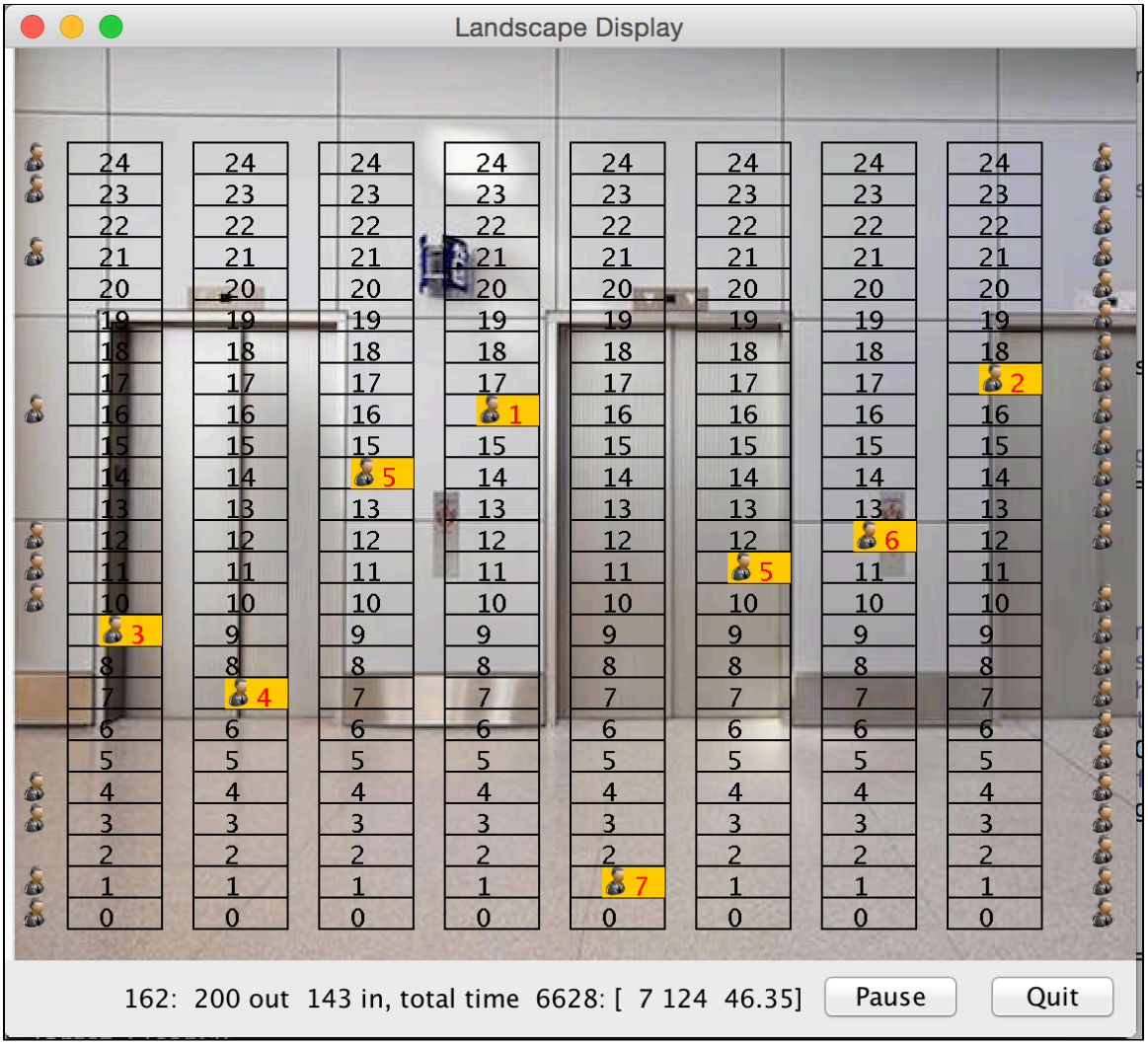
Secondly, I also made a change to the nonEmptyRule method. I wanted to make some changes to adding passengers when the Elevator was not empty. I wrote an else conditional in the nonEmptyRule method that checked 3 things: to see if anyone is waiting(so the floorQueue is not empty), to see if the Elevator wasn't already full of passengers(! this.passengers.isFull()), and finally, to see if the number of passengers in the elevator is less than 6(this.passengers.getSize() < 6). The reason I checked to see if the number of passengers was less than 6 is because I only want to open the doors for new passengers if there are 3 or more spots open in the Elevator. If there is only one of two spots open, bypass the waiting passenger and do NOT open the doors. This is more efficient and will speed up the Elevator Simulation. Here is the code for this change:

```


// Task 2 worked with Steven Parrot and Brendan Doyle
// Check to see if anyone is waiting
// Only add passengers if there is more than one open space
// But, do NOT stop for just one or two person
else if(!this.bank.getFloorQueue(this).isEmpty() && !this.passengers.isFull()
        //only add people if there is less than 6 people on the elevator
        && this.passengers.getSize() < 6){
    this.openDoors();
}

// move in the direction we are headed
```

Task 3 - 7b. Finally, we were told to download a High Volume Data set from the Project site and to run our simulation with the given data. We were told to check for two things: 1) that the simulation runs well and finishes with 200out and 200in & 2) that the average wait time is quicker and less than the original value of 71. When I run my simulation I get between(55-60) around 56.41 for the average wait time. Much quicker! Also, here is a snapshot of my simulation in action:



Extension 1. My first extension was to display the current number of passengers in the Elevator at any given time on the actual elevator. To do this, I edited the Elevator Draw method. Here is a code snippet for Extension 1:

```
//EXTENSION 1
//Writes in the number of passengers on each elevator at a given time
g.setColor(Color.red);
g.drawString(Integer.toString(this.passengers.getSize()),(int)(x0 + (x+2)*scale),(ypos+15));
```

Extension 2. For this extension, I created a more interesting visual representation of the simulation. Instead of circles for passengers, I created little BusinessMen icons. To do this, I had to update both the draw method of the Passenger class and also the Elevator for once the passenger was inside the Elevator. Both code snippets look similar but with different location variables. In addition, I also added a background to my simulation! This code was changed inside the LandscapeDisplay class. Here is a snippet for Extension 2 inside the Elevator class:

```
//EXTENSION 2
Image img = Toolkit.getDefaultToolkit().getImage("BusinessMan.png");
g.drawImage(img,(int) (x0 + (x + 1) * scale), ypos, scale-1, scale-1, null);
```

What I learned. The goal of this project was to implement and understand PriorityQueue. I can say I definitely did both. This project made me read code and think critically than it did actually make me code. I definitely learned how to understand code better and to visualize more efficient ways to complete things. Overall, I learned about PriorityQueues, Elevator Simulations, and how to make methods more efficient.

Who helped me. I received a lot of help from Bruce Maxwell. My simulation was really screwed up and he helped me locate the problem in my PriorityQueue. Throughout most of the Project, I worked with my fellow CS231 classmates Steven Parrott and Brendan Doyle. Also, I got help

from CP Majgaard.