Maddy Placik CS231 Project3

Summary

Our goal for this project was to simulate a parking garage using a "stack" data structure. By using stacks, we were able to simulate parking lanes in a parking garage and retrieve specific cars within these lanes. By using push and pop methods (explained in detail later), we had the ability to remove all cars in front of a given car and place them back into the lane in their original order once the specified car had been removed. In order to successfully execute this project we used a Car Class, CarStack Class, Parking Garage Class, ParkingGarageDisplay class, and a Parking Simulation Class.

Car Class

This class was created to represent car objects. Each car object was associated with a color and specific time to leave the parking lot. This class contained two constructor methods, one of which did not include the capability to specify the color of the car.

CarStack Class

This class represented the lanes in which car objects could park. This class used a "stack" data structure and implemented methods including push, pop, elementAt, size and isEmpty. Each of these methods aided in creating and describing the state of our "parking lanes." Our push method within this class allowed us to add car objects to a specified "CarStack." If a car object needed to be added to a CarStack with insufficient size, the CarStack would be doubled in length.

```java
//pushes new car object onto stack
public void push( Car new_item ){
    /*if the number of car elements exceeds
    the length of the array, double array length*/
    if ( this.topCar >= this.parkingLane.length ){
        Car[] dblElement = new Car[this.parkingLane.length*2];
        for ( int i = 0 ; i < this.parkingLane.length ; i++){
            dblElement[i] = this.parkingLane[i];
        }
        this.parkingLane = dblElement;
    }
    //pushes new item onto stack
    this.parkingLane[topCar] = new_item;
    this.topCar++;
}
```

The pop method within this class was defined by returning and removing the top car element in the given stack.

ParkingGarage Class

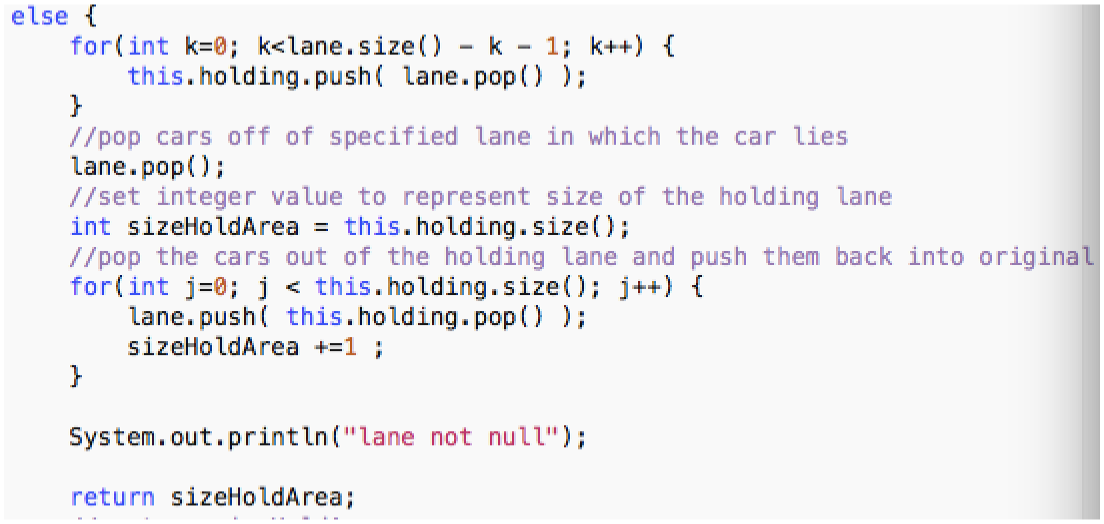
Within this class we were able to park cars in specific lanes as well as retrieve specific car objects. The parkCar method looped through the number of parking lanes and checked whether or not each of these lane were full. If the size of a given lane was less than its maximum size, the method allowed a car to be parked in this specific lane and returned "true" (indicating a car was able to park). For lanes without vacant spots, the method returned false, indicating a car had failed to park.

```java
//if lane is avail. car will park and return true
public boolean parkCar( Car car ){
    for ( int i = 0 ; i < this.getNumLanes() ; i++ ){
       if ( this.lanes[i].size() < this.getMaxLaneSize() ){
           lanes[i].push(car);
           return true;
       }
    }
    //returns false if car is unable to park
    return false;
}
```

This class also contained a "retrieveCar" method which gave us the ability to remove specific cars from a given lane without altering the order of the other cars in the lane. This method used a for loop to loop through the array of CarStacks ("lanes") and through each of the individual CarStacks as well. While looping through each individual stack, we checked to see if we reached the specified car object. If we did reach this specified car, we marked its specific lane as "lane" and its spot in that lane as "carIndex".

If the car we wanted to retrieve was not the first car in the lane, we would need to put the cars in front of it into a holding lane in order to retrieve our wanted car.

The code below includes my extension, which allowed me to return the number of cars in the holding area. In order to to this, I created an integer variable named "sizeHoldArea" which was assigned to the size of the given holding area. I used this variable as a counter and added 1 to its value every time a car was popped from the holding area. I then used the retrieveCar method to return this integer value.



```
else {
    for(int k=0; k<lane.size() - k - 1; k++) {
        this.holding.push( lane.pop() );
    }
    //pop cars off of specified lane in which the car lies
    lane.pop();
    //set integer value to represent size of the holding lane
    int sizeHoldArea = this.holding.size();
    //pop the cars out of the holding lane and push them back into original
    for(int j=0; j < this.holding.size(); j++) {
        lane.push( this.holding.pop() );
        sizeHoldArea +=1 ;
    }

    System.out.println("lane not null");

    return sizeHoldArea;
```

ParkingSimulation Class

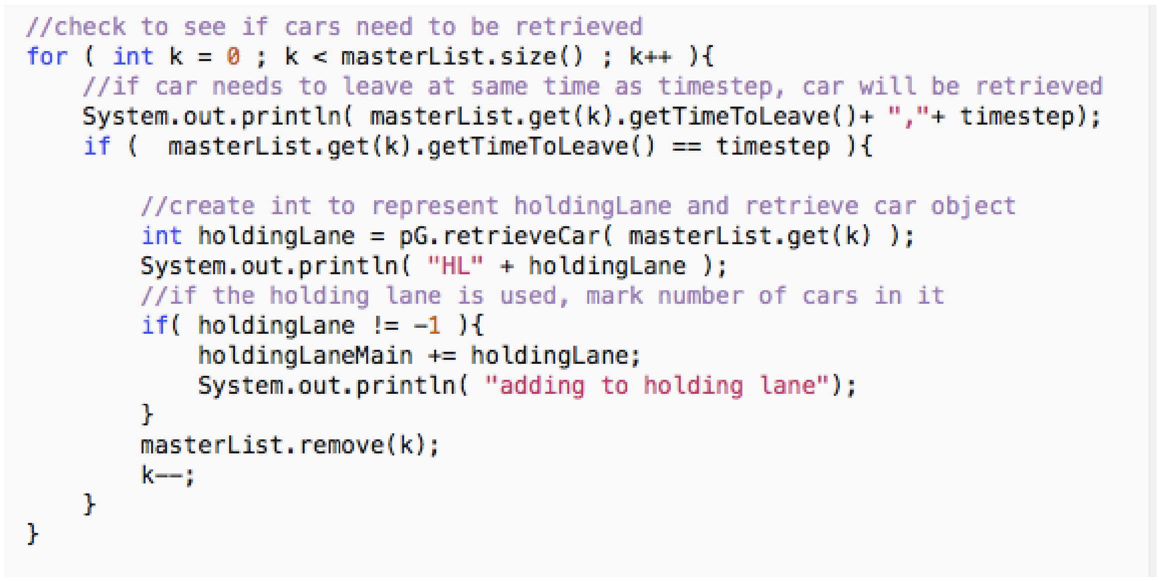
This class included a single method "run" which specified a car's probabilityToPark and another value, timestep. We began this method by looping through the specified timestep integer value. We then looped twice to allow 0, 1, or 2 cars to park at once. We randomly chose a number between 0.0 and 1.0 and compared this value to the specified probabilityToPark. If our random number was less than our probability, a car object would be created. This would indicate that an additional car has tried to park. If this new car was able to find a non empty lane, it would be parked in said lane and would be added to our master list of cars objects. If the car could not find an available space, it would fail to park and we noted this using the parkFail variable. In order to retrieve cars, we checked to see if a specific car's timeToLeave matched our defined timestep.

Extension

For my extension, I wrote code which would specify the number of cars in the holding area. I did this by manipulating my retrieveCar method as well as my run method.

I have explained my alterations to the retrieveCar method above.

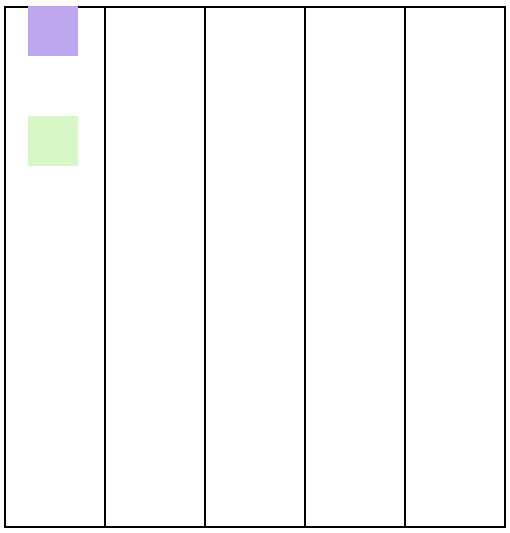
I altered my run method by creating an additional counter named "holdingLaneMain" which would keep track of the number of cars in my holding lane. As long as the holding lane was not empty, I would add the integer number returned by the retrieveCar method to this counting variable. I could then display the number of cars in the holding area by printing this integer value.

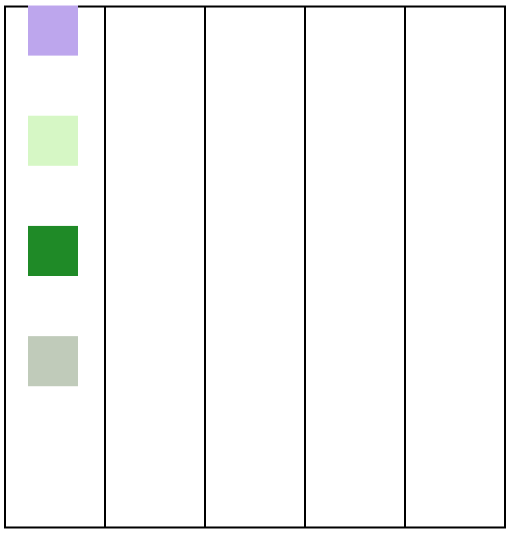


```java
//check to see if cars need to be retrieved
for ( int k = 0 ; k < masterList.size() ; k++ ){
    //if car needs to leave at same time as timestep, car will be retrieved
    System.out.println( masterList.get(k).getTimeToLeave()+ ","+ timestep);
    if (  masterList.get(k).getTimeToLeave() == timestep ){

        //create int to represent holdingLane and retrieve car object
        int holdingLane = pG.retrieveCar( masterList.get(k) );
        System.out.println( "HL" + holdingLane );
        //if the holding lane is used, mark number of cars in it
        if( holdingLane != -1 ){
            holdingLaneMain += holdingLane;
            System.out.println( "adding to holding lane");
        }
        masterList.remove(k);
        k--;
    }
}
```

Parking Garage Images





Conclusions

After working on this project I have gained a much better understanding of stacks and in which ways they are useful. I also gained more practice using "counting variables." I'm having an easier time coming up with code on my own as I gain a better and better understanding of various data structures.

While working on this project I had help from my tutor Liwei, and my TA's Adam, Matt and Kyle.