CS 115 PROJECT

Theo Guidroz

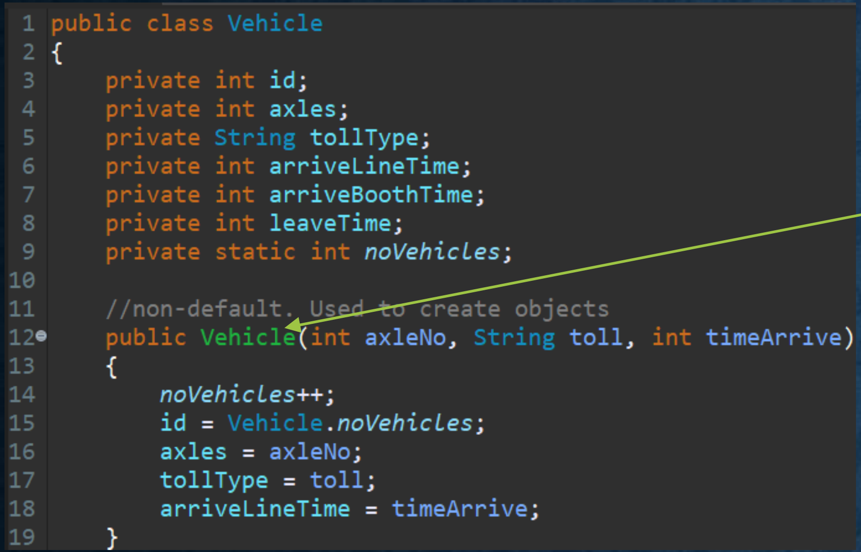
DESIGN AND TEST PLAN

CLASSES CREATED

- Vehicle
- TollBoothLine
- Simulator
- SimulatorClient

CLASS: VEHICLE

1.



```java
public class Vehicle
{
    private int id;
    private int axles;
    private String tollType;
    private int arriveLineTime;
    private int arriveBoothTime;
    private int leaveTime;
    private static int noVehicles;

    //non-default. Used to create objects
    public Vehicle(int axleNo, String toll, int timeArrive)
    {
        noVehicles++;
        id = Vehicle.noVehicles;
        axles = axleNo;
        tollType = toll;
        arriveLineTime = timeArrive;
    }
```

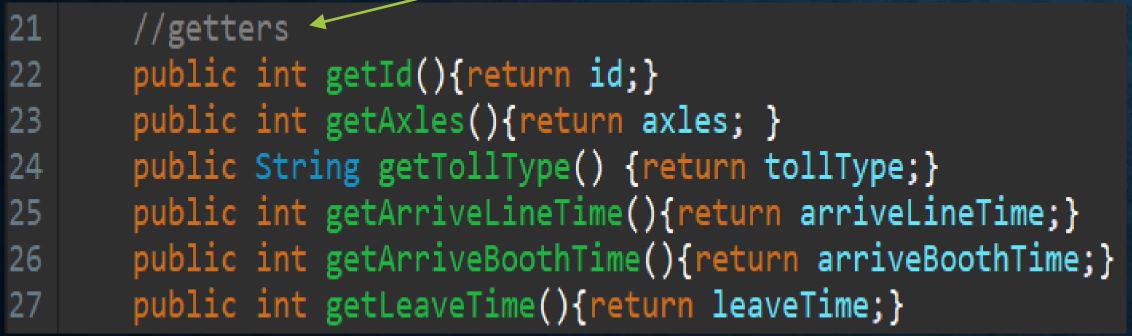
- Creates an object vehicle with argument:

1. Number of axle
2. Toll type
3. Arrive Time

- Associates unique ID to the object.

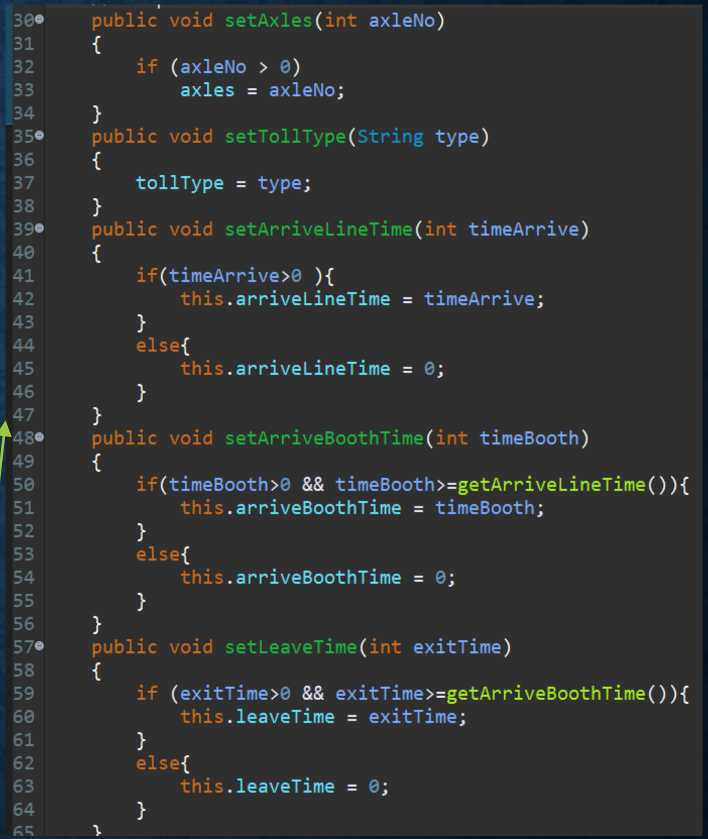
2.

Returns a characteristic



```java
    //getters
    public int getId(){return id;}
    public int getAxles(){return axles; }
    public String getTollType() {return tollType;}
    public int getArriveLineTime(){return arriveLineTime;}
    public int getArriveBoothTime(){return arriveBoothTime;}
    public int getLeaveTime(){return leaveTime;}
```

3.

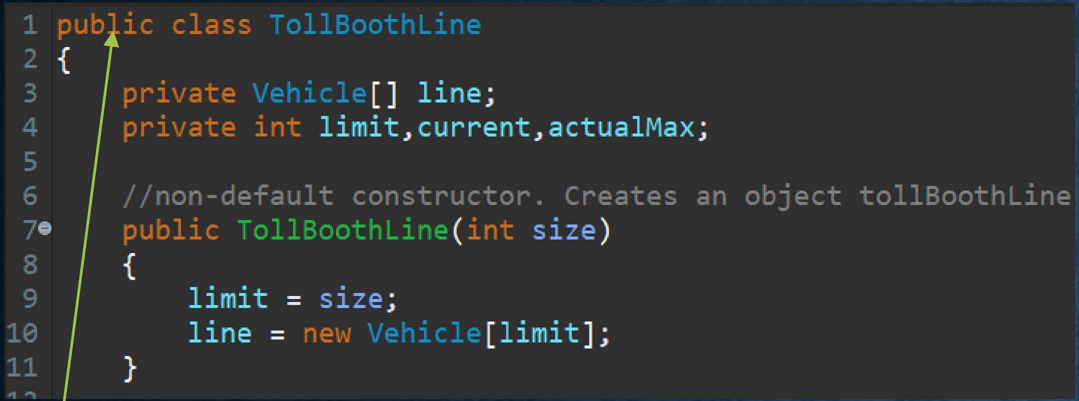


```java
public void setAxles(int axleNo)
{
    if (axleNo > 0)
        axles = axleNo;
}
public void setTollType(String type)
{
    tollType = type;
}
public void setArriveLineTime(int timeArrive)
{
    if(timeArrive>0 ){
        this.arriveLineTime = timeArrive;
    }
    else{
        this.arriveLineTime = 0;
    }
}
public void setArriveBoothTime(int timeBooth)
{
    if(timeBooth>0 && timeBooth>=getArriveLineTime()){
        this.arriveBoothTime = timeBooth;
    }
    else{
        this.arriveBoothTime = 0;
    }
}
public void setLeaveTime(int exitTime)
{
    if (exitTime>0 && exitTime>=getArriveBoothTime()){
        this.leaveTime = exitTime;
    }
    else{
        this.leaveTime = 0;
    }
}
```

Sets a characteristic

CLASS: TOLLBOOTHLINE

1.

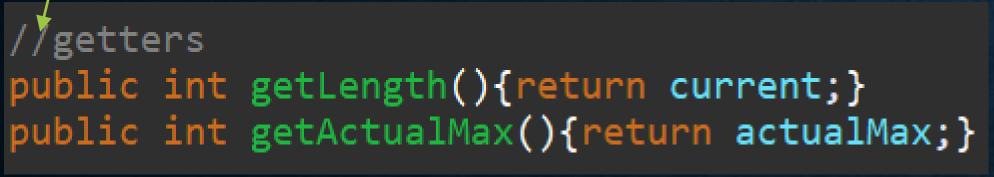


```java
public class TollBoothLine
{
    private Vehicle[] line;
    private int limit,current,actualMax;

    //non-default constructor. Creates an object tollBoothLine
    public TollBoothLine(int size)
    {
        limit = size;
        line = new Vehicle[limit];
    }
```

- Creates an object with argument size.
- Creates an array of the size attributed

Returns a characteristic

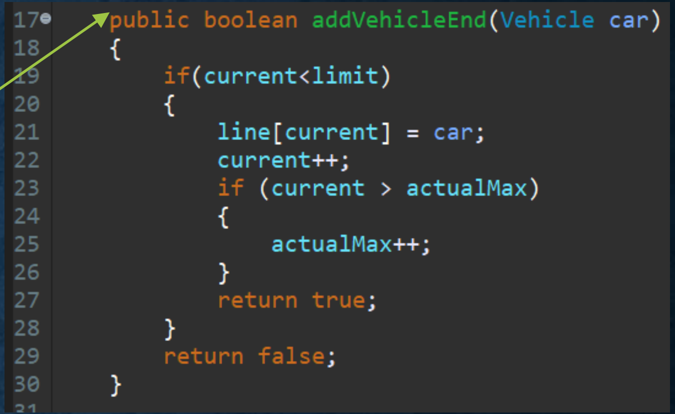
2.



```java
//getters
public int getLength(){return current;}
public int getActualMax(){return actualMax;}
```

3.

Adds a vehicle object at the end of the array.

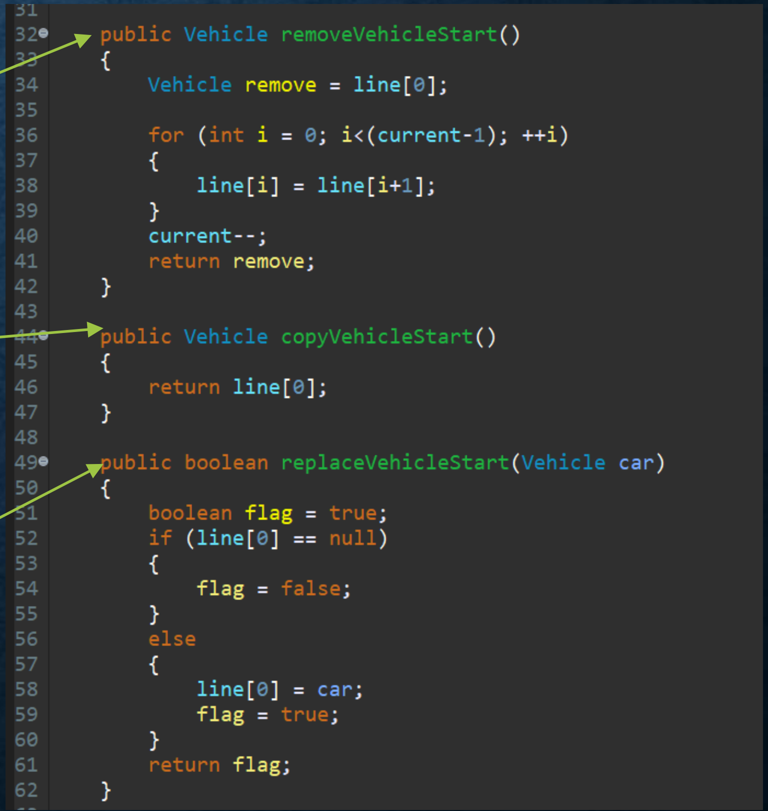


```java
public boolean addVehicleEnd(Vehicle car)
{
    if(current<limit)
    {
        line[current] = car;
        current++;
        if (current > actualMax)
        {
            actualMax++;
        }
        return true;
    }
    return false;
}
```

Removes the first object and moves the rest up the list.

Copies the first object.

Replaces the first object of the array.

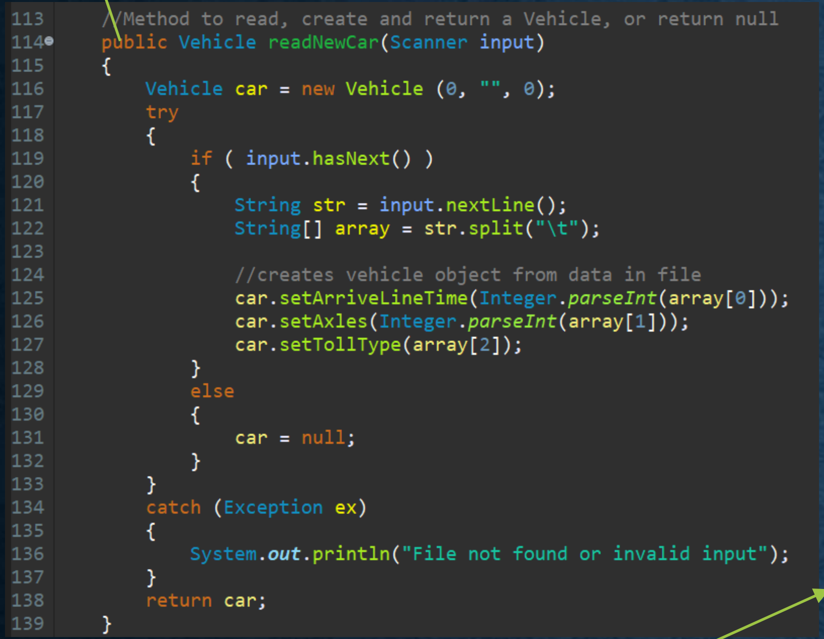


```java
public Vehicle removeVehicleStart()
{
    Vehicle remove = line[0];

    for (int i = 0; i<(current-1); ++i)
    {
        line[i] = line[i+1];
    }
    current--;
    return remove;
}

public Vehicle copyVehicleStart()
{
    return line[0];
}

public boolean replaceVehicleStart(Vehicle car)
{
    boolean flag = true;
    if (line[0] == null)
    {
        flag = false;
    }
    else
    {
        line[0] = car;
        flag = true;
    }
    return flag;
}
```

CLASS: SIMULATOR

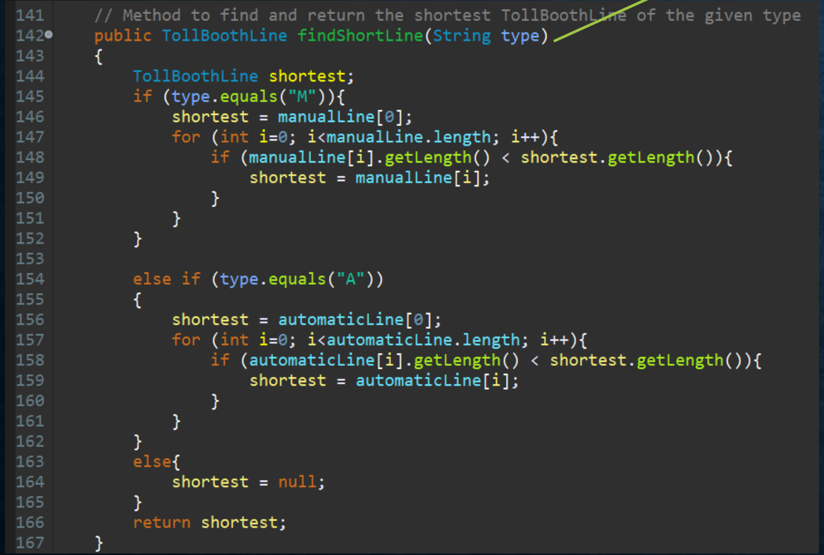
Reads data and creates vehicle objects

1.



```java
    //Method to read, create and return a Vehicle, or return null
    public Vehicle readNewCar(Scanner input)
    {
        Vehicle car = new Vehicle (0, "", 0);
        try
        {
            if ( input.hasNext() )
            {
                String str = input.nextLine();
                String[] array = str.split("\t");

                //creates vehicle object from data in file
                car.setArriveLineTime(Integer.parseInt(array[0]));
                car.setAxles(Integer.parseInt(array[1]));
                car.setTollType(array[2]);
            }
            else
            {
                car = null;
            }
        }
        catch (Exception ex)
        {
            System.out.println("File not found or invalid input");
        }
        return car;
    }
```

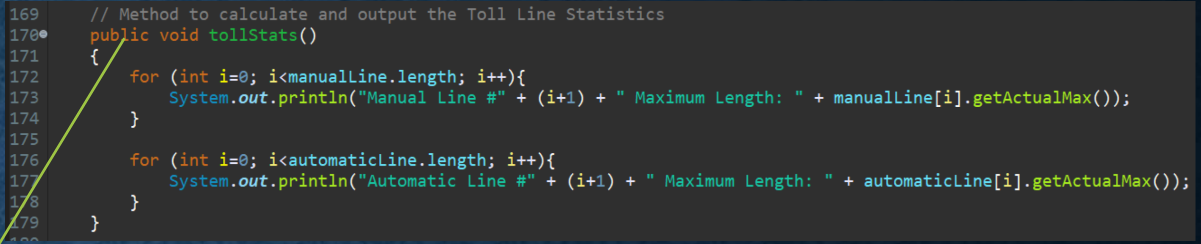
2.



```java
    // Method to find and return the shortest TollBoothLine of the given type
    public TollBoothLine findShortLine(String type)
    {
        TollBoothLine shortest;
        if (type.equals("M")){
            shortest = manualLine[0];
            for (int i=0; i<manualLine.length; i++){
                if (manualLine[i].getLength() < shortest.getLength()){
                    shortest = manualLine[i];
                }
            }
        }

        else if (type.equals("A"))
        {
            shortest = automaticLine[0];
            for (int i=0; i<automaticLine.length; i++){
                if (automaticLine[i].getLength() < shortest.getLength()){
                    shortest = automaticLine[i];
                }
            }
        }
        else{
            shortest = null;
        }
        return shortest;
    }
```

Returns the shortest booth line.

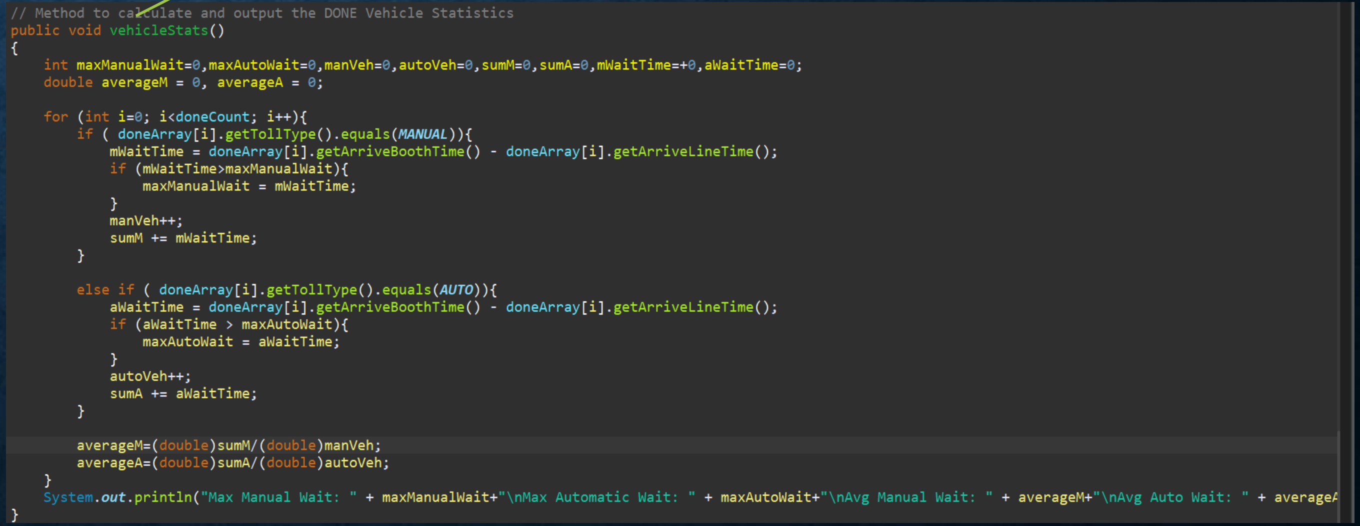
3.



```java
    // Method to calculate and output the Toll Line Statistics
    public void tollStats()
    {
        for (int i=0; i<manualLine.length; i++){
            System.out.println("Manual Line #" + (i+1) + " Maximum Length: " + manualLine[i].getActualMax());
        }

        for (int i=0; i<automaticLine.length; i++){
            System.out.println("Automatic Line #" + (i+1) + " Maximum Length: " + automaticLine[i].getActualMax());
        }
    }
```

Prints the length of the longest automatic booth and manual booth.

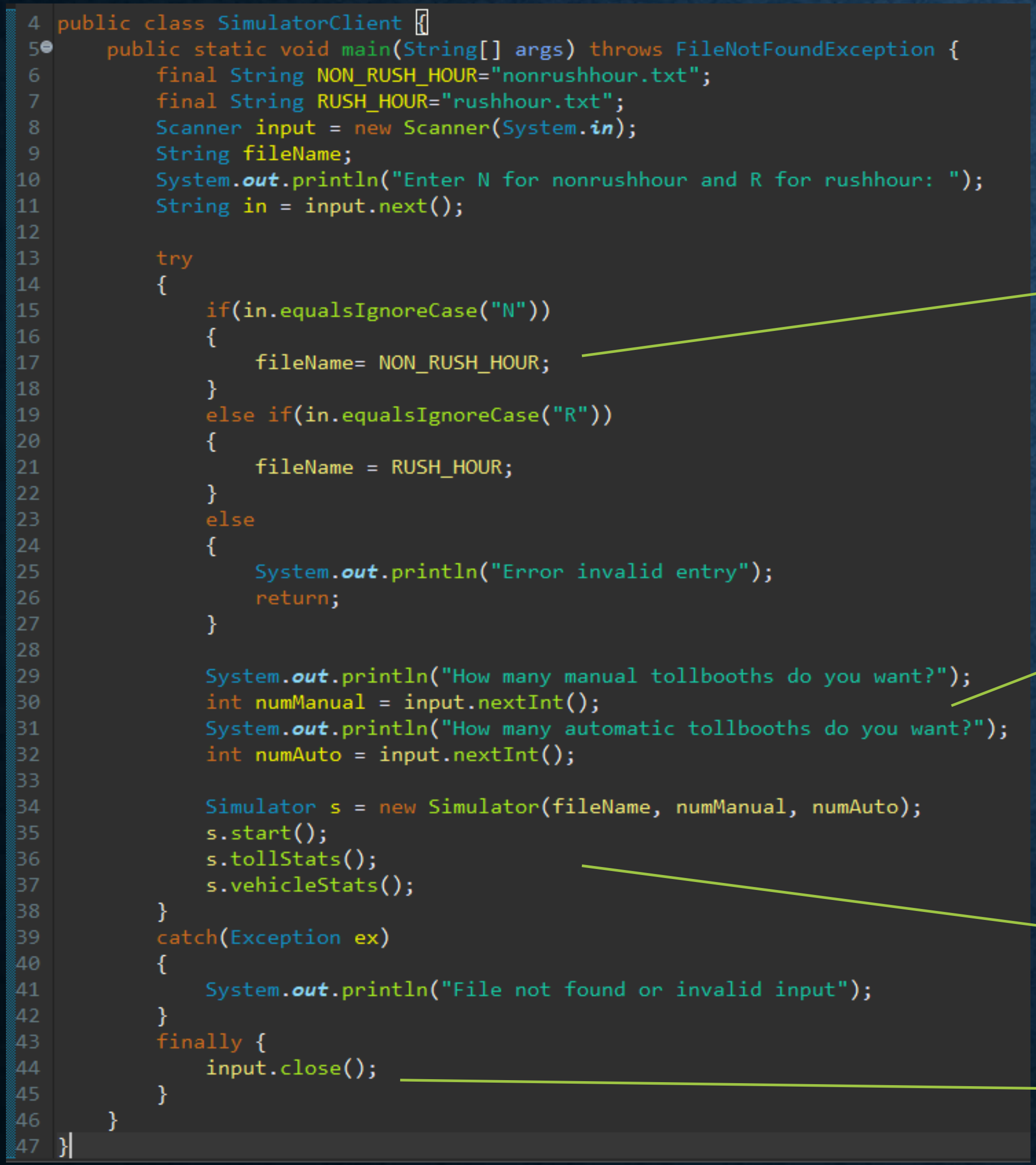
4.



```java
// Method to calculate and output the DONE Vehicle Statistics
public void vehicleStats()
{
    int maxManualWait=0,maxAutoWait=0,manVeh=0,autoVeh=0,sumM=0,sumA=0,mWaitTime=+0,aWaitTime=0;
    double averageM = 0, averageA = 0;

    for (int i=0; i<doneCount; i++){
        if ( doneArray[i].getTollType().equals(MANUAL)){
            mWaitTime = doneArray[i].getArriveBoothTime() - doneArray[i].getArriveLineTime();
            if (mWaitTime>maxManualWait){
                maxManualWait = mWaitTime;
            }
            manVeh++;
            sumM += mWaitTime;
        }

        else if ( doneArray[i].getTollType().equals(AUTO)){
            aWaitTime = doneArray[i].getArriveBoothTime() - doneArray[i].getArriveLineTime();
            if (aWaitTime > maxAutoWait){
                maxAutoWait = aWaitTime;
            }
            autoVeh++;
            sumA += aWaitTime;
        }

        averageM=(double)sumM/(double)manVeh;
        averageA=(double)sumA/(double)autoVeh;
    }
    System.out.println("Max Manual Wait: " + maxManualWait+"\nMax Automatic Wait: " + maxAutoWait+"\nAvg Manual Wait: " + averageM+"\nAvg Auto Wait: " + averageA
}
```

Prints the longest wait time on the manual booth and automatic booth.
Prints the average wait time.

CLASS: SIMULATORCLIENT



```java
public class SimulatorClient {
    public static void main(String[] args) throws FileNotFoundException {
        final String NON_RUSH_HOUR="nonrushhour.txt";
        final String RUSH_HOUR="rushhour.txt";
        Scanner input = new Scanner(System.in);
        String fileName;
        System.out.println("Enter N for nonrushhour and R for rushhour: ");
        String in = input.next();

        try
        {
            if(in.equalsIgnoreCase("N"))
            {
                fileName= NON_RUSH_HOUR;
            }
            else if(in.equalsIgnoreCase("R"))
            {
                fileName = RUSH_HOUR;
            }
            else
            {
                System.out.println("Error invalid entry");
                return;
            }

            System.out.println("How many manual tollbooths do you want?");
            int numManual = input.nextInt();
            System.out.println("How many automatic tollbooths do you want?");
            int numAuto = input.nextInt();

            Simulator s = new Simulator(fileName, numManual, numAuto);
            s.start();
            s.tollStats();
            s.vehicleStats();
        }
        catch(Exception ex)
        {
            System.out.println("File not found or invalid input");
        }
        finally {
            input.close();
        }
    }
}
```

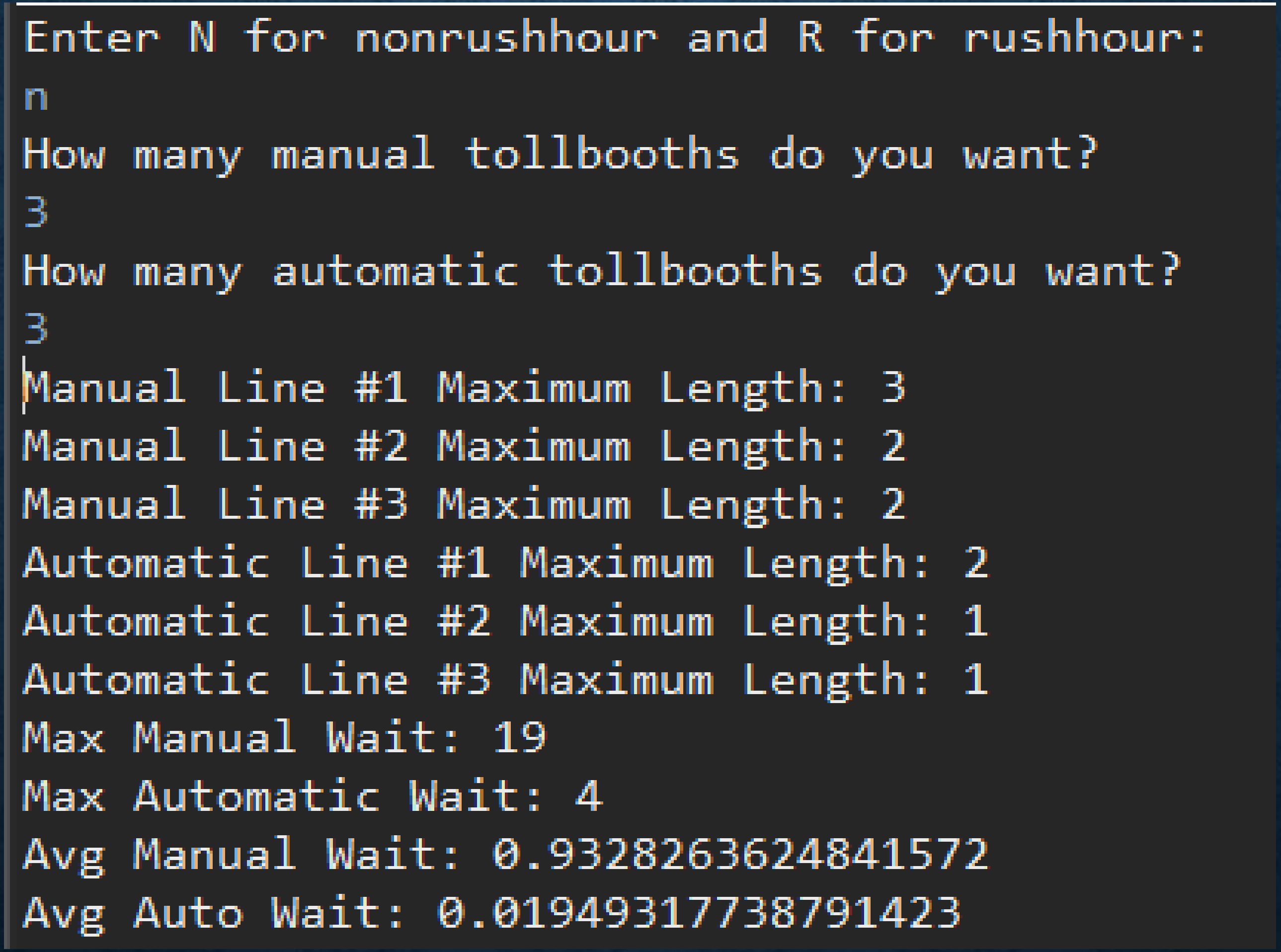
Gives the option of N and R depending on which file to use.

Asks the number of lines wanted.

Prints the statistics

Closes the user input option.

TEST CASE



```
Enter N for nonrushhour and R for rushhour:
n
How many manual tollbooths do you want?
3
How many automatic tollbooths do you want?
3
Manual Line #1 Maximum Length: 3
Manual Line #2 Maximum Length: 2
Manual Line #3 Maximum Length: 2
Automatic Line #1 Maximum Length: 2
Automatic Line #2 Maximum Length: 1
Automatic Line #3 Maximum Length: 1
Max Manual Wait: 19
Max Automatic Wait: 4
Avg Manual Wait: 0.9328263624841572
Avg Auto Wait: 0.01949317738791423
```

ANALYSIS

- Since most vehicles have 2 axles and automatic booths are faster, more automatic booths than manual booths should be installed.
- I would recommend a ratio of 6:1 for automatic booths : manual booth.
- In 10 years the number of booths should double if the amount of vehicles double.