```java
package client;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;
import javax.swing.Timer;
import javax.swing.border.TitledBorder;

import service.ReportGenerator;
import service.SimulationClock;

public class ClockPanel extends JPanel {

        private static final long serialVersionUID = -2166709692460369850L;
        SimulationClock simClock;
        JLabel clockLabel;
        Timer tm = new Timer(0,null);
        ActionListener clockLabelListener;
        ActionListener clockButtonListener;
        ReportGenerator generator;

        public ClockPanel() {
                super();
                setBorder(BorderFactory.createLineBorder(Color.BLACK));

                clockLabel = new JLabel("--", SwingConstants.CENTER);
                clockLabel.setPreferredSize(new Dimension(75,50));
                clockLabel.setBorder(BorderFactory.createTitledBorder("Clock"));
                add(clockLabel);

                JButton start =new JButton("Start");
                JButton stop =new JButton("Stop");
                JButton save = new JButton("Save Report");

                add(start);
                add(stop);
                add(save);

                clockLabelListener = new ActionListener()
                {

                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                clockLabel.setText(""+simClock.getTime());
                                repaint();
                        }
                };

                clockButtonListener = new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                if(e.getActionCommand() == "Start")
                                {
                                        tm.start();
                                }
                                if(e.getActionCommand() == "Stop")
                                {
                                        tm.stop();
                                }
                                if(e.getActionCommand() == "Save Report")
                                {
                                        saveReport();
                                }

                        }
                };

                start.addActionListener(clockButtonListener);
                stop.addActionListener(clockButtonListener);
                save.addActionListener(clockButtonListener);
        }

        public void saveReport()
        {
                final JFileChooser fc = new JFileChooser();
                int returnVal = fc.showSaveDialog(this);

        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
                        generator.saveReport(file.getAbsolutePath());
        }
        }

        public void setReportGenerator(ReportGenerator rg)
        {
                this.generator = rg;
        }

        public void setClock(Timer tm, SimulationClock simClock)
        {
                this.tm = tm;
                this.tm.addActionListener(clockLabelListener);
                this.simClock = simClock;
        }

}
```

```java
package client;

import java.awt.BorderLayout;
import java.util.List;

import javax.swing.JPanel;
import javax.swing.Timer;

import core.endpoints.Destination;
import service.DemandMatrix;
import service.ReportGenerator;
import service.SimulationClock;
import service.TrafficSignalScheduler;

public class ControlPanel extends JPanel {

        private static final long serialVersionUID = 5379117713281763963L;

        private PolicyPanel policy_panel;
        private DemandMatrixPanel demand_matrix_panel;
        private ClockPanel clock_panel;

        public ControlPanel(Timer tm, SimulationClock simClock) {
                super();
                setLayout(new BorderLayout());

                policy_panel = new PolicyPanel();
                policy_panel.setClockTimer(tm);

                add(policy_panel,BorderLayout.CENTER);

                demand_matrix_panel = new DemandMatrixPanel();
                add(demand_matrix_panel,BorderLayout.EAST);

                clock_panel= new ClockPanel();
                clock_panel.setClock(tm, simClock);
                add(clock_panel,BorderLayout.SOUTH);
        }

        public void setDemandMatrixCars(DemandMatrix dm){
                demand_matrix_panel.setDemandMatrixCars(dm);
        }

        public void setDemandMatrixBuses(DemandMatrix dm){
                demand_matrix_panel.setDemandMatrixBuses(dm);
        }

        public void setReportGenerator(ReportGenerator generator)
        {
                clock_panel.setReportGenerator(generator);
        }

        public void addTrafficScheduler(TrafficSignalScheduler scheduler)
        {
                policy_panel.addLightScheduler(scheduler);
        }

        public void addDestinations(Destination d)
        {
                policy_panel.addDesitnation(d);
        }
}
```

```java
package client;

import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

import javax.swing.AbstractListModel;
import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.ListCellRenderer;
import javax.swing.ListModel;
import javax.swing.UIManager;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.JTableHeader;
import javax.swing.table.TableModel;

import core.endpoints.Destination;
import service.DemandMatrix;
import service.DemandMatrixException;

public class DemandMatrixPanel extends JPanel {

        private static final long serialVersionUID = -7688408801570692394L;

        private DemandMatrix dm_cars;
        private DemandMatrix dm_buses;

        JPanel demandPanel;
        JTable table_cars;
        JTable table_buses;
        JList rowHeader;
        JList rowHeader2;
        List <Destination>destinations_cars;
        List <Destination>destinations_buses;

        public DemandMatrixPanel() {
                super();
                setLayout(new BoxLayout(this,BoxLayout.Y_AXIS));
                setBorder(BorderFactory.createLineBorder(Color.BLACK));

                String[] demand_matrix_strings = { "Demand Matrix for Cars", "Demand M
atrix for Buses" };

                //Create the combo box, select item at index 4.
                //Indices start at 0, so 4 specifies the pig.
                JComboBox demand_matrix_combo = new JComboBox(demand_matrix_stri
ngs);

                demand_matrix_combo.setSelectedIndex(0);
                demand_matrix_combo.addActionListener (new ActionListener () {
                    public void actionPerformed(ActionEvent e) {
                        JComboBox cb = (JComboBox)e.getSource();
                        String myselection = (String)cb.getSelectedItem();
                        //updateLabel(petName);
                        //table.

                        CardLayout cl = (CardLayout)(demandPanel.getLayout());

                                if(myselection=="Demand Matrix for Cars"){
                                        cl.show(demandPanel,"Cars");
                                }
                                else if(myselection=="Demand Matrix for Buses"){
                                        cl.show(demandPanel, "Buses");
                                }
                    }
                });
                add(demand_matrix_combo);

                ListModel lm = new AbstractListModel() {
                      String headers[] = { "aaa", "b", "c", "d", "e", "f", "g", "h
", "i" };

                      public int getSize() {
                        return headers.length;
                      }

                      public Object getElementAt(int index) {
                        return headers[index];
                      }
                    };

                    DefaultTableModel dm = new DefaultTableModel(){
                        public boolean isCellEditable(int row,int cols)
                        {
                                if(cols==row ){return false;}
                                return true;
                        }
                    };//(lm.getSize(), 10);
                     table_cars = new JTable(dm);
                     //table_cars.getModel().addTableModelListener(this);
                     table_cars.getModel().addTableModelListener(new TableModelL
istener(){

                                @Override
                                public void tableChanged(TableModelEvent e) {
                                        int row = e.getFirstRow();
                                int column = e.getColumn();
                                if(row==-1 || column==-1){
                                        return;
                                }
                                TableModel model = (TableModel)e.getSource();
                                String columnName = model.getColumnName(column);

                                String data = model.getValueAt(row, column).toSt
ring();

                                ListModel listmodel1=rowHeader.getModel();
                                String rowName=listmodel1.getElementAt(row).toSt
ring();
                                Destination from=new Destination();
                                Destination to=new Destination();
                                for(Destination des: destinations_cars){
                                        String label=des.getLabel();
                                        if(label.equals(columnName)){
                                                to=des;
                                        }
                                        if(label.equals(rowName)){
                                                from=des;
                                        }
                                }
                                try {
                                        try {
                                                double previous_value=dm_cars.ge
tDemand(from, to);

                                                double data_double=Double.parseD
ouble(data);
```

```java
                                                int len = data.length();
                                                char lastChar = data.charAt(len
- 1);

                                                if(data_double>=0 && data_double
<=1 && lastChar!='.'){
                                                        dm_cars.setDemand(from,
to, data_double);
                                                }
                                                else{
                                                        dm_cars.setDemand(from,
to, previous_value);
                                                        model.setValueAt(previou
s_value, row, column);
                                                }

                                        } catch (NumberFormatException error) {
                                                model.setValueAt(dm_cars.getDema
nd(from, to), row, column);
                                        }
                                        } catch (NumberFormatException e1) {
                                                // TODO Auto-generated catch blo
ck
                                                e1.printStackTrace();
                                        } catch (DemandMatrixException e1) {
                                                // TODO Auto-generated catch blo
ck
                                                e1.printStackTrace();
                                        }

                                }

                     });

                    table_cars.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                    table_cars.setRowSelectionAllowed(false);

                    rowHeader = new JList(lm);
                    rowHeader.setBackground(new Color(0f,0f,0f,0f ));
                    ListModel model = new AbstractListModel() {
                              String headers[] = { "hello", "b", "c", "d", "e", "f
", "g", "h", "i" };

                              public int getSize() {
                                return headers.length;
                              }

                              public Object getElementAt(int index) {
                                return headers[index];
                              }
                            };
                    rowHeader.setModel(model);
                    rowHeader.setFixedCellWidth(50);

                    rowHeader.setFixedCellHeight(table_cars.getRowHeight()
                        /*+ table.getRowMargin()*/);
                    //                          + table.getIntercellSpacing().h
eight);
                    rowHeader.setCellRenderer(new RowHeaderRenderer(table_cars))
;

                    JScrollPane scroll = new JScrollPane(table_cars);
                    scroll.setRowHeaderView(rowHeader);
                    //add(scroll, BorderLayout.CENTER);

                    DefaultTableModel dm2 = new DefaultTableModel(){
                        @Override
                        public boolean isCellEditable(int row,int cols)
                        {
                                if(cols==row ){return false;}

                            return true;

                                                        }
                    };//(lm.getSize(), 10);
                     table_buses = new JTable(dm2);
                    //table_buses.getModel().addTableModelListener(this);

                    table_buses.getModel().addTableModelListener(new TableModelL
istener(){

                                @Override
                                public void tableChanged(TableModelEvent e) {
                                        int row = e.getFirstRow();
                                int column = e.getColumn();
                                if(row==-1 || column==-1){
                                        return;
                                }
                                TableModel model = (TableModel)e.getSource();
                                String columnName = model.getColumnName(column);

                                String data = model.getValueAt(row, column).toSt
ring();


                                ListModel listmodel1=rowHeader2.getModel();
                                String rowName=listmodel1.getElementAt(row).toSt
ring();
                                Destination from=new Destination();
                                Destination to=new Destination();
                                for(Destination des: destinations_buses){
                                        String label=des.getLabel();
                                        if(label.equals(columnName)){
                                                to=des;
                                        }
                                        if(label.equals(rowName)){
                                                from=des;
                                        }
                                }
                                try {
                                        try {
                                                double previous_value=dm_buses.g
etDemand(from, to);

                                                double data_double=Double.parseD
ouble(data);

                                                int len = data.length();
                                                char lastChar = data.charAt(len
- 1);

                                                if(data_double>=0 && data_double
<=1 && lastChar!='.'){
                                                        dm_buses.setDemand(from,
 to, data_double);
                                                }
                                                else{
                                                        dm_buses.setDemand(from,
 to, previous_value);
                                                        model.setValueAt(previou
s_value, row, column);
                                                }

                                        } catch (NumberFormatException error) {
                                                model.setValueAt(dm_buses.getDem
and(from, to), row, column);
```

```
                                        }
                                        } catch (NumberFormatException e1) {
                                                // TODO Auto-generated catch blo
ck
                                                e1.printStackTrace();
                                        } catch (DemandMatrixException e1) {
                                                // TODO Auto-generated catch blo
ck
                                                e1.printStackTrace();
                                        }

                                }

                     });
                     table_buses.setRowSelectionAllowed(false);

                    table_buses.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);

                     rowHeader2 = new JList(lm);
                     rowHeader2.setBackground(new Color(0f,0f,0f,0f ));
                    rowHeader2.setFixedCellWidth(50);

                    rowHeader2.setFixedCellHeight(table_buses.getRowHeight());
                    rowHeader2.setCellRenderer(new RowHeaderRenderer(table_buses
));

                    JScrollPane scroll_buses = new JScrollPane(table_buses);
                    scroll_buses.setRowHeaderView(rowHeader2);

                    demandPanel = new JPanel(new CardLayout());
                        demandPanel.add(scroll, "Cars");
                        demandPanel.add(scroll_buses, "Buses");
                        add(demandPanel);
        }



        public void setDemandMatrixCars(DemandMatrix dm){
                this.dm_cars=dm;
                DefaultTableModel dtm = (DefaultTableModel) table_cars.getModel(
);
                destinations_cars=dm_cars.getDestinations();

                final String[] test2=new String[destinations_cars.size()];
                for(int i=0;i<destinations_cars.size();i++){
                        test2[i]=destinations_cars.get(i).getLabel();
                }
                ListModel model = new AbstractListModel() {

                      String headers[] = test2;

                      public int getSize() {
                        return headers.length;
                      }

                      public Object getElementAt(int index) {
                        return headers[index];
                      }
                    };
            rowHeader.setModel(model);

                for(int j=0;j<destinations_cars.size();j++){
                        dtm.addColumn(destinations_cars.get(j).getLabel());
                }

                for(int j=0;j<destinations_cars.size();j++)
            {
                        Object [] test_array=new Object[destinations_cars.size()
];
                        for(int i=0;i<destinations_cars.size();i++){
                                try {
                                        double prob=dm_cars.getDemand(destinatio
ns_cars.get(j), destinations_cars.get(i));
                                        test_array[i]=""+prob;

                                } catch (DemandMatrixException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                            }
                        dtm.addRow(test_array);
                        }
        }

        public void setDemandMatrixBuses(DemandMatrix dm){
                this.dm_buses=dm;
                DefaultTableModel dtm = (DefaultTableModel) table_buses.getModel
();
                destinations_buses=dm_buses.getDestinations();

                final String[] test2=new String[destinations_buses.size()];
                for(int i=0;i<destinations_buses.size();i++){
                        test2[i]=destinations_buses.get(i).getLabel();
                }
                ListModel model = new AbstractListModel() {

                      String headers[] = test2;

                      public int getSize() {
                        return headers.length;
                      }

                      public Object getElementAt(int index) {
                        return headers[index];
                      }
                    };
            rowHeader2.setModel(model);

                for(int j=0;j<destinations_buses.size();j++){
                        dtm.addColumn(destinations_buses.get(j).getLabel());
                }

                for(int j=0;j<destinations_buses.size();j++)
            {
                        Object [] test_array=new Object[destinations_buses.size(
)];
                        for(int i=0;i<destinations_buses.size();i++){
                                try {
                                        double prob=dm_buses.getDemand(destinati
ons_buses.get(j), destinations_buses.get(i));
                                        test_array[i]=""+prob;

                                } catch (DemandMatrixException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                            }
                        dtm.addRow(test_array);
                        }
        }

        class RowHeaderRenderer extends JLabel implements ListCellRenderer {

                  RowHeaderRenderer(JTable table) {
                    JTableHeader header = table.getTableHeader();
                    setOpaque(true);
                    setBorder(UIManager.getBorder("TableHeader.cellBorder"));
```

```
                    setHorizontalAlignment(CENTER);
                    setForeground(header.getForeground());
                    setBackground(header.getBackground());
                    setFont(header.getFont());
                  }

                  public Component getListCellRendererComponent(JList list, Obje
ct value,
                      int index, boolean isSelected, boolean cellHasFocus) {
                    setText((value == null) ? "" : value.toString());
                    return this;
                  }
                }



}
```

```java
package client;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Stroke;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.Timer;

import client.Renderer;
import service.DemandMatrix;
import service.DemandMatrixException;
import service.ReportGenerator;
import service.RoadNetwork;
import service.SimulationClock;
import core.endpoints.Destination;
import core.endpoints.EndPointException;
import core.network.Road;
import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;

/*
 * AM > This is the network for a straight road
 */
public class Network1 extends Network {
        private JPanel view;
        private ControlPanel controls;
        private Timer tm;
        private ActionListener actionListener;
        private RoadNetwork roadNetwork;
        private Road ra_b;
        private Road rb_a;
        private Destination A;
        private Destination B;
        private SimulationClock clock;
        private DemandMatrix dm_cars;
        private DemandMatrix dm_buses;
        private List<Vehicle> vehicleList;
        private int roadLength = 25;
        private int numOfLanes = 2;
        private int carWidth = 20;
        private int vehicleHeight = 10;
        private int busWidth = 30;



        public Network1() {
                super();

                //AM > Every time the clock ticks move cars
                actionListener = new ActionListener(){

                        @Override
                        public void actionPerformed(ActionEvent arg0) {

                                clock.incrementClock();
                                view.repaint();
                        }
                };
                clock = new SimulationClock();
                tm = new Timer(1000, actionListener);

                controls = new ControlPanel(tm,clock);

                //AM > Create a road
                ra_b= new Road(numOfLanes,roadLength);
                rb_a = new Road(numOfLanes, roadLength);
                A = new Destination("A");
                B = new Destination("B");

                A.setClock(clock);
                B.setClock(clock);

                ra_b.setSource(A);
                ra_b.setSink(B);

                rb_a.setSource(B);
                rb_a.setSink(A);

                roadNetwork = new RoadNetwork();
                roadNetwork.addRoad(ra_b);
                roadNetwork.addRoad(rb_a);

                dm_cars = new DemandMatrix();
                dm_cars.addDestination(A);
                dm_cars.addDestination(B);
                dm_cars.setVehicleType(Car.class);
                try {
                        dm_cars.initializeMatrix();
                        dm_cars.setDemand(A, B, 1.0);
                        dm_cars.setDemand(B, A, 0.5);
                } catch (DemandMatrixException e1) {
                        e1.printStackTrace();
                }

                dm_buses = new DemandMatrix();
                dm_buses.addDestination(A);
                dm_buses.addDestination(B);
                dm_buses.setVehicleType(Bus.class);

                try {
                        dm_buses.initializeMatrix();
                        dm_buses.setDemand(A, B, 0.5);
                        dm_buses.setDemand(B, A, 1.0);
                } catch (DemandMatrixException e) {
                        e.printStackTrace();
                }

                clock.addObserver(roadNetwork);
                clock.addObserver(dm_cars);
                clock.addObserver(dm_buses);

                controls.setDemandMatrixCars(dm_cars);
                controls.setDemandMatrixBuses(dm_buses);
                controls.addDestinations(A);
                controls.addDestinations(B);

                vehicleList = new ArrayList<Vehicle>();

                ReportGenerator generator = new ReportGenerator();
                generator.addDestination(A);
                generator.addDestination(B);
                controls.setReportGenerator(generator);

                view = new JPanel()
                {
                        private static final long serialVersionUID = 1L;
```

```java
                        @Override
                        public void paintComponent(Graphics g) {
                                super.paintComponent(g);
                                int panelWidth = (int) getSize().getWidth();
                                int panelHeight = (int) getSize().getHeight();
                                int roadHeight = 150;
                                int destinationWidth = 75;
                                int roadStartX = 0 + destinationWidth;
                                int roadStartY = panelHeight/2 - roadHeight/2;
                                int roadWidth = panelWidth - destinationWidth*2;
                                int roadEndX = roadStartX+roadWidth;
                                int roadEndY = roadStartY;
                                int upperLaneDividerY = panelHeight/2 - roadHeig
ht/4;
                                int lowerLaneDividerY = panelHeight/2 + roadHeig
ht/4;

                                Renderer.renderRoad(g, "A", "B", roadStartX, ro
adStartY, roadWidth, roadHeight, Renderer.Direction.EAST);

                                //AM > Draw cars on road A to B
                                int blockWidth = (int)roadWidth/roadLength;
                                vehicleList = ra_b.getVehiclesOnRoad();

                                //For each vehicle on the road get its co-ordina
tes
                                for(Vehicle v : vehicleList)
                                {
                                        if(v instanceof Car){
                                                g.setColor(Color.RED);
                                        }
                                        else if(v instanceof Bus){
                                                g.setColor(Color.YELLOW);
                                        }
                                        //For each vehicle calculate its X and Y
 co-ordinates
                                        int carX = 0;
                                        int carY = 0;
                                        if(ra_b.getVehicleNodeIndex(v) != -1)
                                        {
                                                carX = roadStartX + blockWidth*r
a_b.getVehicleNodeIndex(v);
                                                if(ra_b.getVehicleLaneIndex(v) =
= 0)
                                                        carY =  upperLaneDivider
Y - roadHeight/8 - vehicleHeight/2;
                                                else
                                                        carY =  (panelHeight/2 -
 roadHeight/8) - vehicleHeight/2;
                                                carWidth =  (int) (blockWidth*0.
5);
                                                busWidth = (int)(blockWidth*0.75
);
                                                if(v instanceof Car){
                                                        g.fillRect(carX,carY,car
Width, vehicleHeight);
                                                }
                                                else if(v instanceof Bus){
                                                        g.fillRect(carX,carY,bus
Width, vehicleHeight);
                                                }

                                        }
                                }

                                List<Vehicle>vehicleListRb_a = rb_a.getVehiclesO
nRoad();
                                for(Vehicle v : vehicleListRb_a)
                                {
                                        if(v instanceof Car){
                                                g.setColor(Color.RED);
                                        }
                                        else if(v instanceof Bus){
                                                g.setColor(Color.YELLOW);
                                        }
                                        //For each vehicle calculate its X and Y
 co-ordinates
                                        int carX = 0;
                                        int carY = 0;
                                        if(rb_a.getVehicleNodeIndex(v) != -1)
                                        {
                                                carX = roadEndX - blockWidth*rb_
a.getVehicleNodeIndex(v) - carWidth;
                                                if(rb_a.getVehicleLaneIndex(v) =
= 0)
                                                        carY =  upperLaneDivider
Y - roadHeight/8 - vehicleHeight/2+ roadHeight/2;
                                                else
                                                        carY =  (panelHeight/2 -
 roadHeight/8) - vehicleHeight/2+ roadHeight/2;
                                                carWidth =  (int) (blockWidth*0.
5);
                                                busWidth = (int)(blockWidth*0.75
);
                                                if(v instanceof Car){
                                                        g.fillRect(carX,carY,car
Width, vehicleHeight);
                                                }
                                                else if(v instanceof Bus){
                                                        g.fillRect(carX,carY,bus
Width, vehicleHeight);
                                                }
                                        }
                                }
                                Image legend = new ImageIcon(getClass().getResou
rce("res/legend.png")).getImage();
                                g.drawImage(legend, 0, 0, null);
                        }
                };
        }

        @Override
        public JPanel getView() {
                return view;
        }


        @Override
        public JPanel getControls() {
                return controls;
        }
}
```

```java
package client;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Stroke;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.Timer;

import client.Renderer;
import service.DemandMatrix;
import service.DemandMatrixException;
import service.ReportGenerator;
import service.RoadNetwork;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.endpoints.EndPointException;
import core.network.Road;
import core.network.interfaces.InterfaceException;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.junction.Junction.JUNCTION;
import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;

public class Network2 extends Network
{

        int counter;
        Timer timer;

        private JPanel view;
        private ControlPanel controls;

        private Destination A;
        private Destination B;
        private Destination C;
        private Destination D;
        private SimulationClock clock;
        private Junction junc;

        Road ra_j;
        Road rb_j;
        Road rc_j;
        Road rd_j;

        Road rj_a;
        Road rj_b;
        Road rj_c;
        Road rj_d;

        RoadNetwork roadNetwork;

        JunctionRouter juncRouter;
        TrafficSignalScheduler scheduler;
        DemandMatrix dm_cars;
        DemandMatrix dm_buses;

        private List<Vehicle> vehicleList;
        private int hcarWidth = 10;
        private int hvehicleHeight = 10;
        private int hbusWidth = 15;
        int number_of_lanes = 2;
        int lane_length = 10;

        private int vcarHeight =5;
        private int vbusHeight = 0;
        private int vvehicleWidth=10;

        public Network2() {
                super();
                ActionListener actionListener = new ActionListener(){

                        @Override
                        public void actionPerformed(ActionEvent arg0) {
                                view.repaint();
                                clock.incrementClock();
                        }
                };

                timer = new Timer(1000, actionListener);

                clock = new SimulationClock();

                counter=0;

                //AM > Setup the destinations
                A = new Destination("A");
                B = new Destination("B");
                C = new Destination("C");
                D = new Destination("D");

                A.setClock(clock);
                B.setClock(clock);
                C.setClock(clock);
                D.setClock(clock);

                dm_cars = new DemandMatrix();
                dm_cars.addDestination(A);
                dm_cars.addDestination(B);
                dm_cars.addDestination(C);
                dm_cars.addDestination(D);
                try {
                        dm_cars.initializeMatrix();
                        dm_cars.setVehicleType(Car.class);
                        dm_cars.setDemand(A, B, 0.2);

                        dm_cars.setDemand(A, C, 1);

                        dm_cars.setDemand(A, D, 1);

                } catch (DemandMatrixException e) {
                        e.printStackTrace();
                }

                dm_buses = new DemandMatrix();
                dm_buses.addDestination(A);
                dm_buses.addDestination(B);
                dm_buses.addDestination(C);
                dm_buses.addDestination(D);
                try {
                        dm_buses.initializeMatrix();
                        dm_buses.setVehicleType(Bus.class);
                        dm_buses.setDemand(A, B, 1.0);

                        dm_buses.setDemand(B, A, 0.9);
```

```java
                        dm_buses.setDemand(C, D, 0.1);
                        dm_buses.setDemand(D, C, 0.6);

                } catch (DemandMatrixException e) {
                        e.printStackTrace();
                }

                junc = new Junction();
                roadNetwork = new RoadNetwork();

                try {
                        ra_j = new Road(number_of_lanes, lane_length);
                        ra_j.setSource(A);
                        ra_j.setSink(junc,JUNCTION.WEST);
                        roadNetwork.addRoad(ra_j);

                        rb_j = new Road(number_of_lanes, lane_length);
                        rb_j.setSource(B);
                        rb_j.setSink(junc, JUNCTION.EAST);
                        roadNetwork.addRoad(rb_j);

                        rc_j = new Road(number_of_lanes, lane_length);
                        rc_j.setSource(C);
                        rc_j.setSink(junc, JUNCTION.NORTH);
                        roadNetwork.addRoad(rc_j);

                        rd_j = new Road(number_of_lanes, lane_length);
                        rd_j.setSource(D);
                        rd_j.setSink(junc, JUNCTION.SOUTH);
                        roadNetwork.addRoad(rd_j);

                        rj_a = new Road(number_of_lanes, lane_length);
                        rj_a.setSink(A);
                        rj_a.setSource(junc,JUNCTION.WEST);
                        roadNetwork.addRoad(rj_a);

                        rj_b = new Road(number_of_lanes, lane_length);
                        rj_b.setSink(B);
                        rj_b.setSource(junc, JUNCTION.EAST);
                        roadNetwork.addRoad(rj_b);

                        rj_c = new Road(number_of_lanes, lane_length);
                        rj_c.setSink(C);
                        rj_c.setSource(junc, JUNCTION.NORTH);
                        roadNetwork.addRoad(rj_c);

                        rj_d = new Road(number_of_lanes, lane_length);
                        rj_d.setSink(D);
                        rj_d.setSource(junc, JUNCTION.SOUTH);
                        roadNetwork.addRoad(rj_d);

                        juncRouter = new JunctionRouter();
                        juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                        juncRouter.add(C, junc.getInterface(JUNCTION.NORTH));
                        juncRouter.add(B, junc.getInterface(JUNCTION.EAST));
                        juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                        junc.setRoutingTable(juncRouter);

                        junc.setSignalController();
                        scheduler = new TrafficSignalScheduler();
                        scheduler.setSignalInterval(10);
                        scheduler.addSignalController(junc.getSignalController()
);

                } catch (InterfaceException e) {
                        e.printStackTrace();
                }


                clock.addObserver(scheduler);
                clock.addObserver(roadNetwork);
                clock.addObserver(dm_cars);
                clock.addObserver(dm_buses);

                controls = new ControlPanel(timer,clock);
                controls.setDemandMatrixCars(dm_cars);
                controls.setDemandMatrixBuses(dm_buses);
                controls.addTrafficScheduler(scheduler);
                controls.addDestinations(A);
                controls.addDestinations(B);
                controls.addDestinations(C);
                controls.addDestinations(D);


                ReportGenerator generator = new ReportGenerator();
                generator.addDestination(A);
                generator.addDestination(B);
                generator.addDestination(C);
                generator.addDestination(D);
                controls.setReportGenerator(generator);

                vehicleList = new ArrayList<Vehicle>();


                view = new JPanel()
                {
                        private static final long serialVersionUID = 1L;

                        @Override
                        public void paintComponent(Graphics g) {
                                super.paintComponent(g);

                                int panelWidth = (int) getSize().getWidth();
                                int panelHeight = (int) getSize().getHeight();
                                int hroadHeight = 60;
                                int hdestinationWidth = 20;
                                int vdestinationHeight = 20;
                                int vdestinationWidth = 60;

                                //AM > Draw a horizontal road from A to junction
                                g.setColor(Color.BLACK);
                                int hra_jStartX = 0 + hdestinationWidth;
                                int hra_jStartY = panelHeight/2 - hroadHeight/2;
                                int hra_jWidth = panelWidth/2 - 2*hdestinationWi
dth;
                                int hra_jEndX = hra_jStartX+hra_jWidth;
                                int hra_jEndY = hra_jStartY;
                                Renderer.renderRoad(g, "A", "", hra_jStartX, hr
a_jStartY, hra_jWidth, hroadHeight, Renderer.Direction.EAST);

                                //AM > Draw a vertical road form C to junction
                                g.setColor(Color.BLACK);
                                int vrc_jStartY = 0 + vdestinationHeight;
                                int vrc_jStartX = panelWidth/2-hroadHeight/2;
                                int vrc_jWidth = vdestinationWidth;
                                int vrc_jHeight= panelHeight/2 - hroadHeight/2 -
 vdestinationHeight;
                                int vrc_jEndY = vrc_jStartY + vrc_jHeight;
                                int vrc_jEndX = vrc_jStartX;
                                Renderer.renderRoad(g, "", "C", vrc_jStartX, vr
c_jStartY, vrc_jHeight, hroadHeight, Renderer.Direction.SOUTH);

                                //AM > Draw vertical road from Junction to D
                                g.setColor(Color.BLACK);
                                int vrj_dStartY = panelHeight/2 + hroadHeight/2;
                                int vrj_dStartX = panelWidth/2-hroadHeight/2;
                                int vrj_dWidth = vdestinationWidth;
                                int vrj_dHeight= panelHeight/2 - hroadHeight/2 -
vdestinationHeight;
                                int vrj_dEndY = vrj_dStartY + vrj_dHeight;
                                int vrj_dEndX = vrj_dStartX;
                                Renderer.renderRoad(g, "", "D", vrj_dStartX, vr
j_dStartY, vrj_dHeight, hroadHeight, Renderer.Direction.SOUTH);

                                //AM > Draw a horizontal road from junction to B
                                g.setColor(Color.BLACK);
                                int hrj_bStartX = panelWidth/2 + vrc_jWidth/2;
                                int hrj_bStartY = panelHeight/2 - hroadHeight/2;
                                int hrj_bWidth = panelWidth/2 - hdestinationWidt
h - vrc_jWidth/2;
                                int hrj_bEndX = hrj_bStartX+hrj_bWidth;
                                int hrj_bEndY = hrj_bStartY;
                                Renderer.renderRoad(g, "", "B", hrj_bStartX, hr
j_bStartY, hrj_bWidth, hroadHeight, Renderer.Direction.EAST);

                                //AM > Draw destination A
                                int textOffsetX = 5;
                                int textOffsetY = 5;
                                g.setColor(Color.GRAY);
                                g.fillRect(0,hra_jStartY, hdestinationWidth,hroa
dHeight);
                                g.setColor(Color.BLACK);
                                g.drawString("A", hdestinationWidth/2 - textOff
setX, hra_jStartY + hroadHeight/2 + textOffsetY);

                                //AM > Draw destination B
                                g.setColor(Color.GRAY);
                                g.fillRect(hrj_bEndX, hrj_bEndY, hdestinationWid
th, hroadHeight);
                                g.setColor(Color.BLACK);
                                g.drawString("B", hrj_bEndX +hdestinationWidth/
2 - textOffsetX, hrj_bStartY + hroadHeight/2 + textOffsetY);

                                //AM > Draw destination C
                                g.setColor(Color.GRAY);
                                g.fillRect(vrc_jStartX,0,vrc_jWidth ,vdestinatio
nHeight);
                                g.setColor(Color.BLACK);
                                g.drawString("C",vrc_jStartX + hroadHeight/2 -
textOffsetX, hdestinationWidth/2 + textOffsetY);


                                //AM > Draw destination D
                                g.setColor(Color.GRAY);
                                g.fillRect(vrj_dEndX, vrj_dEndY, vrj_dWidth,vdes
tinationHeight);
                                g.setColor(Color.BLACK);
                                textOffsetX = 5;
                                textOffsetY = 5;
                                g.drawString("D", vrj_dEndX +vrj_dWidth/2 - tex
tOffsetX, vrj_dEndY + vdestinationHeight/2 + textOffsetY);
                                int upperLaneDividerY = panelHeight/2 - hroadHei
ght/4;
                                int lowerLaneDividerY = panelHeight/2 + hroadHei
ght/4;
                                int leftLaneDividerX = panelWidth/2 - vrc_jWidth
/4;
                                int rightLaneDividerX = panelWidth/2 + vrc_jWidt
h/4;
                                int blockWidth= (int)(hra_jWidth/lane_length);
                                vehicleList = ra_j.getVehiclesOnRoad();

                        //AM > Draw junction box
                        Image img = new ImageIcon(getClass().getResource("res/cycle
"+scheduler.getCycle()+".png")).getImage();
                        g.drawImage(img,panelWidth/2 - vrc_jWidth/2, panelHeight
/2 - hroadHeight/2, vrc_jWidth, hroadHeight, this);

                        //AM > Debug: Draw block width
                        //g.setColor(Color.CYAN);
                        //g.drawRect(hra_jStartX, hra_jStartY, blockWidth, hra_j
Width/16);

                        //AM > Draw vehicles going from A to Junction
                        for(Vehicle v : vehicleList)
                        {
                                //Random r = new Random();
                                //g.setColor(new Color(r.nextFloat(), r.nextFloa
t(), r.nextFloat()));
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(ra_j.getVehicleNodeIndex(v) != -1)
                            {
                                carX = hra_jStartX + blockWidth*ra_j.getVehicleN
odeIndex(v);
                                if(ra_j.getVehicleLaneIndex(v) == 0)
                                        carY =  upperLaneDividerY - hroadHeight/
8 - hvehicleHeight/2;
                                else
                                        carY =  (panelHeight/2 - hroadHeight/8)
- hvehicleHeight/2;
                                hcarWidth =  (int) (blockWidth*0.25);
                                hbusWidth = (int)(blockWidth*0.41);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,hcarWidth, hvehicle
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,hbusWidth, hvehicle
Height);
                                }
                            }
                        }

                        //AM > Drawing vehicles between Junction and B
                                blockWidth= (int)(hrj_bWidth/lane_length);
                        vehicleList = rj_b.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                //Random r = new Random();
                                //g.setColor(new Color(r.nextFloat(), r.nextFloa
t(), r.nextFloat()));
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rj_b.getVehicleNodeIndex(v) != -1)
                            {
                                carX = hrj_bStartX + blockWidth*rj_b.getVehicleN
odeIndex(v);
```

```
                                if(rj_b.getVehicleLaneIndex(v) == 0)
                                        carY =  upperLaneDividerY - hroadHeight/
8 - hvehicleHeight/2;
                                else
                                        carY =  (panelHeight/2 - hroadHeight/8)
- hvehicleHeight/2;
                                hcarWidth =  (int) (blockWidth*0.25);
                                hbusWidth = (int)(blockWidth*0.41);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,hcarWidth, hvehicle
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,hbusWidth, hvehicle
Height);
                                }
                            }
                        }

                        //AM > Draw vehicles from B to junction
                        vehicleList = rb_j.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rb_j.getVehicleNodeIndex(v) != -1)
                            {
                                carX = hrj_bEndX - blockWidth*rb_j.getVehicleNod
eIndex(v) - hcarWidth;
                                if(rb_j.getVehicleLaneIndex(v) == 0)
                                        carY =  upperLaneDividerY - hroadHeight/
8 - hvehicleHeight/2+ hroadHeight/2;
                                else
                                        carY =  (panelHeight/2 - hroadHeight/8)
- hvehicleHeight/2+ hroadHeight/2;
                                hcarWidth =  (int) (blockWidth*0.25);
                                hbusWidth = (int)(blockWidth*0.41);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,hcarWidth, hvehicle
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,hbusWidth, hvehicle
Height);
                                }
                            }
                        }

                        //AM > Draw vehicles from Junction to A
                        vehicleList = rj_a.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rj_a.getVehicleNodeIndex(v) != -1)
                            {
                                carX = hra_jEndX - blockWidth*rj_a.getVehicleNod
eIndex(v) - hcarWidth;
                                if(rj_a.getVehicleLaneIndex(v) == 0)
                                        carY =  upperLaneDividerY - hroadHeight/
8 - hvehicleHeight/2+ hroadHeight/2;
                                else
                                        carY =  (panelHeight/2 - hroadHeight/8)
- hvehicleHeight/2+ hroadHeight/2;
                                hcarWidth =  (int) (blockWidth*0.25);
                                hbusWidth = (int)(blockWidth*0.41);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,hcarWidth, hvehicle
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,hbusWidth, hvehicle
Height);
                                }
                            }
                        }

                        //AM > debug: Draw a center line between lane boundaries
                        //g.setColor(Color.RED);
                        //AM > Lane 0
                        //g.drawLine(rightLaneDividerX - vdestinationWidth/8,vrc
_jStartY, rightLaneDividerX - vdestinationWidth/8,vrc_jEndY );
                        //AM > Lane 1
                        //g.drawLine(rightLaneDividerX + vdestinationWidth/8,vrc
_jStartY, rightLaneDividerX + vdestinationWidth/8, vrc_jEndY);

                        //AM Draw vehicles from C to Junction
                        int vblockHeight = vrc_jHeight/lane_length;

                        vehicleList = rc_j.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rc_j.getVehicleNodeIndex(v) != -1)
                            {
                                carY = vrc_jStartY + vblockHeight*rc_j.getVehicl
eNodeIndex(v);
                                if(rc_j.getVehicleLaneIndex(v) == 0)
                                        carX =  rightLaneDividerX - vdestination
Width/8 - vvehicleWidth/2;
                                else
                                        carX =  rightLaneDividerX + vdestination
Width/8 - vvehicleWidth/2;
                                vcarHeight =  (int) (vblockHeight*0.5);
                                vbusHeight = (int)(vblockHeight*0.9);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,vvehicleWidth, vcar
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,vvehicleWidth, vbus
Height);
```

```
                                }
                            }
                        }

                        //AM > Draw vehicles from Junction to D
                        vblockHeight = vrj_dHeight/lane_length;
                        vehicleList = rj_d.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rj_d.getVehicleNodeIndex(v) != -1)
                            {
                                carY = vrj_dStartY + vblockHeight*rj_d.getVehicl
eNodeIndex(v);
                                if(rj_d.getVehicleLaneIndex(v) == 0)
                                        carX =  rightLaneDividerX - vdestination
Width/8 - vvehicleWidth/2;
                                else
                                        carX =  rightLaneDividerX + vdestination
Width/8 - vvehicleWidth/2;
                                vcarHeight =  (int) (vblockHeight*0.5);
                                vbusHeight = (int)(vblockHeight*0.9);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,vvehicleWidth, vcar
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,vvehicleWidth, vbus
Height);
                                }
                            }
                        }

                      //AM > Draw vehicles from D to Junction
                        vblockHeight = vrj_dHeight/lane_length;
                        vehicleList = rd_j.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rd_j.getVehicleNodeIndex(v) != -1)
                            {
                                carY = vrj_dEndY - vblockHeight*rd_j.getVehicleN
odeIndex(v) - vcarHeight;
                                if(rd_j.getVehicleLaneIndex(v) == 0)
                                        carX =  leftLaneDividerX - vdestinationW
idth/8 - vvehicleWidth/2;
                                else
                                        carX =  leftLaneDividerX + vdestinationW
idth/8 - vvehicleWidth/2;
                                vcarHeight =  (int)(vblockHeight*0.5);
                                vbusHeight = (int)(vblockHeight*0.9);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,vvehicleWidth, vcar
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,vvehicleWidth, vbus
Height);
                                }
                            }
                        }

                      //AM > Draw vehicles from Junction to C
                        vblockHeight = vrj_dHeight/lane_length;
                        vehicleList = rj_c.getVehiclesOnRoad();
                        for(Vehicle v : vehicleList)
                        {
                                if(v instanceof Car){
                                        g.setColor(Color.RED);
                                }
                                else if(v instanceof Bus){
                                        g.setColor(Color.YELLOW);
                                }
                                //For each vehicle calculate its X and Y co-ordi
nates
                            int carX = 0;
                            int carY = 0;
                            if(rj_c.getVehicleNodeIndex(v) != -1)
                            {
                                carY = vrc_jEndY - vblockHeight*rj_c.getVehicleN
odeIndex(v) - vcarHeight;
                                if(rj_c.getVehicleLaneIndex(v) == 0)
                                        carX =  leftLaneDividerX - vdestinationW
idth/8 - vvehicleWidth/2;
                                else
                                        carX =  leftLaneDividerX + vdestinationW
idth/8 - vvehicleWidth/2;
                                vcarHeight =  (int) (vblockHeight*0.5);
                                vbusHeight = (int)(vblockHeight*0.9);
                                if(v instanceof Car){
                                        g.fillRect(carX,carY,vvehicleWidth, vcar
Height);
                                }
                                else if(v instanceof Bus){
                                        g.fillRect(carX,carY,vvehicleWidth, vbus
Height);
                                }
                            }
                        }
                        Image legend = new ImageIcon(getClass().getResource("res/l
egend.png")).getImage();

                                g.drawImage(legend, 0, 0, null);
                        }
                };
        }

        @Override
        public JPanel getView() {
                // TODO Auto-generated method stub
                return view;
        }

        @Override
        public JPanel getControls() {
                // TODO Auto-generated method stub
                return controls;
        }
}
```

```java
package client;

import javax.swing.JPanel;

public abstract class Network {
        public abstract JPanel getView();
        public abstract JPanel getControls();
}
```

```java
package client;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

import javax.swing.BorderFactory;
import javax.swing.InputVerifier;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JTextField;
import javax.swing.Timer;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import core.endpoints.Destination;
import service.TrafficSignalScheduler;

public class PolicyPanel extends JPanel implements ChangeListener,ActionListener,
 ItemListener {

        private static final long serialVersionUID = 6241308576167461723L;
        private Timer timer;
        private List<TrafficSignalScheduler> schedulers = new ArrayList<TrafficS
ignalScheduler>();
        JSlider interval_slider;
        JComboBox<Destination> destinationBox;
        List<Destination> destinations = new ArrayList<Destination>();
        JTextField maxVelocityTF;
        JTextField minVelocityTF;
        JTextField velocityProbTF;
        JTextField maxAccelerationTF;
        JTextField minAccelerationTF;
        JTextField accelerationProbTF;

        public PolicyPanel() {
                super();
                setBorder(BorderFactory.createLineBorder(Color.BLACK));
                setLayout(new GridLayout(3,1));

                interval_slider = new JSlider();
                interval_slider = new JSlider();
                interval_slider.setPaintTicks(true);
                interval_slider.setPaintLabels(true);
                interval_slider.setMaximum(50);
                interval_slider.setMinimum(0);
                interval_slider.setMajorTickSpacing(10);
                interval_slider.setMinorTickSpacing(5);
                interval_slider.setPreferredSize(new Dimension(7,5));
                interval_slider.addChangeListener(this);
                interval_slider.setName("lights");
                interval_slider.setEnabled(false);
                // add(interval_slider);

                JPanel interval_panel = new JPanel(new GridLayout(2, 1));
                interval_panel.add(new JLabel("Traffic Light Interval (clock ticks)"));
                interval_panel.add(interval_slider);
                add(interval_panel);

                JSlider clock_interval_slider = new JSlider();
                clock_interval_slider = new JSlider();
                clock_interval_slider.setPaintTicks(true);
                clock_interval_slider.setPaintLabels(true);
                clock_interval_slider.setMaximum(5000);
                clock_interval_slider.setMinimum(0);
                clock_interval_slider.setMajorTickSpacing(500);
                clock_interval_slider.setMinorTickSpacing(50);
                clock_interval_slider.addChangeListener(this);
                java.util.Hashtable<Integer,JLabel> labelTable = new java.util.H
ashtable<Integer,JLabel>();
                labelTable.put(new Integer(0), new JLabel("0.1"));
                labelTable.put(new Integer(500), new JLabel("0.5"));
                labelTable.put(new Integer(1000), new JLabel("1.0"));
                labelTable.put(new Integer(2000), new JLabel("2.0"));
                labelTable.put(new Integer(3500), new JLabel("3.5"));
                labelTable.put(new Integer(5000), new JLabel("5.0"));
                clock_interval_slider.setLabelTable( labelTable );
                clock_interval_slider.setName("clock");

                JPanel clock_interval_panel = new JPanel(new GridLayout(2,1));
                clock_interval_panel.add(new JLabel("Clock Interval (seconds)"));
                clock_interval_panel.add(clock_interval_slider);
                add(clock_interval_panel);

                //AM > Add controls to set acceleration and velocity profiles at
 destinations
                JPanel profilePanel = new JPanel(new GridLayout(3,1));
                //AM >  Create controls to select destinations
                JPanel destinationSelector = new JPanel(new FlowLayout());
                destinationBox = new JComboBox<Destination>();
                destinationBox.addItemListener(this);
                JLabel destinationLabel = new JLabel("Destination Selected");
                destinationSelector.add(destinationLabel);
                destinationSelector.add(destinationBox);
                profilePanel.add(destinationSelector);

                JLabel max = new JLabel("Max");
                JLabel min = new JLabel("Min");
                JLabel prob = new JLabel("Probability");

                //AM > Create controls to set the velocity profile
                JPanel velocityProfile = new JPanel(new FlowLayout());
                JLabel velocityLabel = new JLabel("Configure Velocity Profile");
                maxVelocityTF = new JTextField("Max velocity",4);
                maxVelocityTF.setInputVerifier(new MaxMinVerifier());
                minVelocityTF = new JTextField("Min veloctiy",4);
                minVelocityTF.setInputVerifier(new MaxMinVerifier());
                velocityProbTF = new JTextField("Profile probability",4);
                velocityProbTF.setInputVerifier(new ProbVerifier());
                JButton applyVelocityProfile = new JButton("Apply");
                applyVelocityProfile.setActionCommand("ApplyVelocity");
                applyVelocityProfile.addActionListener(this);
                velocityProfile.add(velocityLabel);
                velocityProfile.add(max);
                velocityProfile.add(maxVelocityTF);
                velocityProfile.add(min);
                velocityProfile.add(minVelocityTF);
                velocityProfile.add(prob);
                velocityProfile.add(velocityProbTF);
                velocityProfile.add(applyVelocityProfile);
                profilePanel.add(velocityProfile);

                JLabel max2 = new JLabel("Max");
                JLabel min2 = new JLabel("Min");
                JLabel prob2 = new JLabel("Probability");
```

```java
                JPanel accelerationProfile = new JPanel(new FlowLayout());
                JLabel accelerationLabel = new JLabel("Configure Acceleration Profile");
                maxAccelerationTF = new JTextField("Max Acceleration",4);
                maxAccelerationTF.setInputVerifier(new MaxMinVerifier());
                minAccelerationTF = new JTextField("Min Acceleration",4);
                minAccelerationTF.setInputVerifier(new MaxMinVerifier());
                accelerationProbTF = new JTextField("Profile probability",4);
                accelerationProbTF.setInputVerifier(new ProbVerifier());
                JButton applyAccelerationProfile = new JButton("Apply");
                applyAccelerationProfile.setActionCommand("ApplyAcceleration");
                applyAccelerationProfile.addActionListener(this);
                accelerationProfile.add(accelerationLabel);
                accelerationProfile.add(max2);
                accelerationProfile.add(maxAccelerationTF);
                accelerationProfile.add(min2);
                accelerationProfile.add(minAccelerationTF);
                accelerationProfile.add(prob2);
                accelerationProfile.add(accelerationProbTF);
                accelerationProfile.add(applyAccelerationProfile);
                profilePanel.add(accelerationProfile);

                /*  //AM > Create a slider to configure driver behaviour
            JPanel driverBehaviour = new JPanel(new GridLayout(2,1));
            JLabel behaviourLabel = new JLabel("Driver Behaviour");
            JSlider behaviourSlider = new JSlider();
            behaviourSlider.setPaintTicks(true);
            behaviourSlider.setPaintLabels(true);
            behaviourSlider.setMaximum(100);
            behaviourSlider.setMinimum(0);
            behaviourSlider.setMajorTickSpacing(50);
            behaviourSlider.setMinorTickSpacing(10);
            behaviourSlider.addChangeListener(this);
            java.util.Hashtable<Integer,JLabel> labelTable2 = new java.util.Hash
table<Integer,JLabel>();
            labelTable2.put(new Integer(0), new JLabel("Reckless"));
            labelTable2.put(new Integer(25), new JLabel("Aggressive"));
            labelTable2.put(new Integer(50), new JLabel("Cruise Control"));
            labelTable2.put(new Integer(75), new JLabel("Cautious"));
            labelTable2.put(new Integer(100), new JLabel("Extremely Cautious"));
            behaviourSlider.setLabelTable( labelTable2 );
            behaviourSlider.setName("behaviour");
            driverBehaviour.add(behaviourLabel);
            driverBehaviour.add(behaviourSlider);
            profilePanel.add(driverBehaviour);
            gbc.gridy = 2;*/
                add(profilePanel);
        }

        public void setClockTimer(Timer tm)
        {
                this.timer = tm;
        }

        public void addLightScheduler(TrafficSignalScheduler scheduler)
        {
                schedulers.add(scheduler);
                if(schedulers.size() > 0)
                        interval_slider.setEnabled(true);
        }


        @Override
        public void stateChanged(ChangeEvent e) {
                JSlider source = (JSlider) e.getSource();
                if(source.getName().equalsIgnoreCase("clock"))
                {
                        if(!source.getValueIsAdjusting())
                        {
                                timer.setDelay(source.getValue() <= 100 ? 100: s
ource.getValue());
                        }
                }
                if(source.getName().equalsIgnoreCase("lights"))
                {
                        if(!source.getValueIsAdjusting())
                        {
                                for(TrafficSignalScheduler scheduler: schedulers
)
                                {
                                        scheduler.setSignalInterval(source.getVa
lue() < 1 ? 1 : source.getValue());
                                }
                        }
                }
        }

        public void addDesitnation(Destination d)
        {
                if(!destinations.contains(d))
                {
                        destinations.add(d);
                        destinationBox.addItem(d);
                        velocityProbTF.setText(String.valueOf(d.getVelocityProba
bility()));
                        maxVelocityTF.setText(String.valueOf(d.getMaxVehicleVelo
city()));
                        minVelocityTF.setText(String.valueOf(d.getMinVehicleVelo
city()));

                        accelerationProbTF.setText(String.valueOf(d.getAccelerat
ionProbability()));
                        maxAccelerationTF.setText(String.valueOf(d.getMaxVehicle
Acceleration()));
                        minAccelerationTF.setText(String.valueOf(d.getMinVehicle
Acceleration()));
                }

        }

        @Override
        public void actionPerformed(ActionEvent e) {
                if(e.getActionCommand() == "ApplyAcceleration")
                {
                        //AM > Find the selected destination
                        Destination d = (Destination) destinationBox.getSelected
Item();

                        d.setMaxVehicleAcceleration(Integer.parseInt(maxAccelera
tionTF.getText()));
                        d.setMinVehicleAcceleration(Integer.parseInt(minAccelera
tionTF.getText()));
                        d.setAccelerationProbability(Double.parseDouble(accelera
tionProbTF.getText()));

                }
                if(e.getActionCommand() == "ApplyVelocity")
                {
                        //AM > Find the selected destination
                        Destination d = (Destination) destinationBox.getSelected
Item();

                        d.setMaxVehicleVelocity(Integer.parseInt(maxVelocityTF.g
etText()));
                        d.setMinVehicleVelocity(Integer.parseInt(minVelocityTF.g
etText()));
                        d.setVelocityProbability(Double.parseDouble(velocityProb
TF.getText()));
```

```
                }
        }

        @Override
        public void itemStateChanged(ItemEvent event) {
                if (event.getStateChange() == ItemEvent.SELECTED) {
                        Destination d = (Destination)event.getItem();
                        velocityProbTF.setText(String.valueOf(d.getVelocityProba
bility()));
                        maxVelocityTF.setText(String.valueOf(d.getMaxVehicleVelo
city()));
                        minVelocityTF.setText(String.valueOf(d.getMinVehicleVelo
city()));

                        accelerationProbTF.setText(String.valueOf(d.getAccelerat
ionProbability()));
                        maxAccelerationTF.setText(String.valueOf(d.getMaxVehicle
Acceleration()));
                        minAccelerationTF.setText(String.valueOf(d.getMinVehicle
Acceleration()));
                }

        }

        class MaxMinVerifier extends InputVerifier {
            @Override
            public boolean verify(JComponent input) {
                String text = ((JTextField) input).getText();
                try {
                    int value = Integer.parseInt(text);
                    if(value >= 0)
                     return true;
                    else
                        return false;
                } catch (NumberFormatException e) {
                    return false;
                }
            }
        }

            class ProbVerifier extends InputVerifier {
                    @Override
                    public boolean verify(JComponent input) {
                        String text = ((JTextField) input).getText();
                        try {
                            double value = Double.parseDouble(text);
                            if(value >= 0 && value <= 1)
                             return true;
                            else
                                return false;
                        } catch (NumberFormatException e) {
                            return false;
                        }
                    }
            }

}
```

```
package client;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Stroke;

public class Renderer
{
        public enum Direction
        {
                NORTH,
                EAST,
                SOUTH,
                WEST
        }

        public static void renderRoad(Graphics g, String source, String dest, in
t x, int y, int length, int width, Direction direction)
        {
                int offsetX    = 0;
                int offsetY    = 0;
                int blockX     = 0;
                int blockY     = 0;
                int blockWidth = width / 2;

                switch(direction) {
                case NORTH:
                        y -= length;
                case SOUTH:
                        blockY = blockWidth;
                        offsetX = width;
                        offsetY = length;
                        break;
                case WEST:
                        x -= length;
                case EAST:
                        blockX = blockWidth;
                        offsetX = length;
                        offsetY = width;
                        break;
                default:
                        break;
                }


                /* Render Ends */
                g.setColor(Color.GRAY);
                g.fillRect(x - blockX, y - blockY, offsetX + blockX*2, offsetY +
 blockY*2);

                /* Render Road. */
                Color old = g.getColor();
                g.setColor(Color.BLACK);
                g.fillRect(x, y, offsetX, offsetY);

                /* Render Divider. */
                g.setColor(Color.WHITE);
                g.drawLine(x + blockY, y + blockX, x + offsetX - blockY, y + off
setY - blockX);

                /* Render Stripes. */
                Graphics2D g2d = (Graphics2D) g.create();
                g2d.setStroke(new BasicStroke(1, BasicStroke.CAP_BUTT, BasicStro
ke.JOIN_BEVEL, 0, new float[]{9}, 0));
                g2d.drawLine(x + blockY + (blockY / 2), y + blockX + (blockX / 2
), x + offsetX - (blockY / 2), y + offsetY - (blockX / 2));
                g2d.drawLine(x + blockY - (blockY / 2), y + blockX - (blockX / 2
), x + offsetX - 3 * (blockY / 2), y + offsetY - 3 * (blockX / 2));

                /* Render Letters. */
                g.setColor(Color.WHITE);
                g.drawString(source, x + blockY - (blockX / 2), y + blockX - (bl
ockY / 2));
                g.drawString(dest, x + offsetX + blockY + (blockX / 2), y + offs
etY - blockX - (blockY / 2));

                /* Render Letters. */
                g.setColor(old);
        }
}
```

```java
package client;

import javax.swing.BorderFactory;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;

import service.ReportGenerator;

import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;



public class Simulator extends JFrame implements ActionListener
{
        private static final long serialVersionUID = 1L;
        private JPanel controlPanel;
        private JPanel mapPanel;

        private final String MAP1PANEL = "MAP1PANEL";
        private final String MAP2PANEL = "MAP2PANEL";

        private Network network1 = new Network1();
        private Network network2 = new Network2();


        public Simulator() {
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setLayout(new GridLayout(2,1));

                Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize
();
                int width =(int) screenSize.getWidth();
                int height =(int) screenSize.getHeight();
                setBounds(20, 20, (int)(width*0.6),(int)(height*0.75));


                //controlPanel = new ControlPanel();
                //Create the panel that contains the "cards".
        mapPanel = new JPanel(new CardLayout());
        mapPanel.add(network2.getView(), MAP2PANEL);
        mapPanel.add(network1.getView(), MAP1PANEL);


        mapPanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));

        //Create the panel that contains 'cards' for map controls
        controlPanel = new JPanel(new CardLayout());
        controlPanel.add(network2.getControls(), MAP2PANEL);
        controlPanel.add(network1.getControls(), MAP1PANEL);

                add(mapPanel);
                add(controlPanel);

                // Creates a menubar for a JFrame
        JMenuBar menuBar = new JMenuBar();

        // Add the menubar to the frame
        setJMenuBar(menuBar);

        //Define and add two drop down menu to the menubar
        JMenu mapsMenu = new JMenu("Maps");

        menuBar.add(mapsMenu);

        JMenuItem network1 = new JMenuItem("Network 1");
        mapsMenu.add(network1);
        network1.addActionListener(this);

        JMenuItem network2 = new JMenuItem("Network 2");
        mapsMenu.add(network2);
        network2.addActionListener(this);

        setTitle("Traffic Simulator");
        setVisible(true);
        }


        public static void main(String[] args) {
                EventQueue.invokeLater(new Runnable() {
                        public void run() {
                                try {
                                        Simulator frame = new Simulator();
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                });
        }


        @Override
        public void actionPerformed(ActionEvent e) {
                CardLayout view_cl = (CardLayout)(mapPanel.getLayout());
                CardLayout control_cl = (CardLayout)(controlPanel.getLayout());
                if(e.getActionCommand()=="Network 1"){
                        view_cl.show(mapPanel,MAP1PANEL);
                        //AM > Set controls for network1
                        control_cl.show(controlPanel, MAP1PANEL);
                }
                if(e.getActionCommand()=="Network 2"){
                        view_cl.show(mapPanel, MAP2PANEL);
                        //AM > Set controls for network2
                        control_cl.show(controlPanel, MAP2PANEL);
                }
        }




}
```

```java
package service;

public class DemandMatrixException extends Exception {

        public DemandMatrixException(String message) {
                super(message);

        }
}
```

```
package service;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Observable;
import java.util.Observer;
import java.util.Random;

import core.endpoints.Destination;
import core.vehicle.Vehicle;
import core.vehicle.VehicleException;

public class DemandMatrix implements Observer {

        private List<Destination> destinations;
        private HashMap<Destination, HashMap<Destination, Double>> matrix;
        private Class<?> vehicleType;

        public DemandMatrix()
        {
                destinations = new ArrayList<Destination>();
                matrix = new HashMap<Destination, HashMap<Destination, Double>>(
);
        }

        public void addDestination(Destination d)
        {
                if(!destinations.contains(d))
                        destinations.add(d);
        }

        public int getDestinationCount()
        {
                return destinations.size();
        }

        public int getMatrixDimension()
        {
                return matrix.size();
        }

        public void initializeMatrix() throws DemandMatrixException
        {
                if(getDestinationCount() < 2)
                {
                        throw new DemandMatrixException("Atleast two destinations are requir
ed to initialize the matrix");
                }

                for(Destination d1 : destinations)
                {
                        if(!matrix.containsKey(d1))
                        {
                                HashMap<Destination,Double> row = new HashMap<De
stination, Double>();
                                for(Destination d2 : destinations)
                                {
                                        if(!row.containsKey(d2))
                                        {
                                                row.put(d2, 0.0);
                                        }
                                }
                                matrix.put(d1, row);
                        }
                }
        }

        public double getDemand(Destination from, Destination to) throws DemandM
atrixException
        {
                if(matrix.containsKey(from))
                {
                        HashMap<Destination, Double> row = matrix.get(from);
                        if(row.containsKey(to))
                                return row.get(to);
                        else
                                throw new DemandMatrixException("Destination to does no
t exist in the matrix");
                }
                else
                {
                        throw new DemandMatrixException("Destination from does not exist in
 the matrix");
                }
        }

        public void setDemand(Destination from, Destination to, double value) th
rows DemandMatrixException
        {
                if(from == to)
                {
                        throw new DemandMatrixException("Cannot set demand between the s
ame destination");
                }

                if(matrix.containsKey(from))
                {
                        HashMap<Destination, Double> row = matrix.get(from);
                        if(row.containsKey(to))
                        {
                                //AM > Minimum demand can be 0%
                                if(value > 0.0)
                                {
                                        //AM > Maximum demand allowed is 100%
                                        value = value > 1.0 ? 1.0 : value;
                                        row.put(to,value);
                                }
                        }
                        else
                                throw new DemandMatrixException("Destination to does no
t exist in the matrix");
                }
                else
                {
                        throw new DemandMatrixException("Destination from does not exist in
 the matrix");
                }
        }

        public void setVehicleType(Class<?> type)
        {
                vehicleType = type;
        }

        public Class<?> getVehicleType()
        {
                return vehicleType;
        }

        public void generateVehicles() throws InstantiationException, IllegalAcc
essException, VehicleException
        {
                for(Destination from : matrix.keySet())
                {
                        HashMap<Destination, Double> row = matrix.get(from);
                        for(Destination to : row.keySet())
```

```
                        {
                                if(from != to)
                                {
                                        if(new Random().nextDouble() <= row.get(
to))
                                        {
                                                //AM > Generate vehicle
                                                Vehicle v = (Vehicle) vehicleTyp
e.newInstance();
                                                v.setSource(from);
                                                v.setDestination(to);
                                                from.addVehicle(v);
                                        }
                                }
                        }
                }
        }

        @Override
        public void update(Observable o, Object arg) {
                if(o instanceof SimulationClock)
                {
                        try {
                                generateVehicles();
                        } catch (InstantiationException | IllegalAccessException
                                        | VehicleException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }

        public List<Destination> getDestinations(){
                return destinations;
        }
}
```

```
package service;

import java.io.FileWriter;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import core.endpoints.Destination;
import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;

public class ReportGenerator {

        private List<Destination> destinations;
        private List<Vehicle> consumed_vehicles;

        private static final String FILE_HEADER = "Start Time;End Time;Source;Destination;T
ype";

        public ReportGenerator(){
                destinations=new ArrayList<Destination>();
                consumed_vehicles=new ArrayList<Vehicle>();
        }

        public void saveReport(String path){
                FileWriter fileWriter = null;

                try {
                        fileWriter = new FileWriter(path);

                        //Write the CSV file header

                        fileWriter.append(""+new Timestamp((new Date().getTime()
)));
                        fileWriter.write(System.getProperty("line.separator"));
                        fileWriter.append(FILE_HEADER.toString());

                        //Add a new line separator after the header
                        fileWriter.write(System.getProperty("line.separator"));
                        for(Destination d : destinations)
                        {

                                consumed_vehicles.addAll(d.getConsumedVehicles()
);
                                d.clearConsumedQueue();
                        }
                                //Write a new student object list to the CSV fil
e
                                for (Vehicle v : consumed_vehicles) {
                                        String line = "";
                                        if(v instanceof Car){
                                        //      fileWriter.append("Car");
                                                line =  String.format("%s;%s;%s;%
s;%s",v.getStartTime(),v.getEndTime(),v.getSource().getLabel(),v.getDestination(
).getLabel(),"Car");
                                        }
                                        else if(v instanceof Bus){
                                                //fileWriter.append("Bus");
                                                line =  String.format("%s;%s;%s;%
s;%s",v.getStartTime(),v.getEndTime(),v.getSource().getLabel(),v.getDestination(
).getLabel(),"Bus");
                                        }
                                        fileWriter.write(line);
                                        fileWriter.write(System.getProperty("line.
separator"));
                                }
                        fileWriter.flush();
                        fileWriter.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }

        }

        public void addDestination(Destination destination){
                if(!destinations.contains(destination)){
                        destinations.add(destination);
                }
        }

        public int getConsumedVehiclesLength(){
                return consumed_vehicles.size();
        }

}
```

```java
package service;

import java.util.ArrayList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;

import core.endpoints.EndPointException;
import core.network.Road;

public class RoadNetwork implements Observer {

        private List<Road> roads;

        public RoadNetwork() {
                this.roads = new ArrayList<Road>();
        }

        public void addRoad(Road r) {
                if(!roads.contains(r)){
                        roads.add(r);
                }
        }

        @Override
        public void update(Observable clock, Object arg1) {
                for(Road r : roads)
                        {
                                try {
                                        r.moveTraffic();
                                } catch (EndPointException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                }
}
```

```java
package service;

import java.util.Observable;

public class SimulationClock extends Observable implements Runnable{
        private long currentTime;
        private long interval;

        private Thread systemClock;
        private volatile boolean suspended = false;
        private volatile boolean running = false;

        public SimulationClock()
        {
                this.currentTime = 0;
                //AM > Time in ms between each clock tick
                this.interval=1000;
                systemClock = new Thread(this);
        }

        public void run()
        {
                try
                {
                        while(true)
                        {
                                Thread.sleep(this.getInterval());
                                synchronized(this)
                                {
                                        if(!suspended)
                                        {
                                                this.incrementClock();
                                        }
                                }
                        }
                }
                catch(InterruptedException e)
                {
                }
        }

        public static SimulationClock getInstance()
        {
                return new SimulationClock();
        }

        public long getTime()
        {
                return currentTime;
        }

        public void resetClock()
        {
                currentTime = 0;
        }

        public void incrementClock()
        {
                setChanged();
                notifyObservers();
                this.currentTime++;
        }

        public void setInterval(long interval){
                this.interval=interval;
        }

        public long getInterval()
        {
                return interval;
        }

        public synchronized void pauseClock()
        {
                this.suspended = true;
        }

        public synchronized void resumeClock()
        {
                this.suspended = false;
        }

        public synchronized void startClock()
        {
                if(!running)
                {
                        running = true;
                        systemClock.start();
                }
        }
}
```

```java
package service;

import java.util.ArrayList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;

import core.network.interfaces.InterfaceException;
import core.network.junction.TrafficSignalController;

public class TrafficSignalScheduler implements Observer {
        private List<TrafficSignalController> controllers;
        private int signalInterval;

        public TrafficSignalScheduler()
        {
                controllers = new ArrayList<TrafficSignalController>();
                //AM > default interval is 10 clock ticks
                signalInterval = 10;
        }

        public void addSignalController(TrafficSignalController controller)
        {
                if(!controllers.contains(controller))
                        controllers.add(controller);
        }

        public void removeSignalController(TrafficSignalController controller)
        {
                if(controllers.contains(controller))
                        controllers.remove(controller);
        }

        public long getSignalInterval() {
                return signalInterval;
        }

        public void setSignalInterval(int signalInterval) {
                this.signalInterval = signalInterval;
        }

        public void changeSignals() throws InterfaceException
        {
                for(TrafficSignalController sigCont: controllers)
                {
                        sigCont.changeSignals();
                }
        }

        public int getCycle()
        {
                return controllers.get(0).getCycle();
        }

        @Override
        public void update(Observable obs, Object obj)
        {
                SimulationClock clock = (SimulationClock) obs;
                if(clock.getTime() % signalInterval == 0)
                {
                        try {
                                changeSignals();
                        } catch (InterfaceException e) {
                                e.printStackTrace();
                        }
                }
        }
}
```

```java
package client.tools;
import java.util.List;

import service.SimulationClock;
import core.endpoints.Destination;
import core.network.Lane;
import core.network.Road;
import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;
import core.vehicle.VehicleException;

public class Main {

        public static void main(String[] args) {
                int laneLength=20;
                int numOfLanes=5;

                //We create 2 roads
                Road r1 = new Road(numOfLanes, laneLength);
                Road r2 = new Road(3, laneLength);

                //We create 3 destinations
                //It will look like this: |A| ----- |B| ----- |C|
                Destination A = new Destination();
                Destination B = new Destination();
                Destination C = new Destination();

                SimulationClock clock = SimulationClock.getInstance();
                A.setClock(clock);
                B.setClock(clock);
                C.setClock(clock);

                r1.setSource(A);
                r1.setSink(B);

                r2.setSource(B);
                r2.setSink(C);

                Vehicle v1 = new Car(2,0,4);
                Vehicle v2 = new Car(1,1,10);

                Vehicle v3 = new Car(1,0,10);
                Vehicle v4 = new Car(1,0,10);
                Vehicle v5 = new Car(1,0,10);

                Vehicle v6 = new Bus(2,0,10);
                Vehicle v7 = new Bus(3,0,10);
                Vehicle v8 = new Bus(1,0,10);

                Vehicle c9 = new Car(3,0,10);

                try {
                        A.addVehicle(v1);
                } catch (VehicleException e1) {
                        e1.printStackTrace();
                }
                System.out.println("Traffic Simulator");

                for(int i = 0; i < 30; i++)
                {
                        System.out.println("\nTick "+clock.getTime());
                        List<Lane> lanes = r1.getLanes();
                        List<Lane> lanes2 = r2.getLanes();

                        int max= lanes.size()>lanes2.size() ? lanes.size() : lan
es2.size();
                        for(int j = 0; j < max; j++){

                                if(j < lanes.size())
                                        System.out.printf("|A|%s|B|", lanes.get(j)
);
                                else
                                        System.out.printf("|A| %s |B|", "no lane");
                                if(j < lanes2.size())
                                        System.out.printf("%s|C|\n", lanes2.get(j
));
                                else
                                        System.out.printf("%s |C|\n", "no lane");
                        }
                        try {
                                r2.moveTraffic();
                                r1.moveTraffic();

                                if(i == 2){
                                        A.addVehicle(v2);
                                }

                                if(i == 3){
                                        A.addVehicle(v3);
                                }

                                if(i == 4){
                                        A.addVehicle(v4);
                                }
                                if(i == 5){
                                        A.addVehicle(v5);
                                }

                                if(i == 6){
                                        A.addVehicle(v6);
                                }

                                if(i == 7){
                                        A.addVehicle(v7);
                                }
                                if(i == 8){
                                        A.addVehicle(v8);
                                }
                                if(i == 9){
                                        A.addVehicle(c9);
                                }

                        } catch (Exception e) {
                                e.printStackTrace();
                        }


                        clock.incrementClock();
                }
        }
}
```

```java
package client.tools;

import service.DemandMatrix;
import service.RoadNetwork;
import service.ReportGenerator;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.network.Road;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.junction.Junction.JUNCTION;
import core.vehicle.Car;

public class Scenario1Reports {

        /*
         * AM > This program generates the reports for
         *              the 4 scenarios of the shopping mall exercise
         */

        public static void main(String[] args) {
                try
                {
                System.out.println("Simulation started");
                int number_of_lanes = 1;
                int lane_length = 10;

                SimulationClock clock = SimulationClock.getInstance();
                clock.setInterval(1000);

                Destination A = new Destination("Athens");
                Destination B = new Destination("Bonitsa");
                Destination C = new Destination("Cesaloniki");
                Destination D = new Destination("Delfoi");

                A.setClock(clock);
                A.setVehicleAccelerationProfile(3, 1, 0.4);
                A.setVehicleVelocityProfile(6, 1, 0.4);
                B.setClock(clock);
                B.setVehicleAccelerationProfile(3, 1, 0.4);
                B.setVehicleVelocityProfile(6, 1, 0.4);
                C.setClock(clock);
                C.setVehicleAccelerationProfile(3, 1, 0.4);
                C.setVehicleVelocityProfile(6, 1, 0.4);
                D.setClock(clock);
                D.setVehicleAccelerationProfile(3, 1, 0.4);
                D.setVehicleVelocityProfile(6, 1, 0.4);

                Junction junc = new Junction();
                RoadNetwork network = new RoadNetwork();

                /*
                 * AM > Road-junction wiring
                 *                          |B|
                 *              ^
                 *                           |
                 *                           |
                 *                               \/
                 *   |A|<----->|junc|<----->|C|
                 *              ^
                 *                           |
                 *                           |
                 *              \/
                 *                          |D|
                 */

                //AM > Roads from A, B, C and D to the junction
                Road ra_j = new Road(number_of_lanes, lane_length);
                ra_j.setSource(A);
                ra_j.setSink(junc,JUNCTION.WEST);
                network.addRoad(ra_j);

                Road rb_j = new Road(number_of_lanes, lane_length);
                rb_j.setSource(B);
                rb_j.setSink(junc, JUNCTION.NORTH);
                network.addRoad(rb_j);

                Road rc_j = new Road(number_of_lanes, lane_length);
                rc_j.setSource(C);
                rc_j.setSink(junc, JUNCTION.EAST);
                network.addRoad(rc_j);

                Road rd_j = new Road(number_of_lanes, lane_length);
                rd_j.setSource(D);
                rd_j.setSink(junc, JUNCTION.SOUTH);
                network.addRoad(rd_j);

                //AM > Roads from the Junction to A, B, C and D
                Road rj_a = new Road(number_of_lanes, lane_length);
                rj_a.setSink(A);
                rj_a.setSource(junc,JUNCTION.WEST);
                network.addRoad(rj_a);

                Road rj_b = new Road(number_of_lanes, lane_length);
                rj_b.setSink(B);
                rj_b.setSource(junc, JUNCTION.NORTH);
                network.addRoad(rj_b);

                Road rj_c = new Road(number_of_lanes, lane_length);
                rj_c.setSink(C);
                rj_c.setSource(junc, JUNCTION.EAST);
                network.addRoad(rj_c);

                Road rj_d = new Road(number_of_lanes, lane_length);
                rj_d.setSink(D);
                rj_d.setSource(junc, JUNCTION.SOUTH);
                network.addRoad(rj_d);


                //AM > Setup routing table
                JunctionRouter juncRouter = new JunctionRouter();
                juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                juncRouter.add(B, junc.getInterface(JUNCTION.NORTH));
                juncRouter.add(C, junc.getInterface(JUNCTION.EAST));
                juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                junc.setRoutingTable(juncRouter);

                //AM > Setup signal scheduler
                junc.setSignalController();
                TrafficSignalScheduler scheduler = new TrafficSignalScheduler();
                scheduler.setSignalInterval(10);
                scheduler.addSignalController(junc.getSignalController());

                DemandMatrix dm = new DemandMatrix();
                dm.addDestination(A);
                dm.addDestination(B);
                dm.addDestination(C);
                dm.addDestination(D);

                dm.initializeMatrix();
                dm.setVehicleType(Car.class);

                dm.setDemand(A, B, 0.3);
                dm.setDemand(A, C, 0.3);
                dm.setDemand(A, D, 0.3);
```

```java
                dm.setDemand(B, A, 0.3);
                dm.setDemand(B, C, 0.3);
                dm.setDemand(B, D, 0.3);

                dm.setDemand(C, B, 0.3);
                dm.setDemand(C, A, 0.3);
                dm.setDemand(C, D, 0.3);

                dm.setDemand(D, B, 0.3);
                dm.setDemand(D, C, 0.3);
                dm.setDemand(D, A, 0.3);

                clock.addObserver(dm);
                clock.addObserver(network);
                clock.addObserver(scheduler);

                clock.startClock();
                System.out.println("Running scenario 1");
                Thread.sleep(1*60*1000);

                clock.pauseClock();

                ReportGenerator report= new ReportGenerator();
                report.addDestination(A);
                report.addDestination(B);
                report.addDestination(C);
                report.addDestination(D);

                String path="Scenario1_report.txt";

                report.saveReport(path);

                System.out.println("Simulation ended");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}
```

```java
package client.tools;

import service.DemandMatrix;
import service.ReportGenerator;
import service.RoadNetwork;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.network.Road;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.junction.Junction.JUNCTION;
import core.vehicle.Car;

public class Scenario2Report {

        public static void main(String[] args) {

                try
                {
                System.out.println("Simulation started");
                int number_of_lanes = 1;
                int lane_length = 10;

                SimulationClock clock = SimulationClock.getInstance();
                clock.setInterval(1000);

                Destination A = new Destination("Athens");
                Destination B = new Destination("Bonitsa");
                Destination C = new Destination("Cesaloniki");
                Destination D = new Destination("Delfoi");

                A.setClock(clock);
                A.setVehicleAccelerationProfile(3, 1, 0.4);
                A.setVehicleVelocityProfile(6, 1, 0.4);
                B.setClock(clock);
                B.setVehicleAccelerationProfile(3, 1, 0.4);
                B.setVehicleVelocityProfile(6, 1, 0.4);
                C.setClock(clock);
                C.setVehicleAccelerationProfile(3, 1, 0.4);
                C.setVehicleVelocityProfile(6, 1, 0.4);
                D.setClock(clock);
                D.setVehicleAccelerationProfile(3, 1, 0.4);
                D.setVehicleVelocityProfile(6, 1, 0.4);

                Junction junc = new Junction();
                RoadNetwork network = new RoadNetwork();

                /*
                 * AM > Road-junction wiring
                 *                         |B|
                 *              ^
                 *                          |
                 *                          |
                 *                              \/
                 *   |A|<----->|junc|<----->|C|
                 *              ^
                 *                          |
                 *                          |
                 *              \/
                 *                         |D|
                 */

                //AM > Roads from A, B, C and D to the junction
                Road ra_j = new Road(number_of_lanes, lane_length);
                ra_j.setSource(A);
                ra_j.setSink(junc,JUNCTION.WEST);
                network.addRoad(ra_j);

                Road rb_j = new Road(number_of_lanes, lane_length);
                rb_j.setSource(B);
                rb_j.setSink(junc, JUNCTION.NORTH);
                network.addRoad(rb_j);

                Road rc_j = new Road(number_of_lanes, lane_length);
                rc_j.setSource(C);
                rc_j.setSink(junc, JUNCTION.EAST);
                network.addRoad(rc_j);

                Road rd_j = new Road(number_of_lanes, lane_length);
                rd_j.setSource(D);
                rd_j.setSink(junc, JUNCTION.SOUTH);
                network.addRoad(rd_j);

                //AM > Roads from the Junction to A, B, C and D
                Road rj_a = new Road(number_of_lanes, lane_length);
                rj_a.setSink(A);
                rj_a.setSource(junc,JUNCTION.WEST);
                network.addRoad(rj_a);

                Road rj_b = new Road(number_of_lanes, lane_length);
                rj_b.setSink(B);
                rj_b.setSource(junc, JUNCTION.NORTH);
                network.addRoad(rj_b);

                Road rj_c = new Road(number_of_lanes, lane_length);
                rj_c.setSink(C);
                rj_c.setSource(junc, JUNCTION.EAST);
                network.addRoad(rj_c);

                Road rj_d = new Road(number_of_lanes, lane_length);
                rj_d.setSink(D);
                rj_d.setSource(junc, JUNCTION.SOUTH);
                network.addRoad(rj_d);


                //AM > Setup routing table
                JunctionRouter juncRouter = new JunctionRouter();
                juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                juncRouter.add(B, junc.getInterface(JUNCTION.NORTH));
                juncRouter.add(C, junc.getInterface(JUNCTION.EAST));
                juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                junc.setRoutingTable(juncRouter);

                //AM > Setup signal scheduler
                junc.setSignalController();
                TrafficSignalScheduler scheduler = new TrafficSignalScheduler();
                scheduler.setSignalInterval(10);
                scheduler.addSignalController(junc.getSignalController());
                DemandMatrix dm = new DemandMatrix();
                dm.addDestination(A);
                dm.addDestination(B);
                dm.addDestination(C);
                dm.addDestination(D);

                dm.initializeMatrix();
                dm.setVehicleType(Car.class);;

                dm.setDemand(A, B, 0.3);
                dm.setDemand(A, C, 0.8);
                dm.setDemand(A, D, 0.3);

                dm.setDemand(B, A, 0.3);
                dm.setDemand(B, C, 0.8);
                dm.setDemand(B, D, 0.3);

                dm.setDemand(C, B, 0.5);
```

```java
                dm.setDemand(C, A, 0.5);
                dm.setDemand(C, D, 0.5);

                dm.setDemand(D, B, 0.3);
                dm.setDemand(D, C, 0.8);
                dm.setDemand(D, A, 0.3);

                clock.addObserver(dm);
                clock.addObserver(network);
                clock.addObserver(scheduler);

                clock.startClock();
                System.out.println("Running scenario 2");
                Thread.sleep(3*60*1000);

                clock.pauseClock();

                ReportGenerator report= new ReportGenerator();
                report.addDestination(A);
                report.addDestination(B);
                report.addDestination(C);
                report.addDestination(D);

                String path="Scenario2_report.txt";

                report.saveReport(path);
                System.out.println("Simulation ended");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }

}
```

```java
package client.tools;

import service.DemandMatrix;
import service.ReportGenerator;
import service.RoadNetwork;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.network.Road;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.Junction.JUNCTION;
import core.vehicle.Car;

public class Scenario3Report {

public static void main(String[] args) {

                try
                {
                System.out.println("Simulation started");
                int number_of_lanes = 1;
                int lane_length = 10;

                SimulationClock clock = SimulationClock.getInstance();
                clock.setInterval(1000);

                Destination A = new Destination("Athens");
                Destination B = new Destination("Bonitsa");
                Destination C = new Destination("Cesaloniki");
                Destination D = new Destination("Delfoi");

                A.setClock(clock);
                A.setVehicleAccelerationProfile(3, 1, 0.4);
                A.setVehicleVelocityProfile(6, 1, 0.4);
                B.setClock(clock);
                B.setVehicleAccelerationProfile(3, 1, 0.4);
                B.setVehicleVelocityProfile(6, 1, 0.4);
                C.setClock(clock);
                C.setVehicleAccelerationProfile(3, 1, 0.4);
                C.setVehicleVelocityProfile(6, 1, 0.4);
                D.setClock(clock);
                D.setVehicleAccelerationProfile(3, 1, 0.4);
                D.setVehicleVelocityProfile(6, 1, 0.4);

                Junction junc = new Junction();
                RoadNetwork network = new RoadNetwork();

                /*
                 * AM > Road-junction wiring
                 *                         |B|
                 *              ^
                 *                          |
                 *                          |
                 *                              \/
                 *   |A|<----->|junc|<----->|C|
                 *              ^
                 *                          |
                 *                          |
                 *              \/
                 *                         |D|
                 */

                //AM > Roads from A, B, C and D to the junction
                Road ra_j = new Road(number_of_lanes, lane_length);
                ra_j.setSource(A);
                ra_j.setSink(junc,JUNCTION.WEST);
                network.addRoad(ra_j);

                Road rb_j = new Road(number_of_lanes, lane_length);
                rb_j.setSource(B);
                rb_j.setSink(junc, JUNCTION.NORTH);
                network.addRoad(rb_j);

                Road rc_j = new Road(number_of_lanes, lane_length);
                rc_j.setSource(C);
                rc_j.setSink(junc, JUNCTION.EAST);
                network.addRoad(rc_j);

                Road rd_j = new Road(number_of_lanes, lane_length);
                rd_j.setSource(D);
                rd_j.setSink(junc, JUNCTION.SOUTH);
                network.addRoad(rd_j);

                //AM > Roads from the Junction to A, B, C and D
                Road rj_a = new Road(number_of_lanes, lane_length);
                rj_a.setSink(A);
                rj_a.setSource(junc,JUNCTION.WEST);
                network.addRoad(rj_a);

                Road rj_b = new Road(number_of_lanes, lane_length);
                rj_b.setSink(B);
                rj_b.setSource(junc, JUNCTION.NORTH);
                network.addRoad(rj_b);

                Road rj_c = new Road(number_of_lanes, lane_length);
                rj_c.setSink(C);
                rj_c.setSource(junc, JUNCTION.EAST);
                network.addRoad(rj_c);

                Road rj_d = new Road(number_of_lanes, lane_length);
                rj_d.setSink(D);
                rj_d.setSource(junc, JUNCTION.SOUTH);
                network.addRoad(rj_d);


                //AM > Setup routing table
                JunctionRouter juncRouter = new JunctionRouter();
                juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                juncRouter.add(B, junc.getInterface(JUNCTION.NORTH));
                juncRouter.add(C, junc.getInterface(JUNCTION.EAST));
                juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                junc.setRoutingTable(juncRouter);

                //AM > Setup signal scheduler
                junc.setSignalController();
                TrafficSignalScheduler scheduler = new TrafficSignalScheduler();
                scheduler.setSignalInterval(10);
                scheduler.addSignalController(junc.getSignalController());
                DemandMatrix dm = new DemandMatrix();
                dm.addDestination(A);
                dm.addDestination(B);
                dm.addDestination(C);
                dm.addDestination(D);
                dm.initializeMatrix();
                dm.setVehicleType(Car.class);

                dm.setDemand(A, B, 0.5);
                dm.setDemand(A, C, 0.1);
                dm.setDemand(A, D, 0.3);

                dm.setDemand(B, A, 0.3);
                dm.setDemand(B, C, 0.1);
                dm.setDemand(B, D, 0.3);

                dm.setDemand(C, B, 0.3);
                dm.setDemand(C, A, 0.3);
```

```java
                dm.setDemand(C, D, 0.3);

                dm.setDemand(D, B, 0.5);
                dm.setDemand(D, C, 0.1);
                dm.setDemand(D, A, 0.3);

                clock.addObserver(dm);
                clock.addObserver(network);
                clock.addObserver(scheduler);

                clock.startClock();
                System.out.println("Running scenario 3");
                Thread.sleep(3*60*1000);

                clock.pauseClock();

                ReportGenerator report= new ReportGenerator();
                report.addDestination(A);
                report.addDestination(B);
                report.addDestination(C);
                report.addDestination(D);

                String path="Scenario3_report.txt";

                report.saveReport(path);
                System.out.println("Simulation ended");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}
```

```java
package core.endpoints;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import service.SimulationClock;
import core.vehicle.Vehicle;
import core.vehicle.VehicleException;

/*
 * AM > This class represents a Destination.
 *              Destinations are spawn points where cars originate and terminate
 */
public class Destination extends EndPoint {

        private List<Vehicle> waitingQueue;
        private List<Vehicle> consumedQueue;
        private SimulationClock clock;
        private String label;

        //AM > Create a profile for generated vehicle velocity
        private int minVehicleVelocity;
        public int getMinVehicleVelocity() {
                return minVehicleVelocity;
        }

        public void setMinVehicleVelocity(int minVehicleVelocity) {
                if(minVehicleVelocity >= 1 && minVehicleVelocity <= this.maxVehi
cleVelocity)
                        this.minVehicleVelocity = minVehicleVelocity;
        }

        public int getMaxVehicleVelocity() {
                return maxVehicleVelocity;
        }

        public void setMaxVehicleVelocity(int maxVehicleVelocity) {
                if(maxVehicleVelocity >= 1 && maxVehicleVelocity >= this.minVehi
cleVelocity)
                        this.maxVehicleVelocity = maxVehicleVelocity;
        }

        public double getVelocityProbability() {
                return velocityProbability;
        }

        public void setVelocityProbability(double velocityProbability) {
                if(velocityProbability >=0.0 && velocityProbability <= 1.0)
                        this.velocityProbability = velocityProbability;
        }

        public int getMinVehicleAcceleration() {
                return minVehicleAcceleration;
        }

        public void setMinVehicleAcceleration(int minVehicleAcceleration) {
                if(minVehicleAcceleration >= 0 && minVehicleAcceleration <= this
.maxVehicleAcceleration)
                this.minVehicleAcceleration = minVehicleAcceleration;
        }

        public int getMaxVehicleAcceleration() {
                return maxVehicleAcceleration;
        }

        public void setMaxVehicleAcceleration(int maxVehicleAcceleration) {
                if(maxVehicleAcceleration >= 0 && maxVehicleAcceleration >= this
.minVehicleAcceleration)
                        this.maxVehicleAcceleration = maxVehicleAcceleration;
        }

        public double getAccelerationProbability() {
                return accelerationProbability;
        }

        public void setAccelerationProbability(double accelerationProbability) {
                this.accelerationProbability = accelerationProbability;
        }

        private int maxVehicleVelocity;
        private double velocityProbability;

        //AM > Create a profile for generated vehicle acceleration
        private int minVehicleAcceleration;
        private int maxVehicleAcceleration;
        private double accelerationProbability;

        public Destination()
        {
                waitingQueue = new ArrayList<Vehicle>();
                consumedQueue = new ArrayList<Vehicle>();
        }

        public Destination(String label)
        {
                waitingQueue = new ArrayList<Vehicle>();
                consumedQueue = new ArrayList<Vehicle>();
                this.label = label;
        }

        public String getLabel() {
                return label;
        }

        public void setLabel(String label) {
                this.label = label;
        }

        public int getWaitingQueueLength()
        {
                return waitingQueue.size();
        }

        public int getConsumedQueueLength()
        {
                return consumedQueue.size();
        }


        public SimulationClock getClock() {
                return clock;
        }

        public void setClock(SimulationClock clock) {
                this.clock = clock;
        }


        public boolean addVehicle(Vehicle v) throws VehicleException
        {

                if(v != null)
                {
                        v.setSource(this);

                        Random r = new Random();
```

```java
                        //AM > Set a random velocity
                        if(r.nextDouble() < velocityProbability)
                        {
                                int velocity = r.nextInt((maxVehicleVelocity - m
inVehicleVelocity) + 1) + minVehicleVelocity;
                                v.setVelocity(velocity);
                        }

                        //AM > Set a random acceleration
                        if(r.nextDouble() < accelerationProbability)
                        {
                                int acceleration  = r.nextInt((maxVehicleAcceler
ation - minVehicleAcceleration) + 1) + minVehicleAcceleration;
                                v.setAcceleration(acceleration);
                        }

                        if(!waitingQueue.contains(v))
                                waitingQueue.add(v);
                        return true;
                }
                return false;
        }

        public void setVehicleVelocityProfile(int max, int min, double probabili
ty)
        {
                this.maxVehicleVelocity = max > 1 ? max : 1;
                this.minVehicleVelocity = min > 1 && min < maxVehicleVelocity ?
min : 1;
                if(probability >= 0.0 && probability <= 1.0)
                        this.velocityProbability = probability;
        }

        public void setVehicleAccelerationProfile(int max, int min, double proba
bility)
        {
                this.maxVehicleAcceleration = max > 0 ? max : 0;
                this.minVehicleAcceleration = min >= 0 && min < maxVehicleAccele
ration ? min : 0;
                if(probability >= 0.0 && probability <= 1.0)
                        this.accelerationProbability = probability;
        }

        public void consumeVehicle(Vehicle v)
        {
                if(v != null)
                {
                        if(clock != null)
                                v.setEndTime(clock.getTime());
                        if(!consumedQueue.contains(v))
                                consumedQueue.add(v);
                }
        }

        public Vehicle getWaitingVehicle()
        {
                return waitingQueue.get(0);
        }

        public void releaseVehicle(Vehicle v)
        {
                if(v != null)
                {
                        if(clock != null)
                                v.setStartTime(clock.getTime());
                        waitingQueue.remove(v);
                }
        }

        public void clearConsumedQueue()
        {
                consumedQueue.clear();
        }

        public List<Vehicle> getConsumedVehicles(){
                return consumedQueue;
        }

        @Override
        public String toString()
        {
                return label;
        }
}
```

```java
package core.endpoints;

public class EndPointException extends Exception {
        public EndPointException(String message)
        {
                super(message);
        }
}
```

```
package core.endpoints;

/*
 * AM > Endpoints define connections between Roads and Junctions
 */

public abstract class EndPoint {

}
```

```java
package core.endpoints;

import java.util.List;

import core.network.Lane;

public class JunctionEntry extends EndPoint{
        private List<Lane> lanes;

        public List<Lane> getLanes()
        {
                return lanes;
        }

        public void setLanes(List<Lane> lanes)
        {
                this.lanes = lanes;
        }

        public boolean isConnected() {
                if(lanes != null)
                        return true;
                else
                        return false;
        }
}
```

```java
package core.endpoints;

import java.util.List;

import core.network.Lane;

public class JunctionExit extends EndPoint {
        private List<Lane> lanes;

        public List<Lane> getLanes()
        {
                return lanes;
        }

        public void setLanes(List<Lane> lanes)
        {
                this.lanes = lanes;
        }

        public boolean isConnected() {
                if(lanes != null)
                        return true;
                else
                        return false;
        }
}
```

```java
package core.network;
import java.util.*;

import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;

public class Lane extends Observable{
        private List<Node> nodes;
        private int maxLength;
        private LANE state;
        private List<Lane> transferLanes;

        public enum LANE { MOVE, WAIT, TRANSFER };

        public Lane()
        {
                maxLength = 1;
                nodes = new ArrayList<Node>(maxLength);
                Node node=new Node();
                nodes.add(node);
                //AM > Default lane behavior is to move vehicles along
                state = LANE.MOVE;
        }

        public Lane(int n)
        {
                //AM > Lane cannot have length less than 1
                maxLength = n < 1 ? 1 : n;
                nodes = new ArrayList<Node>(maxLength);
                for(int i = 0; i < maxLength; i++)
                {
                        Node node=new Node();
                        nodes.add(node);
                }
                //AM > Default behavior is to move vehicles along
                state = LANE.MOVE;
        }

        public LANE getState() {
                return state;
        }

        public void setState(LANE state) {
                this.state = state;
        }

        public boolean addVehicle(Vehicle vehicle){
                int length=vehicle.getLength();
                // NC > Vehicle length should be less than max length
                if (length > maxLength) {
                        return false;
                }

                for(int i=0;i<length;i++){
                        if(nodes.get(i).isOccupied()){
                                return false;
                        }
                }
                for(int i=0;i<length;i++){
                        nodes.get(i).setVehicle(vehicle);
                        nodes.get(i).setOccupied(true);
                }
                return true;
        }

        public List<Vehicle> moveVehicles()
        {
                int followingVehicleIndex = maxLength;
                List<Vehicle> exitingVehicles = new ArrayList<Vehicle>();

                for(int i = nodes.size()-1; i >= 0; i--)
                {
                        if(nodes.get(i).isOccupied())
                        {
                                //AM > We get the car and compute its next posit
ion
                                int currentIndex = i;
                                Vehicle vehicle = nodes.get(currentIndex).getVeh
icle();

                                int currentVelocity = vehicle.getVelocity() + ve
hicle.getAcceleration();

                                //AM > Ensure there is no over speeding
                                if(currentVelocity > vehicle.getMax_velocity())
                                        currentVelocity = vehicle.getMax_velocit
y();

                                int predictedIndex = currentIndex+currentVelocit
y;

                                if( predictedIndex >= maxLength && followingVehi
cleIndex == maxLength)
                                {
                                        //AM > Notify observers (i.e Road) that
we have an exiting vehicle
                                        setChanged();
                                        notifyObservers(vehicle);

                                        //AM > If lane state is TRANSFER
                                        if(state == LANE.TRANSFER)
                                        {
                                                //AM > If the transfer fails mak
e the vehicle wait
                                                if(!transferVehicle(vehicle))
                                                {
                                                        int finalIndex = followi
ngVehicleIndex - 1;
                                                        //AM > move vehicles to
the end of the lane
                                                        if(finalIndex != current
Index)
                                                        {
                                                                nodes.get(finalI
ndex).setVehicle(vehicle);
                                                                nodes.get(finalI
ndex).setOccupied(true);
                                                                nodes.get(curren
tIndex).setVehicle(null);
                                                                nodes.get(curren
tIndex).setOccupied(false);
                                                        }
                                                        followingVehicleIndex =
finalIndex;
                                                }
                                                else
                                                {
                                                        nodes.get(currentIndex).
setVehicle(null);
                                                        nodes.get(currentIndex).
setOccupied(false);
                                                }
                                        }
                                        else if(state == LANE.WAIT)
                                        {
                                                int finalIndex = followingVehicl
```

```java
eIndex - 1;
                                                //AM > move vehicles to the end
of the lane
                                                if(finalIndex != currentIndex)
                                                {
                                                        nodes.get(finalIndex).se
tVehicle(vehicle);
                                                        nodes.get(finalIndex).se
tOccupied(true);
                                                        nodes.get(currentIndex).
setVehicle(null);
                                                        nodes.get(currentIndex).
setOccupied(false);
                                                }
                                                followingVehicleIndex = finalInd
ex;
                                        }
                                        //AM > Default action is to move cars
                                        else
                                        {
                                                //AM > Remove the car from the n
etwork
                                                int length = vehicle.getLength()
;
                                                for(int index = 0; index < lengt
h; index++)
                                                {
                                                        nodes.get(currentIndex-i
ndex).setVehicle(null);
                                                        nodes.get(currentIndex-i
ndex).setOccupied(false);
                                                }
                                                if(!exitingVehicles.contains(veh
icle)){
                                                        exitingVehicles.add(vehi
cle);
                                                }
                                        }
                                }
                                else
                                {
                                        int finalIndex = currentIndex;
                                        int finalVelocity = 1;
                                        /*
                                         * AM > Iterate from current position to
 predicted position
                                         *      to check for a clear path
                                         */
                                        int j = 1;
                                        while(j <= currentVelocity)
                                        {
                                                if(!nodes.get(currentIndex + j).
isOccupied())
                                                {
                                                        finalIndex++;
                                                        finalVelocity = j;
                                                }
                                                else
                                                        break;
                                                j++;
                                        }

                                        nodes.get(currentIndex).setOccupied(fals
e);
                                        nodes.get(currentIndex).setVehicle(null)
;

                                        vehicle.setVelocity(finalVelocity);

                                        nodes.get(finalIndex).setOccupied(true);
                                        nodes.get(finalIndex).setVehicle(vehicle
);

                                        followingVehicleIndex = finalIndex;


                                }
                        }
                }

                return exitingVehicles;
        }

        //AM > Primitive visualization of lane state
        public String toString(){
                String state="";
                for(int i=0;i<nodes.size();i++){
                        if(nodes.get(i).isOccupied()){
                                if (nodes.get(i).getVehicle() instanceof Car){
                                        state=state.concat("1");
                                }
                                else if (nodes.get(i).getVehicle() instanceof Bu
s){
                                        state=state.concat("2");
                                }

                        }
                        else{
                                state=state.concat("0");
                        }
                }
                return state;
        }

        public int getVehicleIndex(Vehicle v)
        {
                //NC >> returns the index of the car in the lane. If it doesn't
exists returns -1

                for(int i=nodes.size()-1;i>=0;i--){
                        Vehicle currentVehicle = nodes.get(i).getVehicle();
                        if(currentVehicle != null && currentVehicle.equals(v)){
                                return i;
                        }
                }

                return -1;
        }

        public List<Lane> getTransferLanes() {
                return transferLanes;
        }

        public void setTransferLanes(List<Lane> transferLanes) {
                this.transferLanes = transferLanes;
        }

        //AM > Move exiting vehicles to destination lanes
        public boolean transferVehicle(Vehicle v)
        {
                if(transferLanes == null)
                {
                        return false;
                }
                else
                {
                        for(Lane l : transferLanes)
                        {
                                if(l.addVehicle(v))
                                        return true;
                        }
                        return false;
                }
        }

        public List<Vehicle> getVehicles()
        {
                List<Vehicle> vehicles = new ArrayList<Vehicle>();
                for(Node n : nodes)
                {
                        if(n.isOccupied())
                        {
                                Vehicle v = n.getVehicle();
                                if(!vehicles.contains(v))
                                {
                                        vehicles.add(v);
                                }
                        }
                }
                return vehicles;
        }
}
```

```java
package core.network;

import core.vehicle.Vehicle;;

public class Node {

        private boolean isOccupied;
        private Vehicle vehicle;

        public Node()
        {
                isOccupied = false;
                vehicle = null;
        }

        public boolean isOccupied() {
                return isOccupied;
        }

        public void setOccupied(boolean isOccupied) {
                this.isOccupied = isOccupied;
        }

        public Vehicle getVehicle() {
                return vehicle;
        }

        public void setVehicle(Vehicle vehicle) {
                this.vehicle = vehicle;
        }
}
```

```java
package core.network;

import java.util.ArrayList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;
import java.util.Random;

import core.endpoints.Destination;
import core.endpoints.EndPoint;
import core.endpoints.EndPointException;
import core.endpoints.JunctionEntry;
import core.endpoints.JunctionExit;
import core.network.Lane.LANE;
import core.network.interfaces.Interface;
import core.network.interfaces.InterfaceException;
import core.network.junction.InvalidRouteException;
import core.network.junction.Junction;
import core.network.junction.Junction.JUNCTION;
import core.network.junction.JunctionException;
import core.vehicle.Vehicle;

public class Road implements Observer{
        private List<Lane> lanes;
        private int number_of_lanes;
        private EndPoint source;
        private EndPoint sink;
        private Junction sourceJunction;
        private Junction sinkJunction;
        private JUNCTION face;


        //AM > Create lane(s) and set their length
        public Road(int number_of_lanes, int lane_length)
        {
                //AM > There has to be atleast one lane
                this.number_of_lanes = number_of_lanes < 1 ? 1 : number_of_lanes
;

                lanes = new ArrayList<Lane>();
                for(int i = 0; i < this.number_of_lanes; i++)
                {
                        Lane lane = new Lane(lane_length);
                        lane.addObserver(this);
                        lanes.add(lane);

                }

                //AM > Road isn't connected to any junctions
                sourceJunction = null;
                sinkJunction = null;
        }

        public List<Lane> getLanes() {
                return lanes;
        }

        public void setLanes(List<Lane> lanes) {
                this.lanes = lanes;
        }

        public EndPoint getSource() {
                return source;
        }

        public void setSource(Destination source) {
                this.source = source;
        }

        public void setSource(Junction junction, JUNCTION face) throws Interface
Exception
        {
                //AM > Store junction information
                sourceJunction = junction;

                //AM > Set source to JunctionExit
                JunctionExit juncExit = sourceJunction.getJunctionExit(face);
                juncExit.setLanes(lanes);
        }

        public EndPoint getSink() {
                return sink;
        }

        public void setSink(Destination sink) {
                this.sink = sink;
        }

        public void setSink(Junction junction, JUNCTION face) throws InterfaceEx
ception
        {
                //AM > Store junction information
                sinkJunction = junction;

                //AM > Store interface information
                this.face = face;

                //AM > Set sink to JunctionEntry
                JunctionEntry juncEntry = sinkJunction.getJunctionEntry(this.fac
e);

                juncEntry.setLanes(lanes);
                sink = juncEntry;
        }

        /*
         * AM > Randomly add car to a lane
         * if the lane is occupied add car to the next lane
         * if all lanes are full then return false
         * on successful insertion return true;
         */
        public boolean addVehicle(Vehicle v)
        {
                int randomLane = new Random().nextInt((number_of_lanes - 1) + 1)
 + 1;
                Lane chosenLane = lanes.get(randomLane-1);

                if(chosenLane.addVehicle(v))
                {
                        return true;
                }
                else
                {
                        //AM > Attempt to add a car to another lane.
                        for(Lane l: lanes)
                        {
                                if(!l.equals(chosenLane))
                                {
                                        if(l.addVehicle(v))
                                        {
                                                return true;
                                        }

                                }
                        }
                        // AM > We have exhausted all lanes return false;
                        return false;
                }
```

```java
        }

        public boolean addVehicle(Vehicle v, int laneNumber)
        {
                /*
                 * NC >> Add car to a chosen lane
                 */
                if(laneNumber<1 || laneNumber>lanes.size()){
                        return false;
                }
                Lane chosenLane = lanes.get(laneNumber-1);

                if(chosenLane.addVehicle(v))
                {
                        return true;
                }
                return false;
        }

        public int getVehicleLaneIndex(Vehicle v)
        {
                //NC >> Returns the lane number where the car is on. If the car
is not found it returns -1
                int carIndex=-1;

                for(int i=0;i<lanes.size();i++){
                        carIndex=lanes.get(i).getVehicleIndex(v);
                        if(carIndex!=-1){
                                return i;
                        }
                }

                return -1;
        }

        public int getVehicleNodeIndex(Vehicle v)
        {
                //NC >> Returns the car index where the car is on. If the car is
 not found it returns -1
                int carIndex=-1;

                for(int i=0;i<lanes.size();i++){
                        carIndex=lanes.get(i).getVehicleIndex(v);
                        if(carIndex!=-1){
                                return carIndex;
                        }
                }

                return carIndex;
        }

        /*
         * AM > Pull vehicle from the source and add them to the road. Move the
traffic along.
         *              If vehicles are leaving the network then push them into
the sink
         */
        public void moveTraffic() throws EndPointException{

                //AM > If source is a Destination
                if(source instanceof Destination)
                {
                        Destination origin = (Destination) source;
                        while(origin.getWaitingQueueLength() > 0)
                        {
                                Vehicle v = origin.getWaitingVehicle();
                                //AM > If adding vehicle was successful release
the vehicle from the source
                                if(addVehicle(v))
                                {
                                        origin.releaseVehicle(v);
                                }
                                else
                                {
                                        //AM > Road is full cannot add more vehi
cles
                                        break;
                                }
                        }
                }

                //AM > If sink is a destination, then collect exiting vehicles a
nd add them to the destination
                if(sink instanceof Destination)
                {
                        List<Vehicle> exitingVehicles = new ArrayList<Vehicle>()
;
                        for(Lane l : lanes){
                                exitingVehicles.addAll(l.moveVehicles());
                        }
                        Destination dest = (Destination) sink;
                        for(Vehicle v : exitingVehicles)
                        {
                                dest.consumeVehicle(v);
                        }
                }
                else
                {
                        for(Lane l : lanes)
                        {
                                l.moveVehicles();

                        }
                }
        }


        @Override
        public void update(Observable lane, Object vehicle)
        {
                Vehicle v = (Vehicle) vehicle;
                Lane l = (Lane) lane;

                if(sink instanceof Destination)
                {
                        l.setState(LANE.MOVE);
                }
                else if(sink instanceof JunctionEntry)
                {
                        try
                        {
                                //AM > Get the vehicles destination
                                Destination d = v.getDestination();
                                //AM > Get the destination interface
                                Interface exitInterface = sinkJunction.getExitIn
terface(d);
                                //AM > If signal to interface is green
                                if(sinkJunction.isExitGreen(sinkJunction.getInte
rface(face), exitInterface))
                                {
                                        //AM > Get lanes to junction exit
                                        List<Lane> exitLanes = exitInterface.get
Exit().getLanes();

                                        //AM > Perform lane transfer
                                        l.setTransferLanes(exitLanes);
                                        l.setState(LANE.TRANSFER);
                                }
                                else
                                {
                                        l.setState(LANE.WAIT);
                                }
                        }
                        catch(InvalidRouteException e)
                        {
                                e.printStackTrace();
                        } catch (InterfaceException e) {
                                e.printStackTrace();
                        } catch (JunctionException e) {
                                e.printStackTrace();
                        }
                }
                else
                {
                        l.setState(LANE.MOVE);
                }
        }

        public List<Vehicle> getVehiclesOnRoad(){
                List<Vehicle> vehiclesOnRoad = new ArrayList<Vehicle>();
                for(Lane l : lanes)
                {
                        vehiclesOnRoad.addAll(l.getVehicles());
                }
                return vehiclesOnRoad;
        }
}
```

```java
package core.vehicle;

public class Bus extends Vehicle
{
        private int length;

        public Bus() {
                super();
                this.length=2;
        }

        public Bus(int velocity, int acceleration, int max_velocity) {
                //NC > for busses the length is 2
                super(velocity, acceleration, max_velocity);
                this.length=2;

        }

        @Override
        public int getLength()
        {
                return length;
        }

        @Override
        public Color getColor()
        {
                return color;
        }
}
```

```
package core.vehicle;

public class Car extends Vehicle
{
        private int length;

        public Car() {
                super();
                this.length=1;
        }

        public Car(int velocity, int acceleration, int max_velocity) {
                //NC > for cars the length is 1
                super(velocity, acceleration, max_velocity);
                this.length=1;
        }

        @Override
        public int getLength()
        {
                return length;
        }

        @Override
        public Color getColor()
        {
                return this.color;
        }
}
```

```
package core.vehicle;

public class VehicleException extends Exception {
	public VehicleException(String message) {
		super(message);
	}
}
```

```
package core.vehicle;

import java.util.Random;
import core.endpoints.Destination;

public abstract class Vehicle
{
        enum Color {
                YELLOW,
                RED
        }

        private int velocity;
        private int acceleration;
        private int max_velocity;
        private double decelaration_probability;
        private Destination destination;
        private long start_time;
        private long end_time;
        protected Color color;

        private Destination source;


        public abstract int getLength();
        public abstract Color getColor();

        protected Vehicle()
        {
                this.velocity = 1;
                this.acceleration = 0;
                this.max_velocity = 1;
                this.decelaration_probability = 0.0;
                this.destination = null;
                this.start_time=0;
                this.end_time=0;
        }

        protected Vehicle(int velocity, int acceleration, int max_velocity)
        {
                if(velocity < 1)
                {
                        this.velocity = 1;
                        this.acceleration = 0;
                }
                else
                {
                        this.velocity = velocity;
                        this.acceleration = acceleration;
                }
                this.max_velocity = max_velocity < this.velocity ? this.velocity
 : max_velocity;
                this.decelaration_probability = 0.0;
                this.destination = null;

                this.start_time=0;
                this.end_time=0;
        }

        public Destination getDestination() {
                return destination;
        }

        public void setDestination(Destination destination) throws VehicleExcept
ion {
                if(destination == source)
                        throw new VehicleException("Destination cannot be the same as the sou
rce");

                this.destination = destination;
        }

        public double getDecelaration_probability() {
                return decelaration_probability;
        }

        public void setDecelaration_probability(double decelaration_probability)
 {
                this.decelaration_probability = decelaration_probability;
        }

        public int getMax_velocity() {
                return max_velocity;
        }

        public void setMax_velocity(int max_velocity) {
                this.max_velocity = max_velocity;
        }

        public int getAcceleration() {
                if(new Random().nextDouble() <= decelaration_probability)
                {
                        acceleration = acceleration > 1 ? acceleration -1 : 0;
                }
                return acceleration;
        }

        public void setAcceleration(int acceleration) {
                this.acceleration = acceleration;
        }

        public int getVelocity()
        {
                return velocity;
        }

        public void setVelocity(int velocity)
        {
                this.velocity = velocity;
        }

        public long getStartTime()
        {
                return start_time;
        }

        public void setStartTime(long start_time)
        {
                this.start_time = start_time;
        }

        public long getEndTime()
        {
                return end_time;
        }

        public void setEndTime(long end_time)
        {
                this.end_time = end_time;
        }

        public Destination getSource() {
                return source;
        }

        public void setSource(Destination source) throws VehicleException {
                if(source == destination)
```

```
                        throw new VehicleException("Source cannot be the same as the destinat
ion");

                this.source = source;
        }
}
```

```java
package client.tools;
import java.util.List;

import service.SimulationClock;
import core.endpoints.Destination;
import core.network.Lane;
import core.network.Road;
import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;
import core.vehicle.VehicleException;

public class Main {

        public static void main(String[] args) {
                int laneLength=20;
                int numOfLanes=5;

                //We create 2 roads
                Road r1 = new Road(numOfLanes, laneLength);
                Road r2 = new Road(3, laneLength);

                //We create 3 destinations
                //It will look like this: |A| ----- |B| ----- |C|
                Destination A = new Destination();
                Destination B = new Destination();
                Destination C = new Destination();

                SimulationClock clock = SimulationClock.getInstance();
                A.setClock(clock);
                B.setClock(clock);
                C.setClock(clock);

                r1.setSource(A);
                r1.setSink(B);

                r2.setSource(B);
                r2.setSink(C);

                Vehicle v1 = new Car(2,0,4);
                Vehicle v2 = new Car(1,1,10);

                Vehicle v3 = new Car(1,0,10);
                Vehicle v4 = new Car(1,0,10);
                Vehicle v5 = new Car(1,0,10);

                Vehicle v6 = new Bus(2,0,10);
                Vehicle v7 = new Bus(3,0,10);
                Vehicle v8 = new Bus(1,0,10);

                Vehicle c9 = new Car(3,0,10);

                try {
                        A.addVehicle(v1);
                } catch (VehicleException e1) {
                        e1.printStackTrace();
                }
                System.out.println("Traffic Simulator");

                for(int i = 0; i < 30; i++)
                {
                        System.out.println("\nTick "+clock.getTime());
                        List<Lane> lanes = r1.getLanes();
                        List<Lane> lanes2 = r2.getLanes();

                        int max= lanes.size()>lanes2.size() ? lanes.size() : lan
es2.size();
                        for(int j = 0; j < max; j++){

                                if(j < lanes.size())
                                        System.out.printf("|A|%s|B|", lanes.get(j)
);
                                else
                                        System.out.printf("|A| %s |B|", "no lane");
                                if(j < lanes2.size())
                                        System.out.printf("%s|C|\n", lanes2.get(j
));
                                else
                                        System.out.printf("%s |C|\n", "no lane");
                        }
                        try {
                                r2.moveTraffic();
                                r1.moveTraffic();

                                if(i == 2){
                                        A.addVehicle(v2);
                                }

                                if(i == 3){
                                        A.addVehicle(v3);
                                }

                                if(i == 4){
                                        A.addVehicle(v4);
                                }
                                if(i == 5){
                                        A.addVehicle(v5);
                                }

                                if(i == 6){
                                        A.addVehicle(v6);
                                }

                                if(i == 7){
                                        A.addVehicle(v7);
                                }
                                if(i == 8){
                                        A.addVehicle(v8);
                                }
                                if(i == 9){
                                        A.addVehicle(c9);
                                }

                        } catch (Exception e) {
                                e.printStackTrace();
                        }


                        clock.incrementClock();
                }
        }
}
```

```java
package client.tools;

import service.DemandMatrix;
import service.RoadNetwork;
import service.ReportGenerator;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.network.Road;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.junction.Junction.JUNCTION;
import core.vehicle.Car;

public class Scenario1Reports {

        /*
         * AM > This program generates the reports for
         *              the 4 scenarios of the shopping mall exercise
         */

        public static void main(String[] args) {
                try
                {
                System.out.println("Simulation started");
                int number_of_lanes = 1;
                int lane_length = 10;

                SimulationClock clock = SimulationClock.getInstance();
                clock.setInterval(1000);

                Destination A = new Destination("Athens");
                Destination B = new Destination("Bonitsa");
                Destination C = new Destination("Cesaloniki");
                Destination D = new Destination("Delfoi");

                A.setClock(clock);
                A.setVehicleAccelerationProfile(3, 1, 0.4);
                A.setVehicleVelocityProfile(6, 1, 0.4);
                B.setClock(clock);
                B.setVehicleAccelerationProfile(3, 1, 0.4);
                B.setVehicleVelocityProfile(6, 1, 0.4);
                C.setClock(clock);
                C.setVehicleAccelerationProfile(3, 1, 0.4);
                C.setVehicleVelocityProfile(6, 1, 0.4);
                D.setClock(clock);
                D.setVehicleAccelerationProfile(3, 1, 0.4);
                D.setVehicleVelocityProfile(6, 1, 0.4);

                Junction junc = new Junction();
                RoadNetwork network = new RoadNetwork();

                /*
                 * AM > Road-junction wiring
                 *                         |B|
                 *              ^
                 *                           |
                 *                           |
                 *                              \/
                 *   |A|<----->|junc|<----->|C|
                 *              ^
                 *                           |
                 *                           |
                 *              \/
                 *                         |D|
                 */

                //AM > Roads from A, B, C and D to the junction
                Road ra_j = new Road(number_of_lanes, lane_length);
                ra_j.setSource(A);
                ra_j.setSink(junc,JUNCTION.WEST);
                network.addRoad(ra_j);

                Road rb_j = new Road(number_of_lanes, lane_length);
                rb_j.setSource(B);
                rb_j.setSink(junc, JUNCTION.NORTH);
                network.addRoad(rb_j);

                Road rc_j = new Road(number_of_lanes, lane_length);
                rc_j.setSource(C);
                rc_j.setSink(junc, JUNCTION.EAST);
                network.addRoad(rc_j);

                Road rd_j = new Road(number_of_lanes, lane_length);
                rd_j.setSource(D);
                rd_j.setSink(junc, JUNCTION.SOUTH);
                network.addRoad(rd_j);

                //AM > Roads from the Junction to A, B, C and D
                Road rj_a = new Road(number_of_lanes, lane_length);
                rj_a.setSink(A);
                rj_a.setSource(junc,JUNCTION.WEST);
                network.addRoad(rj_a);

                Road rj_b = new Road(number_of_lanes, lane_length);
                rj_b.setSink(B);
                rj_b.setSource(junc, JUNCTION.NORTH);
                network.addRoad(rj_b);

                Road rj_c = new Road(number_of_lanes, lane_length);
                rj_c.setSink(C);
                rj_c.setSource(junc, JUNCTION.EAST);
                network.addRoad(rj_c);

                Road rj_d = new Road(number_of_lanes, lane_length);
                rj_d.setSink(D);
                rj_d.setSource(junc, JUNCTION.SOUTH);
                network.addRoad(rj_d);


                //AM > Setup routing table
                JunctionRouter juncRouter = new JunctionRouter();
                juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                juncRouter.add(B, junc.getInterface(JUNCTION.NORTH));
                juncRouter.add(C, junc.getInterface(JUNCTION.EAST));
                juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                junc.setRoutingTable(juncRouter);

                //AM > Setup signal scheduler
                junc.setSignalController();
                TrafficSignalScheduler scheduler = new TrafficSignalScheduler();
                scheduler.setSignalInterval(10);
                scheduler.addSignalController(junc.getSignalController());

                DemandMatrix dm = new DemandMatrix();
                dm.addDestination(A);
                dm.addDestination(B);
                dm.addDestination(C);
                dm.addDestination(D);

                dm.initializeMatrix();
                dm.setVehicleType(Car.class);

                dm.setDemand(A, B, 0.3);
                dm.setDemand(A, C, 0.3);
                dm.setDemand(A, D, 0.3);
```

```java
                dm.setDemand(B, A, 0.3);
                dm.setDemand(B, C, 0.3);
                dm.setDemand(B, D, 0.3);

                dm.setDemand(C, B, 0.3);
                dm.setDemand(C, A, 0.3);
                dm.setDemand(C, D, 0.3);

                dm.setDemand(D, B, 0.3);
                dm.setDemand(D, C, 0.3);
                dm.setDemand(D, A, 0.3);

                clock.addObserver(dm);
                clock.addObserver(network);
                clock.addObserver(scheduler);

                clock.startClock();
                System.out.println("Running scenario 1");
                Thread.sleep(1*60*1000);

                clock.pauseClock();

                ReportGenerator report= new ReportGenerator();
                report.addDestination(A);
                report.addDestination(B);
                report.addDestination(C);
                report.addDestination(D);

                String path="Scenario1_report.txt";

                report.saveReport(path);

                System.out.println("Simulation ended");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}
```

```java
package client.tools;

import service.DemandMatrix;
import service.ReportGenerator;
import service.RoadNetwork;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.network.Road;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.junction.Junction.JUNCTION;
import core.vehicle.Car;

public class Scenario2Report {

        public static void main(String[] args) {

                try
                {
                System.out.println("Simulation started");
                int number_of_lanes = 1;
                int lane_length = 10;

                SimulationClock clock = SimulationClock.getInstance();
                clock.setInterval(1000);

                Destination A = new Destination("Athens");
                Destination B = new Destination("Bonitsa");
                Destination C = new Destination("Cesaloniki");
                Destination D = new Destination("Delfoi");

                A.setClock(clock);
                A.setVehicleAccelerationProfile(3, 1, 0.4);
                A.setVehicleVelocityProfile(6, 1, 0.4);
                B.setClock(clock);
                B.setVehicleAccelerationProfile(3, 1, 0.4);
                B.setVehicleVelocityProfile(6, 1, 0.4);
                C.setClock(clock);
                C.setVehicleAccelerationProfile(3, 1, 0.4);
                C.setVehicleVelocityProfile(6, 1, 0.4);
                D.setClock(clock);
                D.setVehicleAccelerationProfile(3, 1, 0.4);
                D.setVehicleVelocityProfile(6, 1, 0.4);

                Junction junc = new Junction();
                RoadNetwork network = new RoadNetwork();

                /*
                 * AM > Road-junction wiring
                 *                          |B|
                 *              ^
                 *                           |
                 *                           |
                 *                               \/
                 *   |A|<----->|junc|<----->|C|
                 *              ^
                 *                           |
                 *                           |
                 *              \/
                 *                          |D|
                 */

                //AM > Roads from A, B, C and D to the junction
                Road ra_j = new Road(number_of_lanes, lane_length);
                ra_j.setSource(A);
                ra_j.setSink(junc,JUNCTION.WEST);
                network.addRoad(ra_j);

                Road rb_j = new Road(number_of_lanes, lane_length);
                rb_j.setSource(B);
                rb_j.setSink(junc, JUNCTION.NORTH);
                network.addRoad(rb_j);

                Road rc_j = new Road(number_of_lanes, lane_length);
                rc_j.setSource(C);
                rc_j.setSink(junc, JUNCTION.EAST);
                network.addRoad(rc_j);

                Road rd_j = new Road(number_of_lanes, lane_length);
                rd_j.setSource(D);
                rd_j.setSink(junc, JUNCTION.SOUTH);
                network.addRoad(rd_j);

                //AM > Roads from the Junction to A, B, C and D
                Road rj_a = new Road(number_of_lanes, lane_length);
                rj_a.setSink(A);
                rj_a.setSource(junc,JUNCTION.WEST);
                network.addRoad(rj_a);

                Road rj_b = new Road(number_of_lanes, lane_length);
                rj_b.setSink(B);
                rj_b.setSource(junc, JUNCTION.NORTH);
                network.addRoad(rj_b);

                Road rj_c = new Road(number_of_lanes, lane_length);
                rj_c.setSink(C);
                rj_c.setSource(junc, JUNCTION.EAST);
                network.addRoad(rj_c);

                Road rj_d = new Road(number_of_lanes, lane_length);
                rj_d.setSink(D);
                rj_d.setSource(junc, JUNCTION.SOUTH);
                network.addRoad(rj_d);


                //AM > Setup routing table
                JunctionRouter juncRouter = new JunctionRouter();
                juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                juncRouter.add(B, junc.getInterface(JUNCTION.NORTH));
                juncRouter.add(C, junc.getInterface(JUNCTION.EAST));
                juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                junc.setRoutingTable(juncRouter);

                //AM > Setup signal scheduler
                junc.setSignalController();
                TrafficSignalScheduler scheduler = new TrafficSignalScheduler();
                scheduler.setSignalInterval(10);
                scheduler.addSignalController(junc.getSignalController());
                DemandMatrix dm = new DemandMatrix();
                dm.addDestination(A);
                dm.addDestination(B);
                dm.addDestination(C);
                dm.addDestination(D);

                dm.initializeMatrix();
                dm.setVehicleType(Car.class);;

                dm.setDemand(A, B, 0.3);
                dm.setDemand(A, C, 0.8);
                dm.setDemand(A, D, 0.3);

                dm.setDemand(B, A, 0.3);
                dm.setDemand(B, C, 0.8);
                dm.setDemand(B, D, 0.3);

                dm.setDemand(C, B, 0.5);
```

```java
                dm.setDemand(C, A, 0.5);
                dm.setDemand(C, D, 0.5);

                dm.setDemand(D, B, 0.3);
                dm.setDemand(D, C, 0.8);
                dm.setDemand(D, A, 0.3);

                clock.addObserver(dm);
                clock.addObserver(network);
                clock.addObserver(scheduler);

                clock.startClock();
                System.out.println("Running scenario 2");
                Thread.sleep(3*60*1000);

                clock.pauseClock();

                ReportGenerator report= new ReportGenerator();
                report.addDestination(A);
                report.addDestination(B);
                report.addDestination(C);
                report.addDestination(D);

                String path="Scenario2_report.txt";

                report.saveReport(path);
                System.out.println("Simulation ended");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }

}
```

```java
package client.tools;

import service.DemandMatrix;
import service.ReportGenerator;
import service.RoadNetwork;
import service.SimulationClock;
import service.TrafficSignalScheduler;
import core.endpoints.Destination;
import core.network.Road;
import core.network.junction.Junction;
import core.network.junction.JunctionRouter;
import core.network.junction.Junction.JUNCTION;
import core.vehicle.Car;

public class Scenario3Report {

public static void main(String[] args) {

                try
                {
                System.out.println("Simulation started");
                int number_of_lanes = 1;
                int lane_length = 10;

                SimulationClock clock = SimulationClock.getInstance();
                clock.setInterval(1000);

                Destination A = new Destination("Athens");
                Destination B = new Destination("Bonitsa");
                Destination C = new Destination("Cesaloniki");
                Destination D = new Destination("Delfoi");

                A.setClock(clock);
                A.setVehicleAccelerationProfile(3, 1, 0.4);
                A.setVehicleVelocityProfile(6, 1, 0.4);
                B.setClock(clock);
                B.setVehicleAccelerationProfile(3, 1, 0.4);
                B.setVehicleVelocityProfile(6, 1, 0.4);
                C.setClock(clock);
                C.setVehicleAccelerationProfile(3, 1, 0.4);
                C.setVehicleVelocityProfile(6, 1, 0.4);
                D.setClock(clock);
                D.setVehicleAccelerationProfile(3, 1, 0.4);
                D.setVehicleVelocityProfile(6, 1, 0.4);

                Junction junc = new Junction();
                RoadNetwork network = new RoadNetwork();

                /*
                 * AM > Road-junction wiring
                 *                         |B|
                 *              ^
                 *                          |
                 *                          |
                 *                              \/
                 *   |A|<----->|junc|<----->|C|
                 *              ^
                 *                          |
                 *                          |
                 *              \/
                 *                         |D|
                 */

                //AM > Roads from A, B, C and D to the junction
                Road ra_j = new Road(number_of_lanes, lane_length);
                ra_j.setSource(A);
                ra_j.setSink(junc,JUNCTION.WEST);
                network.addRoad(ra_j);

                Road rb_j = new Road(number_of_lanes, lane_length);
                rb_j.setSource(B);
                rb_j.setSink(junc, JUNCTION.NORTH);
                network.addRoad(rb_j);

                Road rc_j = new Road(number_of_lanes, lane_length);
                rc_j.setSource(C);
                rc_j.setSink(junc, JUNCTION.EAST);
                network.addRoad(rc_j);

                Road rd_j = new Road(number_of_lanes, lane_length);
                rd_j.setSource(D);
                rd_j.setSink(junc, JUNCTION.SOUTH);
                network.addRoad(rd_j);

                //AM > Roads from the Junction to A, B, C and D
                Road rj_a = new Road(number_of_lanes, lane_length);
                rj_a.setSink(A);
                rj_a.setSource(junc,JUNCTION.WEST);
                network.addRoad(rj_a);

                Road rj_b = new Road(number_of_lanes, lane_length);
                rj_b.setSink(B);
                rj_b.setSource(junc, JUNCTION.NORTH);
                network.addRoad(rj_b);

                Road rj_c = new Road(number_of_lanes, lane_length);
                rj_c.setSink(C);
                rj_c.setSource(junc, JUNCTION.EAST);
                network.addRoad(rj_c);

                Road rj_d = new Road(number_of_lanes, lane_length);
                rj_d.setSink(D);
                rj_d.setSource(junc, JUNCTION.SOUTH);
                network.addRoad(rj_d);


                //AM > Setup routing table
                JunctionRouter juncRouter = new JunctionRouter();
                juncRouter.add(A, junc.getInterface(JUNCTION.WEST));
                juncRouter.add(B, junc.getInterface(JUNCTION.NORTH));
                juncRouter.add(C, junc.getInterface(JUNCTION.EAST));
                juncRouter.add(D,  junc.getInterface(JUNCTION.SOUTH));
                junc.setRoutingTable(juncRouter);

                //AM > Setup signal scheduler
                junc.setSignalController();
                TrafficSignalScheduler scheduler = new TrafficSignalScheduler();
                scheduler.setSignalInterval(10);
                scheduler.addSignalController(junc.getSignalController());
                DemandMatrix dm = new DemandMatrix();
                dm.addDestination(A);
                dm.addDestination(B);
                dm.addDestination(C);
                dm.addDestination(D);
                dm.initializeMatrix();
                dm.setVehicleType(Car.class);

                dm.setDemand(A, B, 0.5);
                dm.setDemand(A, C, 0.1);
                dm.setDemand(A, D, 0.3);

                dm.setDemand(B, A, 0.3);
                dm.setDemand(B, C, 0.1);
                dm.setDemand(B, D, 0.3);

                dm.setDemand(C, B, 0.3);
                dm.setDemand(C, A, 0.3);
```

```java
                dm.setDemand(C, D, 0.3);

                dm.setDemand(D, B, 0.5);
                dm.setDemand(D, C, 0.1);
                dm.setDemand(D, A, 0.3);

                clock.addObserver(dm);
                clock.addObserver(network);
                clock.addObserver(scheduler);

                clock.startClock();
                System.out.println("Running scenario 3");
                Thread.sleep(3*60*1000);

                clock.pauseClock();

                ReportGenerator report= new ReportGenerator();
                report.addDestination(A);
                report.addDestination(B);
                report.addDestination(C);
                report.addDestination(D);

                String path="Scenario3_report.txt";

                report.saveReport(path);
                System.out.println("Simulation ended");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}
```

```java
package core.endpoints;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import service.SimulationClock;
import core.vehicle.Vehicle;
import core.vehicle.VehicleException;

/*
 * AM > This class represents a Destination.
 *          Destinations are spawn points where cars originate and terminate
 */
public class Destination extends EndPoint {

        private List<Vehicle> waitingQueue;
        private List<Vehicle> consumedQueue;
        private SimulationClock clock;
        private String label;

        //AM > Create a profile for generated vehicle velocity
        private int minVehicleVelocity;
        public int getMinVehicleVelocity() {
                return minVehicleVelocity;
        }

        public void setMinVehicleVelocity(int minVehicleVelocity) {
                if(minVehicleVelocity >= 1 && minVehicleVelocity <= this.maxVehi
cleVelocity)
                        this.minVehicleVelocity = minVehicleVelocity;
        }

        public int getMaxVehicleVelocity() {
                return maxVehicleVelocity;
        }

        public void setMaxVehicleVelocity(int maxVehicleVelocity) {
                if(maxVehicleVelocity >= 1 && maxVehicleVelocity >= this.minVehi
cleVelocity)
                        this.maxVehicleVelocity = maxVehicleVelocity;
        }

        public double getVelocityProbability() {
                return velocityProbability;
        }

        public void setVelocityProbability(double velocityProbability) {
                if(velocityProbability >=0.0 && velocityProbability <= 1.0)
                        this.velocityProbability = velocityProbability;
        }

        public int getMinVehicleAcceleration() {
                return minVehicleAcceleration;
        }

        public void setMinVehicleAcceleration(int minVehicleAcceleration) {
                if(minVehicleAcceleration >= 0 && minVehicleAcceleration <= this
.maxVehicleAcceleration)
                this.minVehicleAcceleration = minVehicleAcceleration;
        }

        public int getMaxVehicleAcceleration() {
                return maxVehicleAcceleration;
        }

        public void setMaxVehicleAcceleration(int maxVehicleAcceleration) {
                if(maxVehicleAcceleration >= 0 && maxVehicleAcceleration >= this
.minVehicleAcceleration)
                        this.maxVehicleAcceleration = maxVehicleAcceleration;
        }

        public double getAccelerationProbability() {
                return accelerationProbability;
        }

        public void setAccelerationProbability(double accelerationProbability) {
                this.accelerationProbability = accelerationProbability;
        }

        private int maxVehicleVelocity;
        private double velocityProbability;

        //AM > Create a profile for generated vehicle acceleration
        private int minVehicleAcceleration;
        private int maxVehicleAcceleration;
        private double accelerationProbability;

        public Destination()
        {
                waitingQueue = new ArrayList<Vehicle>();
                consumedQueue = new ArrayList<Vehicle>();
        }

        public Destination(String label)
        {
                waitingQueue = new ArrayList<Vehicle>();
                consumedQueue = new ArrayList<Vehicle>();
                this.label = label;
        }

        public String getLabel() {
                return label;
        }

        public void setLabel(String label) {
                this.label = label;
        }

        public int getWaitingQueueLength()
        {
                return waitingQueue.size();
        }

        public int getConsumedQueueLength()
        {
                return consumedQueue.size();
        }


        public SimulationClock getClock() {
                return clock;
        }

        public void setClock(SimulationClock clock) {
                this.clock = clock;
        }


        public boolean addVehicle(Vehicle v) throws VehicleException
        {

                if(v != null)
                {
                        v.setSource(this);

                        Random r = new Random();
```

```java
                        //AM > Set a random velocity
                        if(r.nextDouble() < velocityProbability)
                        {
                                int velocity = r.nextInt((maxVehicleVelocity - m
inVehicleVelocity) + 1) + minVehicleVelocity;
                                v.setVelocity(velocity);
                        }

                        //AM > Set a random acceleration
                        if(r.nextDouble() < accelerationProbability)
                        {
                                int acceleration  = r.nextInt((maxVehicleAcceler
ation - minVehicleAcceleration) + 1) + minVehicleAcceleration;
                                v.setAcceleration(acceleration);
                        }

                        if(!waitingQueue.contains(v))
                                waitingQueue.add(v);
                        return true;
                }
                return false;
        }

        public void setVehicleVelocityProfile(int max, int min, double probabili
ty)
        {
                this.maxVehicleVelocity = max > 1 ? max : 1;
                this.minVehicleVelocity = min > 1 && min < maxVehicleVelocity ?
min : 1;
                if(probability >= 0.0 && probability <= 1.0)
                        this.velocityProbability = probability;
        }

        public void setVehicleAccelerationProfile(int max, int min, double proba
bility)
        {
                this.maxVehicleAcceleration = max > 0 ? max : 0;
                this.minVehicleAcceleration = min >= 0 && min < maxVehicleAccele
ration ? min : 0;
                if(probability >= 0.0 && probability <= 1.0)
                        this.accelerationProbability = probability;
        }

        public void consumeVehicle(Vehicle v)
        {
                if(v != null)
                {
                        if(clock != null)
                                v.setEndTime(clock.getTime());
                        if(!consumedQueue.contains(v))
                                consumedQueue.add(v);
                }
        }

        public Vehicle getWaitingVehicle()
        {
                return waitingQueue.get(0);
        }

        public void releaseVehicle(Vehicle v)
        {
                if(v != null)
                {
                        if(clock != null)
                                v.setStartTime(clock.getTime());
                        waitingQueue.remove(v);
                }
        }

        public void clearConsumedQueue()
        {
                consumedQueue.clear();
        }

        public List<Vehicle> getConsumedVehicles(){
                return consumedQueue;
        }

        @Override
        public String toString()
        {
                return label;
        }
}
```

```java
package core.endpoints;

public class EndPointException extends Exception {
        public EndPointException(String message)
        {
                super(message);
        }
}
```

```java
package core.endpoints;

/*
 * AM > Endpoints define connections between Roads and Junctions
 */

public abstract class EndPoint {

}
```

```
package core.endpoints;

import java.util.List;

import core.network.Lane;

public class JunctionEntry extends EndPoint{
        private List<Lane> lanes;

        public List<Lane> getLanes()
        {
                return lanes;
        }

        public void setLanes(List<Lane> lanes)
        {
                this.lanes = lanes;
        }

        public boolean isConnected() {
                if(lanes != null)
                        return true;
                else
                        return false;
        }
}
```

```
package core.endpoints;

import java.util.List;

import core.network.Lane;

public class JunctionExit extends EndPoint {
        private List<Lane> lanes;

        public List<Lane> getLanes()
        {
                return lanes;
        }

        public void setLanes(List<Lane> lanes)
        {
                this.lanes = lanes;
        }

        public boolean isConnected() {
                if(lanes != null)
                        return true;
                else
                        return false;
        }
}
```

```java
package core.network;
import java.util.*;

import core.vehicle.Bus;
import core.vehicle.Car;
import core.vehicle.Vehicle;

public class Lane extends Observable{
        private List<Node> nodes;
        private int maxLength;
        private LANE state;
        private List<Lane> transferLanes;

        public enum LANE { MOVE, WAIT, TRANSFER };

        public Lane()
        {
                maxLength = 1;
                nodes = new ArrayList<Node>(maxLength);
                Node node=new Node();
                nodes.add(node);
                //AM > Default lane behavior is to move vehicles along
                state = LANE.MOVE;
        }

        public Lane(int n)
        {
                //AM > Lane cannot have length less than 1
                maxLength = n < 1 ? 1 : n;
                nodes = new ArrayList<Node>(maxLength);
                for(int i = 0; i < maxLength; i++)
                {
                        Node node=new Node();
                        nodes.add(node);
                }
                //AM > Default behavior is to move vehicles along
                state = LANE.MOVE;
        }

        public LANE getState() {
                return state;
        }

        public void setState(LANE state) {
                this.state = state;
        }

        public boolean addVehicle(Vehicle vehicle){
                int length=vehicle.getLength();
                // NC > Vehicle length should be less than max length
                if (length > maxLength) {
                        return false;
                }

                for(int i=0;i<length;i++){
                        if(nodes.get(i).isOccupied()){
                                return false;
                        }
                }
                for(int i=0;i<length;i++){
                        nodes.get(i).setVehicle(vehicle);
                        nodes.get(i).setOccupied(true);
                }
                return true;
        }

        public List<Vehicle> moveVehicles()
        {
                int followingVehicleIndex = maxLength;
                List<Vehicle> exitingVehicles = new ArrayList<Vehicle>();

                for(int i = nodes.size()-1; i >= 0; i--)
                {
                        if(nodes.get(i).isOccupied())
                        {
                                //AM > We get the car and compute its next posit
ion
                                int currentIndex = i;
                                Vehicle vehicle = nodes.get(currentIndex).getVeh
icle();

                                int currentVelocity = vehicle.getVelocity() + ve
hicle.getAcceleration();

                                //AM > Ensure there is no over speeding
                                if(currentVelocity > vehicle.getMax_velocity())
                                        currentVelocity = vehicle.getMax_velocit
y();

                                int predictedIndex = currentIndex+currentVelocit
y;

                                if( predictedIndex >= maxLength && followingVehi
cleIndex == maxLength)
                                {
                                        //AM > Notify observers (i.e Road) that
we have an exiting vehicle
                                        setChanged();
                                        notifyObservers(vehicle);

                                        //AM > If lane state is TRANSFER
                                        if(state == LANE.TRANSFER)
                                        {
                                                //AM > If the transfer fails mak
e the vehicle wait
                                                if(!transferVehicle(vehicle))
                                                {
                                                        int finalIndex = followi
ngVehicleIndex - 1;
                                                        //AM > move vehicles to
the end of the lane
                                                        if(finalIndex != current
Index)
                                                        {
                                                                nodes.get(finalI
ndex).setVehicle(vehicle);
                                                                nodes.get(finalI
ndex).setOccupied(true);
                                                                nodes.get(curren
tIndex).setVehicle(null);
                                                                nodes.get(curren
tIndex).setOccupied(false);
                                                        }
                                                        followingVehicleIndex =
finalIndex;
                                                }
                                                else
                                                {
                                                        nodes.get(currentIndex).
setVehicle(null);
                                                        nodes.get(currentIndex).
setOccupied(false);
                                                }
                                        }
                                        else if(state == LANE.WAIT)
                                        {
                                                int finalIndex = followingVehicl
```

```java
eIndex - 1;
                                                        //AM > move vehicles to the end
of the lane
                                                        if(finalIndex != currentIndex)
                                                        {
                                                                nodes.get(finalIndex).se
tVehicle(vehicle);
                                                                nodes.get(finalIndex).se
tOccupied(true);
                                                                nodes.get(currentIndex).
setVehicle(null);
                                                                nodes.get(currentIndex).
setOccupied(false);
                                                        }
                                                        followingVehicleIndex = finalInd
ex;
                                                }
                                                //AM > Default action is to move cars
                                                else
                                                {
                                                        //AM > Remove the car from the n
etwork
                                                        int length = vehicle.getLength()
;
                                                        for(int index = 0; index < lengt
h; index++)
                                                        {
                                                                nodes.get(currentIndex-i
ndex).setVehicle(null);
                                                                nodes.get(currentIndex-i
ndex).setOccupied(false);
                                                        }
                                                        if(!exitingVehicles.contains(veh
icle)){
                                                                exitingVehicles.add(vehi
cle);
                                                        }
                                                }
                                }
                                else
                                {
                                        int finalIndex = currentIndex;
                                        int finalVelocity = 1;
                                        /*
                                         * AM > Iterate from current position to
 predicted position
                                         *      to check for a clear path
                                         */
                                        int j = 1;
                                        while(j <= currentVelocity)
                                        {
                                                if(!nodes.get(currentIndex + j).
isOccupied())
                                                {
                                                        finalIndex++;
                                                        finalVelocity = j;
                                                }
                                                else
                                                        break;
                                                j++;
                                        }

                                        nodes.get(currentIndex).setOccupied(fals
e);
                                        nodes.get(currentIndex).setVehicle(null)
;

                                        vehicle.setVelocity(finalVelocity);

                                        nodes.get(finalIndex).setOccupied(true);
                                        nodes.get(finalIndex).setVehicle(vehicle
);

                                        followingVehicleIndex = finalIndex;


                                }
                        }
                }

                return exitingVehicles;
        }

        //AM > Primitive visualization of lane state
        public String toString(){
                String state="";
                for(int i=0;i<nodes.size();i++){
                        if(nodes.get(i).isOccupied()){
                                if (nodes.get(i).getVehicle() instanceof Car){
                                        state=state.concat("1");
                                }
                                else if (nodes.get(i).getVehicle() instanceof Bu
s){
                                        state=state.concat("2");
                                }

                        }
                        else{
                                state=state.concat("0");
                        }
                }
                return state;
        }

        public int getVehicleIndex(Vehicle v)
        {
                //NC >> returns the index of the car in the lane. If it doesn’t
exists returns -1
                for(int i=nodes.size()-1;i>=0;i--){
                        Vehicle currentVehicle = nodes.get(i).getVehicle();
                        if(currentVehicle != null && currentVehicle.equals(v)){
                                return i;
                        }
                }

                return -1;
        }

        public List<Lane> getTransferLanes() {
                return transferLanes;
        }

        public void setTransferLanes(List<Lane> transferLanes) {
                this.transferLanes = transferLanes;
        }

        //AM > Move exiting vehicles to destination lanes
        public boolean transferVehicle(Vehicle v)
        {
                if(transferLanes == null)
                {
                        return false;
                }
                else
                {
                        for(Lane l : transferLanes)
                        {
                                if(l.addVehicle(v))
                                        return true;
                        }
                        return false;
                }
        }

        public List<Vehicle> getVehicles()
        {
                List<Vehicle> vehicles = new ArrayList<Vehicle>();
                for(Node n : nodes)
                {
                        if(n.isOccupied())
                        {
                                Vehicle v = n.getVehicle();
                                if(!vehicles.contains(v))
                                {
                                        vehicles.add(v);
                                }
                        }
                }
                return vehicles;
        }
}
```

```
package core.network;

import core.vehicle.Vehicle;;

public class Node {

	private boolean isOccupied;
	private Vehicle vehicle;

	public Node()
	{
		isOccupied = false;
		vehicle = null;
	}

	public boolean isOccupied() {
		return isOccupied;
	}

	public void setOccupied(boolean isOccupied) {
		this.isOccupied = isOccupied;
	}

	public Vehicle getVehicle() {
		return vehicle;
	}

	public void setVehicle(Vehicle vehicle) {
		this.vehicle = vehicle;
	}
}
```

```java
package core.network;

import java.util.ArrayList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;
import java.util.Random;

import core.endpoints.Destination;
import core.endpoints.EndPoint;
import core.endpoints.EndPointException;
import core.endpoints.JunctionEntry;
import core.endpoints.JunctionExit;
import core.network.Lane.LANE;
import core.network.interfaces.Interface;
import core.network.interfaces.InterfaceException;
import core.network.junction.InvalidRouteException;
import core.network.junction.Junction;
import core.network.junction.Junction.JUNCTION;
import core.network.junction.JunctionException;
import core.vehicle.Vehicle;

public class Road implements Observer{
        private List<Lane> lanes;
        private int number_of_lanes;
        private EndPoint source;
        private EndPoint sink;
        private Junction sourceJunction;
        private Junction sinkJunction;
        private JUNCTION face;


        //AM > Create lane(s) and set their length
        public Road(int number_of_lanes, int lane_length)
        {
                //AM > There has to be atleast one lane
                this.number_of_lanes = number_of_lanes < 1 ? 1 : number_of_lanes
;

                lanes = new ArrayList<Lane>();
                for(int i = 0; i < this.number_of_lanes; i++)
                {
                        Lane lane = new Lane(lane_length);
                        lane.addObserver(this);
                        lanes.add(lane);

                }

                //AM > Road isn't connected to any junctions
                sourceJunction = null;
                sinkJunction = null;
        }

        public List<Lane> getLanes() {
                return lanes;
        }

        public void setLanes(List<Lane> lanes) {
                this.lanes = lanes;
        }

        public EndPoint getSource() {
                return source;
        }

        public void setSource(Destination source) {
                this.source = source;
        }

        public void setSource(Junction junction, JUNCTION face) throws Interface
Exception
        {
                //AM > Store junction information
                sourceJunction = junction;

                //AM > Set source to JunctionExit
                JunctionExit juncExit = sourceJunction.getJunctionExit(face);
                juncExit.setLanes(lanes);
        }

        public EndPoint getSink() {
                return sink;
        }

        public void setSink(Destination sink) {
                this.sink = sink;
        }

        public void setSink(Junction junction, JUNCTION face) throws InterfaceEx
ception
        {
                //AM > Store junction information
                sinkJunction = junction;

                //AM > Store interface information
                this.face = face;

                //AM > Set sink to JunctionEntry
                JunctionEntry juncEntry = sinkJunction.getJunctionEntry(this.fac
e);
                juncEntry.setLanes(lanes);
                sink = juncEntry;
        }

        /*
         * AM > Randomly add car to a lane
         * if the lane is occupied add car to the next lane
         * if all lanes are full then return false
         * on successful insertion return true;
         */
        public boolean addVehicle(Vehicle v)
        {
                int randomLane = new Random().nextInt((number_of_lanes - 1) + 1)
 + 1;
                Lane chosenLane = lanes.get(randomLane-1);

                if(chosenLane.addVehicle(v))
                {
                        return true;
                }
                else
                {
                        //AM > Attempt to add a car to another lane.
                        for(Lane l: lanes)
                        {
                                if(!l.equals(chosenLane))
                                {
                                        if(l.addVehicle(v))
                                        {
                                                return true;
                                        }

                                }
                        }
                        // AM > We have exhausted all lanes return false;
                        return false;
                }
```

```java
        }

        public boolean addVehicle(Vehicle v, int laneNumber)
        {
                /*
                 * NC >> Add car to a chosen lane
                 */
                if(laneNumber<1 || laneNumber>lanes.size()){
                        return false;
                }
                Lane chosenLane = lanes.get(laneNumber-1);

                if(chosenLane.addVehicle(v))
                {
                        return true;
                }
                return false;
        }

        public int getVehicleLaneIndex(Vehicle v)
        {
                //NC >> Returns the lane number where the car is on. If the car
is not found it returns -1
                int carIndex=-1;

                for(int i=0;i<lanes.size();i++){
                        carIndex=lanes.get(i).getVehicleIndex(v);
                        if(carIndex!=-1){
                                return i;
                        }
                }

                return -1;
        }

        public int getVehicleNodeIndex(Vehicle v)
        {
                //NC >> Returns the car index where the car is on. If the car is
 not found it returns -1
                int carIndex=-1;

                for(int i=0;i<lanes.size();i++){
                        carIndex=lanes.get(i).getVehicleIndex(v);
                        if(carIndex!=-1){
                                return carIndex;
                        }
                }

                return carIndex;
        }

        /*
         * AM > Pull vehicle from the source and add them to the road. Move the
traffic along.
         *              If vehicles are leaving the network then push them into
the sink
         */
        public void moveTraffic() throws EndPointException{

                //AM > If source is a Destination
                if(source instanceof Destination)
                {
                        Destination origin = (Destination) source;
                        while(origin.getWaitingQueueLength() > 0)
                        {
                                Vehicle v = origin.getWaitingVehicle();
                                //AM > If adding vehicle was successful release
the vehicle from the source
                                if(addVehicle(v))
                                {
                                        origin.releaseVehicle(v);
                                }
                                else
                                {
                                        //AM > Road is full cannot add more vehi
cles
                                        break;
                                }
                        }
                }

                //AM > If sink is a destination, then collect exiting vehicles a
nd add them to the destination
                if(sink instanceof Destination)
                {
                        List<Vehicle> exitingVehicles = new ArrayList<Vehicle>()
;
                        for(Lane l : lanes){
                                exitingVehicles.addAll(l.moveVehicles());
                        }
                        Destination dest = (Destination) sink;
                        for(Vehicle v : exitingVehicles)
                        {
                                dest.consumeVehicle(v);
                        }
                }
                else
                {
                        for(Lane l : lanes)
                        {
                                l.moveVehicles();

                        }
                }
        }


        @Override
        public void update(Observable lane, Object vehicle)
        {
                Vehicle v = (Vehicle) vehicle;
                Lane l = (Lane) lane;

                if(sink instanceof Destination)
                {
                        l.setState(LANE.MOVE);
                }
                else if(sink instanceof JunctionEntry)
                {
                        try
                        {
                                //AM > Get the vehicles destination
                                Destination d = v.getDestination();
                                //AM > Get the destination interface
                                Interface exitInterface = sinkJunction.getExitIn
terface(d);
                                //AM > If signal to interface is green
                                if(sinkJunction.isExitGreen(sinkJunction.getInte
rface(face), exitInterface))
                                {
                                        //AM > Get lanes to junction exit
                                        List<Lane> exitLanes = exitInterface.get
Exit().getLanes();

                                        //AM > Perform lane transfer
                                        l.setTransferLanes(exitLanes);
                                        l.setState(LANE.TRANSFER);
                                }
                                else
                                {
                                        l.setState(LANE.WAIT);
                                }
                        }
                        catch(InvalidRouteException e)
                        {
                                e.printStackTrace();
                        } catch (InterfaceException e) {
                                e.printStackTrace();
                        } catch (JunctionException e) {
                                e.printStackTrace();
                        }
                }
                else
                {
                        l.setState(LANE.MOVE);
                }
        }

        public List<Vehicle> getVehiclesOnRoad(){
                List<Vehicle> vehiclesOnRoad = new ArrayList<Vehicle>();
                for(Lane l : lanes)
                {
                        vehiclesOnRoad.addAll(l.getVehicles());
                }
                return vehiclesOnRoad;
        }
}
```

```
package core.vehicle;

public class Bus extends Vehicle
{
        private int length;

        public Bus() {
                super();
                this.length=2;
        }

        public Bus(int velocity, int acceleration, int max_velocity) {
                //NC > for busses the length is 2
                super(velocity, acceleration, max_velocity);
                this.length=2;

        }

        @Override
        public int getLength()
        {
                return length;
        }

        @Override
        public Color getColor()
        {
                return color;
        }
}
```

```java
package core.vehicle;

public class Car extends Vehicle
{
        private int length;

        public Car() {
                super();
                this.length=1;
        }

        public Car(int velocity, int acceleration, int max_velocity) {
                //NC > for cars the length is 1
                super(velocity, acceleration, max_velocity);
                this.length=1;
        }

        @Override
        public int getLength()
        {
                return length;
        }

        @Override
        public Color getColor()
        {
                return this.color;
        }
}
```

```
package core.vehicle;

public class VehicleException extends Exception {
        public VehicleException(String message) {
                super(message);
        }
}
```

```
package core.vehicle;

import java.util.Random;
import core.endpoints.Destination;

public abstract class Vehicle
{
        enum Color {
                YELLOW,
                RED
        }

        private int velocity;
        private int acceleration;
        private int max_velocity;
        private double decelaration_probability;
        private Destination destination;
        private long start_time;
        private long end_time;
        protected Color color;

        private Destination source;


        public abstract int getLength();
        public abstract Color getColor();

        protected Vehicle()
        {
                this.velocity = 1;
                this.acceleration = 0;
                this.max_velocity = 1;
                this.decelaration_probability = 0.0;
                this.destination = null;
                this.start_time=0;
                this.end_time=0;
        }

        protected Vehicle(int velocity, int acceleration, int max_velocity)
        {
                if(velocity < 1)
                {
                        this.velocity = 1;
                        this.acceleration = 0;
                }
                else
                {
                        this.velocity = velocity;
                        this.acceleration = acceleration;
                }
                this.max_velocity = max_velocity < this.velocity ? this.velocity
 : max_velocity;
                this.decelaration_probability = 0.0;
                this.destination = null;

                this.start_time=0;
                this.end_time=0;
        }

        public Destination getDestination() {
                return destination;
        }

        public void setDestination(Destination destination) throws VehicleExcept
ion {
                if(destination == source)
                        throw new VehicleException("Destination cannot be the same as the sou
rce");

                this.destination = destination;
        }

        public double getDecelaration_probability() {
                return decelaration_probability;
        }

        public void setDecelaration_probability(double decelaration_probability)
 {
                this.decelaration_probability = decelaration_probability;
        }

        public int getMax_velocity() {
                return max_velocity;
        }

        public void setMax_velocity(int max_velocity) {
                this.max_velocity = max_velocity;
        }

        public int getAcceleration() {
                if(new Random().nextDouble() <= decelaration_probability)
                {
                        acceleration = acceleration > 1 ? acceleration -1 : 0;
                }
                return acceleration;
        }

        public void setAcceleration(int acceleration) {
                this.acceleration = acceleration;
        }

        public int getVelocity()
        {
                return velocity;
        }

        public void setVelocity(int velocity)
        {
                this.velocity = velocity;
        }

        public long getStartTime()
        {
                return start_time;
        }

        public void setStartTime(long start_time)
        {
                this.start_time = start_time;
        }

        public long getEndTime()
        {
                return end_time;
        }

        public void setEndTime(long end_time)
        {
                this.end_time = end_time;
        }

        public Destination getSource() {
                return source;
        }

        public void setSource(Destination source) throws VehicleException {
                if(source == destination)
```

```
                        throw new VehicleException("Source cannot be the same as the destinat
ion");

                this.source = source;
        }
}
```

```
package core.network.interfaces;

public class InterfaceException extends Exception {

        public InterfaceException(String message)
        {
                super(message);
        }
}
```

```
package core.network.interfaces;

import core.endpoints.JunctionEntry;
import core.endpoints.JunctionExit;

public class Interface
{

        private JunctionExit exit;
        private JunctionEntry entry;
        private boolean enabled;
        private TrafficSignal signals;

        public Interface()
        {
                //AM > Enable the interface
                this.enabled = true;
                exit = new JunctionExit();
                entry = new JunctionEntry();
        }

        public void configureSignal(Interface leftTurn, Interface forward, Inter
face rightTurn)
        {
                //AM > Setup traffic lights
                signals = new TrafficSignal(leftTurn, forward, rightTurn);
        }
        public JunctionEntry getEntry() {
                return entry;
        }

        public JunctionExit getExit() {
                return exit;
        }

        public void enableInterface() {
                this.enabled = true;
        }

        public void disableInterface() {
                this.enabled = false;
        }

        public boolean isEnabled() {
                return enabled;
        }

        public boolean getSignalState(Interface exitInterface) throws InterfaceE
xception {
                return signals.getSignal(exitInterface);
        }

        public void setSignalState(Interface exitInterface, boolean state) throw
s InterfaceException
        {
                signals.setSignal(exitInterface, state);
        }

        public TrafficSignal getSignals() {
                return signals;
        }

        public void setSignals(TrafficSignal signals) {
                this.signals = signals;
        }
}
```

```
package core.network.interfaces;

import java.util.HashMap;

public class TrafficSignal {

        private HashMap<Interface, Boolean> lights;

        public TrafficSignal(Interface leftTurn, Interface forward, Interface ri
ghtTurn)
        {
                lights = new HashMap<Interface, Boolean>();

                lights.put(leftTurn, false);
                lights.put(rightTurn, false);
                lights.put(forward, false);
        }

        public boolean getSignal(Interface face) throws InterfaceException
        {
                if(lights.containsKey(face))
                {
                        return lights.get(face);
                }
                else
                {
                        throw new InterfaceException("Unknown Interface");
                }
        }

        public void setSignal(Interface face, boolean state) throws InterfaceExc
eption
        {
                if(lights.containsKey(face))
                {
                        lights.put(face, state);
                }
                else
                {
                        throw new InterfaceException("Unknown Interface");
                }
        }
}
```

```
package core.network.junction;

public class InvalidRouteException extends Exception {
        public InvalidRouteException(String message)
        {
                super(message);
        }
}
```

```
package core.network.junction;

public class JunctionException extends Exception {
        public JunctionException(String message)
        {
                super(message);
        }
}
```

```
package core.network.junction;

import core.endpoints.Destination;
import core.endpoints.JunctionEntry;
import core.endpoints.JunctionExit;
import core.network.interfaces.Interface;
import core.network.interfaces.InterfaceException;

public class Junction {
        private Interface west;
        private Interface east;
        private Interface north;
        private Interface south;

        private int enabledInterfaceCount;
        private JunctionRouter router;
        private TrafficSignalController signalController;

        public enum JUNCTION {WEST, EAST, NORTH, SOUTH};

        public Junction()
        {
                //AM > A Junction is created with all it's interfaces enabled
                west = new Interface();
                east = new Interface();
                south = new Interface();
                north = new Interface();

                enabledInterfaceCount = 4;

                //AM > Setup traffic signals
                west.configureSignal(north,east,south);
                east.configureSignal(south,west,north);
                south.configureSignal(west,north,east);
                north.configureSignal(east,south,west);
        }

        public void enableInterface(JUNCTION face) throws InterfaceException
        {
                Interface inf = getInterface(face);
                if(!inf.isEnabled())
                {
                        inf.enableInterface();
                        enabledInterfaceCount++;
                }
        }

        public void disableInterface(JUNCTION face) throws InterfaceException, J
unctionException
        {
                Interface inf = getInterface(face);
                if(inf.isEnabled())
                {
                        inf.disableInterface();
                        enabledInterfaceCount--;
                }

                if(enabledInterfaceCount < 2)
                        throw new JunctionException("There needs to be a minimum of two en
abled Inferfaces");
        }

        public int getEnabledInterfaceCount() {
                return enabledInterfaceCount;
        }

        public void setEnabledInterfaceCount(int enabledInterfaceCount) {
                this.enabledInterfaceCount = enabledInterfaceCount;
        }

        public Interface getInterface(JUNCTION face) throws InterfaceException
        {
                if(face == JUNCTION.EAST && east != null)
                {
                        return east;
                }
                else if(face == JUNCTION.NORTH && north != null)
                {
                        return north;
                }
                else if(face == JUNCTION.SOUTH && south != null)
                {
                        return south;
                }
                else if(face == JUNCTION.WEST && west != null)
                {
                        return west;
                }
                else
                {
                        throw new InterfaceException("Invalid Interface selected or interface i
s disabled");
                }
        }

        public JunctionEntry getJunctionEntry(JUNCTION face) throws InterfaceExc
eption
        {
                JunctionEntry entry = getInterface(face).getEntry();
                if(entry.isConnected())
                        throw new InterfaceException("Junction Entry has a Road connected"
);
                else
                        return entry;
        }

        public JunctionExit getJunctionExit(JUNCTION face) throws InterfaceExcep
tion
        {
                JunctionExit exit = getInterface(face).getExit();
                if(exit.isConnected())
                        throw new InterfaceException("Junction Exit has a Road connected")
;
                else
                return exit;
        }

        public JunctionRouter getRoutingTable() {
                return router;
        }

        public void setRoutingTable(JunctionRouter router) {
                this.router = router;
        }

        public Interface getExitInterface(Destination dest) throws InvalidRouteE
xception, JunctionException
        {
                if(router != null)
                        return router.getExitInterface(dest);
                else
                        throw new JunctionException("Routing Table not set");
        }

        //AM > is there a green signal from source to destination
        public boolean isExitGreen(Interface source, Interface dest) throws Inte
rfaceException
```

```
        {
                return source.getSignalState(dest);
        }


        public TrafficSignalController getSignalController() {
                return signalController;
        }

        public void setSignalController() throws InterfaceException {
                this.signalController = new TrafficSignalController(this);
        }
}
```

```java
package core.network.junction;

import java.util.HashMap;

import core.endpoints.Destination;
import core.endpoints.JunctionExit;
import core.network.interfaces.Interface;

public class JunctionRouter {

        private HashMap<Destination,Interface> map;

        public JunctionRouter()
        {
                map = new HashMap<Destination,Interface>();
        }

        public void add(Destination d, Interface face)
        {
                if(d != null && face != null)
                {
                        map.put(d, face);
                }
        }

        public Interface getExitInterface(Destination dest) throws InvalidRouteE
xception {
                Interface inf = map.get(dest);
                if(inf == null)
                {
                        throw new InvalidRouteException("Destination does not exist");
                }
                return inf;
        }
}
```

```java
package core.network.junction;

import core.network.interfaces.InterfaceException;
import core.network.interfaces.TrafficSignal;
import core.network.junction.Junction.JUNCTION;

public class TrafficSignalController {

        private TrafficSignal westSignal;
        private TrafficSignal northSignal;
        private TrafficSignal eastSignal;
        private TrafficSignal southSignal;
        private Junction junction;
        private int cycle;

        public TrafficSignalController(Junction junc) throws InterfaceException
        {
                this.junction = junc;
                westSignal = junc.getInterface(JUNCTION.WEST).getSignals();
                northSignal = junc.getInterface(JUNCTION.NORTH).getSignals();
                southSignal= junc.getInterface(JUNCTION.SOUTH).getSignals();
                eastSignal = junc.getInterface(JUNCTION.EAST).getSignals();
                cycle = 0;
        }

        public void changeSignals() throws InterfaceException
        {
                setWestSignal();
                setNorthSignal();
                setEastSignal();
                setSouthSignal();

                //AM > Change the cycle each time the function is called
                cycle = (cycle + 1) % 4;
        }

        public void setWestSignal() throws InterfaceException
        {
                if(cycle == 0)
                {
                westSignal.setSignal(junction.getInterface(JUNCTION.NORTH), true
);
                westSignal.setSignal(junction.getInterface(JUNCTION.EAST), true)
;
                westSignal.setSignal(junction.getInterface(JUNCTION.SOUTH), fals
e);
                }
                else if(cycle == 1 || cycle == 2)
                {
                        westSignal.setSignal(junction.getInterface(JUNCTION.NORT
H), false);
                        westSignal.setSignal(junction.getInterface(JUNCTION.EAST
), false);
                        westSignal.setSignal(junction.getInterface(JUNCTION.SOUT
H), false);
                }
                else if(cycle == 3)
                {
                        westSignal.setSignal(junction.getInterface(JUNCTION.NORT
H), false);
                        westSignal.setSignal(junction.getInterface(JUNCTION.EAST
), false);
                        westSignal.setSignal(junction.getInterface(JUNCTION.SOUT
H), true);
                }
        }

        public void setNorthSignal() throws InterfaceException
        {
                if(cycle == 0 || cycle == 3)
                {
                northSignal.setSignal(junction.getInterface(JUNCTION.WEST), fals
e);
                northSignal.setSignal(junction.getInterface(JUNCTION.SOUTH), fal
se);
                northSignal.setSignal(junction.getInterface(JUNCTION.EAST), fals
e);
                }
                else if(cycle == 1)
                {
                        northSignal.setSignal(junction.getInterface(JUNCTION.WES
T), true);
                        northSignal.setSignal(junction.getInterface(JUNCTION.SOU
TH), false);
                        northSignal.setSignal(junction.getInterface(JUNCTION.EAS
T), false);
                }
                else if(cycle == 2)
                {
                        northSignal.setSignal(junction.getInterface(JUNCTION.WES
T), false);
                        northSignal.setSignal(junction.getInterface(JUNCTION.SOU
TH), true);
                        northSignal.setSignal(junction.getInterface(JUNCTION.EAS
T), true);
                }
        }

        public void setEastSignal() throws InterfaceException
        {
                if(cycle == 0)
                {
                eastSignal.setSignal(junction.getInterface(JUNCTION.NORTH), fals
e);
                eastSignal.setSignal(junction.getInterface(JUNCTION.WEST), true)
;
                eastSignal.setSignal(junction.getInterface(JUNCTION.SOUTH), true
);
                }
                else if(cycle == 1 || cycle == 2)
                {
                        eastSignal.setSignal(junction.getInterface(JUNCTION.NORT
H), false);
                        eastSignal.setSignal(junction.getInterface(JUNCTION.WEST
), false);
                        eastSignal.setSignal(junction.getInterface(JUNCTION.SOUT
H), false);
                }
                else if(cycle == 3)
                {
                        eastSignal.setSignal(junction.getInterface(JUNCTION.NORT
H), true);
                        eastSignal.setSignal(junction.getInterface(JUNCTION.WEST
), false);
                        eastSignal.setSignal(junction.getInterface(JUNCTION.SOUT
H), false);
                }
        }

        public void setSouthSignal() throws InterfaceException
        {
                if(cycle == 0 || cycle == 3)
                {
                southSignal.setSignal(junction.getInterface(JUNCTION.WEST),false
);
                southSignal.setSignal(junction.getInterface(JUNCTION.NORTH),fals
e);
```

```java
                southSignal.setSignal(junction.getInterface(JUNCTION.EAST),false
);
                }
                else if(cycle == 1)
                {
                        southSignal.setSignal(junction.getInterface(JUNCTION.WES
T),false);
                        southSignal.setSignal(junction.getInterface(JUNCTION.NOR
TH),false);
                        southSignal.setSignal(junction.getInterface(JUNCTION.EAS
T),true);
                }
                else if(cycle == 2)
                {
                        southSignal.setSignal(junction.getInterface(JUNCTION.WES
T),true);
                        southSignal.setSignal(junction.getInterface(JUNCTION.NOR
TH),true);
                        southSignal.setSignal(junction.getInterface(JUNCTION.EAS
T),false);
                }
        }

        public TrafficSignal getWestSignal() {
                return westSignal;
        }

        public TrafficSignal getNorthSignal() {
                return northSignal;
        }

        public TrafficSignal getEastSignal() {
                return eastSignal;
        }

        public TrafficSignal getSouthSignal() {
                return southSignal;
        }

        public int getCycle() {
                return cycle;
        }

        public void setCycle(int cycle) {
                this.cycle = cycle < 0 ? 0 : cycle % 4;
        }
}
```