Yesreshe Bade

```
public class GraphTest {
    private GraphInterface graph;
    private Town[] town;

    public void setUp() throws Exception {
        graph = new Graph();
        town = new Town[12];

        for (int i = 1; i < 12; i++) {
            town[i] = new Town("Town_" + i);
            graph.addVertex(town[i]);
        }

        graph.addEdge(town[1], town[2], 2, "Road_1");
        graph.addEdge(town[1], town[3], 4, "Road_2");
        graph.addEdge(town[1], town[5], 6, "Road_3");
        graph.addEdge(town[3], town[7], 1, "Road_4");
        graph.addEdge(town[2], town[8], 2, "Road_5");
        graph.addEdge(town[4], town[8], 3, "Road_6");
        graph.addEdge(town[6], town[9], 3, "Road_7");
        graph.addEdge(town[9], town[10], 4, "Road_8");
        graph.addEdge(town[8], town[10], 2, "Road_9");
        graph.addEdge(town[5], town[10], 5, "Road_10");
        graph.addEdge(town[10], town[11], 3, "Road_11");
        graph.addEdge(town[2], town[11], 6, "Road_12");
    }

    public void tearDown() throws Exception {
        graph = null;
    }

    public void testGetEdge() {
        assertEquals(new Road(town[2], town[11],6, "Road_12"), graph.getEdge(town[2], town[11]));
        assertEquals(new Road(town[3], town[7],1, "Road_4"), graph.getEdge(town[3], town[7]));
    }

    public void testAddEdge() {
        assertEquals(false, graph.containsEdge(town[3], town[5]));
        graph.addEdge(town[3], town[5], 1, "Road_13");
        assertEquals(true, graph.containsEdge(town[3], town[5]));
    }

    public void testAddVertex() {
        Town newTown = new Town("Town_12");
        assertEquals(false, graph.containsVertex(newTown));
        graph.addVertex(newTown);
        assertEquals(true, graph.containsVertex(newTown));
    }

    public void testContainsEdge() {
        assertEquals(true, graph.containsEdge(town[2], town[11]));
        assertEquals(false, graph.containsEdge(town[3], town[5]));
    }


    public void testContainsVertex() {
        assertEquals(true, graph.containsVertex(new Town("Town_2")));
        assertEquals(false, graph.containsVertex(new Town("Town_12")));
    }


    public void testEdgeSet() {
        Set<Road> roads = graph.edgeSet();
        ArrayList<String> roadArrayList = new ArrayList<String>();
        for(Road road : roads)
            roadArrayList.add(road.getName());
        Collections.sort(roadArrayList);
        assertEquals("Road_1", roadArrayList.get(0));
        assertEquals("Road_10", roadArrayList.get(1));
        assertEquals("Road_11", roadArrayList.get(2));
        assertEquals("Road_12", roadArrayList.get(3));
        assertEquals("Road_2", roadArrayList.get(4));
        assertEquals("Road_8", roadArrayList.get(10));
    }

    public void testEdgesOf() {
        Set<Road> roads = graph.edgesOf(town[1]);
        ArrayList<String> roadArrayList = new ArrayList<String>();
        for(Road road : roads)
            roadArrayList.add(road.getName());
        Collections.sort(roadArrayList);
        assertEquals("Road_1", roadArrayList.get(0));
        assertEquals("Road_2", roadArrayList.get(1));
        assertEquals("Road_3", roadArrayList.get(2));
    }

    public void testRemoveEdge() {
        assertEquals(true, graph.containsEdge(town[2], town[11]));
```

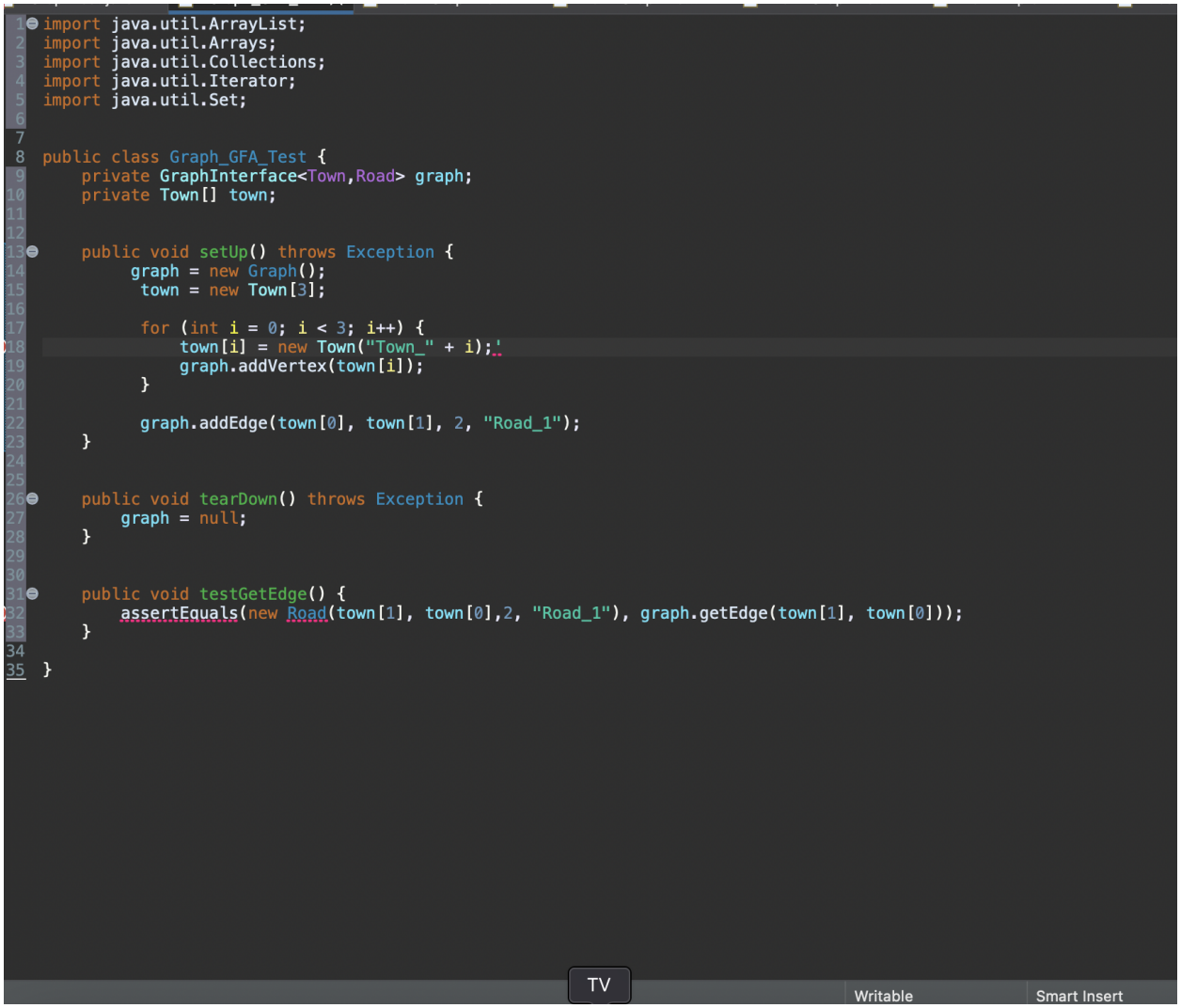
Writable Smart I

Yesreshe Bade

```
        assertEquals(true,roads.contains(town[1]));
        assertEquals(true, roads.contains(town[10]));
        assertEquals(true, roads.contains(town[11]));
        assertEquals(true, roads.contains(town[2]));
        assertEquals(true, roads.contains(town[3]));
    }

    public void testTown_1ToTown_11() {
        String beginTown = "Town_1", endTown = "Town_11";
        Town beginIndex=null, endIndex=null;
        Set<Town> towns = graph.vertexSet();
        Iterator<Town> iterator = towns.iterator();
        while(iterator.hasNext())
        {
            Town town = iterator.next();
            if(town.getName().equals(beginTown))
                beginIndex = town;
            if(town.getName().equals(endTown))
                endIndex = town;
        }
        if(beginIndex != null && endIndex != null)
        {

            ArrayList<String> path = graph.shortestPath(beginIndex,endIndex);
            assertNotNull(path);
            assertTrue(path.size() > 0);
            assertEquals("Town_1 via Road_1 to Town_2 2 mi",path.get(0).trim());
            assertEquals("Town_2 via Road_12 to Town_11 6 mi",path.get(1).trim());
        }
        else
            fail("Town names are not valid");

    }


    public void testTown1ToTown_10() {
        String beginTown = "Town_1", endTown = "Town_10";
        Town beginIndex=null, endIndex=null;
        Set<Town> towns = graph.vertexSet();
        Iterator<Town> iterator = towns.iterator();
        while(iterator.hasNext())
        {
            Town town = iterator.next();
            if(town.getName().equals(beginTown))
                beginIndex = town;
            if(town.getName().equals(endTown))
                endIndex = town;
        }
        if(beginIndex != null && endIndex != null)
        {

            ArrayList<String> path = graph.shortestPath(beginIndex,endIndex);
            assertNotNull(path);
            assertTrue(path.size() > 0);
            assertEquals("Town_1 via Road_2 to Town_3 4 mi",path.get(0).trim());
            assertEquals("Town_3 via Road_5 to Town_8 2 mi",path.get(1).trim());
            assertEquals("Town_8 via Road_9 to Town_10 2 mi",path.get(2).trim());
        }
        else
            fail("Town names are not valid");

    }

    public void testTown_4ToTown_11() {
        String beginTown = "Town_4", endTown = "Town_11";
        Town beginIndex=null, endIndex=null;
        Set<Town> towns = graph.vertexSet();
        Iterator<Town> iterator = towns.iterator();
        while(iterator.hasNext())
        {
            Town town = iterator.next();
            if(town.getName().equals(beginTown))
                beginIndex = town;
            if(town.getName().equals(endTown))
                endIndex = town;
        }
        if(beginIndex != null && endIndex != null)
        {

            ArrayList<String> path = graph.shortestPath(beginIndex,endIndex);
            assertNotNull(path);
            assertTrue(path.size() > 0);
            assertEquals("Town_4 via Road_6 to Town_8 3 mi",path.get(0).trim());
            assertEquals("Town_8 via Road_9 to Town_10 2 mi",path.get(1).trim());
            assertEquals("Town_10 via Road_11 to Town_11 3 mi",path.get(2).trim());
        }
        else
            fail("Town names are not valid");

    }
```

Writable | Smart Insert | 172 : 1 : 5403

Yesreshe Bade



```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.Set;


public class Graph_GFA_Test {
    private GraphInterface<Town,Road> graph;
    private Town[] town;


    public void setUp() throws Exception {
         graph = new Graph();
          town = new Town[3];

          for (int i = 0; i < 3; i++) {
              town[i] = new Town("Town_" + i);
              graph.addVertex(town[i]);
          }

          graph.addEdge(town[0], town[1], 2, "Road_1");
    }


    public void tearDown() throws Exception {
        graph = null;
    }


    public void testGetEdge() {
        assertEquals(new Road(town[1], town[0],2, "Road_1"), graph.getEdge(town[1], town[0]));
    }

}
```

TV Writable Smart Insert

Yesreshe Bade

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

public class FXMainPane extends VBox {
    Label addTownLabel, townNameLabel, addRoadLabel, roadNameLabel, selectTownsForRoadLabel, findConnectionLabel, findConnectionFromLabel, toLabel, distLabel;
    VBox addTownVBox, addRoadVBox, findConnectionVBox, bottomVBox;
    HBox addTownHBox, addRoadNameHBox, addRoadHBox, addRoadTownsHBox, findConnectionHBox, bottomHBox;
    Button addTownButton, addRoadButton, findConnectionButton, readFileButton, exitButton;
    Button displayTownsButton, displayRoadsButton;
    TextField addTownTextField, addRoadTextField, specifyDistanceTextField;
    TextArea findConnectionTextArea, displayTowns, displayRoads;
    ComboBox<String> addSourceTownComboBox, addDestTownComboBox, sourceConnectionComboBox, destConnectionComboBox;
    Insets inset, inset2, inset3;

    TownGraphManager graph;
    private Alert alert = new Alert(AlertType.INFORMATION);


    FXMainPane() {
        //TownGraphManager object
        graph = new TownGraphManager();
        //set up margins
        inset = new Insets(10);


        //add-town components
        addTownLabel = new Label("Add Town");
        addTownLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold");
        townNameLabel = new Label("Town Name: ");

        addTownTextField = new TextField();
        addTownTextField.setPrefColumnCount(10);

        displayTowns = new TextArea();

        addTownButton = new Button("Add Town");
        displayTownsButton = new Button("Display Towns");

        //HBox and VBox for add town area
        addTownHBox = new HBox();
        addTownHBox.getChildren().addAll(townNameLabel, addTownTextField, addTownButton);
        addTownVBox = new VBox();

        VBox.setMargin(addTownLabel, inset);
        HBox.setMargin(townNameLabel, inset);
        VBox.setMargin(addTownHBox, inset);
        HBox.setMargin(addTownLabel, inset);
        HBox.setMargin(addTownButton, inset);
```

```java
        addTownVBox.setStyle("-fx-border-color: gray;");
        addTownVBox.setPrefWidth(400);

        //VBox for the display Towns area
        VBox displayTownVBox = new VBox();
        displayTownVBox.setAlignment(Pos.CENTER);
        displayTownVBox.setStyle("-fx-border-color: gray;");
        displayTownVBox.setPrefWidth(200);
        displayTownVBox.getChildren().addAll(displayTowns, displayTownsButton);
        VBox.setMargin(displayTownsButton, inset);
        VBox.setMargin(displayTowns, inset);

        HBox addTown = new HBox();
        addTown.setAlignment(Pos.CENTER);
        addTown.getChildren().addAll(addTownVBox, displayTownVBox);

        //add-road area components
        addRoadLabel = new Label("Add Road");
        addRoadLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold");
        roadNameLabel = new Label("Road Name: ");
        selectTownsForRoadLabel = new Label("Select Towns the Road Connects");
        distLabel = new Label("Distance");

        displayRoads = new TextArea();

        //ComboBoxes of all towns
        addSourceTownComboBox = new ComboBox<String>();
        addDestTownComboBox = new ComboBox<String>();

        displayRoadsButton = new Button("Display Roads");
        addRoadButton = new Button("Add Road");

        addRoadTextField = new TextField();
        addRoadTextField.setPrefColumnCount(10);
        specifyDistanceTextField = new TextField();
        specifyDistanceTextField.setPrefColumnCount(10);

        //HBoxes and VBoxes to put Add Road area together
        addRoadHBox = new HBox();
        addRoadHBox.getChildren().addAll(roadNameLabel, addRoadTextField);
        addRoadHBox.setAlignment(Pos.CENTER);

        addRoadTownsHBox = new HBox();
        HBox.setMargin(addSourceTownComboBox, inset);
        HBox.setMargin(addDestTownComboBox, inset);
        HBox.setMargin(distLabel, inset);
        HBox.setMargin(specifyDistanceTextField, inset);
        HBox.setMargin(addRoadButton, inset);

        HBox addRoadTownsHBox2 = new HBox();
        addRoadTownsHBox2.getChildren().addAll(distLabel, specifyDistanceTextField);
```

Yesreshe Bade

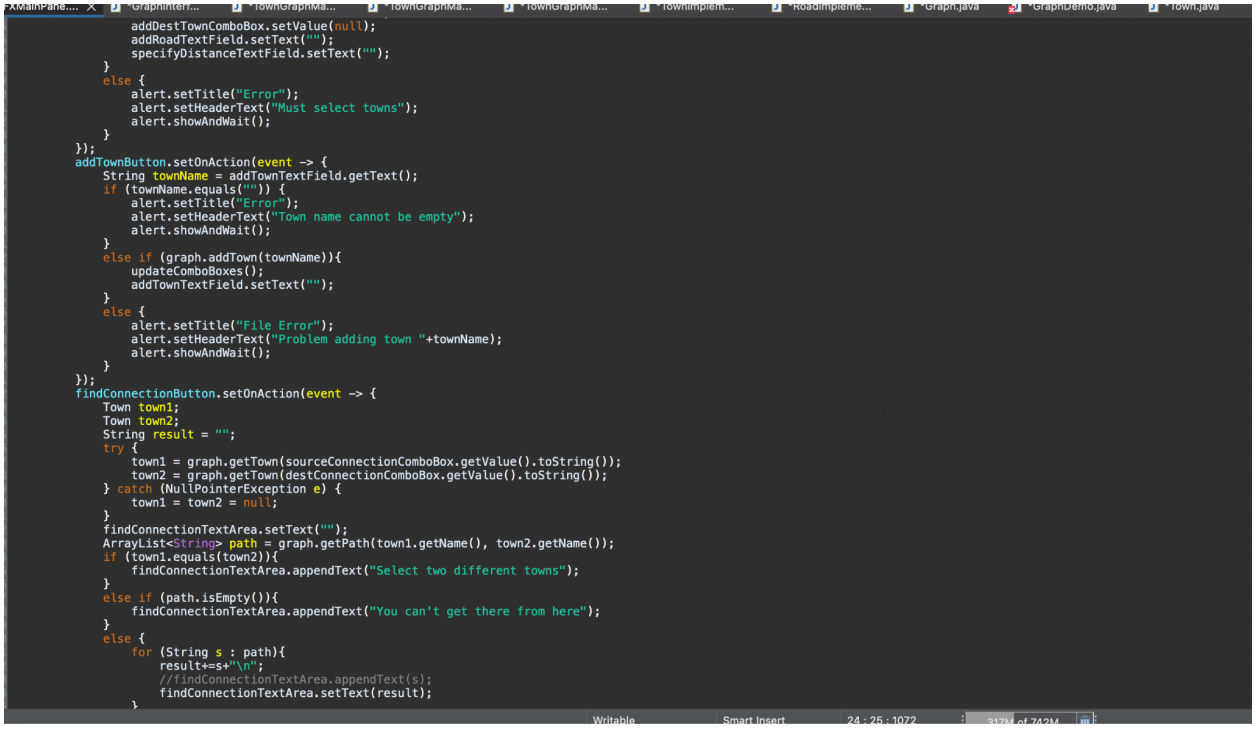
```
        HBox.setMargin(addRoadTextField, inset);
        HBox.setMargin(addRoadTownsHBox, inset);
        HBox.setMargin(addRoadTownsHBox2, inset);


        //find connection area components
        sourceConnectionComboBox = new ComboBox<String>();
        destConnectionComboBox = new ComboBox<String>();

        findConnectionLabel = new Label("Find Connection");
        findConnectionLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold");
        findConnectionFromLabel = new Label("Find connection from ");
        toLabel = new Label("to");

        findConnectionTextArea = new TextArea();

        findConnectionButton = new Button("Find Connection");

        //HBoxes and VBoxes for the Find Connection area
        findConnectionVBox = new VBox();
        findConnectionHBox = new HBox();
        findConnectionHBox.getChildren().addAll(findConnectionFromLabel, sourceConnectionComboBox, toLabel, destConnectionComboBox, findConnectionButton);
        findConnectionVBox.getChildren().addAll(findConnectionLabel, findConnectionHBox, findConnectionTextArea);
        findConnectionVBox.setStyle("-fx-border-color: gray;");
        VBox.setMargin(findConnectionTextArea, inset);

        VBox.setMargin(findConnectionHBox, inset);
        VBox.setMargin(findConnectionLabel, inset);
        HBox.setMargin(findConnectionFromLabel, inset);
        HBox.setMargin(sourceConnectionComboBox, inset);
        HBox.setMargin(toLabel, inset);
        HBox.setMargin(destConnectionComboBox, inset);
        HBox.setMargin(findConnectionButton, inset);

        findConnectionHBox.setAlignment(Pos.CENTER);
        findConnectionVBox.setAlignment(Pos.CENTER);

        //bottom button area components
        bottomHBox = new HBox();
        readFileButton = new Button("Read File");
        exitButton =new Button("Exit");

        bottomVBox = new VBox();
        bottomVBox.getChildren().addAll(bottomHBox);
        bottomVBox.setStyle("-fx-border-color: gray;");

        bottomHBox.getChildren().addAll(readFileButton, exitButton);

        bottomHBox.setAlignment(Pos.CENTER);

        VBox.setMargin(bottomHBox,inset);
```

Writable | Smart Insert | 24 : 25 : 1072 | 296M of 742M

```
        getChildren().addAll(addTown, addRoad, findConnectionVBox, bottomHBox);

        //event handling for buttons
        displayTownsButton.setOnAction(event -> {
            ArrayList<String> towns = graph.allTowns();
            String result = "";
            for(String element : towns)
            {
                result += element+"\n";
            }
            displayTowns.setText(result);
        });
        displayRoadsButton.setOnAction(event -> {
            ArrayList<String> roads = graph.allRoads();
            String result = "";
            for(String element : roads)
            {
                result += element+"\n";
            }
            displayRoads.setText(result);
        });
        addRoadButton.setOnAction(event -> {
            Town town1;
            Town town2;
            try {
                town1 = graph.getTown(addSourceTownComboBox.getValue().toString());
                town2 = graph.getTown(addDestTownComboBox.getValue().toString());
            } catch (NullPointerException e) {
                town1 = town2 = null;
            }
            String name = addRoadTextField.getText();
            String strWeight = specifyDistanceTextField.getText();
            int weight = 0;
            try {
                if (!strWeight.equals("")) weight = Integer.parseInt(strWeight);
            }
            catch (NumberFormatException e) {
                weight = -1;
            }
            if (weight < 0) {
                alert.setTitle("Error");
                alert.setHeaderText("Distance must be an integer");
                alert.showAndWait();
            }
            else if (name.equals("")) {
                alert.setTitle("Error");
                alert.setHeaderText("Road name cannot be blank");
                alert.showAndWait();
            }
            else if (town1 !=null && town2!=null) {
                graph.addRoad(town1.getName(), town2.getName(), weight, name);
```

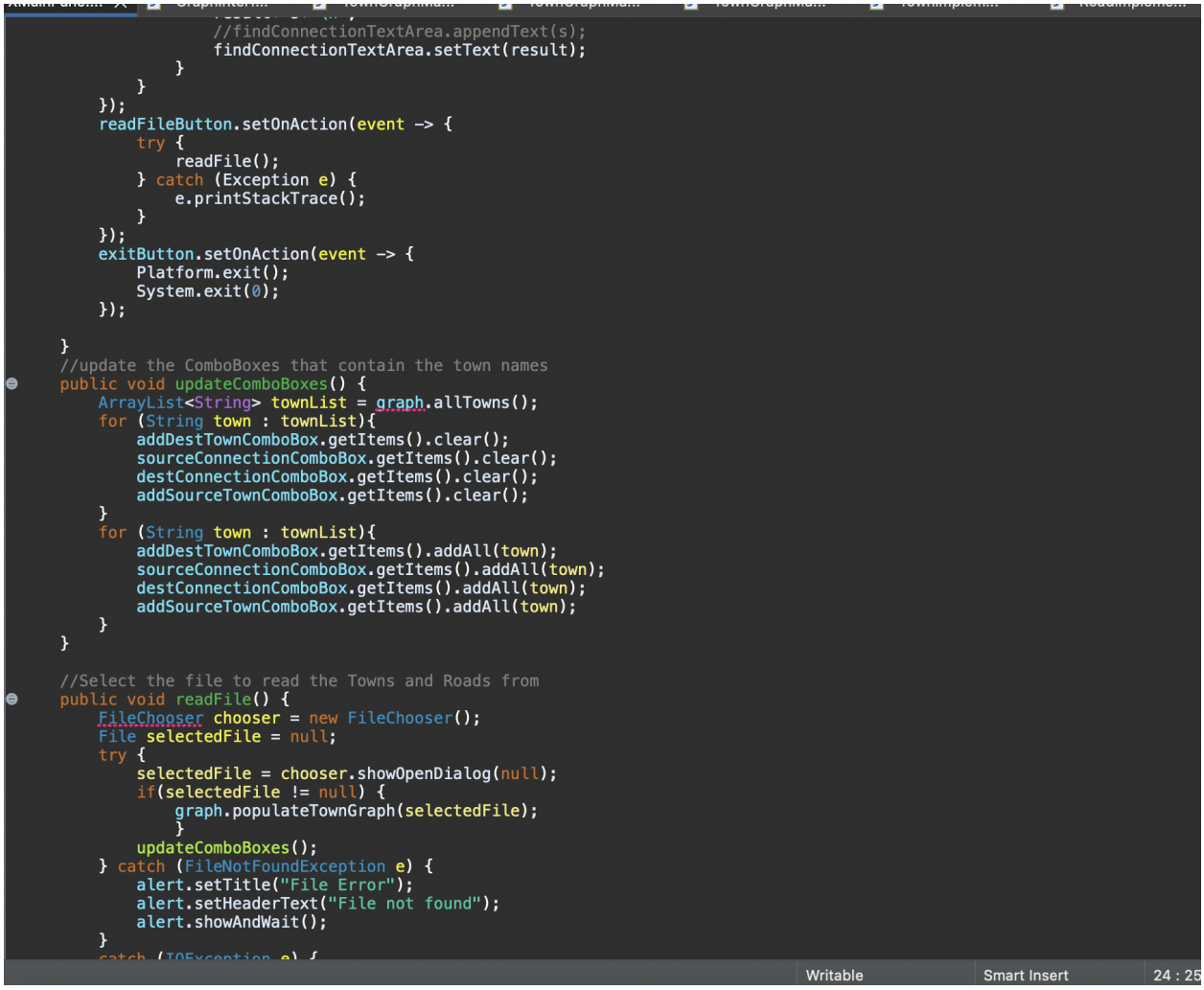
Writable | Smart Insert | 24 : 25 : 1072 | 292M of 742M

Yesreshe Bade



```
            addDestTownComboBox.setValue(null);
            addRoadTextField.setText("");
            specifyDistanceTextField.setText("");
        }
        else {
            alert.setTitle("Error");
            alert.setHeaderText("Must select towns");
            alert.showAndWait();
        }
    });
    addTownButton.setOnAction(event -> {
        String townName = addTownTextField.getText();
        if (townName.equals("")) {
            alert.setTitle("Error");
            alert.setHeaderText("Town name cannot be empty");
            alert.showAndWait();
        }
        else if (graph.addTown(townName)){
            updateComboBoxes();
            addTownTextField.setText("");
        }
        else {
            alert.setTitle("File Error");
            alert.setHeaderText("Problem adding town "+townName);
            alert.showAndWait();
        }
    });
    findConnectionButton.setOnAction(event -> {
        Town town1;
        Town town2;
        String result = "";
        try {
            town1 = graph.getTown(sourceConnectionComboBox.getValue().toString());
            town2 = graph.getTown(destConnectionComboBox.getValue().toString());
        } catch (NullPointerException e) {
            town1 = town2 = null;
        }
        findConnectionTextArea.setText("");
        ArrayList<String> path = graph.getPath(town1.getName(), town2.getName());
        if (town1.equals(town2)){
            findConnectionTextArea.appendText("Select two different towns");
        }
        else if (path.isEmpty()){
            findConnectionTextArea.appendText("You can't get there from here");
        }
        else {
            for (String s : path){
                result+=s+"\n";
                //findConnectionTextArea.appendText(s);
                findConnectionTextArea.setText(result);
            }
```

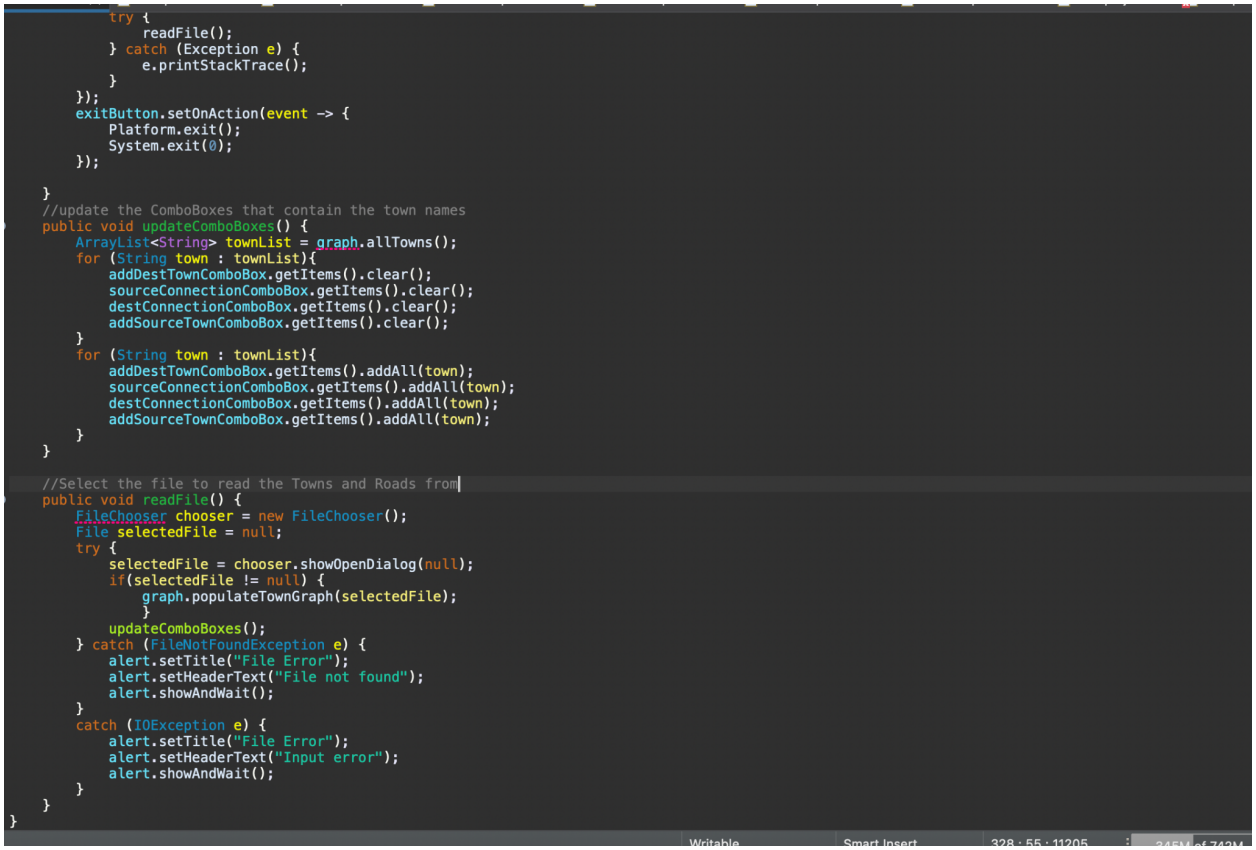
Yesreshe Bade



```
                //findConnectionTextArea.appendText(s);
                findConnectionTextArea.setText(result);
            }
        }
    });
    readFileButton.setOnAction(event -> {
        try {
            readFile();
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
    exitButton.setOnAction(event -> {
        Platform.exit();
        System.exit(0);
    });

}
//update the ComboBoxes that contain the town names
public void updateComboBoxes() {
    ArrayList<String> townList = graph.allTowns();
    for (String town : townList){
        addDestTownComboBox.getItems().clear();
        sourceConnectionComboBox.getItems().clear();
        destConnectionComboBox.getItems().clear();
        addSourceTownComboBox.getItems().clear();
    }
    for (String town : townList){
        addDestTownComboBox.getItems().addAll(town);
        sourceConnectionComboBox.getItems().addAll(town);
        destConnectionComboBox.getItems().addAll(town);
        addSourceTownComboBox.getItems().addAll(town);
    }
}

//Select the file to read the Towns and Roads from
public void readFile() {
    FileChooser chooser = new FileChooser();
    File selectedFile = null;
    try {
        selectedFile = chooser.showOpenDialog(null);
        if(selectedFile != null) {
            graph.populateTownGraph(selectedFile);
            }
        updateComboBoxes();
    } catch (FileNotFoundException e) {
        alert.setTitle("File Error");
        alert.setHeaderText("File not found");
        alert.showAndWait();
    }
    catch (IOException e) {
```

Writable | Smart Insert | 24 : 25

Yesreshe Bade



```
            try {
                readFile();
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
        exitButton.setOnAction(event -> {
            Platform.exit();
            System.exit(0);
        });

    }
    //update the ComboBoxes that contain the town names
    public void updateComboBoxes() {
        ArrayList<String> townList = graph.allTowns();
        for (String town : townList){
            addDestTownComboBox.getItems().clear();
            sourceConnectionComboBox.getItems().clear();
            destConnectionComboBox.getItems().clear();
            addSourceTownComboBox.getItems().clear();
        }
        for (String town : townList){
            addDestTownComboBox.getItems().addAll(town);
            sourceConnectionComboBox.getItems().addAll(town);
            destConnectionComboBox.getItems().addAll(town);
            addSourceTownComboBox.getItems().addAll(town);
        }
    }

    //Select the file to read the Towns and Roads from
    public void readFile() {
        FileChooser chooser = new FileChooser();
        File selectedFile = null;
        try {
            selectedFile = chooser.showOpenDialog(null);
            if(selectedFile != null) {
                graph.populateTownGraph(selectedFile);
                }
            updateComboBoxes();
        } catch (FileNotFoundException e) {
            alert.setTitle("File Error");
            alert.setHeaderText("File not found");
            alert.showAndWait();
        }
        catch (IOException e) {
            alert.setTitle("File Error");
            alert.setHeaderText("Input error");
            alert.showAndWait();
        }
    }
}
```

Yesreshe Bade

```java
import java.io.File;
import java.util.*;


public interface GraphInterface<V, E>
{

    public E getEdge(V sourceVertex, V destinationVertex);


     */
    public E addEdge(V sourceVertex, V destinationVertex, int weight, String description);


    public boolean addVertex(V v);


    public boolean containsEdge(V sourceVertex, V destinationVertex);


    public boolean containsVertex(V v);


    public Set<E> edgeSet();


    public Set<E> edgesOf(V vertex);


    public E removeEdge(V sourceVertex, V destinationVertex, int weight, String description);


    public boolean removeVertex(V v);


    public Set<V> vertexSet();


    public ArrayList<String> shortestPath(V sourceVertex, V destinationVertex);


    public void dijkstraShortestPath(V sourceVertex);
}
```

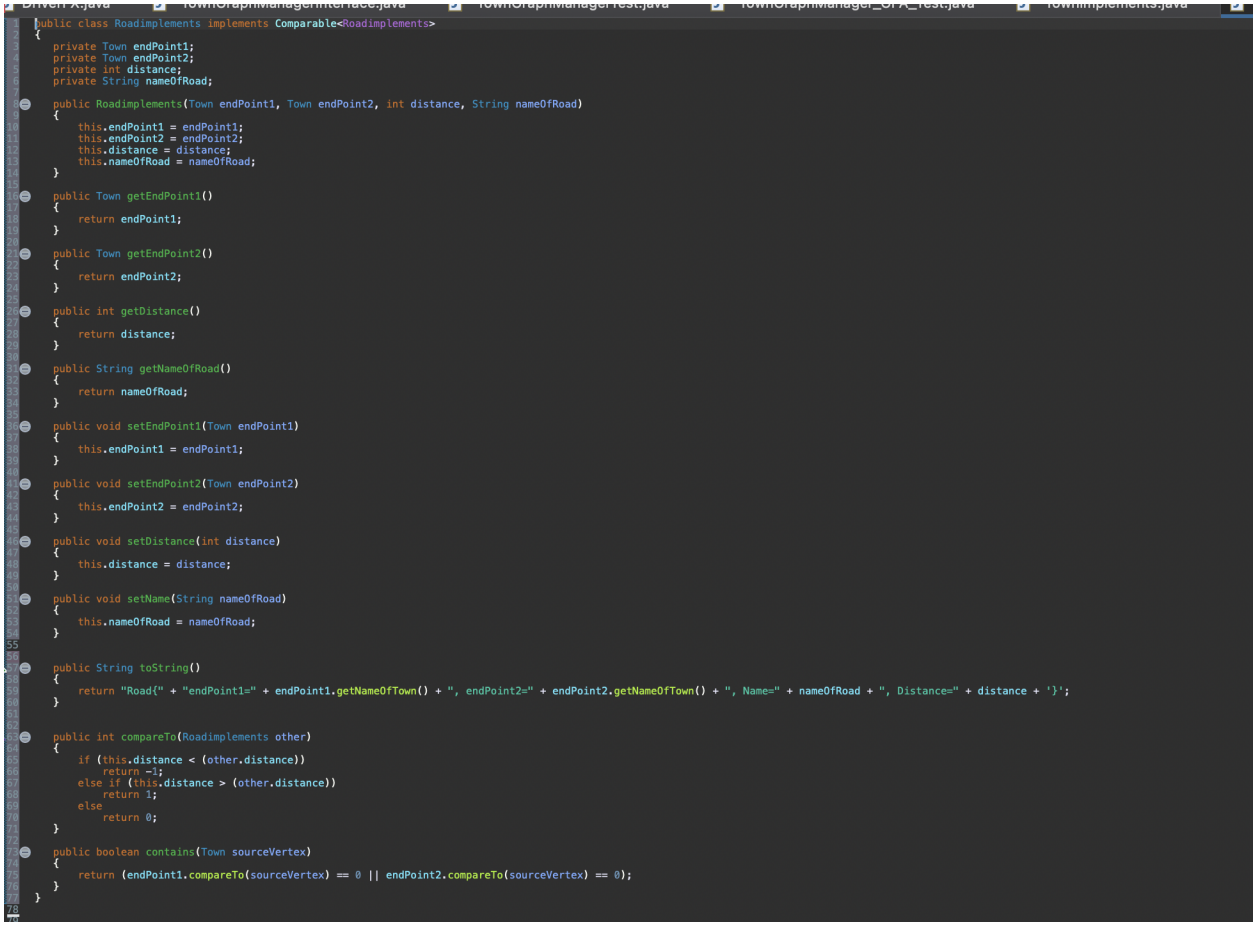
```java
public class GraphDemo
{
   public static void main(String[] args)
   {
       Graph graph = new Graph();
       Town[] towns = { new Town("A"), new Town("B"), new Town("C"), new Town("D"), new Town("E"), new Town("F") };

       graph.addEdge(towns[0], towns[1], 10, "AB");
       graph.addEdge(towns[0], towns[2], 15, "AC");
       graph.addEdge(towns[1], towns[3], 12, "BD");
       graph.addEdge(towns[1], towns[5], 15, "BF");
       graph.addEdge(towns[2], towns[4], 10, "CE");
       graph.addEdge(towns[3], towns[4], 2, "DE");
       graph.addEdge(towns[3], towns[5], 1, "DF");
       graph.addEdge(towns[5], towns[4], 5, "FE");

       for (int i = 0; i < towns.length; i++)
       {
           for (int j = i+1; j < towns.length; j++)
           {
               System.out.println(towns[i].getNameOfTown() + " to " + towns[j].getNameOfTown() + ": ");
               System.out.println(graph.shortestPath(towns[i], towns[j]) + "\n");
           }
       }
   }
}
```

Yesreshe Bade

```java
import java.util.*;
import java.util.Map.Entry;
public class Graph implements GraphInterface<Town, Road>
{
    private Set<Town> towns;
    private Set<Road> roads;

    public Graph()
    {
        towns = new HashSet<Town>();
        roads = new HashSet<Road>();
    }


    public Road getEdge(Town sv, Town dv)
    {
        for (Road road : roads)
        {
            if (((road.getEndPoint1().compareTo(sv) == 0) && (road.getEndPoint2().compareTo(dv) == 0))
                    || ((road.getEndPoint2().compareTo(sv) == 0)
                            && (road.getEndPoint1().compareTo(dv) == 0)))
            {
                return road;
            }
        }

        return null;
    }


    public Road addEdge(Town sv, Town dv, int dist, String name)
    {
        Road road = new Road(sv, dv, dist, name);
        roads.add(road);
        sv.addAdjacentTown(dv, dist);
        dv.addAdjacentTown(sv, dist);

        return road;
    }


    public boolean addVertex(Town v)
    {
        return towns.add(v);
    }


    public boolean containsEdge(Town sv, Town dv)
    {
        for (Road road : roads)
        {
            if (road.contains(sv) && road.contains(dv))
                return true;

        }
        return false;
    }


    public boolean containsVertex(Town v)
    {
        for (Town town : towns)
        {
            if (town.getNameOfTown().compareToIgnoreCase(v.getNameOfTown()) == 0)
                return true;
        }

        return false;
    }
```

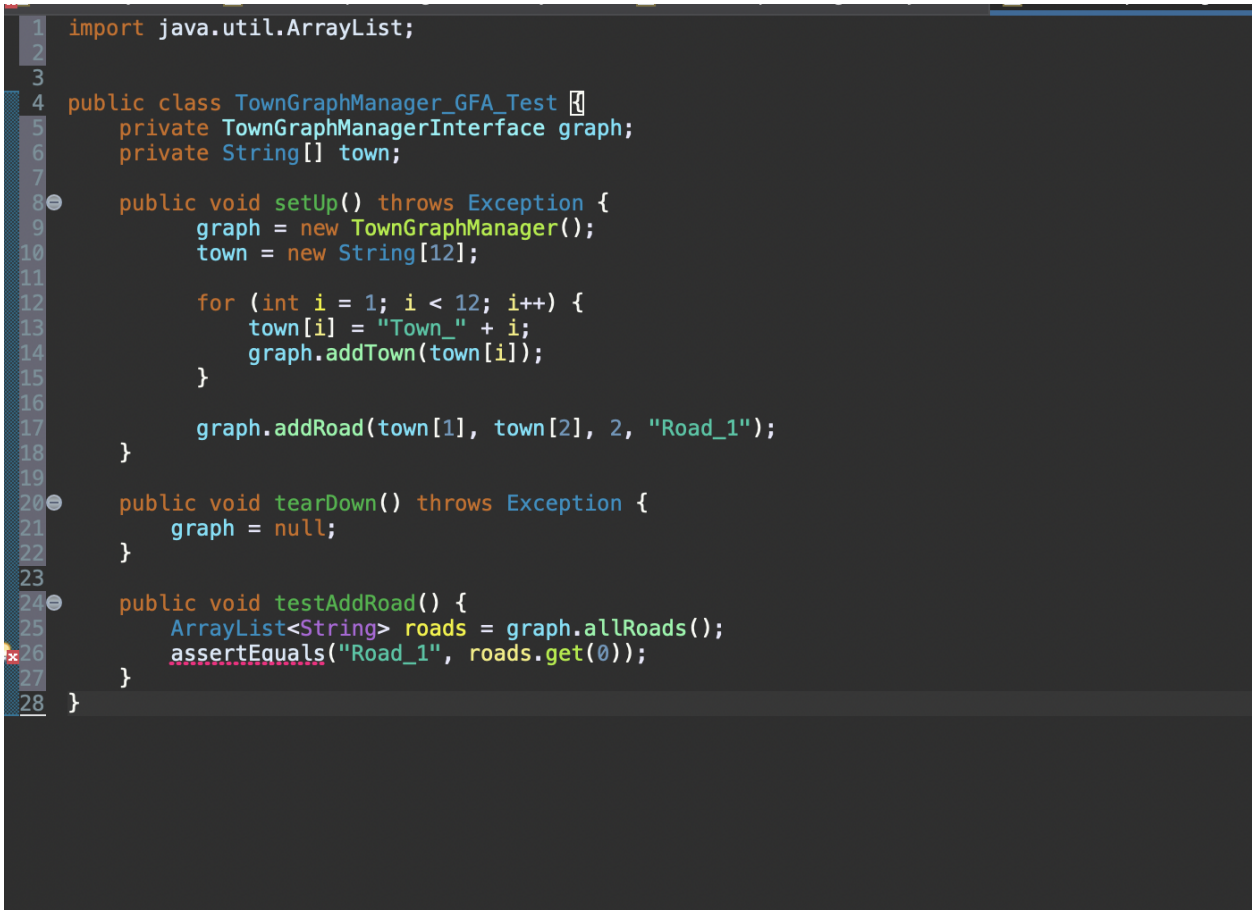
```java
        for (int i = 0; i < listOfTowns.size(); i++)
        {
            sp.add(listOfTowns.get(i).toString());
        }
        sp.add(dv.toString());

        return sp;
    }

    // dijkstraShortestPath method implementation
    @Override
    public void dijkstraShortestPath(Town sv)
    {
        sv.setDistance(0);
        Set<Town> setOfTowns1 = new HashSet<>();
        Set<Town> setOfTowns2 = new HashSet<>();
        setOfTowns2.add(sv);

        while (setOfTowns2.size() != 0)
        {
            Town currentTown = getLowestDistanceTown(setOfTowns2);
            setOfTowns2.remove(currentTown);

            for (Entry<Town, Integer> entry : currentTown.getAdjacentTowns().entrySet())
            {
                Town adjTown = entry.getKey();
                Integer edgedistance = entry.getValue();

                if (!setOfTowns1.contains(adjTown))
                {
                    findMinDistance(adjTown, edgedistance, currentTown);
                    setOfTowns2.add(adjTown);
                }
            }
            setOfTowns1.add(currentTown);
        }
    }

    // getLowestDistanceTown method implementation
    private static Town getLowestDistanceTown(Set<Town> setOfTowns)
    {
        Town lowDistanceTown = null;
        int lowDistance = Integer.MAX_VALUE;
        for (Town town : setOfTowns)
        {
            int townDistance = town.getDistance();
            if (townDistance < lowDistance)
            {
                lowDistance = townDistance;
                lowDistanceTown = town;
            }
        }
        return lowDistanceTown;
    }

    // findMinDistance method implementation
    private static void findMinDistance(Town evaluationTown, Integer edgeWeigh, Town sourceTown)
    {
        Integer sourceTownDistance = sourceTown.getDistance();
        if (sourceTownDistance + edgeWeigh < evaluationTown.getDistance())
        {
            evaluationTown.setDistance(sourceTownDistance + edgeWeigh);
            LinkedList<Town> sp = new LinkedList<>(sourceTown.getShortestPath());
            sp.add(sourceTown);
            evaluationTown.setShortestPath(sp);
        }
    }
```

Yesreshe Bade



```java
public class Roadimplements implements Comparable<Roadimplements>
{
    private Town endPoint1;
    private Town endPoint2;
    private int distance;
    private String nameOfRoad;

    public Roadimplements(Town endPoint1, Town endPoint2, int distance, String nameOfRoad)
    {
        this.endPoint1 = endPoint1;
        this.endPoint2 = endPoint2;
        this.distance = distance;
        this.nameOfRoad = nameOfRoad;
    }

    public Town getEndPoint1()
    {
        return endPoint1;
    }

    public Town getEndPoint2()
    {
        return endPoint2;
    }

    public int getDistance()
    {
        return distance;
    }

    public String getNameOfRoad()
    {
        return nameOfRoad;
    }

    public void setEndPoint1(Town endPoint1)
    {
        this.endPoint1 = endPoint1;
    }

    public void setEndPoint2(Town endPoint2)
    {
        this.endPoint2 = endPoint2;
    }

    public void setDistance(int distance)
    {
        this.distance = distance;
    }

    public void setName(String nameOfRoad)
    {
        this.nameOfRoad = nameOfRoad;
    }


    public String toString()
    {
        return "Road{" + "endPoint1=" + endPoint1.getNameOfTown() + ", endPoint2=" + endPoint2.getNameOfTown() + ", Name=" + nameOfRoad + ", Distance=" + distance + '}';
    }


    public int compareTo(Roadimplements other)
    {
        if (this.distance < (other.distance))
            return -1;
        else if (this.distance > (other.distance))
            return 1;
        else
            return 0;
    }

    public boolean contains(Town sourceVertex)
    {
        return (endPoint1.compareTo(sourceVertex) == 0 || endPoint2.compareTo(sourceVertex) == 0);
    }
}
```

Yesreshe Bade

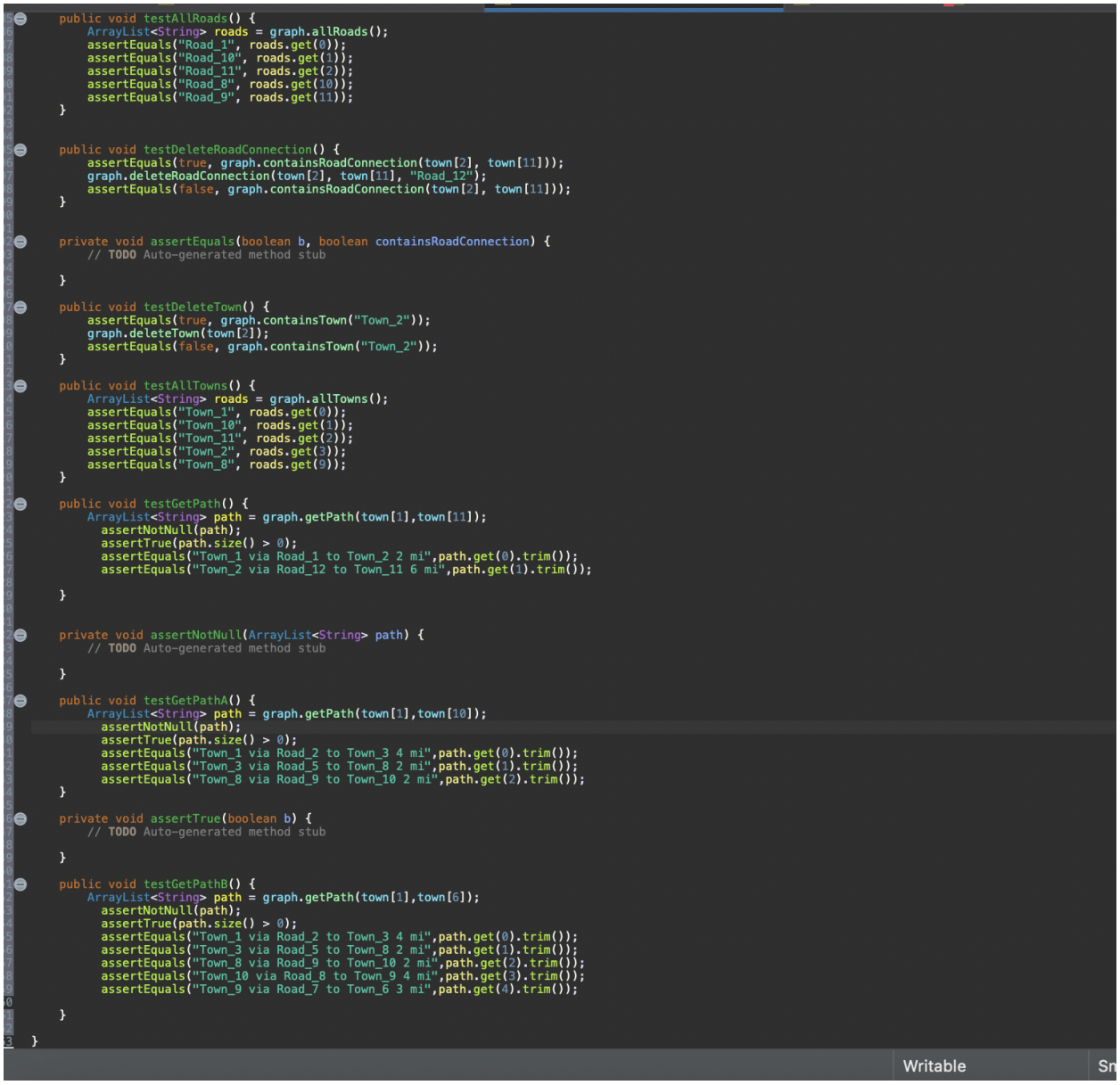


```java
import java.util.ArrayList;


public class TownGraphManager_GFA_Test {
    private TownGraphManagerInterface graph;
    private String[] town;

    public void setUp() throws Exception {
          graph = new TownGraphManager();
          town = new String[12];

          for (int i = 1; i < 12; i++) {
              town[i] = "Town_" + i;
              graph.addTown(town[i]);
          }

          graph.addRoad(town[1], town[2], 2, "Road_1");
    }

    public void tearDown() throws Exception {
        graph = null;
    }

    public void testAddRoad() {
        ArrayList<String> roads = graph.allRoads();
        assertEquals("Road_1", roads.get(0));
    }
}
```

Yesreshe Bade

```java
import java.util.ArrayList;


public class TownGraphManagerTest {
    private TownGraphManagerInterface graph;
    private String[] town;

    public void setUp() throws Exception {
          graph = new TownGraphManager();
          town = new String[12];

          for (int i = 1; i < 12; i++) {
              town[i] = "Town_" + i;
              graph.addTown(town[i]);
          }

          graph.addRoad(town[1], town[2], 2, "Road_1");
          graph.addRoad(town[1], town[3], 4, "Road_2");
          graph.addRoad(town[1], town[5], 6, "Road_3");
          graph.addRoad(town[3], town[7], 1, "Road_4");
          graph.addRoad(town[3], town[8], 2, "Road_5");
          graph.addRoad(town[4], town[8], 3, "Road_6");
          graph.addRoad(town[6], town[9], 3, "Road_7");
          graph.addRoad(town[9], town[10], 4, "Road_8");
          graph.addRoad(town[8], town[10], 2, "Road_9");
          graph.addRoad(town[5], town[10], 5, "Road_10");
          graph.addRoad(town[10], town[11], 3, "Road_11");
          graph.addRoad(town[2], town[11], 6, "Road_12");

    }

    public void tearDown() throws Exception {
        graph = null;
    }

    public void testAddRoad() {
        ArrayList<String> roads = graph.allRoads();
        assertEquals("Road_1", roads.get(0));
        assertEquals("Road_10", roads.get(1));
        assertEquals("Road_11", roads.get(2));
        assertEquals("Road_12", roads.get(3));
        graph.addRoad(town[4], town[11], 1,"Road_13");
        roads = graph.allRoads();
        assertEquals("Road_1", roads.get(0));
        assertEquals("Road_10", roads.get(1));
        assertEquals("Road_11", roads.get(2));
        assertEquals("Road_12", roads.get(3));
        assertEquals("Road_13", roads.get(4));

    }

    private void assertEquals(String string, String string2) {
        // TODO Auto-generated method stub

    }

    public void testGetRoad() {
        assertEquals("Road_12", graph.getRoad(town[2], town[11]));
        assertEquals("Road_4", graph.getRoad(town[3], town[7]));
    }

    public void testAddTown() {
        assertEquals(false, graph.containsTown("Town_12"));
        graph.addTown("Town_12");
        assertEquals(true, graph.containsTown("Town_12"));
    }

    public void testDisjointGraph() {
        assertEquals(false, graph.containsTown("Town_12"));
        graph.addTown("Town_12");
        ArrayList<String> path = graph.getPath(town[1],"Town_12");
        assertFalse(path.size() > 0);
    }

    public void testContainsTown() {
        assertEquals(true, graph.containsTown("Town_2"));
        assertEquals(false, graph.containsTown("Town_12"));
    }
```

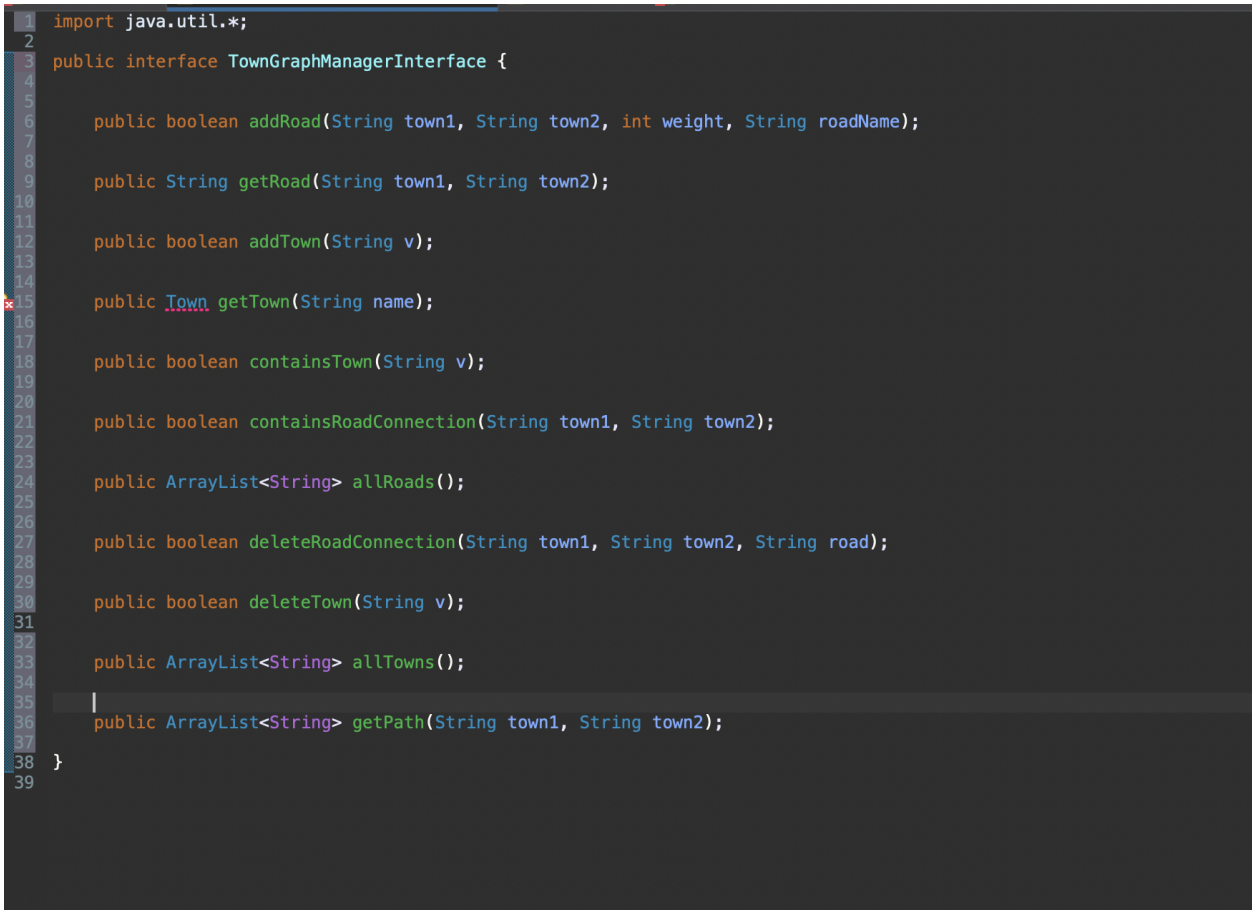
Yesreshe Bade



```
    public void testAllRoads() {
        ArrayList<String> roads = graph.allRoads();
        assertEquals("Road_1", roads.get(0));
        assertEquals("Road_10", roads.get(1));
        assertEquals("Road_11", roads.get(2));
        assertEquals("Road_8", roads.get(10));
        assertEquals("Road_9", roads.get(11));
    }


    public void testDeleteRoadConnection() {
        assertEquals(true, graph.containsRoadConnection(town[2], town[11]));
        graph.deleteRoadConnection(town[2], town[11], "Road_12");
        assertEquals(false, graph.containsRoadConnection(town[2], town[11]));
    }


    private void assertEquals(boolean b, boolean containsRoadConnection) {
        // TODO Auto-generated method stub

    }

    public void testDeleteTown() {
        assertEquals(true, graph.containsTown("Town_2"));
        graph.deleteTown(town[2]);
        assertEquals(false, graph.containsTown("Town_2"));
    }

    public void testAllTowns() {
        ArrayList<String> roads = graph.allTowns();
        assertEquals("Town_1", roads.get(0));
        assertEquals("Town_10", roads.get(1));
        assertEquals("Town_11", roads.get(2));
        assertEquals("Town_2", roads.get(3));
        assertEquals("Town_8", roads.get(9));
    }

    public void testGetPath() {
        ArrayList<String> path = graph.getPath(town[1],town[11]);
          assertNotNull(path);
          assertTrue(path.size() > 0);
          assertEquals("Town_1 via Road_1 to Town_2 2 mi",path.get(0).trim());
          assertEquals("Town_2 via Road_12 to Town_11 6 mi",path.get(1).trim());

    }


    private void assertNotNull(ArrayList<String> path) {
        // TODO Auto-generated method stub

    }

    public void testGetPathA() {
        ArrayList<String> path = graph.getPath(town[1],town[10]);
          assertNotNull(path);
          assertTrue(path.size() > 0);
          assertEquals("Town_1 via Road_2 to Town_3 4 mi",path.get(0).trim());
          assertEquals("Town_3 via Road_5 to Town_8 2 mi",path.get(1).trim());
          assertEquals("Town_8 via Road_9 to Town_10 2 mi",path.get(2).trim());
    }

    private void assertTrue(boolean b) {
        // TODO Auto-generated method stub

    }

    public void testGetPathB() {
        ArrayList<String> path = graph.getPath(town[1],town[6]);
          assertNotNull(path);
          assertTrue(path.size() > 0);
          assertEquals("Town_1 via Road_2 to Town_3 4 mi",path.get(0).trim());
          assertEquals("Town_3 via Road_5 to Town_8 2 mi",path.get(1).trim());
          assertEquals("Town_8 via Road_9 to Town_10 2 mi",path.get(2).trim());
          assertEquals("Town_10 via Road_8 to Town_9 4 mi",path.get(3).trim());
          assertEquals("Town_9 via Road_7 to Town_6 3 mi",path.get(4).trim());

    }

}
```

Writable

Yesreshe Bade



```java
import java.util.*;

public interface TownGraphManagerInterface {

    public boolean addRoad(String town1, String town2, int weight, String roadName);

    public String getRoad(String town1, String town2);

    public boolean addTown(String v);

    public Town getTown(String name);

    public boolean containsTown(String v);

    public boolean containsRoadConnection(String town1, String town2);

    public ArrayList<String> allRoads();

    public boolean deleteRoadConnection(String town1, String town2, String road);

    public boolean deleteTown(String v);

    public ArrayList<String> allTowns();

    public ArrayList<String> getPath(String town1, String town2);

}
```

Yesreshe Bade

```java
import java.io.IOException;

public class DriverFX extends Application {
    /**
     * The main method for the GUI JavaFX version
     * @param args not used
     * @throws IOException
     */
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {
        //  instantiate the FXMainPane, name it root
        FXMainPane root = new FXMainPane();
        //  set the scene to hold root
        stage.setScene(new Scene(root, 600,700));
        //set stage title
        stage.setTitle("Travelling Student");
        //display the stage
        stage.show();

    }

}
```

Lesson learned: