DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

CS-112 OBJECT ORIENTED PROGRAMMING

PROJECT REPORT
Object Oriented Modeling and Simulation of Airport Surface Traffic Control (ASTC) System

SUBMITTED TO:
Lec Anum Abdul Salam

SUBMITTED BY:
Shaheer Muhktiar
Shafla Rehman
Areesha Mumtaz (409723)
Dep C&SE
DE- 44 Syn B

Submission date: 1/17/24

INTRODUCTION:

Since, in popular airport and airspace simulation models, procedural programming paradigms are used, adding to the difficulty of model debugging and maintenance, hence reducing the efficiency and increasing the cost.

Therefore, the project's goal was to model and simulate an Airport Surface Traffic Control (ASTC) system using object-oriented principles, helping in analyzing and managing the system efficiently. Also optimizing development time, cutting down on coding work, and—above all—lowering maintenance expenses are the main reasons for implementing OOP, unlike using standard procedural programming in which 45 to 65% of the cost is needed to maintain a software model.

DESCRIPTION:

ASTC can be viewed as a task driven system, where using message exchange multiple tasks are performed by the interaction of entities involved, where the creation and execution of tasks is linked with a timing mechanism. Also, each entity involved implements its own local procedures for completing its share of the task.

Generally the mechanism can be explained as, firstly a request would be sent for performing the task, according to its priority it would be assigned a certain time when it is supposed to be performed, also the availabilities of other entities would be checked, meaning if the task can be performed or not, once the time comes and task can be performed, it starts executing.

DESCRIPTION AND MODELING:

Task:

Three components will be used to define any task: a secondary label, a primary label, and its priority data set. The primary and secondary labels will help identify the task's main label as well as the subcategories, depending on which task would be sent to respective entities while the priority data set will be used to determine the task's priority.

Task queue:

One of the fundamental requirements of our project is a task queue, which will be a list of tasks arranged in order of priority. This means that some tasks are obviously more important than others, and that while it is possible that we will receive a request to perform a task later, it may also be necessary to perform it at the earliest. Consequently, upon receiving a request to perform a task, it is compared to all the other tasks in the queue and positioned there based on its priority.

GLOBAL CLOCK:

As previously indicated, the tasks in ASTC would be connected to the Global Clock, an independent timekeeping entity that can also be used to connect other entities. Upon receiving a request to perform a task, the priority of the task will determine the time it is assigned, either at the next available spot or by adjusting the queue to perform the task earlier, current time.

TASK DRIVEN SIMULATION ENGINE:

It is necessary for all simulated simulations because this entity is the one that will receive requests to complete tasks, which means it will add tasks to the task queue, depending upon priority and availability, send tasks for execution based on the specifications of the task, and once it has been completed removing tasks from the priority queue, are all done by the simulated task engine.

AIRPLANE:

An airplane will have all the information needed to identify it, including its ID, orientation (i.e., whether it is needed to depart or arrive), and all the data needed to carry out its tasks, such as its current and next positions, the total cost of its journey, and the optimal route needed to get there.

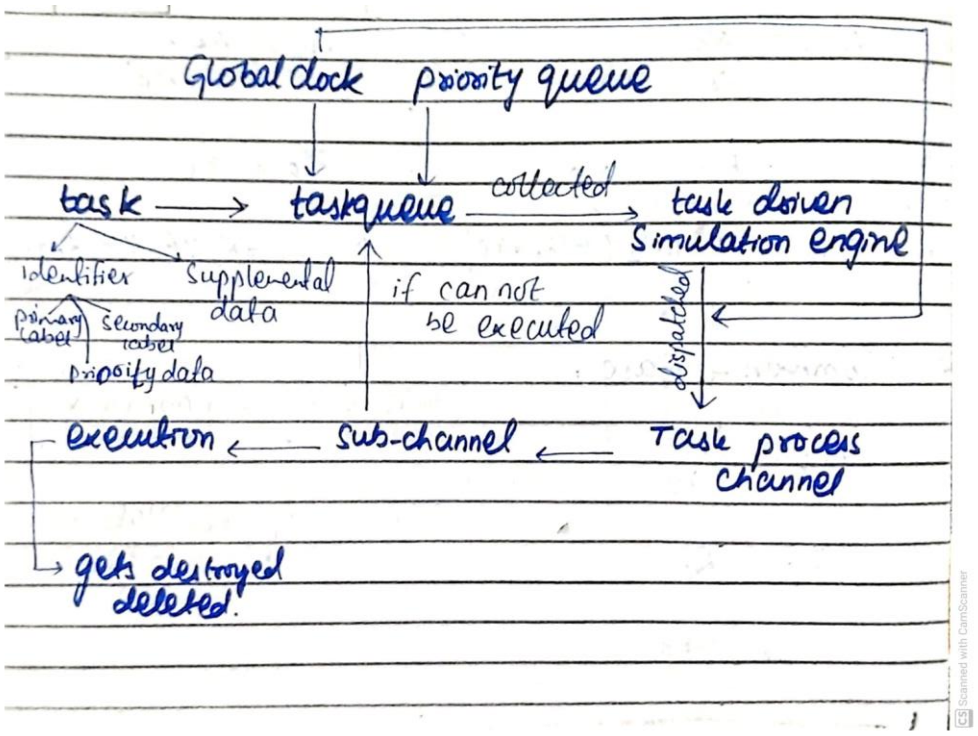
AIRPLANE LIST:

It would consist of a list of airplane objects that can be accessed using pointers and it would consist of functions like creating an airplane while assigning it an id at the time of creation, also adding airplane object, and depending upon the task requirement sending it to the required entity, and finally removing the airplane objects from the list once the task has been executed.

AIRPLANE GROUND NETWORK:

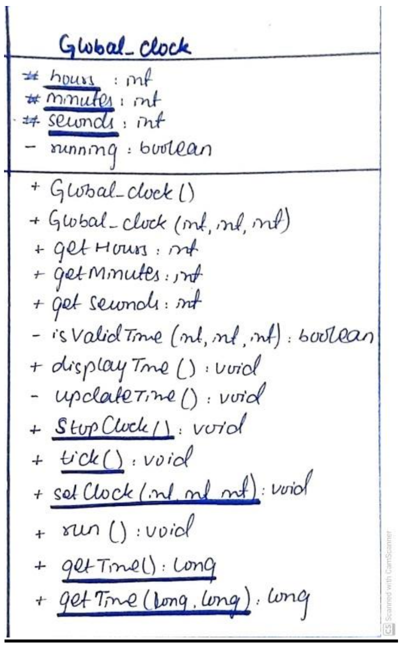
Air traffic control will be handled by the airplane ground network, which will include data about runways, taxiways, and gates, such as which links are available, and which are occupied. It will also regulate the state of all three, opening and closing them and changing their status from occupied to available based on the positioning and orientation (giving information if it's departure or arrival).

FLOW CHART:



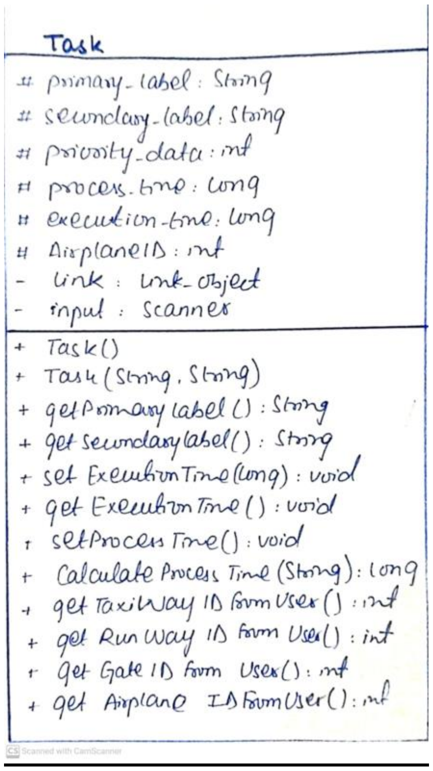
UML:

Global clock:



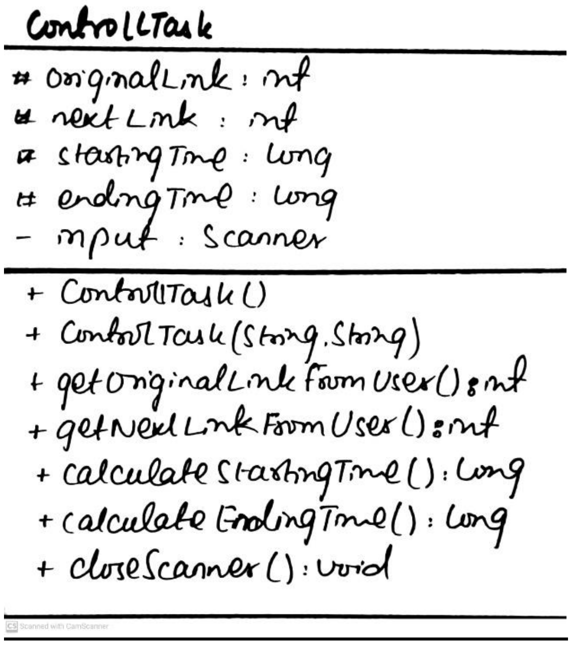
```
Global_clock
---------------------------------
# hours : int
# minutes : int
# seconds : int
- running : boolean
---------------------------------
+ Global_clock()
+ Global_clock(int, int, int)
+ getHours : int
+ getMinutes : int
+ get seconds : int
- isValidTime(int, int, int) : boolean
+ displayTime() : void
- updateTime() : void
+ StopClock() : void
+ tick() : void
+ setClock(int int int) : void
+ run() : void
+ getTime() : Long
+ getTime(long, long) : long
```

Task:

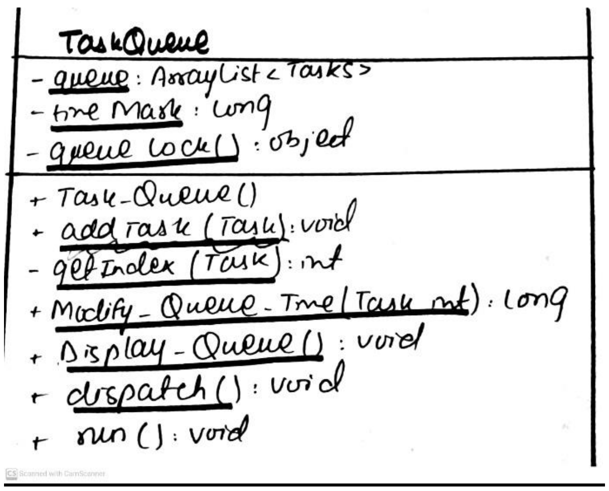


```
Task
---------------------------------
# primary_label : String
# secondary_label : String
# priority_data : int
# process_time : long
# execution_time : long
# AirplaneID : int
- link : Link_object
- input : Scanner
---------------------------------
+ Task()
+ Task(String, String)
+ getPrimary label() : String
+ get secondarylabel() : String
+ set ExecutionTime(long) : void
+ get ExecutionTime() : void
+ setProcessTime() : void
+ Calculate Process Time(String) : long
+ get TaxiWay ID from User() : int
+ get RunWay ID from User() : int
+ get Gate ID from User() : int
+ get Airplane ID from User() : int
```

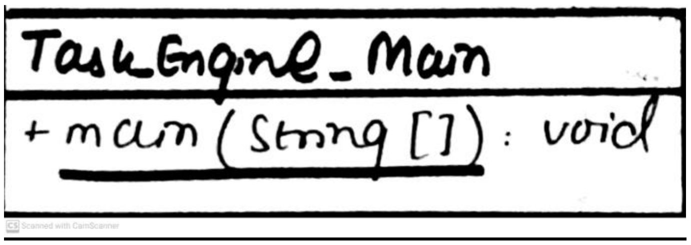
Control task:


ControllTask
# originalLink : int
# nextLink : int
# startingTime : Long
# endingTime : Long
- input : Scanner
+ ControllTask()
+ ControlTask(String, String)
+ getOriginalLinkFromUser() : int
+ getNextLinkFromUser() : int
+ calculateStartingTime() : Long
+ calculateEndingTime() : Long
+ closeScanner() : void

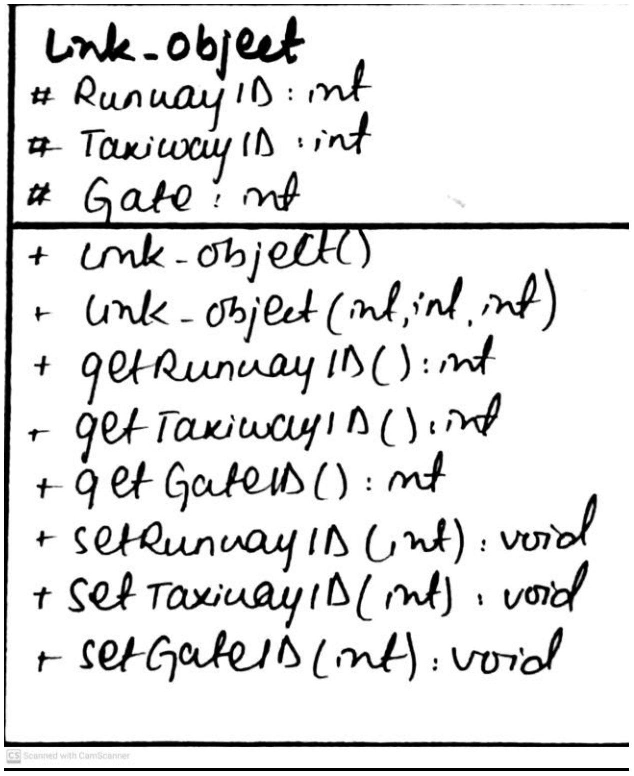
Task queue:



Task engine main:



Link object:

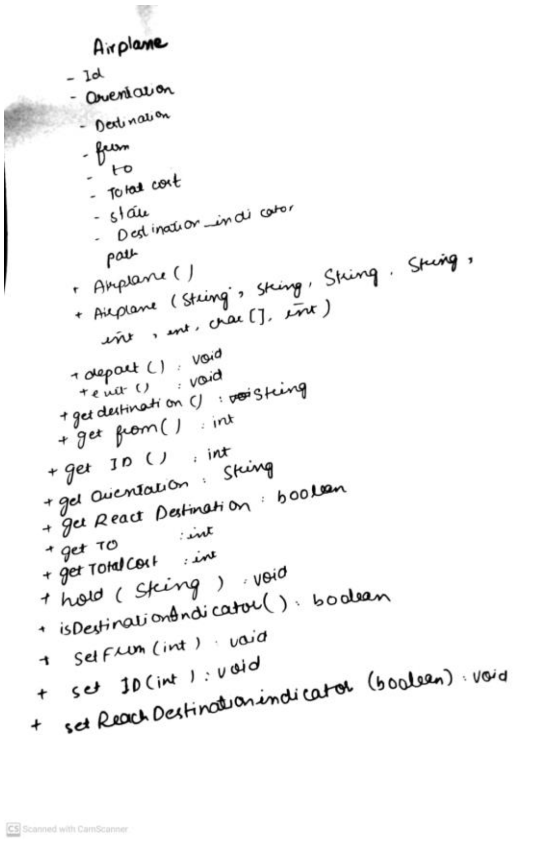


Link-object

RunwayID : int
TaxiwayID : int
Gate : int

+ Link-object()
+ Link-object(int,int,int)
+ getRunwayID() : int
+ getTaxiwayID() : int
+ getGateID() : int
+ setRunwayID(int) : void
+ setTaxiwayID(int) : void
+ setGateID(int) : void

Airplane:



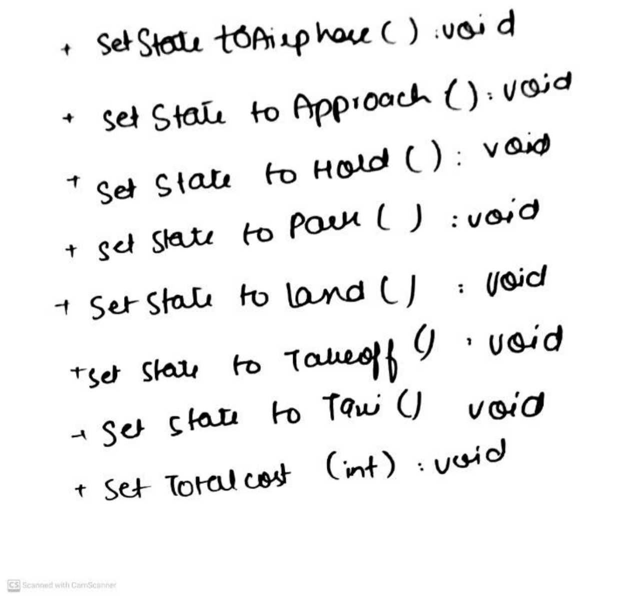
Airplane

- Id
- Orientation
- Destination
- from
- to
- Total cost
- state
- Destination_indicator

path

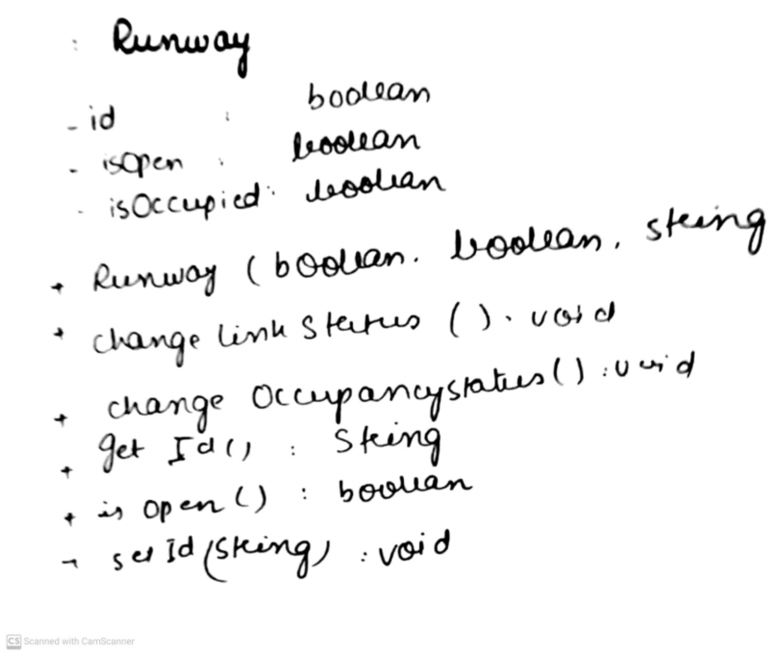
+ Airplane ( )
+ Airplane (String, String, String . String, int , int, char [], int)

+ depart ( ) : void
+ exit ( ) : void
+ get destination ( ) : ~~voi~~String
+ get from( ) : int
+ get ID ( ) : int
+ get Orientation : String
+ get React Destination : boolean
+ get TO : int
+ get TotalCost : int
+ hold (String) : void
+ isDestinationIndicator( ) : boolean
+ SetFrom (int) : void
+ set ID (int) : void
+ set ReachDestinationindicator (boolean) : void



+ SetState toAirplane ( ) : void
+ set State to Approach ( ) : void
+ set State to Hold ( ) : void
+ set State to Park ( ) : void
+ Set State to land ( ) : void
+ set state to Takeoff ( ) : void
+ set state to Taxi ( ) void
+ Set Totalcost (int) : void

Runway:

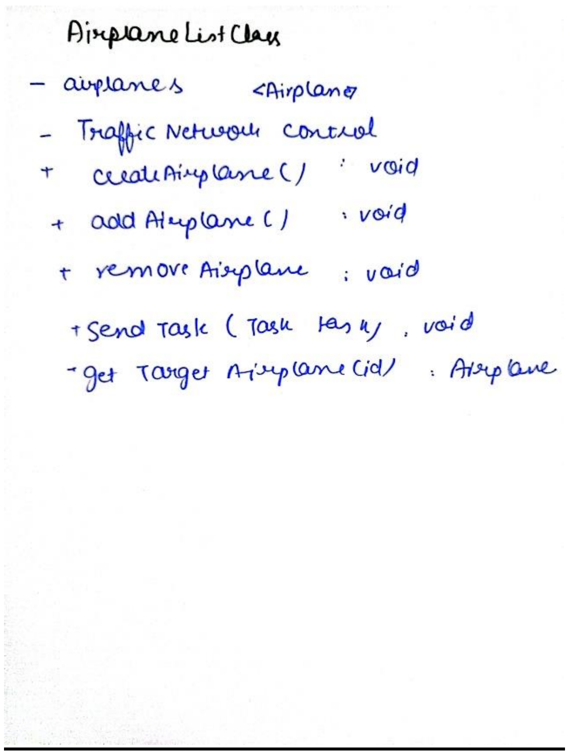


Runway

- id : boolean
- isOpen : boolean
- isOccupied : boolean

+ Runway (boolean. boolean, string
+ change link Status ( ) . void
+ change OccupancyStatus ( ) : void
+ get Id ( ) : String
+ is Open ( ) : boolean
+ set Id (String) : void

Airplane list class:



Airplane List Class

- airplanes <Airplane

- Traffic Network control

+ createAirplane() : void

+ add Airplane() : void

+ remove Airplane : void

+ Send Task (Task task) , void

- get Target Airplane (id) : Airplane

MATHEMATICAL MODELING:

Mathematical modeling is required at one instance in the project which is using a path finding algorithm known as Dijkstra’s algorithm, used to find the shortest path between the starting point and destination, basically finding the most optimized path for traveling which requires least cost.

CODES AND OUTPUT:

Shortest path code:

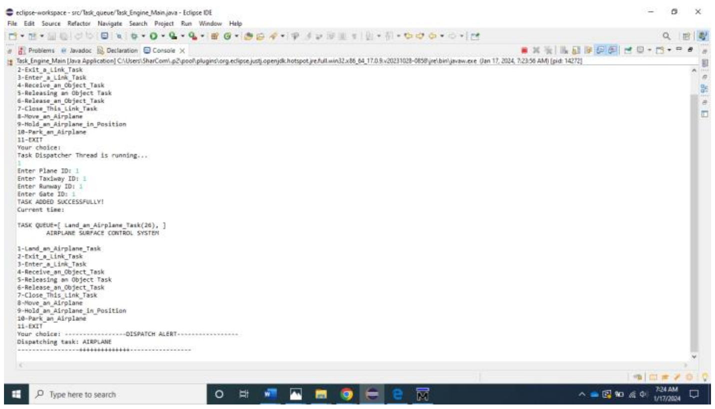
```java
package end_semester_project;

import java.util.*;

public class ShortestPathProcessor {
	private  Map<String, Map<String, Integer>> graph;
	  public ShortestPathProcessor() {}
	  public static void main(String[] args) {
		    Map<String, Map<String, Integer>> graph = new HashMap<>();

		    addEdge(graph, "Pak", "Russia", 7);
		    addEdge(graph, "Pak", "India", 10);
		    addEdge(graph, "Pak", "Usa", 12);
		    addEdge(graph, "Russia", "India", 4);
```

```
                  addEdge(graph, "Usa", "Russia", 23);
                  addEdge(graph, "Usa", "India", 11);

                  Scanner scanner = new Scanner(System.in);
                  System.out.println("Press 1 for Direct flight: ");
                  System.out.println("Press 2 for shortest flight");
                  int choice = scanner.nextInt();

                  System.out.print("Enter the departure point: ");
                  String departure = scanner.next().toUpperCase();
                  System.out.print("Enter the arrival point: ");
                  String arrival = scanner.next().toUpperCase();

                  if (choice == 1) {
                      // Direct flight
                      if (hasDirectFlight(graph, departure, arrival)) {
                          System.out.println("Direct flight available from " +
departure + " to " + arrival);
                      } else {
                          System.out.println("No direct flight available from " +
departure + " to " + arrival);
                      }
                  } else if (choice == 2) {
                      // Shortest flight
                      List<String> shortestPath = findShortestPath(graph, departure,
arrival);

                      if (!shortestPath.isEmpty()) {
                          System.out.println("Shortest Path from " + departure + " to
" + arrival + ": " + shortestPath);
                      } else {
                          System.out.println("No path found from " + departure + " to
" + arrival);
                      }
                  } else {
                      System.out.println("Invalid choice");
                  }

                  scanner.close();
              }

              private static boolean hasDirectFlight(Map<String, Map<String, Integer>>
graph, String departure, String arrival) {
                  return graph.containsKey(departure) &&
graph.get(departure).containsKey(arrival);
              }

            private static List<String> findShortestPath(Map<String, Map<String,
Integer>> graph, String start, String end) {
                 Map<String, Integer> distances = new HashMap<>();
                 Map<String, String> predecessors = new HashMap<>();
                 PriorityQueue<String> priorityQueue = new
PriorityQueue<>(Comparator.comparingInt(distances::get));
```

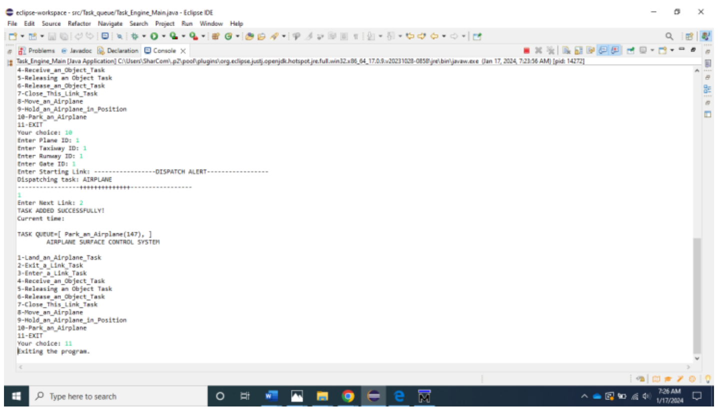
```
                // Initialize distances and predecessors
                for (String vertex : graph.keySet()) {
                    if (vertex.equals(start)) {
                        distances.put(vertex, 0);
                    } else {
                        distances.put(vertex, Integer.MAX_VALUE);
                    }
                    predecessors.put(vertex, null);
                    priorityQueue.add(vertex);
                }

                // Dijkstra's algorithm
                while (!priorityQueue.isEmpty()) {
                    String current = priorityQueue.poll();
                    for (String neighbor : graph.get(current).keySet()) {
                        int newDistance = distances.get(current) +
graph.get(current).get(neighbor);
                        if (newDistance < distances.get(neighbor)) {
                            distances.put(neighbor, newDistance);
                            predecessors.put(neighbor, current);
                            priorityQueue.remove(neighbor);
                            priorityQueue.add(neighbor);
                        }
                    }
                }

                // Reconstruct the path
                List<String> path = new ArrayList<>();
                String current = end;
                while (current != null) {
                    path.add(current);
                    current = predecessors.get(current);
                }
                Collections.reverse(path);

                return (List<String>) (path.size() > 1 ? path :
Collections.emptyList());
            }

            private static void addEdge(Map<String, Map<String, Integer>> graph, String
source, String destination, int weight) {
                graph.computeIfAbsent(source, k -> new HashMap<>()).put(destination,
weight);
                graph.computeIfAbsent(destination, k -> new HashMap<>()).put(source,
weight);
            }
           private void processShortestPath(String start, String destination) {
                    // Call ShortestPathProcessor method with the correct arguments
           List<String> shortestPath = findShortestPath(graph, start, destination);

                    if (!shortestPath.isEmpty()) {
                        System.out.println("Shortest Path from " + start + " to " +
destination + ": " + shortestPath);
                    } else {
                        System.out.println("No path found from " + start + " to " + 
destination);
                    }
                }


            }
```

Output:

Global clock and task queue:

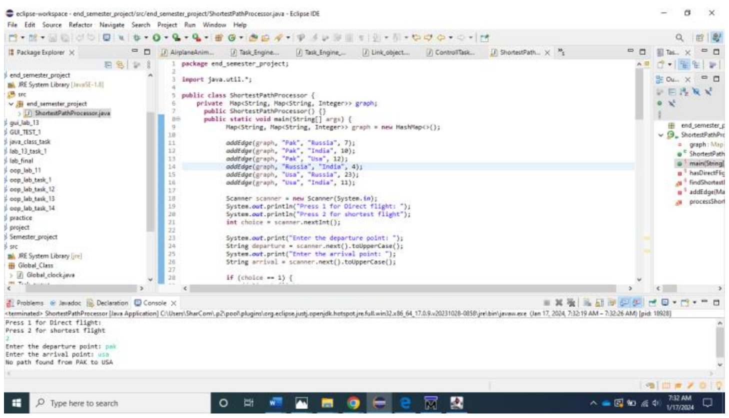


```
Task_Engine_Main [Java Application] C:\Users\SharCom\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Jan 17, 2024, 7:23:56 AM) [pid: 14272]
2-Exit_a_Link_Task
3-Enter_a_Link_Task
4-Receive_an_Object_Task
5-Releasing an Object Task
6-Release_an_Object_Task
7-Close_This_Link_Task
8-Move_an_Airplane
9-Hold_an_Airplane_in_Position
10-Park_an_Airplane
11-EXIT
Your choice:
Task Dispatcher Thread is running...
1
Enter Plane ID: 1
Enter Taxiway ID: 1
Enter Runway ID: 1
Enter Gate ID: 1
TASK ADDED SUCCESSFULLY!
Current time:

TASK QUEUE=[ Land_an_Airplane_Task(26), ]
        AIRPLANE SURFACE CONTROL SYSTEM

1-Land_an_Airplane_Task
2-Exit_a_Link_Task
3-Enter_a_Link_Task
4-Receive_an_Object_Task
5-Releasing an Object Task
6-Release_an_Object_Task
7-Close_This_Link_Task
8-Move_an_Airplane
9-Hold_an_Airplane_in_Position
10-Park_an_Airplane
11-EXIT
Your choice: -----------------DISPATCH ALERT-----------------
Dispatching task: AIRPLANE
-----------------++++++++++++++-----------------
```



```
Task_Engine_Main [Java Application] C:\Users\SharCom\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Jan 17, 2024, 7:23:56 AM) [pid: 14272]
4-Receive_an_Object_Task
5-Releasing an Object Task
6-Release_an_Object_Task
7-Close_This_Link_Task
8-Move_an_Airplane
9-Hold_an_Airplane_in_Position
10-Park_an_Airplane
11-EXIT
Your choice: 10
Enter Plane ID: 1
Enter Taxiway ID: 1
Enter Runway ID: 1
Enter Gate ID: 1
Enter Starting Link: -----------------DISPATCH ALERT-----------------
Dispatching task: AIRPLANE
-----------------++++++++++++++-----------------
1
Enter Next Link: 2
TASK ADDED SUCCESSFULLY!
Current time:

TASK QUEUE=[ Park_an_Airplane(147), ]
        AIRPLANE SURFACE CONTROL SYSTEM

1-Land_an_Airplane_Task
2-Exit_a_Link_Task
3-Enter_a_Link_Task
4-Receive_an_Object_Task
5-Releasing an Object Task
6-Release_an_Object_Task
7-Close_This_Link_Task
8-Move_an_Airplane
9-Hold_an_Airplane_in_Position
10-Park_an_Airplane
11-EXIT
Your choice: 11
Exiting the program.
```

Shortest path:



```
package end_semester_project;

import java.util.*;

public class ShortestPathProcessor {
    private  Map<String, Map<String, Integer>> graph;
      public ShortestPathProcessor() {}
      public static void main(String[] args) {
            Map<String, Map<String, Integer>> graph = new HashMap<>();

            addEdge(graph, "Pak", "Russia", 7);
            addEdge(graph, "Pak", "India", 10);
            addEdge(graph, "Pak", "Usa", 12);
            addEdge(graph, "Russia", "India", 4);
            addEdge(graph, "Usa", "Russia", 23);
            addEdge(graph, "Usa", "India", 11);

            Scanner scanner = new Scanner(System.in);
            System.out.println("Press 1 for Direct flight: ");
            System.out.println("Press 2 for shortest flight");
            int choice = scanner.nextInt();

            System.out.print("Enter the departure point: ");
            String departure = scanner.next().toUpperCase();
            System.out.print("Enter the arrival point: ");
            String arrival = scanner.next().toUpperCase();

            if (choice == 1) {
```

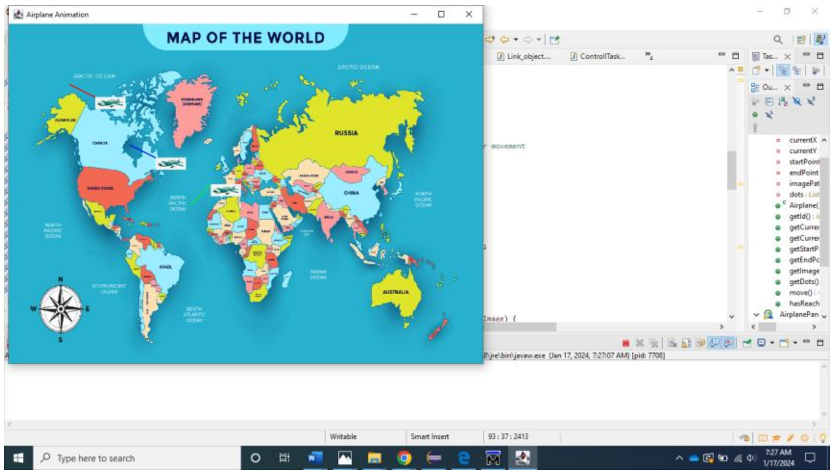
```
<terminated> ShortestPathProcessor [Java Application] C:\Users\SharCom\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Jan 17, 2024, 7:32:19 AM – 7:32:26 AM) [pid: 18928]
Press 1 for Direct flight:
Press 2 for shortest flight
2
Enter the departure point: pak
Enter the arrival point: usa
No path found from PAK to USA
```

Gui:

Code:

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.geom.Line2D;
import java.util.ArrayList;
import java.util.List;

class Airplane {
	//private double angle = 0.0; // Initial angle
    private int id;
    private double currentX;
    private double currentY;
    private Point startPoint;
    private Point endPoint;
    private String imagePath;
    private List<Point> dots;

    public Airplane(int id, Point startPoint, Point endPoint, String imagePath) {
        this.id = id;
        this.startPoint = startPoint;
        this.endPoint = endPoint;
        this.currentX = startPoint.x;
        this.currentY = startPoint.y;
        this.imagePath = imagePath;
        this.dots = new ArrayList<>();
    }

    public int getId() {
        return id;
    }

    public double getCurrentX() {
        return currentX;
    }

    public double getCurrentY() {
        return currentY;
    }

    public Point getStartPoint() {
        return startPoint;
    }

    public Point getEndPoint() {
        return endPoint;
    }

    public String getImagePath() {
        return imagePath;
    }

    public List<Point> getDots() {
        return dots;
    }

    public void move() {
        // Move airplane towards the end point
        double dx = endPoint.x - currentX;
        double dy = endPoint.y - currentY;
        double distance = Math.sqrt(dx * dx + dy * dy);

        double unitX = dx / distance;
        double unitY = dy / distance;
        double stepSize = 1.0; // Adjust the stepSize for smoother movement

        currentX += unitX * stepSize;
        currentY += unitY * stepSize;

        // Add the current position as a dot
        dots.add(new Point((int) currentX, (int) currentY));
    }


    public boolean hasReachedDestination() {
        return (currentX == endPoint.x && currentY == endPoint.y);
    }
}

class AirplanePanel extends JPanel {
    private List<Airplane> airplanes;
    private Image worldMapImage;

    public AirplanePanel(List<Airplane> airplanes, Image worldMapImage) {
        this.airplanes = airplanes;
        this.worldMapImage = worldMapImage;
    }

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

        // Draw world map background
        g.drawImage(worldMapImage, 0, 0, getWidth(), getHeight(), this);

        // Draw airplanes and paths
        for (Airplane airplane : airplanes) {
            drawDottedPath(g, airplane.getDots(), airplane.getId());
            drawAirplane(g, airplane);
        }
    }

    private void drawDottedPath(Graphics g, List<Point> dots, int id) {
        Graphics2D g2d = (Graphics2D) g;
        float[] dashPattern = {5, 5}; // Adjust the dash pattern as needed
        BasicStroke dashedStroke = new BasicStroke(2, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER, 10, dashPattern, 0);
        g2d.setStroke(dashedStroke);

        Color[] colors = {Color.RED, Color.BLUE, Color.GREEN}; // Different colors
for each plane
        g2d.setColor(colors[id - 1]);

        for (int i = 1; i < dots.size(); i++) {
            Point p1 = dots.get(i - 1);
            Point p2 = dots.get(i);
            g2d.drawLine(p1.x, p1.y, p2.x, p2.y);
        }
    }

    private void drawAirplane(Graphics g, Airplane airplane) {
        ImageIcon airplaneIcon = new ImageIcon(airplane.getImagePath());
        Image airplaneImage = airplaneIcon.getImage();
        g.drawImage(airplaneImage, (int) airplane.getCurrentX(), (int)
airplane.getCurrentY(), 50, 20, this);
    }
}

public class AirplaneAnimation extends JFrame {

    private List<Airplane> airplanes;
    private Image worldMapImage;

    public AirplaneAnimation() {
        setTitle("Airplane Animation");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Load world map image
        ImageIcon worldMapIcon = new ImageIcon("C:\\Users\\SharCom\\eclipse-
workspace\\Semester_project\\src\\airplane_animation\\world_map.jpg");
        worldMapImage = worldMapIcon.getImage();

        // Initialize airplanes
        airplanes = new ArrayList<>();
        airplanes.add(new Airplane(1, new Point(100, 100), new Point(500, 300),
"C:\\Users\\SharCom\\eclipse-
workspace\\Semester_project\\src\\airplane_animation\\airplane.jfif"));
        airplanes.add(new Airplane(2, new Point(200, 200), new Point(600, 400),
"C:\\Users\\SharCom\\eclipse-
workspace\\Semester_project\\src\\airplane_animation\\airplane.jfif"));
        airplanes.add(new Airplane(3, new Point(300, 300), new Point(400, 200),
"C:\\Users\\SharCom\\eclipse-
workspace\\Semester_project\\src\\airplane_animation\\airplane.jfif"));

        Timer timer = new Timer(100, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                moveAirplanes();
                repaint();
            }
        });

        timer.start();

        AirplanePanel airplanePanel = new AirplanePanel(airplanes, worldMapImage);
        add(airplanePanel);
    }

    private void moveAirplanes() {
        for (Airplane airplane : airplanes) {
            if (!airplane.hasReachedDestination()) {
                airplane.move();
            }
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            new AirplaneAnimation().setVisible(true);
        });
    }
}
```

Output:



```
\jre\bin\javaw.exe (Jan 17, 2024, 7:27:07 AM) [pid: 7708]
```

LIMITATIONS AND CONCLUSION:

After this project we can conclude that it is better to use object-oriented principles for modeling systems like this one, since it increases modularity hence helping in maintaining the software which as a result reduces the cost required for maintenance and makes code more efficient.