1.

```
package javaprefixstack;
import java.util.*;
/**
 *
 * @author aslas
 */
public class JavaPrefixStack {
Set<String> operatorsList = new HashSet<>(Arrays.asList("+","-","*","/"));

    public boolean checkOperator(String inputItem){
        return operatorsList.contains(inputItem);


    }

    public int prefixEvaluate(String Operator, String leftNum, String rightNum){
        int leftValue = Integer.parseInt(leftNum);
        int rightValue =Integer.parseInt(rightNum);

       switch(Operator) {
          case "+":
             return leftValue + rightValue;
          case "-":
             return leftValue - rightValue;
          case "*":
             return leftValue * rightValue;
          case "/":
             return leftValue / rightValue;
          default:
             System.out.println("Please input a correct operator: +,-,*,/.");
             return 0;
       }

    }

    public void createStack(String prefixExpression){
        String[] valuesArray = prefixExpression.split(",");


        Stack<String> prefixStack = new Stack<>();
        for (int i=valuesArray.length-1;i>=0;i--){
            if (checkOperator(valuesArray[i])){
                String leftValue = prefixStack.pop();
```

```
            String rightValue = prefixStack.pop();
           String result = prefixEvaluate(valuesArray[i],leftValue,rightValue) + "";
            prefixStack.push(result);
         }
         else{
            prefixStack.push(valuesArray[i]);
        }
      }
       System.out.println(prefixStack.pop());
   }


   public static void main(String[] args) {
       JavaPrefixStack PS = new JavaPrefixStack();
       System.out.println("Please enter a prefix equation (make sure to include spaces between all values
and operators):");
      Scanner scnr = new Scanner(System.in);
      String prefixProblem = scnr.nextLine();
       System.out.println("Your Problem is: "+ prefixProblem);
       System.out.print("Your Solution is: ");
      PS.createStack(prefixProblem);
   }

}
```

2.

```
import java.util.Random;
import java.util.Stack;
public class ArrayVLList {

    public static String[] createTest(int size){
    // Create test size (1,000,000) using push, pop, peek

    Random rnd = new Random();
    String[] operators = {"push", "pop", "peek"};
    String[] testData = new String[size];
    /*Here the for loop pushes a random operator to testData
      size() amount of times. Here we're doing 1,000,000 */
        for (int i = 0; i < size; i++) {
            testData[i] = operators[rnd.nextInt(operators.length)];
        }
        return testData;
    }

    public static long testStackSpeed(Stack<String> stack, String[] operators){
    // This is the method to actually test and time the two Stacks

    long start = System.currentTimeMillis();
        for (String ops : operators) {
            switch (ops){
                case "push":
                    stack.push("test");
                    break;
                case "pop":
                    if (!stack.isEmpty()) {
                        stack.pop();
                    }
                    break;
                case "peek":
                    if (!stack.isEmpty()) {
                        stack.peek();
                    }
                    break;
            }
        }
        long end = System.currentTimeMillis();
        return end - start;
    }



    public static void main(String[] args) {

        String[] testSize = createTest(1000000);
        Stack<String> arrayStack = new Stack<>();
        Stack<String> linkListStack = new Stack<>();

        // Test Array Stack
        long arrayStackTime = testStackSpeed(arrayStack, testSize);
```

```
        System.out.println("ArrayStack results: " + arrayStackTime + " milliseconds.");

        // Test Linked List Stack
        long LListStackTime = testStackSpeed(linkListStack, testSize);
        System.out.println("LListStack results: " + LListStackTime + " milliseconds.");
    }
}
```

3.

```
package javaapplication6;

/**
 *
 * @author aslas
 */
public class ArrayStack {
    private int[] myArray;
    private int top;
    public ArrayStack(int capacity){
        myArray = new int[capacity];
        top = 0;
    }

    public boolean empty(){
        return top ==0;
    }

   public void push(int pushValue){
       if(top == myArray.length+1){
           System.out.println("Stack Overflow");


       }
       else{

           myArray[top] = pushValue;
            top =top+1;
       }
   }
   int returnValue;
   public int pop(){
       if (empty()){
           System.out.println("Stack is empty");
       }
       else{
            returnValue = myArray[top];
           top=top-1;

       }
       return returnValue;
   }
```

```
   public int peek(){
      if (empty()){
          System.out.println("Stack is empty");
      }
      else{
         returnValue = myArray[top];

      }
      return returnValue;
   }
   public int size(){
      return myArray.length;
   }
    public void listAll(){
        for (int i=0; i<= myArray.length-2;i++){
            System.out.print(myArray[i]+ " ,");
        }
        int i = myArray.length -1;
        System.out.println(myArray[i]+"");
    }
}

package javaapplication1;

/**
 *
 * @author soblab
 */
import java.util.*;
/**
 *
 * @author soblab
 */
public class JavaApplication1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
       ArrayStack myStack  =new ArrayStack(10);
      for(int i=0; i<=myStack.size()-1;i++){
          Random rand = new Random();
          int randnum = rand.nextInt(100);
         myStack.push(randnum);
```

```
  }
   myStack.listAll();
}
```

4.

```
import java.util.Stack;

public class LongestIncreasingSubsequence {

    public static int longestIncreasingSubsequence(int[] arr) {
        if (arr == null || arr.length == 0) {
            return 0;
        }

        int n = arr.length;

        // lisLengths array stores the length of LIS ending at each index
        int[] lisLengths = new int[n];

        // Array-based stack to store indices of elements
        Stack<Integer> stack = new Stack<>();

        // Initialize the stack with the first index
        stack.push(0);

        for (int i = 1; i < n; i++) {
            // Remove elements from the stack if they are greater than or equal to the current element
            while (!stack.isEmpty() && arr[i] <= arr[stack.peek()]) {
                stack.pop();
            }

            // If the stack is not empty, update the LIS length using the top element of the stack
            if (!stack.isEmpty()) {
                lisLengths[i] = lisLengths[stack.peek()] + 1;
            } else {
                // If the stack is empty, the current element is the start of a new LIS
                lisLengths[i] = 1;
            }

            // Push the current index onto the stack
            stack.push(i);
        }

        // Find the maximum value in lisLengths array
        int maxLength = 0;
        for (int length : lisLengths) {
            maxLength = Math.max(maxLength, length);
        }
```

```
        return maxLength;
    }

    public static void main(String[] args) {
        int[] arr = {10, 22, 9, 33, 21, 50, 41, 60, 80};
        int result = longestIncreasingSubsequence(arr);
        System.out.println("Length of Longest Increasing Subsequence: " + result);
    }
}
```

5.

```
import java.util.*;
public class reversewords{
   // Push the words from the read line onto the stack
   public static void Sentence(String[] words){
      Stack <String> words_stack = new Stack <>();
      for (int i = 0; i < words.length; i++) {
         words_stack.push(words[i]);
      }
   }
   // Push the individual letters from the words on the Stack
   private static String reverseWord(String letters){
      Stack<Character> letters_stack = new Stack<>();
      for (char i = 0; i < letters.length(); i++) {
         letters_stack.push(letters.charAt(i));
      }

   // While Stack of letters is not empty keep popping letters off the Stack
      StringBuilder reverseWord = new StringBuilder();
      while(!letters_stack.empty()) {
         reverseWord.append(letters_stack.pop());
      }

   // Returns the letters in Last in First out order to a string to create a word/String
      return reverseWord.toString();
   }

   public static void main(String[] args) {
      System.out.print("Please enter your joke: ");

   // Scanner to read the words entered Line
      Scanner sc = new Scanner(System.in);
      String input = sc.nextLine();

   // This is where the line is split, "\\s+" is the delimiter meaning white space.
   // So it will break down the complete String into subStrings (words)
      String [] words = input.split("\\s+");

   // Calls the reverseWord method
      for (int i = 0; i < words.length; i++) {
      words[i] = reverseWord(words[i]);
      }


   // Reconstruct the reversed words in sentence
      String reversedSentence = String.join(" ", words);

   // Display the original and reversed sentences
      System.out.println("Original sentence: " + input);
      System.out.println("Reversed sentence: " + reversedSentence);
   }

}
```

6.

```
import java.util.Stack;

public class MazeSolver {
    // Define constants for maze symbols
    private static final char START = 'S';
    private static final char GOAL = 'G';
    private static final char WALL = 'W';
    private static final char VISITED = 'V';

    public static void main(String[] args) {
        // Define the maze
        char[][] maze = {
                {'S', ' ', ' ', ' ', 'W'},
                {'W', 'W', ' ', 'W', 'W'},
                {' ', ' ', ' ', ' ', ' '},
                {'W', 'W', 'W', 'W', 'W'},
                {'G', ' ', ' ', ' ', ' '}
        };

        // Create a stack to store the path
        Stack<int[]> stack = new Stack<>();

        // Find the starting point and push it onto the stack
        int[] start = findStart(maze);
        if (start != null) {
            stack.push(start);

            // Perform depth-first search until the stack is empty
            while (!stack.isEmpty()) {
                int[] current = stack.peek();
                int row = current[0];
                int col = current[1];

                // Check if the goal is reached
                if (maze[row][col] == GOAL) {
                    System.out.println("Maze solved!");
                    printSolution(stack);
                    return;
                }

                // Check if the current cell is not visited
```

```
            if (maze[row][col] != VISITED) {
                maze[row][col] = VISITED;

                // Explore neighbors (up, down, left, right)
                int[][] neighbors = {
                        {row - 1, col},  // Up
                        {row + 1, col},  // Down
                        {row, col - 1},  // Left
                        {row, col + 1}   // Right
                };

                // Try to push a valid neighbor onto the stack
                for (int[] neighbor : neighbors) {
                    int newRow = neighbor[0];
                    int newCol = neighbor[1];

                    if (isValid(newRow, newCol, maze.length, maze[0].length) &&
                            maze[newRow][newCol] != WALL && maze[newRow][newCol] != VISITED) {
                        stack.push(new int[]{newRow, newCol});
                        break;
                    }
                }
            } else {
                // Backtrack if the cell is already visited
                stack.pop();
            }
        }

        // If the stack is empty and the goal is not reached, no solution is found
        System.out.println("No solution found.");
    } else {
        System.out.println("Starting point not found.");
    }
}

// Helper method to find the starting point in the maze
private static int[] findStart(char[][] maze) {
    for (int i = 0; i < maze.length; i++) {
        for (int j = 0; j < maze[0].length; j++) {
            if (maze[i][j] == START) {
                return new int[]{i, j};
            }
        }
    }
```

```
        return null;
    }

    // Helper method to check if a cell is within the maze boundaries
    private static boolean isValid(int row, int col, int numRows, int numCols) {
        return row >= 0 && row < numRows && col >= 0 && col < numCols;
    }

    // Helper method to print the solution path
    private static void printSolution(Stack<int[]> solution) {
        System.out.println("Solution path:");
        for (int[] point : solution) {
            System.out.println("(" + point[0] + ", " + point[1] + ")");
        }
    }
}
```

7.

```
import java.util.Queue;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.Random;

public class PriorityQueue {

	// Creating 4 Priority Lists
	static Queue<String> P1 = new LinkedList<>();
	static Queue<String> P2 = new LinkedList<>();
	static Queue<String> P3 = new LinkedList<>();
	static Queue<String> P4 = new LinkedList<>();
	// Using static for all of these because they are going to be used throughout program
	static Random rndTime = new Random(15);
	static int count = 0;
	static int orderNum = 1;
	static String customerName;
	static int priority = 0;
	static int processingTime = 0;

	public static void main(String[] args) {
	Scanner sc = new Scanner(System.in);
	int choice = 0;
	// Creating menu
	while (true) {
	System.out.println("\n\nChoose an option: 1-4 \t" + "[Total count: " + count + "]");
	System.out.println("1. Create A New Order");
	System.out.println("2. Process Next Order");
	System.out.println("3. Process Random Order");
	System.out.println("4. View Order Queue\n");
	System.out.print("Selection: ");

	try{ // Accepting selection into temp location for sterilization
	choice = sc.nextInt();

	if (choice < 1 || choice > 4) {
		System.out.println("Invalid input. Please enter a valid number between 1 and 4.");
	} else {
		switch (choice) {
		case 1: newOrder();
		break;
		case 2: nextOrder();
		break;
		case 3: randomOrder();
		break;
		case 4: viewOrderQueue();
		break;
		default:System.out.println("Invalid input. Please enter a valid number between 1 and 4.");
		break;
		}
	}
	// This is to catch everything else that's not a number
	} catch (java.util.InputMismatchException e) {
```

```
System.out.println("Invalid input. Please enter a valid number between 1 and 4.");
sc.next();  // Clears invalid input
}
}
}

private static void newOrder() {

Scanner sc = new Scanner(System.in);
int pChoice = 0;

System.out.print("\nEnter Customer Name: ");
customerName = sc.nextLine();

while (true) { // Establishes the order Priority
System.out.print("Enter Priority 1-4: ");
pChoice = sc.nextInt();
if (pChoice < 1 || pChoice > 4) {
System.out.println("Invalid input. Please enter a valid number between 1 and 4.");
} else {
priority = pChoice;
break; // Break out of the loop if the priority is valid
}
}
orderNum++;
count++;
// Creating random processing times
processingTime = new Random().nextInt(15) + 1;
// Passing everything along to queueOrder
queueOrder(customerName, orderNum, priority, processingTime);
System.out.println("\n\n"+ customerName +" your order number is " + orderNum + " and you are in group " + priority + ".");


}

private static void nextOrder() {
// Processes the next order in queue
String order = null;
if (!P1.isEmpty()) {
order = P1.poll();

} else if (!P2.isEmpty()) {
order = P2.poll();

} else if (!P3.isEmpty()) {
order = P3.poll();

} else if (!P4.isEmpty()) {
order = P4.poll();

} else {
System.out.println("\nNo orders to process.\n");
return; // Return early if there are no orders to process
}
```

```
            System.out.println("\n**PROCESSING ORDER**\n" + order);
            // Process the order using orderProcessing
            orderProcessing(order);
            count--;
}

            private static String orderProcessing(String order) {
            // Pull information from the order string
            order = order.trim(); // Put this in because I was getting errors, so just wanted to clean up the order
            String[] orderDetails = order.split("[\\s\\n]+"); // Delimiters can be spaces, tabs, and newlines
                        // I was getting a Parsing integers from order error because of my formatting with new lines
            if (orderDetails.length < 4) {
            return "Invalid order format: " + order;

            }

            String customerName = orderDetails[0];
            try{ // Converting the String versions into Ints and storing in position[i]
            int orderNum = Integer.parseInt(orderDetails[1]);
            int priority = Integer.parseInt(orderDetails[2]);
            int processingTime = Integer.parseInt(orderDetails[3]);

            // Perform order processing actions
            System.out.println("\nNow Processing:");
            System.out.println("Customer Name:   " + customerName);
            System.out.println("Order No:          " + orderNum);
            System.out.println("Priority:            " + priority);
            System.out.println("Processing Time: " + processingTime + " minutes");
            count--;
            return null;

            } catch (NumberFormatException e) {
            return "Error parsing integers from order:\n" + order;

            }
            }

            private static void randomOrder() {
            Random rndOrder = new Random();

            // Select a random selection from P1, P2, P3, or P4
            int priority = rndOrder.nextInt(4) + 1;
            Queue<String> selectedQueue;

            switch (priority) {
            case 1:
            selectedQueue = P1;
            break;
            case 2:
            selectedQueue = P2;
            break;
            case 3:
            selectedQueue = P3;
            break;
```

```
case 4:
selectedQueue = P4;
break;
default:
System.out.println("Error: Invalid priority.");
count--;
return;
}

if (!selectedQueue.isEmpty()) {
// Polling the random order and passing it along to orderProcessing
String randomOrder = selectedQueue.poll();
System.out.println("\n**PROCESSING RANDOM ORDER**\n" + randomOrder);
orderProcessing(randomOrder);
} else {
System.out.println("\nNo orders in priority " + priority + " to process.\n");
}
}

private static void queueOrder(String customerName, int orderNum, int priority, int processingTime) {
// This is the method to (.add) the new orders to their proper LL
switch (priority) {
case 1: P1.add("\nName: "+ customerName +
            "\nOrder number: " + orderNum +
            "\nGroup: " + priority +
            "\nProcessing Time: " + processingTime + " minutes.\n");
break;
case 2: P2.add("\nName: "+ customerName +
            "\nOrder number: " + orderNum +
            "\nGroup: " + priority +
            "\nProcessing Time: " + processingTime + " minutes.\n");
break;
case 3: P3.add("\nName: "+ customerName +
            "\nOrder number: " + orderNum +
            "\nGroup: " + priority +
            "\nProcessing Time: " + processingTime + " minutes.\n");
break;
case 4: P4.add("\nName: "+ customerName +
            "\nOrder number: " + orderNum +
            "\nGroup: " + priority +
            "\nProcessing Time: " + processingTime + " minutes.\n");
break;

}
}

private static void viewOrderQueue() {
// This is option 4 from the table, and just shows what is the queue and the positions
System.out.println("Order Queue:\n\n");
int position = 1;

for (String order : P1) {
System.out.println("Position: " + position + order);
position++;
```

```
        }
        for (String order : P2) {
        System.out.println("Position: " + position + order);
        position++;
        }
        for (String order : P3) {
        System.out.println("Position: " + position + order);
        position++;
        }
        for (String order : P4) {
        System.out.println("Position: " + position + order);
        position++;
        }
        }
}
```

8.

```
import csc229.csc229_final_hw.Plane;
import java.util.*;
import java.lang.Math;
import java.time.LocalDateTime;

public class Airport {
   //create the 4 runways, used the 4 cardinal directions
   private Queue<String> northRunway = new LinkedList<>();
   private Queue<String> southRunway = new LinkedList<>();
   private Queue<String> eastRunway = new LinkedList<>();
   private Queue<String> westRunway = new LinkedList<>();
   //the queue of all planes
   private Queue<String> allPlanes = new LinkedList<>();

   //need the info from the Plane class
   public void assignRunway(LocalDateTime arrivalTime, String departurePoint,String planeID, String
destination) {
      //create random int between 1- 100 to act as percentage
      int ranPercent = (int) (Math.random() * 100) + 1;

      // acts as check for the 70%
      if (ranPercent <= 70) {
         // get random int 0-3 to act as the runway choice
         int ranDirection = (int) (Math.random() * 4);
         //create a switch case to add the correct runway to direction
         //add planeID and destination to the desired runway
         switch (ranDirection) {
            case 0:
               //adds to the north runway
               northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
               //adds to the overall queue
               allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
               break;
            case 1:
               southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
               allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
               break;
            case 2:
               eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
               allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
               break;
            case 3:
               westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
```

```
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
                break;
            }
        //acts as the check for 20%
        } else if (ranPercent <= 90) {
            //20% has two runways to choose from so it needs 2 random directions
            int ranDirection1 = (int) (Math.random() * 4);
            int ranDirection2 = (int) (Math.random() * 4);

            //check if the 2 runways are the same
            //if same redo random direction
            while (ranDirection2 == ranDirection1) {
                ranDirection2 = (int) (Math.random() * 4);
            }
            //if random direction 1 or R.D. 2 equal 0(north) then compair size
            if (ranDirection1 == 0 || ranDirection2 == 0) {
            // see if north is shorter then the rest if so add to its runway
                if (northRunway.size() <= southRunway.size() && northRunway.size() <= eastRunway.size() &&
northRunway.size() <= westRunway.size()) {
                    northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                    allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            // remove north and see if south is smaller then east and west
                } else if (southRunway.size() <= eastRunway.size() && southRunway.size() <= westRunway.size())
{
                    southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                    allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            // remove north and south. compare east size to west size
                } else if (eastRunway.size() <= westRunway.size()) {
                    eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                    allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            // only west is left so add to it runway
                } else {
                    westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                    allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
                }//end inner if
            // if R.D.1 or 2 are 1(south)
            } else if (ranDirection1 == 1 || ranDirection2 == 1) {
                //see if south runway is the smallest, if so then add
                if( southRunway.size() <= northRunway.size() && southRunway.size() <= eastRunway.size() &&
southRunway.size() <= westRunway.size()) {
                    southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                    allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
                //same logic as above
```

```
            } else if (northRunway.size() <= eastRunway.size() && northRunway.size() <= westRunway.size())
{
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
             //same logic as above
            }else if (eastRunway.size() <= westRunway.size()) {
                eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            } else {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            }  //end inner if
        // if R.D.1 or 2 are 2(east)
        } else if (ranDirection1 == 2 || ranDirection2 == 2){
            //see if east runway is the smallest, if so then add
            if( eastRunway.size() <= northRunway.size() && eastRunway.size() <= southRunway.size() &&
eastRunway.size() <= westRunway.size()) {
                eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            //same logic as above
             } else if (westRunway.size() <= northRunway.size() && westRunway.size() <=
southRunway.size()) {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
             //same logic as above
            }else if (northRunway.size() <= southRunway.size()) {
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            } else {
                southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            }  //end inner if
        // and finally if R.D.1 or 2 are 3(west)
        }else if (ranDirection1 == 3 || ranDirection2 == 3){
            //see if east runway is the smallest, if so then add
            if( westRunway.size() <= northRunway.size() && westRunway.size() <= southRunway.size() &&
westRunway.size() <= eastRunway.size()) {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            //same logic as above
             } else if (eastRunway.size() <= northRunway.size() && eastRunway.size() <= southRunway.size())
{
                eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
```

```
            //same logic as above
            }else if (northRunway.size() <= southRunway.size()) {
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            } else {
                southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            }  //end inner if
        }

    //checks everything else, 10%
    } else {
        //10% has 3 runways
        int ranDirection1 = (int) (Math.random() * 4);
        int ranDirection2 = (int) (Math.random() * 4);
        int ranDirection3 = (int) (Math.random() * 4);

        //check to make sure they are not the same runways
        while (ranDirection2 == ranDirection1 || ranDirection2 == ranDirection3) {
            ranDirection2 = (int) (Math.random() * 4);
        }
        while (ranDirection3 == ranDirection1 || ranDirection3 == ranDirection2) {
            ranDirection3 = (int) (Math.random() * 4);
        }
        // if R.D.1 2 or 3  are 0(north)
        if (ranDirection1 == 0 || ranDirection2 == 0 || ranDirection3 == 0) {
        // see if north is shorter then the rest if so add to its runway
            if (northRunway.size() <= southRunway.size() && northRunway.size() <= eastRunway.size() &&
northRunway.size() <= westRunway.size()) {
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
        // remove north and see if south is smaller then east and west
            } else if (southRunway.size() <= eastRunway.size() && southRunway.size() <= westRunway.size())
{
                southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
        // remove north and south. compare east size to west size
            } else if (eastRunway.size() <= westRunway.size()) {
                eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
        // only west is left so add to it runway
            } else {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
```

```
            }//end inner if
        // if R.D.1 2 or 3  are 1(south)
        } else if (ranDirection1 == 1 || ranDirection2 == 1 || ranDirection3 == 1) {
            //see if south runway is the smallest, if so then add
            if( southRunway.size() <= northRunway.size() && southRunway.size() <= eastRunway.size() &&
southRunway.size() <= westRunway.size()) {
                southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            //same logic as above
             } else if (northRunway.size() <= eastRunway.size() && northRunway.size() <= westRunway.size())
{
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
             //same logic as above
            }else if (eastRunway.size() <= westRunway.size()) {
                eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            } else {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            }  //end inner if
        // if R.D.1 2 or 3 are 2(east)
        } else if (ranDirection1 == 2 || ranDirection2 == 2 || ranDirection3 == 2){
            //see if east runway is the smallest, if so then add
            if( eastRunway.size() <= northRunway.size() && eastRunway.size() <= southRunway.size() &&
eastRunway.size() <= westRunway.size()) {
              eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
              allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            //same logic as above
             } else if (westRunway.size() <= northRunway.size() && westRunway.size() <=
southRunway.size()) {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
             //same logic as above
            }else if (northRunway.size() <= southRunway.size()) {
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            } else {
                southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            }  //end inner if
        // and finally if R.D.1 2 or 3 are 3(west)
        }else if (ranDirection1 == 3 || ranDirection2 == 3 || ranDirection3 == 3){
            //see if east runway is the smallest, if so then add
```

```
            if( westRunway.size() <= northRunway.size() && westRunway.size() <= southRunway.size() &&
westRunway.size() <= eastRunway.size()) {
                westRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            //same logic as above
             } else if (eastRunway.size() <= northRunway.size() && eastRunway.size() <= southRunway.size())
{
                eastRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
             //same logic as above
            }else if (northRunway.size() <= southRunway.size()) {
                northRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            } else {
                southRunway.add("Plane ID: " + planeID + ", Destination: " + destination);
                allPlanes.add("Plane ID: " + planeID + ", Destination: " + destination);
            }  //end inner if
          }
       }//end else
   }//end assign
   //the queue getters
   public Queue getNorthRunPlanes() {
      return northRunway;
   }
   public Queue getSouthRunPlanes() {
      return southRunway;
   }
   public Queue getEastRunPlanes() {
      return eastRunway;
   }
   public Queue getWestRunPlanes() {
      return westRunway;
   }
   public Queue getAllPlanes() {
      return allPlanes;
   }
   //the printer
   public void printPlanes( Queue<String> queue) {
      for(String plane: queue){
         System.out.println(plane);
      }
   }
}//end airport
package csc229.csc229_final_hw;
```

```
import java.time.LocalDateTime;
public class Plane {
    //set up of plane details
    private String planeID;
    private String destination;
    private LocalDateTime arrivalTime;
    private LocalDateTime departureTime;
    private String departurePoint;
    //constructor
    public Plane (String planeID,String destination, LocalDateTime arrivalTime,
                  LocalDateTime departureTime, String departurePoint ){
        this.planeID = planeID;
        this.destination = destination;
        this.arrivalTime = arrivalTime;
        this.departureTime = departureTime;
        this.departurePoint = departurePoint;
    }
    //setters
    public void setPlaneID(String planeID){
        this.planeID = planeID;
    }
    public void setDestination(String destination){
        this.destination = destination;
    }
    public void setArivalTime(LocalDateTime arrivalTime){
        this.arrivalTime = arrivalTime;
    }
    public void setDeparture(LocalDateTime departureTime){
        this.departureTime = departureTime;
    }
    public void setDeparturePoint(String departurePoint) {
        this.departurePoint = departurePoint;
    }
   //getters
    public String getPlaneID(){
        return planeID;
    }
    public String getDestination(){
        return destination;
    }
    public LocalDateTime getArivalTime(){
        return arrivalTime;
    }
    public LocalDateTime getDeparture(){
```

```
        return departureTime;
    }
    public String getDeparturePoint() {
        return departurePoint;
    }
}//end of plane

Import java.util.*;
public static void main(String[] args) {

    Airport airport = new Airport();
    LocalDateTime arrivalTime = LocalDateTime.now();
    //first create your plane instances
    Plane plane1 = new Plane("FLY123", "New York", LocalDateTime.now(),
LocalDateTime.now().plusHours(2), "North");
    Plane plane2 = new Plane("DLT449", "Los Angeles", LocalDateTime.now(),
LocalDateTime.now().plusHours(3), "South");
    Plane plane3 = new Plane("BLU293", "Boston",LocalDateTime.now(),LocalDateTime.now().plusHours(5),
"East" );
    Plane plane4 = new Plane("TNY888", "San
Diego",LocalDateTime.now(),LocalDateTime.now().plusHours(7), "West" );
    Plane plane5 = new Plane("AIR222", "Pensacola", LocalDateTime.now(),
LocalDateTime.now().plusHours(1), "South");
    Plane plane6 = new Plane("UNT483", "Falon", LocalDateTime.now(), LocalDateTime.now().plusHours(4),
"East");
    Plane plane7 = new Plane("SPR243",
"Olympia",LocalDateTime.now(),LocalDateTime.now().plusHours(2), "West" );
    Plane plane8 = new Plane("PLN636",
"Woodstock",LocalDateTime.now(),LocalDateTime.now().plusHours(3), "North" );

//put the planes at the airport
    airport.assignRunway(plane1.getArivalTime(), plane1.getDeparturePoint(), plane1.getPlaneID(),
plane1.getDestination());
    airport.assignRunway(plane2.getArivalTime(), plane2.getDeparturePoint(), plane2.getPlaneID(),
plane2.getDestination());
    airport.assignRunway(plane3.getArivalTime(), plane3.getDeparturePoint(), plane3.getPlaneID(),
plane3.getDestination());
    airport.assignRunway(plane4.getArivalTime(), plane4.getDeparturePoint(), plane4.getPlaneID(),
plane4.getDestination());
    airport.assignRunway(plane5.getArivalTime(), plane5.getDeparturePoint(), plane5.getPlaneID(),
plane5.getDestination());
    airport.assignRunway(plane6.getArivalTime(), plane6.getDeparturePoint(), plane6.getPlaneID(),
plane6.getDestination());
```

```
    airport.assignRunway(plane7.getArivalTime(), plane7.getDeparturePoint(), plane7.getPlaneID(),
plane7.getDestination());
    airport.assignRunway(plane8.getArivalTime(), plane8.getDeparturePoint(), plane8.getPlaneID(),
plane8.getDestination());


    //prints runways
    System.out.println("North Runway:");
    airport.printPlanes(airport.getNorthRunPlanes());
    System.out.println("---------------------------------");
    System.out.println("South Runway:");
    airport.printPlanes(airport.getSouthRunPlanes());
    System.out.println("---------------------------------");
    System.out.println("East Runway:");
    airport.printPlanes(airport.getEastRunPlanes());
    System.out.println("---------------------------------");
    System.out.println("West Runway:");
    airport.printPlanes(airport.getWestRunPlanes());
    System.out.println("---------------------------------");
    System.out.println("The Plane Queue:");
    airport.printPlanes(airport.getAllPlanes());

    }
}
```