Jackie Zheng and Anthony Polanco

- What ADTs you used and the rationale **for** your choices
ADT used is QueueCRAB and ListArrayBasedPlus. We used QueueCRAB since we knew that queue is what we should use **for** airplanes since we only need the beginning and not needed to change anything other than that. Itâ\200\231s also the most efficient compared to the other queue ADTs. ListArrayBased was our choice **for** storing runways since we need a direct index access. Itâ\200\231s always going to be more than half full and **this** list uses less memory compared to others when itâ\200\231s half full.

- What instances of ADTs you used, why you used them and what you used them **for**
Plane and Runway. We used them **for** keeping the info such as flight number, destination, and runway in planes. We used them since its the most logical way of containing that information.
The Runway **class** contains the queueCRAB of planes where we have getters and setters which also contain additional information such as runway name.
- **for** each menu option, present the data flow and the instances affected
1. Plane enters the system.
Adds a plane into the runway list by providing planes flight number, destination, the location of which runway it goes to.

2. Plane takes off.
Each time **this** is chosen the next available plane gets removed from the runway list. The runway that removes a plane is chosen in a round robin fashion each time **this** option is pressed the next available plane in the next runway gets removed. The user can either deny or allow the plane to take off, **if** the user denies it then it goes to a separate list (waiting list) to await re-entry

3. Plane is allowed to re-enter a runway.
Ask the user to input a flight number, and runs through the collection to see **if** the plane is in there. If not, keep prompting the user to enter a plane that exist in the collection. Otherwise, when the collection is empty, it would just display its empty. The Plane that is chosen will re-enter a runway list.

4. Runway opens.
Add a **new** runway into the runway collection by asking the user to provide the name which must not currently exist in the collection.

5. Runway closes.
Closes the given runway, and adds each plane to an existing runway list depending on where the user wants each plane to go to, by going through the runway collection until all planes are moved out. Once there's no planes left, the runway is removed from the airport.

6. Display info about planes waiting to take off.
Runs through the list of runaways and display each plane in the runway, and once it displays all the planes in that runway, we move on to the next runway and display the planes until all planes in each of the runway is displayed.

7. Display info about planes waiting to be allowed to re-enter a runway.
Displays the planes in the waiting list to be re-entered into the runway. It will display all the planes in there and their destination.

8. Display number of planes who have taken off.
Have a counter variable in the driver and increase the counter when a plane takes off. When option 8 is called, we would just basically print the counter variable.

- the classes you have used and their functionality
Driver, ListArrayBased, ListArrayBasedPlus, Plane, QueueCRAB, and Runway. Driver does all the user question and input and output. ListArrayBased and ListArrayBasedPlus are ADT are arrays with direct index access which will store the Runway. The plane contains all the information about the flight and destination. The QueueCRAB is a queue that store the planes and handle the takeoff and adding to the runway. The runway contains the queue crab that can remove and add queue and get/set runwayname.

- what is the most frequent operation that is expected to be performed during the entire execution of the program and how your choices optimize that frequent operation. The choices could include the choice of a specific ADT or specific implementation.
Most frequent operation is searchRunWay where we would look and see **if** the runway exists or not, **if** not, it would **return** -1 and otherwise would **return** the index of that runway found.

Jackie - Plane
Driver Layout
Option1
Option7
Option8
Option6

Anthony -Runway
Option2
Option4
Option5
Option3

```
import java.util.Scanner;
// TODO: Auto-generated Javadoc
/*

 * Purpose: A airport system that adds and removes planes from runways

 * Status: Complete and thoroughly tested

 * Last update: 12/05/19

 * Submitted:  12/05/19

 * Comment: test suite and sample run attached

 * @author: Anthony Polanco / Jackie Zheng

 * @version: 2019.12.05

 */

/**
 * The Class Driver.
 */
public class Driver {

    /** The sc. */
    static Scanner sc;

    /** The number. */
    static int number;

    /** The round robin. */
    static int roundRobin = 0;

    /** The wait. */
    static int wait = 0;
```

```
    /**
     * The main method.
     *
     * @param args the arguments
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        sc = new Scanner(System.in);

        ListArrayBasedPlus<Runway> airport = new ListArrayBasedPlus<Runway>();

        ListArrayBasedPlus<Plane> waiting = new ListArrayBasedPlus<Plane>();
        System.out.print("Welcome to the Airport program!\nEnter number of runways
: ");
        int runways = sc.nextInt();
        System.out.print(runways);

        for (int i = 1; i <= runways; i++) {
            System.out.print("\nEnter the name of runway number " + i + ":");
            String runwayName = sc.next();
            System.out.print(runwayName);
            Runway run = new Runway(runwayName);
            airport.add(i - 1, run);
        }

        System.out.println("\nSelect from the following menu:\r\n" + "  0. Exit pr
ogram.\r\n"
                           + "  1. Plane enters the system.\r\n" + "  2. Plane tak
es off.\r\n"
                           + "  3. Plane is allowed to re-enter a runway.\r\n" + "
  4. Runway opens.\r\n"
                           + "  5. Runway closes.\r\n" + "  6. Display info about
planes waiting to take off.\r\n"
                           + "  7. Display info about planes waiting to be allowed
 to re-enter a runway.\r\n"
                           + "  8. Display number of planes who have taken off.");

        boolean check = true;

        while (check == true) {
            try {
                int input;
                System.out.print("\nMake your menu selection now: ");
                input = sc.nextInt();
                System.out.print(input + "\n");

                switch (input) {
                case 0: {
                    check = false;
                    end();
                    break;
                }
                case 1: {

                    enter(airport);

                    break;
                }
                case 2: {
                    takeOff(airport, waiting);

                    break;
                }
                case 3: {
                    reEnter(airport, waiting);

                    break;
                }
                case 4: {
                    runwayOpen(airport);

                    break;
                }
                case 5: {
                    runwayClose(airport, waiting);

                    break;
                }
                case 6: {
                    displayWaitingToTakeOff(airport);

                    break;
                }
                case 7: {
                    displayWaitingToReEnter(waiting);
                    break;
                }
                case 8: {
                    displayNumberOfPlanes();

                    break;
                }
                }
            } catch (ListException | ListIndexOutOfBoundsException | QueueExceptio
n e) {
                System.out.print("Error Exception!\n");
            }
        }
    }

    /**
     * End.
     */
    public static void end() { // ends the program
        System.out.print("The Airport is closing :Bye Bye....");
    }

    /**
     * enters a plane in the runway
     *
     * @param airport the airport
     */
    public static void enter(ListArrayBasedPlus<Runway> airport) {

        System.out.print("Enter flight number: ");

        String flightNumber = sc.next();
        System.out.print(flightNumber);
        if (checkIfValid(airport, flightNumber)) {
            while (checkIfValid(airport, flightNumber)) {
                System.out.print("Enter flight number: ");

                flightNumber = sc.next();
```

```
                System.out.print(flightNumber);
            }
        }

        System.out.print("\nEnter destination: ");
        String destination = sc.next();
        System.out.print(destination+"\n");
        String runway = "";
        int check = -1;
        while (check == -1) {

            System.out.print("Enter runway: ");

            runway = sc.next();
            System.out.print(runway + "\n");
            check = searchRunWay(airport, runway);
            if (check == -1) {
                System.out.print("No such runway! \n");

            }
        }

        Plane newPlane = new Plane(flightNumber, destination, runway);
        int index = check;
        Runway run = airport.get(index);

        run.addPlane(newPlane);
        System.out.print("Flight " + flightNumber + " is now waiting for takeoff o
n runway " + runway + ".\n");
    }

    /**
     * makes a plane takeoff in a roundRobin
     * fashion and asks the user if they want to make the plane takeoff
     * or await to enter runway
     *
     * @param airport the airport
     * @param waiting the waiting
     */
    public static void takeOff(ListArrayBasedPlus<Runway> airport, ListArrayBasedP
lus<Plane> waiting) {

        boolean flag = true;

        for (int i = 0; i < airport.size() && flag; i++) {

            if (airport.get(roundRobin).isEmpty()) {

                roundRobin = (roundRobin + 1) % airport.size();
            } else {

                String name = airport.get(roundRobin).getPlane().getFlightNumber()
;
                System.out.print("Is flight" + name + "cleared for takeoff(Y/N): "
);
                String option = sc.next();
                System.out.print(option+"\n");
                if (option.contentEquals("Y")) {

                    airport.get(roundRobin).removePlane();
                    System.out.print("Flight " + name + " has now taken off from r
unway "
                                             + airport.get(roundRobin).getRunwayName()+"\n
");
                    number++;
                } else {

                    waiting.add(wait, airport.get(roundRobin).removePlane());
                    System.out.print("Flight " + name + " is now waiting to be all
owed to re-enter a runway \n");

                    wait++;
                }
                roundRobin = (roundRobin + 1) % airport.size();
                flag = false;

            }

        }

        if (flag == true) {

            System.out.print("All runways are empty \n");
        }

    }

    /**
     * re-enters a plane specified by the user
     * thats waiting to re-enter a runway
     *
     * @param airport the airport
     * @param waiting the waiting
     */
    public static void reEnter(ListArrayBasedPlus<Runway> airport, ListArrayBasedP
lus<Plane> waiting) {

        System.out.print("Enter flight number: ");

        String flightNumber = sc.next();
        System.out.print(flightNumber + "\n");
        boolean flag = true;
        if (waiting.isEmpty()) {
            System.out.println("There are no planes waiting for clearance!\n ");

        } else {


            for (int i = 0; i < waiting.size() && flag; i++) {

                if (waiting.get(i).getFlightNumber().equals(flightNumber)) {

                    flag = false;

                    Plane plane = waiting.get(i);

                    int index = searchRunWay(airport, plane.getRunway());

                    Runway run = airport.get(index);

                    run.addPlane(plane);
                    waiting.remove(i);
                    System.out.print("Flight " + plane.getFlightNumber() + " is no
w waiting for takeoff on runway "
```

```
                                    + run.getRunwayName() + "\n ");


                }
            }

        }

    }

    /**
     * opens a new runway
     *
     * @param airport the airport
     */
    public static void runwayOpen(ListArrayBasedPlus<Runway> airport) {
        boolean flag = true;
        while (flag == true) {
            System.out.print("Enter the name of the new runway : ");
            String name = sc.next();
            System.out.print(name + "\n");
            if (searchRunWay(airport, name) == -1) {
                airport.add(airport.size(), new Runway(name));
                System.out.print("Runway " + name + " has opened.\n");
                flag = false;
            } else {
                System.out.print("Runway " + name + " already exists, please choos
e another name.\n");
            }
        }
    }

    /**
     * closes a runway specified by the user and asks the user where
     * to move all of the planes in the runway
     *
     * @param airport the airport
     * @param waiting the waiting
     */
    public static void runwayClose(ListArrayBasedPlus<Runway> airport, ListArrayBa
sedPlus<Plane> waiting) {
        int way = 0;
        boolean flag = true;
        int check = -1;
        String runway = "";
        while (check == -1) {

            System.out.print("Enter runway: ");

            runway = sc.next();
            System.out.print(runway + "\n");
            check = searchRunWay(airport, runway);
            if (check == -1) {
                System.out.print("No such runway! \n");

            }
        }

        for (int i = 0; i < airport.size() && flag; i++) {

            if (airport.get(i).getRunwayName().contentEquals(runway)) {
                flag = false;
                way = i;
                Runway run = airport.get(i);
                for (int j = 0; j <= run.getSize(); j++) {

                    if (!run.isEmpty()) {

                        Plane plane = run.removePlane();

                        System.out.print("Enter new runway for plane " + plane.get
FlightNumber() + ":");

                        String newRunName = sc.next();
                        System.out.print(newRunName + "\n");
                        while (run.getRunwayName().equals(newRunName) || check ==
-1) {

                            System.out.print("Enter new runway for plane " + plane
.getFlightNumber() + ":");

                            newRunName = sc.next();
                            System.out.print(newRunName + "\n");
                            check = searchRunWay(airport, newRunName);

                            if(run.getRunwayName().equals(newRunName)) {
                                System.out.print("This is the runway that is closi
ng!\n");
                            }

                            if (check == -1) {
                                System.out.print("No such runway! \n");

                            }
                        }


                        System.out.print("Flight " + plane.getFlightNumber() + "
is now waiting for takeoff on runway "
                                + newRunName + "\n");
                        plane.setRunway(newRunName);

                        Runway newRun = airport.get(searchRunWay(airport, newRunNa
me));

                        newRun.addPlane(plane);

                    }
                }
            }
        }


        for(int i = 0; i < waiting.size(); i++) {
            if(waiting.get(i).getRunway().equals(runway)) {

                Plane plane = waiting.get(i);
                waiting.remove(i);
                System.out.print("Enter new runway for plane " + plane.getFlightNu
mber() + ":");

                String newRunName = sc.next();
                System.out.print(newRunName + "\n");
                while (runway.equals(newRunName)) {
```

```
                    System.out.print("This is the runway that is closing!");
                    System.out.print("Enter new runway for plane " + plane.getFlig
htNumber() + ":");
                    newRunName = sc.next();
                }

                System.out.print(
                    "Flight " + plane.getFlightNumber() + " has changed runway to
" + newRunName + "\n");

                plane.setRunway(newRunName);
            }

        }

        System.out.println("Runway "+runway+" has been closed.");
        airport.remove(way);
    }

    /**
     * Display waiting to take off.
     *
     * @param airport the airport
     */
    public static void displayWaitingToTakeOff(ListArrayBasedPlus<Runway> airport)
 {
        for(int i = 0; i < airport.size(); i++) {
            if(airport.get(i).getQueue().isEmpty() != true) {
                System.out.print(
                    "These planes are waiting for takeoff on runway " + airport.ge
t(i).getRunwayName() + " :");
                System.out.println(airport.get(i).toString());
            } else {
                System.out
                .print("No planes are waiting for takeoff on runway " + airport.ge
t(i).getRunwayName() + "!\n");
            }
        }
    }

    /**
     * Display waiting to re enter.
     *
     * @param waiting the waiting
     */
    public static void displayWaitingToReEnter(ListArrayBasedPlus<Plane> waiting)
{
        if(waiting.isEmpty() == true) {
            System.out.print("No planes are waiting to be cleared to re-enter a ru
nway!\n");
        } else {
            System.out.print("These planes are waiting to be cleared to re-enter a
 runway:\n");

            for(int i = 0; i < waiting.numItems; i++) {
                Plane plane = waiting.get(i);
                System.out.print("Flight " + plane.getFlightNumber() + " to " + pl
ane.getDestination() + "\n");
            }
        }
    }

    /**
     * Display number of planes.
     */
    public static void displayNumberOfPlanes() {
        System.out.print(number + " planes have taken off from the airport.\n");
    }

    /**
     * this class searches through the airport for a specific class
     *
     * @param airport the airport
     * @param name the name
     * @return index returns the index of where the runway is and if its not in it
 returns -1
     */
    public static int searchRunWay(ListArrayBasedPlus<Runway> airport, String name
) {
        boolean flag = false;
        int index = -1;
        for(int i = 0; i < airport.size() && flag != true; i++) {
            if(airport.get(i) != null) {
                if((airport.get(i)).getRunwayName().equals(name)) {
                    flag = true;
                    index = i;
                }

            }
        }
        return index;
    }

    /**
     * this class checks if the flightnumber is valid or not and if theres any dup
licates
     *
     * @param airport the airport
     * @param flightNumber the flight number
     * @return true, if successful
     */
    public static boolean checkIfValid(ListArrayBasedPlus<Runway> airport, String
flightNumber) {
        boolean alreadyInuse = false;

        for(int i = 0; i < airport.size(); i++) {

            Runway run = airport.get(i);
            for(int j = 0; i < run.getSize(); i++) {

                if(run.getPlane().getFlightNumber().contentEquals(flightNumber)) {

                    System.out.print("Enter another flightNumber already in use \n
");

                    alreadyInuse = true;

                }
            }
        }
        return alreadyInuse;

    }
```

```
}


/*

 * Purpose: makes planes sets and gets attributes for the plane

 * Status: Complete and thoroughly tested

 * Last update: 12/05/19

 * Submitted:  12/05/19

 * Comment: test suite and sample run attached

 * @author: Anthony Polanco / Jackie Zheng

 * @version: 2019.12.05

 */
/**
 * The Class Plane which has the flight number, destination, and runway.
 */
public class Plane {

    /** The flight number. */
    private String flightNumber;

    /** The destination. */
    private String destination;

    /** The runway. */
    private String runway;

    /**
     * Creates a new plane.
     *
     * @param flight the flight
     * @param dest the destination
     * @param run the run
     */
    public Plane(String flight, String dest, String run) {
        flightNumber = flight;
        destination = dest;
        runway = run;
    }

    /**
     * Gets the flight number.
     *
     * @return the flight number
     */
    public String getFlightNumber() {
        return flightNumber;
    }

    /**
     * Sets the flight number.
     *
     * @param flightNumber the new flight number
     */
    public void setFlightNumber(String flightNumber) {
        this.flightNumber = flightNumber;
    }

    /**
     * Gets the destination.
     *
     * @return the destination
     */
    public String getDestination() {
        return destination;
    }

    /**
     * Sets the destination.
     *
     * @param destination the new destination
     */
    public void setDestination(String destination) {
        this.destination = destination;
    }

    /**
     * Gets the runway.
     *
     * @return the runway
     */
    public String getRunway() {
        return runway;
    }

    /**
     * Sets the runway.
     *
     * @param runway the new runway
     */
    public void setRunway(String runway) {
        this.runway = runway;
    }
}

/*

 * Purpose: A runWay class that holds planes

 * Status: Complete and thoroughly tested

 * Last update: 12/05/19

 * Submitted:  12/05/19

 * Comment: test suite and sample run attached

 * @author: Anthony Polanco / Jackie Zheng

 * @version: 2019.12.05

 */
/**
 * The Class Runway.
 */
public class Runway {
```

```
/** The queue containing Plane type. */
private QueueCRAB<Plane> queue;

/** The runway name. */
private String runwayName;

/**
 * Create a new runway.
 *
 * @param name the name
 */
public Runway (String name) {
    runwayName = name;
    queue = new QueueCRAB<Plane>();
}

/**
 * Create a new runway.
 */
public Runway () {
}

/**
 * Adds the plane into the queue.
 *
 * @param plane the plane
 */
public void addPlane(Plane plane) {
    queue.enqueue(plane);
}

/**
 * Removes the plane from the queue.
 *
 * @return the plane
 */
public Plane removePlane() {
    return queue.dequeue();
}

/**
 * Gets the queue.
 *
 * @return the queue
 */
public QueueCRAB<Plane> getQueue() {
    return queue;
}

/**
 * Sets the queue.
 *
 * @param queue the new queue
 */
public void setQueue(QueueCRAB<Plane> queue) {
    this.queue = queue;
}

/**
 * Gets the runway name.
 *
 * @return the runway name
 */
public String getRunwayName() {
    return runwayName;
}

/**
 * Sets the runway name.
 *
 * @param runwayName the new runway name
 */
public void setRunwayName(String runwayName) {
    this.runwayName = runwayName;
}

/**
 * Gets the plane.
 *
 * @return the plane
 */
public Plane getPlane() {
    return queue.peek();
}

/**
 * adds all the plane flight number and destination into a string and return i
t.
 *
 * @return the string
 */
public String toString() {
    StringBuilder string = new StringBuilder();
    for(int i =0; i<queue.getSize(); i++) {
        if(queue.peek() != null) {
            Plane plane = queue.dequeue();
            string.append("\nFlight " + plane.getFlightNumber() + " to " + pla
ne.getDestination());
            queue.enqueue(plane);
        }
    }
    return string.toString();
}

/**
 * Gets the size.
 *
 * @return the size
 */
public int getSize() {
    return queue.numItems;
}
public boolean isEmpty() {
    return queue.isEmpty();
}
}


// TODO: Auto-generated Javadoc
/**
 * The Class QueueCRAB.
 *
 * @param <T> the generic type
```

```
 */
/*
 * Purpose: Data Structure and Algorithms project
 * Status: Complete and thoroughly tested
 * Last update: 12/5/19
 * Submitted:  12/5/19
 * Comment: test suite and sample run attached
 * @author: Jackie Zheng/Anthony
 * @version: 2019.10.10
 */
public class QueueCRAB<T> implements QueueInterface<T> {

    /** The number of items. */
    protected int numItems;

    /** The front. */
    protected int front;

    /** The back. */
    protected int back;

    /** The items array. */
    protected T []items;

    /**
     * Create a new queue CRAB.
     */
    public QueueCRAB() {
        items = (T[])new Object[3];
        numItems = 0;
        front = 0;
        back = 0;
    }

    /**
     * Checks if is empty.
     *
     * @return true, if is empty
     */
    public boolean isEmpty() {
        return  numItems == 0;
    }

    /**
     * Enqueue or add item into the queue.
     *
     * @param newItem the new item
     */
    public void enqueue (T newItem) {
        if(numItems==items.length) {
            resize();
        }
        items[back] = newItem;
        back = (back+1)%items.length;
        numItems++;
    }

    /**
     * Dequeue or remove item from the queue.
     *
     * @return the t
     * @throws QueueException the queue exception
     */
    public T dequeue() throws QueueException {
        T curr;
        if(items[front] != null) {
            curr = items[front];
            items[front] = null;
            front = (front+1)%items.length;
            numItems--;
            return curr;
        }
        else {
            throw new QueueException("Exception! Queue is empty");
        }
    }

    /**
     * Dequeue all or remove all items in the queue.
     */
    public void dequeueAll() {
        items = (T[])new Object[3];
        front = 0;
        back = 0;
        numItems = 0;
    }

    /**
     * Peek or show the item in the top.
     *
     * @return the generic type
     * @throws QueueException the queue exception
     */
    public T peek() throws QueueException {
        if(numItems != 0) {
            return items[front];
        }
        else {
            throw new QueueException("Exception! Queue is empty");
        }
    }

    /**
     * adds all the items into the string and return that string.
     *
     * @return the string
     */
    public String toString() {
        StringBuilder string = new StringBuilder();
        int modFront = front;
        for(int i = 0; i<items.length; i++) {
            modFront = (modFront+1)%items.length;
            if(items[modFront] != null) {
                string.append(items[modFront] + " \n");
            }
        }
        return string.toString();
    }

    /**
     * Resize the array.
     */
    protected void resize() {
        T []newArray = (T[]) new Object[items.length*2];
```

```
            int modFront = front;
            for(int i = 0; i<items.length; i++) {
                newArray[i]=items[modFront];
                modFront = (modFront+1)%items.length;
            }
            items = newArray;
            front = 0;
            back = numItems;
        }

        /**
         * Gets the size.
         *
         * @return the size
         */
        public int getSize() {

            return numItems;
        }


}

// TODO: Auto-generated Javadoc
/*
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and thoroughly tested
 * Last update: 12/5/19
 * Submitted:  12/5/19
 * Comment: test suite and sample run attached
 * @author: Jackie Zheng/Anthony
 * @version: 2019.12.19
 */

/**
 * The Class ListArrayBasedPlus.
 *
 * @param <T> the generic type
 */
public class ListArrayBasedPlus<T> extends ListArrayBased<T> {

    /**
     * Resize the array list.
     */
    private void resize() {
        if(items.length == numItems) { //Checks if numItems and item length is equ
al
            T []newArray = (T[])new Object[(int) Math.round(items.length * 1.5)];
            for(int i = 0; i<items.length; i++) { //adds the old values to the new
 one
                newArray[i] = items[i];
            }
            items = newArray; //set the old array to the new one.
        }
    }

    /**
     * Adds the item into the array using index and if array is full, resize.
     *
     * @param index the index
     * @param item the item
     */
    @Override
    public void add(int index, T item) {
        if(numItems == items.length) {
            resize();
        }
        super.add(index, item);
    }

    /**
     * returns a string which items are all added into a single string.
     *
     * @return the string
     */
    public String toString() {
        StringBuilder string = new StringBuilder();
        for(int i = 0; i<numItems; i++) { //Runs through the array
            string.append(items[i] + " "); //Add the values to the stringbuilder.
        }
        return string.toString(); //return the string.
    }

    /**
     * Flips the items in the array.
     */
    public void reverse() {
        T[] reversed = (T[])new Object[items.length]; //new temporary array.
        for(int i = items.length-1; i>=0; i--) { //adds old to new array
            reversed[(items.length-1)-i] = items[i];
        }
        items = reversed; //set the reversed array.
    }
}

// TODO: Auto-generated Javadoc
// ****************************************************
// Array-based implementation of the ADT list.
/**
 * The Class ListArrayBased.
 *
 * @param <T> the generic type
 */
// ****************************************************
public class ListArrayBased<T> implements ListInterface<T>
{

    /** The Constant MAX_LIST. */
    private static final int MAX_LIST = 3;

    /** The array of list items. */
    protected T []items;  // an array of list items

    /** The number of items in list. */
    protected int numItems;  // number of items in list

    /**
     * Creates a new list array based.
     */
    public ListArrayBased()
    {
        items = (T[])new Object[MAX_LIST];
        numItems = 0;
    }  // end default constructor
```

```
    /**
     * Checks if is empty.
     *
     * @return true, if is empty
     */
    public boolean isEmpty()
    {
        return (numItems == 0);
    } // end isEmpty

    /**
     * return the size.
     *
     * @return the int
     */
    public int size()
    {
        return numItems;
    }  // end size

    /**
     * Removes all items and create a new empty array.
     */
    public void removeAll()
    {
        // Creates a new array; marks old array for
        // garbage collection.
        items = (T[])new Object[MAX_LIST];
        numItems = 0;
    } // end removeAll

    /**
     * Adds the new item into the array with the index.
     *
     * @param index the index
     * @param item the item
     * @throws ListIndexOutOfBoundsException the list index out of bounds exceptio
n
     */
    public void add(int index, T item)
    throws  ListIndexOutOfBoundsException
    {
        if (numItems == items.length)
        {
            throw new ListException("ListException on add");
        }  // end if
        if (index >= 0 && index <= numItems)
        {
            // make room for new element by shifting all items at
            // positions >= index toward the end of the
            // list (no shift if index == numItems+1)
            for (int pos = numItems-1; pos >= index; pos--)  //textbook code modif
ied to eliminate logic error causing ArrayIndexOutOfBoundsException
            {
                items[pos+1] = items[pos];
            } // end for
            // insert new item
            items[index] = item;
            numItems++;
        }
        else
        {
            // index out of range
            throw new ListIndexOutOfBoundsException(
                "ListIndexOutOfBoundsException on add");
        }  // end if
    } //end add

    /**
     * Gets the item in the list based off of the index.
     *
     * @param index the index
     * @return the generic type
     * @throws ListIndexOutOfBoundsException the list index out of bounds exceptio
n
     */
    public T get(int index)
    throws ListIndexOutOfBoundsException
    {
        if (index >= 0 && index < numItems)
        {
            return items[index];
        }
        else
        {
            // index out of range
            throw new ListIndexOutOfBoundsException(
                "ListIndexOutOfBoundsException on get");
        }  // end if
    } // end get

    /**
     * Removes the item in the index.
     *
     * @param index the index
     * @throws ListIndexOutOfBoundsException the list index out of bounds exceptio
n
     */
    public void remove(int index)
    throws ListIndexOutOfBoundsException
    {
        if (index >= 0 && index < numItems)
        {
            // delete item by shifting all items at
            // positions > index toward the beginning of the list
            // (no shift if index == size)
            for (int pos = index+1; pos < numItems; pos++) //textbook code modifie
d to eliminate logic error causing ArrayIndexOutOfBoundsException

            {
                items[pos-1] = items[pos];
            }  // end for
            numItems--;
            items[numItems] = null;
        }
        else
        {
            // index out of range
            throw new ListIndexOutOfBoundsException(
                "ListIndexOutOfBoundsException on remove");
        }  // end if
    } //end remove
}Welcome to the Airport program!
```

```
Enter number of runways: 3
Enter the name of runway number 1:North
Enter the name of runway number 2:South
Enter the name of runway number 3:East
Select from the following menu:
  0. Exit program.
  1. Plane enters the system.
  2. Plane takes off.
  3. Plane is allowed to re-enter a runway.
  4. Runway opens.
  5. Runway closes.
  6. Display info about planes waiting to take off.
  7. Display info about planes waiting to be allowed to re-enter a runway.
  8. Display number of planes who have taken off.

Make your menu selection now: 6
No planes are waiting for takeoff on runway North!
No planes are waiting for takeoff on runway South!
No planes are waiting for takeoff on runway East!

Make your menu selection now: 1
Enter flight number: USA12
Enter destination: Tampa
Enter runway: North
Flight USA12 is now waiting for takeoff on runway North.

Make your menu selection now: 6
These planes are waiting for takeoff on runway North :
Flight USA12 to Tampa
No planes are waiting for takeoff on runway South!
No planes are waiting for takeoff on runway East!

Make your menu selection now: 1
Enter flight number: USA55
Enter destination: France
Enter runway: South
Flight USA55 is now waiting for takeoff on runway South.

Make your menu selection now: 1
Enter flight number: United123
Enter destination: NewYork
Enter runway: East
Flight United123 is now waiting for takeoff on runway East.

Make your menu selection now: 1
Enter flight number: AirFrance87
Enter destination: LasVegas
Enter runway: North
Flight AirFrance87 is now waiting for takeoff on runway North.

Make your menu selection now: 1
Enter flight number: United123
Enter destination: Cleveland
Enter runway: South
Flight United123 is now waiting for takeoff on runway South.

Make your menu selection now: 1
Enter flight number: Delta5
Enter destination: Paris
Enter runway: South
Flight Delta5 is now waiting for takeoff on runway South.

Make your menu selection now: 6
These planes are waiting for takeoff on runway North :
Flight USA12 to Tampa
Flight AirFrance87 to LasVegas
These planes are waiting for takeoff on runway South :
Flight USA55 to France
Flight United123 to Cleveland
Flight Delta5 to Paris
These planes are waiting for takeoff on runway East :
Flight United123 to NewYork

Make your menu selection now: 2
Is flightUSA12cleared for takeoff(Y/N): N
Flight USA12 is now waiting to be allowed to re-enter a runway

Make your menu selection now: 7
These planes are waiting to be cleared to re-enter a runway:
Flight USA12 to Tampa

Make your menu selection now: 2
Is flightUSA55cleared for takeoff(Y/N): N
Flight USA55 is now waiting to be allowed to re-enter a runway

Make your menu selection now: 2
Is flightUnited123cleared for takeoff(Y/N): Y
Flight United123 has now taken off from runway East

Make your menu selection now: 6
These planes are waiting for takeoff on runway North :
Flight AirFrance87 to LasVegas
These planes are waiting for takeoff on runway South :
Flight United123 to Cleveland
Flight Delta5 to Paris
No planes are waiting for takeoff on runway East!

Make your menu selection now: 4
Enter the name of the new runway : SouthEast
Runway SouthEast has opened.

Make your menu selection now: 1
Enter flight number: Continental21
Enter destination: Tokyo
Enter runway: SouthEast
Flight Continental21 is now waiting for takeoff on runway SouthEast.

Make your menu selection now: 6
These planes are waiting for takeoff on runway North :
Flight AirFrance87 to LasVegas
These planes are waiting for takeoff on runway South :
Flight United123 to Cleveland
Flight Delta5 to Paris
No planes are waiting for takeoff on runway East!
These planes are waiting for takeoff on runway SouthEast :
Flight Continental21 to Tokyo

Make your menu selection now: 8
1 planes have taken off from the airport.

Make your menu selection now: 5
Enter runway: North
Enter new runway for plane AirFrance87:SouthEast
Flight AirFrance87  is now waiting for takeoff on runway SouthEast
```

```
Enter new runway for plane USA12:East
Flight USA12 has changed runway to East
Runway North has been closed.

Make your menu selection now: 6
These planes are waiting for takeoff on runway South :
Flight United123 to Cleveland
Flight Delta5 to Paris
No planes are waiting for takeoff on runway East!
These planes are waiting for takeoff on runway SouthEast :
Flight Continental21 to Tokyo
Flight AirFrance87 to LasVegas

Make your menu selection now: 7
These planes are waiting to be cleared to re-enter a runway:
Flight USA55 to France

Make your menu selection now: 3
Enter flight number: USA55
Flight USA55 is now waiting for takeoff on runway South

Make your menu selection now: 0
The Airport is closing :Bye Bye....
```