```
::::::::::::::
Driver.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */

import java.io.*;

public class Driver

{
    /** BufferedReader to intake our keyboard input */
    static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.
in));

    public static void main(String args[]) throws IOException
    {
        /** boolean for use of the menu - determines if we are done, or not */
        Boolean quit = false;
        /** AirTrafficControl holds most of our control methods */
        AirTrafficControl atc = new AirTrafficControl();
        /** String flightNumber used for creating new planes */
        String flightNumber;
        /** String destination used for creating new planes */
        String destination;
        /** String runway used for creating new planes */
        String runway;
        /** Boolean determining if a plane is successfully added */
        boolean succeful;
        /** integer holding our int from setting up initial runways */
        int intHolder;
        /** String holding our initial runway names */
        String stringHolder;
        /** Temporary plane used for creating new planes */
        Plane planeHolder;


        System.out.println("Welcome to the Airport program!");
        //Initial runway setup - Nico Kaegi
        System.out.print("Enter number of runways: ");

        intHolder = Integer.parseInt(stdin.readLine());
        System.out.println(intHolder);

        for (int pos = 1; pos <= intHolder; pos++)
        {

            System.out.print(pos + "Enter the name of runway number " + pos + ": "
);

            stringHolder = stdin.readLine();

            System.out.println(stringHolder);

            if (atc.addRunWay(stringHolder))
            {
                //System.out.println("success");

            } else
            {
                System.out.println("Error, this name is not unique.");
                intHolder++;
            }
        }

        // Initial menu list
        System.out.println("Select from the following menu: ");
        System.out.println("\t0. End the program");
        System.out.println("\t1. Plane enters the system.");
        System.out.println("\t2. Plane takes off.");
        System.out.println("\t3. Plane is allowed to re-enter a runway.");
        System.out.println("\t4. Runway opens.");
        System.out.println("\t5. Runway closes.");
        System.out.println("\t6. Display info about planes waiting to take off.");
        System.out.println("\t7. Display info about planes waiting to be allowed t
o re-enter a runway.");
        System.out.println("\t8. Display number of planes who have taken off.");

        while (!quit)
        {
            System.out.print("\n\tEnter your selection now : ");
            int menu = Integer.parseInt(stdin.readLine());
            System.out.println(menu);

            switch (menu)
            {
            case  0:
                quit = true;
                break;
            case 1:
                //Case 1:  Enter new plane into system
                // Nico Kaegi
                System.out.print("Enter flight number: ");
                String newFlight = stdin.readLine();
                System.out.println(newFlight);
                System.out.print("Enter destination: ");
                String newDest = stdin.readLine();
                System.out.println(newDest);
                System.out.print("Enter runway: ");
                String newRunway = stdin.readLine();
                System.out.println(newRunway);

                while( atc.findRunway(newRunway) == null) {
                    System.out.println("No such runway!");
                    System.out.print("Enter runway: ");
                    newRunway = stdin.readLine();
                    System.out.println(newRunway);

                }

                atc.enterAirPlane(new Plane(newFlight, newDest, newRunway));
                System.out.println("Flight " + newFlight + " is now waiting for ta
keoff on runway " + newRunway + ".");
```

```
                break;
            case 2:
                //Case 2: Plane takes off in a round robin style
                //Nico Kaegi
                planeHolder = atc.currentTakeOfPlane();

                if(planeHolder != null) {

                    System.out.println(
                        "Is Flight " + planeHolder.getFlightNumber() + "  cleared
for take off (Y/N): ");

                    System.out.println();//remove this later.

                    stringHolder = stdin.readLine().toUpperCase();
                    System.out.println(stringHolder);

                    if (stringHolder.equals("Y"))
                    {

                        atc.takeOff(true);

                    } else
                    {

                        atc.takeOff(false);

                    }
                }
                else {

                    //incase there are no planes at a runways (the minus on is int
entional since postion get updated even when currentTakeOfPlane() returns null)
                    //System.out.println("no planes at runway" + ( atc.getPosition
() - 1 ) );
                    System.out.println("No plane on any runway!");
                }

                break;
            case 3:
                reEnterPlane(atc);
                break;
            case 4:
                openRunway(atc);
                break;
            case 5:
                closeRunway(atc);
                break;
            case 6:
                atc.printRunways();
                break;
            case 7:
                atc.printClearance();
                break;
            case 8:
                atc.printCount();
                break;
            default:
                System.out.println("\n\tInvalid input - Make your menu selection n
ow : ");
            }// end switch

            if (quit)
            {
                System.out.println("The Airport is closing : Bye Bye....");
                System.exit(0);
            }
        }// end while

    }// End main
    /**
     * Calls the ATC to search through the clearance list, find the relevant plane
, and put it back on its runway if it is now clear for takeoff
     * @author Theresa Morris
     * @param atc
     *            the AirTrafficControl object
     */

    private static void reEnterPlane(AirTrafficControl atc) throws IOException
    {
        boolean cleared = false;

        if(!atc.getClearance().isEmpty())
        {
            while(!cleared)
            {
                System.out.print("Enter flight number : ");
                String flight = stdin.readLine();
                System.out.println(flight);
                Plane searchPlane = new Plane(flight, null, null);

                if(atc.getClearance().search(searchPlane) >= 0)
                {
                    try
                    {
                        cleared = atc.reEnterRunway(searchPlane);
                    }
                    catch (Exception e)
                    {
                        System.out.println("Unable to re-queue plane to runway!");
                    }
                }
            }
        }
        else
        {
            System.out.println("There are no planes waiting for clearance!");
        }
    }// end reEnterPlane
    /**
     * Calls the ATC to add a new runway
     * @author Theresa Morris
     * @param atc
     *            the AirTrafficControl object
     */
    private static void openRunway(AirTrafficControl atc) throws IOException
    {
        System.out.print("Enter the name of the new runway : ");
        String newRunway = stdin.readLine();
        System.out.println(newRunway);

        while(!(atc.addRunWay(newRunway)))
        {
            System.out.println("Runway " + newRunway + " already exists please cho
```

```
ose another name. ");

            System.out.print("Enter the name of the new runway : ");
            newRunway = stdin.readLine();
            System.out.println(newRunway);

        }
        System.out.println("Runway " + newRunway + " has opened.");
    }// end openRunway

    /**
     * Calls the ATC to first find the closing runway, and loop through it, puttin
g all of its planes on new runways.  It then flattens the clearance list (temporar
ily), searches through it to find any planes from the closing runway, and reassign
 them to new runways, without re-enqueueing them.
     * @author Theresa Morris
     * @param atc
     *            the AirTrafficControl object
     */
    private static void closeRunway(AirTrafficControl atc) throws IOException
    {

        System.out.print("Enter runway: ");
        String oldRunway = stdin.readLine();
        System.out.println(oldRunway);
        while(atc.findRunway(oldRunway) == null)
        {
            System.out.println("No such runway!");
            System.out.print("Enter runway: ");
            oldRunway = stdin.readLine();
            System.out.println(oldRunway);
        }
        atc.runwayLoop(oldRunway, stdin);
        atc.clearanceLoop(oldRunway, stdin);

        System.out.println("Runway " + oldRunway + " has been closed.");
    }// end closeRunway
}
::::::::::::::
AirTrafficControl.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */
import java.io.*;

public class AirTrafficControl
{
    /** List of current runways */
    private ListArrayBasedPlus<Runway> runways;

    /** AscendinglyOrderedList that holds all planes waiting for clearance to laun
ch */
    private AscendinglyOrderedList<Plane<?>, String> clearance = new AscendinglyOr
deredList<Plane<?>, String>();

    /**Integer counting how many planes have taken off */
    private int count = 0;

    /** Integer keeping our position when planes take off from runways that they m
ay be incremented in a round robin fashion*/
    private int position = 0;

    /** Int holding the total amount of runways we have, so we may always find the
 end index. */
    private int totalRunways = 0;

    public AirTrafficControl()
    {
        runways = new ListArrayBasedPlus<Runway>();
        //clearance = new AscendinglyOrderedList<Plane<?>, String>();
    }

    /**
     * Method to return our list of runways
     * @return the runways
     */

    public ListArrayBasedPlus<Runway> getRunways()
    {
        return runways;
    }

    /**
     * Method to take in a string representation of a runway name, and search the
runway list to locate it.
     * @author Nico Kaegi
     * @param runwayName
     *          The runway we are searching for
     * @return the located runway
     */
    public Runway findRunway(String runwayName) {

        for(int pos = 0; pos < runways.size(); pos++) {

            if(runways.get(pos)!= null)
            {
                if(runways.get(pos).getName().equals(runwayName)) {

                    return runways.get(pos);
                }
            }
        }

        return null;

    }

    /**
     * Method to enter a plane that has been created into an already existing runw
ay.
     * @author Nico Kaegi
     * @param newPlane
     *          the plane we are entering into a runway
     * @return whether the plane successfully entered or not.
```

```
     */
    public boolean enterAirPlane(Plane<?> newPlane) {

        //it's boolean because im to lazy to add trycatch blocks
        //create a new excpetion; plus this provideds a easy way to exit the metho
d;

        Runway temp = findRunway(newPlane.getRunway());

        if(temp != null) {
            temp.enqueueToRunway(newPlane);
            return true;
        }
        else {
            return false;
        }
    }
    /**
     * Method to add a new runway at the end of the runway list, then return wheth
er it was successful.
     * @author Nico Kaegi
     * @param name
     *  A string representation of a runway name
     * @return whether the runway was added or not
     */
    public boolean addRunWay(String name) {

        if(findRunway(name) == null) {
            runways.add(totalRunways, new Runway(name));
            totalRunways++;
            return true;
        }
        else {
            return false;
        }
    }

    /**
     * MEthod that looks at what runway we need to take off from.
     * @return Plane
     *          The plane that has taken off
     */
    public Plane<?> currentTakeOfPlane() {

        for(int n = 0; n < runways.size(); n++ ) {

            if(runways.get(position).isEmpty()) {

                position = (position + 1) % runways.size();
            }
            else {
                return runways.get(position).peekRunway();
            }
        }

        return null;
    }

    /**
     * Method that accepts whether a plane has clearance to launch or not, and the
n either launches it, or puts it in the clearance list.
     * @author Nico Kaegi
     * @param clearnce
     *          Whether the plane has clearance to launch
     */
    public void takeOff(boolean clearnce) {

        if(clearnce) {
            Plane tempPlane = runways.get(position).peekRunway();
            runways.get(position).dequeueFromRunway();
            System.out.println("Flight " + tempPlane.getFlightNumber() + " has now
 taken off from runway " + tempPlane.getRunway());
            position = (position + 1) % (runways.size());
            count++;
        } else {
            Plane tempPlane = runways.get(position).peekRunway();
            clearance.add(runways.get(position).dequeueFromRunway());
            System.out.println("Flight " + tempPlane.getFlightNumber() + " is now
waiting to be allowed to re-enter a runway.");
            position = (position + 1) % (runways.size());

        }
    }

    /**
     * MEthod to return the list of planes waiting for clearance.
     * @return the clearance list
     */
    public AscendinglyOrderedList<Plane<?>, String> getClearance()
    {
        return clearance;
    }
    /**
     * method to return the count of planes launched
     * @return the count
     */
    public int getCount()
    {
        return count;
    }
    /**
     * method to return the current position in the runway list
     * @return the position
     */
    public int getPosition()
    {
        return position;
    }
    /**
     * method to set the runway list
     * @param runways
     *          a list of runways
     */
    public void setRunways(ListArrayBasedPlus<Runway> runways)
    {
        this.runways = runways;
    }
    /**
     * method to set the clearance list
     * @param clearance
     *          a clearance list.
     */
    public void setClearance(AscendinglyOrderedList<Plane<?>, String> clearance)
    {
```

```
        this.clearance = clearance;
    }
    /**
     * method to set the current count.
     * @param count
     *          the current count of planes launched
     */
    public void setCount(int count)
    {
        this.count = count;
    }
    /**
     * method to set the current position in the runway list.
     * @param position
     *          the current position in the runway list
     */
    public void setPosition(int position)
    {
        this.position = position;
    }

    /**
     * Method to have a plane from the clearance list re-enter a runway
     *
     * This is done by creating a temporary plane, retrieving the plane from the c
learance list by searching by it's flight number, re-enqueuing it if it exists at
all, then deleting it from the clearance list.
     * @author Theresa Morris
     *
     * @param flight
     *          The flight number of the plane we are looking for.
     * @return whether the flight was successfully added
     */
    public boolean reEnterRunway(Plane<?> flight)
    {
        int tempIndex = clearance.search(flight);

        if(tempIndex < 0)
        {
            System.out.println("Flight " + flight.getFlightNumber() + " is not wai
ting for clearance.");
            return false;

        }
        else
        {
            Plane<?> tempPlane = clearance.get(tempIndex);
            clearance.remove(tempIndex);

            findRunway(tempPlane.getRunway()).enqueueToRunway(tempPlane);
            System.out.println("Flight " + tempPlane.getFlightNumber() + " is now
waiting for takeoff on runway " + tempPlane.getRunway());
            return true;
        }

    }

    /**
     * Method to loop through the runway being closed to reassign all planes to ne
w runways.
     * @author Theresa Morris
     * @param oldRunway
     *          The runway being closed down
     * @param stdin
     *          Our standard input, as we only need one buffered reader
     * @throws IOException
     */
    public void runwayLoop(String oldRunway, BufferedReader stdin) throws IOExcept
ion
    {
        if(findRunway(oldRunway) != null)
        {
            while(!findRunway(oldRunway).isEmpty())
            {
                Plane tempPlane = findRunway(oldRunway).peekRunway();

                System.out.print("Enter new runway for plane " + tempPlane.getFlig
htNumber() + " : ");
                String newRunway = stdin.readLine();
                System.out.println(newRunway);

                if(findRunway(newRunway) != null)
                {
                    if(newRunway.equals(oldRunway))
                    {
                        System.out.println("This is the runway that is closing!");
                    }
                    else
                    {
                        tempPlane.setRunway(newRunway);
                        findRunway(newRunway).enqueueToRunway(tempPlane);
                        System.out.println("Flight " +tempPlane.getFlightNumber()
+ " is now waiting for takeoff on runway " + tempPlane.getRunway());
                        findRunway(oldRunway).dequeueFromRunway();
                    }
                }
                else
                {
                    System.out.println("No such runway!");
                }
            }
        }
    }

    /**
     * Method to look through the clearance list for any runway matching the closi
ng runway, then reassign any of those planes to new runways, without re-enqueueing
 them for launch.
     * @author Theresa Morris
     * @param oldRunway
     *          The runway being closed
     * @param stdin
     *          The bufferedreader handling input
     * @throws IOException
     */
    public void clearanceLoop(String oldRunway, BufferedReader stdin) throws IOExc
eption
    {
        for(int i = 0; i < clearance.size(); i++)
        {
            if(clearance.get(i).getRunway().equals(oldRunway))
            {
                Plane tempPlane = clearance.get(i);
                System.out.print("Enter new runway for plane " + tempPlane.getFlig
```

```
htNumber() + " : ");
                String newRunway = stdin.readLine();
                System.out.println(newRunway);

                if(findRunway(newRunway) != null)
                {

                    if(newRunway.equals(oldRunway))
                    {
                        System.out.println("This is the runway that is closing!");
                    }
                    else
                    {
                        clearance.get(i).setRunway(newRunway);

                        System.out.println("Flight " + tempPlane.getFlightNumber()
 + " is now waiting for takeoff on Runway " + tempPlane.getRunway());
                    }
                }
                else
                {
                    System.out.println("No such runway!");
                }
            }
        }
    }
    /**
     * @author Theresa Morris
     * A method to print out the amount of planes who have taken off.
     */
    public void printCount()
    {
        System.out.println(count + " planes have taken off from the airport");
    }

    /**
     * @author Theresa Morris
     * A method to print everything waiting on the clearance list for takeoff.
     */
    public void printClearance()
    {
        if(!clearance.isEmpty())
        {
            System.out.println("These planes are waiting to be cleared to re-enter
 a runway:");
            System.out.println(clearance.toString());
        }
        else
        {
            System.out.println("No planes are waiting to be cleared to re-enter a
runway!");
        }

    }

    /**
     * @author Theresa Morris
     * A method to print everything currently waiting on all of the runways.
     */
    public void printRunways()
    {
        for(int i = 0; i < runways.size(); i++)
        {
            if(runways.get(i).isEmpty())
            {
                System.out.println("No planes are waiting for takeoff on runway "
+ runways.get(i).getName() + "!");
            }
            else
            {
                System.out.println("These planes are waiting for takeoff on runway
 " + runways.get(i).getName());
                System.out.print(runways.get(i).toString());
            }
        }
    }

    /**
     * @author Theresa Morris
     * A method to delete a runway from the list
     * @param runway the runway we are looking to delete
     */

    public void deleteRunway(String runway)
    {
        if (findRunway(runway) != null)
        {
            runways.remove(findRunwayIndex(runway));
        }
    }

    /**
     * @author Theresa Morris
     *A method to find the INDEX of the runway we are looking to delete.
     *@param runway the runway we are looking to delete
     *@return the index our runway is at in the runway list
     */
    public Integer findRunwayIndex(String runway)
    {
        for(int pos = 0; pos < runways.size(); pos++) {

            if(runways.get(pos)!=null)
            {
                if(runways.get(pos).getName().equals(runway)) {
                    return new Integer(pos);
                }
            }
        }
        return null;
    }

}
::::::::::::::
Plane.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
```

```
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */


public class Plane<KT> extends KeyedItem<String>
{
    /** A string to hold our flight number (this is our comparable field) */
    private String flightNumber;
    /** A string to hold our destination */
    private String destination;
    /** a string representation of the runway this belongs on. for use in reenteri
ng runways */
    private String runway;

    public Plane(String flight, String dest, String run)
    {
        super(flight);
        flightNumber = flight;
        destination = dest;
        runway = run;
    }

    /**
     * A method to return the flight number
     * @return the flightNumber
     */
    public String getFlightNumber()
    {
        return flightNumber;
    }

    /**
     * a method to return the destination
     *
     * @return the destination
     */
    public String getDestination()
    {
        return destination;
    }

    /**
     * A method to return the name of the runway
     * @return the runway
     */
    public String getRunway()
    {
        return runway;
    }

    /**
     * a method to set the flight number
     * @param flightNumber the flightNumber to set
     */
    public void setFlightNumber(String flightNumber)
    {
        this.flightNumber = flightNumber;
    }

    /**
     * a method to set the desination
     * @param destination the destination to set
     */
    public void setDestination(String destination)
    {
        this.destination = destination;
    }

    /**
     * a method to set the runway
     * @param runway the runway to set
     */
    public void setRunway(String runway)
    {
        this.runway = runway;
    }

    @Override
    /**
     * toString takes the flight number and destination, and concatinates them int
o a single string.
     */
    public String toString()
    {
        return "Flight " + flightNumber + " to " + destination + ".";
    }



}
::::::::::::::
Runway.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */


public class Runway
{
    /**A string holding the runway name */
    String name;
    /**A queue holding all of the planes waiting for takeoff from this runway */
    QueueSLS<Plane> planes;

    public Runway(String name) {

        this.name = name;
        planes = new QueueSLS<Plane>();
    }
    /**
```

```
     * return the runway's name
     * @return the name
     */
    public String getName()
    {
        return name;
    }
    /**
     * return the runway's queue of planes
     * @return the planes
     */
    public QueueSLS<Plane> getPlanes()
    {
        return planes;
    }
    /**
     * set the runway's name
     * @param name the name to set
     */
    public void setName(String name)
    {
        this.name = name;
    }
    /**
     * set the runway's queue of planes
     * @param planes the planes to set
     */
    public void setPlanes(QueueSLS<Plane> planes)
    {
        this.planes = planes;
    }
    @Override
    /**
     *  if we have planes at all, calls the toString method on the queue, which ca
lls the toString method on each plane
     */
    public String toString()
    {
        if(!planes.isEmpty())
        {
            return (planes.toString());
        }
        else
        {
            return null;
        }
    }

    /**
     * A method to enqueue a plane to the planelist in the runway
     * @param plane the plane to enqueue
     */
    public void enqueueToRunway(Plane plane)
    {
        planes.enqueue(plane);
    }
    /**
     * A method to peek at the plane at the end of the queue
     * @return the plane at the end of the queue
     */
    public Plane peekRunway()
    {
        return planes.peek();

    }
    /**
     * A method to dequeue a plane from the runway
     * @return the plane at the end of the queue
     */
    public Plane dequeueFromRunway()
    {
        return planes.dequeue();
    }

    /**
     * A method to tell us if the runway is empty
     * @return true if the runway is empty, false otherwise.
     */
    public boolean isEmpty()
    {
        return planes.isEmpty();
    }

}
::::::::::::::
AscendinglyOrderedList.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */

public class AscendinglyOrderedList<T extends KeyedItem<KT>, KT extends Comparable
    <? super KT>> implements AscendinglyOrderedListInterface<T, KT> {
    /**Initial size of the Ascendingly Ordered List */
    private static final int MAX_LIST = 3;
    /** array of generic items */
    protected T[] items;
    /** how many items are stored (as opposed to repeatedly counting) */
    protected int numItems;
    /** Whether we succussfully added something*/
    boolean success = false;

    public AscendinglyOrderedList()
    {
        items = (T[]) new KeyedItem[MAX_LIST];
        numItems = 0;
    }
    /**
     * returns whether the array is empty
     * @return whether the array is empty
     */
    public boolean isEmpty() {
        return (numItems == 0);
```

```
    }

    /**
     * returns number of items in the array
     * @return number of items in the array
     */
    public int size() {
        return numItems;
    }

    /**
     * takes a generic item, searches for where it should be located (in this case
 lexicographically), and inserts it in place.
     * @param item the item we are looking to add
     */
    public void add(T item) throws ListIndexOutOfBoundsException {
        if (numItems >= items.length)
        {
            resize();
        }
        int index = search(item);

        if(!success)
        {
            if(index < 0)
            {
                //convert negative index back to a positive.
                index = index -(index*2);
            }
            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 m
odified to eliminate logic error causing ArrayIndexOutOfBoundsException
                {
                    items[pos+1] = items[pos];
                } // end for
                // insert new item
                items[index] = (T) item;
                numItems++;
            }
            else
            {
                // index out of range
                throw new ListIndexOutOfBoundsException(
                    "ListIndexOutOfBoundsException on add");
            }  // end if
        }
        else
        {
            System.out.println("Item already exists in list!");
        }
    }
    //end add



    /**
     * takes in the index of an item we are looking for, and returns that item.
     * @param index the index of the item we want
     * @return the item we are retrieving
     */
    public T get(int index) throws ListIndexOutOfBoundsException {
        {
            if (index >= 0 && index < numItems)
            {
                return (T) items[index];
            }
            else
            {
                // index out of range
                throw new ListIndexOutOfBoundsException(
                    "ListIndexOutOfBoundsException on get");
            }  // end if
        } // end get
    }

    /**
     * Takes in an index and deletes the relevant item, shifting everything to acc
omidate.
     * @param index the index we are removing
     */
    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

            items[--numItems] = null; //Fixes memory leak (Hopefully)

        }
        else
        {
            // index out of range
            throw new ListIndexOutOfBoundsException(
                "ListIndexOutOfBoundsException on remove");
        }  // end if
    }

    /**
     * Searches for an item in teh array.  If it finds it, it just tells us where
it was and does not allow us to insert a duplcate.
     * If the item isn't found at all, it returns a negative version of the high i
ndex (the negative indicates the item was not succssfully located)
     * @param item a generic item we are searching for
     * @return The index where we found the item, or the index where we need to in
sert it.
     */
    public int search(T item) {

        int low = 0;
        int high = numItems - 1;
        int mid = 0;
        //Special case one, we have nothing in the list...
        if (numItems == 0)
```

```
        {
            return 0;
        }
        //handling this as a special case because no matter which way I would chec
k, I would get a
        //0,0 check, which messes EVERYTHING up.
        if (numItems == 1)
        {
            if(item.getKey().compareTo((items[mid]).getKey()) > 0)
            {
                return 1;
            }
            else
            {
                return 0;
            }
        }

        while (low < high)
        {
            mid = (low + high)/2;
            if(item.getKey().compareTo((items[mid]).getKey()) > 0)
            {
                low = mid + 1;
            }
            else
            {
                high = mid;
            }
        }

        if(item.equals(items[low]))
        {
            success = true;
            return low;
        }
        else
        {
            success = false;
            if(item.getKey().compareTo((items[low]).getKey()) > 0)
            {
                return -(high + 1);
            }
            else
            {
                return - high;
            }
        }
    }
    /**
     * Empties the array by deleting it
     */
    public void clear() {
        items = (T[]) new KeyedItem[MAX_LIST];
        numItems = 0;
    }
    /**
     * When the array is full, resizes it as needed.
     */
    private void resize()
    {
        T []temp = (T[]) new KeyedItem[items.length + (items.length/2)];
        for (int i = 0; i < items.length; i++)
        {
            temp[i] = items[i];
        }
        items = temp;

    }
    /**
     * parses through all of the array and concatinates it into one string.
     * @return a string representation of the array
     */
    public String toString()
    {
        StringBuilder buildList = new StringBuilder();

        for (int i = 0; i < numItems; i++)
        {
            buildList.append(items[i] + "\n");
        }

        return buildList.toString();
    }

}
::::::::::::::
KeyedItem.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */
public abstract class KeyedItem<KT extends
    Comparable<? super KT>> {
    private KT searchKey;

    public KeyedItem(KT key) {
        searchKey = key;
    }  // end constructor
    /**
     * @return our comparable key
     */
    public KT getKey() {
        return searchKey;
    } // end getKey
} // end KeyedItem
::::::::::::::
ListArrayBased.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
```

```
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */
// ******************************************************
// Array-based implementation of the ADT list.
// ******************************************************
public class ListArrayBased<T> implements ListInterface<T>
{
    /** the initial size of the array */
    private static final int MAX_LIST = 3;
    /** our list of items */
    protected T []items;
    /** how many items we are holding */
    protected int numItems;

    public ListArrayBased()
    {
        items = (T[]) new Object[MAX_LIST];
        numItems = 0;
    }  // end default constructor
    /**
     * @return true if the array is empty
     */
    public boolean isEmpty()
    {
        return (numItems == 0);
    } // end isEmpty
    /**
     * @return how many items are in the array
     */
    public int size()
    {
        return numItems;
    }  // end size
    /**
     * Creates a new array, deleting the old one.
     */
    public void removeAll()
    {
        // Creates a new array; marks old array for
        // garbage collection.
        items = (T[])new Object[MAX_LIST];
        numItems = 0;
    } // end removeAll
    /**
     * takes in an index and an item, and adds them to the array.
     * @param index the index we are adding at
     * @param item the item we are adding
     */
    public void add(int index, T item)
    throws  ListIndexOutOfBoundsException
    {
        if (numItems == items.length) //Fixes Implementation Error and programming
 style
        {
            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
    /**
     * takes in an index and returns the item stored there.
     * @param index the index of the item we are looking for
     * @return the item we are searching for
     */
    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
    /**
     * takes in an index and removes the item at that index
     * @param index the index to be removed
     */
    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

            items[--numItems] = null; //Fixes memory leak (Hopefully)
```

```
        }
        else
        {
            // index out of range
            throw new ListIndexOutOfBoundsException(
                "ListIndexOutOfBoundsException on remove");
        }  // end if
    } //end remove
    @Override
    /**
     * Parses through the array and returns a string representation of all of the
items in it.
     */
    public String toString()
    {
        StringBuilder buildItems = new StringBuilder();

        for(int i = 0; i<numItems; i++)
        {
            buildItems.append(items[i] + "\n");
        }

        return buildItems.toString();
    }
}
::::::::::::::
ListArrayBasedPlus.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */
public class ListArrayBasedPlus<T> extends ListArrayBased<T>
{
    /**
     * Takes in an index and an item, and if our array is full, resizes before add
ing
     * @param index the index we are adding at
     * @param item the item we are adding
     */
    public void add(int index, T item) throws ListIndexOutOfBoundsException
    {
        if (numItems >= items.length)
        {
            resize();
        }
        super.add(index, item);

    }
    /**
     * If our array is full, resizes the array to half again its size.
     */
    private void resize()
    {
        T []temp = (T[]) new Object[items.length + (items.length/2)];

        for (int i = 0; i < items.length; i++)
        {
            temp[i] = items[i];
        }
        items = temp;

    }
    /**
     * Reverses the array.
     */
    public void reverse()
    {
        T []temp = (T[]) new Object[items.length];

        for (int i = 0; i < numItems; i++)
        {
            temp[(numItems-i)-1] = items[i];
        }
        items = temp;
    }
    @Override
    /**
     * Parses through the array and creates a sting representation of all of the i
tems in it.
     * @return the string repersentation of the list.
     */
    public String toString()
    {
        StringBuilder buildList = new StringBuilder();

        for (int i = 0; i < numItems; i++)
        {
            buildList.append(items[i] + "\n");
        }

        return buildList.toString();
    }
}
::::::::::::::
Node.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */
//please note that this code is different from the textbook code, because the data
 is encapsulated!

public class Node<T>
{
```

```
    /** The generic item the node holds */
    private T item;
    /** the next node in the list from the current one */
    private Node<T> next;

    public Node(T newItem)
    {
        item = newItem;
        next = null;
    } // end constructor

    public Node(T newItem, Node<T> nextNode)
    {
        item = newItem;
        next = nextNode;
    } // end constructor

    /**
     * Sets the item in the node
     * @param newItem the item to be set
     */
    public void setItem(T newItem)
    {
        item = newItem;
    } // end setItem

    /**
     * @return the item in the node
     */
    public T getItem()
    {
        return item;
    } // end getItem

    /**
     * sets the next node in the list
     * @param nextNode the next node in the list
     */
    public void setNext(Node<T> nextNode)
    {
        next = nextNode;
    } // end setNext
    /*
     * @return the next node in the list
     */
    public Node<T> getNext()
    {
        return next;
    } // end getNext
} // end class Node
::::::::::::::
QueueSLS.java
::::::::::::::
/**
 * Purpose: Data Structure and Algorithms Project
 * Status: Complete and Thoroughly Tested
 * Last update: 12/04/19
 * Submitted: 12/05/19
 * Comment: test suite and sample run attached
 *
 * @author: Theresa Morris - Section 2
 *
 * @author: Nico Kaegi - Section 2
 *
 * @version: 2019.12.04
 */


public class QueueSLS<T> implements QueueInterface<T>
{
    /** the front of the queue */
    protected Node<T> front;
    /** the back of the queue */
    protected Node<T> back;

    public QueueSLS()
    {
        front = null;
        back = null;
    }

    /**
     * @return whether the queue is empty.
     */
    public boolean isEmpty()
    {
        return front == null;
    }

    /**
     * adds an item at the back of the queue.
     * @param newItem the item we are adding
     */
    public void enqueue(T newItem) throws QueueException
    {
        if (front == null)
        {
            front = new Node<>(newItem, back);
            back = front;
        }
        else
        {
            Node<T> temp = new Node<T>(newItem);
            back.setNext(temp);
            back = temp;
        }
    }
    /**
     * Returns to us the item at the front of the queue.
     * @return the item we are dequeueing
     * @throws QueueException
     */
    public T dequeue() throws QueueException
    {
        if(front != null)
        {
            T temp = front.getItem();
            if (front == back)
            {
                front = null;
                back = null;
            }
            else
            {
```

```
                front = front.getNext();
            }
            return temp;
        }
        else
        {
            throw new QueueException("Queue Exception on dequeue!");
        }
    }
    /**
     * empties the queue by creating a new one.
     */
    public void dequeueAll()
    {
        front = null;
        back = null;
    }
    /**
     * @return the item at the front of the queue
     */
    public T peek() throws QueueException
    {
        if (front != null)
        {
            return front.getItem();
        }
        else
        {
            throw new QueueException("Queue Exception on peek!");
        }
    }
    @Override
    /**
     * parses through the queue and returns a string representation of it.
     * @return a string representation of the queue
     */
    public String toString()
    {
        StringBuilder buildList = new StringBuilder();
        Node<T> curr = front;

        while (curr != null)
        {
            buildList.append(curr.getItem().toString() + "\n ");
            curr = curr.getNext();
        }

        return buildList.toString();
    }
}
::::::::::::::
airport.output
::::::::::::::
Welcome to the Airport program!
Enter number of runways: 3
1Enter the name of runway number 1: NorthEast
2Enter the name of runway number 2: SouthWest
3Enter the name of runway number 3: West
Select from the following menu:
        0. End the 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.

        Enter your selection now : 2
No plane on any runway!

        Enter your selection now : 3
There are no planes waiting for clearance!

        Enter your selection now : 6
No planes are waiting for takeoff on runway NorthEast!
No planes are waiting for takeoff on runway SouthWest!
No planes are waiting for takeoff on runway West!

        Enter your selection now : 7
No planes are waiting to be cleared to re-enter a runway!

        Enter your selection now : 8
0 planes have taken off from the airport

        Enter your selection now : 1
Enter flight number: USAir705
Enter destination: Boston
Enter runway: NorthEast
Flight USAir705 is now waiting for takeoff on runway NorthEast.

        Enter your selection now : 1
Enter flight number: AirFrance212
Enter destination: Paris
Enter runway: NorthEast
Flight AirFrance212 is now waiting for takeoff on runway NorthEast.

        Enter your selection now : 1
Enter flight number: British909
Enter destination: London
Enter runway: NorthEast
Flight British909 is now waiting for takeoff on runway NorthEast.

        Enter your selection now : 1
Enter flight number: United954
Enter destination: Pittsburgh
Enter runway: NorthWest
No such runway!
Enter runway: West
Flight United954 is now waiting for takeoff on runway West.

        Enter your selection now : 1
Enter flight number: Delta204
Enter destination: Chicago
Enter runway: NorthEast
Flight Delta204 is now waiting for takeoff on runway NorthEast.

        Enter your selection now : 1
Enter flight number: USAir305
Enter destination: San Diego
Enter runway: West
Flight USAir305 is now waiting for takeoff on runway West.
```

```
        Enter your selection now : 1
Enter flight number: United572
Enter destination: Fort Lauderdale
Enter runway: SouthWest
Flight United572 is now waiting for takeoff on runway SouthWest.

        Enter your selection now : 3
There are no planes waiting for clearance!

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight USAir705 to Boston.
 Flight AirFrance212 to Paris.
 Flight British909 to London.
 Flight Delta204 to Chicago.
 These planes are waiting for takeoff on runway SouthWest
Flight United572 to Fort Lauderdale.
 These planes are waiting for takeoff on runway West
Flight United954 to Pittsburgh.
 Flight USAir305 to San Diego.

        Enter your selection now : 7
No planes are waiting to be cleared to re-enter a runway!

        Enter your selection now : 8
0 planes have taken off from the airport

        Enter your selection now : 2
Is Flight USAir705  cleared for take off (Y/N):

N
Flight USAir705 is now waiting to be allowed to re-enter a runway.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight AirFrance212 to Paris.
 Flight British909 to London.
 Flight Delta204 to Chicago.
 These planes are waiting for takeoff on runway SouthWest
Flight United572 to Fort Lauderdale.
 These planes are waiting for takeoff on runway West
Flight United954 to Pittsburgh.
 Flight USAir305 to San Diego.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight USAir705 to Boston.


        Enter your selection now : 8
0 planes have taken off from the airport

        Enter your selection now : 1
Enter flight number: American493
Enter destination: Seattle
Enter runway: West
Flight American493 is now waiting for takeoff on runway West.

        Enter your selection now : 2
Is Flight United572  cleared for take off (Y/N):

Y
Flight United572 has now taken off from runway SouthWest

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight AirFrance212 to Paris.
 Flight British909 to London.
 Flight Delta204 to Chicago.
 No planes are waiting for takeoff on runway SouthWest!
These planes are waiting for takeoff on runway West
Flight United954 to Pittsburgh.
 Flight USAir305 to San Diego.
 Flight American493 to Seattle.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight USAir705 to Boston.


        Enter your selection now : 8
1 planes have taken off from the airport

        Enter your selection now : 2
Is Flight United954  cleared for take off (Y/N):

N
Flight United954 is now waiting to be allowed to re-enter a runway.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight AirFrance212 to Paris.
 Flight British909 to London.
 Flight Delta204 to Chicago.
 No planes are waiting for takeoff on runway SouthWest!
These planes are waiting for takeoff on runway West
Flight USAir305 to San Diego.
 Flight American493 to Seattle.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
1 planes have taken off from the airport

        Enter your selection now : 2
Is Flight AirFrance212  cleared for take off (Y/N):

N
Flight AirFrance212 is now waiting to be allowed to re-enter a runway.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight British909 to London.
 Flight Delta204 to Chicago.
 No planes are waiting for takeoff on runway SouthWest!
These planes are waiting for takeoff on runway West
Flight USAir305 to San Diego.
 Flight American493 to Seattle.

        Enter your selection now : 7
```

```
These planes are waiting to be cleared to re-enter a runway:
Flight AirFrance212 to Paris.
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
1 planes have taken off from the airport

        Enter your selection now : 2
Is Flight USAir305  cleared for take off (Y/N):

Y
Flight USAir305 has now taken off from runway West

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight British909 to London.
 Flight Delta204 to Chicago.
 No planes are waiting for takeoff on runway SouthWest!
These planes are waiting for takeoff on runway West
Flight American493 to Seattle.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight AirFrance212 to Paris.
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
2 planes have taken off from the airport

        Enter your selection now : 1
Enter flight number: Continental339
Enter destination: Montreal
Enter runway: NorthEast
Flight Continental339 is now waiting for takeoff on runway NorthEast.

        Enter your selection now : 1
Enter flight number: jetBlue856
Enter destination: Atlanta
Enter runway: SouthWest
Flight jetBlue856 is now waiting for takeoff on runway SouthWest.

        Enter your selection now : 1
Enter flight number: AmericaWest691
Enter destination: San Francisco
Enter runway: West
Flight AmericaWest691 is now waiting for takeoff on runway West.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight British909 to London.
 Flight Delta204 to Chicago.
 Flight Continental339 to Montreal.
 These planes are waiting for takeoff on runway SouthWest
Flight jetBlue856 to Atlanta.
 These planes are waiting for takeoff on runway West
Flight American493 to Seattle.
 Flight AmericaWest691 to San Francisco.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight AirFrance212 to Paris.
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
2 planes have taken off from the airport

        Enter your selection now : 4
Enter the name of the new runway : West
Runway West already exists please choose another name.
Enter the name of the new runway : East
Runway East has opened.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight British909 to London.
 Flight Delta204 to Chicago.
 Flight Continental339 to Montreal.
 These planes are waiting for takeoff on runway SouthWest
Flight jetBlue856 to Atlanta.
 These planes are waiting for takeoff on runway West
Flight American493 to Seattle.
 Flight AmericaWest691 to San Francisco.
 No planes are waiting for takeoff on runway East!

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight AirFrance212 to Paris.
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
2 planes have taken off from the airport

        Enter your selection now : 1
Enter flight number: Lufthansa581
Enter destination: Muenchen
Enter runway: East
Flight Lufthansa581 is now waiting for takeoff on runway East.

        Enter your selection now : 1
Enter flight number: Alitalia576
Enter destination: Rome
Enter runway: East
Flight Alitalia576 is now waiting for takeoff on runway East.

        Enter your selection now : 1
Enter flight number: Continental304
Enter destination: Miami
Enter runway: SouthWest
Flight Continental304 is now waiting for takeoff on runway SouthWest.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight British909 to London.
 Flight Delta204 to Chicago.
 Flight Continental339 to Montreal.
 These planes are waiting for takeoff on runway SouthWest
```

```
Flight jetBlue856 to Atlanta.
 Flight Continental304 to Miami.
 These planes are waiting for takeoff on runway West
Flight American493 to Seattle.
 Flight AmericaWest691 to San Francisco.
 These planes are waiting for takeoff on runway East
Flight Lufthansa581 to Muenchen.
 Flight Alitalia576 to Rome.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight AirFrance212 to Paris.
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
2 planes have taken off from the airport

        Enter your selection now : 2
Is Flight British909  cleared for take off (Y/N):

Y
Flight British909 has now taken off from runway NorthEast

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight Delta204 to Chicago.
 Flight Continental339 to Montreal.
 These planes are waiting for takeoff on runway SouthWest
Flight jetBlue856 to Atlanta.
 Flight Continental304 to Miami.
 These planes are waiting for takeoff on runway West
Flight American493 to Seattle.
 Flight AmericaWest691 to San Francisco.
 These planes are waiting for takeoff on runway East
Flight Lufthansa581 to Muenchen.
 Flight Alitalia576 to Rome.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight AirFrance212 to Paris.
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
3 planes have taken off from the airport

        Enter your selection now : 3
Enter flight number : USAir705
Enter flight number : 6
Flight AirFrance212 is now waiting for takeoff on runway NorthEast

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
3 planes have taken off from the airport

        Enter your selection now : 2
Is Flight jetBlue856  cleared for take off (Y/N):

Y
Flight jetBlue856 has now taken off from runway SouthWest

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight Delta204 to Chicago.
 Flight Continental339 to Montreal.
 Flight AirFrance212 to Paris.
 These planes are waiting for takeoff on runway SouthWest
Flight Continental304 to Miami.
 These planes are waiting for takeoff on runway West
Flight American493 to Seattle.
 Flight AmericaWest691 to San Francisco.
 These planes are waiting for takeoff on runway East
Flight Lufthansa581 to Muenchen.
 Flight Alitalia576 to Rome.

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
4 planes have taken off from the airport

        Enter your selection now : 2
Is Flight American493  cleared for take off (Y/N):

Y
Flight American493 has now taken off from runway West

        Enter your selection now : 5
Enter runway: North
No such runway!
Enter runway: West
Enter new runway for plane AmericaWest691 : West
This is the runway that is closing!
Enter new runway for plane AmericaWest691 : North
No such runway!
Enter new runway for plane AmericaWest691 : NorthEast
Flight AmericaWest691 is now waiting for takeoff on runway NorthEast
Enter new runway for plane United954 : SouthWest
Flight United954 is now waiting for takeoff on Runway SouthWest
Runway West has been closed.

        Enter your selection now : 6
These planes are waiting for takeoff on runway NorthEast
Flight Delta204 to Chicago.
 Flight Continental339 to Montreal.
 Flight AirFrance212 to Paris.
 Flight AmericaWest691 to San Francisco.
 These planes are waiting for takeoff on runway SouthWest
Flight Continental304 to Miami.
 No planes are waiting for takeoff on runway West!
These planes are waiting for takeoff on runway East
Flight Lufthansa581 to Muenchen.
 Flight Alitalia576 to Rome.
```

```
        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight USAir705 to Boston.
Flight United954 to Pittsburgh.


        Enter your selection now : 8
5 planes have taken off from the airport

        Enter your selection now : 3
Enter flight number : United953
Enter flight number : United954
Enter flight number : 6
Flight USAir705 is now waiting for takeoff on runway NorthEast

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight United954 to Pittsburgh.


        Enter your selection now : 8
5 planes have taken off from the airport

        Enter your selection now : 0
The Airport is closing : Bye Bye....
Welcome to the Airport program!
Enter number of runways: 4
1Enter the name of runway number 1: North
2Enter the name of runway number 2: South
3Enter the name of runway number 3: East
4Enter the name of runway number 4: West
Select from the following menu:
        0. End the 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.

        Enter your selection now : 1
Enter flight number: Poke143
Enter destination: Galar
Enter runway: North
Flight Poke143 is now waiting for takeoff on runway North.

        Enter your selection now : 1
Enter flight number: Poke543
Enter destination: Hoenn
Enter runway: South
Flight Poke543 is now waiting for takeoff on runway South.

        Enter your selection now : 3
There are no planes waiting for clearance!

        Enter your selection now : 4
Enter the name of the new runway : NorthWest
Runway NorthWest has opened.

        Enter your selection now : 6
These planes are waiting for takeoff on runway North
Flight Poke143 to Galar.
 These planes are waiting for takeoff on runway South
Flight Poke543 to Hoenn.
 No planes are waiting for takeoff on runway East!
No planes are waiting for takeoff on runway West!
No planes are waiting for takeoff on runway NorthWest!

        Enter your selection now : 7
No planes are waiting to be cleared to re-enter a runway!

        Enter your selection now : 8
0 planes have taken off from the airport

        Enter your selection now : 1
Enter flight number: Poke789
Enter destination: Unova
Enter runway: NorthWest
Flight Poke789 is now waiting for takeoff on runway NorthWest.

        Enter your selection now : 1
Enter flight number: Poke323
Enter destination: Kalos
Enter runway: West
Flight Poke323 is now waiting for takeoff on runway West.

        Enter your selection now : 1
Enter flight number: Hyrule342
Enter destination: Kakariko
Enter runway: NorthWest
Flight Hyrule342 is now waiting for takeoff on runway NorthWest.

        Enter your selection now : 1
Enter flight number: Hyrule4224
Enter destination: Death Mountain
Enter runway: South
Flight Hyrule4224 is now waiting for takeoff on runway South.

        Enter your selection now : 1
Enter flight number: Wily5433
Enter destination: Wily's Fortress
Enter runway: East
Flight Wily5433 is now waiting for takeoff on runway East.

        Enter your selection now : 6
These planes are waiting for takeoff on runway North
Flight Poke143 to Galar.
 These planes are waiting for takeoff on runway South
Flight Poke543 to Hoenn.
 Flight Hyrule4224 to Death Mountain.
 These planes are waiting for takeoff on runway East
Flight Wily5433 to Wily's Fortress.
 These planes are waiting for takeoff on runway West
Flight Poke323 to Kalos.
 These planes are waiting for takeoff on runway NorthWest
Flight Poke789 to Unova.
 Flight Hyrule342 to Kakariko.

        Enter your selection now : 7
No planes are waiting to be cleared to re-enter a runway!
```

```
        Enter your selection now : 8
0 planes have taken off from the airport

        Enter your selection now : 2
Is Flight Poke143  cleared for take off (Y/N):

N
Flight Poke143 is now waiting to be allowed to re-enter a runway.

        Enter your selection now : 2
Is Flight Poke543  cleared for take off (Y/N):

Y
Flight Poke543 has now taken off from runway South

        Enter your selection now : 2
Is Flight Wily5433  cleared for take off (Y/N):

N
Flight Wily5433 is now waiting to be allowed to re-enter a runway.

        Enter your selection now : 2
Is Flight Poke323  cleared for take off (Y/N):

Y
Flight Poke323 has now taken off from runway West

        Enter your selection now : 1
Enter flight number: Wily323
Enter destination: Crash Man
Enter runway: South
Flight Wily323 is now waiting for takeoff on runway South.

        Enter your selection now : 1
Enter flight number: Hyrule3002
Enter destination: Zora Kingdom
Enter runway: NorthWest
Flight Hyrule3002 is now waiting for takeoff on runway NorthWest.

        Enter your selection now : 5
Enter runway: NorthWest
Enter new runway for plane Poke789 : North
Flight Poke789 is now waiting for takeoff on runway North
Enter new runway for plane Hyrule342 : North
Flight Hyrule342 is now waiting for takeoff on runway North
Enter new runway for plane Hyrule3002 : North
Flight Hyrule3002 is now waiting for takeoff on runway North
Runway NorthWest has been closed.

        Enter your selection now : 6
These planes are waiting for takeoff on runway North
Flight Poke789 to Unova.
 Flight Hyrule342 to Kakariko.
 Flight Hyrule3002 to Zora Kingdom.
 These planes are waiting for takeoff on runway South
Flight Hyrule4224 to Death Mountain.
 Flight Wily323 to Crash Man.
 No planes are waiting for takeoff on runway East!
No planes are waiting for takeoff on runway West!
No planes are waiting for takeoff on runway NorthWest!

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight Poke143 to Galar.
Flight Wily5433 to Wily's Fortress.


        Enter your selection now : 8
2 planes have taken off from the airport

        Enter your selection now : 2
Is Flight Poke789  cleared for take off (Y/N):

Y
Flight Poke789 has now taken off from runway North

        Enter your selection now : 2
Is Flight Hyrule4224  cleared for take off (Y/N):

Y
Flight Hyrule4224 has now taken off from runway South

        Enter your selection now : 2
Is Flight Hyrule342  cleared for take off (Y/N):

Y
Flight Hyrule342 has now taken off from runway North

        Enter your selection now : 2
Is Flight Wily323  cleared for take off (Y/N):

Y
Flight Wily323 has now taken off from runway South

        Enter your selection now : 2
Is Flight Hyrule3002  cleared for take off (Y/N):

Y
Flight Hyrule3002 has now taken off from runway North

        Enter your selection now : 2
No plane on any runway!

        Enter your 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!
No planes are waiting for takeoff on runway West!
No planes are waiting for takeoff on runway NorthWest!

        Enter your selection now : 7
These planes are waiting to be cleared to re-enter a runway:
Flight Poke143 to Galar.
Flight Wily5433 to Wily's Fortress.


        Enter your selection now : 8
7 planes have taken off from the airport

        Enter your selection now : 0
The Airport is closing : Bye Bye....
::::::::::::::
rationale.txt
::::::::::::::
```

Looking at **this** project, we decided to split it up into its component operations. Our most common operations would be adding and removing planes from runways, as well as searching when we enter things into the list of flights needed to clear.

We plan to use three ADTs, as follows:
ListRAB runways (we originally thought of using a CDLS, until we realized that using a list with a position integer updated via a modulus avoids repeated traversal)
AscendinglySortedList clearance (direct index access **for** sorting and searching - our intent is to sort flights into that list by lexicographic order of flight names)
QueueSLS Planes (held in Runway **class**, we only need to care about the front **for** taking off, and the back **for** adding **new** planes)

4 classes were created, counting our driver.

To hold all the information regarding planes themselves, we have the plane **class**. They hold the name of their runway **for** convenience with the clearance list, essentially. When we remove them from their runway, we want to maintain what runway they came from **for** putting them back in, later. It is worth the space tradeoff to hold the runway name. This also makes the toString method more convenient.

Plane:
- String flightNumber
- String destination
- String runway
- toString()
- Accessers and Mutators

We also need a runway object - This runway holds its name, a queue of all the planes on it, as well as methods to manipulate the planes on the runway (basically just add/remove methods **for** accessing the queue.) This also as it stands exists to be inherited from.

Runway:
- String name
- queueSLS planes
- toString
- add/remove plane
- Accessers and mutators

Finally, we are making an AirTrafficControl object - This is with future expandability in mind - potentially the ability to control several airports from one location. This also makes the driver cleaner. This **class** will hold our list (Array based) of runways. We chose ListRAB because we want the direct index access offered by it. This will make the round robin style of takeoff easier to deal with. We also hold a list, also array based (**for** direct index access - we are sorting **this** lexicographically by flight number, so that when we search **for** flight numbers, it will take less work overall. (For **this** reason, we went with the ascendingly sorted list, from lab 8) This will make the option to change runways slightly more of a pain, but with evaluation, it was determined that searching **for** a flight number to take something off of **this** list would be more common than closing a runway, so we optimize **for** the more common operation.

AirTrafficControl:
- ListRAB runways
- AscendinglySortedList clearance
- Int count
- Int position

Driver:
- BufferedReader stdin

Program flow:
Initialization:
Ask user **for**: number of runways
Run loop to create initial list of runways
- Prompt user **for** name of each runway

Load menu.
Option 1:
- Ask user **for**: flight number
  - destination
  - Runway
  - Check **for** runway\222s existence. If non-existant, prompt **for new** Runway
- If runway exists:
  - Create plane with flight number, destination, runway
  - Enqueue plane to runway.

Option 2:
- Check runway list (using position to access the index) to get runway we are launching from
- Dequeue plane into a temp plane.
- Ask user **if** flight # is cleared **for** takeoff.
  - If yes:
    - Increment position counter
    - Increment plane takeoff count
  - If no:
    - Add plane to clearance list

Option 3:
- Ask user **for** flight number
- Binary search clearance list **for** flight number
- Read the plane\222s runway (**this** is why we store the runway in the plane)
- Find the runway in the runway list
- Enqueue plane to runway

Option 4:
- Ask user **for** runway name
- Add runway to end of runway list

Option 5:
- Ask user what runway to close
- Check **for** runway existence
- Loop one:
  - Until runway is empty
    - Dequeue from runway into temp plane
    - Ask user **for new** runway
      - Check **for** runway existence
    - Set temp plane\222s runway to the **new** runway
    - Enqueue temp plane to runway
- Loop two:
  - Sequential search clearance list **for** runway
    - Ask user **for new** runway
      - Check **for** runway existence
    - Set plane\222s runway to the **new** runway
- Delete runway from runway list

Option 6:
- Parse through runway list
- Use toString method in runway to print out each runway\222s planes

Option 7:
- Parse through clearance list
- Use toString method in planes to print each plane\222s information

```
Option 8:
    Print our counter variable.
```