COMP285 – Data Structures & Algorithm Analysis

Spring 2018

Programming Project #3

Maximilian Escutia & Patrick Walker

Dr. Eric Jiang

5/3/2018

Assertion:

- We believe the project is complete. We have saved proj3 on our terminal under the specified directory.
- We managed to successfully finish this project before the deadline.
- We decided to create all of our classes within one single class file because we just wanted to have less classes to handle simultaneously.
- Implementing the logic and algorithms learned in class was a challenge however, we managed to successfully implement them into our code. :)
- At first we didn't know what was going to be the best way to store a flight's path information, however though trial and error, and through experimentation we found linked lists to be a very useful abstract data type to do this.
- Making sure that our cities were being added correctly into our stack and making sure they were popping and peeking properly was another challenging task we encountered.
- Overall this was a good project to end the semester and comeback from the previous one!

Summary:

USAir.java is a simple flight search system for *USAir* to help process customer requests of purchasing one-way tickets. The system first asks the user to input the text files which the program will read from. It then takes the departing city and a destination city as input and determines whether the airline serves a flight path from the departing city to the destination city. A trip may involve multiple flights and the output also includes the cost of each flight, and the total cost of the trip. The system is based on two input test files that specify cities *USAir* serves, and all other flight/cost information.

USAir.java Description:

USAir is the title for our program and the first class. It first asks the user to input the name of the text file that includes the cities associated with the USAir's flights & the name of the text file that includes the the flights of USAir. Then, through a do-while loop, the program continually has the user input a "to" and "from" city so the user can keep searching flights.

Our main method *USAir* has as parameters cityName & flightName - that are useful for creating new File objects, and a from and to city. This method:

- First uses the cityName and the flightName to create new files.
- It then creates two separate scanners for the cityFile and a single scanner for the flightFile. One cityFile scanner is used to find the number of cities in the cityFile and the second cityFile scanner is used, through a while loop, to input the cities into a new array of cities.
- It then uses the flightFile scanner, flightInput, to read each line of the flightFile. For each line of the flightFile, the use of a delimiter (",") allows the sourceName City and the destName City and the cost of the flight (parsed int) to be separated, checked (through separate methods described below) and stored into a new flight object. This flight is then added to the listofFlights, a LinkedList for the source City object.
- After the loops in this method, USAir also calls on a separate method "fly" that gives an output to the user.

Validation Methods:

- One of the “validation” methods of this program is the *findCity* method that checks to see if the parameter city exists within the array of cities (that had been created and filled in the main method). If the city array is empty, an EmptyStackException is thrown. If the city is in the array of cities, this method returns the parameter city.
- The second “validation” method of this program is the *findFlight* method that uses two parameter cities to check if the source city’s listofFlights (LinkedList) destination city is the same city as the second parameter (the destination city “destinationCity”). If the flight is in the listofFlights, the method returns the flight between the two cities.
- The third “validation” method of this program is the *isPath* method, that through the use of a stack of cities, checks to see if there is a path between the two parameter cities.

Other Methods:

- The method *fly* first checks to see if the sourceCity and/or the destinationCity is in the array of cities by using the findCity method. If one or both of the cities is not in the array of cities, the method returns a message to the user. If the cities exist in the array of cities, the method then checks to see if there is path between the cities. If the path does not exists the user is returned a message. Thus the only case left is if there is a path between two valid cities. In the final else statement, through the use of a new stack “reverse”, the user is returned the individual flights with their respective costs as well as the total cost of the flight path. After the total cost message is printed, the method resets all the cities using the *reset* method (described below); this allows the user to use the fly method again, and in turn allows the user to search for another flight path. The USAir method can then be used again.
- We also wrote a method *allVisited* that checks to see if a city has been visited, and returns an integer of that city, and if that city has not been visited, returns a number other than -1. If the city has been visited, the method returns -1.
- The method *reset* has all the cities of the city array marked univisited.
- The method *getCost* returns the total cost of travel from the bottom to the top of the stack.

Other Classes:

- The Class *Path* has a Stack <City> and int costOfFlights as constructors; this class implements a method *Path* that establishes the Stack <City> as the stack of cities for the path & establishes the costOfFlights of the path (by calling on the *getCost* method described heretofore) on the Stack of cities of the path
- The Class *City* has a name, a LinkedList<Flight listofFlights, and a visited boolean as constructors; this class implements a method *City* that establishes the name of the city to nameofCity (from the parameter), sets the LinkedList<Flight> to listofFlights and visited to false.

The Class *Flight* has a source City sourceCity, destination City destinationCity, int costOfFlights as constructors; this class implements a method *Flight* that establishes the sourceCity of the flight to the City source in the parameter, the destinationCity of the flight to the City destination in the parameter and the costOfFlights to the integer city1 of in the parameter.

Sample runs:
USAir.java

```
Script started on Tue 01 May 2018 06:06:42 PM PDT
^[]0;pwalker@cslab1:~/COMP285/proj3^G^[[?1034h[pwalker@cslab1 proj3]$ java
US^GAir
Enter the name of the cityFile:
cityFile.txt
Enter the name of the flightFile:
flightFile.txt

From?
Albuquerque

To?
San Diego

Request is to fly from Albuquerque to San Diego.
Flight from Albuquerque to Chicago   Cost: $250
Flight from Chicago to Los Angeles   Cost: $230
Flight from Los Angeles to San Diego   Cost: $80
Total Cost ..................... $560

Do you want to search again? (Type y or n)
y

From?
Albuquerque

To?
Paris

Sorry. USAir does not serve Paris.


Do you want to search again? (Type y or n)
y

From?
Mexico

To?
Paris

Sorry. USAir does not serve Mexico or Paris.


Do you want to search again? (Type y or n)
y

From?
San Diego

To?
Chicago

Sorry. USAir does not fly from San Diego to Chicago.
```

```
Do you want to search again? (Type y or n)
y

From?
Chicago

To?
San Diego

Request is to fly from Chicago to San Diego.
Flight from Chicago to Los Angeles   Cost: $230
Flight from Los Angeles to San Diego   Cost: $80
Total Cost ..................... $310

Do you want to search again? (Type y or n)
n

Goodbye...
Thank you for using USAir search system!
^[]0;pwalker@cslab1:~/COMP285/proj3^G[pwalker@cslab1 proj3]$ exit
exit

Script done on Tue 01 May 2018 06:08:00 PM PDT
```

Source code:

USAir.java

```
/**
 * Name: USAir.java
 * Last Modified: 04/30/2018
 * Author: Max Escutia & Patrick Walker
 * Description: A simple flight search system for USAir to help process customer requests
 *          of purchasing one-way tickets.
 * */

import java.io.*;
import java.util.*;

public class USAir {

        public static void main(String [] args) throws FileNotFoundException {
                //Request the user for files to be read.
                Scanner input = new Scanner(System.in);
                System.out.println("Enter the name of the cityFile:");
                String cityName = input.nextLine();
                System.out.println("Enter the name of the flightFile:");
                String flightName = input.nextLine();

                //Will loop through logic of code if user wants to search again
                Scanner scan = new Scanner(System.in);
                boolean stop = false;
                while(!stop) {
                        System.out.println("\nFrom?");
                        String from = input.nextLine();


                        System.out.println("\nTo?");
                        String to = input.nextLine();

                        USAir finalOutput = new USAir(cityName, flightName, from, to);

                        System.out.println("\nDo you want to search again? (Type y or n)");
                        String s = scan.nextLine();
```

```
                        if(s.equals("y") || s.equals("Y")) {
                                stop = false;

                        } else if (s.equals("n") || s.equals("N")){
                                stop = true;
                                System.out.println("\nGoodbye... \nThank you for using USAir search system!");
                        }
                }
}


private City [] cities;
public LinkedList<Path> path = new LinkedList<Path>();

public USAir(String cityName, String flightName, String from, String to)throws FileNotFoundException{

        File cityFile = new File(cityName);
        File flightFile = new File(flightName);

        Scanner cityInput = new Scanner(cityFile);
        Scanner cityInput2 = new Scanner(cityFile);
        Scanner flightInput = new Scanner(flightFile);

        int counter = 0;
        while(cityInput.hasNext()){
                cityInput.nextLine();
                counter++;
        }
        cities = new City[counter];
        for(int i = 0;cityInput2.hasNext();i++){
                cities[i]=new City(cityInput2.nextLine().trim());

        }
        while(flightInput.hasNext()){
                String line = flightInput.nextLine();
                Scanner flightInformation = new Scanner(line);
                flightInformation.useDelimiter(",");
                String sourceName = flightInformation.next().trim();
                City sourceCity = findCity(sourceName);
                String destName = flightInformation.next().trim();
                City destinationCity = findCity(destName);
                int flightCost = Integer.parseInt(flightInformation.next().trim());
                Flight newFlight = new Flight(sourceCity,destinationCity,flightCost);
                sourceCity.listofFlights.add(newFlight);


        }
        //Will run the algorithm to find the cheapest flight cost path.
                fly(from, to);

}

private City findCity(String s){
        if(cities.length==0){
                throw new EmptyStackException ();

        }

        for(int i =0;i<cities.length;i++){
                if(s.equals(cities[i].name)){
                        return cities[i];
                }

        }
        return null;
}

//
public void fly(String sourceCity, String destinationCity){
        if(findCity(sourceCity)==null){
                System.out.print("\nSorry. USAir does not serve "+sourceCity);
                if(findCity(destinationCity)==null){
```

```
                                        System.out.print(" or "+destinationCity+".\n\n");
                                }else{
                                        System.out.print("\n");
                                }
                        }else if(findCity(destinationCity)==null){
                                System.out.println("\nSorry. USAir does not serve "+destinationCity+".\n");
                        }else{
                                Stack<City> userPlan = isPath(findCity(sourceCity),findCity(destinationCity));
                                if(userPlan.isEmpty()){
                                System.out.print("\nSorry. USAir does not fly from "+sourceCity+" to "+destinationCity+".\n");
                                }else{
                                        System.out.print("\nRequest is to fly from "+sourceCity+" to "+destinationCity+".\n");
                                        Stack<City> reverse = new Stack<City>();
                                        while(!userPlan.isEmpty()){
                                                reverse.push(userPlan.pop());
                                        }
                                        int totalCost = 0;
                                        while(!reverse.isEmpty() && reverse.size()>1){
                                                City city1 = reverse.pop();
                                                City city2 = reverse.peek();
                                                Flight f = findFlight(city1,city2);
                                                System.out.println("Flight from "+f.sourceCity.name+" to
"+f.destinationCity.name+"   Cost: $"+f.costOfFlights);
                                                totalCost+=f.costOfFlights;
                                        }
                                        System.out.println("Total Cost ..... $"+totalCost);
                                        reset();
                                }
                        }
                }

                //This will check if there is a flight that exists between cities 'sourceCity' and 'destinationCity'
                private static Flight findFlight(City sourceCity, City destinationCity){
                        for(int i = 0; i<sourceCity.listofFlights.size();i++){
                          if (sourceCity.listofFlights.get(i).destinationCity == destinationCity){
                                  return sourceCity.listofFlights.get(i);
                          }
                        }
                          return null;
                }

                private Stack<City> isPath(City sourceCity, City destinationCity){
                        Stack<City> userPlan = new Stack<City>();

                        userPlan.add(sourceCity);
                        sourceCity.visited = true;
                        while(!userPlan.isEmpty() && userPlan.peek() != destinationCity){
                                City currentCity = userPlan.peek();
                                //make sure that there are no unvisited cities from currentCity remaining
                                if(allVisited(currentCity) == -1){
                                        userPlan.pop();
                                }else{
                                        //the next unvisited city will be currentCity
                                        City city1 = currentCity.listofFlights.get(allVisited(currentCity)).destinationCity;
                                        city1.visited = true;
                                        userPlan.push(city1);
                                }
                        }
                        return userPlan;
                }

                //checking if a city has been visited
                private int allVisited(City curr){
                        for(int i = 0; i < curr.listofFlights.size(); i++){
                                if(!curr.listofFlights.get(i).destinationCity.visited)
                                        return i;
                        }
                        return -1;
                }


                //reset returns all City.visited booleans to false
```

```
	private void reset(){
		for(int i = 0;i < cities.length; i++){
			cities[i].visited = false;
		}
	}


	//This method will return the total cost of the travel from the bottom to the top of the stack
	private int getCost(Stack<City> it){
		int total = 0;
		Stack<City> reverse = new Stack<City>();
		for(int k = 1; k <= it.size(); k++){
			reverse.push(it.get(it.size()-k));
		}
		while(!reverse.isEmpty() && reverse.size() > 1){
			City city1 = reverse.pop();
			City city2 = reverse.peek();
			Flight flight1 = findFlight(city1,city2);
			total += flight1.costOfFlights;
		}
		return total;
	}


	//This class will find a path and get the cost of that path
	public class Path{
		public Stack<City> itin;
		public int costOfFlights;

		public Path(Stack<City> i){
			itin = i;
			costOfFlights = getCost(itin);

		}
	}

	//This will establish a city object
	public class City{
		public String name;
		public LinkedList<Flight> listofFlights;
		public boolean visited;

		public City(String nameofCity){
			name = nameofCity;
			listofFlights = new LinkedList<Flight>();
			visited = false;
		}
	}

	//This will establish a flight object. It will take the 'from' city and 'to' city and get the cost of that specific flight
	public class Flight {
		public City sourceCity;
		public City destinationCity;
		public int costOfFlights;
		public Flight(City source, City destination ,int city1){
			sourceCity = source;
			destinationCity = destination;
			costOfFlights = city1;
		}
	}
}
```