Assignment 5 Sustech Airline

> Contributors:
> ZHU Yueming (designer),
>
> WANG YITONG, WANG LISHUANG (junit),
>
> ZHANG WENCHENG (document),
>
> LI KAI (document)
>
> ZHENG BOWEN (tester)

Quick Navigation

1. `SearchPlan`
2. `SustechTime`

——Fields: `hour` and `time`

3. `Flight`

——Fields: `flightNo`, `departTime`, `arriveTime`, `price`

——Methods: `Flight`, `toString`

4. `Airline`
5. `SustechAirline`

——Methods: `addFlight`, `getFlightInfo`, `getFlightRoute`, `removeFlight`, `getAllFlightsAboutDepartPort`, `isContainsFlight`, `isRoundTrip`, `searchAllRoutes`, `searchBestRoutes`

6. `Search`

——Fields: `AIRPORT_FEE`, `TRANSIT_FEE`

——Methods: `searchRoutes`, `searchBestRoutes`

Description

Lily's dream is to build an airline for her old university, SUSTech. Now she is trying to code by Java. There are N **ports** the airplane leaves or goes off. The info of the flight is always changing. Sometimes there will be a flight added or removed. Each flight contains depart port and arrive port. If the flight is added to the airline, it means that the airplane can fly from one port to the other by it with the proper time. The **flight** describes the information about airplane on the airline by `flightInfo`, including `flightNo`, `departPort`, `arrivePort`, `departTime`, `arriveTime`, `price`.

In this Assignment, you should design a project to support all the method below to solve the SUSTech Airline Problem. Lily asks you to search the best way how to move from one port to the other (maybe they are not connected directly) in **only one** day. She has two search plans: **LESS_TIME** and **LESS_PRICE**. She also shows three search ways: **DirectSearch**, **TransitOnceSearch** and **UnlimitedSearch**(bonus). Obviously, you should design different methods when facing the different search plans and ways. **There is only one flight connecting the same depart port and the same arrive port.** If you take more than one route, you should make sure: **The end station of i-th airline you choose and the start station of i+1-th airline is the same!** Besides, the transfer time should be **more than** 30 min! The specific description and format shows below.

Questions and Classes Introduction

1. `SearchPlan`:

```
public enum SearchPlan {
    LESS_TIME, LESS_PRICE
}
```

`SearchPlan` is an `enum` to distinguish the strategy you use when searching routes

- `LESS_TIME`: Find the routes taking less time.
- `LESS_PRICE`: Find the routes costing less.

2. Question 1 [10 points]: `SustechTime`:

- **Description to the class:** This class is designed to denote time.
- **Fields:**

  ```
  private int hour;
  private int minute;
  ```

- **Method** `SustechTime`:

  ```
  public SustechTime(String timeInfo);
  ```

- Constructor of class `SustechTime`.
- The format of timeInfo is `[hour]:[minute]`
- The test cases of time are all correct value and the range of all test cases are in 00:00 to 23:59

- **Method toString()**:

  return a String format like `[hour]:[minute]`, if the hour or minute is less then 10, adding `0` in front.

- **Method `timeDifference`**:

```
public int timeDifference(SustechTime targetTime);
```

  - You should find out how many minutes difference between the two different time.

    eg. minutes difference between `13 30` and `17 20` is 230 minutes.

- **Example**:

  Sample code:

```
SustechTime time1 = new SustechTime("5:3");
SustechTime time2 = new SustechTime("15:3");
SustechTime time3 = new SustechTime("05:13");
System.out.println(time1);
System.out.println(time2);
System.out.println(time3);
System.out.println(time1.timeDifference(time2));
System.out.println(time2.timeDifference(time1));
```

  Sample output:

```
05:03
15:03
05:13
600
600
```

Hints

Do not modify or remove any methods or fields that have been already defined.

You can add other methods or fileds that you think are necessary

Q1. What to Submit:

SustechTime.java

Q1. LocalJudge:

TestLocalQ1SustechTime.java

3. Question 2 [20 points]: `Flight`:

- **Description to the class:**

  The class to denote flights of airline.

- **Property `flightNo`**:

  ```
  private String flightNo;
  ```

  - Every flight has its own `flightNo`.

- **Property `departTime`**:

  ```
  private SustechTime departTime;
  ```

  - It means the time when the flight departs.

- **Property `arriveTime`**:

  ```
  private SustechTime arriveTime;
  ```

  - It means the time when the flight arrive its destination.

- **Property `price`**:

  ```
  private int price;
  ```

  - Literally, it represent how much money you should pay for the flight.

- **Method `Flight`**:

  ```
  public Flight(String flightInfo);
  ```

  - Constructor of class `Flight`.
  - Format of parameter `flightInfo`:

    `flightInfo`: `[flightNo] [departPort] [arrivePort] [departTime] [arriveTime] [price]`

    eg. `S102 A C 9:20 14:20 1210`

- **Method `toString`**:

```
public String toString();
```

- Format of returned value:

  `[flightNo] [[departPort] -> [arrivePort]] [departTime] -> [arriveTime] ([price])`

  eg. `S103 [A -> E] 03:00 -> 05:30 (900)`

- **Hints**

  Do not modify or remove any methods or fields that have been already defined.

  You can add other methods or fileds that you think are necessary

Q2. What to Submit:

SustechTime.java, Flight.java

Q2. LocalJudge:

TestLocalQ2Flight.java

4. Question 3 [30 points]: `Airline`:and `SustechAirline`:

SustechAirline is an implement class of interface Airline, which you need to provide and implement all abstract methods defined in Ariline.

- **Method** `addFlight`:

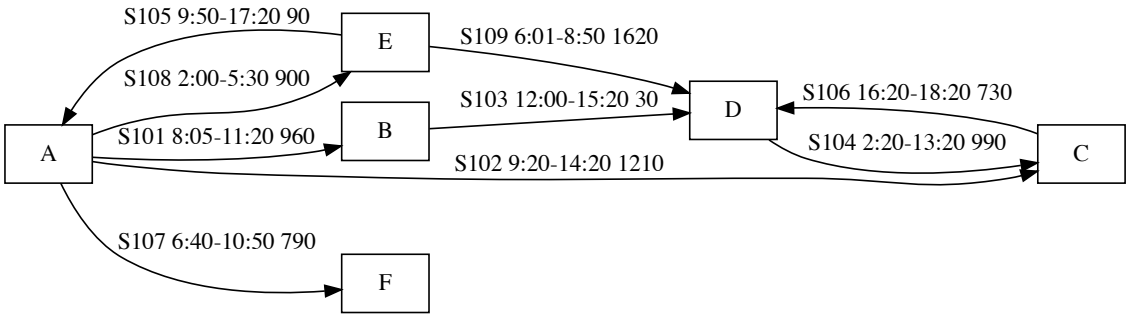
```
public void addFlight(String flightInfo);
```

  - Add flight according to the `flightInfo`, which format is `[flightNo] [departPort] [arrivePort] [departTime] [arriveTime] [price]`
  - We can not add a flight which `flightNo` has been in Airline, which means you need to check whether flight no is in airline.

    Code example 1:

```
Airline sustechAirline = new SustechAirline();
sustechAirline.addFlight("S103 B D 12:00 15:20 30");
sustechAirline.addFlight("S101 A B 8:05 11:20 960");
sustechAirline.addFlight("S102 A C 9:20 14:20 1210");
sustechAirline.addFlight("S106 C D 16:20 18:20 730");
sustechAirline.addFlight("S104 D C 2:20 13:20 990");
sustechAirline.addFlight("S105 E A 9:50 17:20 90");
sustechAirline.addFlight("S107 A F 6:40 10:50 790");
sustechAirline.addFlight("S108 A E 2:00 5:30 900");
sustechAirline.addFlight("S109 E D 6:01 8:50 1620");
sustechAirline.addFlight("S109 H D 6:01 8:50 1620");//ignore duplicate fligthNo
```

Graph of example 1:



- **Method `getFlightInfo`**:

```
public String getFlightInfo(String flightNo);
```

  - Get the information of the certain flight according to `flightNo`.
  - **Format**:

    `[flightNo] [[departPort] -> [arrivePort]] [departTime] -> [arriveTime] ([price])`

    The format is the same as `toString()` method of Flight

    Example:

    `S103 [A -> E] 03:00 -> 05:30 (270)`
  - **Returns**: the format of flightInfo. If cannot find the flight, return null.
  - **Code Example 2**:

    ```
    System.out.println(sustechAirline.getFlightInfo("S108"));
    ```

    Output:

    ```
    S108 [A -> E] 02:00 -> 05:30 (900)
    ```

- **Method `getFlightRoute`**:

```
public List<String> getFlightRoute()
```

  - Return a `List<String>`. Each String represents flights in airline from one departPort to its all arrivePort.
  - Format:

    All departPorts are separated by only **one space**，and **there is no space after the last arrivePort**

```
[departPort1]->[arrivePort1] [arrivePort2]...
[departPort2]->[arrivePort3] [arrivePort4]...
```

- Sorted rules:
  - The sorted rules are based on alphabetical order.
  - String in List sorted by departPort firstly
  - In each String sorted by arrivalPort

  eg.

```
A->B C E F
B->D
C->D
D->C
E->A D
```

- **Code Example 3**: According to Code Example 1, the output of do

```
System.out.println(Arrays.deepToString(sustechAirline.getFlightRoute().
toArray()));
for (String str:sustechAirline.getFlightRoute()) {
            System.out.println(str);
        }
```

  would be:

```
[A->B C E F, B->D, C->D, D->C, E->A D]
A->B C E F
B->D
C->D
D->C
E->A D
```

- **Method `removeFlight`**:

```
public boolean removeFlight(String flightNo);
```

- Remove the certain flight according to the `flightNo`.
- **Returns**: **true** if the `flightNo` is included in your airline, otherwise **false**
- **Code Example 4**: According to Code Example 1, do following code:

```
 sustechAirline.removeFlight("S103");
 sustechAirline.removeFlight("S102");
 System.out.println(Arrays.deepToString(sustechAirline.getFlightRoute()
.toArray()));
```

output

```
[A->B E F, C->D, D->C, E->A D]
```

- **Method `getAllFlightsAboutDepartPort`**:

```
public List<Flight> getAllFlightsAboutDepartPort(String departPort);
```

  - The method is to get all flights according to departPort by **ascending** order of **flightNo**.
  - **Parameters**: `departPort` - the port the flight departs from
  - **Returns**: all the fights whose departPort is the same as this `departPort`
  - **Returns:** Return a empty list if there is no flight about departPort.
  - **Code Example 5**: Continue to Code Example 4, do following code:

```
for (Flight flight : sustechAirline.getAllFlightsAboutDepartPort("E"))
{
            System.out.println(flight);
      }
System.out.println(sustechAirline.getAllFlightsAboutDepartPort("MM").si
ze());//empty list
```

Output:

```
S105 [E -> A] 09:50 -> 17:20 (90)
S109 [E -> D] 06:01 -> 08:50 (1620)
0
```

- **Method `isContainsFlight`**:

```
public boolean isContainsFlight(String deportPort, String arrivePort);
```

  - Check whether there is a flight that departs from `departPort` and arrive at `arrivePort`.
  - **Returns**: **true** if there is a flight in your airline, otherwise return **false**
- **Method `isRoundTrip`**:

```
public boolean isRoundTrip(String port1, String port2);
```

- It is verified that **no more than two flight between two ports**

  Example:
  For ports A and B, we have two flights: `A->B` and `B->A`. Then there is a round trip between there two ports
- **Returns**: **true** if there are two flights(with different direction) between two ports, otherwise **false**

- **Method `searchAllRoutes`**:

```
public List<String> searchAllRoutes(String departPort, String arrivePort,
Search search);
```

- Find all routes, which depart from `departPort` and arrive at `arrivePort`, according to `search`(**DirectSearch**, **TransitOnceSearch**, or **UnlimitedSearch**)
- The implement algorithm of search is written in subclasses of `Search`, so that the code in this class should contain following statement:

  `search.searchRoutes(departPort, arrivePort);`
- **Returns**: A list of string that contains all routes under the way `search` (**DirectSearch**, **TransitOnceSearch**, or **UnlimitedSearch**). The string of each route can be got by method `toString` (in class `Flight`). In one string of a certain route, different flights should be seperated by `\t` ([Tab]). **eg.**

  ```
  "S101 [B -> C] 8:00 -> 9:00 (1050)\tS102 [C -> D] 10:50 -> 12:10
  (1100)"  //The string of a certain route
  ```

- **Method `searchBestRoute`**:

```
public String searchBestRoute(String departPort, String arrivePort, Search
search, SearchPlan searchPlan);
```

- Find all routes, which depart from `departPort` and arrive at `arrivePort`, according to `search`(**DirectSearch**, **TransitOnceSearch**, or **UnlimitedSearch**) and `searchPlan`(`LESS_TIME` or `LESS_PRICE`).
- The implement algorithm of search is written in subclasses of Search, so that the code in this class should contain following statement:

  `search.searchRoutes(departPort, arrivePort);`
- **Returns**:
  - The string that contains the best route under the way `search` (**DirectSearch**, **TransitOnceSearch**, or **UnlimitedSearch**) and the plan `searchPlan`(`LESS_TIME`

or `LESS_PRICE`).

- The string of each route can be got by method `toString` (in class `Flight`). In one string of a certain route, different flights should be seperated by `\t` ([Tab]). **eg.**

```
"S101 [B -> C] 8:00 -> 9:00 (1050)\tS102 [C -> D] 10:50 -> 12:10
(1100)"  //The string of a certain route
```

- If there are over one best routes, return the first route sorted by alphabetical order. **eg.**

  Code example 6

```
        Airline airline = new SustechAirline();
        Search search = new TransitOnceSearch();
        airline.addFlight("S101 A B 12:00 13:00 3000");
        airline.addFlight("S102 B C 14:00 15:00 3000");
        airline.addFlight("S103 A E 12:00 18:00 30");
        airline.addFlight("S104 E C 19:00 21:20 30");
        airline.addFlight("S105 A F 12:00 13:00 3000");
        airline.addFlight("S106 F C 14:00 15:00 3000");
        airline.addFlight("S107 A G 12:00 18:00 30");
        airline.addFlight("S108 G C 19:00 21:20 30");

        airline.getFlightRoute().forEach(System.out::println);



 System.out.println(airline.searchBestRoute("A","C",search,SearchPl
an.LESS_PRICE));

 System.out.println(airline.searchBestRoute("A","C",search,SearchPl
an.LESS_TIME));
```

  Output:

```
A->B E F G
B->C
E->C
F->C
G->C
S103 [A -> E] 12:00 -> 18:00 (30) S104 [E -> C] 19:00 -> 21:20 (30)
S101 [A -> B] 12:00 -> 13:00 (3000) S102 [B -> C] 14:00 -> 15:00
(3000)
```

Q3. What to Submit:

SustechTime.java, Flight.java, Airline.java, SustechAirline.java, SearchPlan.java, Search.java or other you think is necessary

Q3. LocalJudge:

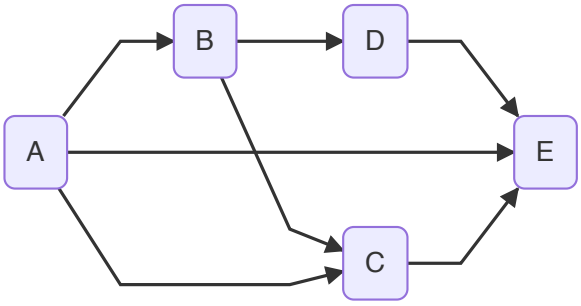
TestLocalQ3Airline.java

5. Question 4, Question 5, Question 6(bonus): `Search`:

- **Description to the class:**

  This class is an **abstract** class that can find the determined routes(with specified departPort and specified arrivePort) in 3 different ways: **DirectSearch.** Q4, **TransitOnceSearch** Q5, **UnlimitedSearch(bonus)** Q6

  (class `Search` should be implemented in 3 concrete classed: **DirectSearch, TransitOnceSearch, UnlimitedSearch**)



  - **DirectSearch [15 points]: only** find the route that travels **directly** from departPort to arrivePort (without any transfer stations)

    eg. `searchRoutes("A", "E")` returns `A -> E`, not `A -> B -> D - >E` or `A -> C -> E`

  - **TransitOnceSearch [25 points]:** find the route that involves **only one** transfer station

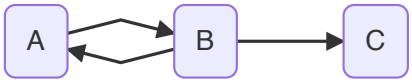
    eg. `searchRoutes("A", "E")` returns `A -> C -> E`, not `A -> B -> D -> E`

  - **UnlimitedSearch [30 points](bonus) :** find **all possible** routes that can travel from departPort to arrivePort (no matter how many transfer stations there are in the routes)

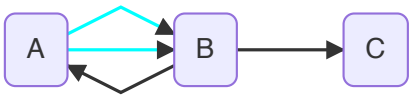
    eg. `searchRoutes("A", "E")` returns `A -> B -> D -> E`, `A -> B -> C -> E`, `A -> E`, `A -> C -> E`

Things to be noticed of our provided test cases:

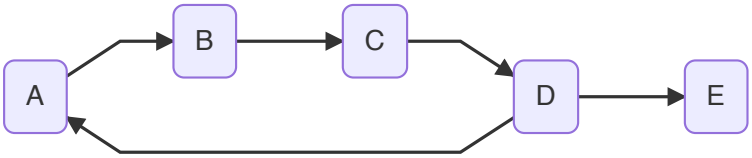
- There is only one flight from one airport to another, which means the case below is possible,



but the case below is impossible(one of the flights marked in blue is redundant)



- There might be a ring within the flights, which means the case below is possible.



- **Property `AIRPORT_FEE`:**

```
protected final int AIRPORT_FEE = 80;
```

  - `AIRPORT_FEE`: You need to pay `AIRPORT_FEE` **whenever** you arrive at an airport (**even you are at the depart port**)
- **Property `TRANSIT_TIME`:**

```
protected final int TRANSIT_TIME = 30;
```

  - `TRANSIT_TIME`: It will cost **over** 30 minutes for you to complete transfer formalities and wait for the next flight **only** when you are in **a transfer station**.
- **Method Search**

```
public Search()
```

  We only use none parameter constructor to test those three search classes.
- **Method `searchRoutes`:**

```
public abstract List<String> searchRoutes(String departPort, String
arrivePort);
```

  - Implement this method according to the requirement in the 3 different subclass (**DirectSearch, TransitOnceSearch, UnlimitedSearch(bonus)**) separately. If no routes satisfying the requirement are found, it should return a `List<String>` but with no elements in it.

- The search result may contain a ring route, but for each flight in the route, it can only appear exactly once.
- The transfer time between two flight should be larger then **30** minutes.

  For example:

  - We **can not** take flight (S101) then transfer to flight(S102) , because the transit time is no larger then 30 minutes.
  - We **can** take flight(S102) then transfer to flight (S103), because the transit time is larger then 30 minute.
  - We **can not** take flight(S103) then transfer to flight (S104), because the depart time of S104 is earlyer then arrive time of S103

- **Method** `Search.searchBestRoute`:

```
public abstract String searchBestRoute(String departPort, String
arrivePort, SearchPlan searchPlan);
```

  - This method finds the route with `SearchPlan`(less time or less price) to be considered. There should be only one.
  - If there are over one best routes, return the first route sorted by alphabetical order.
  - specified:

BestRoute	SearchPlan.LESS_TIME	SearchPlan.LESS_PRICE
Amog all routs, find min of that.	`Arrival time of final flight` - `depart time of first flight`	for all flights in route, calculate the sum of the `price` and `AIRPORT_FEE` of each

- **Hints**

  Do not modify or remove any methods or fields that have been already defined.

  You can add other methods or fileds that you think are necessary

Q4. What to Submit:

SustechTime.java, Flight.java, Airline.java, SustechAirline.java, SearchPlan.java, Search.java, DirectSearch.java or other you think is necessary

Q4. LocalJudge:

TestLocalQ4Direct.java

Q5. What to Submit:

SustechTime.java, Flight.java, Airline.java, SustechAirline.java, SearchPlan.java, Search.java, TransitOnceSearch.java or other you think is necessary

Q5. LocalJudge:

TestLocalQ5TransitOnce.java

Q6. What to Submit:

SustechTime.java, Flight.java, Airline.java, SustechAirline.java, SearchPlan.java, Search.java, UnlimitedSearch.java or other you think is necessary

Q6. LocalJudge:

TestLocalQ6Unlimited.java

Other Class

You can add other class that you think is necessary.