Gestione di un sistema aeroportuale combinando Observer e Strategy

Mihail Teodor Gurzu

18 giugno 2020

1 Scenario di Applicazione

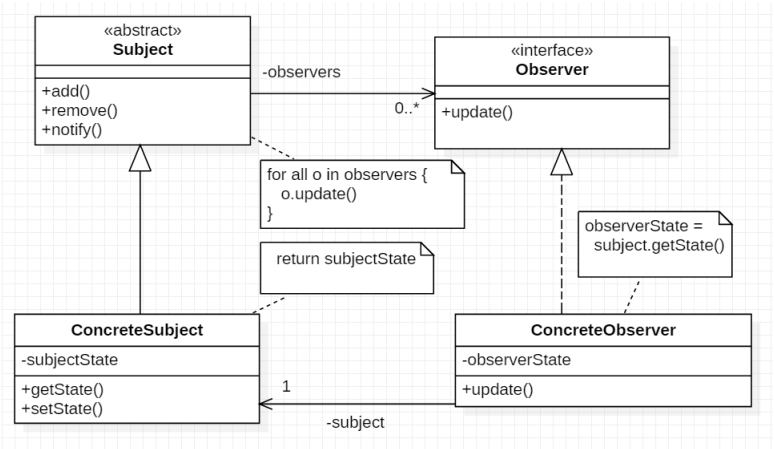
Supponiamo di voler gestire un sistema aeroportuale, all'interno del quale è presente una torre di controllo. Questa ha il compito di inserire i voli in arrivo ed in partenza nelle apposite liste nel sistema e di visualizzarle sui tabelloni degli arrivi e delle partenze presenti all'interno dell'edificio.
In particolare, un volo in arrivo possiede uno stato che che può essere di tipo *standard* o *emergency*. Se lo stato è di tipo *emergency*, il volo in questione verrà posizionato in cima alla lista degli arrivi (se presenti altri voli con questo stato verrà posizionato in coda a questi) e quindi avrà la priorità rispetto ai voli con stato *standard*.
Una volta che un volo compie l'atterraggio, la torre di controllo può rimuoverlo dalla lista degli arrivi che viene quindi ricalcolata.
Inoltre è presente una biglietteria il cui compito è quello di inserire una persona nella lista dei passeggeri di un determinato volo.

2 Design Patterns Coinvolti

Per la realizzazione del sistema si possono combinare due design patterns: *Observer* e *Strategy*.

2.1 Observer Pattern

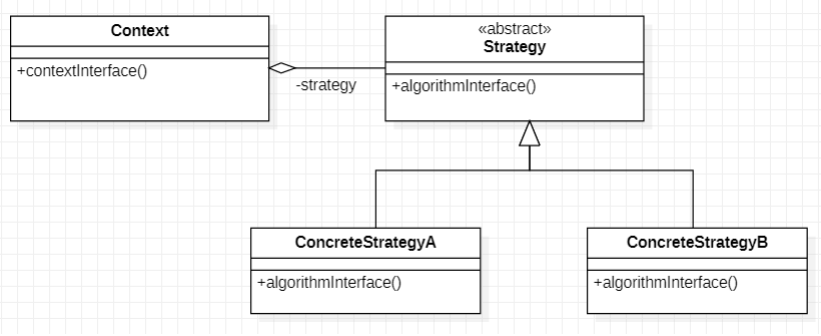
Figura 1: Observer UML diagram



Observer consente la definizione di associazioni di dipendenza uno a molti fra oggetti, in modo tale che se un oggetto cambia stato, tutti gli oggetti dipendenti da questo siano notificati e aggiornati automaticamente.
Questo pattern consente di partizionare un insieme di classi collaboranti mantenendo un alto livello di consistenza fra classi correlate mantenendo un alto livello di disaccoppiamento(Subject e Observer possono appartenere a diversi livelli di astrazione del sistema), permettendo quindi la riusabilità delle classi.

2.2 Strategy Pattern

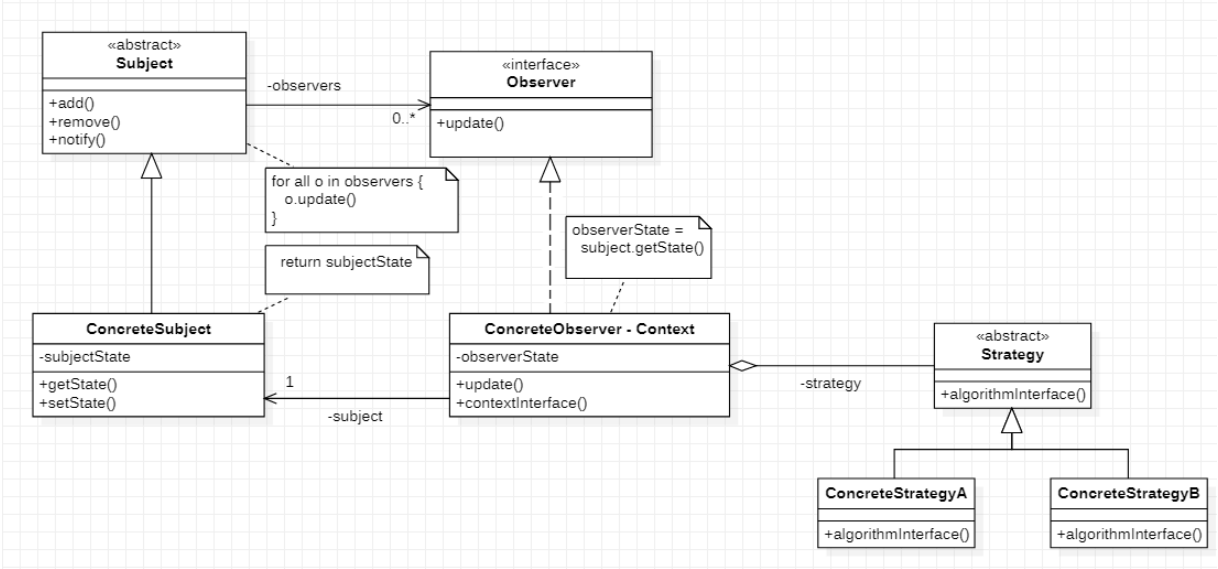
Figura 2: Strategy UML diagram



Consente di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Strategy permette agli algoritmi di variare indipendentemente dai client che ne fanno uso.

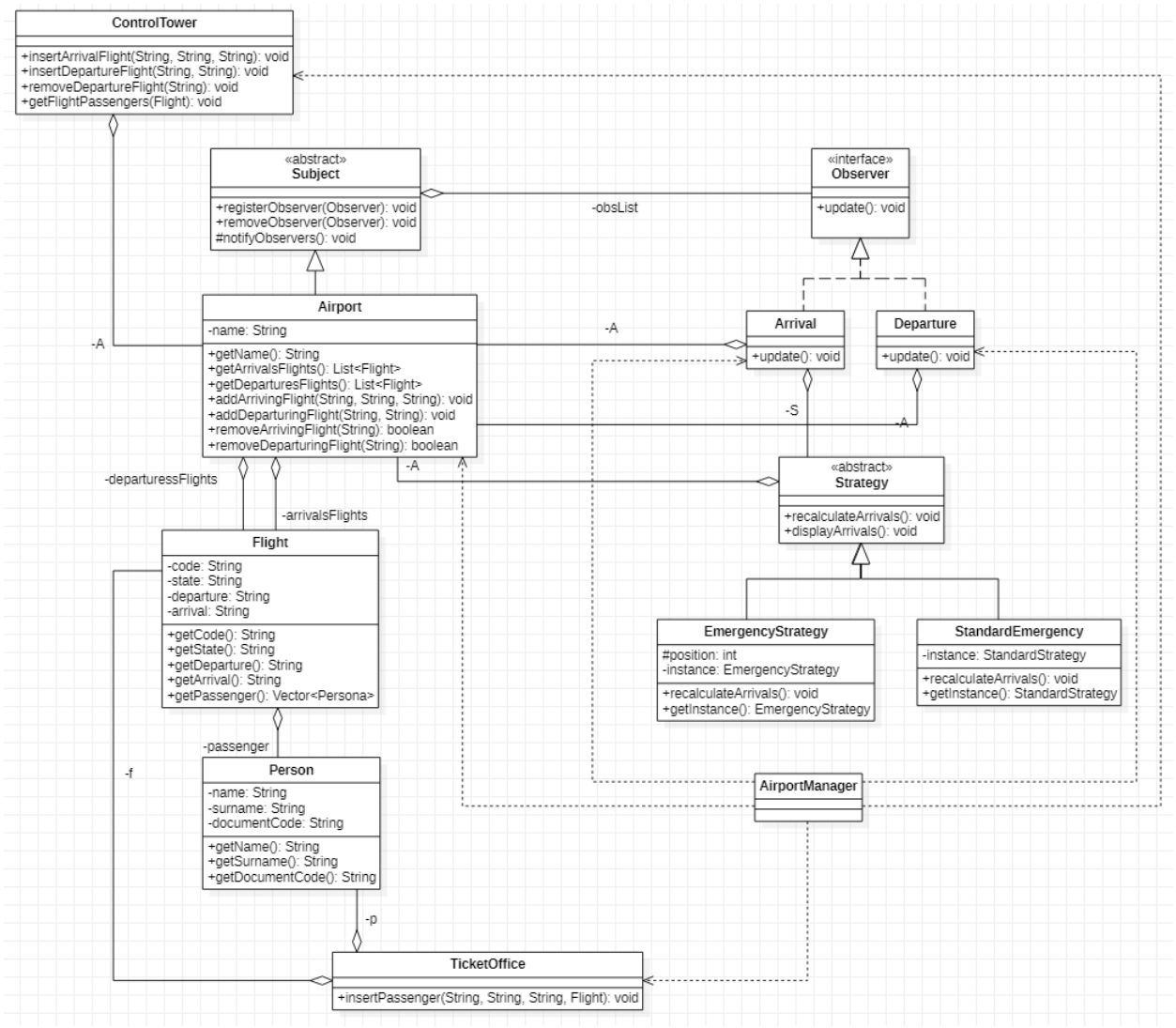
2.3 Observer and Strategy

Figura 3: Observer and Strategy combination UML class diagram



3 Soluzione

Figura 4: system implementation UML class diagram



3.1 Subject

Fornisce un'interfaccia per associare e dissociare oggetti *Observer*. Conosce i propri *Observer*.

```
import java.util.*;

public abstract class Subject {

    private Vector<Observer> obsList = new Vector<Observer>();

    public void registerObserver(Observer o) {
        obsList.add(o);

    }
```

```
11
12      public void removeObserver(Observer o) {
13          obsList.remove(o);
14
15      }
16
17      protected void notifyObservers() {
18          for(int i = 0; i < obsList.size(); i++)
19              obsList.elementAt(i).update();
20
21      }
22  }
```

Listing 1: Subject

3.2 Airport (ConcreteSubject)

Contiene gli stati (*arrivalsFlights* e *departuresFlights*) a cui gli oggetti *ConcreteObserver* (*Arrival, Departure*) sono interessati.
Inoltra una notifica ai suoi *Observer* quando il proprio stato si modifica.

```
1   import java.util.*;
2
3   public class Airport extends Subject {
4
5       private String name;
6       private List<Flight> arrivalsFlights = new ArrayList<Flight>();
7       private List<Flight> departuresFlights = new ArrayList<Flight>();
8
9       public Airport(String name) {
10
11          this.name = name;
12      }
13
14      public String getName() {
15
16          return name;
17      }
18
19      public List<Flight> getArrivalsFlights() {
20
21          return arrivalsFlights;
22      }
23
24      public List<Flight> getDeparturesFlights() {
25
26          return departuresFlights;
27      }
28
29      public void addArrivingFlight(String code, String state, String departure) {
30          this.getArrivalsFlights().add(new Flight(code, state, departure, null));
31          notifyObservers();
32      }
33
34      public void addDeparturingFlight(String code, String arrival) {
35          this.getDeparturesFlights().add(new Flight(code, null, null, arrival));
36          notifyObservers();
37      }
38
39      public boolean removeArrivingFlight(String code) {
40         boolean valid = this.getArrivalsFlights().removeIf(f -> f.getCode().equals
    (code));
41         notifyObservers();
42         return valid;
```

```
43      }
44
45      public boolean removeDeparturingFlight(String code) {
46          boolean valid = this.getDeparturesFlights().removeIf(f -> f.getCode().
    equals(code));
47          notifyObservers();
48          return valid;
49      }
50  }
```

Listing 2: Airport

3.3 Observer

Fornisce un'interfaccia di notifica per gli oggetti a cui devono essere notificati i cambiamenti nel *Subject*.

```
1  public interface  Observer {
2
3      public  void update();
4
5  }
```

Listing 3: Observer

3.4 Arrival (ConcreteObserver)

Memorizza un riferimento ad un oggetto *Airport* ed uno ad un oggetto *Strategy*. Implementa l'interfaccia di update dell'*Observer* per mantenere il proprio stato consistente con quello del *Subject*. In base allo stato dell'ultimo volo inserito per l'atterraggio (inizialmente inserito in coda) viene chiesta l'istanza di un oggetto *StandardStrategy* oppure *EmergencyStrategy* a cui viene data la responsabilità di ricalcolare la lista degli atterraggi. Questa classe quindi svolge anche il ruolo di *Context* per il pattern *Strategy*.

```
1  public class Arrival implements Observer {
2
3      private Airport A;
4      private Strategy S;
5
6      public Arrival(Airport A) {
7
8          this.A = A;
9          A.registerObserver(this);
10     }
11
12     @Override
13     public void update() {
14
15         if(A.getArrivalsFlights().size() == 0){
16             S = StandardStrategy.getInstance(A);
17             S.recalculateArrivals();
18             return;
19         }
20
21         if(A.getArrivalsFlights().get(A.getArrivalsFlights().size() - 1).getState
    ().equals("emergency")) {
22             S = EmergencyStrategy.getInstance(A);
23             S.recalculateArrivals();
24         }
25         else {
26             S = StandardStrategy.getInstance(A);
27             S.recalculateArrivals();
```

```
        }
    }

}
```

Listing 4: Arrival

3.5 Departure (ConcreteObserver)

Il compito di questo *ConcreteObserver*, poiché la lista delle partenze non segue alcuna particolare strategia di precedenza (è una coda FIFO), è quello di mantenere la lista delle partenze aggiornata e di pubblicarla sui tabelloni. Quindi in questo caso non c'è bisogno di tenere un riferimento ad un oggetto *Strategy* poiché non se ne fa uso.

```

public class Departure implements Observer {

    private Airport A;

    public Departure(Airport A) {

        this.A = A;
        A.registerObserver(this);
    }

    @Override
    public void update() {
        System.out.println("                    DEPARTURES");
        for(int i = 0; i < A.getDeparturesFlights().size(); i++) {
            System.out.print(" CODE: " + A.getDeparturesFlights().get(i).getCode
    () + " | ");
            System.out.println(" DESTINATION: " + A.getDeparturesFlights().get(i)
    .getArrival());
        }
        System.out.println("_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ "
    );
        System.out.println();
        System.out.println();
    }

}
```

Listing 5: Departure

3.6 Strategy

Dichiara un'interfaccia comune a tutti gli algoritmi supportati (*recalculateArrivals*). *Arrival* usa quest'interfaccia per invocare l'opportuno algoritmo definito da un *ConcreteStrategy* (*StandardStrategy*, *EmergencyStrategy*). Questa classe dispone anche il metodo comune a tutti gli algoritmi per visualizzare la lista degli arrivi.

```

public abstract class Strategy {

    public Airport A;

    public Strategy(Airport A) {

        this.A = A;
    }

    public abstract void recalculateArrivals();
```

```
12
13      public void displayArrivals() {
14
15          System.out.println("                        ARRIVALS");
16          for(int i = 0; i < A.getArrivalsFlights().size(); i++) {
17              System.out.print(" Code: " + A.getArrivalsFlights().get(i).getCode()
    + " | ");
18              System.out.print(" State: " + A.getArrivalsFlights().get(i).getState
    () + " | ");
19              System.out.println(" Provenance: " + A.getArrivalsFlights().get(i).
    getDeparture());
20          }
21          System.out.println("_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ "
    );
22      }
23
24  }
```

Listing 6: Strategy

3.7 EmergencyStrategy (ConcreteStrategyA)

Usa l'interfaccia *Strategy* per implementare l'algoritmo di calcolo della coda degli atterraggi dando priorità ai voli con stato *Emergency*.

```
1
2   public class EmergencyStrategy extends Strategy {
3
4       static int position = 0;
5       private static EmergencyStrategy instance = null;
6       private EmergencyStrategy(Airport A) {
7
8           super(A);
9       }
10
11      public static EmergencyStrategy getInstance(Airport A) {
12          if(instance == null) {
13              instance = new EmergencyStrategy(A);
14          }
15          return instance;
16
17      }
18
19      @Override
20      public void recalculateArrivals() {
21          position = (int) (A.getArrivalsFlights().stream().filter(fl -> fl.
    getState().equals("emergency")).count() - 1 );
22          Flight f = new Flight(null, null, null, null);
23          if(A.getArrivalsFlights().get(A.getArrivalsFlights().size() - 1).getState
    ().equals("emergency")) {
24              if (A.getArrivalsFlights().stream().allMatch(fl -> fl.getState().
    equals("emergency"))) {
25                  displayArrivals();
26                  return;
27              }
28              f = A.getArrivalsFlights().get(A.getArrivalsFlights().size() - 1);
29              A.getArrivalsFlights().remove(A.getArrivalsFlights().size() - 1);
30              A.getArrivalsFlights().add(position, f);
31          }
32          displayArrivals();
33      }
34
35  }
```

Listing 7: EmergencyStrategy

3.8 StandardStrategy (ConcreteStrategyB)

Poichè i voli standard non necessitano di essere riposizionati nella lista, questa classe implementa l'interfaccia semplicemente visualizzando la coda.

```
1
2  public class StandardStrategy extends Strategy {
3
4      private static StandardStrategy instance = null;
5      private StandardStrategy(Airport A) {
6          super(A);
7      }
8
9      public static StandardStrategy getInstance(Airport A){
10         if(instance == null) {
11             instance = new StandardStrategy(A);
12         }
13         return instance;
14     }
15     @Override
16     public void recalculateArrivals() {
17
18         displayArrivals();
19     }
20
21 }
```

Listing 8: StandardStrategy

3.9 ControlTower

Mantiene un riferimento al *Subject* ed implementa le operazioni di aggiunta e rimozione dal sistema dei voli in arrivo e dei voli in partenza facendo forwarding su *Airport*. Inoltre implementa i vari check i cui nel caso in cui l'operatore prova ad effettuare un operazione non autorizzata visualizzano un messaggio di errore (più avanti queste operazioni vengono elencate).

```
1
2  public class ControlTower {
3
4      private Airport A;
5
6      public ControlTower(Airport A) {
7
8          this.A = A;
9      }
10
11     public void insertArrivalFlight(String code, String state, String departure)
       {
12
13         boolean valid = true;
14         for(int i = 0; i < A.getArrivalsFlights().size(); i++) {
15             if(A.getArrivalsFlights().get(i).getCode().equals(code)) {
16                 System.err.println("Inserted code already in the system. Check
       again the arriving flight code!");
17                 valid = false;
18             }
19         }
20
21         if(valid)
22             A.addArrivingFlight(code, state, departure);
23
24     }
25
26     public void insertDepartureFlight(String code, String arrival) {
```

```
27
28          boolean valid = true;
29          for(int i = 0; i < A.getDeparturesFlights().size(); i++) {
30              if(A.getDeparturesFlights().get(i).getCode().equals(code)) {
31                  System.err.println("Inserted code already in the system. Check
    again the departuring flight code!");
32                  valid = false;
33              }
34          }
35          if(valid)
36              A.addDeparturingFlight(code, arrival);
37
38      }
39
40      public void removeArrivalFlight(String code) {
41
42          boolean valid = true;
43          if(A.getArrivalsFlights().size() != 0)
44             valid = A.removeArrivingFlight(code);
45          else
46              System.err.println("No arrivals programmed!");
47
48          if (!valid)
49              System.err.println("Inserted code not in the system!");
50
51      }
52
53      public void removeDepartureFlight(String code) {
54
55          boolean valid = true;
56          if(A.getDeparturesFlights().size() != 0)
57              valid = A.removeDeparturingFlight(code);
58          else
59              System.err.println("No departures programmed!");
60
61          if (!valid)
62              System.err.println("Inserted code not in the system!");
63
64      }
65
66      public void getFlightPassengers(Flight f) {
67
68          System.out.println("");
69          System.out.println("                                        PASSENGERS");
70          System.out.println(" _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    _ _ _ _ _ _ _ _ _ _ _ _ ");
71          for(int i = 0; i < f.getPassenger().size(); i++) {
72              System.out.print("| Name: " + f.getPassenger().get(i).getName() + " |
     ");
73              System.out.print("Surname: " + f.getPassenger().get(i).getSurname() +
     " | ");
74              System.out.print("Document ID: " + f.getPassenger().get(i).
    getDocumentCode() + " | ");
75              System.out.println("Flight code: " + f.getCode());
76              System.out.println("
    |-------------------------------------------------------------------------------|
    ");
77          }
78      }
79
80  }
```

Listing 9: ControlTower

Infine, le classi *Flight, Person e TicketOffice* non offrono particalari funzionalità, quindi non verranno mostrate.

3.10 Collaborazioni

ConcreteSubject (Airport) notifica ai propri *Observer (Arrival, Departure)* quando un volo viene aggiunto(rimosso) dalla lista di arrivo o partenza. Il *ConcreteObserver (Arrival)* viene notificato dell'aggiunta di un volo in arrivo e controlla lo stato di questo volo, scegliendo la strategia corretta da applicare per il ricalcolo della lista degli arrivi, poiché *Arrival* svolge anche il ruolo di *Context* per il pattern *Strategy*.

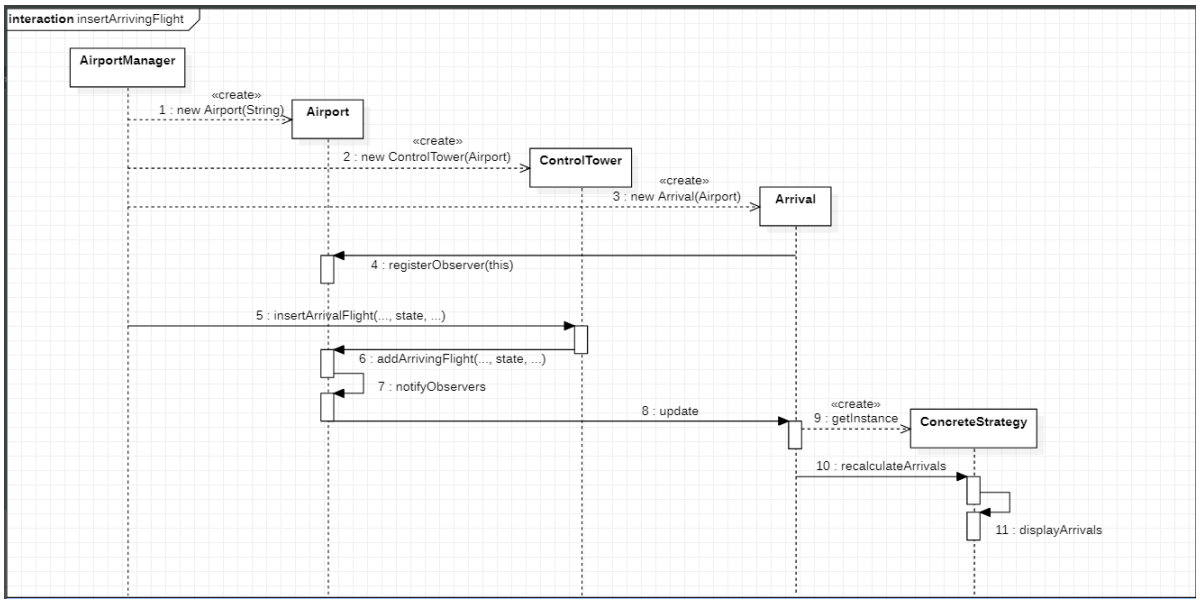
3.11 Conseguenze

Per modificare il modo in cui vengono gestite le code in base allo stato del volo, basta modificare la classe *ConcreteStrategy* corrispondente. Invece se si desidera aggiungere altri stati ai voli, se la strategia di gestione della coda d'atterraggio è già implementata da uno *ConcreteStrategy* esistente, bisogna modificare la classe *Arrival (ConcreteObserver)* che funge da *Context* per il pattern *Strategy*. Se invece dobbiamo aggiungere una nuova strategia, oltre alle modifiche alla classe *Arrival*, bisogna implementare un opportuno *ConcreteStrategy* che estenda la classe astratta *Strategy*. Quindi la classe *Arrival* deve sempre conoscere tutti i *ConcreteStrategy* implementati per poterli usare.

3.12 Sequence Diagram

Di seguito è mostrato il sequence diagram relativo all'inserimento di un volo in arrivo.

Figura 5: Insert Flight Sequence Diagram



4 Test

E' stata implementata una semplice interfaccia *(AirportManager)* con cui l'operatore della torre di controllo può inserire o rimuovere voli in arrivo e in partenza, inserire passeggeri per un determinato volo oppure richiedere la lista dei passeggeri di un determinato volo.

4.1 AirportManager

```
1
2  import java.util.*;
3  public class AirportManager {
4
5      public static void main(String[] args) {
6
7          Airport FlorenceAirport = new Airport("Florence");
8          ControlTower CT = new ControlTower(FlorenceAirport);
9          Arrival A = new Arrival(FlorenceAirport);
10         Departure D = new Departure(FlorenceAirport);
11         TicketOffice TO = new TicketOffice();
12
13         int choice;
14         String arrivalCode;
15         String departureCode;
16         String state;
17         String provenance;
18         String destination;
19         String departuringRemovalCode;
20         String arrivingRemovalCode;
21
22         String name;
23         String surname;
24         String documentId;
25         String flightCode;
26
27         for(;;) {
28             try
29             {
30                 Thread.sleep(50);
31             }
32             catch(InterruptedException ex)
33             {
34                 Thread.currentThread().interrupt();
35             }
36
37             System.out.println(" _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ ");
38             System.out.println("| 1 - Insert arrival             |");
39             System.out.println("| 2 - Insert departure           |");
40             System.out.println("| 3 - Remove arrival             |");
41             System.out.println("| 4 - Remove departure           |");
42             System.out.println("|-------------------------------|");
43             System.out.println("| 5 - Insert passengers          |");
44             System.out.println("| 6 - Print flight passengers    |");
45             System.out.println("|-------------------------------|");
46             System.out.println("| 7 - Exit                       |");
47             System.out.println("|_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _|");
48             System.out.println();
49
50             System.out.print("Choose an action from above: ");
51             Scanner S = new Scanner(System.in);
52             choice = S.nextInt();
53             System.out.println();
54
55             switch (choice) {
```

```
                case 1:

                    System.out.print("Insert flight code: ");
                    arrivalCode = S.next();
                    System.out.print("Insert state: ");
                    state = S.next();
                    System.out.print("Insert provenance: ");
                    provenance = S.next();

                    System.out.println();
                    System.out.println();

                    CT.insertArrivalFlight(arrivalCode, state, provenance);
                    break;

                case 2:

                    System.out.print("Insert flight code: ");
                    departureCode = S.next();
                    System.out.print("Insert destination: ");
                    destination = S.next();

                    System.out.println();
                    System.out.println();

                    CT.insertDepartureFlight(departureCode, destination);
                    break;

                case 3:

                    if(FlorenceAirport.getArrivalsFlights().size() == 0) {
                        System.err.println("No arrivals flights programmed!");
                        break;
                    }
                    System.out.println("Insert arriving flight code to remove: ")
    ;
                    arrivingRemovalCode = S.next();
                    CT.removeArrivalFlight(arrivingRemovalCode);
                    break;

                case 4:

                    if(FlorenceAirport.getDeparturesFlights().size() == 0) {
                        System.err.println("No departuring flights programmed!");
                        break;
                    }
                    System.out.println("Insert departuring flight code to remove:
     ");
                    departuringRemovalCode = S.next();
                    CT.removeDepartureFlight(departuringRemovalCode);
                    break;

                case 5:

                    if(FlorenceAirport.getDeparturesFlights().size() == 0) {
                        System.err.println("No departuring flights programmed!");
                        System.out.println("");
                        break;
                    }


                    System.out.print("Insert name: ");
                    name = S.next();
                    System.out.print("Insert surname: ");
                    surname = S.next();
```

```
                    System.out.print("Insert document Id: ");
                    documentId = S.next();
                    System.out.print("Insert departuring flight code: ");
                    flightCode = S.next();

                    int i = 0;
                    boolean found = false;
                    while(i < FlorenceAirport.getDeparturesFlights().size() && !found) {
                        if(FlorenceAirport.getDeparturesFlights().get(i).getCode().equals(flightCode)) {
                            found = true;
                            Flight f = FlorenceAirport.getDeparturesFlights().get(i);
                            TO.insertPassenger(name, surname, documentId, f);
                            CT.getFlightPassengers(FlorenceAirport.getDeparturesFlights().get(i));
                        }
                        i++;
                    }
                    if (!found)
                        System.err.println("Inserted departuring flight code does not exist!");
                    System.out.println("");

                    break;

                case 6:

                    if(FlorenceAirport.getDeparturesFlights().size() == 0) {
                        System.err.println("No departuring flights programmed!");
                        System.out.println("");
                        break;
                    }

                    System.out.print("Insert departuring flight code: ");
                    flightCode = S.next();

                    int j = 0;
                    boolean found2 = false;
                    while(j < FlorenceAirport.getDeparturesFlights().size() && !found2) {
                        if (FlorenceAirport.getDeparturesFlights().get(j).getCode().equals(flightCode)){
                            found2 = true;
                            Flight f = FlorenceAirport.getDeparturesFlights().get(j);

                            CT.getFlightPassengers(f);
                        }
                        j++;
                    }
                    if (!found2)
                        System.err.println("Inserted departuring flight code does not exist!");
                    System.out.println("");

                    break;

                case 7:

                    return;
            }
        }
    }

```

```
}
```

Listing 10: AirportManager

4.2 Output

All'avvio del programma ci si presenta una schermata come la seguente (per motivi di spazio, la prossima schermata non verrà più visualizzata nelle prossime immagini).

```
 - - - - - - - - - - - - - - - - -
| 1 - Insert arrival              |
| 2 - Insert departure            |
| 3 - Remove arrival              |
| 4 - Remove departure            |
|---------------------------------|
| 5 - Insert passengers           |
| 6 - Print flight passengers     |
|---------------------------------|
| 7 - Exit                        |
|_ - - - - - - - - - - - - - - - -|

Choose an action from above:
```

Listing 11: Flight Insert

Di seguito viene mostrato l'output delle seguenti azioni in ordine:

- inserimento di un volo in arrivo *standard*
- inserimento di un volo in arrivo *emergency*
- inserimento di un volo in partenza
- inserimento di un volo in arrivo *emergency*
- cancellazione di un volo in arrivo con stato *emergency*
- cancellazione di un volo in partenza.

```

Choose an action from above: 1

Insert flight code: 001
Insert state: standard
Insert provenance: rome


                     ARRIVALS
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                     DEPARTURES
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -



Choose an action from above: 1

Insert flight code: 002
Insert state: emergency
Insert provenance: florence


                     ARRIVALS
 Code: 002 |  State: emergency |  Provenance: florence
 Code: 001 |  State: standard |  Provenance: rome
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    DEPARTURES
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -



Choose an action from above: 2

Insert flight code: 005
Insert destination: venice


                     ARRIVALS
 Code: 002 |  State: emergency |  Provenance: florence
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    DEPARTURES
 CODE: 005 |  DESTINATION: venice
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -



Choose an action from above: 1

Insert flight code: 003
Insert state: emergency
Insert provenance: paris


                     ARRIVALS
 Code: 002 |  State: emergency |  Provenance: florence
 Code: 003 |  State: emergency |  Provenance: paris
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    DEPARTURES
 CODE: 005 |  DESTINATION: venice
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -



Choose an action from above: 3

Insert arriving flight code to remove:
002
                     ARRIVALS
 Code: 003 |  State: emergency |  Provenance: paris
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    DEPARTURES
 CODE: 005 |  DESTINATION: venice
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


Choose an action from above: 4

Insert departuring flight code to remove:
005
                     ARRIVALS
 Code: 003 |  State: emergency |  Provenance: paris
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                    DEPARTURES
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Listing 12: operations examples

Se l'utente prova ad effettuare delle azioni 'illegali', verrà visualizzato un opportuno messaggio di errore. Queste azioni consistono in:

- inserimento di un volo in arrivo avente codice uguale a quello di un altro volo in arrivo già presente nel sistema
- inserimento di un volo in partenza con codice uguale a quello di un altro volo in partenza già presente nel sistema
- cancellazione di un volo in arrivo (partenza) quando la lista degli arrivi (partenze) è vuota

```
1
2                       ARRIVALS
3  Code: 003 |  State: emergency |  Provenance: paris
4  Code: 001 |  State: standard |  Provenance: rome
5 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
6                       DEPARTURES
7 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
8
9
10 Choose an action from above: 1
11
12 Insert flight code: 001
13 Insert state: emergency
14 Insert provenance: oslo
15
16
17                      ARRIVALS
18  Code: 003 |  State: emergency |  Provenance: paris
19  Code: 001 |  State: standard |  Provenance: rome
20 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
21                      DEPARTURES
22 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
23
24 Inserted code already in the system. Check again the arriving flight code!
25
26
27 Choose an action from above: 2
28
29 Insert flight code: 005
30 Insert destination: barcelona
31
32
33                      ARRIVALS
34  Code: 003 |  State: emergency |  Provenance: paris
35  Code: 001 |  State: standard |  Provenance: rome
36 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
37                      DEPARTURES
38  CODE: 005 |  DESTINATION: barcelona
39 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
40
41 Choose an action from above: 2
42
43 Insert flight code: 005
44 Insert destination: london
45
46
47                      ARRIVALS
48  Code: 003 |  State: emergency |  Provenance: paris
49  Code: 001 |  State: standard |  Provenance: rome
50 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
51                      DEPARTURES
52  CODE: 005 |  DESTINATION: barcelona
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


Inserted code already in the system. Check again the departuring flight code!


Choose an action from above: 4

Insert departuring flight code to remove:
005
                      ARRIVALS
 Code: 003 |  State: emergency |  Provenance: paris
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                     DEPARTURES
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


Choose an action from above: 4

No departuring flights programmed!


Choose an action from above: 3

Insert arriving flight code to remove:
003
                      ARRIVALS
 Code: 001 |  State: standard |  Provenance: rome
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                     DEPARTURES
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Choose an action from above: 3

Insert arriving flight code to remove:
001
                      ARRIVALS
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                     DEPARTURES
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


Choose an action from above: 3

No arrivals flights programmed!
```

Listing 13: error messages