

AIR-AFPA

Activité-type 1 : Développer une application client/serveur.

Situation 4 Coder en Sql.

El Moussaoui Salim

Sommaire :

1.	Introduction.....	3
2.	Productions.....	3
2.1	Connexion Connexion la base de donnée avec la Classe ConnectDB (Singleton).....	3-4
2.2	Model d'une table en java : Flight (Model).....	5-10
2.3	La classe DAO	11
2.4	la classe FlightDAO.....	12-19
2.5	Test Unitaire de la Classe FlightDAO.....	20-23

1. Introduction

AIR-AFPA est une filiale de la société AFPA TRAVEL France créée en décembre 2012. Spécialisée dans le marché low-cost du transport de passagers, elle a pour but de se faire une place sur ce marché. En effet, d'ici 2020 le nombre de vol low-cost devrait s'accroître de 50 %. Jusqu'en mi-2013, la filiale avait recours aux logiciels de sa maison-mère, mais ses besoins changeants elle souhaite développer son propre système d'information, c'est dans cette optique que le service des vols demande à présent la réalisation d'une application de gestion des vols.

2. Productions

2.1 Connexion la base de donnée avec la Classe ConnectDB (Singleton)

```
13  /**
14  *
15  * @author Salim El Moussaoui <salim.elmoussaoui.afpa2017@gmail.com>
16  */
17  public class ConnectDB {
18
19      // Assign values for connection db mysql
20      private static final String DB_URL = "jdbc:mysql://localhost:3306/airafpa";
21      private static final String DB_JDBC_DRIVER = "com.mysql.jdbc.Driver";
22      private static final String DB_USER = "afpa";
23      private static final String DB_PASSWORD = "afpa";
24
25      private Connection cn = null;
26
27      private ConnectDB() {
28          // import jdbc mysql
29          try {
30              Class.forName(ConnectDB.DB_JDBC_DRIVER);
31
32          } catch (ClassNotFoundException ex) {
33              ex.printStackTrace();
34              System.exit(1);
35          }
36      }
37
38
39      /**
40      * Singleton for connexion DB
41      *
42      * @return
43      */
44      public static ConnectDB getInstance() {
45          return ConnexionBdHolder.INSTANCE;
46      }
47
48      /**
49      * Assign Instance class ConnectDB
50      */
51      private static class ConnexionBdHolder {
52
53          private static final ConnectDB INSTANCE = new ConnectDB();
54      }
55  }
```

```

56  /**
57   * check connect DB
58   *
59   * @return
60   */
61  public boolean connect() {
62      // connect db is null
63      if (this.cn == null) {
64          // connect db
65          try {
66              this.cn = DriverManager.getConnection(ConnectDB.DB_URL, ConnectDB.DB_USER, ConnectDB.DB_PASSWORD);
67          } catch (SQLException ex) {
68              ex.printStackTrace();
69              return false;
70          }
71      } else {
72          // check connect db
73          try {
74              Statement st = this.cn.createStatement();
75              String requete = "SELECT 1";
76              st.executeQuery(requete);
77          } catch (SQLException ex) {
78              try {
79                  this.cn = DriverManager.getConnection(ConnectDB.DB_URL, ConnectDB.DB_USER, ConnectDB.DB_PASSWORD);
80              } catch (SQLException ex1) {
81                  ex1.printStackTrace();
82                  return false;
83              }
84          }
85      }
86  }
87
88  return true;
89  }
90
91  /**
92   * connect db
93   *
94   * @return connect db
95   */
96  public Connection getConnectionManager() {
97      return this.cn;
98  }
99  }
100

```

La classe ConnectDB permet de connecter a la base de donnée mysql par un jdbc de java. Il est instancié qu'une seule fois pour tout la durée de l'utilisation donc une utilise la pattern Singleton pour cela.

2.2 Model d'une table en java : Flight (Model)

```
11  *
12  * @author Salim El Moussaoui <salim.elmoussaoui.afpa2017@gmail.com>
13  */
14  public class Flight {
15      // name columns in table flights
16      private long id;
17      private String departing_aita;
18      private String arrival_aita;
19      private String departing_hour;
20      private int duration;
21      private double price;
22      private long id_pilot;
23      private long id_copilot;
24      private long id_staff1;
25      private long id_staff2;
26      private long id_staff3;
27      private boolean planned;
28
29      // this construct create flight empty
30      public Flight() {
31          this.id = 0;
32          this.departing_aita = "";
33          this.arrival_aita = "";
34          this.departing_hour = "";
35          this.duration = 0;
36          this.price = 0;
37          this.id_pilot = 0;
38          this.id_copilot = 0;
39          this.id_staff1 = 0;
40          this.id_staff2 = 0;
41          this.id_staff3 = 0;
42          this.planned = false;
43      }
```

```

44 // this construct create flight full
45 public Flight(
46     long id, String departing_aita, String arrival_aita,
47     String departing_hour, int duration, double price, long id_pilot,
48     long id_copilot, long id_staff1, long id_staff2, long id_staff3,
49     boolean planned) {
50     this.id = id;
51     this.departing_aita = departing_aita;
52     this.arrival_aita = arrival_aita;
53     this.departing_hour = departing_hour;
54     this.duration = duration;
55     this.price = price;
56     this.id_pilot = id_pilot;
57     this.id_copilot = id_copilot;
58     this.id_staff1 = id_staff1;
59     this.id_staff2 = id_staff2;
60     this.id_staff3 = id_staff3;
61     this.planned = planned;
62 }
63
64 public long getId() {
65     return id;
66 }
67
68 public String getDeparting_aita() {
69     return departing_aita;
70 }
71
72 public String getArrival_aita() {
73     return arrival_aita;
74 }
75
76 public String getDeparting_hour() {
77     return departing_hour;
78 }
79
80 public int getDuration() {
81     return duration;
82 }
83
84 public double getPrice() {
85     return price;
86 }
87

```

```

88 public long getId_pilot() {
89     return id_pilot;
90 }
91
92 public long getId_copilot() {
93     return id_copilot;
94 }
95
96 public long getId_staff1() {
97     return id_staff1;
98 }
99
100 public long getId_staff2() {
101     return id_staff2;
102 }
103
104 public long getId_staff3() {
105     return id_staff3;
106 }
107
108 public boolean getPlanned() {
109     return planned;
110 }
111
112 public void setId(long id) {
113     this.id = id;
114 }
115 public void setDeparting_aita(String departing_aita) {
116     this.departing_aita = departing_aita;
117 }
118
119 public void setArrival_aita(String arrival_aita) {
120     this.arrival_aita = arrival_aita;
121 }
122
123 public void setDeparting_hour(String departing_hour) {
124     this.departing_hour = departing_hour;
125 }
126
127 public void setDuration(int duration) {
128     this.duration = duration;
129 }
130
131 public void setPrice(double price) {
132     this.price = price;
133 }

```

```

135 public void setId_pilot(long id_pilot) {
136     this.id_pilot = id_pilot;
137 }
138
139 public void setId_copilot(long id_copilot) {
140     this.id_copilot = id_copilot;
141 }
142
143 public void setId_staff1(long id_staff1) {
144     this.id_staff1 = id_staff1;
145 }
146
147 public void setId_staff2(long id_staff2) {
148     this.id_staff2 = id_staff2;
149 }
150
151 public void setId_staff3(long id_staff3) {
152     this.id_staff3 = id_staff3;
153 }
154
155 public void setPlanned(boolean planned) {
156     this.planned = planned;
157 }
158
159 @Override
160 public String toString() {
161     return "Flight{" + "id=" + id + ", departing_aita=" + departing_aita
162         + ", arrival_aita=" + arrival_aita + ", departing_hour=" +
163         departing_hour + ", duration=" + duration + ", price=" + price
164         + ", id_pilot=" + id_pilot + ", id_copilot=" + id_copilot
165         + ", id_staff1=" + id_staff1 + ", id_staff2=" + id_staff2
166         + ", id_staff3=" + id_staff3 + ", planned=" + planned + '}';
167 }
168

```



```

169 @Override
170 public int hashCode() {
171     int hash = 7;
172     hash = 29 * hash + (int) (this.id ^ (this.id >>> 32));
173     hash = 29 * hash + Objects.hashCode(this.departing_aita);
174     hash = 29 * hash + Objects.hashCode(this.arrival_aita);
175     hash = 29 * hash + Objects.hashCode(this.departing_hour);
176     hash = 29 * hash + this.duration;
177     hash = 29 * hash + (int) (Double.doubleToLongBits(this.price) ^
178         (Double.doubleToLongBits(this.price) >>> 32));
179     hash = 29 * hash + (int) (this.id_pilot ^ (this.id_pilot >>> 32));
180     hash = 29 * hash + (int) (this.id_copilot ^ (this.id_copilot >>> 32));
181     hash = 29 * hash + (int) (this.id_staff1 ^ (this.id_staff1 >>> 32));
182     hash = 29 * hash + (int) (this.id_staff2 ^ (this.id_staff2 >>> 32));
183     hash = 29 * hash + (int) (this.id_staff3 ^ (this.id_staff3 >>> 32));
184     hash = 29 * hash + (this.planned ? 1 : 0);
185     return hash;
186 }
187
188 @Override
189 public boolean equals(Object obj) {
190     if (this == obj) {
191         return true;
192     }
193     if (obj == null) {
194         return false;
195     }
196     if (getClass() != obj.getClass()) {
197         return false;
198     }
199     final Flight other = (Flight) obj;
200     if (this.id != other.id) {
201         return false;
202     }
203     if (this.duration != other.duration) {
204         return false;
205     }
206     if (Double.doubleToLongBits(this.price) != Double.doubleToLongBits(other.price)) {
207         return false;
208     }
209     if (this.id_pilot != other.id_pilot) {
210         return false;
211     }
212     if (this.id_copilot != other.id_copilot) {
213         return false;
214     }

```

```

215         if (this.id_staff1 != other.id_staff1) {
216             return false;
217         }
218         if (this.id_staff2 != other.id_staff2) {
219             return false;
220         }
221         if (this.id_staff3 != other.id_staff3) {
222             return false;
223         }
224         if (this.planned != other.planned) {
225             return false;
226         }
227         if (!Objects.equals(this.departing_aita, other.departing_aita)) {
228             return false;
229         }
230         if (!Objects.equals(this.arrival_aita, other.arrival_aita)) {
231             return false;
232         }
233         if (!Objects.equals(this.departing_hour, other.departing_hour)) {
234             return false;
235         }
236         return true;
237     }
238
239
240
241 }

```

La classe Fligh est un model de la table flights dans la base de donnée, il ya tous les champs avec leur type. On a aussi les assesseurs et mutateurs qui permettrons de l'utiliser à l'extérieur.

2.3 La classe DAO

```
11  /**
12  *
13  * @author Salim El Moussaoui <salim.elmoussaoui.afpa2017@gmail.com>
14  */
15  public abstract class DAO<T,S> {
16      protected ConnectDB bddmanager = null;
17      public DAO() {
18          this.bddmanager = ConnectDB.getInstance();
19      }
20      /**
21       * Insert row in table
22       * @param obj
23       * @return if success insert then return true else false
24       */
25      public abstract T create(T obj);
26      /**
27       * Update row in table
28       * @param obj
29       * @return if success update then return true else false
30       */
31      public abstract boolean update(T obj);
32      /**
33       * Delete row in table
34       * @param obj
35       * @return if success delete then return true else false
36       */
37      public abstract boolean delete(S primary_key);
38      /**
39       * get all row in table
40       * @return list collection all row
41       */
42      public abstract ArrayList<T> getAll();
43      /**
44       * get row
45       * @param id
46       * @return collection row
47       */
48      public abstract T find(S primary_key);
49
50  }
```

Le DOA est un design pattern permet de faire la distinction entre les données auxquelles qu'on souhaite accéder et de la manière de les stockées. On utilise la méthode CRUD :

C (Créate) : on crée un vol

R (Read) : on sélection un ou plusieurs vol

U (Update) : on met a jour le vol

D (Delete) : on supprimer le vole

Lorsque qu'on crée un model DAO et qu'on étend la classe DAO , la classe fille doit obligatoirement les méthodes mère car la Classe DAO est abstract, il y a des généricités <T> pour le model et <S> pour le type de clé primaire du model.

2.4 la classe FlightDAO

```
18  * @author Salim El Moussaoui <salim.elmoussaoui.afpa2017@gmail.com>
19  */
20  public class FlightDAO extends DAO<Flight, Long> {
21
22      public FlightDAO() {
23          super();
24      }
25
26      /**
27       * Create flight
28       *
29       * @param flight
30       * @return flight object
31       */
32      @Override
33      public Flight create(Flight flight) {
34
35          Flight flightCreate = new Flight();
36          if (this.bddmanager.connect()) {
37
38              try {
39
40                  // create requete
41                  String requete = "INSERT INTO flights ( "
42                      + "departing_aita,\n"
43                      + " arrival_aita,\n"
44                      + " departing_hour,\n"
45                      + " duration,\n"
46                      + " price,\n"
47                      + " id_pilot,\n"
48                      + " id_copilot,\n"
49                      + " id_staff1,\n"
50                      + " id_staff2,\n"
51                      + " id_staff3,\n"
52                      + " planned"
53                      + ") VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
54
55                  // prepared requete and get return generated key
56                  PreparedStatement pst = this.bddmanager.getConnectionManager()
57                      .prepareStatement(requete, Statement.RETURN_GENERATED_KEYS);
58                  // insert value in requete
59                  pst.setString(1, flight.getDeparting_aita());
60                  pst.setString(2, flight.getArrival_aita());
61                  pst.setString(3, flight.getDeparting_hour());
62                  pst.setInt(4, flight.getDuration());
63                  pst.setDouble(5, flight.getPrice());
```

```

63 // if it's null
64 if (flight.getId_pilot() == 0) {
65     pst.setNull(6, Types.BIGINT);
66 } else {
67     pst.setLong(6, flight.getId_pilot());
68 }
69 if (Long.valueOf(flight.getId_copilot()) == 0) {
70
71     pst.setNull(7, Types.BIGINT);
72 } else {
73     pst.setLong(7, flight.getId_copilot());
74 }
75 if (flight.getId_staff1() == 0) {
76     pst.setNull(8, Types.BIGINT);
77 } else {
78     pst.setLong(8, flight.getId_staff1());
79 }
80
81 if (flight.getId_staff2() == 0) {
82     pst.setNull(9, Types.BIGINT);
83 } else {
84     pst.setLong(9, flight.getId_staff2());
85 }
86 if (flight.getId_staff3() == 0) {
87     pst.setNull(10, Types.BIGINT);
88 } else {
89     pst.setLong(10, flight.getId_staff3());
90 }
91 pst.setBoolean(11, flight.getPlanned());
92
93 // excute insert row in table
94 int insert = pst.executeUpdate();
95 // if insert in table
96 if (insert != 0) {
97     // get generate key
98     ResultSet id_increment = pst.getGeneratedKeys();
99     // if it's generate key
100     if (id_increment.next()) {
101         // assign key in fligh object
102         flight.setId(id_increment.getInt(1));
103         // assign object fligh in object return
104         flightCreate = flight;
105     }
106
107 }
108 } catch (SQLException ex) {

```

```

109     ex.printStackTrace();
110     return flightCreate;
111 }
112
113 } else {
114     return flightCreate;
115 }
116
117 return flightCreate;
118 }

```

```

119  /**
120   * Update flight
121   *
122   * @param flight
123   * @return true is update flight, false isn't update
124   */
125  @Override
126  public boolean update(Flight flight) {
127      boolean success = false;
128
129      if (this.bddmanager.connect()) {
130
131          try {
132
133              // create requete
134              String requete = "Update flights set departing_aita = ?,\n"
135                  + " arrival_aita = ?,\n"
136                  + " departing_hour = ?,\n"
137                  + " duration = ?,\n"
138                  + " price = ?,\n"
139                  + " id_pilot = ?,\n"
140                  + " id_copilot = ?,\n"
141                  + " id_staff1 = ?,\n"
142                  + " id_staff2 = ?,\n"
143                  + " id_staff3 = ?,\n"
144                  + " planned = ?"
145                  + " WHERE id = ?";
146
147              // prepared requete
148              PreparedStatement pst = this.bddmanager
149                  .getConnectionManager().prepareStatement(requete);
149
150              // insert value in requete
151              pst.setString(1, flight.getDeparting_aita());
152              pst.setString(2, flight.getArrival_aita());
153              pst.setString(3, flight.getDeparting_hour());
154              pst.setInt(4, flight.getDuration());
155              pst.setDouble(5, flight.getPrice());
156              // if it's null
157              if (flight.getId_pilot() == 0) {
158                  pst.setNull(6, Types.BIGINT);
159
160              } else {
161                  pst.setLong(6, flight.getId_pilot());
162              }
163              if (Long.valueOf(flight.getId_copilot()) == 0) {
164
165                  pst.setNull(7, Types.BIGINT);
166              } else {

```

```

166         pst.setLong(7, flight.getId_copilot());
167     }
168     if (flight.getId_staff1() == 0) {
169         pst.setNull(8, Types.BIGINT);
170     } else {
171         pst.setLong(8, flight.getId_staff1());
172     }
173
174     if (flight.getId_staff2() == 0) {
175         pst.setNull(9, Types.BIGINT);
176     } else {
177         pst.setLong(9, flight.getId_staff2());
178     }
179     if (flight.getId_staff3() == 0) {
180         pst.setNull(10, Types.BIGINT);
181     } else {
182         pst.setLong(10, flight.getId_staff3());
183     }
184     pst.setBoolean(11, flight.getPlanned());
185     pst.setLong(12, flight.getId());
186     // excute update row in table
187     int insert = pst.executeUpdate();
188     // if insert in table
189     if (insert != 0) {
190         success = true;
191     }
192     } catch (SQLException ex) {
193         ex.printStackTrace();
194         return success;
195     }
196
197     } else {
198         return success;
199     }
200     return success;
201 }

```

```

203  /**
204  * delete flight
205  *
206  * @param primary_key
207  * @return true is delete flight, false isn't delete
208  */
209  @Override
210  public boolean delete(Long primary_key) {
211      boolean success = false;
212
213      if (this.bddmanager.connect()) {
214
215          try {
216
217              // create requete
218              String requete = "DELETE FROM flights WHERE id = ?";
219              // prepared requete
220              PreparedStatement pst = this.bddmanager.getConnectionManager()
221                  .prepareStatement(requete);
222              // insert value in requete
223              pst.setLong(1, primary_key);
224              // excute delete row in table
225              int insert = pst.executeUpdate();
226              // if delete in table
227              if (insert != 0) {
228                  success = true;
229              }
230          } catch (SQLException ex) {
231              ex.printStackTrace();
232              return success;
233          }
234
235      } else {
236          return success;
237      }
238      return success;
239  }
240  }

```



```

241  /**
242   * get all flights
243   *
244   * @return all flights
245   */
246  @Override
247  public ArrayList getAll() {
248      // create array list airport empty
249      ArrayList<Flight> listFlight = new ArrayList<>();
250      if (this.bddmanager.connect()) {
251
252          try {
253              // create statement
254              Statement st = this.bddmanager
255                  .getConnectionManager()
256                  .createStatement();
257              // create requete
258              String requete = "SELECT * FROM flights";
259              // excute requete
260              ResultSet rs = st.executeQuery(requete);
261              // insert all airports in array object airport
262              while (rs.next()) {
263                  Flight el = new Flight(
264                      rs.getInt("id"),
265                      rs.getString("departing_aita"),
266                      rs.getString("Arrival_aita"),
267                      rs.getString("departing_hour"),
268                      rs.getInt("duration"),
269                      rs.getDouble("price"),
270                      rs.getInt("id_pilot"),
271                      rs.getInt("id_copilot"),
272                      rs.getInt("id_staff1"),
273                      rs.getInt("id_staff2"),
274                      rs.getInt("id_staff3"),
275                      rs.getBoolean("planned")
276                  );
277                  listFlight.add(el);
278              }
279
280              } catch (SQLException ex) {
281                  ex.printStackTrace();
282                  return listFlight;
283              }
284
285      } else {
286          return listFlight;
287      }
288
289      return listFlight;
290  }
291
292

```

```

292  /**
293   * find flight
294   *
295   * @param primary_key
296   * @return flight
297   */
298  @Override
299  public Flight find(Long primary_key) {
300      // create array airport empty
301      Flight flight = new Flight();
302      //check if connect db
303      if (this.bddmanager.connect()) {
304
305          try {
306              // create statement for find
307              Statement st = this.bddmanager.getConnectionManager()
308                  .createStatement();
309              // create requete add primary key
310              String requete = "SELECT * FROM flights WHERE id = " + primary_key;
311              // excute requete
312              ResultSet rs = st.executeQuery(requete);
313              // if result is full
314              if (rs.next()) {
315                  // insert airports in object
316                  flight.setId(rs.getLong("id"));
317                  flight.setDeparting_aita(rs.getString("departing_aita"));
318                  flight.setArrival_aita(rs.getString("Arrival_aita"));
319                  flight.setDeparting_hour(rs.getString("departing_hour"));
320                  flight.setDuration(rs.getInt("duration"));
321                  flight.setPrice(rs.getDouble("price"));
322                  flight.setId_pilot(rs.getLong("id_pilot"));
323                  flight.setId_copilot(rs.getLong("id_copilot"));
324                  flight.setId_staff1(rs.getLong("id_staff1"));
325                  flight.setId_staff2(rs.getLong("id_staff2"));
326                  flight.setId_staff3(rs.getLong("id_staff3"));
327                  flight.setPlanned(rs.getBoolean("planned"));
328              }
329
330              } catch (SQLException ex) {
331                  ex.printStackTrace();
332                  return flight;
333              }
334
335          } else {
336              return flight;
337          }
338
339      return flight;
340  }
341

```

```

342  /**
343   * It checks if the object flight is filled or empty
344   *
345   * @param flight
346   * @return false is empty and true is full
347   */
348  public boolean isValid(Flight flight) {
349      boolean isValid = true;
350
351      // if id is empty
352      if (flight.getId() == 0) {
353          isValid = false;
354      }
355
356      return isValid;
357  }
358
359
360
361  }

```

La Classe FlightDAO comme son nom l'indique on étend la classe DAO en utilisant le model Flight comme généricité <T> et le la clé primaire de type Long car coté db est un type BIGINT.

Le CRUD il y est, on juste ajouter une méthode isValid pour vérifier si le vol est valide on retourne TRUE ou FALSE en vérifiant ID si il est égal a 0 ca veux dire que le vol n'a pas été enregistré dans la base de donnée.

Test Unitaire de la Classe FlightDAO

```
18  /**
19  *
20  * @author Salim El Moussaoui <salim.elmoussaoui.afpa2017@gmail.com>
21  */
22  public class FlightDAOTest {
23
24      /**
25       * Test of create method, of class FlightDAO.
26       */
27      @Test
28      public void testCreate() {
29          System.out.println("create");
30          // create object flight
31          Flight flight = new Flight(0, "MRS", "FNJ",
32                                   "2017-03-20 05:30:00.0", 400, 275.75, 1, 2, 3, 4, 5, true);
33          FlightDAO flightDAO = new FlightDAO();
34          // find flight create
35          Flight expResult = flightDAO.find(flight.getId());
36
37          // if find aita is empty or null
38          if (!flightDAO.isValid(expResult)) {
39              // insert flight in table
40              Flight result = flightDAO.create(flight);
41              // find flight
42              expResult = flightDAO.find(flight.getId());
43              // delete flight test
44              flightDAO.delete(result.getId());
45              assertEquals(expResult, result);
46          }
47      }
48  }
49  }
```

```

51  /**
52   * Test of update method, of class FlightDAO.
53   */
54  @Test
55  public void testUpdate() {
56      System.out.println("update");
57      // create object flight
58      Flight flightInsert = new Flight(0, "FNJ", "LAS", "2017-03-20 05:30:00",
59                                     400, 478.75, 0, 0, 0, 0, 0, false);
60      // instatied flightDAO
61      FlightDAO flightDAO = new FlightDAO();
62      // find flight create
63      Flight findFlight = flightDAO.find(flightInsert.getId());
64      // if find flight is empty
65      if (!flightDAO.isValid(findFlight)) {
66          // Insert flight in table
67          Flight resultFlight = flightDAO.create(flightInsert);
68          // find flight
69          findFlight = flightDAO.find(resultFlight.getId());
70      }
71      // modify flight
72      findFlight.setDeparting_aita("MRS");
73      findFlight.setArrival_aita("FNJ");
74      findFlight.setDeparting_hour("2017-03-21 05:30:00.0");
75      findFlight.setDuration(400);
76      findFlight.setPrice(278.75);
77      findFlight.setId_pilot(1);
78      findFlight.setId_copilot(0);
79      findFlight.setId_staff1(0);
80      findFlight.setId_staff2(0);
81      findFlight.setId_staff3(5);
82      findFlight.setPlanned(true);
83
84      // update flight
85      boolean result = flightDAO.update(findFlight);
86      boolean expResult = true;
87      // delete flight test
88      flightDAO.delete(findFlight.getId());
89      assertEquals(expResult, result);
90  }

```

```

92  /**
93   * Test of delete method, of class FlightDAO.
94   */
95  @Test
96  public void testDelete() {
97      System.out.println("delete");
98      FlightDAO flightDAO = new FlightDAO();
99
100     // create object flight
101     Flight flightInsert = new Flight(0, "TLS", "MRS", "2017-03-20 05:30:00",
102                                     250, 175.75, 1, 2, 3, 4, 5, true);
103     // find flight create
104     Flight findFlight = flightDAO.find(flightInsert.getId());
105
106     // if find flight is empty
107     if (!flightDAO.isValid(findFlight)) {
108         // Insert flight in table
109         Flight resultFlight = flightDAO.create(flightInsert);
110         // find flight
111         findFlight = flightDAO.find(resultFlight.getId());
112         // delete flight
113         boolean result = flightDAO.delete(findFlight.getId());
114         boolean expResult = true;
115         assertEquals(expResult, result);
116     }
117 }

```

```

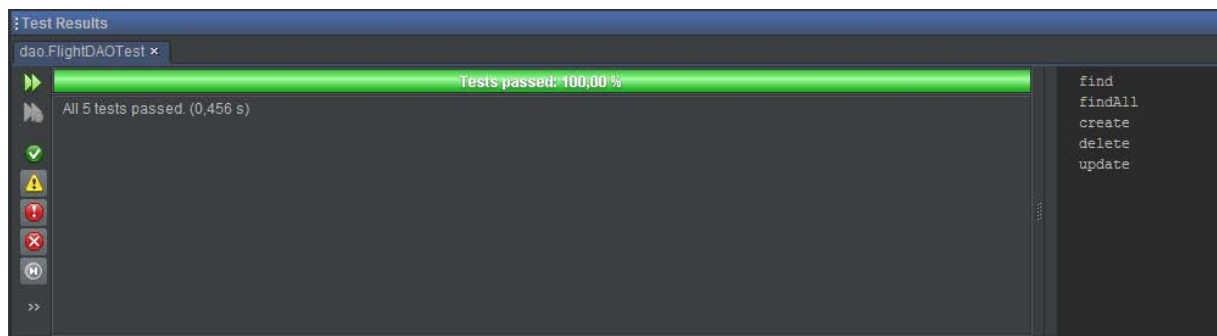
119 /**
120  * Test of findAll method, of class FlightDAO.
121  */
122  @Test
123  public void testGetdAll() {
124      System.out.println("findAll");
125      FlightDAO flightDAO = new FlightDAO();
126      ArrayList<Flight> arrayFlight = flightDAO.getAll();
127      String expResult = "";
128      String result = "";
129      for (int i = 0; i < arrayFlight.size(); i++) {
130          expResult += flightDAO.find(arrayFlight.get(i).getId());
131          result += arrayFlight.get(i);
132      }
133      assertEquals(expResult, result);
134  }
135 }
136

```

```

137  /**
138   * Test of find method, of class FlightDAO.
139   */
140  @Test
141  public void testFind() {
142      System.out.println("find");
143      FlightDAO flightDAO = new FlightDAO();
144      // assign primary key
145      long primary_key = 5;
146      // assign result exemple
147      String expResult = "Flight(id=5, departing_aita=DXB, arrival_aita=ICN, "
148          + "departing_hour=2017-05-01 08:00:00.0, duration=420, "
149          + "price=358.67, id_pilot=1, id_copilot=0, id_staff1=4, "
150          + "id_staff2=0, id_staff3=0, planned=false)";
151
152      // find flight
153      String result = flightDAO.find(primary_key).toString();
154
155      assertEquals(expResult, result);
156  }
157  }

```



On fait un test unitaire sur la classe FlightDAO avec JUNIT, on test tout les méthodes.