# Spécification, conception et vérification

## Projet gestion des vols d'une compagnie aérienne

| Avion | VOL | Dest | Date | Type | Site | Prénom | Nom | Fonction |
|---|---|---|---|---|---|---|---|---|
| 13562 | AF347 | Londres | 11/10/06 | A320 | Orly | Corinne | Lancel | Pilote |
| 13562 | AF347 | Londres | 11/10/06 | A320 | Orly | Amy | Bosch | Copilote |
| 13562 | AF347 | Londres | 11/10/06 | A320 | Orly | Maureen | Gates | PNC |
| 13562 | AF347 | Londres | 11/10/06 | A320 | Orly | Richard | Tata | PNC |
| 13562 | AF347 | Londres | 11/10/06 | A320 | Orly | Ben | Gamma | PNC |
| 32156 | AF545 | New-York | 12/10/06 | B747 | Roissy | Jacques | Vlisside | Pilote |
| 32156 | AF545 | New-York | 12/10/06 | B747 | Roissy | Jean-Louis | Dupont | Copilote |
| 32156 | AF545 | New-York | 12/10/06 | B747 | Roissy | Ségolène | Orval | PNC |
| 32156 | AF545 | New-York | 12/10/06 | B747 | Roissy | François | Saadi | PNC |
| 32156 | AF545 | New-York | 12/10/06 | B747 | Roissy | Nicolas | Harper | PNC |

FIGURE 1 – Extrait du tableau de service des vols AF347 et AF545 sur une période

## Introduction

Au cours de ce semestre nous avons étudié les différents aspects de la conception logicielle. La conception d'un projet passe notamment par la création des différents tests permettant de valider les implémentations. Ces tests prennent toute leur importance s dans avec des méthodes de conception comme Test Driven Development qui préconisent de réaliser l'ensemble des méthodes de tests avant l'implémentation des classes fonctionnelle afin de s'assurer de développer toutes les fonctionnalités nécessaires sans implémentations superflues.

Le but de ce projet était donc de mieux nous permettre d'appréhender ces concepts vus en cours par le biais de la création de classes de tests. Nous avons utilisé le framework JUnit qui est la la partie Java du framework XUnit permettant de mettre en place simplement et rapidement des tests automatiques.

Dans ce rapport nous répondrons donc aux différentes questions du sujet en commentant les différentes implémentations que nous avons mises en place.

## Question 1: Attributs de la classe Equipage

Les attributs de la classe Equipage sont les suivants:

```
private Boolean auMin;
private Boolean auMax;
private Vector<PNC> pnc;
private Pilote pilote;
private Copilote copilote;
private Vol vol;
```

## Question 2: implémentation de Equipage.contains()

L'implémentation de la méthode est la suivante:

```
/* question 2 */
public Boolean contains (Personne membre) {
    if (membre == null) return false;
    if (pnc != null && pnc.contains(membre) ) return true;
    if (pilote != null && pilote.equals(membre) ) return true;
    if (copilote != null && copilote.equals(membre)) return true;
    return false;
}
```

La méthode vérifie que les différents objets manipulés sont initialisés afin de ne pas lever d'exception du type NullPointerException. De plus la méthode fait également appel à la méthode contains de la classe Vector pour l'objet pnc.

## Question 3: méthode Equipage.toString()

L'implémentation est la suivante:

```
@Override
public String toString() {
    String base = vol.getAvion().getReference() + " " + vol.getNumero() + " "
                + vol.getDestination() + " " + vol.getDate() + " "
                + vol.getAvion().getType().getNom() + " " + vol.getSite() + " ";

    String s = "Equipage: " + base + "\n";
    if (pilote != null) s += "pilote\t\t" + pilote + "\n";
    if (copilote != null) s += "copilote\t" + copilote + "\n";
    if (pnc != null) s += "PNCs\t\t" + pnc + "\n";

    return s;
}
```

Ce qui produit la sortie suivante pour le tableau d'exemple demandé:

```
Equipage: 13562 AF347 Londres 11/10/06 A320 Orly
pilote          Lancel Corinne
copilote        Bosch Amy
PNCs            [Gates Maureen, Tata Richard, Gamma Ben]

Equipage: 32156 AF545 New-York 12/10/06 B747 Roissy
pilote          Vlisside Jacques
copilote        Dupont Jean-Louis
PNCs            [Orval Segolene, Saadi Francois, Harper Nicolas]
```

## Question 4: création du tableau exemple

```
11      public static void question4() {
12
13          try {
14              /* creation de la date */
15              String dep1 = "11/10/06";
16              String dep2 = "12/10/06";
17
18              /* creation des types d avions */
19              TypeAvion type1 = new TypeAvion("A320", 2, 3);
20              TypeAvion type2 = new TypeAvion("B747", 3, 4);
21
22              /* creation des avions */
23              Avion avion1 = new Avion(type1, "13562");
24              Avion avion2 = new Avion(type2, "32156");
25
26              /* creation des personnes */
27              Pilote pilote1 = new Pilote("Lancel", "Corinne");
28              Pilote pilote2 = new Pilote("Vlisside", "Jacques");
29
30              Copilote copilote1 = new Copilote("Bosch", "Amy");
31              Copilote copilote2 = new Copilote("Dupont", "Jean-Louis");
32
33              PNC pnc11 = new PNC("Gates", "Maureen");
34              PNC pnc12 = new PNC("Tata", "Richard");
35              PNC pnc13 = new PNC("Gamma", "Ben");
36              Vector<PNC> pnc1 = new Vector<PNC>();
37              pnc1.add(pnc11);
38              pnc1.add(pnc12);
39              pnc1.add(pnc13);
40
41              PNC pnc21 = new PNC("Orval", "Segolene");
42              PNC pnc22 = new PNC("Saadi", "Francois");
43              PNC pnc23 = new PNC("Harper", "Nicolas");
44              Vector<PNC> pnc2 = new Vector<PNC>();
45              pnc2.add(pnc21);
46              pnc2.add(pnc22);
47              pnc2.add(pnc23);
48
49              /* creation du vol */
50              Vol vol1 = new Vol("AF347", "Orly", "Londres", dep1,
    avion1);
51              Vol vol2 = new Vol("AF545", "Roissy", "New-York", dep2,
    avion2);
52
53              /* creation de l equipage */
54              Equipage equipage1 = new Equipage(vol1);
55              equipage1.setPilote(pilote1);
56              equipage1.setCopilote(copilote1);
57              equipage1.setPnc(pnc1);
58              equipage1.setAuMax(true);
59              equipage1.setAuMin(false);
60
61              Equipage equipage2 = new Equipage(vol2);
62              equipage2.setPilote(pilote2);
63              equipage2.setCopilote(copilote2);
64              equipage2.setPnc(pnc2);
65              equipage2.setAuMax(false);
66              equipage2.setAuMin(true);
67
68              System.out.println(equipage1);
69              System.out.println(equipage2);
70          } catch (InvariantBroken e) {
71              System.out.println("Viol dun invaraint: " + e.
    getMessage());
72          } catch (Exception e){
73              System.out.println("Autre exception: " + e.getMessage());
74          }
75          return;
76      }
77
```

## Question 5: Tests unitaires

Les tests unitaires (T.U.) ont pour but de tester les fonctionnalités d'une classe de manière isolée sans prendre en compte les interactions avec d'autres parties de l'application. Pour mener a bien ces T.U. nous avons donc implémenté une classe contenant différents objets à tester:

```
public class Question5 {

    Pilote p;
    Copilote cp;
    PNC pncs[];
    Equipage e;
    TypeAvion ta;
    Avion a;
    Vol v;
```

Deux méthodes annotées @Before et @After sont également présentes pour remettre ces objets dans un état stable entre chaque tests:

```java
@Before
public void setUp() throws Exception {
    ta = new TypeAvion("A320", 2, 3);
    a = new Avion(ta, "13562");
    v = new Vol("AF437", "Orly", "Londres", "11/11/2011", a);
    pncs = new PNC[3];

    e = new Equipage(v);
    v.setEquipage(e);

    p = new Pilote("Corinne", "Lancel");
    p.addQualification(ta);

    cp = new Copilote("Amy", "Bosch");
    cp.addQualification(ta);

    pncs[0] = new PNC("Maureen", "Gates");
    pncs[0].addQualification(ta);

    pncs[1] = new PNC("Richard", "Tata");
    pncs[1].addQualification(ta);

    pncs[2] = new PNC("Ben", "Gamma");
    pncs[2].addQualification(ta);
}
```

La méthode setUp est utilisée pour initialiser les différents objets.

```java
@After
public void tearDown() throws Exception {
    p = null;
    cp = null;
    ta = null;
    a = null;
    v = null;
    pncs = null;
    e = null;
}
```

La méthode tearDown permet quand a elle de détruire les objets après chacun des tests.

Par soucis de concision nous ne détaillerons pas l'intégralités des méthodes de tests développées. Intéressons nous à la première:

```java
/* verifie que la methode d'ajout dun pilote leve une exception si le pilote a deja ete ajoute */
@Test (expected = EquipageException.class)
public void addPilote_piloteAlreadySet() throws EquipageException, InvariantBroken {
    System.out.println("vol - methode addPilote en situation derreur");
    v.addPilote(p);
    v.addPilote(new Pilote("Herve", "Dumont"));
}
```

Un vol ne peut pas se voir attribuer deux pilotes, ainsi si un pilote a déjà été affecté à un vol la méthode Vol.addPilote() lève une exception de type EquipageException. Nous précisons donc à la méthode de test qu'une telle exception devrait être attrapée au cours du test.

La méthode ajoute simplement deux pilotes au vol. Le test est validé quand le deuxième appel à addPilote lève une exception.

La deuxième méthode de test présentée est la suivante:

```
/* verifie le bon fonctionnement de la methode contains() */
@Test
public void contains() throws Exception {
    System.out.println("equipage - methode contains");
    v.addPilote(p);
    v.addCopilote(cp);
    v.addPNC(pncs[0]);
    v.addPNC(pncs[1]);
    v.addPNC(pncs[2]);

    assertTrue("Pilote present dans lequipage", e.contains(p));
    assertTrue("Copilote present dans lequipage", e.contains(cp));
    assertTrue("PNC present dans lequipage", e.contains(pncs[2]));
    assertFalse("Pilote non present dans lequipage", e.contains(new Pilote("Dumont", "Herve")));
    assertFalse("Copilote non present dans lequipage", e.contains(new Copilote("Dumont", "Herve")));
    assertFalse("PNC non present dans lequipage", e.contains(new PNC("Dumont", "Herve")));
    assertFalse("Recherche dun membre null", e.contains(null));
}
```

Ici nous testons le bon fonctionnement de la méthode Equipage.contains(). Aucune exception n'est attendue, nous utilisons donc les méthodes Assert.assertTrue et Assert.assertFalse. Ces deux méthodes prennent en paramètre une chaine de caractères ainsi qu'une expression booléenne. Le test est validé si l'expression passée à assertFalse est fausse et si celle passée à assertTrue est vraie.

Ici nous testons les valeurs de retour de la méthode contains() avec un comportement défini.

## Question 6: Tests de validation

Les tests de validations permettent de valider les jeux de données passée aux classes de notre application. Cette fois encore des méthodes annotées @Before et @After nous permettent de réinitialiser nos données entre les différents tests. Etudions le premier test présenté:

```
/* test l'allocation du pilote */
@Test
public void vol_addPilote() throws Exception {
    System.out.println("Vol.addPilote => e.getPilote");
    v.addPilote(p);
    assertEquals(p, e.getPilote());
}
```

Cette fois un objet Pilot p est initialisé dans la méthode setUp(), dans notre méthode de test nous affectons le pilote au vol puis nous utilisons la méthode assertEquals(). Cette méthode prend en paramètre deux objets et valide le test si les deux objets sont identiques. Nous initialisons donc un objet Pilote p dans la méthode setUp(), notre méthode de test ajoute ensuite ce pilote au vol et nous testons que cette allocation à bien eu lieu en comparant le pilote du vol avec celui utilisé par la méthode addPilote().

Le deuxième test présenté est le suivant:

```java
/* test l'invariant sur les nb de PNC dans un type davion */
@Test
public void typeAvion_Invariant() throws Exception {
    System.out.println("TypeAvion - invariant sur nbPNCmin et nbPNCmax");
    assertTrue(ta.invariant());

    ta.setNbPNCmax(1);
    assertFalse(ta.invariant());

    ta.setNbPNCmax(-1);
    assertFalse(ta.invariant());

    ta.setNbPNCmax(5);
    ta.setNbPNCmin(-1);
    assertFalse(ta.invariant());
}
```

Nous testons cette fois le respect de l'invariant défini dans la classe TypeAvion. Cet invariant spécifie que la variable nbPNCmax doit toujours être supérieure ou égale à la variable nbPNCmin. De plus ces deux valeurs doivent toujours être positives ou nulles puisqu'elle représentent un nombre de membres d'équipage nécessaires au bon déroulement d'un vol.

Une méthode invariant() à donc été implémentée dans la classe TypeAvion renvoyant faux si l'invariant n'est plus respecté et vrai s'il l'es toujours. Afin de mener à bien notre test, nous affectons différentes valeurs à ces variables et nous testons le retour d'invariant() avec les méthodes Assert.assertFalse() et Assert.assertTrue() vues précédemment.

## Question 7: Tests d'intégration

Nous allons baser nos tests d'intégration sur l'opération "affecter un PNC à un vol". Le but des tests d'intégration est de vérifier les bon fonctionnement des interactions entre les classes. Pour ce faire nous avons mis en place des tests basés sur la méthode setUp() suivante:

```java
30      @Before
31      public void setUp() throws Exception {
32          ta = new TypeAvion("A320", 2, 3);
33          a = new Avion(ta, "13562");
34          v = new Vol("AF437", "Orly", "Londres", "11/11/2011", a);
35          pncs = new Vector<PNC>();
36
37          e = new Equipage(v);
38          v.setEquipage(e);
39
40          p = new Pilote("Corinne", "Lancel");
41          p.addQualification(ta);
42
43          cp = new Copilote("Amy", "Bosch");
44          cp.addQualification(ta);
45
46          pncs.add(new PNC("Maureen", "Gates"));
47          pncs.get(0).addQualification(ta);
48
49          pncs.add(new PNC("Richard", "Tata"));
50          pncs.get(1).addQualification(ta);
51
52          pncs.add(new PNC("Ben", "Gamma"));
53          pncs.get(2).addQualification(ta);
54
55          e.setPnc(pncs);
56          e.addPilote(p);
57          e.addCopilote(cp);
58      }
```

Pour notre premier test intéressons d'abord aux lignes 47, 50 et 53. Nous faisons appel à la méthode addQualification() de la classe personne. Cette méthode prend en paramètre un objet de type TypeAvion, son rôle est donc d'ajouter le type d'avion dans les qualifications de la personne mais également d'ajouter la personne dans la liste des qualifiés du type avion. Nous pouvons donc valider ce fonctionnement avec le test suivant:

```
/* verifie que les qualifications sont bien allouées */
@Test
public void qualifications() {
    System.out.println("Personne.addQualification => TypeAvion contient les qualifies");
    assertTrue(p.getQualifications().contains(ta));
    assertTrue(ta.getQualifies().contains(p));
    assertTrue(cp.getQualifications().contains(ta));
    assertTrue(ta.getQualifies().contains(cp));
}
```

Notre deuxième test permet quand à lui de vérifier qu'un objet vol ne peut pas ajouter de membre à son équipage lorsque celui ci indique qu'il est déjà au maximum de sa capacité:

```
/* verifie qu'on ne peut pas ajouter un pnc a un equipage plein */
@Test (expected = EquipageException.class)
public void affecterPNC_equipePleine() throws Exception {
    System.out.println("Equipage.equipageAuComplet => Vol.peutVoler");
    e.setAuMax(true);
    v.addPNC(new PNC("Herve", "Dumont"));
}
```

## Conclusion

Ce projet à donc été l'occasion de mettre en place différents types de tests afin de mieux cerner leur utilité mais il nous a également permis de manipuler JUnit et de nous convaincre de la puissance et de la simplicité de cet outil.

Cependant il est à noter que les tests que nous avons mis en place étaient intéressants d'un point de vue pédagogique mais ne seraient pas suffisants pour valider dans son ensemble une application de manière rigoureuse.