

# TP1 – Rapport

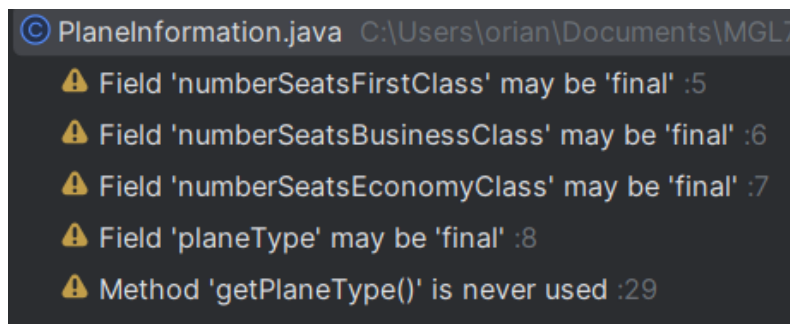
## 1. Amélioration de la qualité du code

Pour ce TP, j'ai fait une analyse statique du code dans un premier temps en utilisant SonarQube et l'analyse de l'IDE IntelliJ IDEA.

J'ai ensuite créé un fichier checkstyle.xml qui répertorie quelques règles qui ne sont pas vérifiées par les outils cités plus haut. Par exemple, une des règles permet de vérifier, grâce à une regexp, qu'il n'existe pas d'espace superflu dans le code.

Par exemple, il existe beaucoup d'imports inutilisés dans de nombreuses classes. Il y a également certains champs ou certaines méthodes inutilisés. Certains champs devraient être « final » car leurs valeurs ne doivent pas être modifiées après le constructeur.

























Un bon exemple est sur la classe PlaneInformation, où le type d'avion et le nombre de places ne peut pas changer : les champs devraient être « final ». Il y avait également une méthode et un champ jamais utilisés (getPlaneType() et planeType) que j'ai donc supprimés.



Il reste encore des warnings sur certaines classes : SonarQube me conseille d'utiliser un logger au lieu des System.out.print. Cependant, je trouve que ce n'est pas utile car dans notre cas, les print font partie du fonctionnement de l'application. Les loggers sont utilisés pour des print de debug, qui peuvent alors être activés ou désactivés.

## 2. Couverture des tests unitaires

Afin de vérifier la couverture de mes tests, j'utilise le plugin maven JaCoCo. Ce plugin permet d'obtenir un rapport détaillé de la couverture des tests. Ce rapport est au format html et peut donc être consulté sur un navigateur. Voici un exemple de rapport obtenu :

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ca.uqam.mgl7230.tp1.model.flight		65 %		n/a	1	8	4	18	1	8	1	2
ca.uqam.mgl7230.tp1.service		98 %		96 %	1	28	1	76	0	11	0	3
ca.uqam.mgl7230.tp1		98 %		100 %	1	7	1	37	1	4	0	2
ca.uqam.mgl7230.tp1.adapter.flight		100 %		n/a	0	2	0	16	0	2	0	1
ca.uqam.mgl7230.tp1.service.prompt		100 %		100 %	0	12	0	41	0	5	0	2
ca.uqam.mgl7230.tp1.model.passenger		100 %		n/a	0	13	0	28	0	13	0	6
ca.uqam.mgl7230.tp1.adapter.plane		100 %		n/a	0	5	0	13	0	5	0	1
ca.uqam.mgl7230.tp1.model.plane		100 %		n/a	0	5	0	13	0	5	0	2
ca.uqam.mgl7230.tp1.utils		100 %		n/a	0	2	0	8	0	2	0	1
ca.uqam.mgl7230.tp1.adapter.persist		100 %		n/a	0	2	0	14	0	2	0	1
ca.uqam.mgl7230.tp1.config		100 %		n/a	0	2	0	5	0	2	0	1
ca.uqam.mgl7230.tp1.exception		100 %		n/a	0	1	0	1	0	1	0	1
Total	27 of 1 114	97 %	1 of 48	97 %	3	87	6	270	2	60	1	23

Avec ce plugin, j'ai mis en place un « check » qui vérifie lors de l'exécution de la commande « mvn test » que la couverture des tests est de 100%

### 3. Correction des bugs trouvés

#### 1 – BookingService.book()

Un test a permis de détecter un bug sur cette méthode. Lorsqu'un passager business est ajouté à un vol où il n'y a plus de place en business, ce passager est ajouté dans l'avion en tant que business et n'enlève aucune place economy.

Voici le test qui m'a permis de trouver ce bug, et son erreur associée :

```
@Test new *
void bookBusinessClassSeats() {
    //Given
    given(passenger.getType()).willReturn(PassengerClass.BUSINESS_CLASS);
    int businessClassSeats = flightPassengerService.numberOfBusinessClassSeatsAvailable();
    int economyClassSeats = flightPassengerService.numberOfEconomyClassSeatsAvailable();

    //When
    for (int i = 0; i < businessClassSeats; i++) {
        bookingService.book(passenger, flightInformation);
        assertThat(flightPassengerService.numberOfBusinessClassSeatsAvailable()).isEqualTo(expected: businessClassSeats - i - 1);
    }

    bookingService.book(passenger, flightInformation);

    //Then
    assertThat(flightPassengerService.numberOfBusinessClassSeatsAvailable()).isZero();
    assertThat(flightPassengerService.numberOfEconomyClassSeatsAvailable()).isEqualTo(expected: economyClassSeats - 1);
}
```

```
✘ Tests failed: 1 of 1 test - 3 sec 793 ms
java.net.SocketException: Socket server was running, but
Passenger added successfully
Passenger added successfully
Passenger added successfully
Passenger added successfully
Passenger added successfully
Business Class is Full. Trying Economy...
Passenger added successfully

org.opentest4j.AssertionFailedError:
expected: 11
but was: 12
Expected :11
Actual   :12
```

Une place en classe Economy aurait du être enlevée, passant de 12 à 11 mais ce n'est pas le cas, le nombre est resté à 12.

L'erreur vient du fait qu'un passager de type Economy n'est pas créé lorsqu'il n'y a plus de place en business. Je corrige donc ce bug :

```
public void book(Passenger passenger, FlightInformation flightInformation) { 6 usages Oriane Doudet *
    Map<PassengerKeyConstants, Object> passengerDataMap = getPassengerKeyConstantsObjectMap(passenger);
    if (flightPassengerService.numberOfTotalSeatsAvailable() != 0) {
        if (PassengerClass.FIRST_CLASS.name().equalsIgnoreCase(passenger.getType().name())) {
            if (flightPassengerService.numberOfFirstClassSeatsAvailable() == 0) {
                System.out.println("First Class is Full. Trying Business...");
                passengerDataMap.put(PassengerKeyConstants.PASSENGER_CLASS, PassengerClass.BUSINESS_CLASS);
                passenger = passengerService.createPassenger(flightInformation, passengerDataMap);
            }
        }
        if (PassengerClass.BUSINESS_CLASS.name().equalsIgnoreCase(passenger.getType().name())) {
            if (flightPassengerService.numberOfBusinessClassSeatsAvailable() == 0) {
                System.out.println("Business Class is Full. Trying Economy..");
                passengerDataMap.put(PassengerKeyConstants.PASSENGER_CLASS, PassengerClass.ECONOMY_CLASS);
                passenger = passengerService.createPassenger(flightInformation, passengerDataMap);
            }
        }
        if (PassengerClass.ECONOMY_CLASS.name().equalsIgnoreCase(passenger.getType().name())) {
            if (flightPassengerService.numberOfEconomyClassSeatsAvailable() == 0) {
```

Avec cette correction, le test passe !

#### 4. Amélioration du code

J'ai amélioré la méthode `getPassengerData(Scanner scanner)` de la classe `PassengerPromptService`. Cette méthode permet de récupérer les informations du passager entrées par l'utilisateur. J'ai remarqué que l'algorithme ne laisse pas de place à l'erreur humaine : si l'utilisateur n'entre pas un nombre pour l'âge, le programme s'arrête en renvoyant une erreur. Il se passe la même chose avec la classe du passager : si elle n'existe pas, le programme s'arrête.

J'ai donc modifié cela en programmant en TDD : j'ai d'abord implémenté des tests qui essayent des mauvaises valeurs avant d'entrer les bonnes. J'ai ensuite implémenté cette nouvelle fonctionnalité à l'aide des erreurs provoquées par les tests.

Voici donc la nouvelle implémentation pour récupérer la classe du passager en prenant en compte les éventuelles erreurs humaines.

```
validEntry = false;
PassengerClass passengerClass = null;
while (!validEntry) {
    try {
        passengerClass = getPassengerClass(scanner.nextLine());
        validEntry = true;
    } catch (PassengerTypeNotFoundException e) {
        System.out.println("Passenger type does not exist");
    }
}
```

J'ai également décidé de modifier le fonctionnement de `savePassengerInFlight`, qui inscrivait dans le fichier les passagers n'ayant pas pu être ajoutés au vol. De plus, la classe des passagers n'était pas modifiée en fonction de la place qu'ils obtenaient : si un passager souhaitant être `FIRST_CLASS` obtenait une place en `ECONOMY`, il était inscrit en tant que `FIRST_CLASS` dans le fichier `passengerData.csv`

J'ai modifié ce comportement pour que seuls les passagers ayant obtenu une place dans l'avion y soient inscrits avec la classe correspondante au siège qu'ils ont obtenu.

Voici les modifications

Dans `BookingService.book()` :

```
public Passenger book(Passenger passenger, FlightInformation flightInformation) {
```

La méthode retourne les données du passager ajouté (le type du passager a été modifié si besoin). Si le passager n'a pas pu être ajouté au vol, la méthode retourne `null`.

Dans `Application.main` :

```
passenger = bookingService.book(passenger, result.flightCatalog().getFlightInformation(result.flightNumber()));  
if (passenger != null) {  
    result.savePassengerInFlight().save(result.file(), passenger, result.flightNumber());  
}
```