

03/12/2020

Conception et programmation objet avancées

TD4



Michael McCarthy, Baptiste Herr et Samuel Husson
DUT INFORMATIQUE | UNIVERSITÉ CÔTE D'AZUR

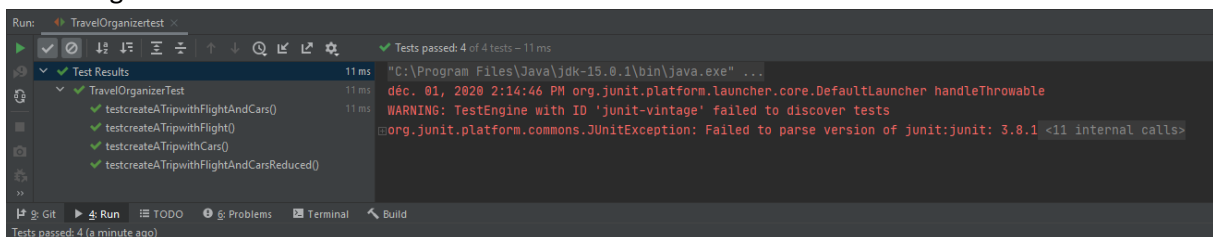
Sommaire

Vérifications principales.....	2
Diagramme de classe	3
Couverture des tests	4
SonarLint	5
Compte rendu	6
1.1. Avez-vous bien respecté les exigences de l'ingénieur ?	6
1.2. Couverture de tests :	6
1.2.1. Quels sont les principaux points faibles de votre couverture de test?	6
1.2.2. Quels points amélioreriez-vous si vous en aviez le temps?	6
1.3. Expliquez comment vous avez cherché à respecter le principe ouvert/fermé, par exemple expliquez l'utilisation des interfaces, etc.	6
1.4. Si vous pensez que votre code respecte d'autres principes expliquez.....	6
1.5. Commentez le diagnostic de SonarLint.....	6
Instructions de remises.....	7

Vérifications principales

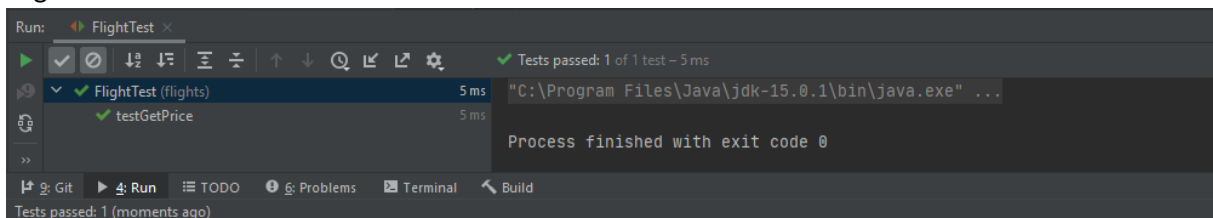
1. Tous nos codes compilent.
2. Tous nos tests passent. Nous avons remarqué des warnings à propos de problèmes de versions JUnit. Nous pensons que c'est dû à certains imports qui sont formatés pour des anciennes versions de JUnit alors que nous utilisons une version plus récente dans nos Path. En fonction de nos machines chacun avait des warnings différents c'est pour cela que nous avons préféré ne rien toucher. Voici nos résultats :

TravelOrganizerTest :



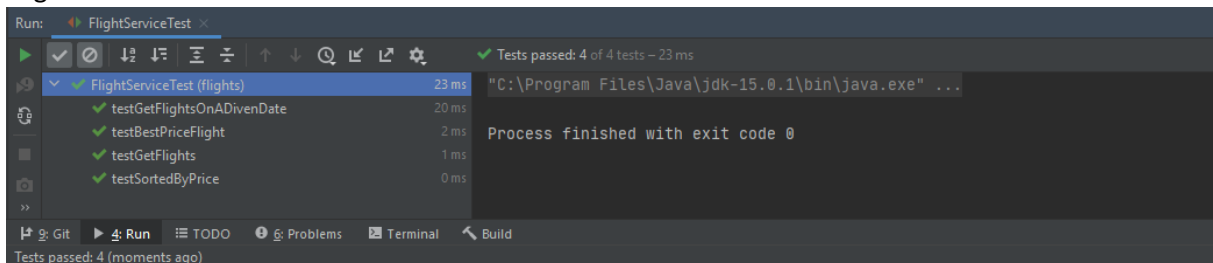
```
Run: TravelOrganizerTest
Tests passed: 4 of 4 tests - 11 ms
Test Results
  TravelOrganizerTest 11 ms
    testcreateATripwithFlightAndCars() 11 ms
    testcreateATripwithFlight()
    testcreateATripwithCars()
    testcreateATripwithFlightAndCarsReduced()
Process finished with exit code 0
Tests passed: 4 (a minute ago)
```

FlightTest :




```
Run: FlightTest
Tests passed: 1 of 1 test - 5 ms
FlightTest (flights) 5 ms
  testGetPrice 5 ms
Process finished with exit code 0
Tests passed: 1 (moments ago)
```

FlightServiceTest :



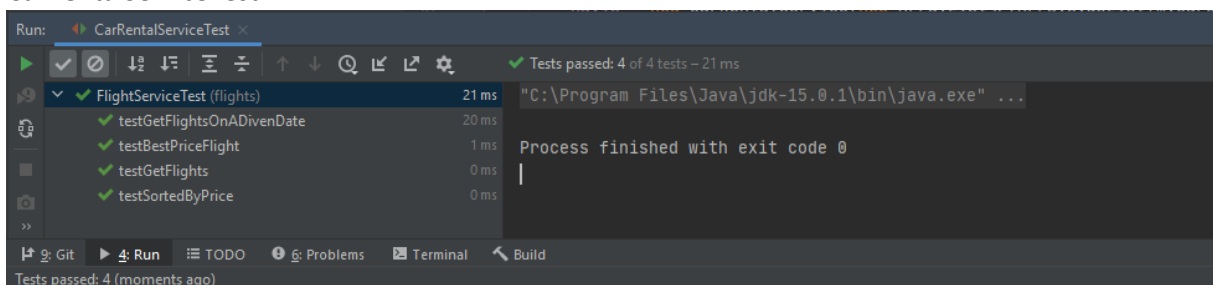
```
Run: FlightServiceTest
Tests passed: 4 of 4 tests - 23 ms
FlightServiceTest (flights) 23 ms
  testGetFlightsOnADivenDate 20 ms
  testBestPriceFlight 2 ms
  testGetFlights 1 ms
  testSortedByPrice 0 ms
Process finished with exit code 0
Tests passed: 4 (moments ago)
```

CarRentalTest :



```
Run: CarRentalTest
Tests passed: 3 of 3 tests - 8 ms
Test Results
  CarRentalTest 8 ms
    testGetPrice() 7 ms
    testCarRentalCreation()
    testCarRentalAvailability() 1 ms
Process finished with exit code 0
Tests passed: 3 (moments ago)
```

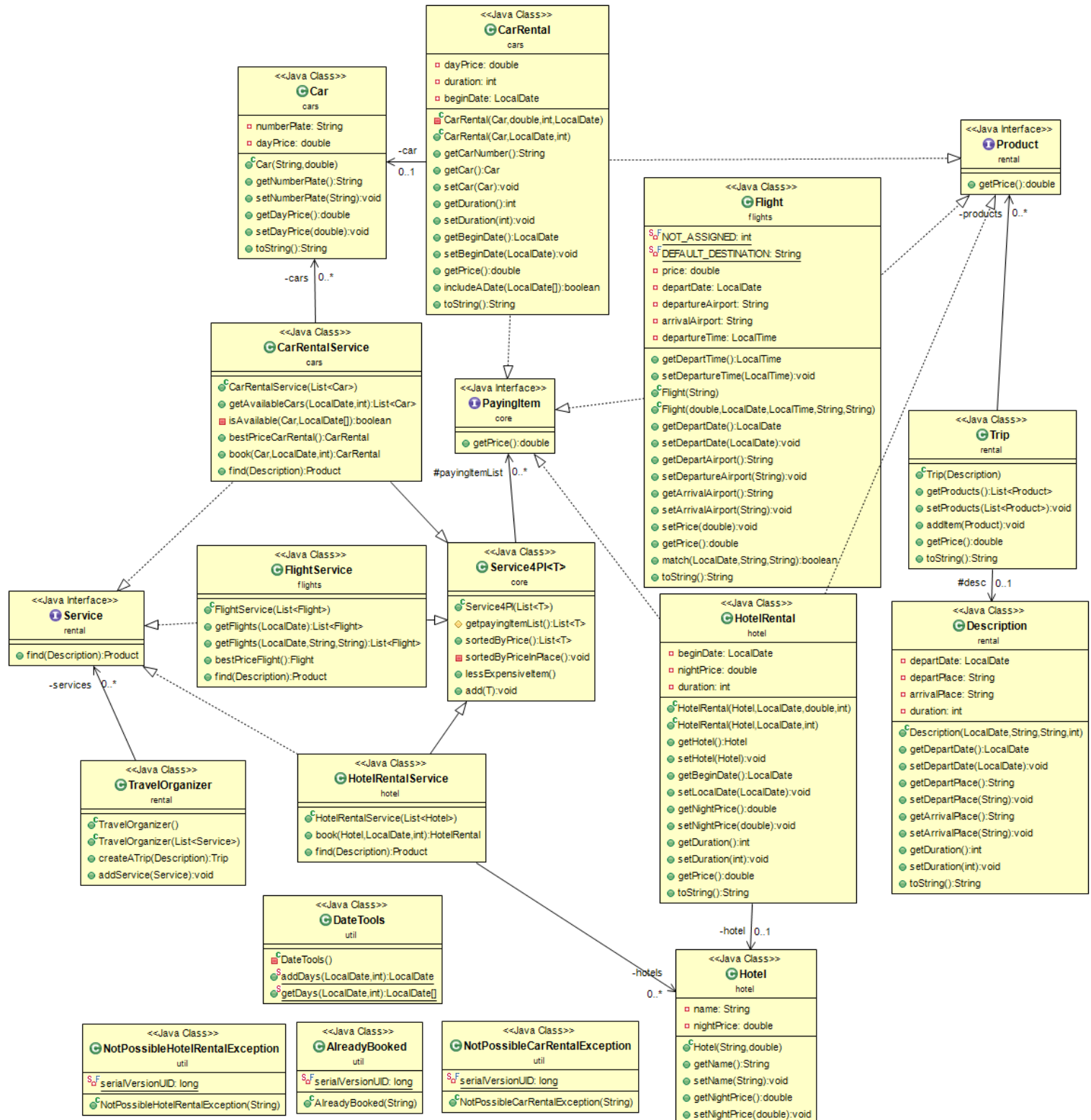
CarRentalServiceTest :



```
Run: CarRentalServiceTest
Tests passed: 4 of 4 tests - 21 ms
FlightServiceTest (flights) 21 ms
  testGetFlightsOnADivenDate 20 ms
  testBestPriceFlight 1 ms
  testGetFlights 0 ms
  testSortedByPrice 0 ms
Process finished with exit code 0
Tests passed: 4 (moments ago)
```

3. Nous avons implémenté l'organisateur de voyage.
4. Nous avons également implémenté le service de réservation de nuits d'hôtels qui nous sert en cas de réserver des hôtels pour tout voyage durant plus d'un jour.

Diagramme de classe



Couverture des tests

Coverage x JUnit					
TravelOrganizerTest (Dec 1, 2020 5:41:44 PM)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
▼ a315CooTd4BMS	86.4 %	1,948	306	2,254	
▼ src	70.9 %	615	253	868	
▼ hotel	52.7 %	98	88	186	
> HotelRental.java	25.0 %	23	69	92	
> Hotel.java	52.2 %	12	11	23	
> HotelRentalService.java	88.7 %	63	8	71	
▼ cars	74.6 %	214	73	287	
> CarRental.java	60.7 %	74	48	122	
> Car.java	41.4 %	12	17	29	
> CarRentalService.java	94.1 %	128	8	136	
▼ rental	75.0 %	84	28	112	
> Description.java	62.8 %	27	16	43	
> TravelOrganizer.java	86.0 %	37	6	43	
> Trip.java	76.9 %	20	6	26	
▼ core	48.9 %	22	23	45	
> Service4Pl.java	48.9 %	22	23	45	
▼ util	57.7 %	30	22	52	
> DateTools.java	75.0 %	30	10	40	
> AlreadyBooked.java	0.0 %	0	4	4	
> NotPossibleCarRentalException.java	0.0 %	0	4	4	
> NotPossibleHotelRentalException.java	0.0 %	0	4	4	
▼ flights	89.8 %	167	19	186	
> Flight.java	82.1 %	87	19	106	
> FlightService.java	100.0 %	80	0	80	
▼ tests	96.2 %	1,333	53	1,386	
▼ testRental	92.3 %	541	45	586	
> TravelOrganizerTest.java	92.3 %	541	45	586	
▼ cars	98.9 %	539	6	545	
> CarRentalServiceTest.java	98.4 %	381	6	387	
> CarRentalTest.java	100.0 %	158	0	158	
▼ flights	99.2 %	253	2	255	
> FlightTest.java	96.0 %	48	2	50	
> FlightServiceTest.java	100.0 %	205	0	205	

SonarLint

SonarLint Report X	
23 items	
Resource	Description
CarRentalService.java	Replace the type specification in this constructor call with the diamond operator ("<>").
CarRentalServiceTest.java	Remove this assertion from production code. [+18 locations]
CarRentalTest.java	Remove this assertion from production code. [+9 locations]
CarRentalTest.java	Remove the non-escaped \u00A0 character from this literal.
CarRentalTest.java	Remove the non-escaped \u00A0 character from this literal.
FlightServiceTest.java	Remove this assertion from production code. [+7 locations]
FlightTest.java	Remove this assertion from production code. [+2 locations]
HotelRentalService.java	Replace the type specification in this constructor call with the diamond operator ("<>").
Service4Pl.java	Replace the type specification in this constructor call with the diamond operator ("<>").
TravelOrganizer.java	Replace the type specification in this constructor call with the diamond operator ("<>").
TravelOrganizerTest.java	Rename this field "F1" to match the regular expression '^([a-z][a-zA-Z0-9])\$'.
TravelOrganizerTest.java	Rename this field "Ibis" to match the regular expression '^([a-z][a-zA-Z0-9])\$'.
TravelOrganizerTest.java	Rename this local variable to match the regular expression '^([a-z][a-zA-Z0-9])\$'.
TravelOrganizerTest.java	Rename this local variable to match the regular expression '^([a-z][a-zA-Z0-9])\$'.
TravelOrganizerTest.java	Rename this local variable to match the regular expression '^([a-z][a-zA-Z0-9])\$'.
TravelOrganizerTest.java	Rename this package name to match the regular expression '^([a-z_]+(\.[a-z_][a-z0-9_])*)\$'.
TravelOrganizerTest.java	This block of commented-out lines of code should be removed.
TravelOrganizerTest.java	This block of commented-out lines of code should be removed.
TravelOrganizerTest.java	This block of commented-out lines of code should be removed.
TravelOrganizerTest.java	This block of commented-out lines of code should be removed.
TravelOrganizerTest.java	This block of commented-out lines of code should be removed.
TravelOrganizerTest.java	Define a constant instead of duplicating this literal "Paris" 8 times. [+8 locations]
Trip.java	Replace the type specification in this constructor call with the diamond operator ("<>").

Compte rendu

1.1. Avez-vous bien respecté les exigences de l'ingénieur ?

Oui, après avoir suivi la totalité des consignes, notre diagramme final est plus complet que je diagramme de classe fourni dans le sujet. Nous pensons donc que les exigences de l'ingénieur ont bien toutes été respectées.

1.2. Couverture de tests :

1.2.1. Quels sont les principaux points faibles de votre couverture de test?

Tous les tests fournis avec le sujet sont passés comme montré en page 2. Néanmoins nous sommes conscients qu'il manque des tests en rapport avec certaines méthodes implémentées.

1.2.2. Quels points amélioreriez-vous si vous en aviez le temps?

Si nous avions le temps, nous essaierions d'ajouter dans nos tests, la couverture de ces différentes méthodes.

1.3. Expliquez comment vous avez cherché à respecter le principe ouvert/fermé, par exemple expliquez l'utilisation des interfaces, etc.

Pour respecter le OCP (Open/Closed Principle), nous avons ouvert nos classes à l'extension mais pas à la modification. La grande majorité de nos attributs sont déclarés en « private » et les autres en « protected ». Nous avons donc utilisé les principes d'héritages et d'interfaces pour coller au mieux aux exigences du OCP. Par exemple notre classe CarRentalService qui est une extension de Sercive4PI et qui implémente notre interface Service.

1.4. Si vous pensez que votre code respecte d'autres principes expliquez.

Notre code respecte d'autres principes comme le SRP car certaines classes n'ont qu'une seule responsabilité comme la classe Cars qui sert uniquement à manipuler des voitures par exemple et l'ISP car nous utilisons plusieurs interfaces plutôt qu'une seule globale.

1.5. Commentez le diagnostic de SonarLint.

SonarLint analyse le code comme s'il allait partir en production. Nous voyons donc bien qu'une partie des erreurs provient des tests. En effet dans un code prêt pour la production, les tests et donc les asserts ne doivent plus apparaître dans le code. Les seuls problèmes relatés par SonarLint qui ne sont pas dans les tests sont des optimisations de typage par exemple :

```
products = new ArrayList<Product>(); aurait pu s'écrire : products = new ArrayList<>();
```

Nous avons décidé de les laisser comme repères.

Instructions de remises

Vous rendez

1. Un document pdf contenant :
 1. Comme d'habitude soignez la 1e page avec les informations le groupe et le nom du ou des étudiants.
 1. Préciser sur la première page si :
 1. [] vos codes compilent
 2. [] les tests passent
 3. [] vous avez implémenté l'organisateur de voyage
 4. [] vous avez implémenté le service de réservation de nuits d'hôtels
 2. Un diagramme de classe au format pdf dans lequel apparait toute vos classes dans leur version finale
 3. La couverture de tests (pas de souci si elle n'est pas optimale)
 4. Le diagnostic SonarLint.
 5. La réponse aux questions suivantes :
 1. Avez-vous bien respecté les exigences de l'ingénieur?
 1. Expliquez-vous, en une ou deux lignes. (e.g. oui, j'ai exactement le même diagramme de classes mais...; oui, mais j'ai ajouté...; non,)
 2. Couverture de tests :
 1. Quels sont les principaux points faibles de votre couverture de test?
 2. Quels points amélioreriez-vous si vous en aviez le temps?
 3. Expliquez comment vous avez cherché à respecter le principe ouvert/fermé, par exemple expliquez l'utilisation des interfaces, etc.
 4. Si vous pensez que votre code respecte d'autres principes expliquez.
 5. Commentez le diagnostic de Sonarlint.
2. Vos codes sous la forme d'un .zip contenant exclusivement
 1. La dernière version des codes sources et des tests.
 2. Et pas de binaires ou d'anciennes versions.

Le rendu se fait en binôme ou en trinôme

- UN des étudiants rend le devoir qui contient le nom de son ou ses camarades,
- L'autre étudiant rend simplement le devoir par un message précisant le nom de son camarade qui a rendu le devoir pour lui.

Si cela n'est pas fait (a) aucun recours possible en cas d'absence de rendu (b) un malus est appliqué pour non-respect de la consigne à l'étudiant qui n'a rien rendu.