

# Agilité

La rencontre du Loto et de la Course et boom... ça fait gagner de l'argent.

Version: 2.1

13 – 02 – 2019

# Sommaire

---

<b>Sommaire</b>	<b>2</b>
<b>Historique du document</b>	<b>3</b>
<b>1. Objectif &amp; Contexte</b>	<b>4</b>
1.1 Objectif	4
1.2 Contexte	4
<b>2. Implémentation des Designs Patterns</b>	<b>5</b>
2.1 Design Pattern Observer	5
2.2 Design Pattern AbstractFactory	7
<b>3. Mini tutoriel</b>	<b>10</b>
3.1 Tutoriel du pattern Observer	10
3.2 Tutoriel du pattern AbstractFactory	12
3.3 Tutoriel de tests JUnit	13
3.4 Tutoriel de tests Cucumber	14
<b>4. Annexes</b>	<b>17</b>
4.1 Création d'un projet Eclipse	17
4.2 Tests JUnit	20
4.3 Cucumber via Eclipse	22
4.3.1 Installation de Cucumber	22
4.3.2 Configuration de Cucumber pour exécution des tests	23
4.3.3 Création des fichiers pour description des Users Story	23

## Historique du document

---

### Informations générales

**Rédacteur(s) :** HADDAD Abderraouf  
AISSAT Yacine  
TIOUCHICHINE Lounes  
ANDRIAMBELOMAHERY Sam Michael

**Valideur(s) :** ZAMFIROIU MICHEL

### Historique des mises à jour

Version	Date	Description
1.1	01/01/2019	Création du document
2.1	13/02/2019	Finalisation du document

### Documents de référence

[https://github.com/yacineaissat/Agilite\\_Course\\_Loto\\_AAHT.git](https://github.com/yacineaissat/Agilite_Course_Loto_AAHT.git)

# 1. Objectif & Contexte

---

## 1.1 Objectif

Ce document a pour objectif de décrire les étapes de réalisation du projet d'Agilité. Ce document a été conçu à la suite de la fusion des deux projets qui étaient pourtant très prometteurs l'un et l'autre... Le jeu de Loto et la Course. Depuis leur fusion, le projet a pris une toute autre ampleur, les 4 contributeurs ont alliés leurs forces pour produire un projet de qualité. Accrochez-vous pour la suite...

## 1.2 Contexte

A une époque lointaine, où la Terre n'était composée que d'un seul continent, le monde était vaste et des lieux improbables existaient, citons par exemple Onizuka, Uranus, Numidia et l'incontournable Madina, des places mythiques où le lendemain de chaque pleine lune se déroulaient des courses mythiques. Durant ces courses, de nombreux paris étaient organisés et les joueurs tentaient de gagner le plus d'or possible en pariant sur le classement des courses.

« Uhtred le jeune », fils d'Uhtred de Madina, était un de ces parieurs. Paresseux, il ne s'intéressait que peu aux coureurs, à leurs capacités et à leurs résultats. Il n'avait que faire des favoris, des coureurs blessés ou d'autres caractéristiques. Sa manière de jeu était assez singulière. Il classait simplement les joueurs de manière aléatoire grâce à leur numéro de dossard, et à la fin de la course il comparait son classement avec le résultat pour savoir si oui ou non il avait gagné de l'or.

L'histoire d'Uhtred le jeune a inspiré, des siècles plus tard, de nombreux jeux, notamment le jeu du Loto.

Afin de retracer l'histoire de ce valeureux parieur, nous avons décidé d'unir les forces de la classe fétiche Loto et de la classe fétiche Course afin de recréer la méthode de pari d'Uhtred.

L'objectif est simple. 6 joueurs avec un numéro de dossard participent à une course. Le parieur va alors entrer ces 6 numéros de manière aléatoire afin d'avoir le résultat de sa course. Il aura la possibilité notamment de connaître le gain potentiel qu'il peut faire avec ce classement aléatoire, mais il pourra surtout savoir si oui ou non il a gagné à l'issue de la course.

## 2. Implémentation des Designs Patterns

---

C'est connu, les Designs Patterns n'ont pas été inventés pour faire beau. Ils ont un objectif bien défini. Répondre à des problèmes qui se posent couramment par une conception. Ici, on va donc présenter les deux principaux problèmes qui se sont présentés à nous et les deux designs patterns que nous avons choisis pour tenter d'y répondre.

### 2.1 Design Pattern Observer

Le premier Design Pattern que nous avons choisi d'implémenter est l'Observateur.

Le problème qui se posait à nous est que les valeurs de certains de nos attributs changent régulièrement. De plus, un certain nombre d'autres classes du projet doivent être tenues au courant de l'évolution de ces valeurs. C'est un problème récurrent qui se pose dans beaucoup de projets informatiques.

Forts de ce constat, nous avons cherché un moyen pour pallier ce problème de mise à jour des valeurs.

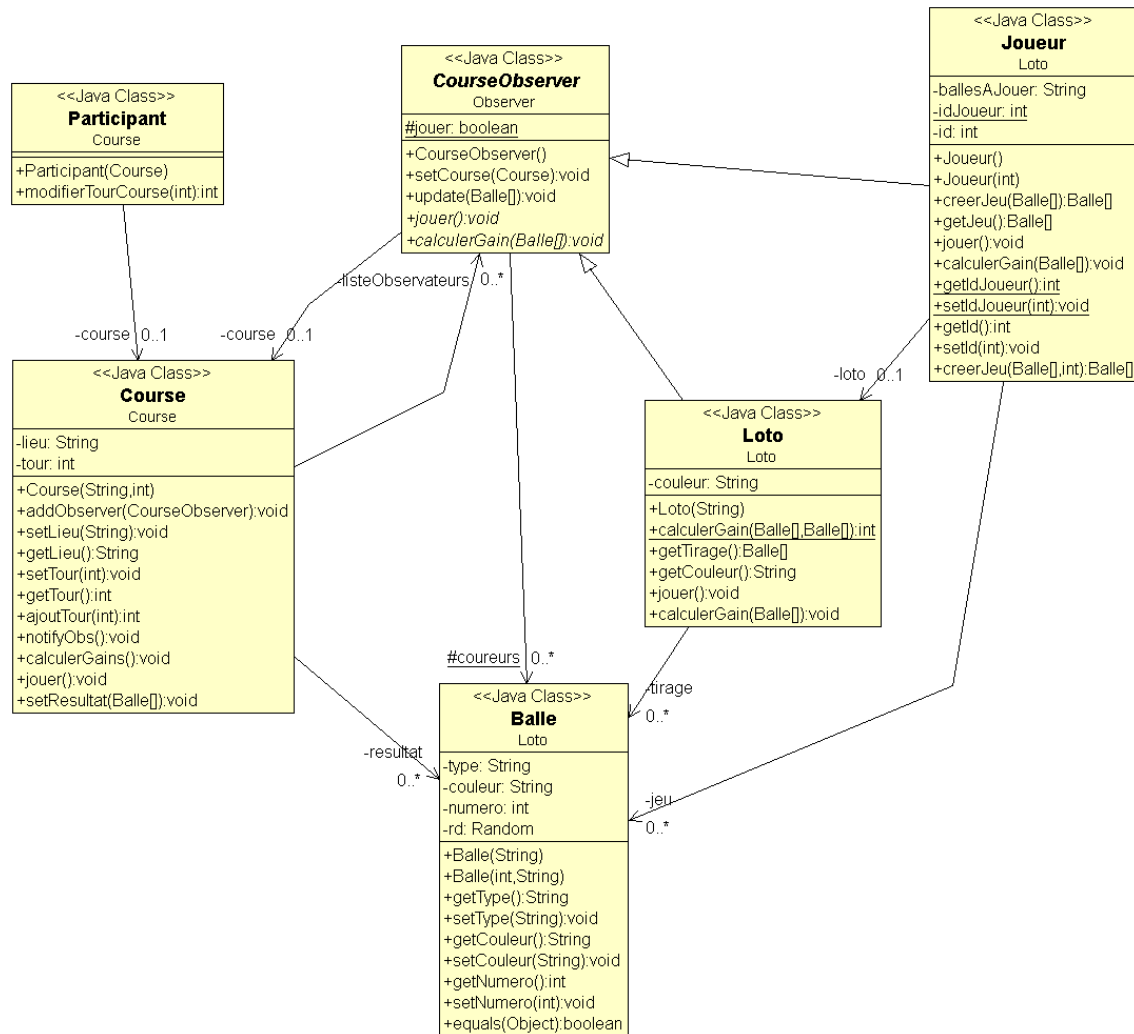
Première idée ? Les classes interrogent la classe qui est mise à jour régulièrement afin de lui demander si elle a mis à jour des valeurs récemment. Elles récupèrent alors les nouvelles valeurs et les mettent à jour localement.

On voit très vite que cette idée est... comment dire... un peu pourrie. En effet, quand les classes doivent demander si une valeur est mise à jour ? Toutes les 10 secondes ? Toutes les minutes ? Tous les 30 ans ? Bonne question... Et peu importe, pour tous les appels faits 'inutilement', dans le sens où on va demander si une modification a eu lieu et que la réponse est non, on aura une surcharge inutile d'appels qui peut être assez importante si on a beaucoup de classes et que les demandes sont faites régulièrement. Mauvaise idée donc.

Deuxième idée ? Le plus intelligent dans ce cas serait de mettre en place une conception qui permet que, dès lors que la classe se met à jour, elle va notifier les autres classes pour leur dire « Hé ho ! J'ai mis à jour plusieurs valeurs les amis, mettez-les à jour chez vous aussi sinon on va être en décalage. Bises ! »

C'est donc dans cette optique là que nous avons mis en place le premier design pattern, l'Observateur, qui répond parfaitement à ce problème d'implémentation.

Voici le diagramme de classe de l'implémentation du design pattern Observer.



Voici enfin quelques informations intéressantes concernant l'implémentation de ce pattern.

D'abord, une analyse globale de l'implémentation. Qui contient notamment le nombre de lignes de code, le nombre de classes et de packages.

Analysis of Observer\_Loto\_Course  
working set:

{Observer\_Loto\_Course}

General Information

Total lines of code: 305

Number of classes: 12

Number of packages: 6

Number of external packages: 1

Number of external classes: 1

Number of problematic classes: 0

Number of highly problematic classes: 0

Ensuite, une analyse détaillée de l'implémentation de chaque classe.

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Joueur	■	■	■	■	64	low	low	low	low-medium
2	BalleTest	■	■	■	■	40	low	low	low	low
3	CourseTest	■	■	■	■	39	low	low	low	low
4	Balle	■	■	■	■	34	low	low	low	low
5	Course	■	■	■	■	32	low	low	low-medium	low
6	Loto	■	■	■	■	28	low	low	low	low
7	TestLoto	■	■	■	■	21	low	low	low	low
8	JoueurTest	■	■	■	■	19	low	low	low	low
9	CourseObserver	■	■	■	■	10	low	low	low	low
10	Main	■	■	■	■	9	low	low	low	low
11	Participant	■	■	■	■	6	low	low	low	low
12	RunnerTestLoto	■	■	■	■	3	low	low	low	low

## 2.2 Design Pattern AbstractFactory

Le second Design Pattern que nous avons choisi d'implémenter est l'AbstractFactory.

Le second problème qui se posait à nous est que, étant donné que nous faisons des paris sur des courses, très vite, étant donné que le business s'agrandit, nous allons devoir gérer différents types de courses. Et alors, me direz-vous ? Eh bien, on ne gère pas pareillement une course de voiture, une course de coureurs ou une course d'escargots... Nous avons donc besoin d'une conception simple qui nous permettent d'ajouter facilement de nouveaux types de courses à gérer. C'est la même chose pour les joueurs/parieurs, nous avons plusieurs types de parieurs. Les habitués, les professionnels, ceux qui jouent en ligne... Nous devons pouvoir les différencier et les implémenter facilement.

Forts de ce constat, nous avons cherché un moyen pour pallier ce problème de genericité et d'adaptabilité de notre code.

Première idée ? Ecrire une succession de conditions qui suivant le type de courses/ de personnes, instancie et retourne l'objet correspondant.

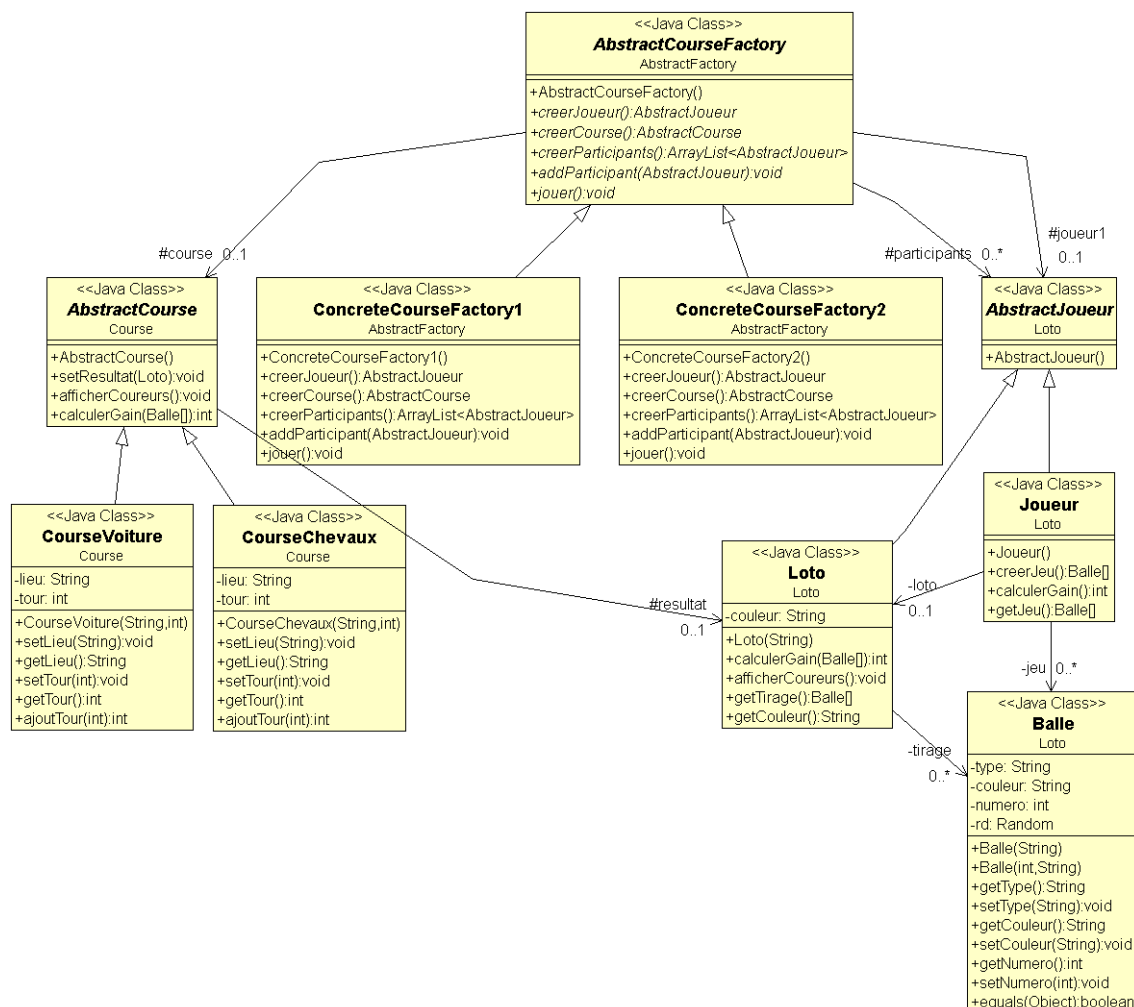
Le problème ici, c'est que la classe mère va être forcément couplée à tous les produits qu'elle va instancier. Et le problème se posera lorsqu'on souhaitera ajouter de nouveaux types de joueurs ou de nouvelles courses... et également lorsqu'on souhaitera supprimer par exemple un type de courses qui ne rapporte pas d'argent. Mauvaise idée encore...

Deuxième idée ? La meilleure solution serait d'avoir un créateur qui va pouvoir manipuler les sous-produits. Chaque créateur pourra donc créer des produits dont il est responsable. Tous les produits implémentent bien-sûr la même interface afin que les classes créatrices puissent s'y référer sans forcément connaître le type concret.

Dans notre cas, étant donné qu'une partie de l'implémentation est identique à tous les produits, on peut alors choisir de faire de l'interface générale une classe abstraite afin d'intégrer le code partagé dans celle-ci.

C'est donc dans cette optique là que nous avons mis en place le second design pattern, l'AbstractFactory, qui répond parfaitement à ce problème d'implémentation.

Voici le diagramme de classe de l'implémentation du design pattern AbstractFactory.





Voici enfin quelques informations intéressantes concernant l'implémentation de ce pattern.

D'abord, une analyse globale de l'implémentation. Qui contient notamment le nombre de lignes de code, le nombre de classes et de packages.

## Analysis of AbstractFactory\_Loto\_Course

General Information

**Total lines of code: 311**

**Number of classes: 15**

**Number of packages: 6**

**Number of external packages: 1**

**Number of external classes: 1**

**Number of problematic classes: 0**

**Number of highly problematic classes: 0**

Ensuite, une analyse détaillée de l'implémentation de chaque classe.

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	CourseTest	■	■	■	■	47	low	low	low	low
2	BalleTest	■	■	■	■	40	low	low	low	low
3	Balle	■	■	■	■	34	low	low	low	low
4	ConcreteCourseFac...	■	■	■	■	27	low	low	low	low
5	ConcreteCourseFac...	■	■	■	■	27	low	low	low	low
6	Loto	■	■	■	■	26	low	low	low	low
7	TestLoto	■	■	■	■	22	low	low	low	low
8	Joueur	■	■	■	■	22	low	low	low	low
9	CourseVoiture	■	■	■	■	16	low	low	low	low
10	CourseChevaux	■	■	■	■	16	low	low	low	low
11	Client	■	■	■	■	13	low	low	low	low
12	AbstractCourseFac...	■	■	■	■	9	low	low	low	low
13	AbstractCourse	■	■	■	■	8	low	low	low	low
14	RunnerTestLoto	■	■	■	■	3	low	low	low	low
15	AbstractJoueur	■	■	■	■	1	low	low	low	low

## 3. Mini tutoriel

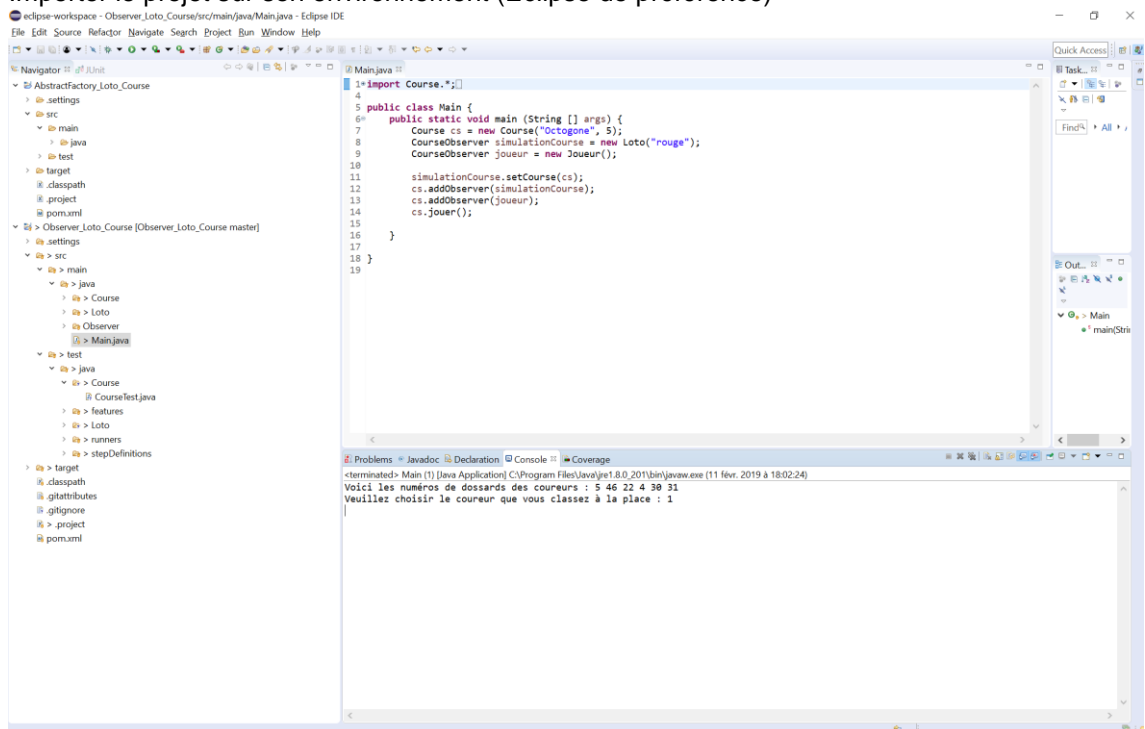
### 3.1 Tutoriel du pattern Observer

Dans cette partie, nous allons tenter d'expliquer de manière simple à l'utilisateur comment s'approprier notre code et surtout comment le faire fonctionner.

L'unique prérequis est d'avoir un IDE fonctionnel (Eclipse de préférence) sur son poste.

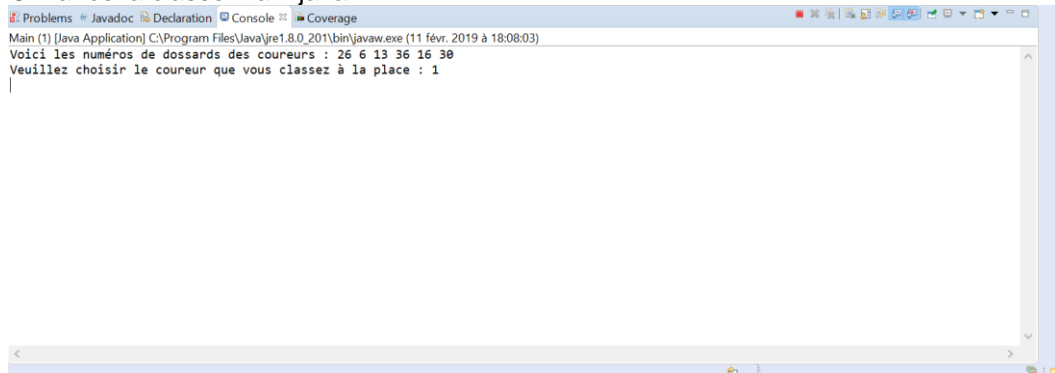
Pour commencer, l'utilisateur doit :

- 1- Récupérer le code via le lien Github suivant :  
[https://github.com/yacineaissat/Agilite\\_Course\\_Loto\\_AAHT.git](https://github.com/yacineaissat/Agilite_Course_Loto_AAHT.git)
- 2- Importer le projet sur son environnement (Eclipse de préférence)



- 3- Une fois que les projets sont finalement ouverts, on va tester le design Pattern Observer.

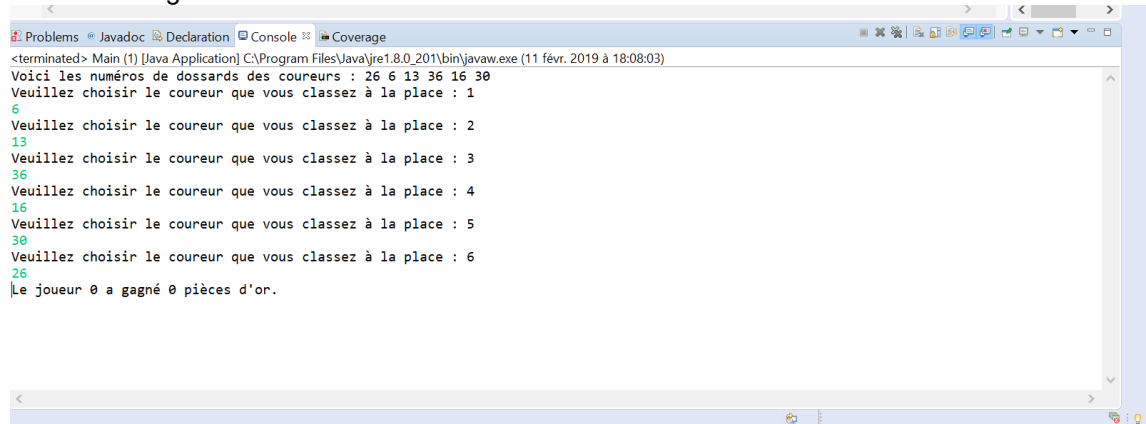
On lance la classe Main.java



On va alors tenter de deviner l'ordre d'arrivée des coureurs.

Pour cela, on va entrer un à un l'ordre d'arrivée des joueurs.

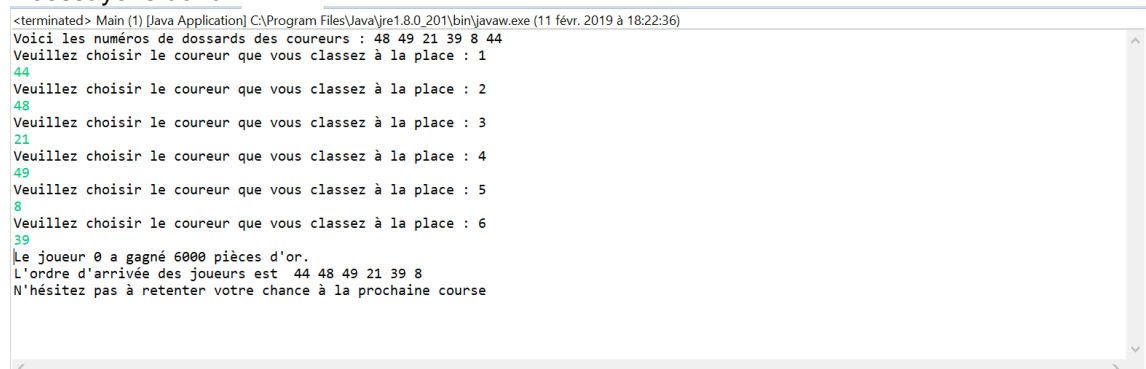
Et là... notre gain s'affiche...



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (11 févr. 2019 à 18:08:03)
Voici les numéros de dossards des coureurs : 26 6 13 36 16 30
Veuillez choisir le coureur que vous classez à la place : 1
6
Veuillez choisir le coureur que vous classez à la place : 2
13
Veuillez choisir le coureur que vous classez à la place : 3
36
Veuillez choisir le coureur que vous classez à la place : 4
16
Veuillez choisir le coureur que vous classez à la place : 5
30
Veuillez choisir le coureur que vous classez à la place : 6
26
Le joueur 0 a gagné 0 pièces d'or.
```

Dans notre cas, on a tout perdu !! Nous finissons pauvres et endettés... On a pensé à braquer une banque pour essayer de se refaire mais finalement on va peut être retenter notre chance une deuxième fois sur la course suivante... peut-être qu'on arrivera à renflouer les caisses...

Réessayons donc une deuxième fois...



```
<terminated> Main (1) [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (11 févr. 2019 à 18:22:36)
Voici les numéros de dossards des coureurs : 48 49 21 39 8 44
Veuillez choisir le coureur que vous classez à la place : 1
44
Veuillez choisir le coureur que vous classez à la place : 2
48
Veuillez choisir le coureur que vous classez à la place : 3
21
Veuillez choisir le coureur que vous classez à la place : 4
49
Veuillez choisir le coureur que vous classez à la place : 5
8
Veuillez choisir le coureur que vous classez à la place : 6
39
Le joueur 0 a gagné 6000 pièces d'or.
L'ordre d'arrivée des joueurs est 44 48 49 21 39 8
N'hésitez pas à retenter votre chance à la prochaine course
```

YESSSS ! On a trouvé 2 bonnes positions ! On termine riches et on se met à la retraite pour le restant de nos jours 😊😊😊

Trêve de plaisanteries...

A vous de tenter votre chance maintenant !

On a ici pu tester le code du pattern Observer, maintenant, testons le code du pattern AbstractFactory.

## 3.2 Tutoriel du pattern AbstractFactory

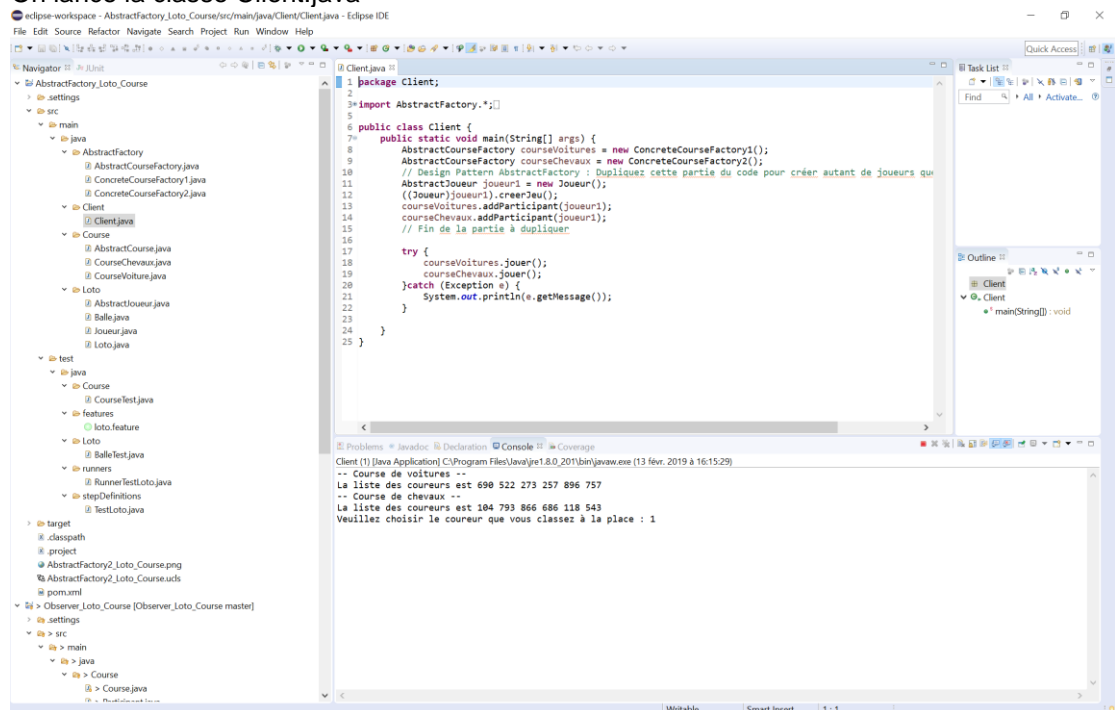
Dans cette partie, nous allons expliquer de manière simple à l'utilisateur comment s'approprier le code du design pattern AbstractFactory et surtout comment le faire fonctionner.

Vous êtes censé avoir déjà importé le code du pattern AbstractFactory sur Eclipse. Si ce n'est pas le cas, suivez les étapes 1 et 2 de la partie 3.1.

Pour commencer, l'utilisateur doit :

- 1- Une fois que le projet est finalement ouvert, on va tester le design pattern AbstractFactory

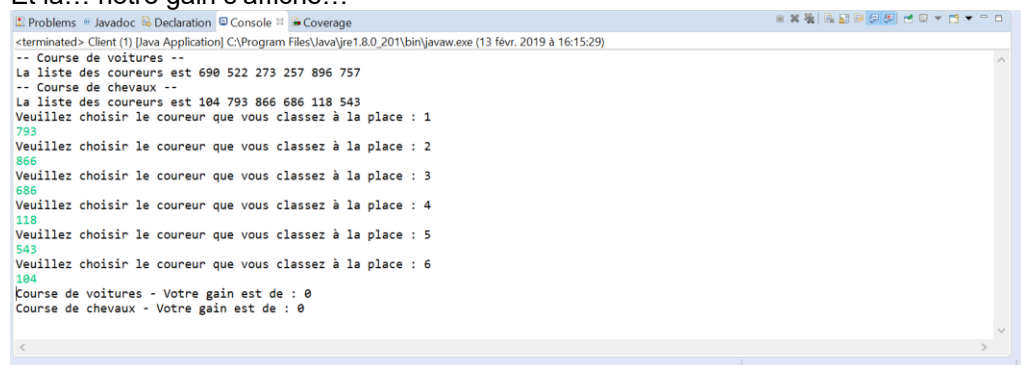
On lance la classe Client.java



On va alors tenter de deviner l'ordre d'arrivée des coureurs.

Pour cela, on va entrer un à un l'ordre d'arrivée des joueurs.

Et là... notre gain s'affiche...



Dans notre cas, on a tout perdu.

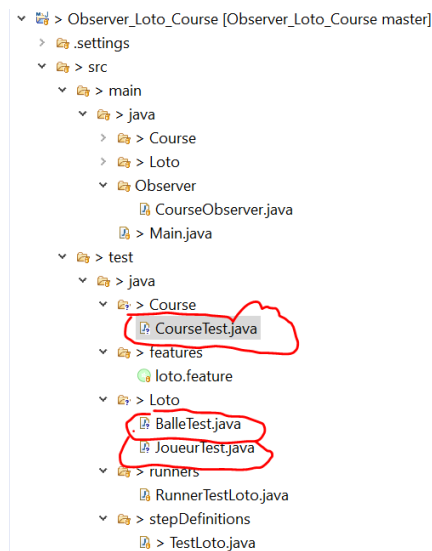
On vous laisse, encore une fois, tenter votre chance 😊.

Maintenant, passons à la partie la plus intéressante, les tests ! On vous laisse découvrir dans la suite les tests JUnit et les tests Cucumber... restez bien accroché ça va être très intense.

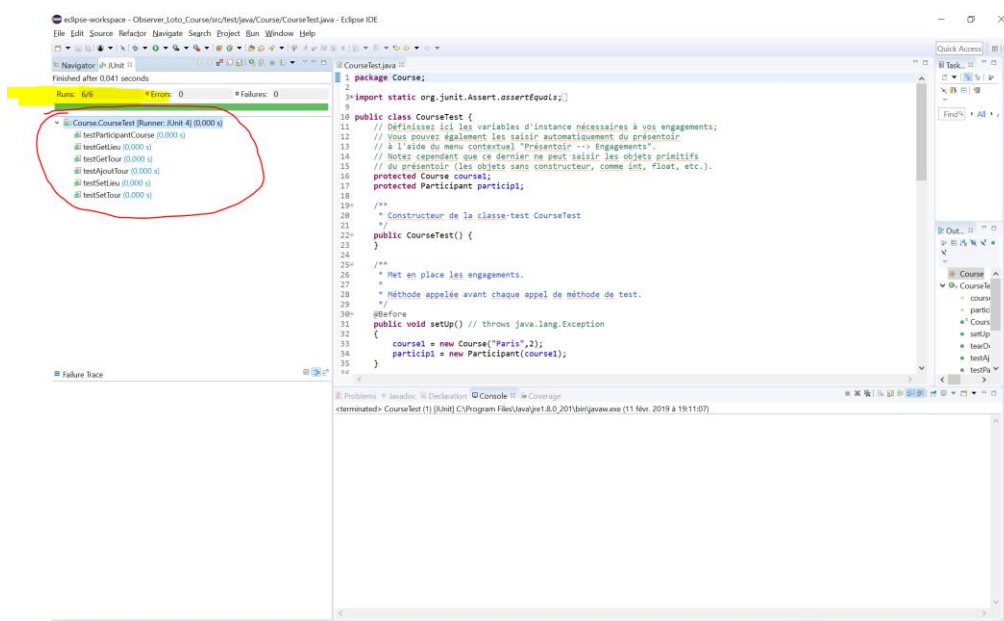
### 3.3 Tutoriel de tests JUnit

Afin d'exécuter les tests JUnit, il y a 3 classes à exécuter.

Vous les trouverez ci-dessous entourées en rouge.

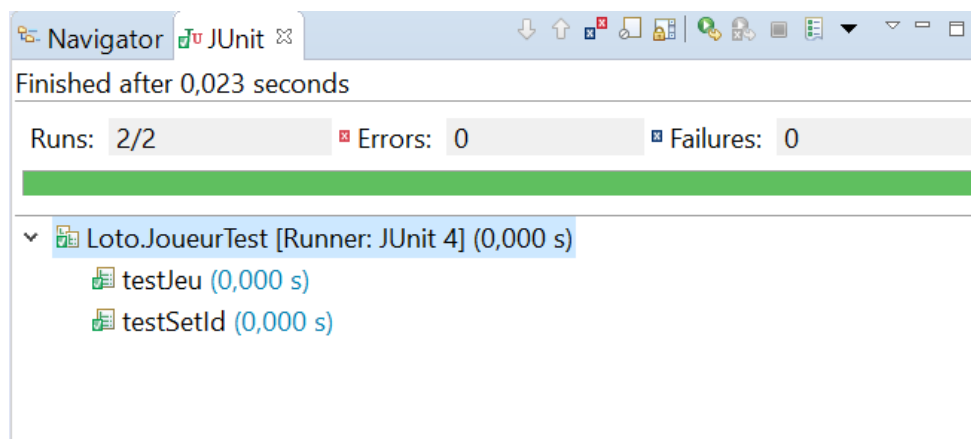
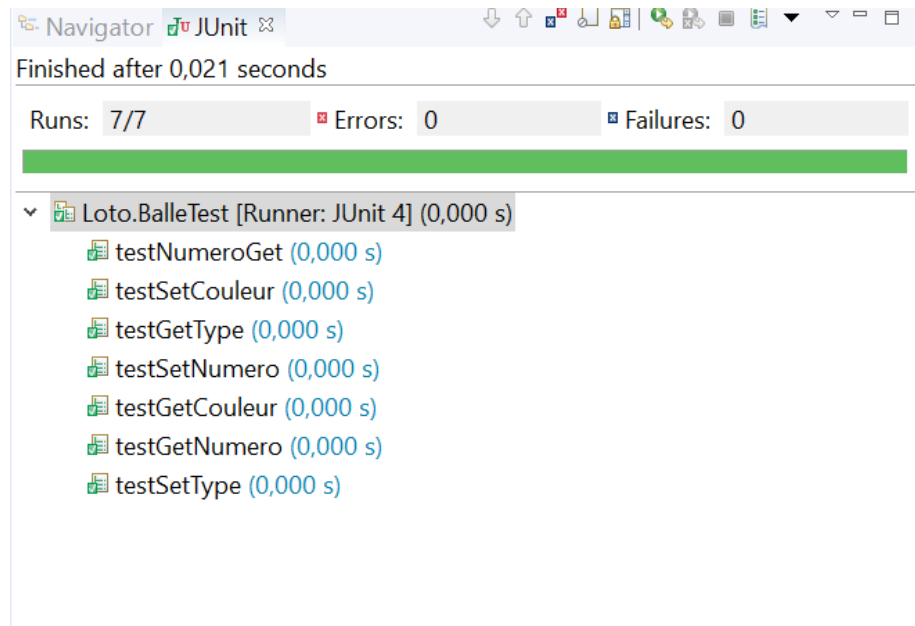


Exemple d'exécution de la classe : CourseTest.java



On voit à gauche que l'exécution s'est extrêmement bien passée, tout va bien dans le meilleur des mondes, on a la barre verte pour les 6 tests.

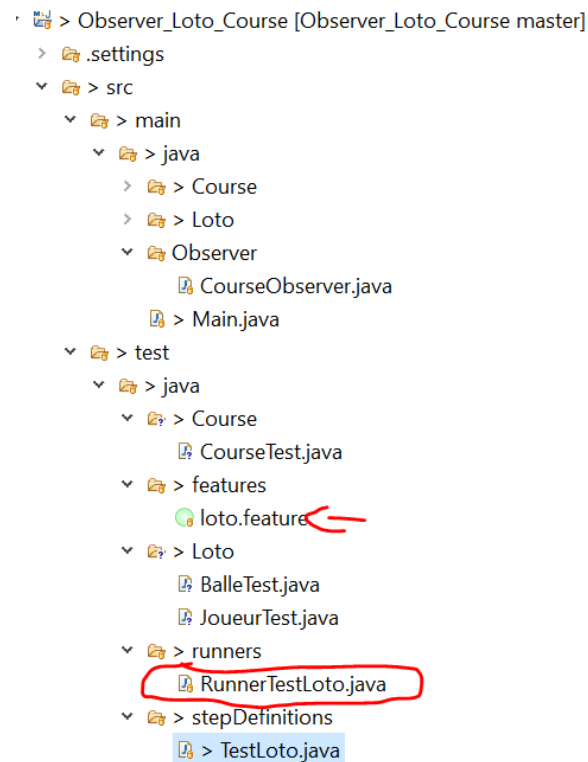
Exécution des deux autres classes de tests.



Les exécutions se déroulent bien pour chacun des classes... à vous maintenant de faire le test. Les tests sont identiques pour les deux patterns, cela n'a donc aucun intérêt de les exécuter pour les deux.

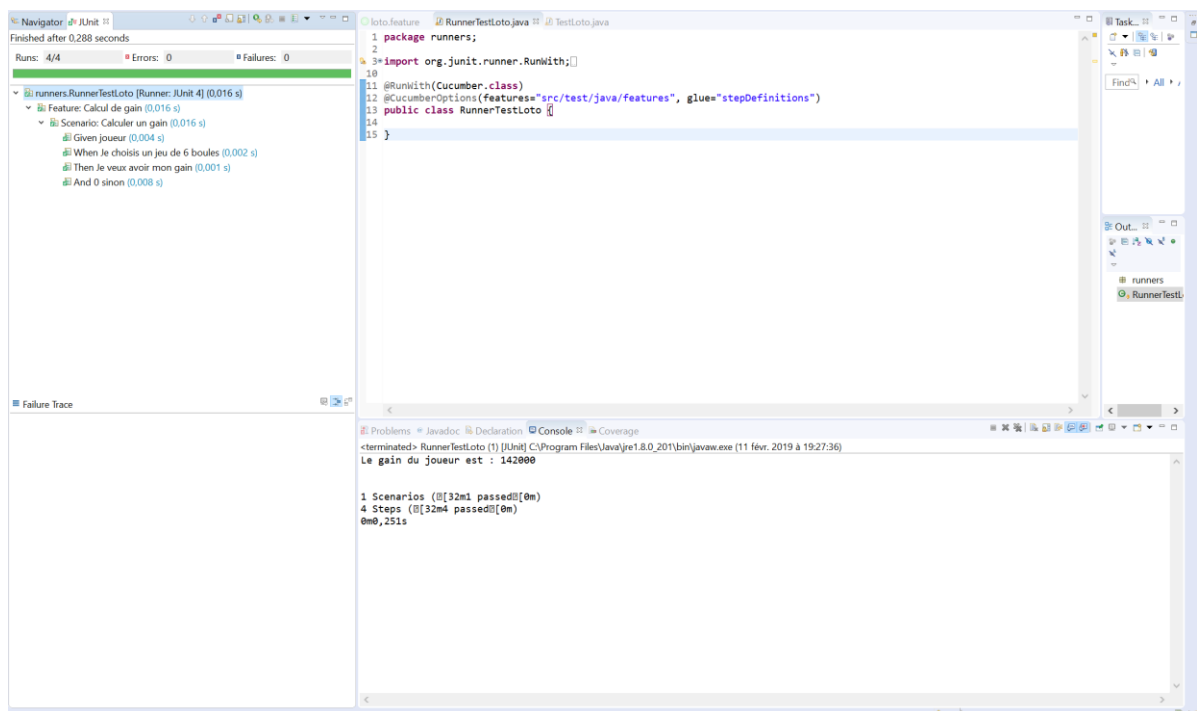
### 3.4 Tutoriel de tests Cucumber

Afin d'exécuter les scenarii Cucumber, il va falloir exécuter la classe RunnerTestLoto.java comme (celle entourée sur la capture suivante) :



Petit détail : le fichier loto.feature (celui suivi d'une flèche rouge) contient le scénario qui est exécuté.

Voici le résultat de l'exécution :



TADAM ! Encore des barres vertes qui signifient que les tests se sont bien exécutés...

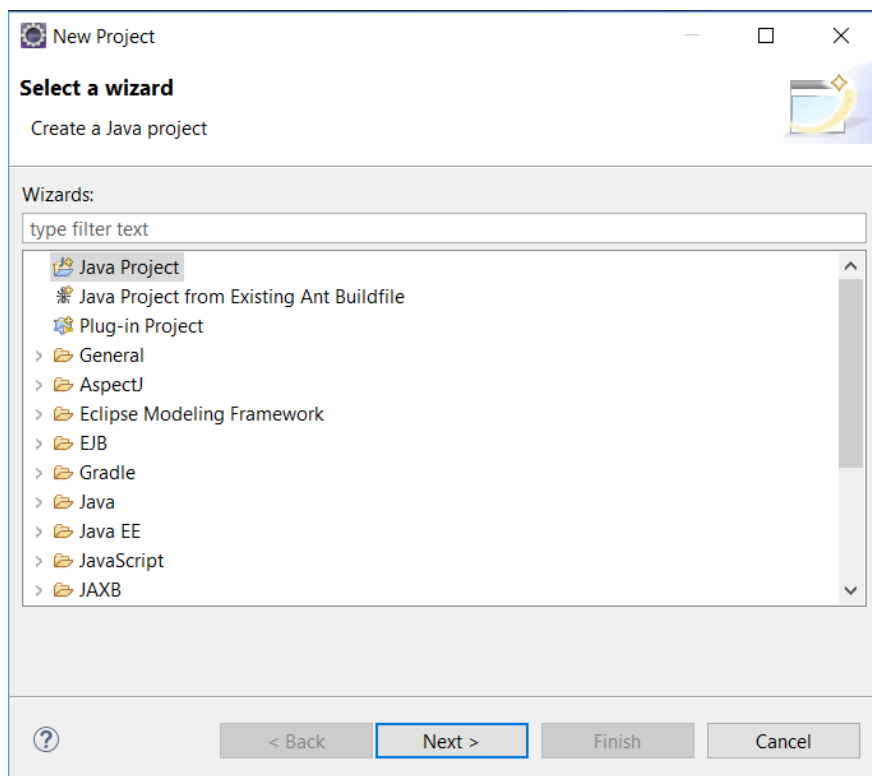
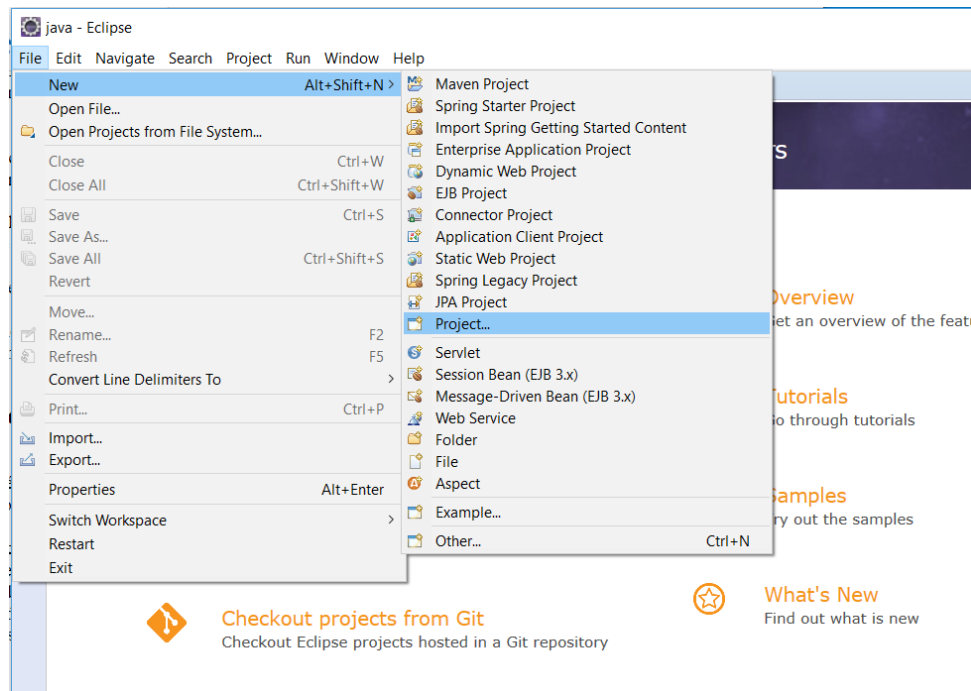
Encore une fois, nous vous laissons exécuter par vous-même les tests si l'envie vous prend... 😊 Attention, allez-y doucement, c'est super addictif !

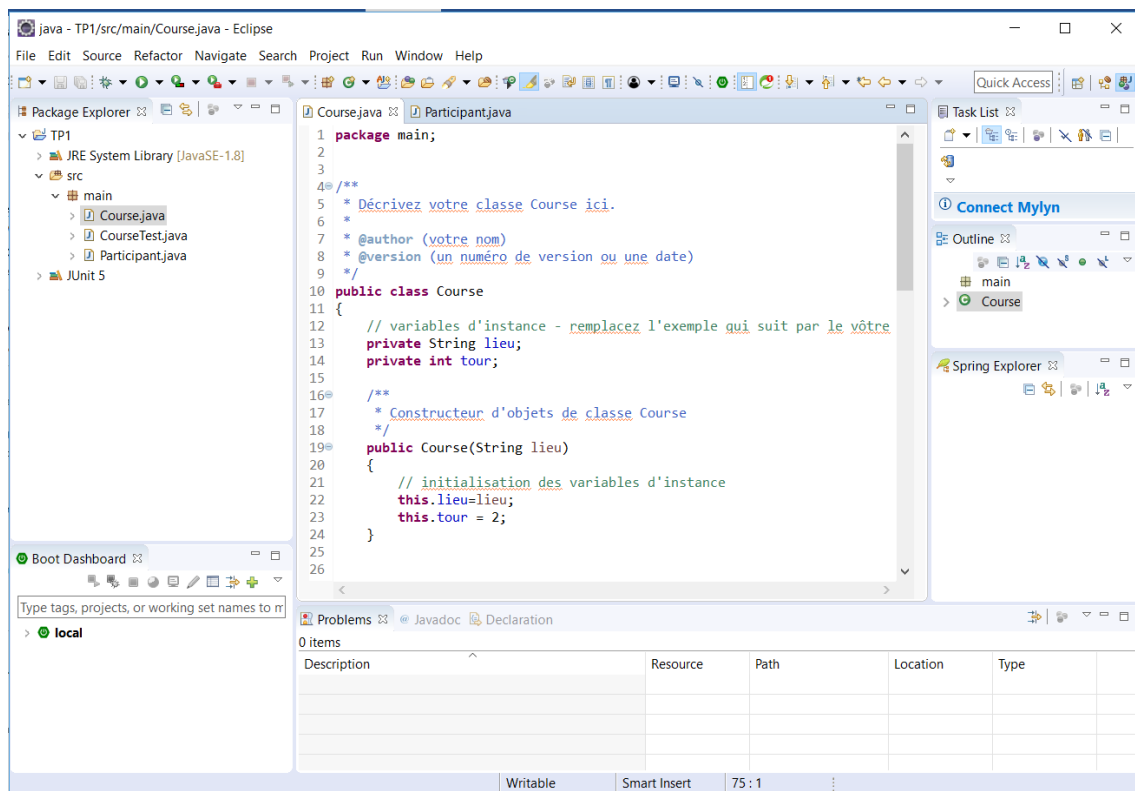
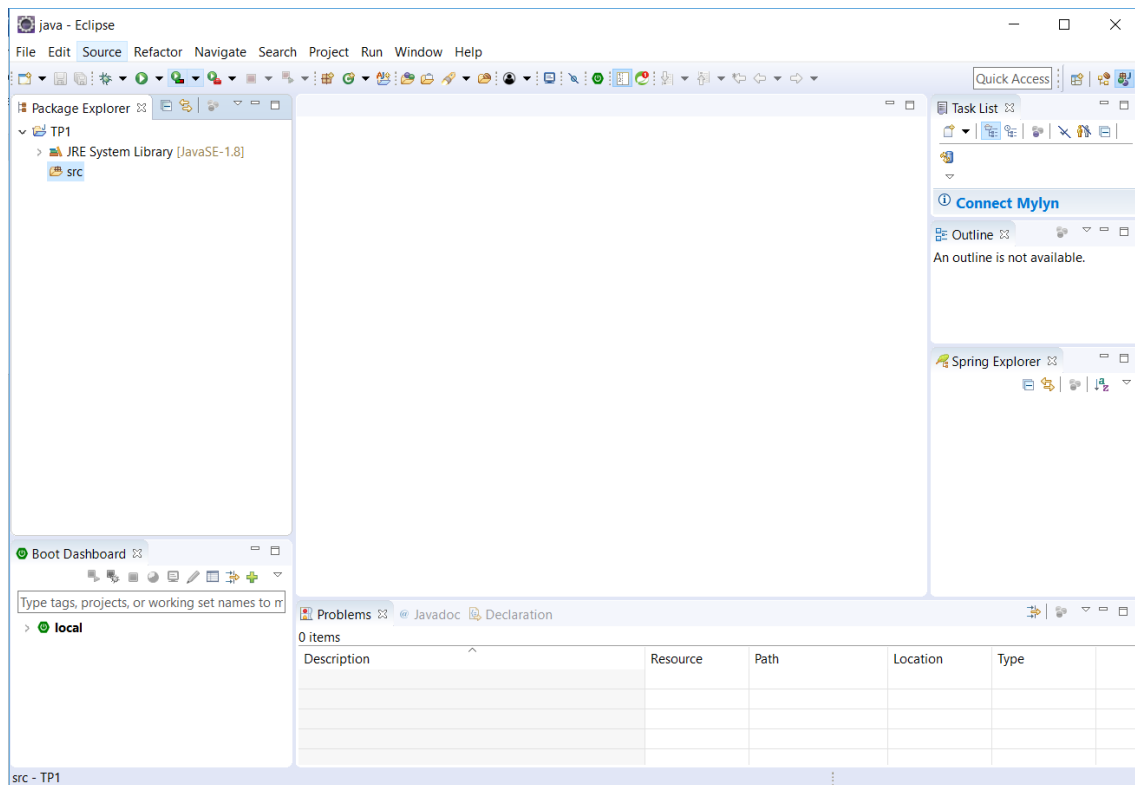


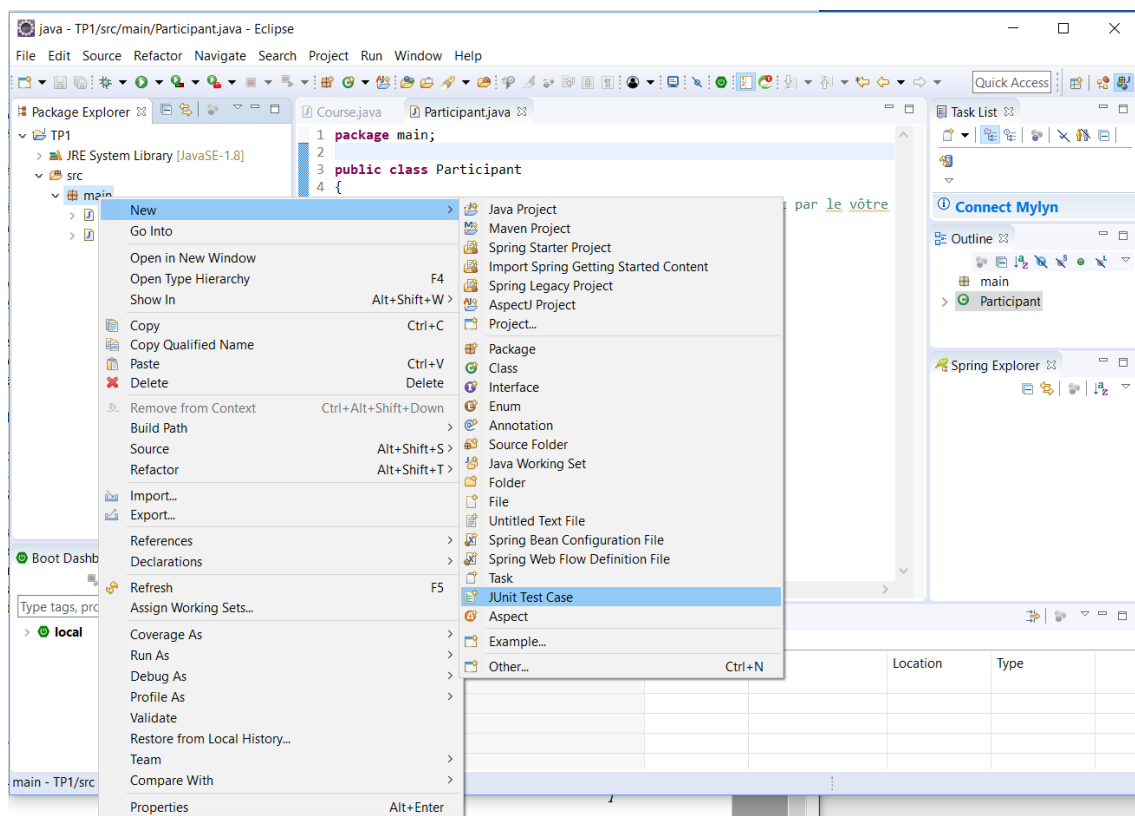
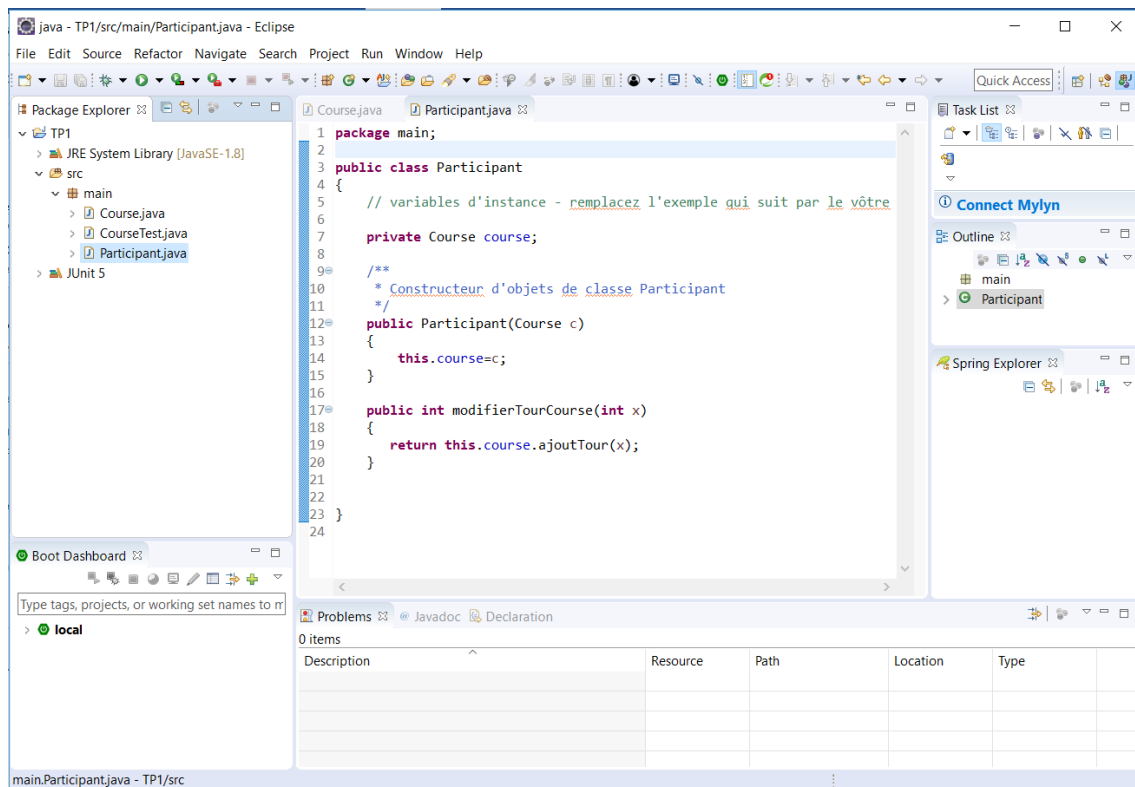
## 4. Annexe

En annexe, vous trouverez quelques trucs et astuces qui nous ont beaucoup servi tout au long de la réalisation de notre projet. Nous trouverons notamment des archives des précédents rendus de TP d'Agilité et quelques tutoriels pour commencer à utiliser certains outils. Enjoy 😊

### 4.1 Création d'un projet Eclipse







**New JUnit Test Case**

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

☐ New JUnit 3 test ☐ New JUnit 4 test ☒ New JUnit Jupiter test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()  
☐ setUp() ☐ tearDown()  
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Class under test:

## 4.2 Tests JUnit

```
1 package main;
2
3 import java.util.ArrayList;
4
5 /**
6  * Décrivez votre classe Course ici.
7  *
8  * @author (votre nom)
9  * @version (un numéro de version ou une date)
10 */
11 public class Course {
12     private String lieu;
13     private int tour;
14     private ArrayList<Participant> liste;
15
16     /**
17      * Constructeur d'objets de classe Course
18      */
19     public Course(String lieu, int tour) {
20         // initialisation des variables d'instance
21         this.lieu = lieu;
22         this.tour = tour;
23         this.liste = new ArrayList();
24     }
25
26     /**
27      * @param y le paramètre de la méthode
28      * @return la modification du lieu
29      */
30     public void setLieu(String y) {
31         // Insérez votre code ici
32         this.lieu = y;
33     }
34
35     /**
36      * @return le lieu
```

```
Course.java Participant.java CourseTest.java x
61 // TP2 Couverture de test
62 // Test des Getters
63 @Test
64 public void testGetLieu() {
65     Course course1 = new Course("Paris",2);
66     Assert.assertEquals("Paris", course1.getLieu());
67 }
68
69 @Test
70 public void testGetTour() {
71     Course course1 = new Course("Paris",2);
72     assertEquals("Vrai", 2, course1.getTour());
73     Assert.assertTrue("Faux", 5 > course1.getTour());
74     Assert.assertNotEquals("Faux", 7, course1.getTour());
75 }
76
77 // Test des Setters
78 @Test
79 public void testSetLieu() {
80     Course course1 = new Course("Paris",2);
81     course1.setLieu("Lyon");
82     assertEquals("Lyon", course1.getLieu());
83     Assert.assertNotEquals("Dijon", course1.getLieu());
84 }
85
86 @Test
87 public void testSetTour() {
88     Course course1 = new Course("Paris",2);
89     assertEquals("Vrai", 2, course1.getTour());
90 }
91
92 // test Association
93 @Test
94 public void testAddParticipant() {
95     course1 = new Course("Paris",2);
96     particip1 = new Participant(course1);
97     assertEquals(1, course1.addParticipant(particip1));
98     Participant particip2 = particip1 = new Participant(course1);
99     assertEquals(2, course1.addParticipant(particip2));
100     Assert.assertNotEquals(4, course1.addParticipant(particip2));
101 }
102
```

Package Explorer JUnit x

Finished after 0,11 seconds

Runs: 7/7 Errors: 0 Failures: 0

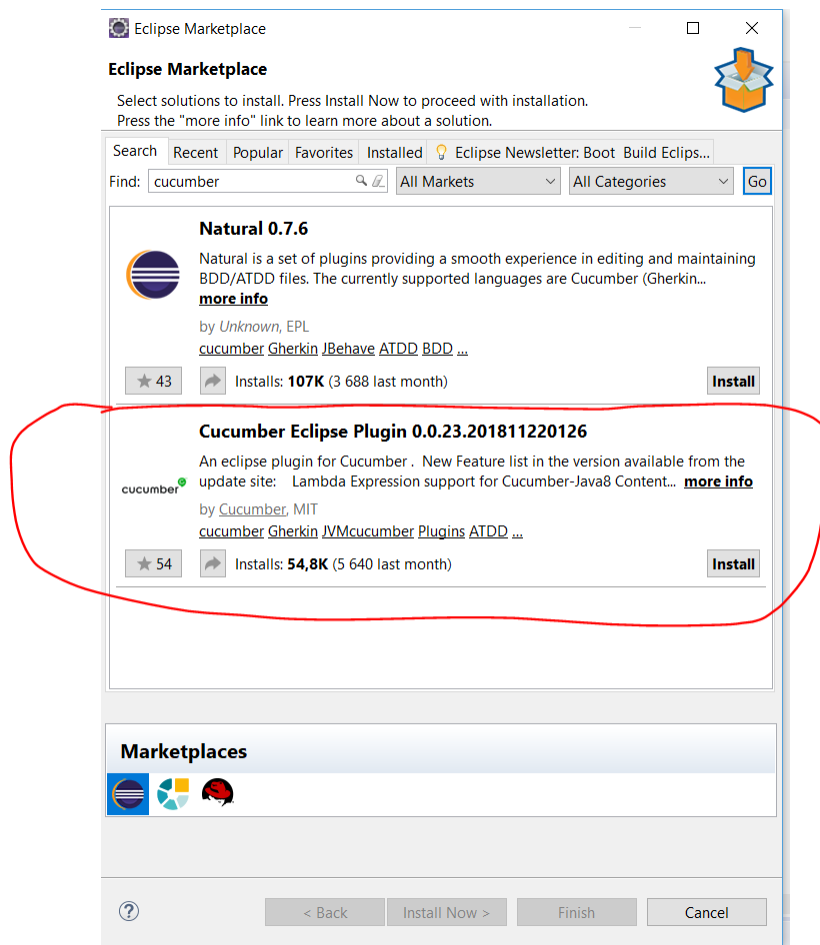
CourseTest [Runner: JUnit 5] (0,032 s)

- testParticipantCourse() (0,016 s)
- testGetLieu() (0,000 s)
- testGetTour() (0,000 s)
- testAddParticipant() (0,000 s)
- testAjoutTour() (0,000 s)
- testSetLieu() (0,000 s)
- testSetTour() (0,016 s)

## 4.3 Cucumber via Eclipse

### 4.3.1 Installation de Cucumber

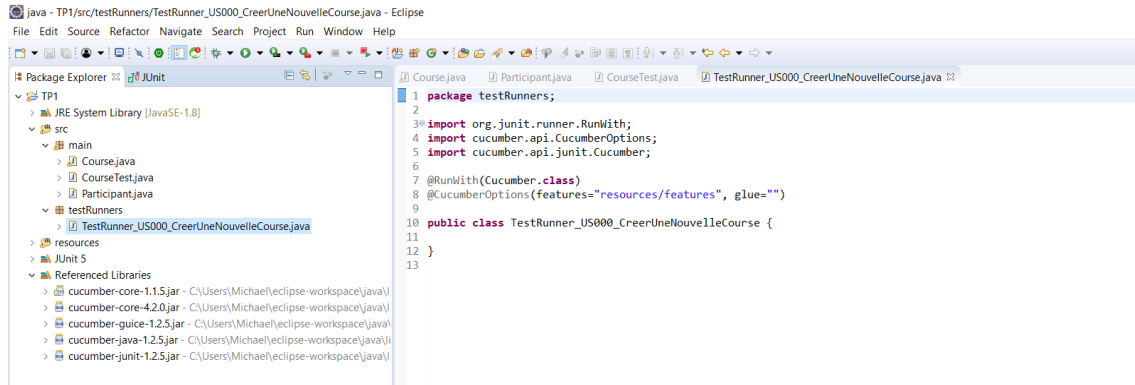
Installation de Cucumber



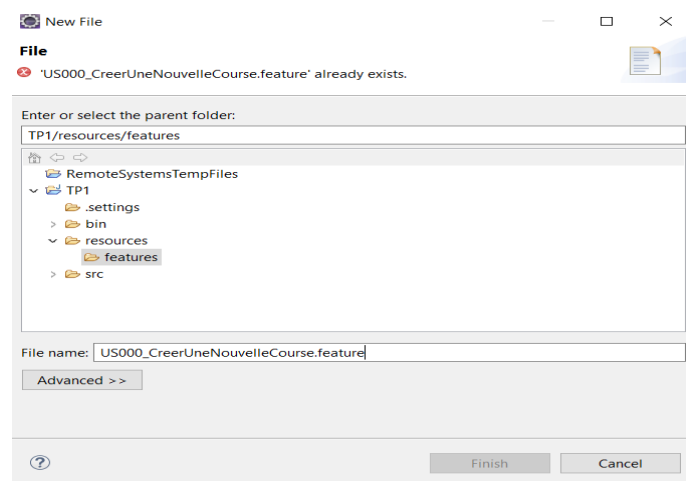
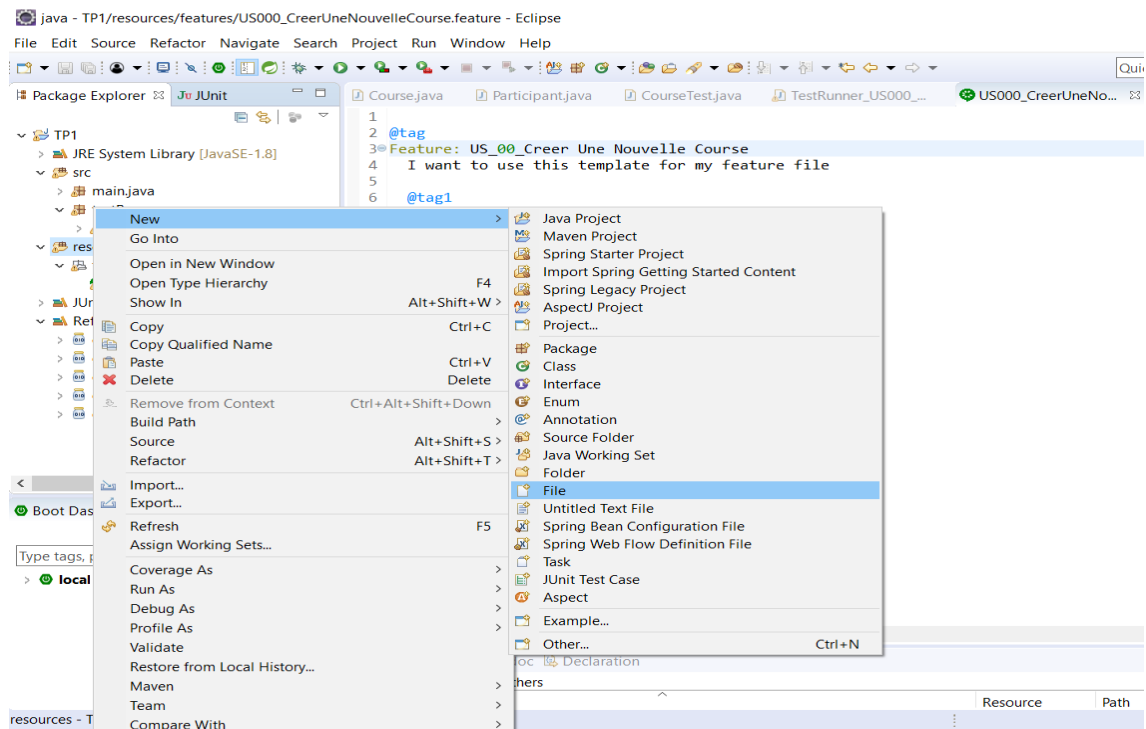
Ajout via le Build Path Eclipse :

- ▼ Referenced Libraries
  - > cucumber-core-1.1.5.jar - C:\Users\Michael\eclipse-workspace\java\l
  - > cucumber-core-4.2.0.jar - C:\Users\Michael\eclipse-workspace\java\l
  - > cucumber-guice-1.2.5.jar - C:\Users\Michael\eclipse-workspace\java\l
  - > cucumber-java-1.2.5.jar - C:\Users\Michael\eclipse-workspace\java\li
  - > cucumber-junit-1.2.5.jar - C:\Users\Michael\eclipse-workspace\java\l

### 4.3.2 Configuration de Cucumber pour exécution des tests



### 4.3.3 Création des fichiers pour description des Users Story



```

Course.java Participant.java CourseTest.java TestRunner_US000_CreerUneNouvelleCourse.java US000_CreerUneNouvelleCourse.feature
1 @tag
2 Feature: US00_Creer Une Nouvelle Course
3   En tant que participant
4   Je veux pouvoir créer une nouvelle course, avec lieu de la course, nombre de tour.
5   Afin que les participants puisse accéder aux informations de la course et y participer.
6
7 Scenario Outline: Création automatique de la liste des participants
8   Given la création d'une nouvelle course
9   And some other precondition
10  When le participant la valide
11  Then la liste des participants, vide, est créée automatiquement
12
13 Examples:
14   | lieu1 | tour1 | listeParticipants |
15   | Paris | 4     | {}                |
16
17 Scenario Outline: Refus de nombre de tours négatif ou nul
18   Given la création d'une nouvelle course
19   When le participant la valide
20   Then le système refuse avec le <messageErreur>
21 Examples:
22   | lieu1 | tour1 | messageErreur |
23   | Paris | -4    | nb tours négatif |
24   | Lyon  | 0     | nb tours nul    |
25
26 Scenario Outline: Refus de lieu vide
27   Given la création d'une nouvelle course
28   When le participant la valide
29   Then le système refuse avec le <messageErreur>
30 Examples:
31   | lieu1 | tour1 | messageErreur |
32   | « »   | 4     | lieu non défini |
33
34
35

```

```

Course.java Participant.java CourseTest.java US001_ModifierCourseExistante.feature
1 @test1
2 Feature: US_001 Modifier course existante
3   En tant que participant
4   Je veux pouvoir modifier les caractéristiques d'une course, à savoir son lieu et son nombre de tours
5   Afin de s'assurer de la possibilité de mise à jour d'une course.
6
7 Scenario Outline: Refus de nombre de tours négatif ou nul
8   Given la modification d'une course existante
9   When le participant la valide
10  Then le système refuse avec le <messageErreur>
11 Examples:
12   | lieu1 | tour1 | messageErreur |
13   | Paris | -4    | nb tours négatif |
14   | Lyon  | 0     | nb tours nul    |
15
16 Scenario Outline: Refus de lieu vide
17   Given la modification d'une course existante
18   When le participant la valide
19   Then le système refuse avec le <messageErreur>
20 Examples:
21   | lieu1 | tour1 | messageErreur |
22   | « »   | 4     | lieu non défini |

```

```

Course.java Participant.java CourseTest.java US002_AjouterParticipantsAUneCourse.feature
1 @test1
2
3 Feature: US_002 Ajouter participants à une course
4   En tant que participant
5   Je veux pouvoir ajouter d'autres participants à une course
6   Afin que cela soit plus fun.
7
8 Scenario Outline: Refus de participant ou course inexistante
9   Given l'ajout d'un participant à une course
10  When le participant la valide
11  Then le système refuse avec le <messageErreur>
12 Examples:
13   | course1 | particip1 | messageErreur |
14   | NULL    | Raouf    | course inexistante |
15   | Lyon    | NULL     | participant inexistant |

```