# Tutoriel : Tests unitaires – Partie II

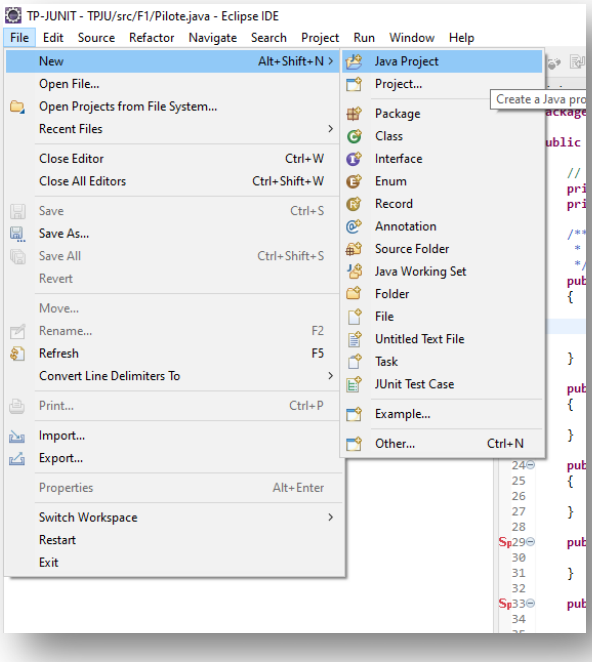
Logiciel : Eclipse
Langage : Java

Groupe :
**DOUR** Marcellino
**PARDINI** Raphaël

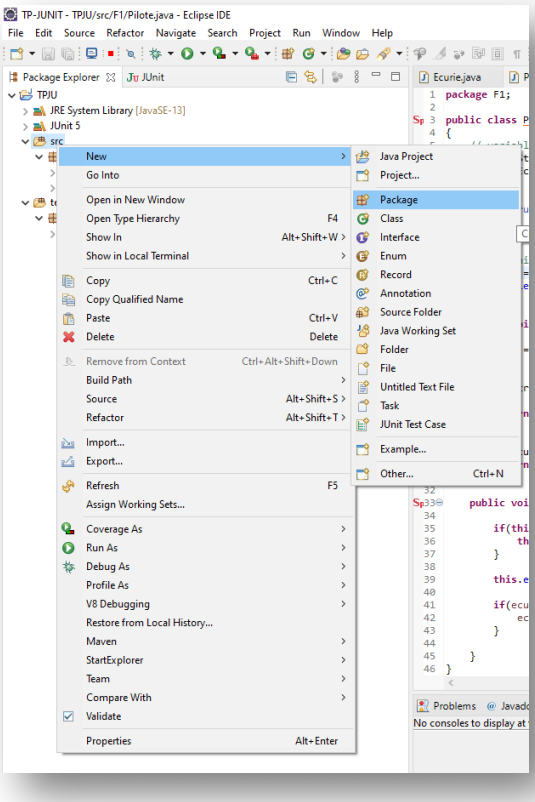
Aujourd'hui, nous vous retrouvons pour la partie 2 de ce tutoriel !
Afin de finir notre application de réservation à temps pour le Grand Chelem, nous allons devoir passer à la vitesse supérieure et utiliser un nouvel outil : Éclipse.

14. Après avoir téléchargé Eclipse, il faut recréer un projet.

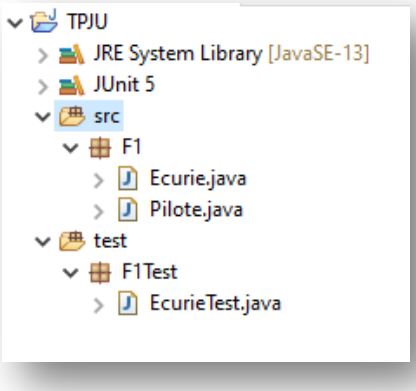
Pour cela, utilisez les commandes suivantes : File > New > Java Project




15. Après avoir créé notre projet, nous voulons récupérer notre travail de la dernière séance sur BlueJ. Pour cela, récupérer le code écrit précédemment et utilisez les commandes suivantes : clic droit sur le dossier « src » > New > Folder (pour créer un dossier) Package (pour créer un paquet) ou Class (pour créer une classe)




Info : Par convention le dossier (« Folder ») source (« src ») et test (« test ») doivent être distingué comme suit :




16. La fédération de formule 1 nous indique qu'il faut ajouter une évolution dans notre projet. En effet, un pilote peut avoir une écurie mais une écurie peut également avoir plusieurs pilotes. Nous allons donc ajouter une liste de pilotes dans les attributs de l'écurie.

Afin de mettre en place la bidirectionnalité « * .. 0.1 », il faut ajouter une dépendance dans chacune des classes.

- Créer un attribut liste de pilotes dans la classe Ecurie.

```java
public class Ecurie
{
    // variables d'instance - remplacez l'exemple qui suit par le vôtre
    private int victoire;
    private String nom;
    private ArrayList<Pilote> pilotes;

    /**
     * Constructeur d'objets de classe Ecurie
     */
    public Ecurie()
    {
        // initialisation des variables d'instance
        this.victoire = 0;
        this.nom = null;
        this.pilotes = new ArrayList<>();
    }
```

Attention, ne pas oublier d'instancier l'attribut dans le constructeur et de créer les méthodes d'ajout et de suppression de pilote (en faisant attention à garantir sa mise à jour dans l'objet pilote associé)

```java
public void addPilote(Pilote p)
{
    if(this.pilotes.contains(p) == false) {
        this.pilotes.add(p);
    }

    if(p.getEcurie() == null) {
        p.setEcurie(this);

    } else if (!p.getEcurie().getNom().equals(this.getNom())){
        p.setEcurie(this);
    }

}

public void removePilote(Pilote p)
{
    if(this.pilotes.contains(p)) {
        this.pilotes.remove(p);
        return;
    }

    if(p.getEcurie() != null) {
        p.setEcurie(null);
    }

}
```

- Créer un attribut ecurie de type Ecurie dans la classe Pilote

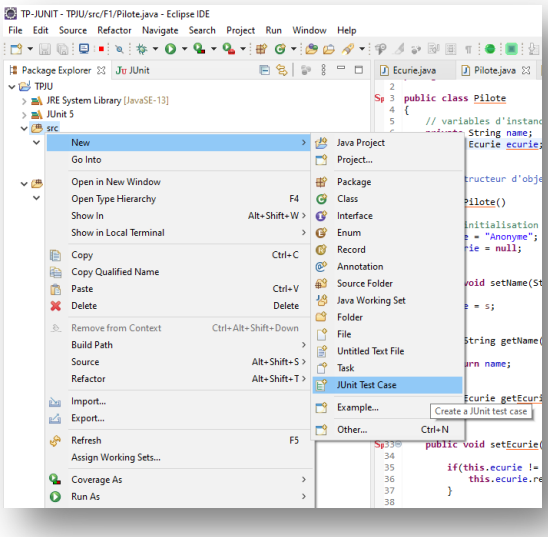
```java
public class Pilote
{
    // variables d'instance - remplacez l'exemple qui suit par le vôtre
    private String name;
    private Ecurie ecurie;

    /**
     * Constructeur d'objets de classe Pilote
     */
    public Pilote()
    {
        // initialisation des variables d'instance
        name = "Anonyme";
        ecurie = null;
    }
```

Attention, ne pas oublier d'instancier l'attribut dans le constructeur (en faisant attention à garantir sa mise à jour dans l'objet écurie associé)

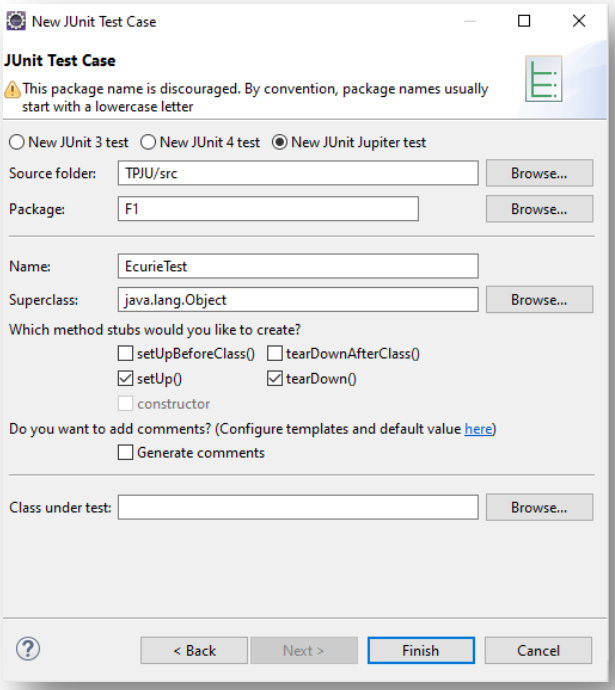
```java
public Ecurie getEcurie() {
    return ecurie;
}

public void setEcurie(Ecurie ecurie) {

    if(this.ecurie != null) {
        this.ecurie.removePilote(this);
    }

    this.ecurie = ecurie;

    if(ecurie != null) {
        ecurie.addPilote(this);
    }

}
```

Testons maintenant cette nouvelle fonctionnalité !

Pour cela, faite un clic droit sur le dossier « src » > New > JUnit Test Case
A noter si ce dernier n'est pas proposé, aller dans « Other » et chercher « JUnit Test Case dans la barre de recherche.




Remplir les champs suivants :




Info : par convention le nom de la classe test est sous format « *ClassTest.java* ».
N'oubliez pas de cocher setUp() et tearDown() afin de paramétrer les manipulations commune à chaque test si besoin (comme nous l'avions déjà fait dans la première partie).

Reprenons l'exemple de la partie 1 :

```java
public class EcurieTest {

    private Pilote pilote1;
    private Ecurie ecurie1;

    /**
     * Constructeur de la classe-test EcurieTest
     */
    public EcurieTest()
    {
    }

    /**
     * Met en place les engagements.
     *
     * Méthode appelée avant chaque appel de méthode de test.
     */
    @BeforeEach
    public void setUp() // throws java.lang.Exception
    {
        pilote1 = new Pilote();
        ecurie1 = new Ecurie();

        pilote1.setName("Carlos");
        ecurie1.setNom("Ferrari");

        pilote1.setEcurie(ecurie1);
    }
```

Ajouter d'autre test pour voir si la bidirectionnalité fonctionne :

- Test d'ajout de Pilote

  Ici, nous allons créer le Pilote Charles, et nous allons vérifier qu'il s'ajoute bien à la listes des pilotes de l'écurie Ferrari.

```java
@Test
public void testAddPilotes()
{
    Pilote pilote2 = new Pilote();
    pilote2.setName("Charles");

    ecurie1.addPilote(pilote2);
    assertEquals(2, ecurie1.getPilotes().size());
    assertEquals("Ferrari", pilote2.getEcurie().getNom());
}
```

- Test de suppression de pilote

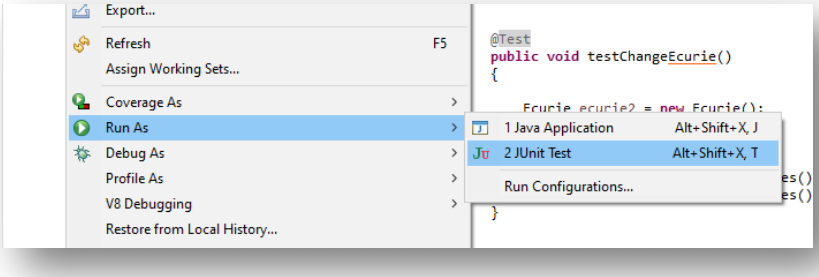
  De même, nous allons tester la méthode de supprésion de la liste.

```java
@Test
public void testRemovePilotes()
{
    Pilote pilote2 = new Pilote();
    pilote2.setName("Charles");

    ecurie1.removePilote(pilote2);
    assertEquals(1, ecurie1.getPilotes().size());
    assertNull(pilote2.getEcurie());
}
```

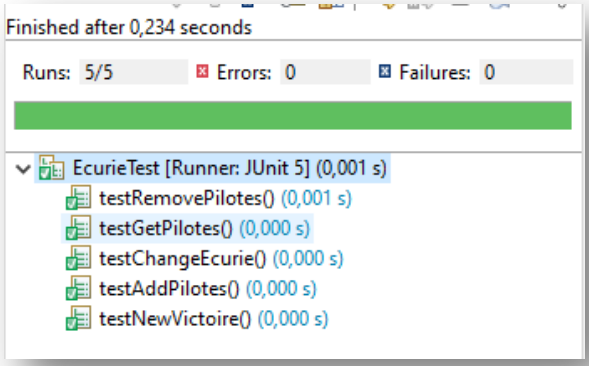
- Test de changement d'écurie

```java
@Test
public void testChangeEcurie()
{

    Ecurie ecurie2 = new Ecurie();
    ecurie2.setNom("Renault");

    pilote1.setEcurie(ecurie2);
    assertEquals(1, ecurie2.getPilotes().size());
    assertEquals(0, ecurie1.getPilotes().size());
}
```

Afin d'exécuter les tests, il faut faire les commandes suivantes :
Clic droit sur le dossier « src » > Run As > JUnit Test

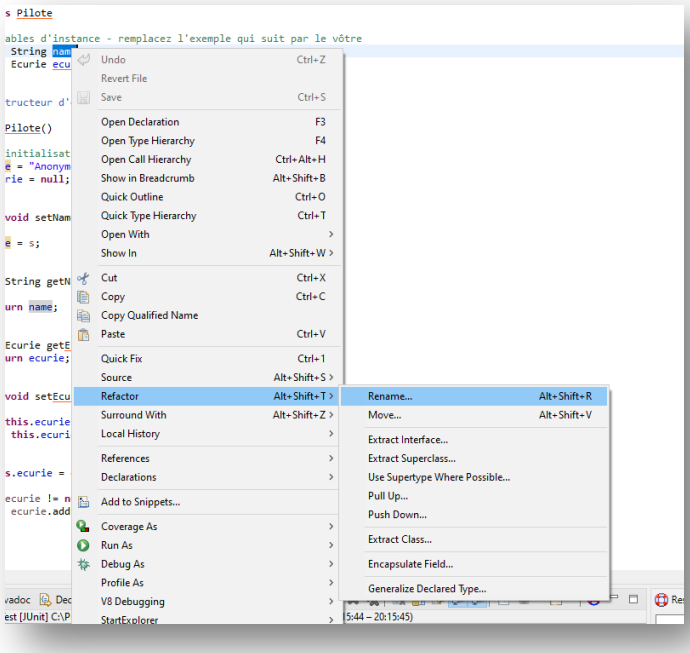



Résultats :



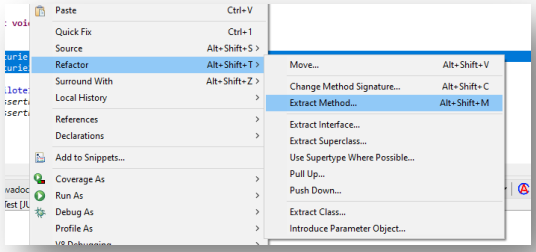

17. Mince les utilisateurs du logiciel seront tous des Français (cocorico !).
Nous devrons changer le nom de l'attributs « name » par nom. Qu'à cela ne tienne ! nous avons la possibilité de changer son nom d'un seul coup par le biais du refactor rename. Pour cela :

Sélectionner l'attribut > clic droit > Refactor > Rename…

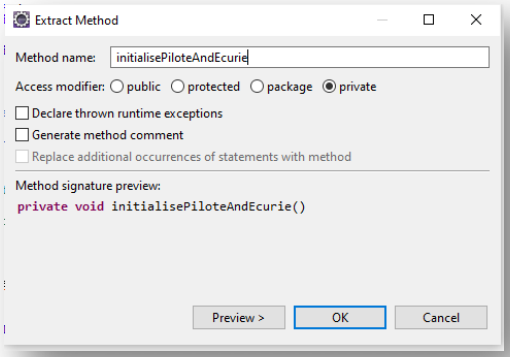



Plus de temps à perdre, nous avons besoin de gagner des secondes. Je vois que l'on répète souvent l'instanciation du coupe écurie et pilote. Et si on créer une méthode ? EN DEUX CLICS ? Oui c'est possible …

Sélectionner les lignes de méthodes dont nous souhaitons générer une méthode > clic droit > Refactor > Extract Method…




Remplir les champs suivants :




Tadaaaa :

```
private void initialisePiloteAndEcurie() {
    pilote1 = new Pilote();
    ecurie1 = new Ecurie();
}
```

18. Afin d'en apprendre plus sur les bonnes pratiques de tests, rendons nous sur le lien suivant : http://junit.sourceforge.net/doc/testinfected/testing.htm

Une bonne pratique est de faire de nombreux tests sur des petits morceaux de codes, de les faire au fur et à mesure des développements et de les exécuter fréquemment.

Dans l'idéal, nous aurions pu d'abord écrire nos tests pour les méthodes d'ajout et de suppression de pilote dans la liste, puis après écrire le contenu de ces méthodes. Ainsi, on définit par avance l'attendu et on s'assure que nos développements seront valides !

19. Notre responsable nous demande d'exécuter nos tests afin de lui montrer que notre travail est fini. Cependant, il ne dispose pas d'Eclipse sur son ordinateur mais seulement d'un terminal. Pour exécuter JUinit en ligne de commande, il faut écrire les lignes suivantes :

Pour compiler le fichier → `javac –cp junit–4.0.0.jar;. JUnitProgram.java`

Ici, javac est le compilateur Java qui utilise l'option -cp.

La commande javac -cp recherche les paramètres suivants :

1. Le fichier jar JUnit est suivi d'un point-virgule.
2. Le chemin du répertoire dans lequel se trouve le fichier source.
3. Le nom du fichier de classe

Pour interpréter le fichier → `java –cp junit–4.0.0.jar;. JUnitProgram nomClasse`

20. Voici la version de George Nichols de la loi de Murphy :« Si cela peut mal se passer, cela arrivera ».
    Durant notre périple, nous avons vu fait des tests au fur et à mesure du développement. En effet, la probabilité qu'il y ai des bugs dans nos développements est très élevé, et s'il y a une possibilité que notre programme marche mal, cela va arriver. Faire des tests au fils de l'eau permet donc d'identifier les problèmes au plus tôt et de la manière la plus efficace. Ainsi, la loi de Murphy nous permet de garder en tête l'importance des tests.