

## TP5-6

# Test avec JUnit et Test Driven Development

## TP 5 : Mise en place de JUnit

### 5.1 Objectif

L'objectif de ce TP est de tester votre package `fr.univrennes.istic.l2gen.geometrie` à l'aide de JUnit.

### 5.2 Installation

Vérifier que JUnit est correctement installé dans VS Code. Sinon, l'installer.

### 5.3 Création d'un dossier de test (un seul membre de l'équipe)

A la racine de votre projet, c'est-à-dire au même niveau que votre dossier `src`, créer un dossier `test`. Dans celui-ci, créer un dossier `fr/univrennes/istic/l2gen/geometrie` afin de permettre d'ajouter vos classes de test au package `fr.univrennes.istic.l2gen.geometrie`. Votre hiérarchie de packages de tests devra toujours suivre une forme parallèle à la hiérarchie du dossier `src`.

### 5.4 Partage du dossier test

La personne ayant créé le dossier `test` ci-dessus l'enregistre dans git (`add`, `commit` puis `push`) comme vu au TP précédent. Les autres font un **pull** sur le projet et vérifient qu'ils ont bien récupéré ce dossier `test` et son contenu.

### 5.5 Écriture des tests unitaires

**Question 1** Répartissez-vous le travail pour que pour chacune des classes de votre package `fr.univrennes.istic.l2gen.geometrie`, il soit créé dans `test/fr/univrennes/istic/l2gen/geometrie` une classe de test JUnit.

Pour accélérer la saisie, vous pouvez demander à VS Code de générer le squelette d'une classe de test pour vous. Pour cela, dans l'éditeur d'une classe (par exemple `Cercle`) faites : `click droit`>`source Action`>`generate tests`. Déplacer alors la classe obtenue (`CercleTest`) dans le dossier `fr/univrennes/istic/l2gen/geometrie`.

Par exemple pour la classe `Cercle` on devra avoir une classe `CercleTest` ressemblant à :

```
package fr.univrennes.istic.l2gen.geometrie;
import static org.junit.Assert.*;
import org.junit.Test;

public class CercleTest {
    @Test
```

```

    public void testDescription() {
        //...
    }

    @Test
    public void testDupliquer() {
        //...
    }

    @Test
    public void testEnSVG() {
        //...
    }
//...
}

```

Ecrire le contenu de ces tests en suivant les instructions vues en cours, en particulier en se servant des différentes formes d'`assert` disponibles dans `org.junit.Assert.*`. Toutes les méthodes de chaque classe doivent être testées ainsi.

Pensez notamment à utiliser les préluces de test pour factoriser les parties communes, par exemple pour Cercle :

```

    Cercle c;
    @Before
    public void setUp() {
        c = new Cercle(100,100,50);
    }
}

```


## 5.6 Exécution des tests unitaires

Lancez vos tests à l'aide de 

Si un test montre une erreur, celle-ci peut venir soit du test soit du code de la classe testée. Corrigez toutes les erreurs puis enregistrez vos modifications dans gitlab. Corrigez d'éventuels conflits. Puis mettez à jour avec `git pull`.

**Important :** On ne supprime/commente jamais un test quand celui-ci passe!

JUnit est un outil de tests unitaires **automatisés** :

- Chaque test est écrit une seule fois, mais il peut-être relancé à volonté (en particulier après toute modification) 
- Les tests sont sauvegardés/versionnés sur le GitLab du projet

Un test est plus précieux (et pérenne) qu'une ligne de code, car il va dans le futur servir de test de non-régression. Un test de non-régression vérifie en effet que l'ajout de nouvelles fonctionnalités dans un programme ne perturbe pas les fonctions initiales.

## TP 6 : Test Driven Development

### 6.1 Objectif

L'objectif de ce TP est d'utiliser l'approche Test Driven Development vue en cours pour commencer à développer un nouveau package Java permettant de représenter graphiquement des données statistiques. Lisez le texte de ce sujet de TP jusqu'au bout, puis prenez 5 minutes pour vous répartir le travail au sein de votre équipe.

**Rappel :** Pour ajouter une fonctionnalité X en TDD :

1. Ecrire le(s) test(s) montrant que X fonctionne (dans la classe `ClasseXTest` du répertoire `test`)
2. Lancer JUnit et **vérifier que le nouveau test échoue** (X n'est pas encore développée). Ceci permet de vérifier que le test est valide!
3. Ecrire le code nécessaire pour passer le test (**et pas plus**)
4. Lancer JUnit et **vérifier que le test passe**, sinon retourner en 3.
5. Si nécessaire, remaniez le code pour le simplifier (principe KISS : "Keep It Simple Stupid")

### 6.2 TDD de la méthode `tourner(int angle)`

**Question 2** Commencer par utiliser cette approche TDD pour ajouter dans `IForme` et toutes ses sous-classes la méthode `tourner(int angle)`, qui effectue une rotation de la figure d'une valeur d'angle exprimée en degrés.

Dans chacune de vos classes de test, vous devrez donc commencer par écrire une méthode de test :

```
@Test
public void testTourner() {
    // metre ici votre code de test de l'appel de la methode tourner(int)
    // faisant appel a description() pour son Oracle
}
```

Répartissez-vous le travail d'écriture de ces tests et de l'ajout de fonctionnalités, car il faudra modifier partout les méthodes `dupliquer()`, `description()` et `enSVG()` pour tenir compte de cette nouvelle fonctionnalité. Pour le SVG, il suffit d'ajouter (seulement si l'angle est différent de 0) le texte suivant : `transform="rotate(angle)"`. Dans la classe `Groupe`, se contenter de déléguer la méthode `tourner` aux objets du groupe (comme pour la méthode `déplacer`).

### 6.3 TDD de la méthode `aligner()`

**Question 3** Ajouter dans `IForme` une méthode `default IForme aligner(Alignement alignement, double cible)`, qui aligne le HAUT (ou le BAS, la DROITE ou la GAUCHE) de cette forme sur l'axe cible, i.e. la droite  $y=cible$  (pour les cas HAUT et BAS) ou la droite  $x=cible$  (pour les cas GAUCHE et DROITE).

Pour cela on ajoutera au package `geometrie` un Enum :

```
public enum Alignement {
    HAUT, BAS, DROITE, GAUCHE
}
```

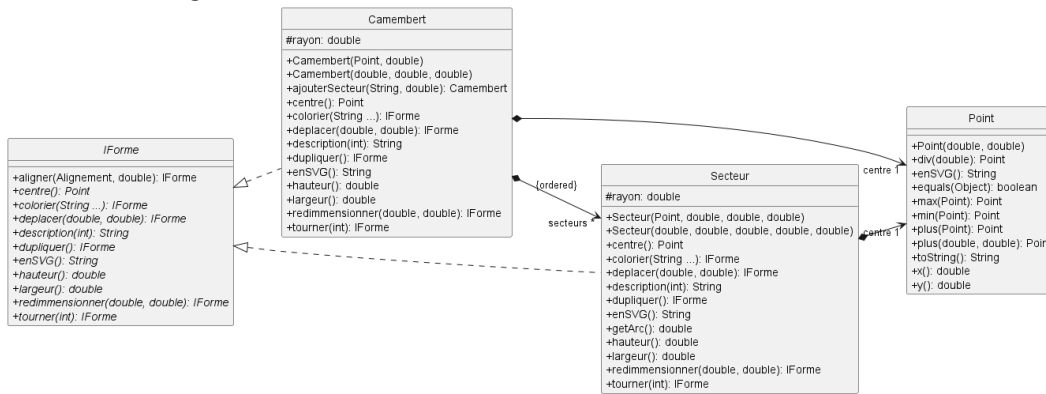
et on ajoutera le test de cette méthode dans les différentes classes de test des classes implémentant `IForme`.

### 6.4 TDD de la classe Camembert

#### Initialisation d'un nouveau package

Un membre de l'équipe doit créer un nouveau package `fr.univrennes.istic.l2gen.visustats` dans le dossier `src`, et y ajouter une classe `Camembert` qui implemente `IForme`.

Un autre membre de l'équipe fait de même dans votre dossier `test` : package `fr.univrennes.istic.l2gen.visustats` et classe JUnit `CamembertTest`.

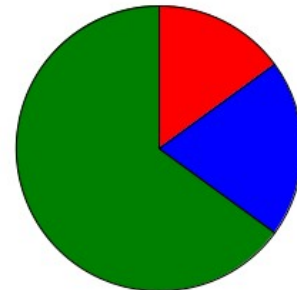


**Question 4** Après la synchronisation habituelle de votre équipe sur votre GitLab, répartissez-vous le travail pour réaliser les tests de `Camembert` dans `test` et en parallèle avec son implantation dans `src` selon l'architecture ci-dessus.

Le code ci-dessous devra produire la figure ci-contre en SVG :

```

Camembert c = new Camembert(110,110, 100);
c.ajouterSecteur("red", 0.15);
c.ajouterSecteur("blue", 0.2);
c.ajouterSecteur("green", 0.65);
    
```



### 6.5 TDD de la classe Faisceau

**Question 5** Commencer par ajouter en mode TDD dans la classe `Groupe` les méthodes suivantes :

```

/**
 * Vide ce groupe de tous ses elements
 * @return IForme : ce groupe qui ne contient plus aucun element
 */
public IForme vider() {
    //..
}
/**
 * @param alignement direction HAUT, BAS, DROITE, GAUCHE tc.
 * @param cible ligne horizontale ou verticale sur laquelle
 * doivent s'aligner chacun des elements du groupe
    
```

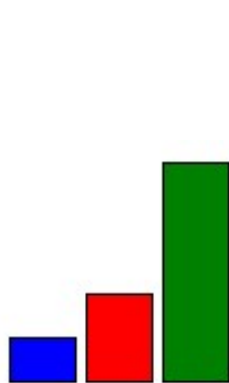
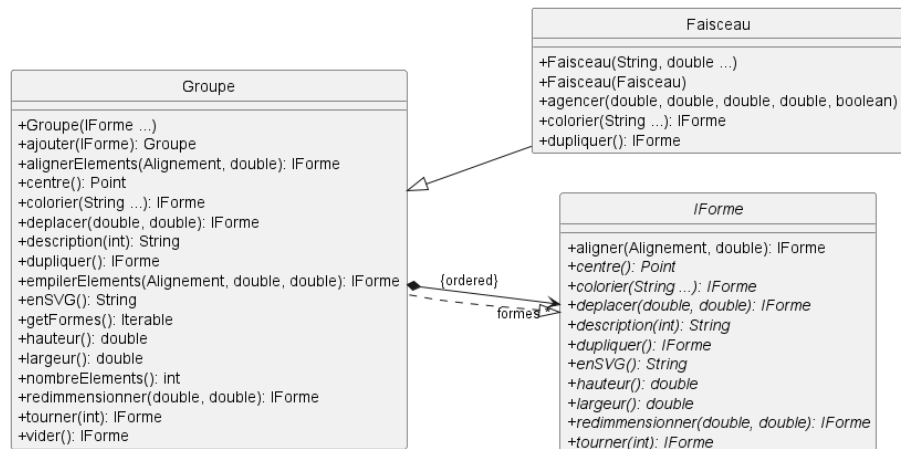
```

* @return IForme
*/
public IForme alignerElements(Alignement alignement, double cible) {
//...
}

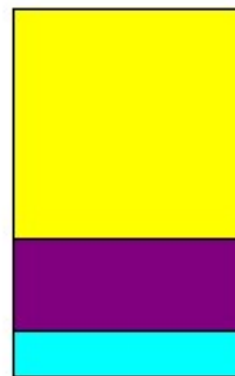
/**
 * @param alignement direction HAUT, BAS, DROITE, GAUCHE tc.
 * @param cible ligne horizontale ou verticale sur laquelle
 * doivent s'empiler chacun des elements du groupe
 * @param separation : distance entre chaque element empile
 * @return IForme
 */
public IForme empilerElements(Alignement alignement, double cible, double separation) {
}

```

**Question 6** Grâce à ces méthodes `alignerElements` et `empilerElements`, créez dans `visustats` une nouvelle classe appelée `Faisceau` qui permet d'organiser un groupe de barres (i.e. des `Rectangles` de largeur identique) soit les unes à cotés des autres, soit de les empiler comme illustré ci-dessous.



Exemple de Faisceau horizontal



Exemple de Faisceau vertical

La méthode `agencer()` règle la manière dont les barres d'un faisceau doivent être présentées :

```

/**
 * @param axeX Axe sur lequel aligner la gauche du faisceau
 * @param axeY Axe sur lequel aligner le bas du faisceau
 * @param largeur largeur totale du faisceau
 * @param echelle facteur d'echelle vertical pour les differentes barres
 * @param verticalement selon que les barres sont empilees
 *           verticalement ou horizontalement
 */
public void agencer(double axeX, double axeY, double largeur,
                   double echelle, boolean verticalement)

```

Par exemple les 2 faisceaux ci-dessus sont obtenus de la manière suivante :

```

Faisceau fh = new Faisceau("Exemple_de_Faisceau_horizontal", 100, 200, 500);
fh.colorier("blue","red","green");
fh.agencer(20, 250, 100, 0.2, false);
Faisceau fg = new Faisceau("Exemple_de_Faisceau_vertical", 100, 200, 500);
fg.colorier("cyan","purple","yellow");
fg.agencer(20, 250, 100, 0.2, true);

```

**Attention :** A l'issu de ce TP, votre encadrant vous fera un premier retour sur la complétude et la qualité de votre code tel que trouvé dans votre dépôt GitLab. Il est donc impératif de respecter le nommage de tous les éléments d'interface.