Départ. Informatique

BUT ASPE – S1.10. Qualité de Développement

V. Deslandres

TD 4 – Tests unitaires, TDD, refactoring, GIT

Contexte

Les **objectifs pédagogiques de ce TP pour le cours de Qualité de Développement** sont les suivants :

- Maîtriser les tests unitaires.
- Appréhender le concept de programmation dirigée par les tests (TDD, Test Driven Development).
- Apprendre à mieux maîtriser les différents aspects de Netbeans (ou de tout IDE équivalent).
- Utiliser Git.

Ce TP est à faire seul ou en binôme.

Mise en place

1. Créez un projet sur la forge de l'IUT nommé `TPBibliotheque.`

2. Ajoutez votre binôme si nécessaire, et l'enseignant (v.deslandres). Clonez votre projet et importez-le sur NetBeans.

A partir d'ici, vous devez faire au moins un commit par question !

Vous pouvez en faire plus si cela vous parait pertinent.

N'oubliez pas de push de temps en temps pour sauvegarder votre travail.

Partie 1. Création de la classe Personne

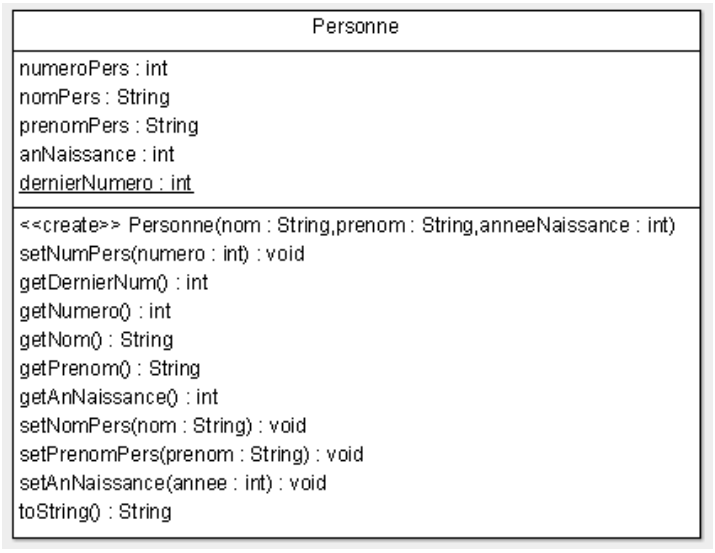
1. Squelette de la classe

Dans le package `bibliotheque`, créez une classe `Personne.java` selon les spécifications données ci-dessous. **N'implémentez pas les méthodes**. Contentez-vous de retourner des valeurs par défaut lorsque cela est nécessaire. Assurez-vous qu'aucune erreur n'est présente à la fin de la création de la classe (pas de symbole de Warning ou d'erreur dans la marge, ni auprès du nom du fichier dans l'explorateur de projet).

Attention : plus tard dans le TP, vous devrez générer la Javadoc. Pensez-y dès le début, sinon, ce sera très fastidieux.

Pendant que vous implémentez votre méthode, observez quelques outils que vous offre votre IDE (Netbeans ou autre) :

- Auto-complétion : lorsque vous créez une méthode, l'accolade fermante s'insère automatiquement.
- Aide à la génération de la Javadoc : lorsque vous tapez le début d'un commentaire de Javadoc, les paramètres sont générés automatiquement.
- Suggestions d'implémentation : Netbeans propose de définir comme `Final` certaines variables. Pourquoi ?



2. Classe de test

Créez une classe de test pour la classe `Personne.java`. Vous pouvez choisir d'utiliser les fonctionnalités de génération automatique, ou bien le faire à la main, mais dans tous les cas, prenez bien garde à séparer le code de test du code métier. Le contenu de votre classe de test doit être le suivant. :

PersonneTest.java

```
package bibliotheque;

import org.junit.After;
```

```java
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class PersonneTest {

    public PersonneTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of getNumero method, of class Personne.
     */
    @Test
    public void testGetNumero() {
        fail("Test à implémenter plus tard");
    }

    /**
```

```
     * Test of getNom method, of class Personne.
     */
    @Test
    public void testGetNom() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        assertEquals("Turing", alan.getNom());
    }

    /**
     * Test of getPrenom method, of class Personne.
     */
    @Test
    public void testGetPrenom() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        assertEquals("Alan", alan.getPrenom());
    }

    /**
     * Test of getAnNaissance method, of class Personne.
     */
    @Test
    public void testGetAnNaissance() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        assertEquals(1912, alan.getAnNaissance());
    }

    /**
     * Test of getDernierNum method, of class Personne.
     */
    @Test
    public void testGetDernierNum() {
        fail("Test à implémenter plus tard");
    }
```

```java
    /**
     * Test of setNumPers method, of class Personne.
     */
    @Test
    public void testSetNumPers() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        alan.setNumPers(18);
        assertEquals(18, alan.getNumero());
    }

    /**
     * Test of setNomPers method, of class Personne.
     */
    @Test
    public void testSetNomPers() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        alan.setNomPers("Minsky");
        assertEquals("Minsky", alan.getNom());
    }

    /**
     * Test of setPrenomPerso method, of class Personne.
     */
    @Test
    public void testSetPrenomPers() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        alan.setPrenomPers("Marvin");
        assertEquals("Marvin", alan.getPrenom());
    }

    /**
     * Test of setAnNaissance method, of class Personne.
     */
    @Test
    public void testSetAnNaissance() {
        Personne alan = new Personne("Turing", "Alan", 1912);
```

```
        alan.setAnNaissance(1990);
        assertEquals(1990, alan.getAnNaissance());
    }

    /**
     * Test of toString method, of class Personne.
     */
    @Test
    public void testToString() {
        Personne alan = new Personne("Turing", "Alan", 1912);
        assertEquals("Turing, Alan, 1912", alan.toString());
    }

}
```

Notez que pour l'instant, on ne se préoccupe pas de la gestion du numéro de personne. **Exécutez les tests et observer qu'aucun d'eux ne passe.**

Si, au moment de l'exécution, votre projet contient des erreurs, c'est que vous avez manqué une étape. Corrigez les erreurs avant d'aller plus loin.

3. Implémentation de la classe

Implémentez les méthodes de la classe Personne de sorte que tous les tests passent (sauf les tests relatifs à la gestion des numéros). Ajoutez l'annotation `@Ignore` (`@Disabled` en JUnit5) pour ignorer les deux tests que l'on ne souhaite pas gérer tout de suite. Observez :

1. Comment gérer l'import nécessaire ?
2. Quel est l'impact de @Ignore sur le résultat de l'évaluation des tests ?

4. Numéro de personne

Nous allons maintenant traiter le cas du numéro de personne.

- Quelles sont les différentes stratégies pour gérer le numéro de personne ? Quels sont les avantages et les inconvénients de chacune ?
- Concevez et écrivez les tests permettant de vérifier le bon fonctionnement de l'affectation des numéros de personne.
- Implémentez les méthodes correspondantes
- Mettez à jour la méthode toString() de Personne pour afficher les n° de personne, et mettez également à jour le test correspondant.

Partie 2. Gestion d'une liste de personnes

1. Squelette de la classe

Créez une nouvelle classe `ListePersonnes.java` dont la mission sera de gérer une liste de `Personne`. Comme dans la question 2, n'implémentez pas les méthodes. Contentez-vous d'écrire un squelette fonctionnel. La classe doit contenir un constructeur, une méthode ajouter qui permet d'ajouter une personne à la liste, et une méthode appartient, qui permet de vérifier qu'une personne appartient bien à la liste.

On peut légitimement se demander comment implémenter la méthode “appartient”. En effet, soit on considère que l'on doit vérifier si une `Personne p` appartient à la liste, ce qui est trivial à implémenter, mais pas forcément très utile dans la pratique, soit on considère que le problème est de savoir si une personne dont on connaît le nom et le prénom appartient à la liste… Ce deuxième cas est un peu plus difficile, mais c'est probablement ce qui va intéresser les utilisateurs de notre méthode ! Faites-donc les deux !

2. Classe de test

Cette fois-ci, c'est votre tour d'écrire les tests de cette classe. Générez la classe de test (manuellement, ou automatiquement, à votre convenance). Si vous la générez automatiquement, observez qu'il est parfois nécessaire de modifier le code généré ;) Une fois que vous avez fini d'écrire les tests, vérifiez qu'ils fonctionnent et qu'ils échouent (puisque le code métier n'est pas encore écrit).

3. Implémentation de la classe

Complétez maintenant votre classe `ListePersonnes` pour qu'elle passe les tests.

4. Javadoc

Il est temps de générer la **javadoc** pour vous assurer que le processus fonctionne correctement (sur le projet : clic droit - *Generate Javadoc*). Vous devrez la mettre à jour plus tard. Que se passe-t-il pour les tests : est-ce qu'une javadoc est générée ?

Partie 3. Création de la classe Livre

1. Squelette de la classe

Nous allons maintenant nous occuper de la classe `Livre.java`. Comme pour la classe `Personne.java`, c'est à vous d'écrire le squelette de la méthode selon les spécifications ci-dessous :

Livre
private int NumLivre private String titre private int nombreDePages private Personne auteur private static int dernierNum
public Livre(String titre, int nbpages, Personne auteur) public int getNumLivre() public String getTitre() public int getNombreDePages() public Personne getAuteur() public void setNumLivre(int numero) public void setTitre(String titre) public void setAuteur(Personne p) public void setNombreDePages(int nbPages) public String toString()

2. Implémentation et classe de test

Il est temps de finir la classe `Livre`. Récupérez la classe de test ci-dessous, complétez-là si vous le jugez utile, puis implémentez la classe livre de sorte à ce que tous les tests passent.

LivreTest.java

```
/**
 * Bibliothèque
 */
package bibliotheque;

import org.junit.After;
import org.junit.AfterClass;
```

```java
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;


/**
 * Classe LivreTest
 */
public class LivreTest {

    static Personne alan;
    static Livre computing;

    /**
     *
     */
    public LivreTest() {
    }

    /**
     * Création d'une personne et d'un livre à l'initialisation des tests.
     * Ces éléments ne seront pas modifiées par la suite.
     */
    @BeforeClass
    public static void setUpClass() {
        alan = new Personne("Turing", "Alan", 1912);
        computing = new Livre("Computing Machinery and Intelligence", 250,
alan);
    }

    /**
     *
     */
    @AfterClass
    public static void tearDownClass() {
    }
```

```java
/**
 *
 */
@Before
public void setUp() {
}

/**
 *
 */
@After
public void tearDown() {
}

/**
 * Test de getNumLivre de la classe Livre.
 */
@Test
public void testGetNumLivre() {
    assertEquals(0, computing.getNumLivre());
}

/**
 * Test de getNumLivre de la classe Livre.
 */
@Test
public void testGetNumLivreTwoBooks() {
    Livre mind = new Livre("mind", 0, alan);
    Livre mind2 = new Livre("mind2", 0, alan);
    assertEquals(mind.getNumLivre()+1, mind2.getNumLivre());
}

/**
 * Test de getTitre de la classe Livre.
 */
```

```java
    @Test
    public void testGetTitre() {
        String titreAttendu = "Computing Machinery and Intelligence";
        assertTrue(computing.getTitre().equals(titreAttendu));
    }

    /**
     * Test de getNombreDePages de la classe Livre.
     */
    @Test
    public void testGetNombreDePages() {
        assertEquals(250, computing.getNombreDePages());
    }

    /**
     * Test de getAuteur de la classe Livre.
     */
    @Test
    public void testGetAuteur() {
        assertEquals(alan, computing.getAuteur());
    }

    /**
     * Test de setTitre de la classe Livre.
     */
    @Test
    public void testSetTitre() {
        Livre mind = new Livre("?", 0, alan);
        mind.setTitre("Mind");
        assertTrue(mind.getTitre().equals("Mind"));
    }

    /**
     * Test de setAuteur de la classe Livre.
     */
    @Test
```

```
    public void testSetAuteur() {
        Livre ged = new Livre("?", 0, alan);
        Personne douglas = new Personne("Hofstadter", "Douglas", 1945);
        ged.setAuteur(douglas);
        assertTrue(ged.getAuteur().equals(douglas));
    }

    /**
     * Test de setNombreDePages de la classe Livre.
     */
    @Test
    public void testSetNombreDePages() {
        Livre mind = new Livre("?", 0, alan);
        mind.setNombreDePages(500);
        assertEquals(500, mind.getNombreDePages());
    }

    /**
     * Test de toString de la classe Livre.
     */
    @Test
    public void testToString() {
        String expStr = "Computing Machinery and Intelligence, Alan Turing,
250p.";
        assertTrue(computing.toString().equals(expStr));
    }
}
```

Partie 4. La pause s'impose

Dans la prochaine partie, vous allez passer à la gestion de la bibliothèque. Avant cela, il est nécessaire de prendre un peu de recul sur ce que vous avez fait, de vérifier que tout est codé correctement, etc. Prenez donc quelques minutes pour :

- Vérifier que les parties cruciales de votre code sont commentées correctement
- Vérifier que la Javadoc de ce qui est implémenté est correcte et complète
- Vérifier que vous n'avez pas oublié des tests importants

- Vérifier que vous respectez bien les bonnes pratiques de programmation (pensez au “refactor” et au “format”), dont
    - L’indentation correcte du code
    - Le respect des conventions de nommage des variables
    - Le non “mixage” du français et de l'anglais dans votre code
- Vérifier que tous les tests passent, après les avoir refactorés au besoin
- Faire un *commit* de cette version “stable” de votre projet.

Partie 5. Gestion de la bibliothèque

Passez maintenant à l'implémentation de la bibliothèque selon la méthode que vous préférez. Pour mémoire : une bibliothèque est une structure dans laquelle on peut mettre des livres. La classe `Bibliotheque` doit proposer des méthodes pour ajouter un livre, afficher le contenu de la bibliothèque, rechercher par auteur, et rechercher par titre. Vous devez gérer les exceptions pour ces deux derniers cas (i.e. quand le livre n'est pas trouvé).

Vous devez adopter une démarche TDD pour réaliser cette partie :

- Création du squelette de la classe puis commiter
- Écriture des tests unitaires puis commiter
- Complétion de la classe puis commiter
- Exécution des tests
- Refactoring jusqu'à ce que la classe soit complète et fonctionnelle puis commiter.

Pensez à documenter votre code !

Partie 6. Finalisation du projet

A nouveau, prenez le temps d'inspecter votre code. Refactorez ce qui nécessite de l'être et vérifiez grâce aux tests que vous n'avez rien cassé.

Lorsque vous aurez terminé votre projet, il ne vous restera plus qu'à mettre à jour la Javadoc, pour prendre en compte toutes les nouvelles améliorations que vous avez apportées.

De plus, si vous ne l'avez pas encore fait, il est probablement temps de pousser votre historique de *commits* sur un dépôt distant.