

Tutoriel du projet du module Agilité

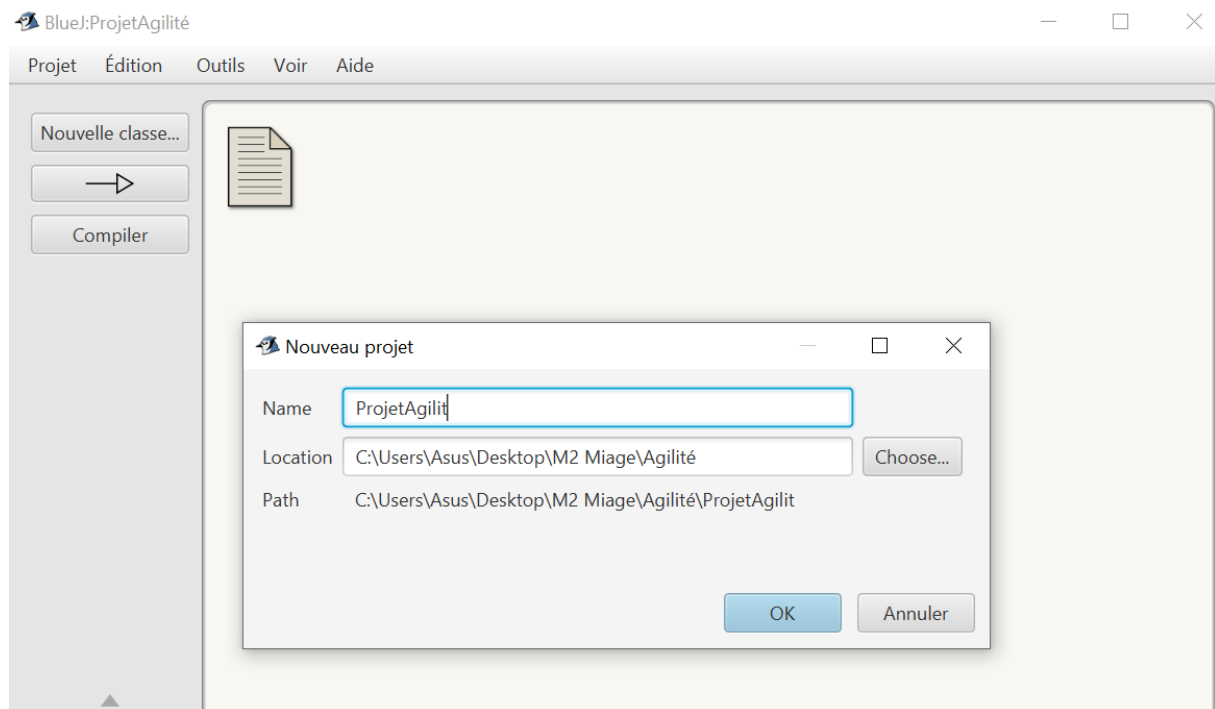
Introduction

Bienvenue dans la Bibliothèque Virtuelle, où la gestion des livres devient une expérience éducative et interactive ! Ce projet vous invite à explorer les bases de la programmation orientée objet en Java, tout en apprenant à manipuler des collections de données et à garantir la fiabilité de votre code grâce à des tests unitaires rigoureux. À travers ce tutoriel, vous découvrirez comment construire et tester un système de gestion de bibliothèque qui pourrait être le cœur de toute application de gestion de livres. Plongez dans cet univers littéraire numérique et préparez-vous à transformer chaque ligne de code en une brique solide de votre propre bibliothèque virtuelle. Que l'aventure commence !

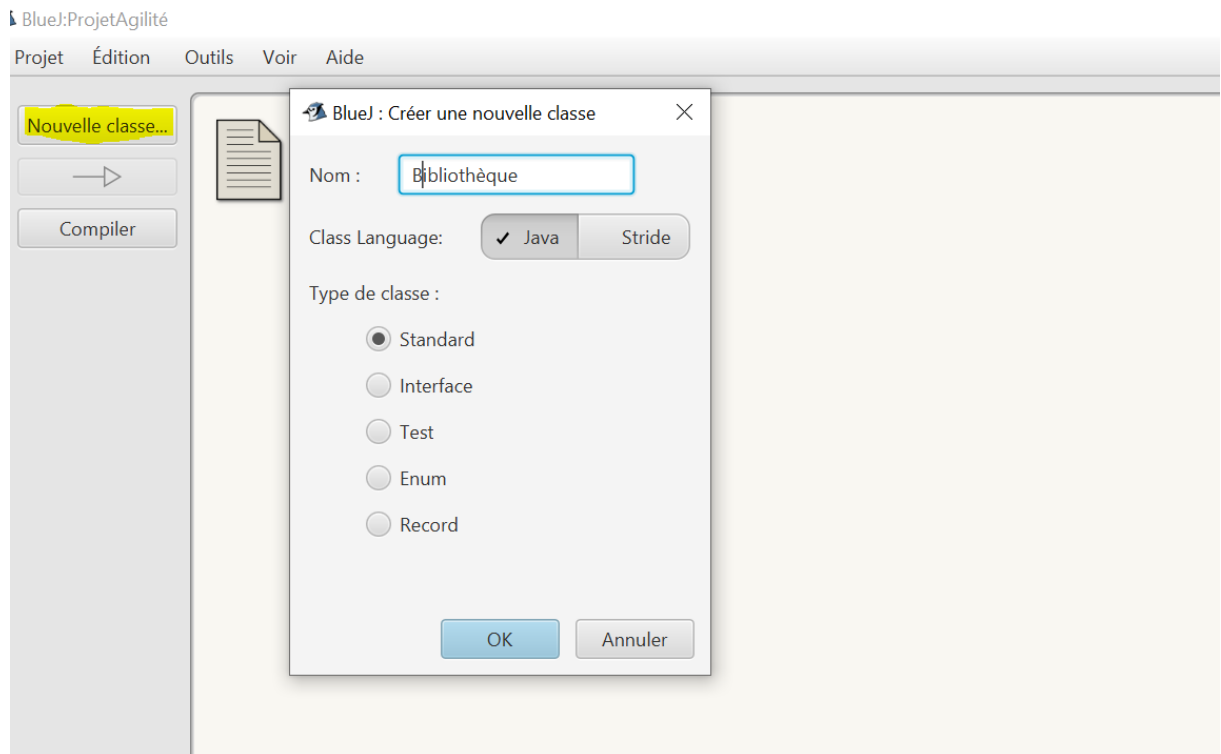
Ce projet se distingue par son approche pédagogique, vous guidant pas à pas à travers les concepts essentiels tout en illustrant chaque étape avec des exemples concrets. Vous apprendrez non seulement à créer des composants réutilisables et modulaires, mais aussi à adopter les bonnes pratiques du développement piloté par les tests (TDD) et du développement piloté par le comportement (BDD). Ce tutoriel s'adresse aussi bien aux débutants souhaitant comprendre les fondamentaux qu'aux développeurs plus expérimentés désireux d'affiner leurs compétences en matière de tests et de conception logicielle.

Partie 1 : BlueJ

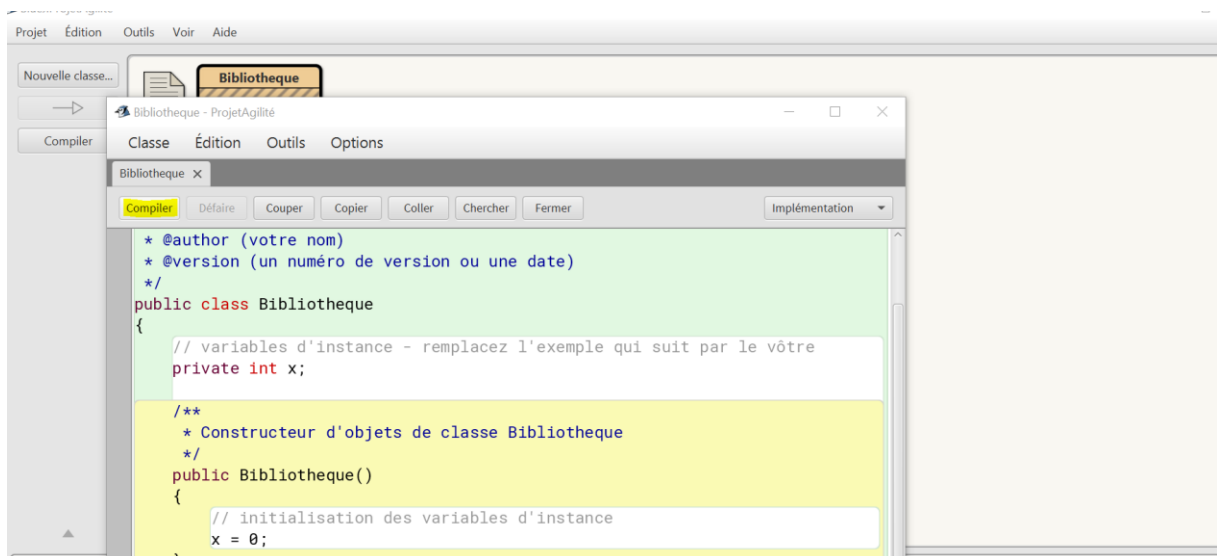
- 1) Tout commence par la création d'un nouveau projet sur BlueJ, cela en cliquant sur projet -> Nouveau projet



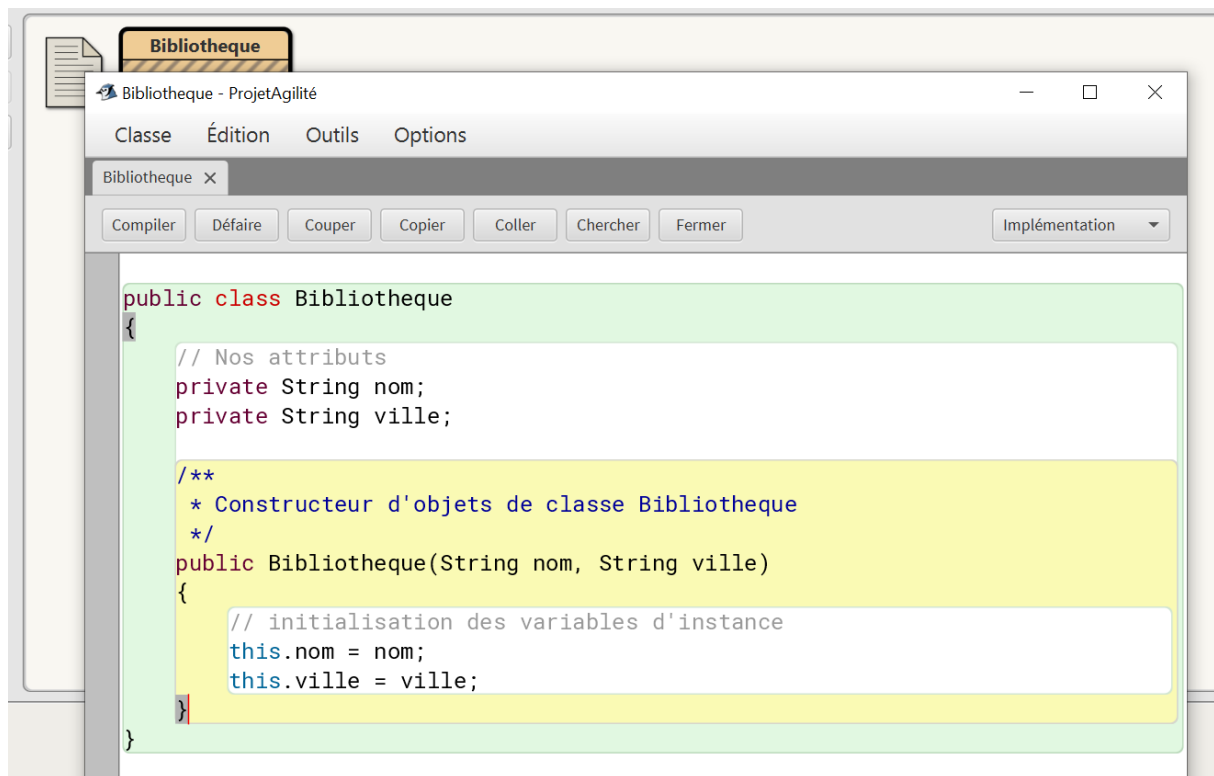
- 2) Ensuite vous allez créer votre classe maîtresse en appuyant sur « Nouvelle classe » puis en remplissant les champs qui apparaissent dans la fenêtre



- 3) Après la création de la classe, vous pourriez la compiler en double cliquant sur la classe créée et puis en cliquant sur le bouton « Compiler »



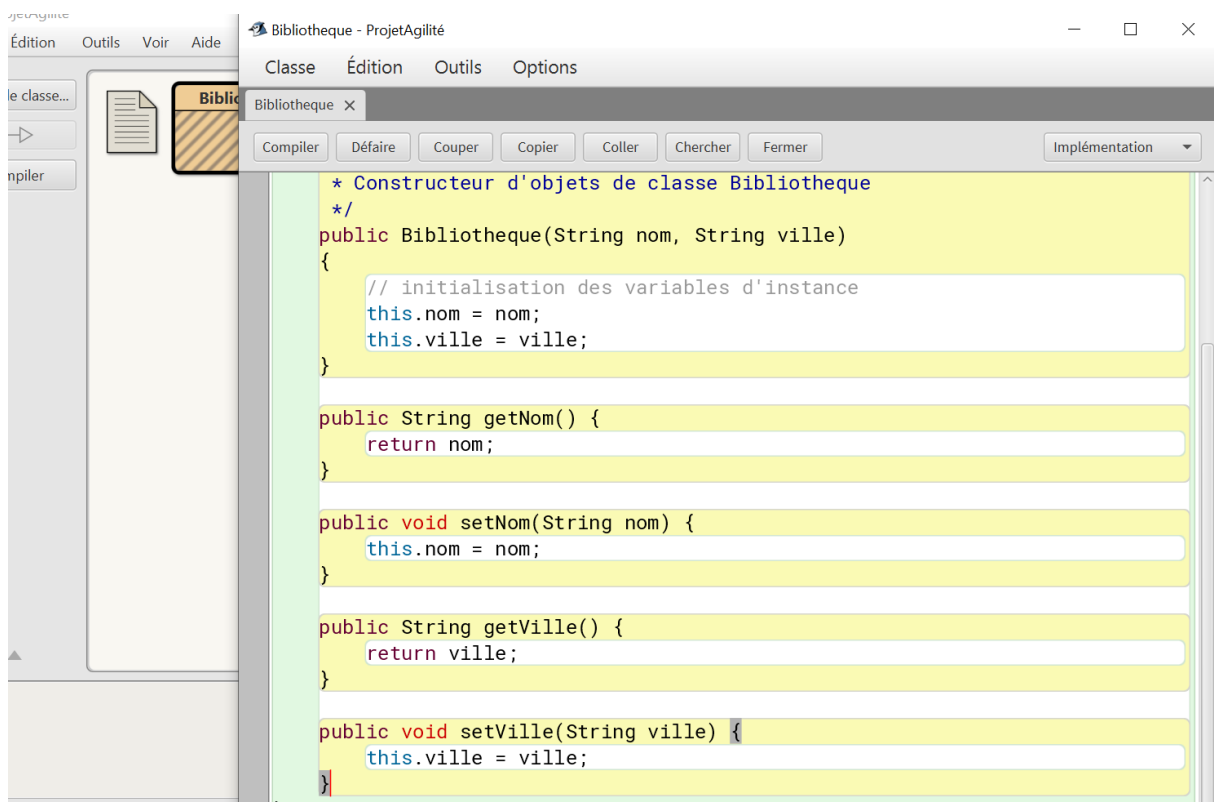
- 4) Maintenant en double cliquant une autre fois sur la classe créée, vous allez remplacer le code apparent par le code suivant :



Les premières lignes représentent les attributs de notre classe

La fonction créée représente le rôle de « constructeur »

- 5) Maintenant vous allez créer pour chaque « attribut » une méthode de type « getter » qui retourne l'objet en soit et une méthode de type « setter » qui peut modifier l'attribut en soit.



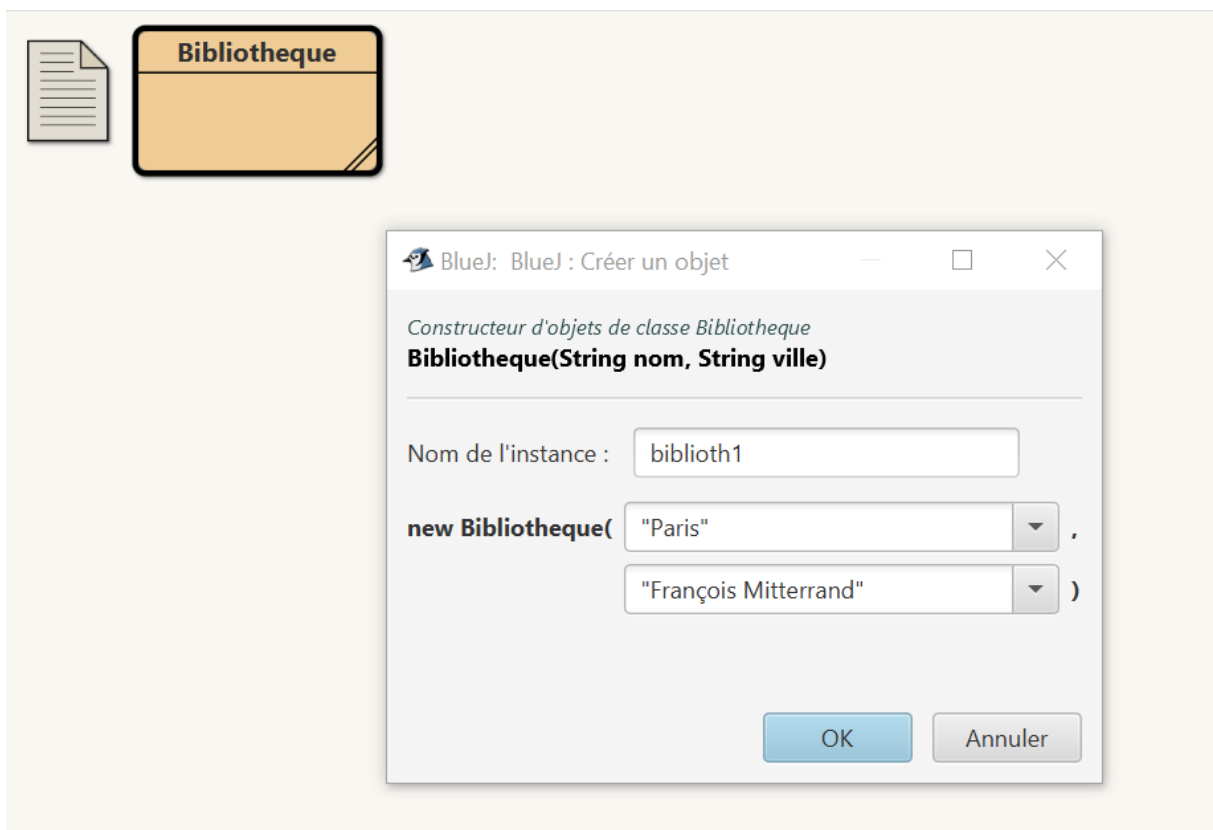
- 6) Ensuite vous pourriez rajouter une méthode qui manipulera les attributs créés en rajoutant le code suivant après avoir double cliquer sur le rectangle « Bibliothèque »

```
public void villeToUpperCase() {  
    this.ville = this.ville.toUpperCase();  
}
```

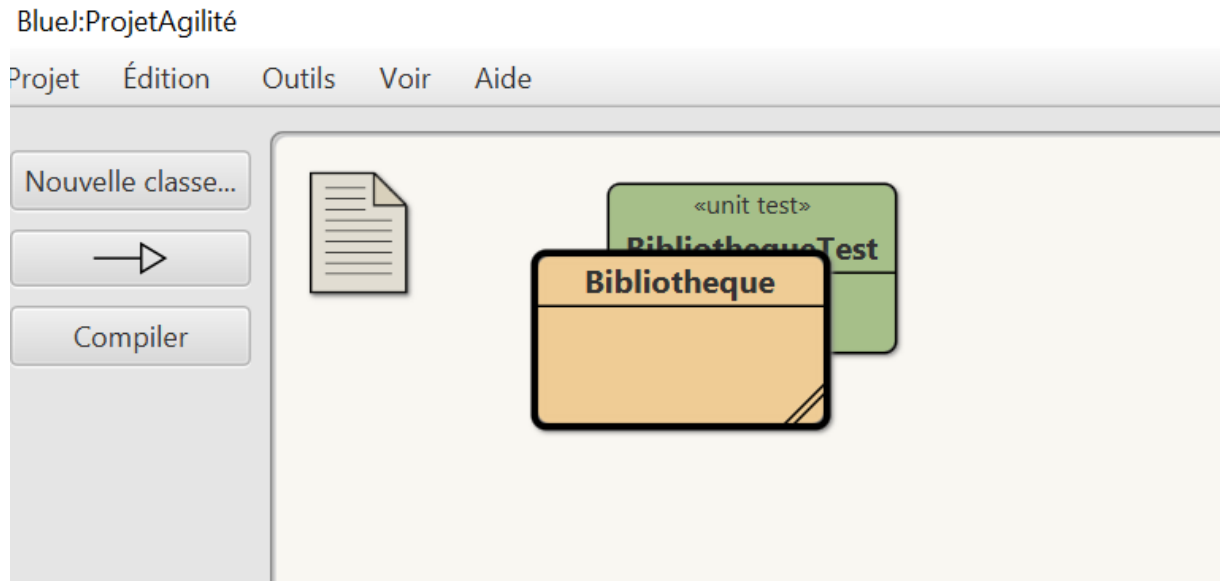
Cette méthode a pour but de convertir l'attribut « location » en lettre majuscule.

- 7) Maintenant vous allez créer une nouvelle instance de notre classe, cela avec un clic droit sur la classe créée et puis vous cliquez sur new Bibliothèque(String nom, String ville)

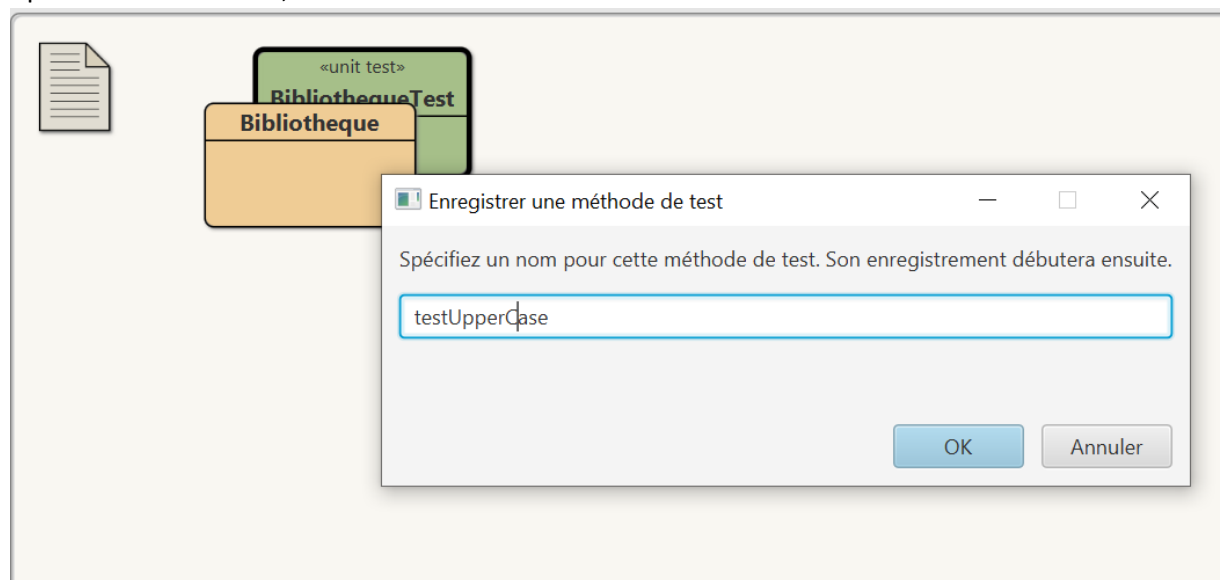
Votre nouvelle instance de « Bibliothèque » comportera deux attributs : nom, représentant le nom de la bibliothèque, et ville, qui indiquera la ville où elle se situe. Ces deux attributs seront de type String (chaîne de caractères)



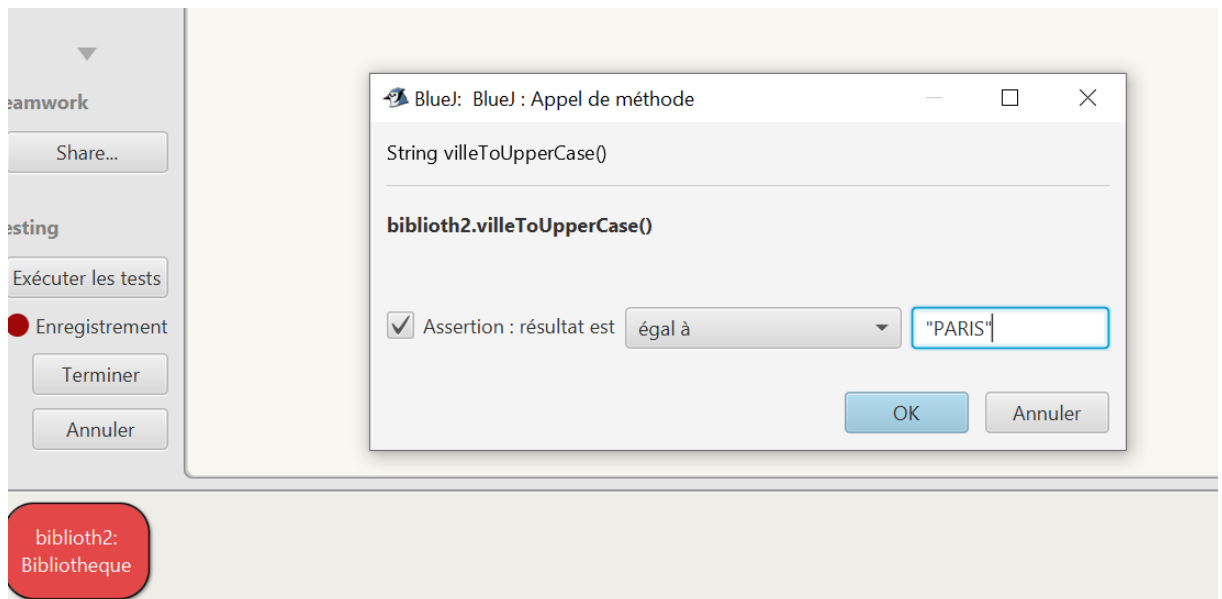
- 8) Maintenant vous allez tester la conformité de vos objets créés, d'abord il faut faire un clic droit sur le rectangle « Bibliothèque » ensuite choisir « Créer classe Test »
Vous aurez un rectangle vert qui porte le nom « BibliothèqueTest » comme ci-dessous



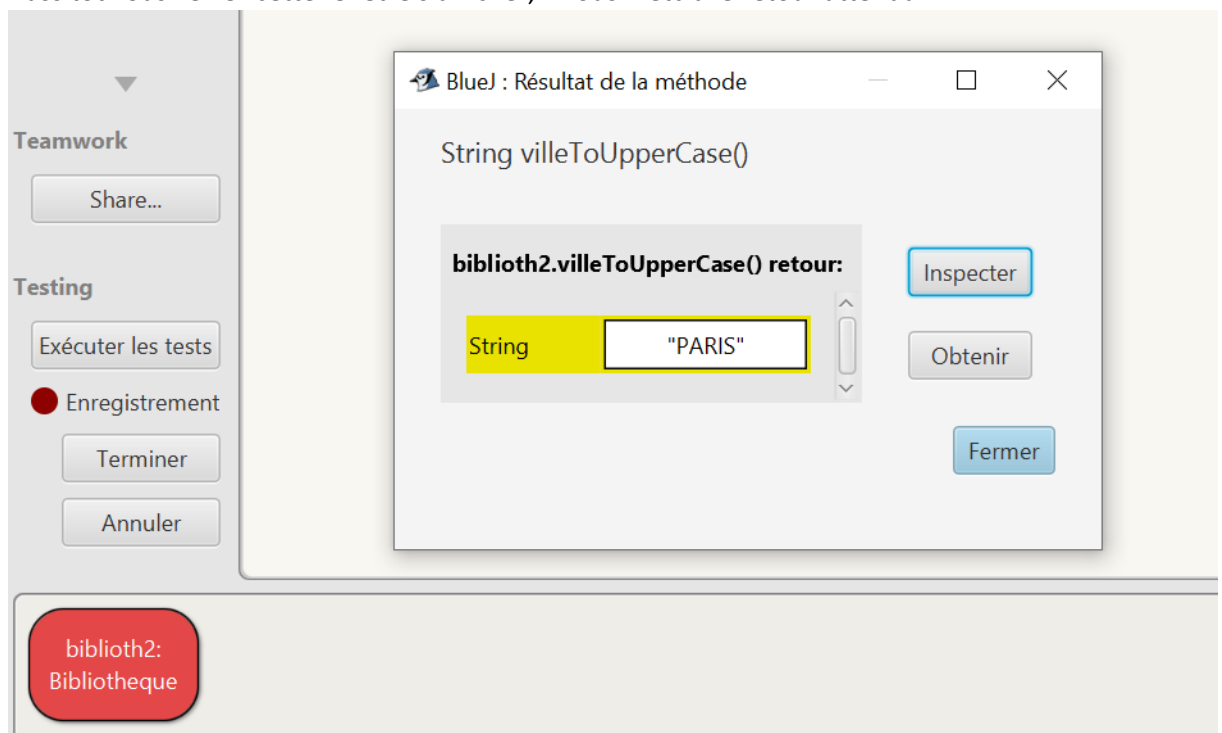
- 9) Ensuite un autre clic droit sur le nouveau rectangle « BibliothèqueTest » et puis sélectionner « Enregistrer une méthode de test »
Après cela vous devrez, nommer votre méthode de test



- 10) Maintenant vous créez une instance à tester, et puis vous faites un clic droit sur cette instance (elle apparaîtra en bas à gauche) et vous sélectionnez la méthode « villeToUpperCase() » et cette fenêtre de dialogue apparaît
Vous entrez la valeur attendue de la méthode « villeToUpperCase() » qui est « PARIS » dans notre cas et puis vous cliquez sur « Ok »



Aussitôt vous verrez cette fenêtre s'afficher, il vous mettra le retour attendu :



- 11) Par la suite vous pourriez lancer le test en appuyant sur « Exécutez les test » et vous obtiendrez le résultat suivant :

BlueJ : Résultats des tests

✓ BibliothequeTest.testUpperCase()

Exécutions : 1

✗ Erreurs : 0

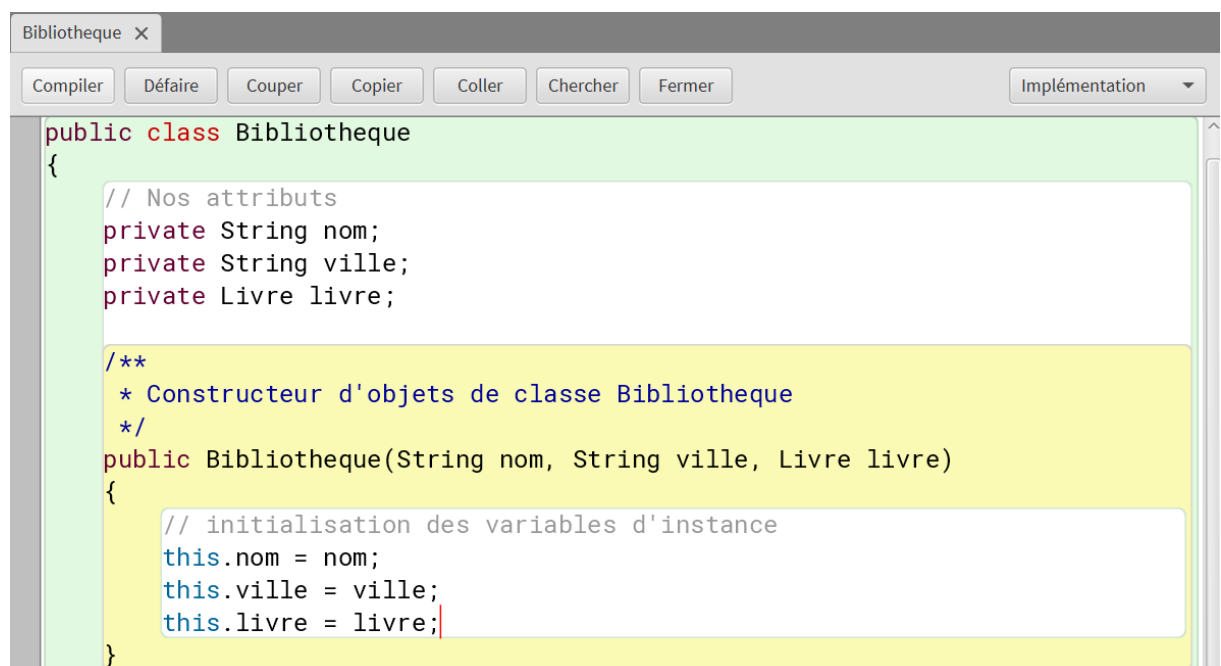
✗ Échecs : 0

Temps total: 16ms

12) Maintenant vous allez créer une nouvelle classe « Livre » qui sera unique pour chaque instance de « Bibliotheque »

Cliquez sur « Nouvelle classe » ensuite mettez « Livre » comme nom

Double cliquez sur le rectangle « Bibliotheque » ensuite rajoutez la ligne « **private Livre livre ;** » et modifiez le constructeur comme ci-dessous

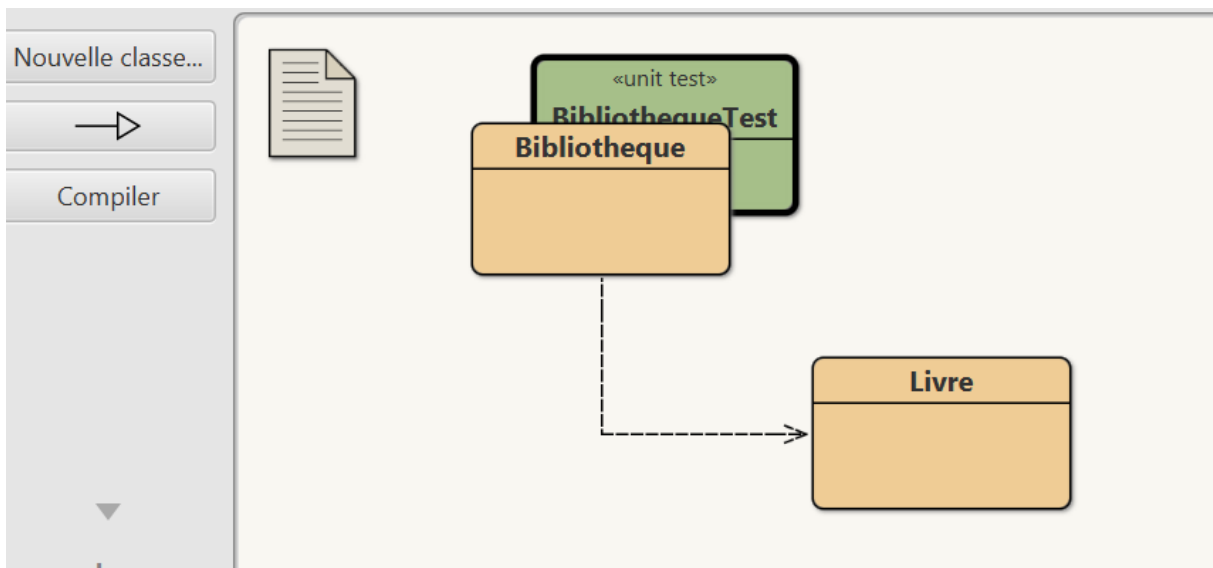


```
public class Bibliotheque
{
    // Nos attributs
    private String nom;
    private String ville;
    private Livre livre;

    /**
     * Constructeur d'objets de classe Bibliotheque
     */
    public Bibliotheque(String nom, String ville, Livre livre)
    {
        // initialisation des variables d'instance
        this.nom = nom;
        this.ville = ville;
        this.livre = livre;
    }
}
```

Maintenant on va créer la relation entre nos 2 classes, faites un clic droit sur le rectangle « Bibliotheque » puis choisissez « Bureau Objets -> Engagements »

Vous devrez ensuite visualiser la flèche suivante qui relie les 2 classes.



- 13) Maintenant vous allez implémenter la classe « Livre » vous pourriez rajouter le corps suivant après avoir double cliquer sur le rectangle « Livre »

```
public class Livre
{
    private String titre;
    private String auteur;
    private String isbn;

    /**
     * Constructeur d'objets de classe Livre
     */
    public Livre(String titre, String auteur, String isbn) {
        this.titre = titre;
        this.auteur = auteur;
        this.isbn = isbn;
    }

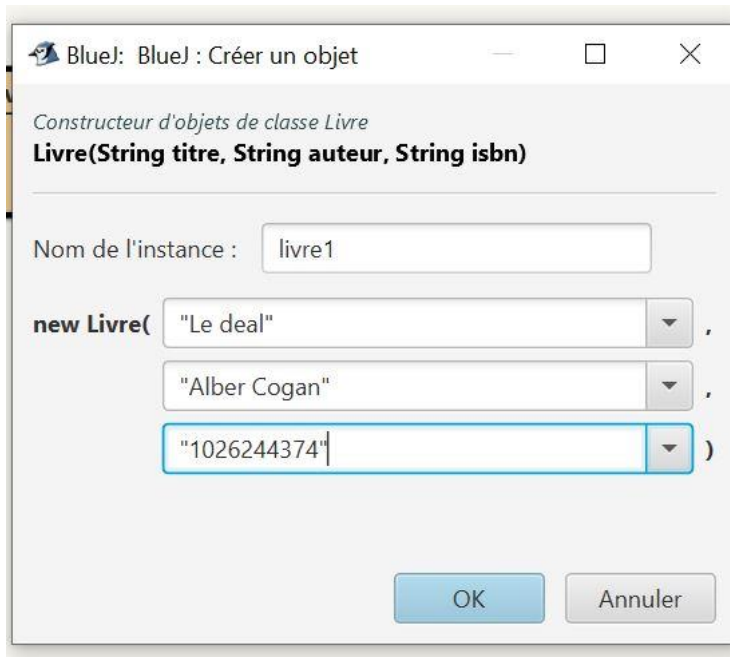
    // Getter pour le titre
    public String getTitre() {
        return titre;
    }

    // Setter pour le titre
    public void setTitre(String titre) {
        this.titre = titre;
    }
}
```

- 14) Maintenant, vous allez ajouter une nouvelle méthode à votre classe « Bibliothèque ». Double-cliquez sur le rectangle la représentant, puis ajoutez les lignes de code suivantes dans la classe « Bibliothèque ».

```
public String toUpperCase() {
    return this.ville.toUpperCase() + " " + this.livre.getTitre().toUpperCase();
}
```

Maintenant nous allons créer une nouvelle instance de livre et bibliotheque, pour cela vous commencez par créer un livre



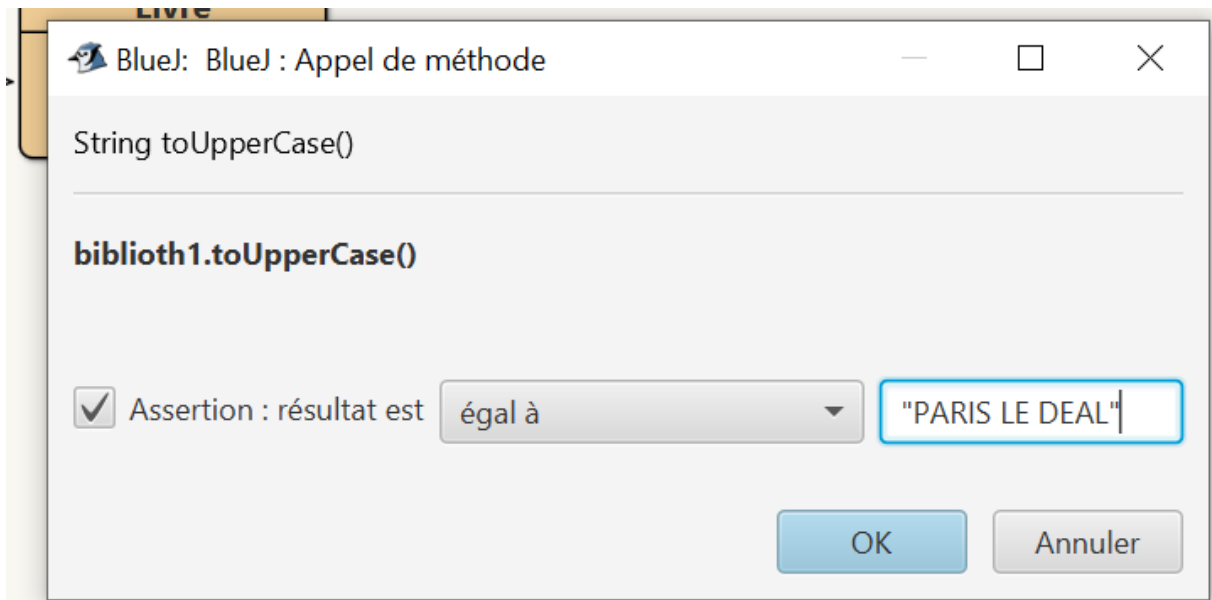
Ensuite vous créer une instance de la classe bibliotheque en utilisant le « livre1 » créé

15) Ici vous allez refaire les mêmes étapes que la question 9

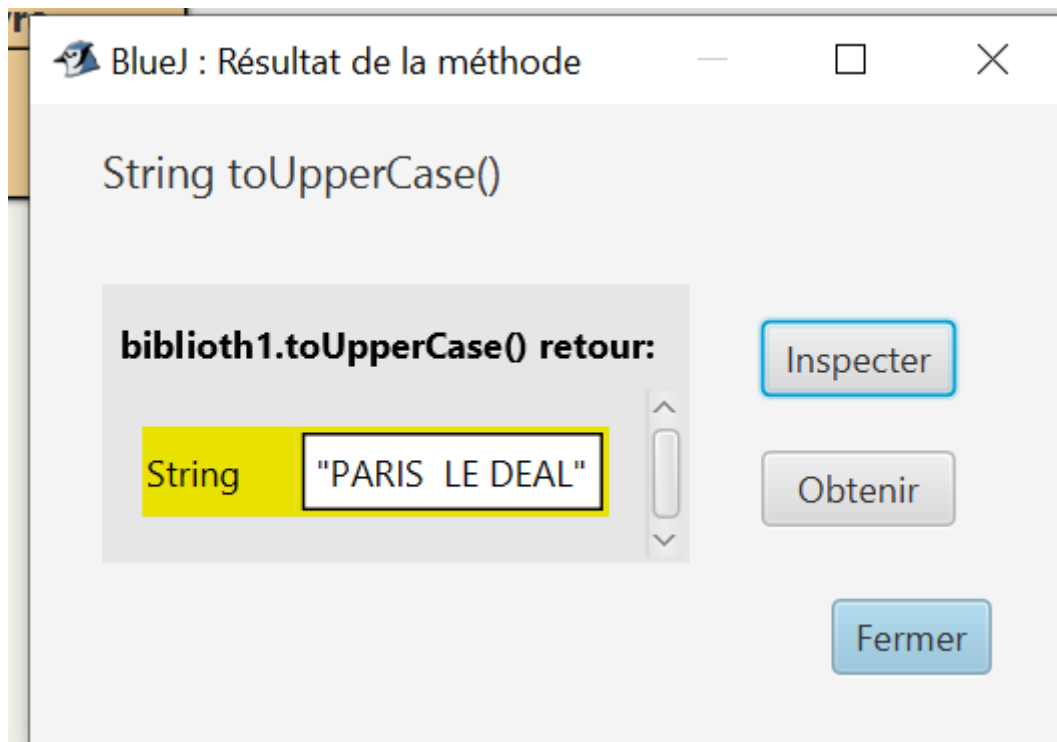
Faites un clic droit sur la classe « Bibliotheque » ensuite choisissez « Enregistrer une méthode de test », et puis choisissez un nom

Ensuite un rectangle rouge devrait apparaître en bas à gauche faites un clic droit dessus ensuite choisissez la méthode que vous allez tester (la nouvelle méthode toUpperCase() créée)

Maintenant remplissez la fenêtre qui apparaît par l'exemple que vous avez choisi :

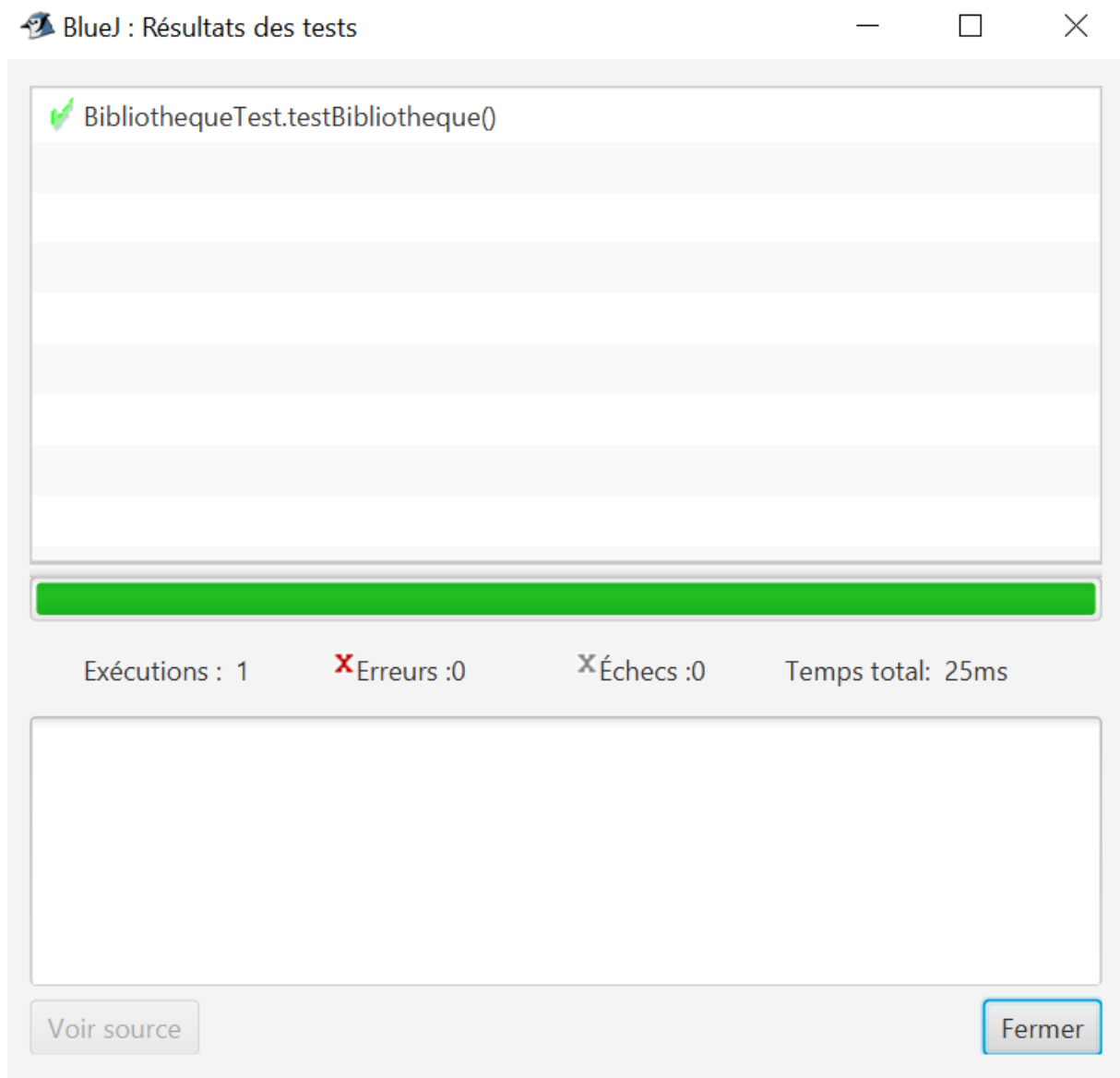


Aussitôt vous verrez cette fenêtre s'afficher, il vous mettra le retour attendu :



Ensuite sélectionnez « Exécuter les test » situé en bas à gauche de votre écran

Et vous devriez avoir la fenêtre montrant le résultat de l'exécution du test.

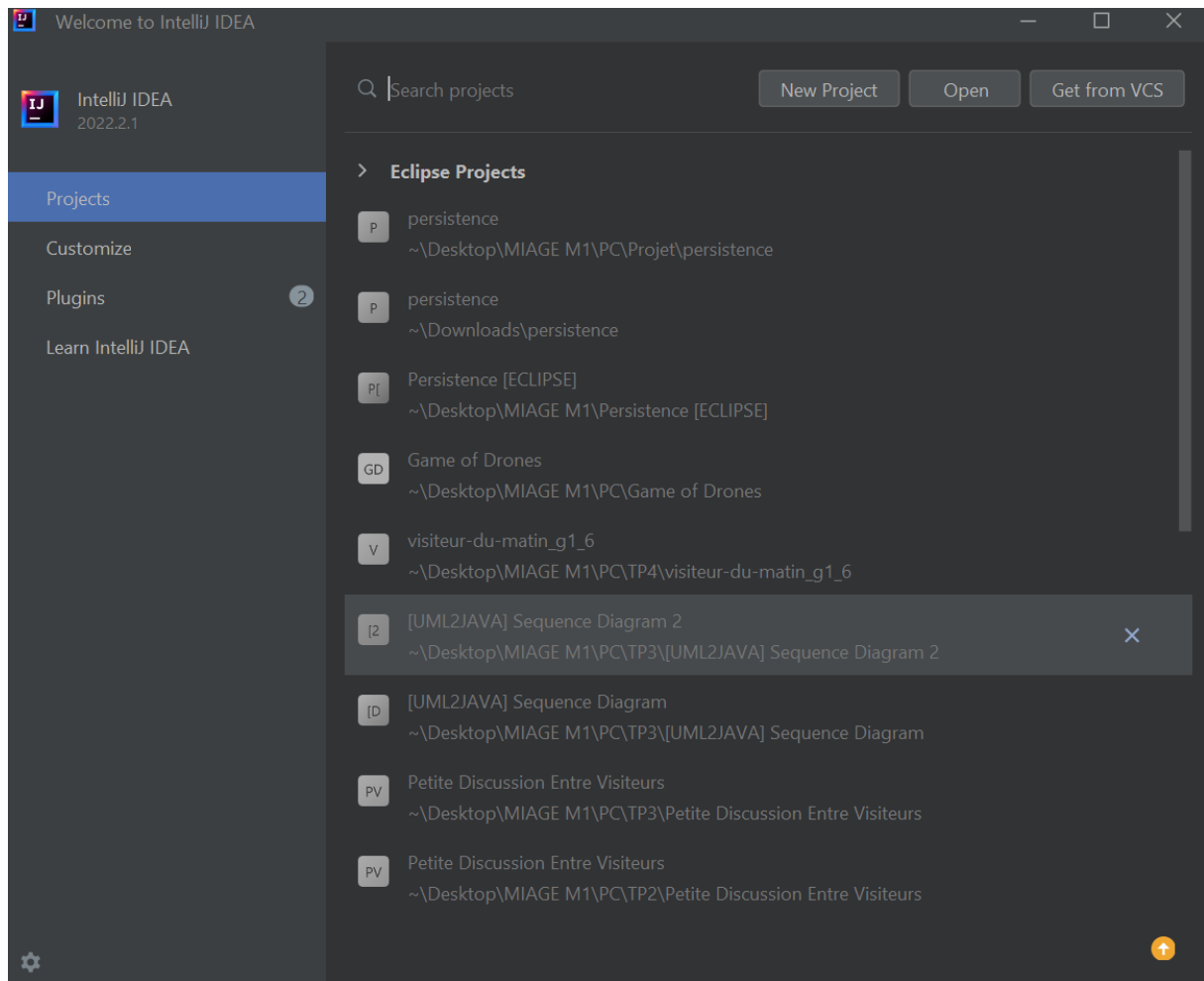


PARTIE 2 : Seconde itération – Eclipse1 et junit

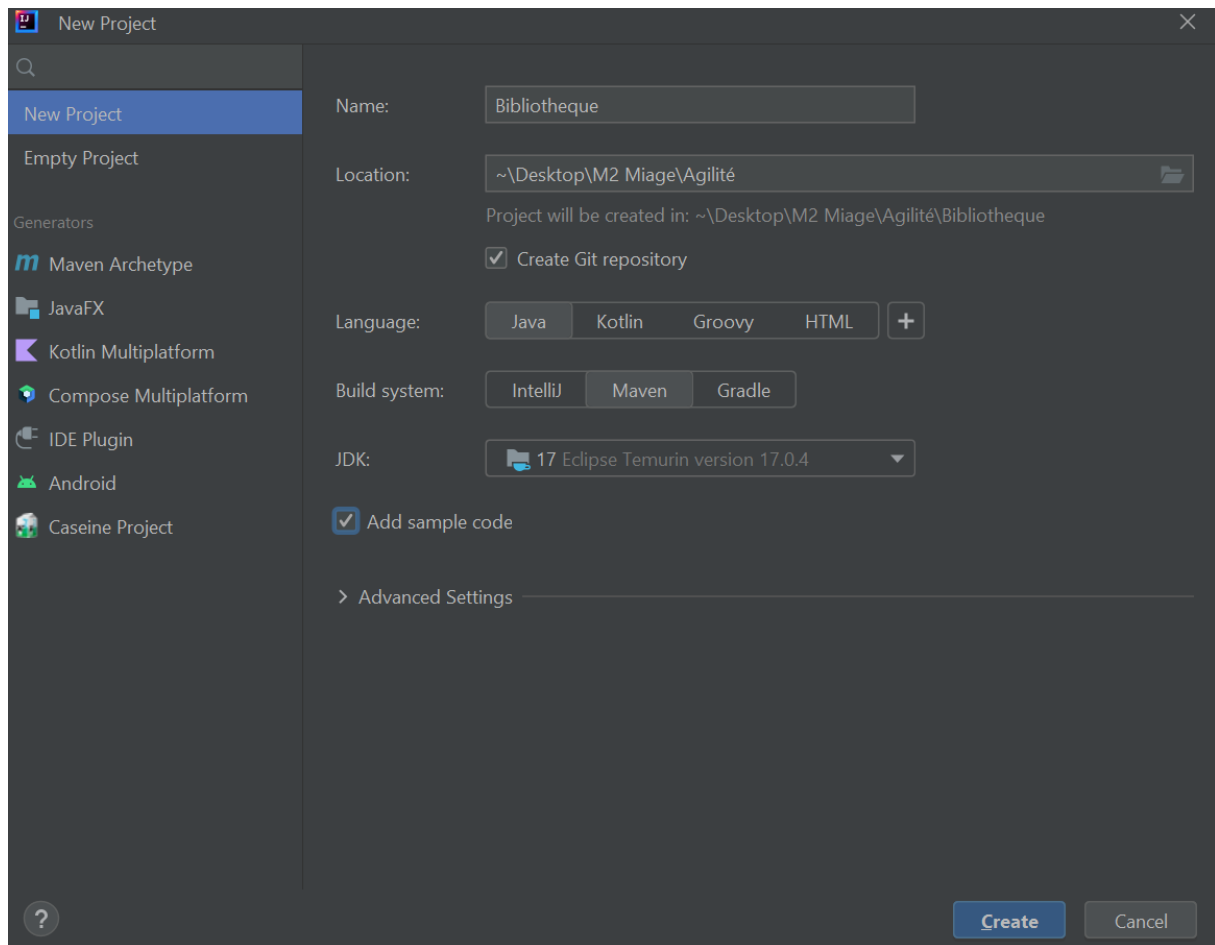
Maintenant vous allez faire votre premier projet sur IntelliJ (<https://www.jetbrains.com/fr-fr/idea/>) IntelliJ est un IDE spécialisé pour écrire du code en Java, le langage de programmation qu'on a utilisé sur BlueJ lors de la première partie du TP.

Après avoir installé le logiciel à partir du lien fourni vous pourriez créer le projet sur lequel vous allez travailler en suivant les étapes suivantes :

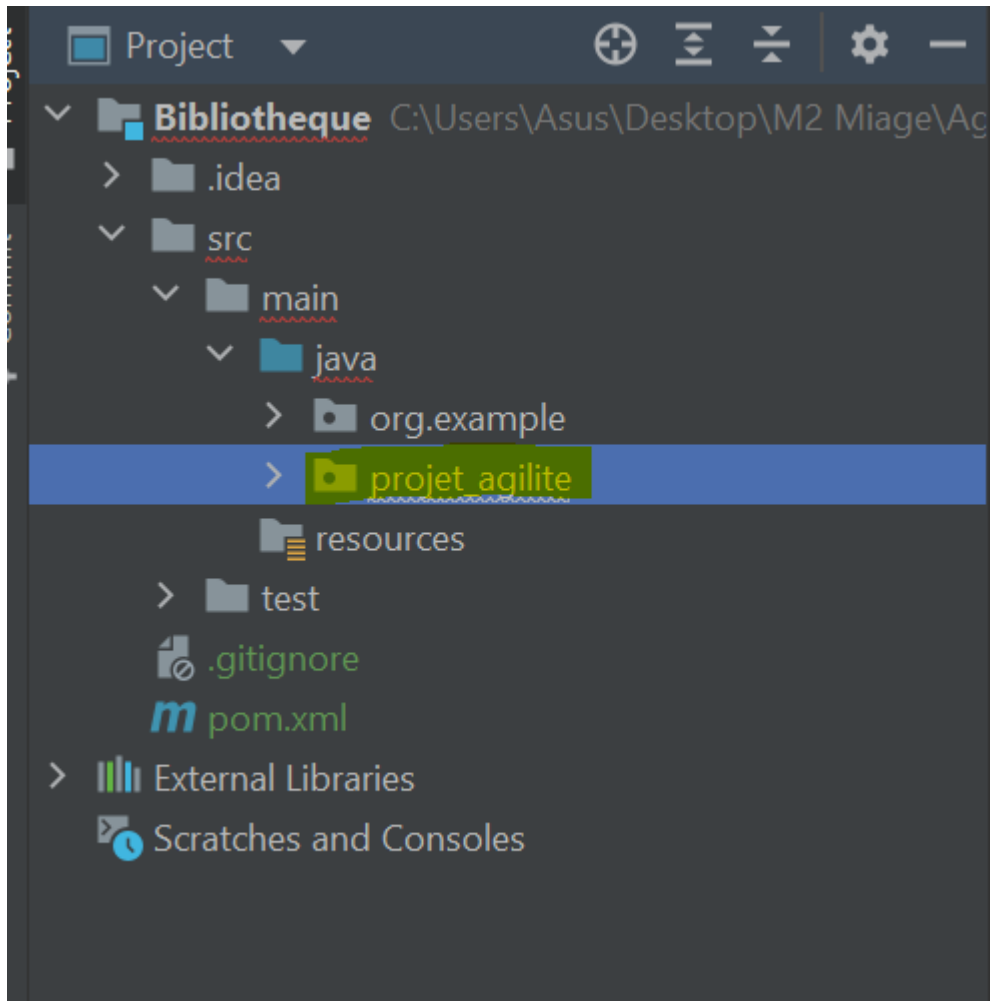
Quand vous ouvrez le logiciel, vous allez trouver l'interface suivante :



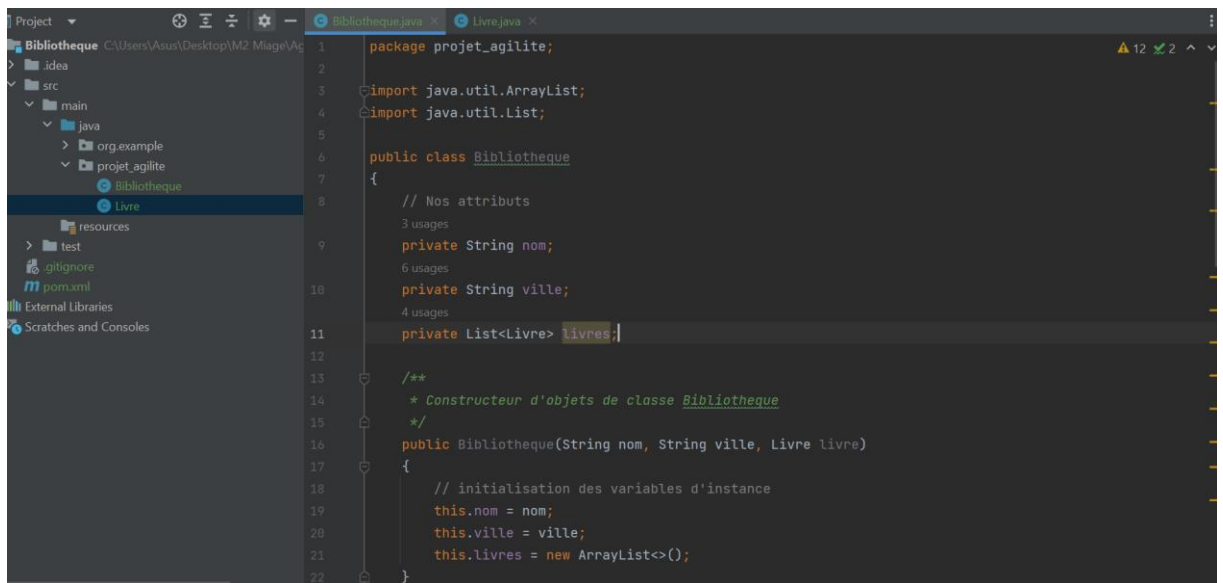
14) Appuyez sur « New Project », et puis nommez votre projet et modifiez si vous voulez l'emplacement du projet



- 16) Maintenant une fois le projet créé vous devez créer ce qu'on appelle « package ». Un package est un genre de conteneur où vous mettrez vos fichiers java ou n'importe quel autre fichier de votre choix. En gros il servira d'espace de stockage des fichiers que vous utiliserez lors du projet
- A gauche de votre écran faites un clic droit ensuite choisissez « New » ensuite « Package » et puis donner un nom à votre package



- 17) Maintenant une fois le package créé, faite clic droit sur le package créé et vous allez créer le fichier java ou vous allez mettre le code qu'on a fait dans la première partie. Choisissez « New » puis « Java Class » ensuite vous allez commencer par créer la première classe qui est « Bibliothèque » dans notre cas
Sur IntelliJ chaque classe va être créée dans un fichier séparé



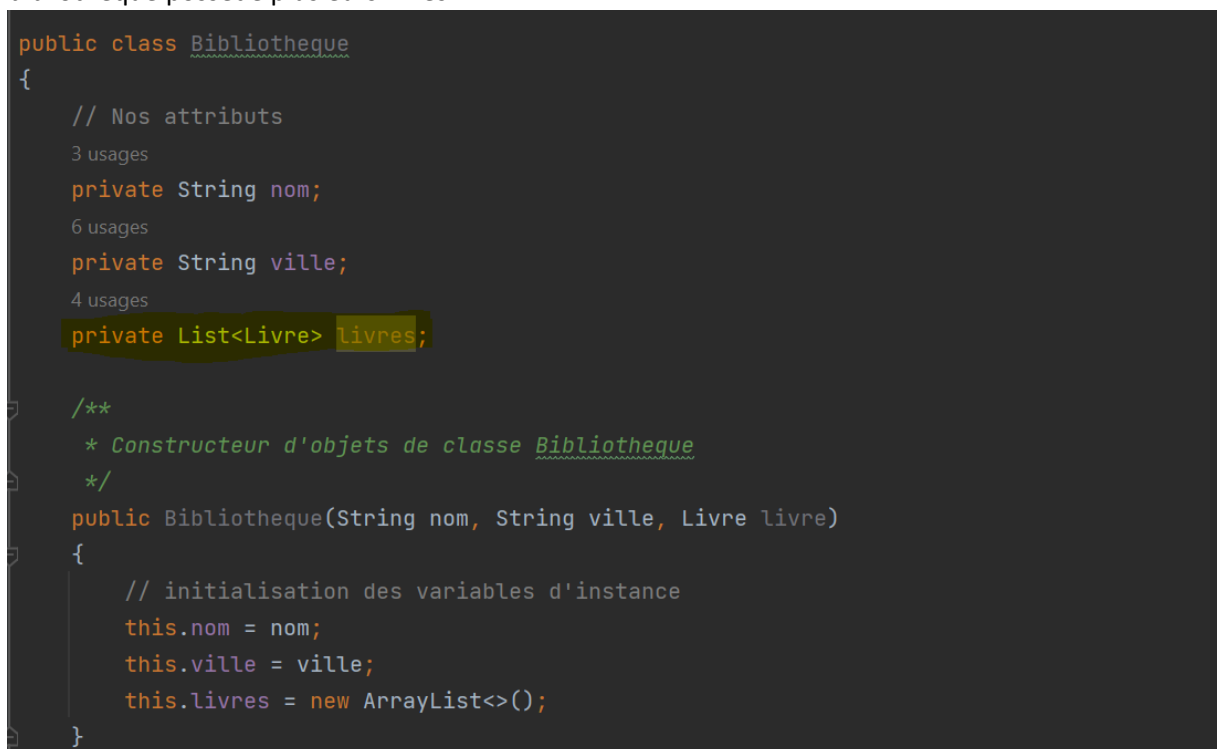
```
24 public String getNom() {
25     return nom;
26 }
27 public void setNom(String nom) {
28     this.nom = nom;
29 }
30 public String getVille() {
31     return ville;
32 }
33 public void setVille(String ville) {
34     this.ville = ville;
35 }
36 public String villeToUpperCase() {
37     return this.ville = this.ville.toUpperCase();
38 }
39 public String toUpperCase() {
40     return this.ville.toUpperCase() + ", " + this.livres.get(0).getTitre().toUpperCase();
41 }
42 public List<Livre> getLivres() {
43     return livres;
44 }
45 public void addLivre(Livre livre) {
46     this.livres.add(livre);
47 }
```

Maintenant vous allez refaire la même manipulation pour la classe « Livre », clic droit sur le package ensuite « New » ensuite « Java class » et nommé la « Livre » avant de rajouter le code suivant

```
1 package projet_agilite;
2
3 public class Livre
4 {
5     3 usages
6     private String titre;
7     3 usages
8     private String auteur;
9     3 usages
10    private String isbn;
11
12    /**
13     * Constructeur d'objets de classe Livre
14     */
15    public Livre(String titre, String auteur, String isbn) {
16        this.titre = titre;
17        this.auteur = auteur;
18        this.isbn = isbn;
19    }
20    // Getter pour le titre
21    1 usage
22    public String getTitre() {
23        return titre;
24    }
25 }
```



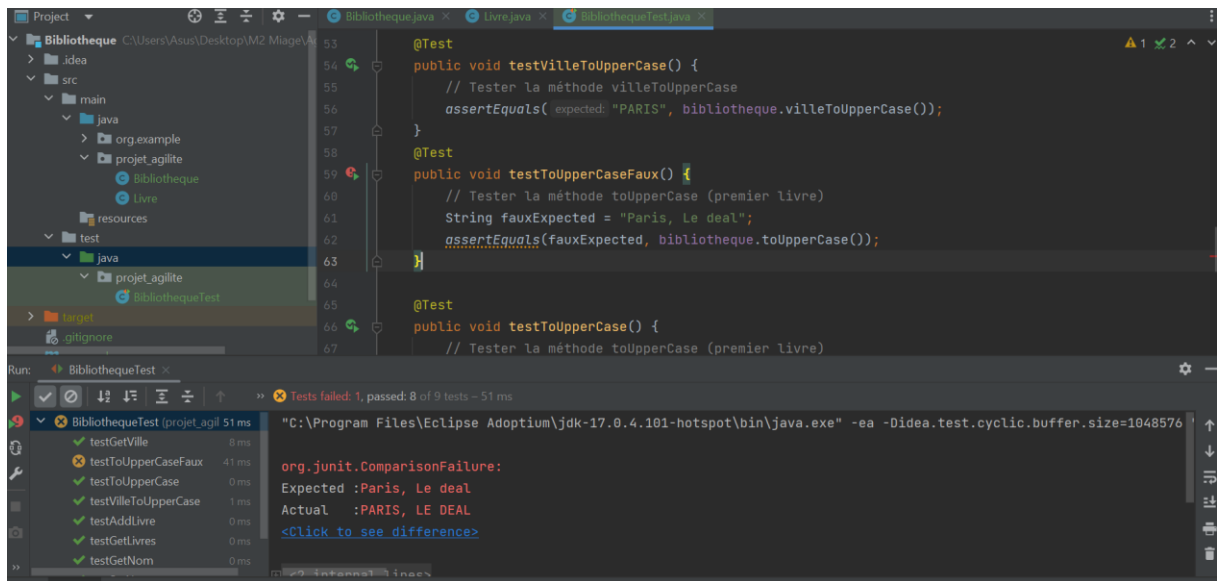
Pour réaliser la liaison entre les 2 classes (l'équivalent de la flèche sur BlueJ) il suffit de rajouter une ligne de code dans la classe « Bibliotheque », cette ligne signifie qu'une bibliothèque possède plusieurs livres



- 18) Maintenant nous allons créer les tests comme ce qu'on a fait hier sur BlueJ. Il s'agit d'un fichier java que vous allez créer comme ceux déjà créés. Aller vers le répertoire « test » -> « java » et puis faites un clic droit sur le package ensuite choisissez « New » puis « Java Class » et nommez la « BibliothequeTest »
Ensuite rajouter le code ci-dessous dans la classe créée

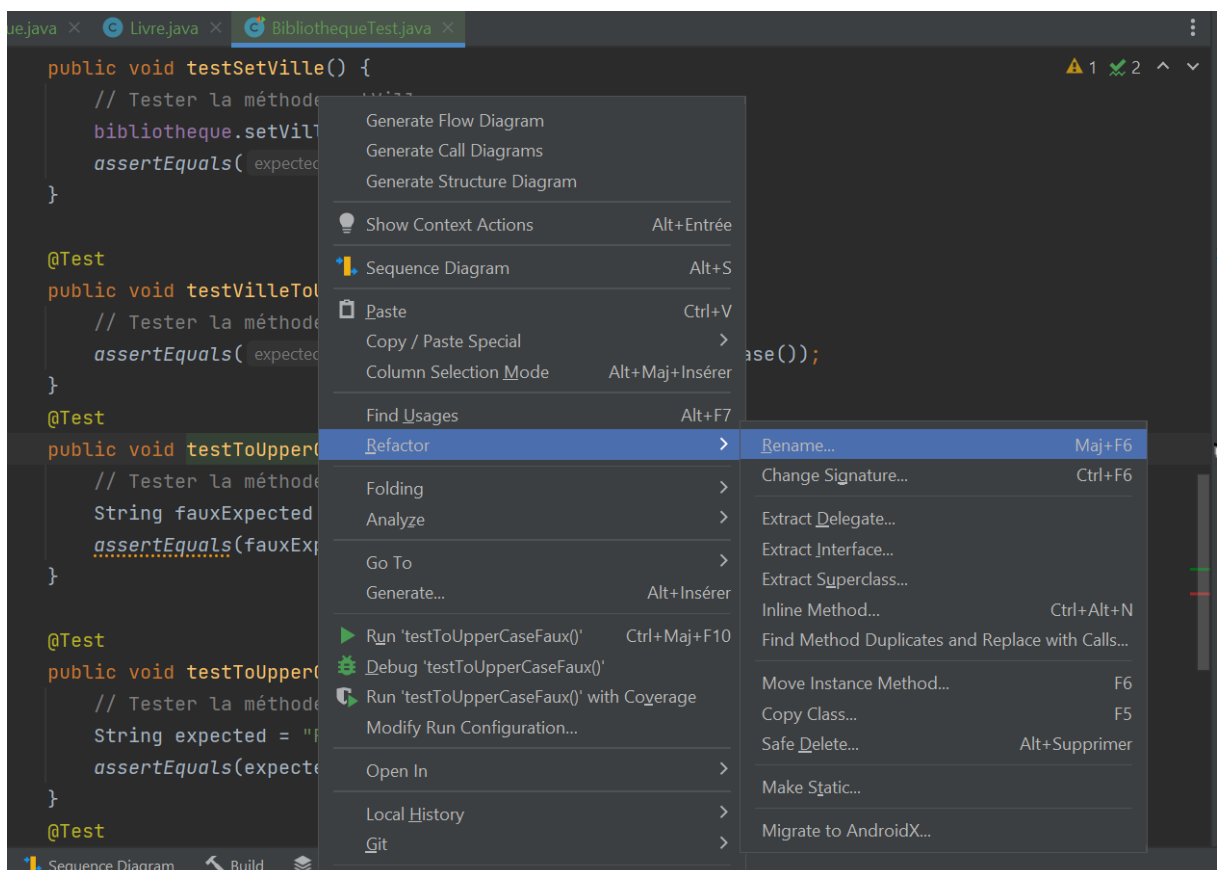

```
Bibliothèque.java x Livre.java x BibliothèqueTest.java x
14
15 @Before
16 public void setUp() {
17     // Initialisation de la bibliothèque et des livres avant chaque test
18     bibliothèque = new Bibliothèque( nom: "François Mitterrand", ville: "Paris", livre: null);
19     livre1 = new Livre( titre: "Le deal", auteur: "Alber Cogan", isbn: "1026244374");
20     livre2 = new Livre( titre: "Les Misérables", auteur: "Victor Hugo", isbn: "978-1234567890");
21
22     // Ajout des livres à la bibliothèque
23     bibliothèque.addLivre(livre1);
24     bibliothèque.addLivre(livre2);
25 }
26
27 @Test
28 public void testGetNom() {
29     // Tester la méthode getNom
30     assertEquals( expected: "François Mitterrand", bibliothèque.getNom());
31 }
32
33 @Test
34 public void testSetNom() {
35     // Tester la méthode setNom
36     bibliothèque.setNom("Nouvelle Bibliothèque");
37     assertEquals( expected: "Nouvelle Bibliothèque", bibliothèque.getNom());
38 }
39
40 @Test
41 public void testVilleToUpperCase() {
42     // Tester la méthode villeToUpperCase
43     assertEquals( expected: "PARIS", bibliothèque.villeToUpperCase());
44 }
45
46 @Test
47 public void testToUpperCaseFaux() {
48     // Tester la méthode toUpperCase (premier livre)
49     String fauxExpected = "Paris, Le deal";
50     assertEquals(fauxExpected, bibliothèque.toUpperCase());
51 }
52
53 @Test
54 public void testToUpperCase() {
55     // Tester la méthode toUpperCase (premier livre)
56     String expected = "PARIS, LE DEAL";
57     assertEquals(expected, bibliothèque.toUpperCase());
58 }
59
60 @Test
61 public void testAddLivre() {
62     // Tester l'ajout d'un livre à la bibliothèque
63     Livre livre3 = new Livre( titre: "Germinal", auteur: "Émile Zola", isbn: "978-2253004222");
64     bibliothèque.addLivre(livre3);
65
66     // Vérifier que le livre a bien été ajouté
67 }
```

19) Nous avons testé toutes les méthodes de la classe Bibliothèque, y compris la vérification de la conversion en majuscules avec la méthode toUpperCase(), qui transforme simplement les chaînes de caractères en lettres majuscules. Comme démontré ci-dessous, le premier test échoue car la chaîne "Paris, Le deal" n'est pas entièrement en majuscules. En revanche, un test correctement formulé avec "PARIS, LE DEAL" passerait sans problème. Vous pouvez vous amuser à modifier les exemples pour observer comment les résultats changent en fonction des différents cas.



20) Maintenant vous allez apprendre à renommer vos différentes classes/fonctions ou même packages.

Il suffit de faire un clic droit sur l'objet que vous voulez changer ensuite appuyez sur «Refactor » puis « Rename » et vous pourriez choisir le nom que vous voudrez



21) Maintenant vous allez découvrir ce que c'est le concept des "test-infected". En fait ils font référence à une approche de développement où les tests sont écrits avant même que le code réel ne soit implémenté. Cette approche met l'accent sur l'écriture des tests pour définir les attentes et les spécifications du code à développer. Les développeurs "test-infected" écrivent des tests avant même de commencer à écrire le code d'implémentation.

Maintenant vous allez en devenir un, vous venez de commencer à faire vos premiers pas dans la programmation et déjà vous progressez et vous apprenez de la meilleure des manières.

Dans les tests que vous avez fait sur BlueJ ou tout à l'heure sur IntelliJ, les tests sont écrits après l'implémentation du code. Ils sont utilisés pour vérifier le comportement des méthodes de la classe « Bibliothèque ». Ces tests sont écrits pour valider le fonctionnement des méthodes après leur implémentation. Il s'agit d'une approche plus traditionnelle où les tests sont utilisés pour valider le code déjà écrit.

Par exemple ici dans notre exemple on a utilisé une méthode getTotalLivres() qu'on n'a pas encore créée mais qu'on sait qu'on pourrait utiliser dans le futur du coup on a créé un nouveau code de test qui utilise cette méthode qui n'est pas encore existante

```
@Test
public void testGetTotalLivres() {
    // Étape 1 : Écrire un test qui échoue
    Bibliotheque bibliotheque = new Bibliotheque( nom: "François Mitterrand", ville: "Paris", livre: null);
    assertEquals(0, bibliotheque.getTotalLivres());
    // La méthode getTotalLivres() n'existe pas encore
}

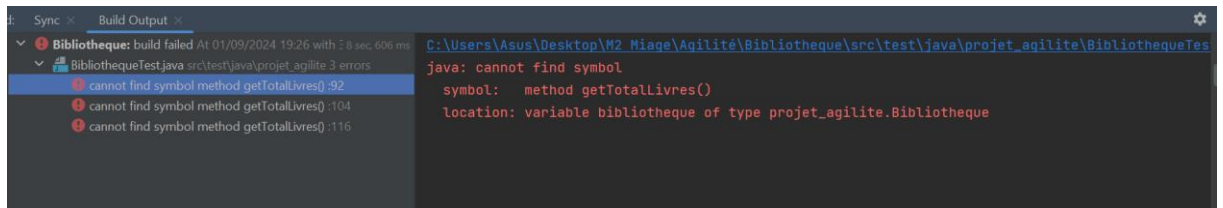
@Test
public void testAddLivresAndGetTotalLivres() {
    // Étape 2 : Implémenter le code pour faire passer le premier test
    Bibliotheque bibliotheque = new Bibliotheque( nom: "François Mitterrand", ville: "Paris", livre: null);
    Livre livre1 = new Livre( titre: "Le deal", auteur: "Alber Cogan", isbn: "1026244374");
    bibliotheque.addLivres(livre1);

    // Étape 3 : Écrire un nouveau test qui échoue
    assertEquals(1, bibliotheque.getTotalLivres());
}

@Test
public void testMultipleLivresAndGetTotalLivres() {
    // Étape 2 : Implémenter le code pour faire passer le deuxième test
    Bibliotheque bibliotheque = new Bibliotheque( nom: "François Mitterrand", ville: "Paris", livre: null);
    Livre livre1 = new Livre( titre: "Le deal", auteur: "Alber Cogan", isbn: "1026244374");
    Livre livre2 = new Livre( titre: "Le second livre", auteur: "John Doe", isbn: "1234567890");
    bibliotheque.addLivres(livre1);
    bibliotheque.addLivres(livre2);

    // Étape 3 : Écrire un nouveau test qui échoue
    assertEquals(2, bibliotheque.getTotalLivres());
}
```

Comme vous pouvez le voir ci-dessous le test retourne une erreur comme prévu car la fonction getTotalLivres() n'a pas été implémentée.



22) Maintenant nous allons implémenter cette nouvelle méthode et vous pourriez rajouter le code suivant :

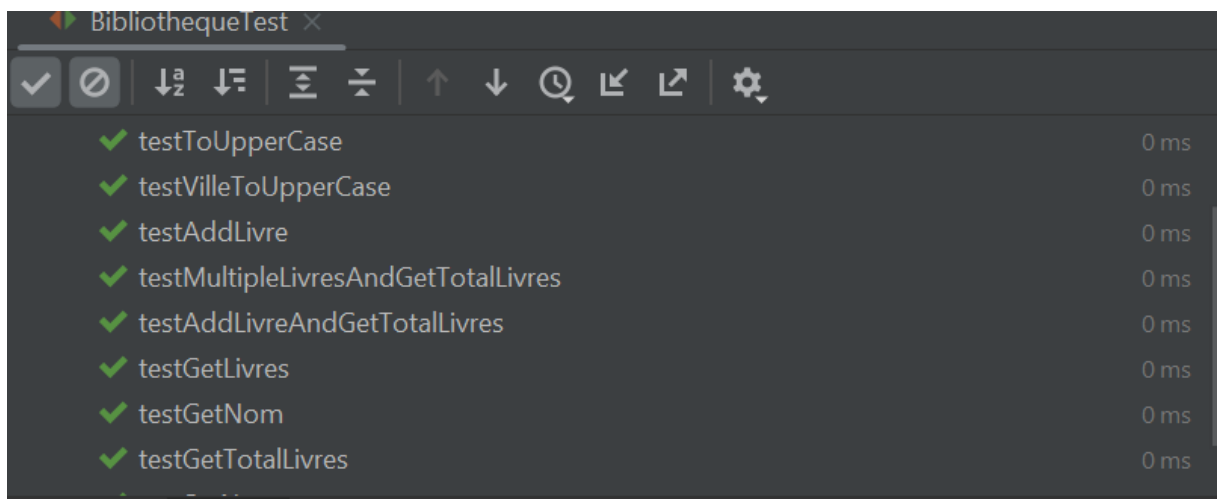
```
private int totalLivres;

3 usages
public int getTotalLivres() {
    return totalLivres;
}

public void setTotalLivres(int totalLivres) {
    this.totalLivres = totalLivres;
}

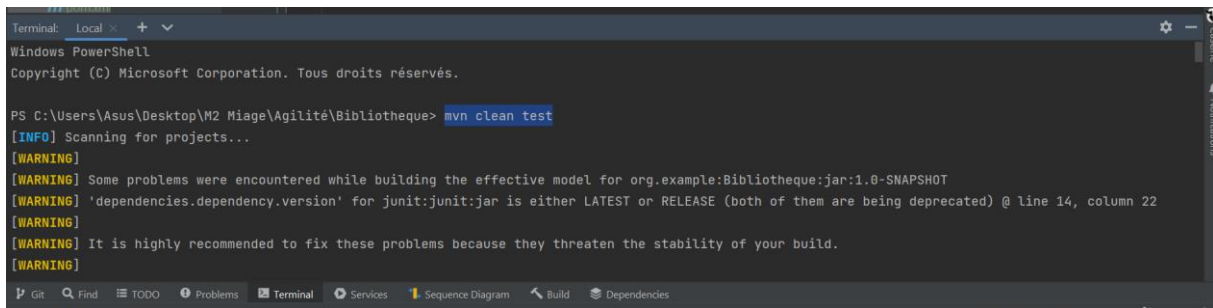
6 usages
public void addLivres(Livre livre) {
    this.livres.add(livre);
    this.totalLivres++;
}
```

Ensuite vous pourriez relancer le test de getTotalLivres() et vous verriez que l'erreur n'apparaît plus :



23) Vous allez maintenant découvrir une nouvelle méthode pour exécuter les tests. Tout bon informaticien (et je pense que vous êtes en bonne voie pour en devenir un) doit être à l'aise avec l'utilisation de ce qu'on appelle le « terminal ». En réalité, de nombreuses actions que vous réalisez en cliquant dans Windows peuvent être effectuées via ce qu'on appelle la ligne de commande. En résumé, les tests que vous avez réalisés seront exécutés à l'aide de la ligne

de commande suivante :



```
Terminal: Local x + v
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

PS C:\Users\Asus\Desktop\M2 Miage\Agilité\Bibliothèque> mvn clean test
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for org.example:Bibliothèque:jar:1.0-SNAPSHOT
[WARNING] 'dependencies.dependency.version' for junit:junit:jar is either LATEST or RELEASE (both of them are being deprecated) @ line 14, column 22
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
```

24)) Dans le domaine du développement logiciel, il arrive que des erreurs inattendues surgissent malgré toutes les précautions prises. Par exemple, vous avez peut-être élaboré des tests complets pour votre classe Bibliothèque, mais lors de l'exécution, une erreur apparaît dans une section du code qui semblait robuste. Cette situation peut être causée par des facteurs externes tels que des bibliothèques tierces, des configurations système ou d'autres éléments hors de votre contrôle direct.

La loi de Murphy rappelle aux développeurs que, malgré nos efforts pour anticiper et résoudre les problèmes, des imprévus surviennent inévitablement. Elle souligne l'importance de la résilience, de la gestion efficace des erreurs et de la préparation face à l'incertitude tout au long du processus de développement logiciel.

PARTIE 3 : CUCUMBER & GIT

Vous allez maintenant découvrir un nouvel outil appelé « Cucumber ». Cucumber est un logiciel qui facilite le développement axé sur le comportement. Son approche, la BDD (Behavior-Driven Development), repose sur un analyseur de langage naturel nommé Gherkin. Gherkin permet de définir les comportements attendus du logiciel dans un langage simple et compréhensible, en français ou en anglais.

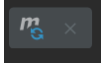
Pour commencer à utiliser cet outil, vous devrez d'abord ouvrir le fichier « pom.xml » et y ajouter le code XML suivant :



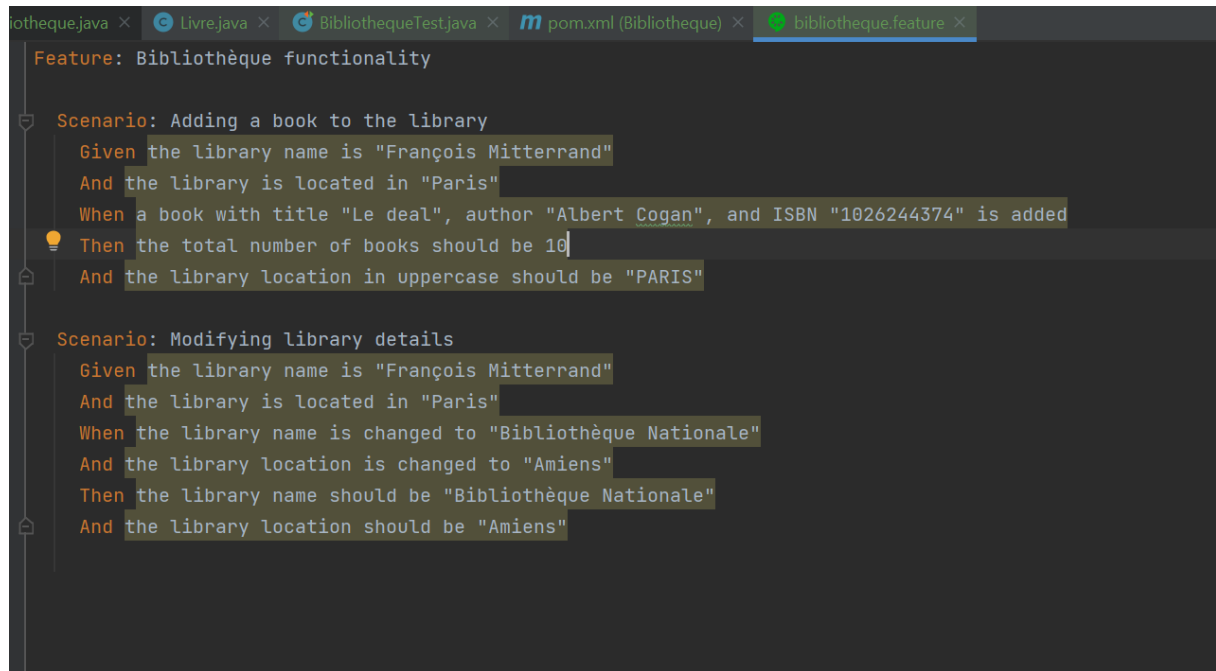
```
</dependency>

<!-- Cucumber Core -->
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-core</artifactId>
  <version>7.0.0</version>
  <scope>test</scope>
</dependency>

<!-- Cucumber JUnit -->
<dependency>
  <groupId>io.cucumber</groupId>
  <artifactId>cucumber-junit</artifactId>
  <version>7.0.0</version>
  <scope>test</scope>
</dependency>
```

Et pour télécharger les nouvelles dépendances vous allez cliquer sur le bouton  apparent dans la capture précédente pour recharger le maven

Une fois les dépendances installées, créer un autre fichier et nommez le « bibliotheque.feature » dans test->ressources->votre package ici vous allez rajouter les scénarios que vous voudrez pour commencer vous pourriez rajouter notre exemple :



The screenshot shows an IDE with several tabs open: Bibliotheque.java, Livre.java, BibliothequeTest.java, pom.xml (Bibliotheque), and bibliotheque.feature. The 'bibliotheque.feature' tab is active, displaying Gherkin syntax for a feature named 'Bibliothèque functionality'. It contains two scenarios: 'Adding a book to the library' and 'Modifying library details'. The first scenario includes steps for setting library name and location, adding a book, and verifying the total number of books and location. The second scenario includes steps for changing the library name and location and verifying the changes.

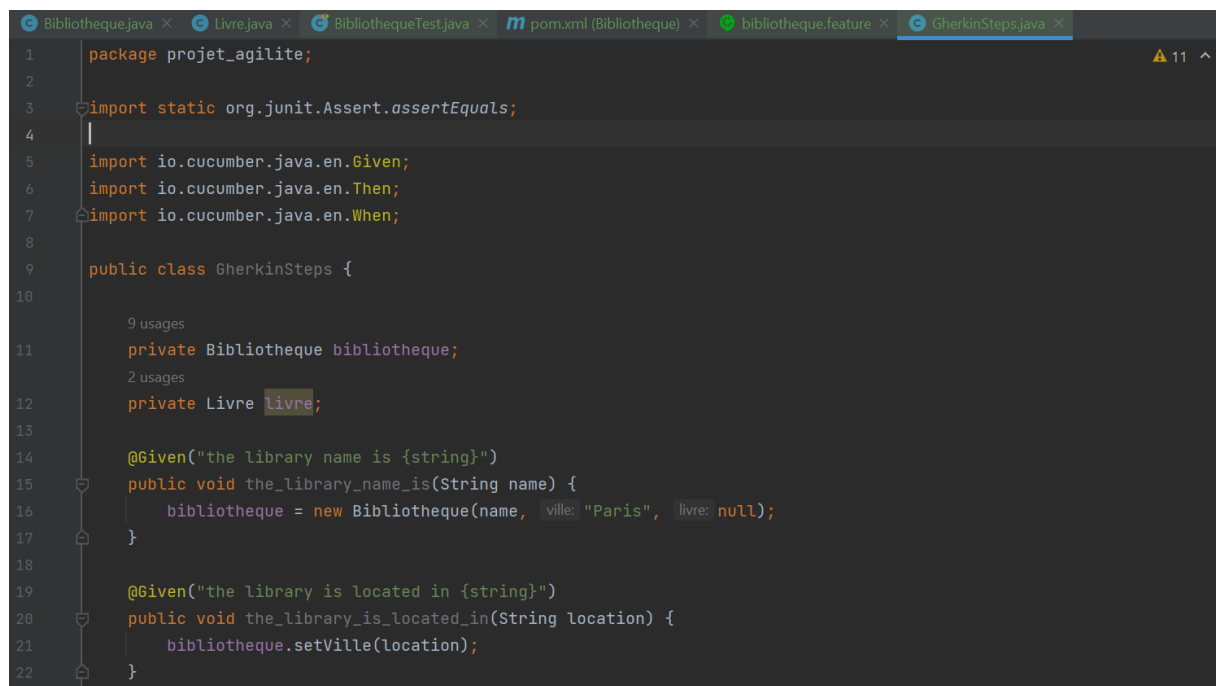
```
Feature: Bibliothèque functionality

Scenario: Adding a book to the library
  Given the library name is "François Mitterrand"
  And the library is located in "Paris"
  When a book with title "Le deal", author "Albert Cogan", and ISBN "1026244374" is added
  Then the total number of books should be 10
  And the library location in uppercase should be "PARIS"

Scenario: Modifying library details
  Given the library name is "François Mitterrand"
  And the library is located in "Paris"
  When the library name is changed to "Bibliothèque Nationale"
  And the library location is changed to "Amiens"
  Then the library name should be "Bibliothèque Nationale"
  And the library location should be "Amiens"
```

Une fois les scénarios créés, créez un nouveau fichier et nommez-le « GherkinsStep.java » dans test->java->votre package

Ensuite dans ce fichier rajouter le code suivant :



The screenshot shows an IDE with several tabs open: Bibliotheque.java, Livre.java, BibliothequeTest.java, pom.xml (Bibliotheque), bibliotheque.feature, and GherkinSteps.java. The 'GherkinSteps.java' tab is active, displaying Java code for the GherkinSteps class. The code includes package declarations, imports for JUnit and Cucumber, and two @Given methods for setting up the library name and location.

```
package projet_agilite;

import static org.junit.Assert.assertEquals;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class GherkinSteps {

    9 usages
    private Bibliotheque bibliotheque;
    2 usages
    private Livre livre;

    @Given("the library name is {string}")
    public void the_library_name_is(String name) {
        bibliotheque = new Bibliotheque(name, ville: "Paris", livre: null);
    }

    @Given("the library is located in {string}")
    public void the_library_is_located_in(String location) {
        bibliotheque.setVille(location);
    }
}
```

```

24     @When("a book with title {string}, author {string}, and ISBN {string} is added")
25     public void a_book_is_added(String title, String author, String isbn) {
26         livre = new Livre(title, author, isbn);
27         bibliotheque.addLivre(livre);
28     }
29
30     @Then("the total number of books should be {int}")
31     public void the_total_number_of_books_should_be(int totalBooks) {
32         assertEquals(totalBooks, bibliotheque.getLivres().size());
33     }
34
35     @Then("the library location in uppercase should be {string}")
36     public void the_library_location_in_uppercase_should_be(String locationUpper) {
37         assertEquals(locationUpper, bibliotheque.villeToUpperCase());
38     }
39
40     @When("the library name is changed to {string}")
41     public void the_library_name_is_changed_to(String newName) {
42         bibliotheque.setNom(newName);
43     }
44
45     @When("the library location is changed to {string}")
46     public void the_library_location_is_changed_to(String newLocation) {
47         bibliotheque.setVille(newLocation);
48     }

```

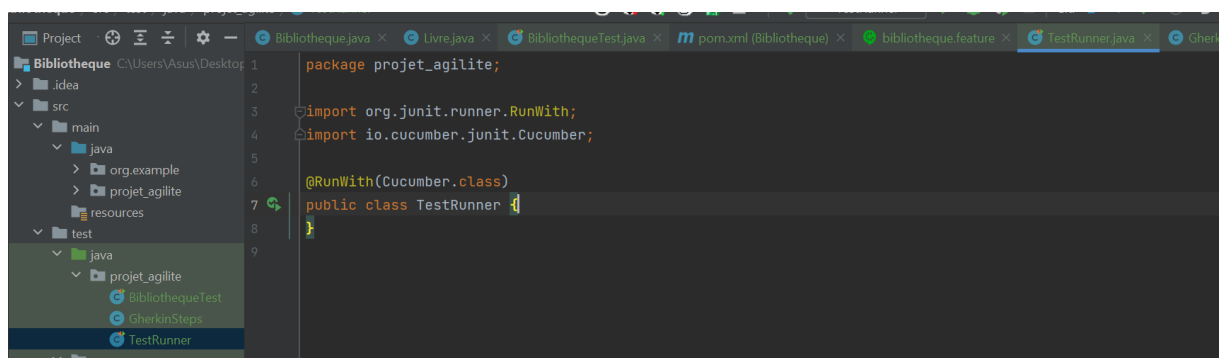
```

49
50     @Then("the library name should be {string}")
51     public void the_library_name_should_be(String expectedName) {
52         assertEquals(expectedName, bibliotheque.getNom());
53     }
54
55     @Then("the library location should be {string}")
56     public void the_library_location_should_be(String expectedLocation) {
57         assertEquals(expectedLocation, bibliotheque.getVille());
58     }
59 }
60

```

Ensuite créez un nouveau fichier et nommez-le « TestRunner.java » ou autre dans test->java->votre package

Et dans ce fichier rajouter le code suivant :



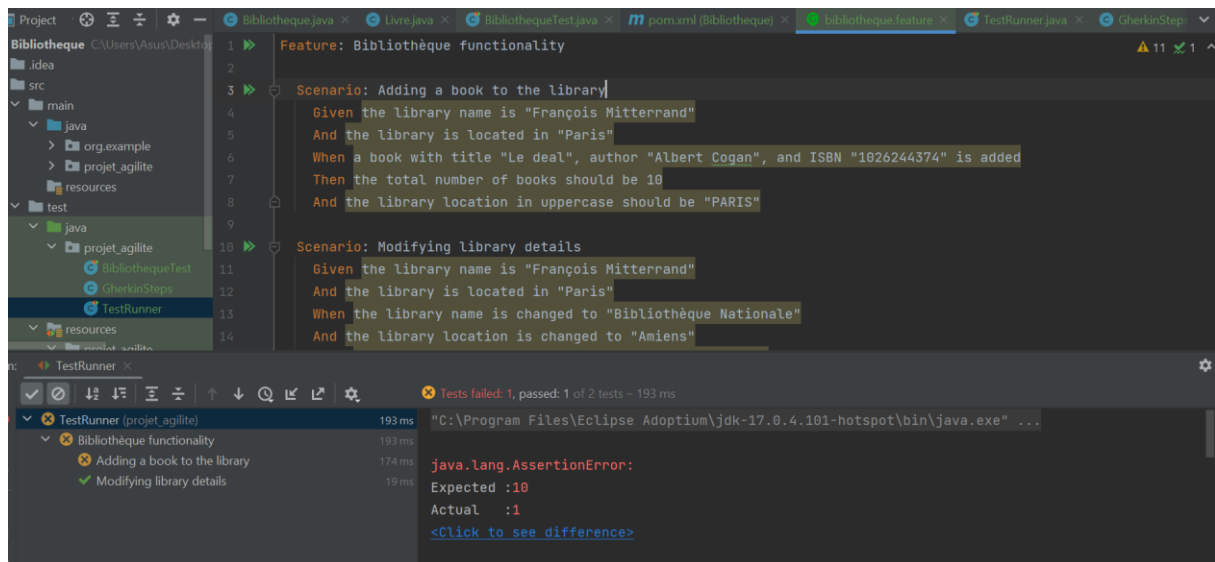
```

1 package projet_agillite;
2
3 import org.junit.runner.RunWith;
4 import io.cucumber.junit.Cucumber;
5
6 @RunWith(Cucumber.class)
7 public class TestRunner {
8
9

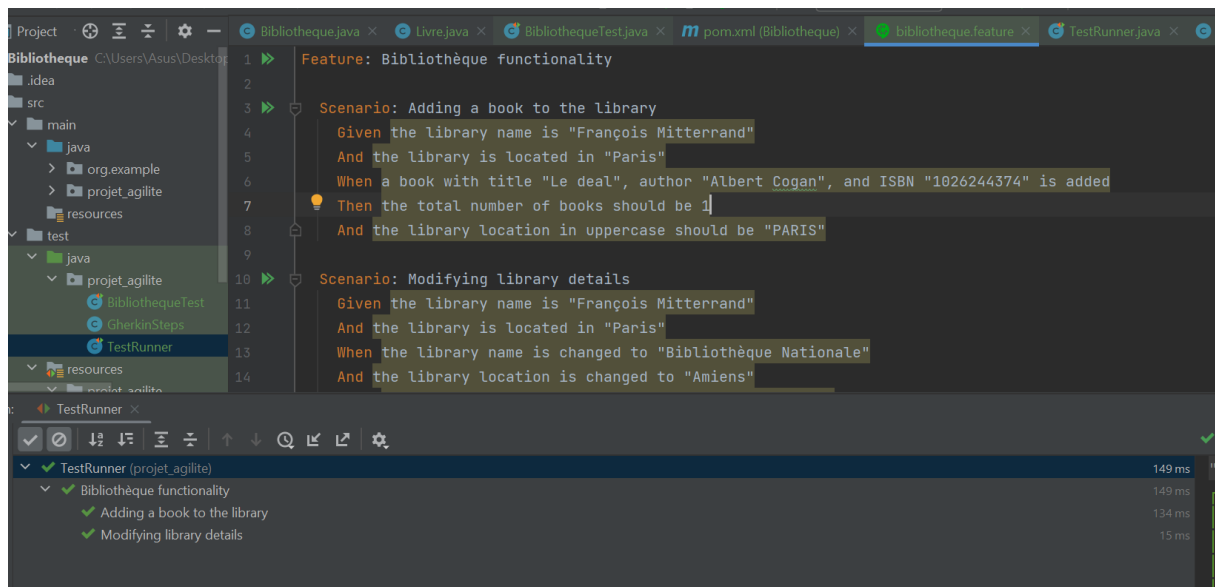
```

Une fois terminé, ici vous avez implémentez le code le de vos scénarios créés tout à l'heure.

Pour tester on vous a réalisé 2 scénarios le premier où on a créé une bibliothèque avec un livre ensuite on lui dit qu'il devrait en trouver 10 ça renvoie dans ce cas une erreur



Ensuite dans le fichier « `bibliothèque.feature` » changez le 10 par un 1 et vous verrez qu’il ne renvoie plus d’erreur



Dans le domaine de l'informatique, la collaboration en équipe est essentielle. Les développeurs utilisent des outils de gestion de versions pour faciliter cette collaboration et gérer les différentes versions de leur code. L'un des outils les plus connus pour cela est « [GitHub](#) ».

Pour connecter votre projet à un dépôt Git, commencez par créer un compte GitHub en suivant le lien ci-dessous suivant [github](#)

Ensuite créez ce qu’on appelle un dépôt en cliquant sur « [Create a new repository](#) » puis rajoutez un nom à votre dépôt

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * RedaZenagui / Repository name * Bibliothèque
 ✔ Bibliothèque is available.

Great repository names are short and memorable. Need inspiration? How about [scaling-telegram](#) ?

Description (optional)
 Ce dépôt contient un projet Java conçu pour simuler un système simple de gestion de bibliothèque. L'objectif

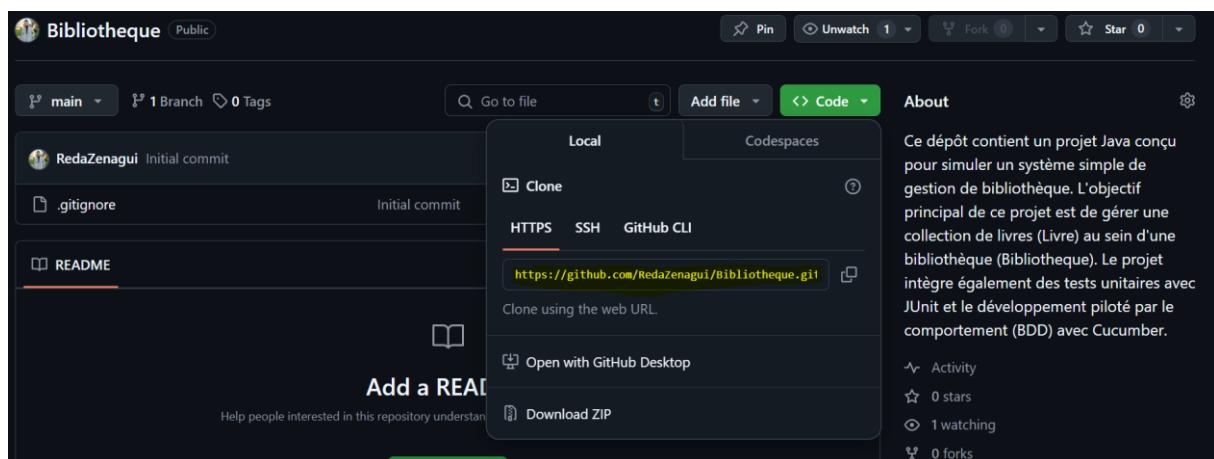
☒ **Public**
 Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
 You choose who can see and commit to this repository.

Initialize this repository with:
☐ **Add a README file**
 This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
 .gitignore template: **Maven**

Copiez le lien de votre dépôt github comme ci-dessous



Et puis vous exécutez les commandes suivantes :

- Git remote add origin « le lien copié précédemment »
- Git status
- Git add .
- Git commit -m 'votre message'
- Git push origin master

Vous avez maintenant associé votre projet local à un dépôt Git. À partir de maintenant, après chaque modification, vous pourrez utiliser git commit suivi de git push pour mettre à jour la version en ligne de votre projet.

Ce tutoriel touche à sa fin. Nous espérons qu'il vous a été utile et que vous avez acquis de nouvelles compétences. Vous avez fait vos premiers pas avec le logiciel BlueJ et écrit vos premières lignes de code en Java. À votre âge, c'est une réalisation remarquable et vous vous distinguez déjà de vos camarades. De plus, vous avez découvert Cucumber et GitHub, deux outils précieux pour les tests et la collaboration en équipe. En attendant le prochain cours, toute l'équipe vous souhaite une excellente continuation dans votre apprentissage !