Tester c’est douter ?

Travaux pratiques

Tests unitaires

Les fichiers dont vous aurez besoin pour les exercices sont dans le dossier `exercices-java` du dépôt du cours.

---

https://gitlab.com/cours-iut/2026/-/tree/main

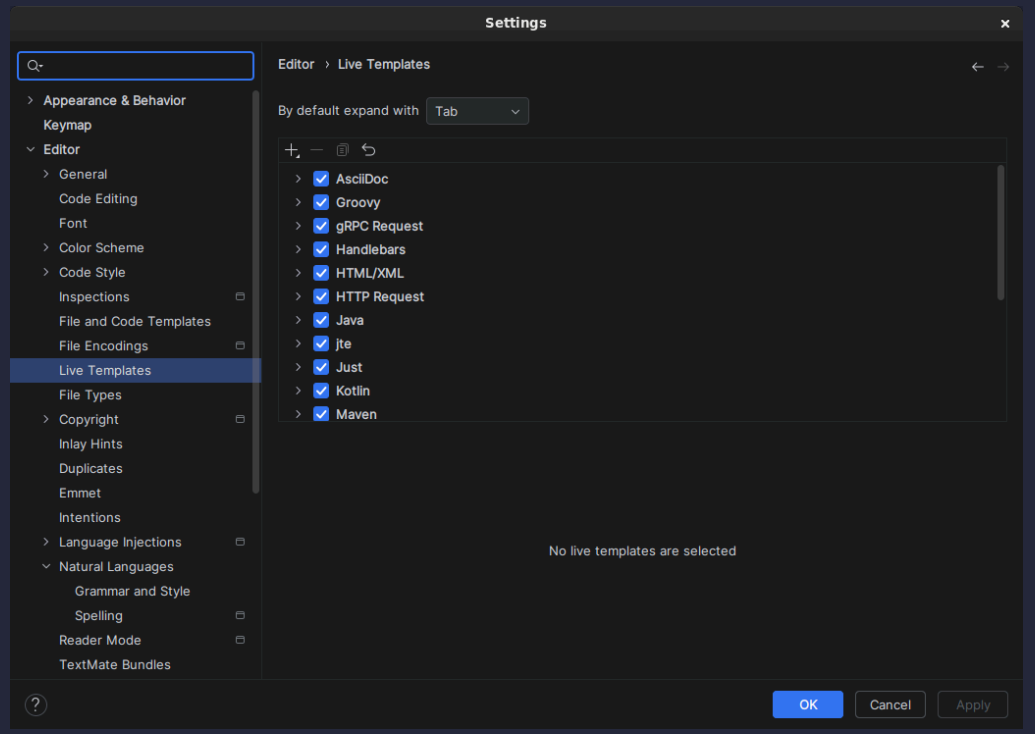
Exercice 1

Ajouter le snippet AAA dans votre IDE

Exercice 1 : Dans Intellij IDEA

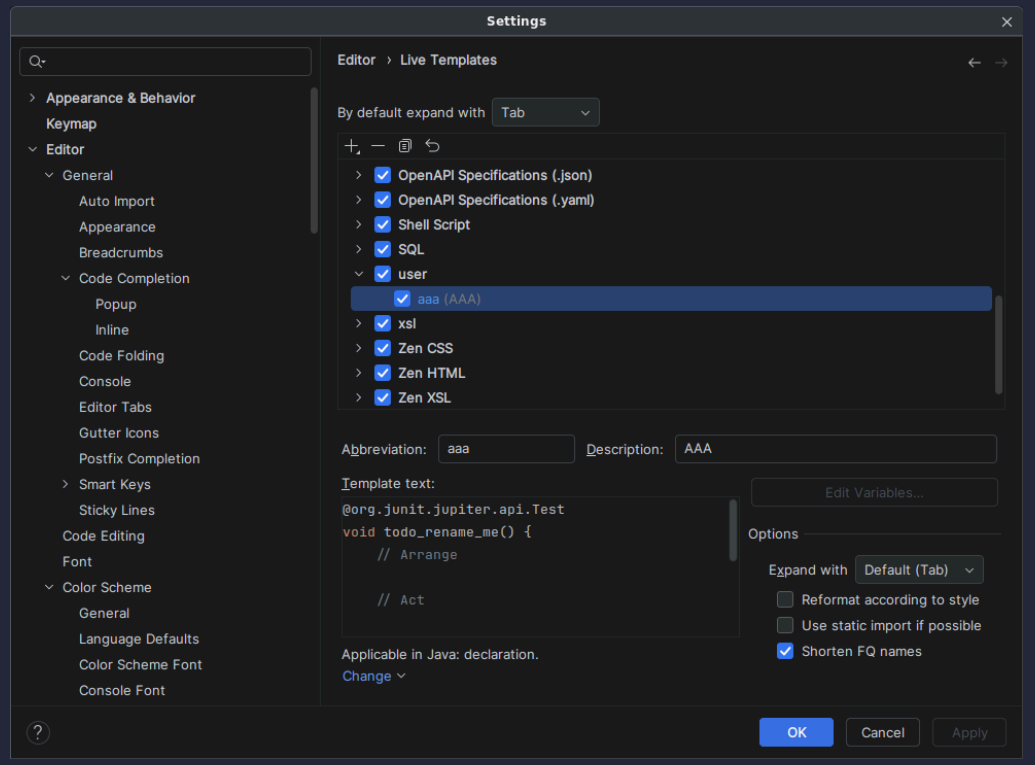
Aller dans :

- Settings
  - Editor
    - Live Templates



Exercice 1 : Dans Intellij IDEA

- Ajouter un Live Template



Exercice 1 : Dans Intellij IDEA

Mettre les valeurs :

- Abbreviation : `aaa`
- Template text :

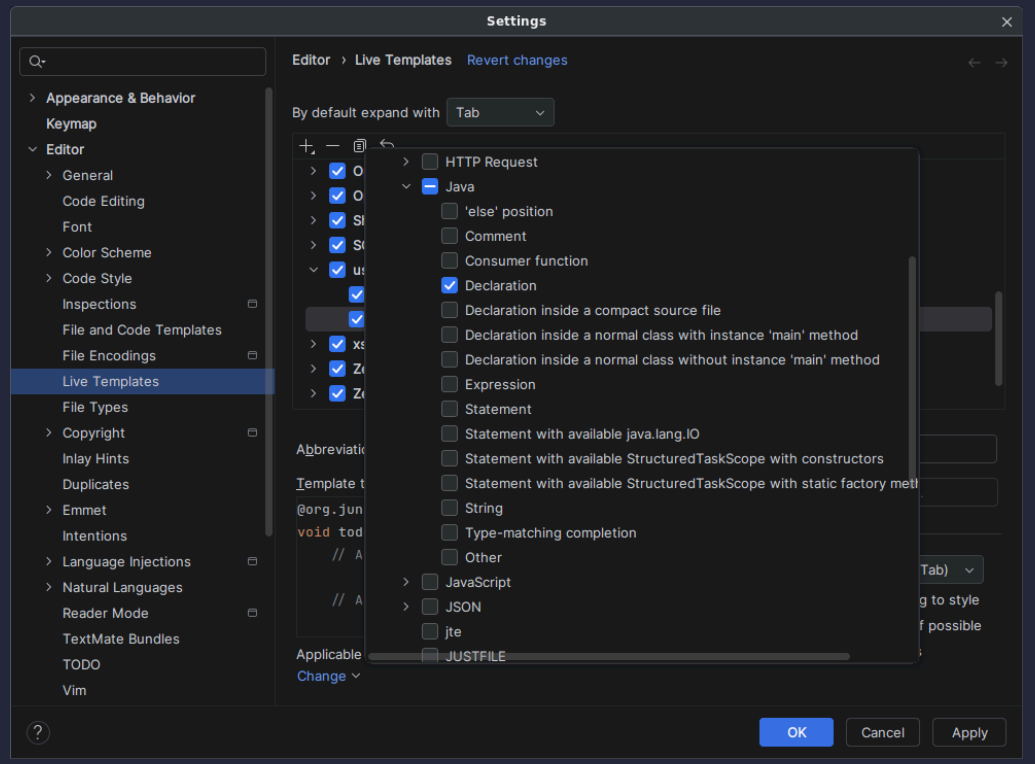
```
@org.junit.jupiter.api.Test
void todo_rename_me() {
    // Arrange

    // Act

    // Assert
}
```

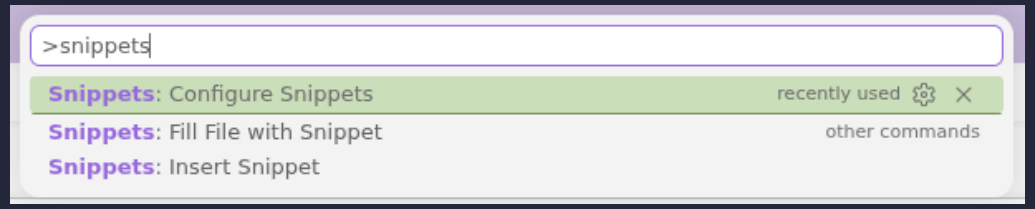
Exercice 1 : Dans Intellij IDEA

- Cocher l’option "Shorten FQ names"
- Sélectionenr le contexte suivant :
  - Java
    - Declaration

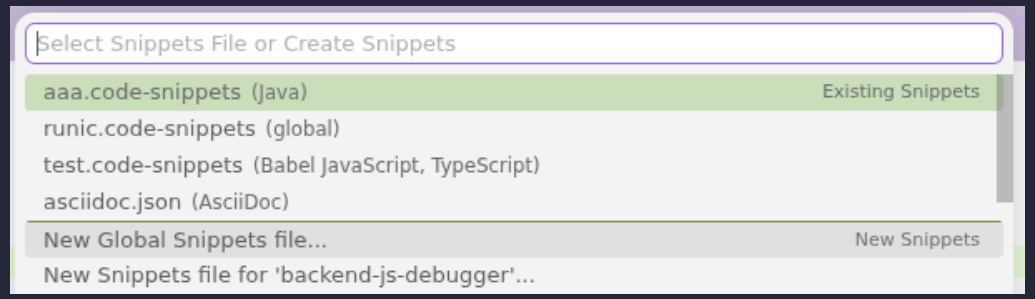


Exercice 1 : Dans VSCode

Depuis la palette de commande :



- Tapper "snippets"
- Choisir "Configure snippets"
- Puis choisir "New global snippets file"
- Donner un nom : `aaa`



Exercice 1 : Dans VSCode

Dans le fichier json qui s'ouvre dans votre IDE, ajouter l'entrée suivante :

```
"aaa": {
    "scope": "java",
    "prefix": "aaa",
    "body": [
        "@org.junit.jupiter.api.Test",
        "void todo_rename_me() {",
        "    // Arrange",
        "",
        "    // Act",
        "",
        "    // Assert",
        "}",
    ],
}
```

Exercice 2a : Lancer les tests

Fichier : `src/test/java/iut/rpg/ArmeTest.java`

- Lancer la suite de test via Maven :
  - `mvn test`
  - ou `./mvn.sh test`
- Lancer la suite de test via l'IDE

Exercice 2b : Implémenter le test

- Implémentez et exécuter le test
- Ajoutez un nouveau test
  - Pensez à utiliser le snippet AAA

Exercice 2c : Lancer le test en mode debug

- Via l’IDE, poser un point d’arrêt et lancer le test en mode debug

Exercice 3a : Implémenter le test

Fichier : `src/test/java/iut/rpg/JoueurTest.java`

- Implémentez les deux tests

Exercice 3b : Refactoring

- Utiliser `@BeforeEach` et `@AfterEach` pour simplifier les tests

Exercice 3c : Refactoring

- Utiliser AssertJ pour simplifier les tests

Documentation AssertJ

Exercice 3d : Refactoring

- Utiliser des factories pour générer les joueurs et les armes de test

Exercice 4 : Implémenter le test

Fichier : `src/test/java/iut/rpg/TweetJoueurTest.java`

- Implémenter le test
  - Vous aurez besoin d’un peu de refactoring

Exercice 5 : Coverage

Avec Maven :

- `mvn clean test jacoco:report`
- Ou `./mvn.sh clean test jacoco:report`
- Ouvrir le rapport `target/site/jacoco/index.html` dans votre navigateur

Exercice 5 : Coverage

Avec l'IDE :

- Exécuter les tests avec l'IDE
- Visualisez le coverage directement dans l'IDE

Tests d’intégration

Exercice 1 : Client REST

- Lancer l'API de demo "Rover API"
  `./mvn.sh compile exec:java`
  - Swagger : http://localhost:8080/swagger
  - Documentation : http://localhost:8080/redoc
  - JSON OpenAPI : http://localhost:8080/openapi
- Importer le fichier json dans votre client REST

Exercice 2 : Client REST

- Lancer des requêtes via votre client REST
- Écrire les tests suivants :
  - GET /rover retourne une liste non vide
  - GET /rover contient un rover du nom de "R2D2"

🔗 Documentation pour Bruno

Exercice 3 : Playwright

- Lancer Keyboard Factory
  - `npm install`
  - `npm run dev`
  - `npx playwright install`
- Lancer Playwright : `npx playwright test`

Exercice 4 : Playwright

- Ecrire les tests suivants [1] :
  - La page d'accueil affiche un panier
  - Le panier est vide
  - On peut ajouter un article dans le panier

1. https://playwright.dev/docs/writing-tests