Votre ami le débugger

Consignes

Les exercices portent sur la classe `Fibonacci` [1] et sa suite de tests unitaires [2]

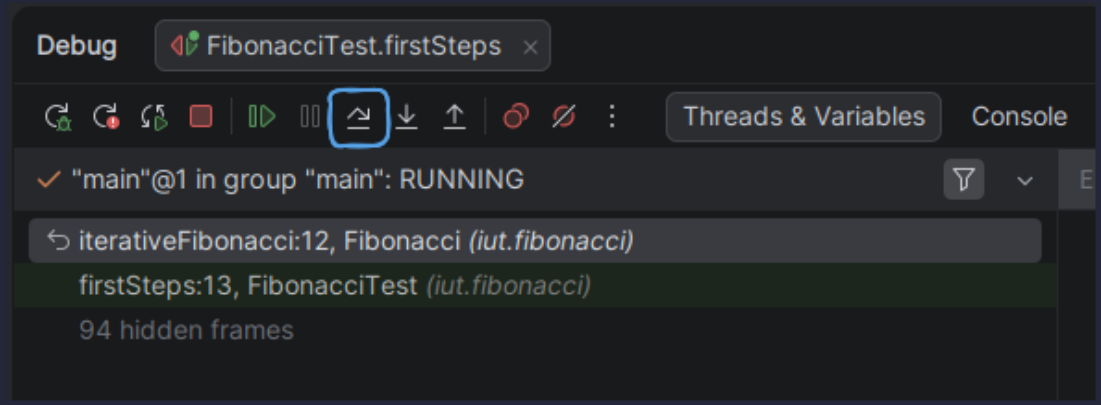
---

1. `exercices-java/src/main/java/iut/fibonacci`
2. `exercices-java/src/test/java/iut/fibonnaci`

Exercice 1 : découverte

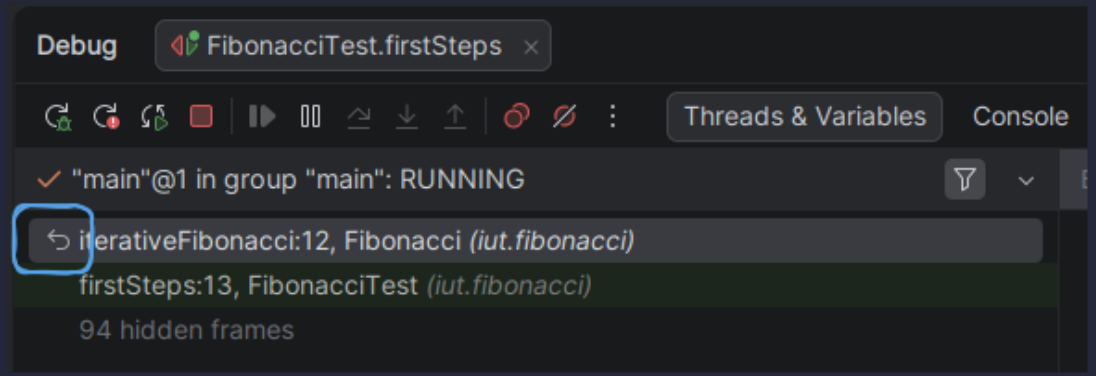
1.a : lancer le debugger

- Poser un breakpoint à la ligne 5
- Lancer le test `firstSteps` en mode debug
- Dérouler l'algorithme en pas à pas (F8)



1.b : revenir en arrière

- Poser un breakpoint à la ligne 12
- Lancer le test `firstSteps` en mode debug
- Depuis la pile d'appel, utiliser l'icône "retour arrière"

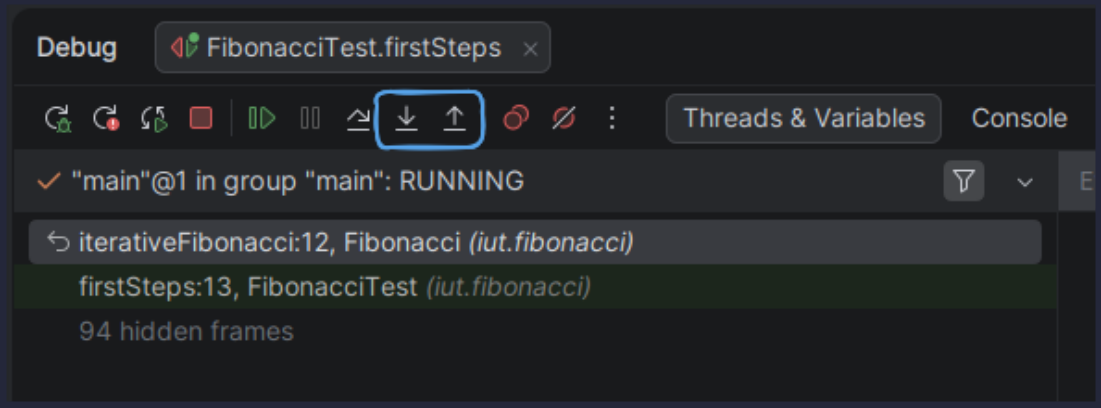


1.c : remonter la pile d'exécution

- Poser un breakpoint à la ligne 27
- Lancer le test `exploreCallStack` en mode debug
- Inspecter la pile d'exécution
- Cliquer sur les appels précédents dans la pile et inspecter les valeurs

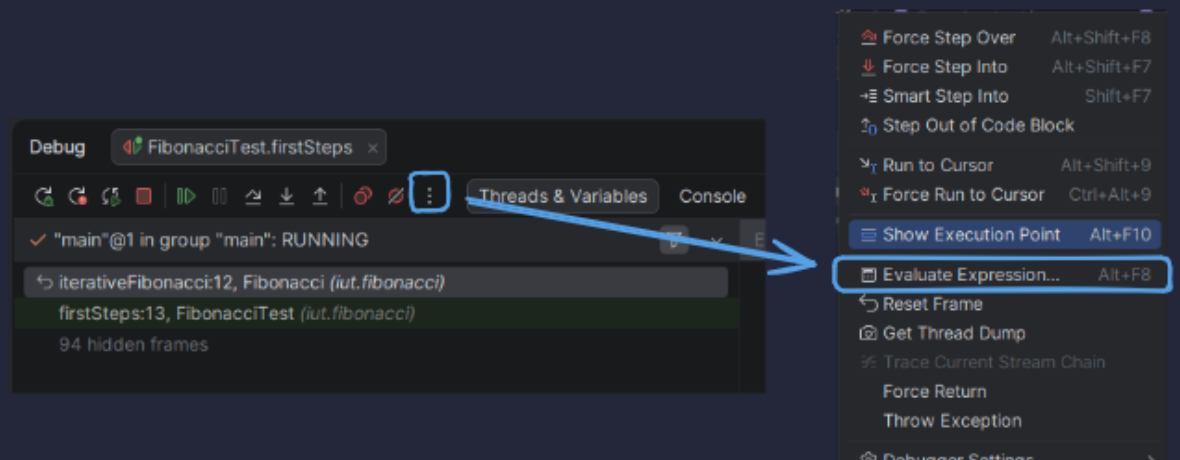
1.d : inspecter les fonctions

- Poser un breakpoint à la ligne 24
- Lancer le test `exploreCallStack` en mode debug
- Entrer dans la méthode avec "Step into" (F7)
- Sorter de la méthode avec "Step out" (Shift+F8)



1.e : exécuter du code

- Poser un breakpoint à la ligne 24
- Lancer le test `exploreCallStack` en mode debug
- Ouvrir la fenêtre de "Evaluate expression" (Alt+F8)
- Évaluer la fonction `recursiveFibonacci(n - 1);`



1.f : utiliser un watcher

- Poser un breakpoint à la ligne 14
- Lancer le test `firstSteps` en mode debug
- Ajouter un watcher pour évaluer la formule : `n0 + n1`
- Laisser la boucle avancer et s'arrêter sur le breakpoint (F9)
- Visualiser le watcher qui change

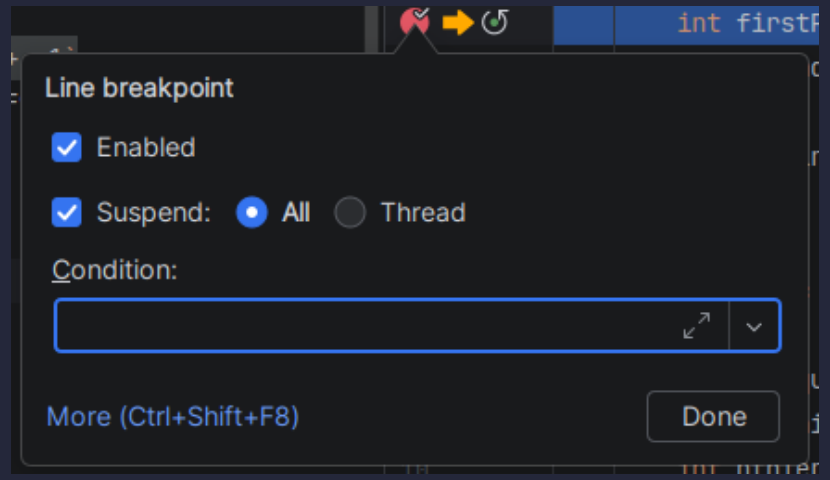
1.g : stopper l'exécution à la volée

- Ne pas poser de breakpoint
- Lancer le test `longExecutionTest` en mode debug
- Appuyer sur le bouton pause
- Inspecter où en est l'algorithme

Exercice 2 : techniques avancées

2.a : le breakpoint conditionnel par valeur

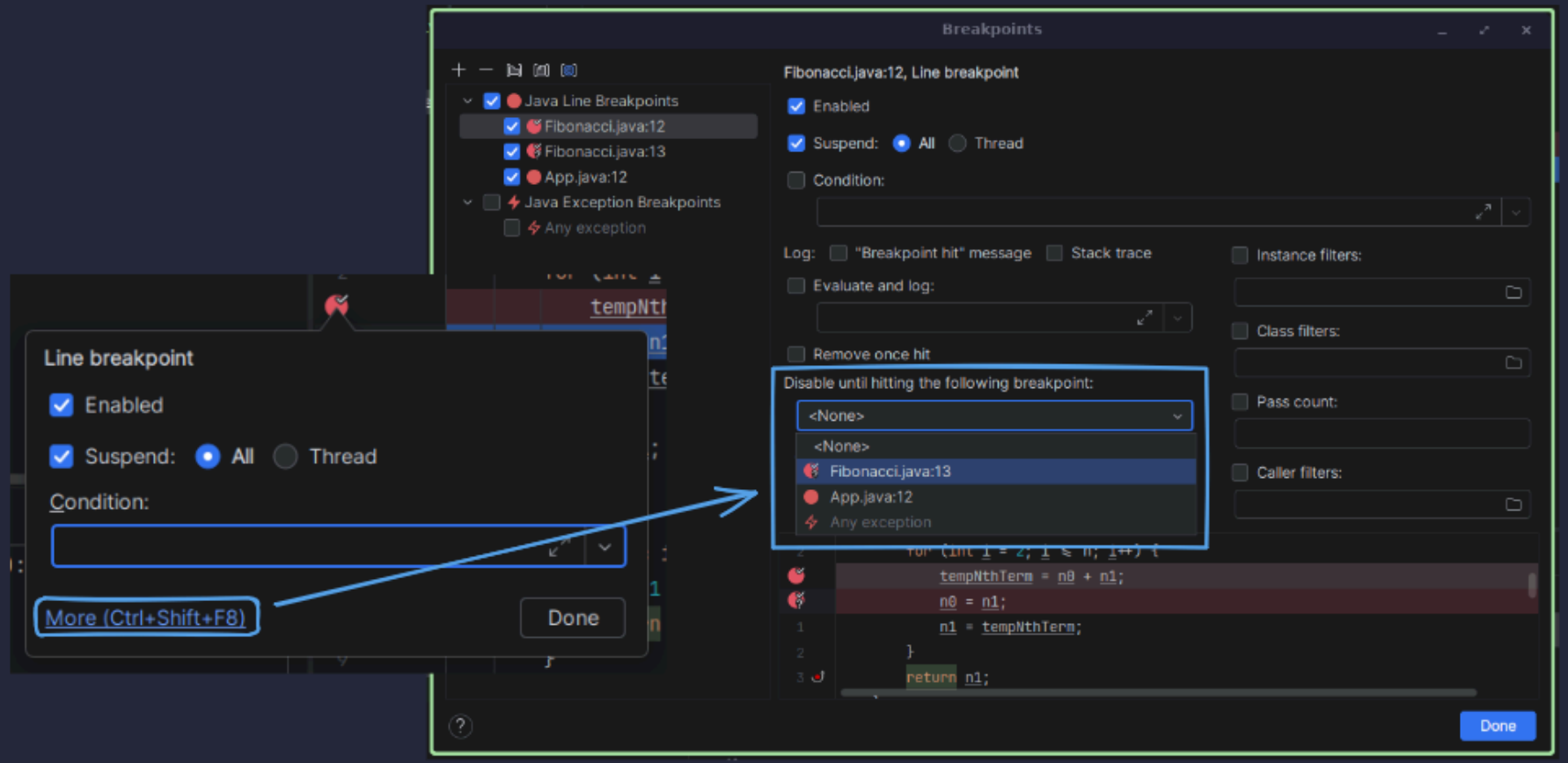
- Poser un breakpoint à la ligne 14
- Editer le breakpoint pour ajouter la condition `i == n`
- Lancer le test `firstSteps` en mode debug



2.b : le breakpoint conditionnel par breakpoint

- Poser un breakpoint à la ligne 13
  - Lui assigner la condition `tempNthTerm == 1`
- Poser un breakpoint à la ligne 12
  - Clic droit sur le breakpoint, puis cliquer sur "More" (ou Shift+Ctrl+F8),
  - Sélectionner le breakpoint de la ligne 13 dans le menu déroulant "Disable until hitting the following breakpoint"

2.b : le breakpoint conditionnel par breakpoint



2.b : le breakpoint conditionnel par breakpoint

- Lancer le test `firstSteps` en mode debug
- Observer le déroulement
  - D’abord de breakpoint ligne 13, sans déclencher celui de la ligne 12
  - Puis, à l’itération suivante, celui de la ligne 12

2.c : le breakpoint conditionnel par exception

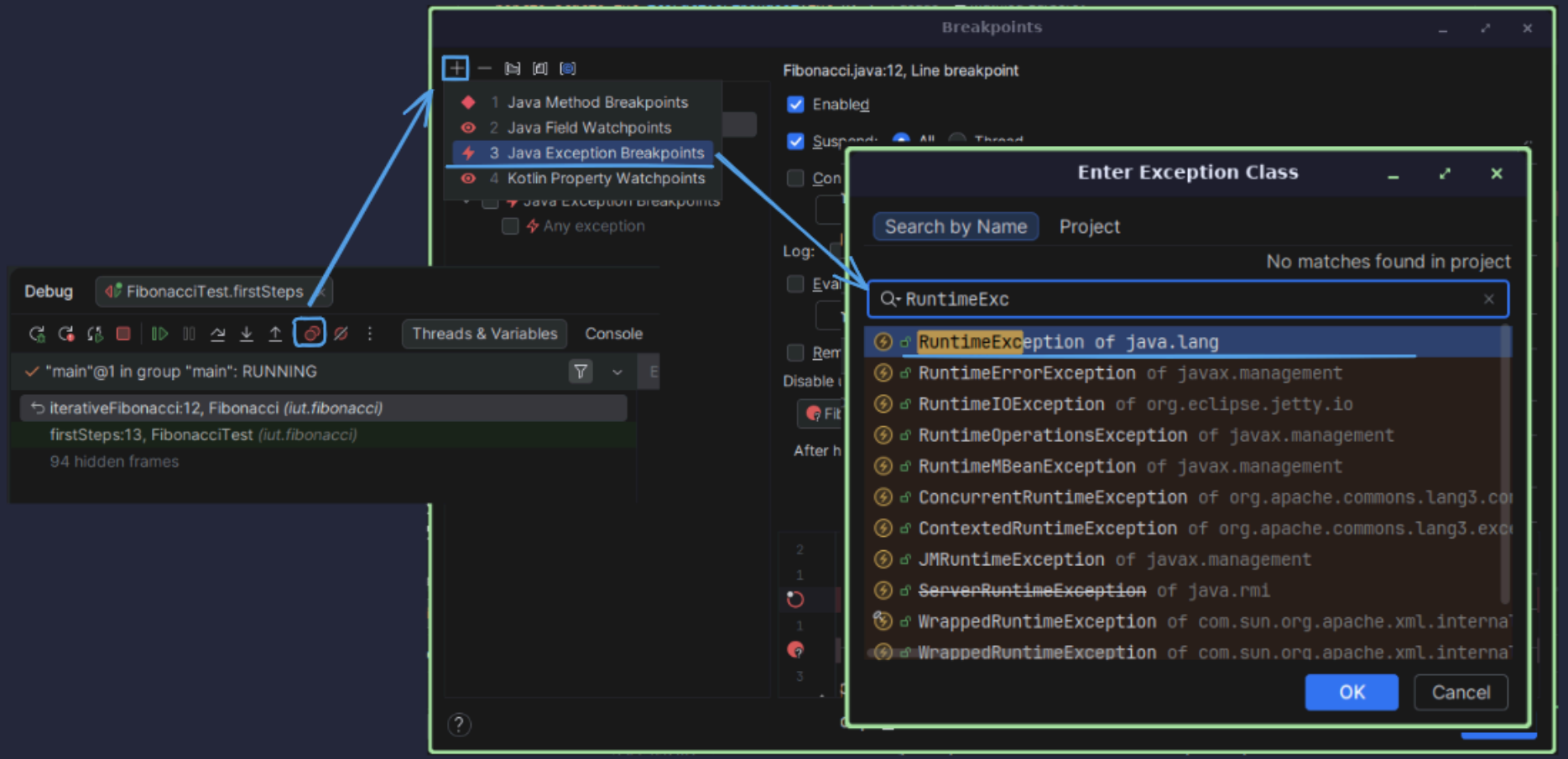
- Modifier le code de la fonction `recursiveFibonacci` pour lever une exception :

```
if (n == 1 || n == 0) {
    throw new RuntimeException("Ooops");
}
```

2.c : le breakpoint conditionnel par exception

- Ajouter un breakpoint sur l’exception :
  - Ouvrir le menu des breakpoint (Shift+Ctrl+F8)
  - Créer manuellement un nouveau breakpoint avec le menu "+"
  - Choisir l’exception dans le menu

2.c : le breakpoint conditionnel par exception



2.c : le breakpoint conditionnel par exception

- Lancer le test `exploreCallStack` en mode debug
- Observer le comportement :
  - Le programme s’arrête automatiquement sur l’erreur levée ligne 21

2.d : modifier une variable à chaud

- Poser un breakpoint à la ligne 14
- Lancer le test `firstSteps` en mode debug
- Modifier la variable `i` pour lui affecter la valeur `5`
- L’exécution sort de la boucle plus tôt que prévu et le test échoue

2.e : altérer le flux d’exécution

- Poser un breakpoint à la ligne 9
- Lancer le test `firstSteps` en mode debug
- Changer le flux d’exécution :
  - Clic-droit sur la ligne 16
  - "Skip to cursor"
- L’exécution sort sans passer par la boucle et le test échoue

3 : Appliquer sur un de vos projets

- Lancer un de vos projets en mode debug
- Poser et déclencher les breakpoints des exercices précédents dans votre code