AutoParams

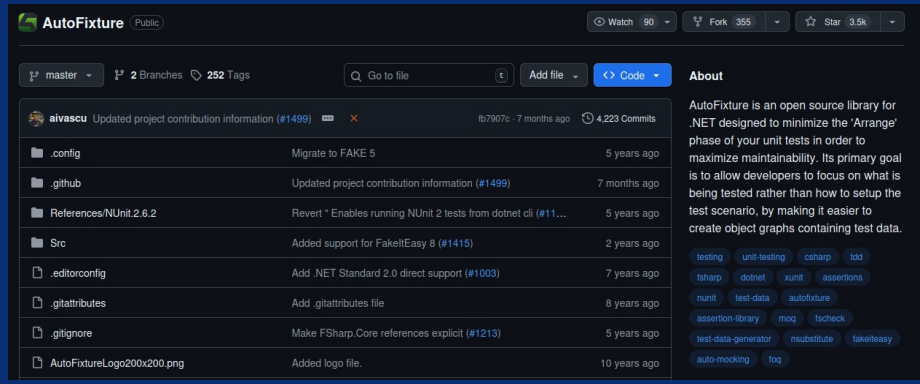
pour les tests paramétrés en Java

Simplifiez vos tests unitaires

Concept AutoParams

AutoParams est une extension pour JUnit 5 inspirée par AutoFixture, automatisant la génération des données de test.

Il élimine le code répétitif en injectant automatiquement des valeurs dans les paramètres des méthodes de test.



https://github.com/AutoFixture/AutoFixture

Dans pom.xml avec maven:

```xml
<dependency>
  <groupId>io.github.autoparams</groupId>
  <artifactId>autoparams</artifactId>
  <version>11.3.1</version>
  <scope>test</scope>
</dependency>
```

[2]

Quelques fonctionnalités d’AutoParams (annotations)

@AutoSource
Test paramétré : génère automatiquement des valeurs aléatoires pour chaque paramètre
(la fonctionnalité la plus courante).

@Repeat(n)
Exécute un test plusieurs fois avec de nouveaux arguments aléatoires à chaque fois
(utile pour les tests de charge ou de robustesse).

@ValueAutoSource
Test paramétré : permet de spécifier des valeurs fixes tandis qu’AutoParams génère les autres.

@CsvAutoSource
Test paramétré : permet de passer des valeurs fixes au format CSV pour certains paramètres, tout en générant les autres.

```
@AutoSource
@Repeat(10)
void testMethod(int a, int b) {
    Calculator sut = new Calculator();
    int actual = sut.add(a, b);
    assertEquals(a + b, actual);
}
```

```
@ValueAutoSource(ints = { 1, 2, 3 })
```

```
@CsvAutoSource({
    "Product 1, 1000",
    "Product 2, 100"
})
```

[3]

Quelques fonctionnalités d’AutoParams

@AutoParams
Fonctionne avec un simple @Test : injecte automatiquement des arguments générés dans des méthodes de test normales (pas besoin d’un test paramétré si on teste avec un objet, il va regarder les attributs dans le constructeur).

@Customization(...)
Spécifie des contraintes personnalisés à un test (par ex. forcer un âge entre 1 et 120, ou un certain format de string).
(voir la documentation d’AutoParams: https://github.com/AutoParams/AutoParams)

Annotations de contraintes (@Min, @Max, @NotBlank, @Size, etc.)
À mettre directement sur les paramètres de test pour restreindre les valeurs générées (ex. @Min(1) @Max(120) int age). Fonctionne comme des contraintes de validation.
(Ajouter la dépendance jakarta.validation-api)

```
@Test
@AutoParams
void testMethod(@Min(1) @Max(120) int age) {
    assertTrue(value >= 1);
    assertTrue(value <= 120);
}
```

Quelques fonctionnalités d’AutoParams

@Freeze
AutoParams crée un seul objet, et tous les tests qui en ont besoin reçoivent cette même instance.

@FreezeBy
Version configurable de @Freeze: peut choisir comment AutoParams partage une instance. Peut freeze selon:

EXACT_TYPE: Tous les paramètres du même type exact partagent une seule instance
PARAMETER_NAME: distinguer parmis plusieurs paramètres du même type
IMPLEMENTED_INTERFACES: Partage une instance entre un paramètre et toutes les variables dont le type est une interface que cet objet implémente.
ASSIGNABLE_TYPES: Si l’objet peut être affecté (assignable) à un paramètre, il est partagé

```
@Test
@AutoParams
void test(@Freeze Product product, Review r1,
Review r2) {
...
}
```

Démonstration

https://github.com/will13cb/autoparams-demo.git

Person

```java
package com.demo;

public class Person {  2 usages  William Caron-Bastarache
    private final String name;  3 usages
    private final int age;  3 usages

    public Person(String name, int age) {  no usages  William Caron-Bastarache
        this.name = name;
        this.age  = age;
    }

    public String name() { return name; }  no usages  William Caron-Bastarache
    public int age()     { return age; }  no usages  William Caron-Bastarache

    @Override public String toString() { return "Person{name='%s', age=%d}".formatted(name, age); }
}
```

Test1

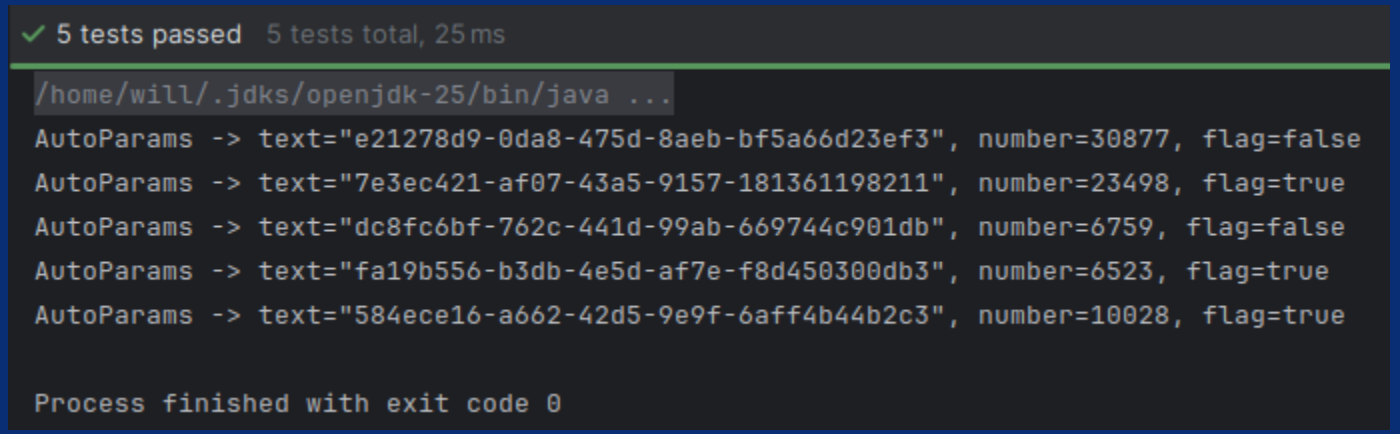
```java
// Génère des valeurs string, int, bool

package com.demo;

import ...

class Test1 {  William Caron-Bastarache +1

    @ParameterizedTest(name = "[{index}] text={0}, number={1}, flag={2}")  William Caron-Bastarache
    @AutoSource
    @Repeat(5)
    void demo_generated_values(String text,int number, boolean flag) {
        System.out.printf("AutoParams -> text=\"%s\", number=%d, flag=%s%n",
                text, number, flag);
    }
}
```



Test2

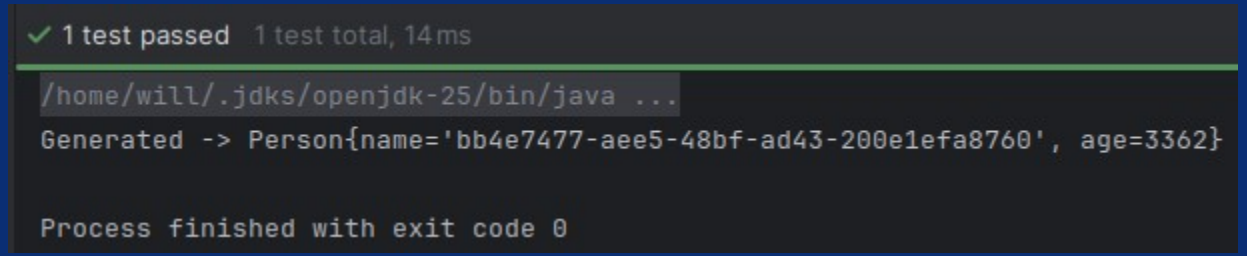
```java
// Génère des objets

package com.demo;

import ...

class Test2 {  William Caron-Bastarache +1 *

    @ParameterizedTest(name = "[{index}] Generated person: {0}")  William Caron-Bastarache
    @AutoSource
    void auto_generates_person(Person person) {
        System.out.println("Generated -> " + person);
    }
}
```



Test3

```java
// Mix entre valeurs fixes et générées

package com.demo;

import ...

class Test3 {  will13cb

    @ParameterizedTest  will13cb
    @ValueAutoSource(ints = {18, 30, 65})
    void mixes_fixed_and_auto(int age, String name) { System.out.printf("Fixed age=%d, Random name=%s%n", age, name); }
}
```

✓ 3 tests passed 3 tests total, 24 ms

```
/home/will/.jdks/openjdk-25/bin/java ...
Fixed age=18, Random name=2b40534d-2aca-4fcf-ab59-725cd766915d
Fixed age=30, Random name=bd944eef-b7bb-4a1f-923e-e7ac75ed1841
Fixed age=65, Random name=645ec39a-c925-4f05-bba1-33b50765e2cb

Process finished with exit code 0
```

Réflexion & Recul

Productivité

Moins de temps perdu à fabriquer des données de test
Moins de code à maintenir
Plus de focus sur la logique de test
Facilite l'ajout de nouveaux cas

Fiabilité

Tests plus exhaustifs grâce à une variété de données
Réduction des erreurs humaines
Meilleure couverture des cas limites

Limite pratique

- Nécessite des tests ciblés pour scénarios critiques

Conclusion

- Simplifie la création de données de test
- Réduit le code répétitif
- S'intègre parfaitement avec JUnit 5
- Offre des fonctionnalités avancées comme @Freeze, @Customization et @ValueAutoSource

Simple:

- Ajoutez les dépendances nécéssaires
- Ajoutez les annotations que vous voulez utiliser (@)
- Roulez les tests

Merci pour votre attention !

Sources

Documentation officielle AutoParams

Dépôt GitHub AutoParams

Chatgpt (explication des fonctionnalités et du code)