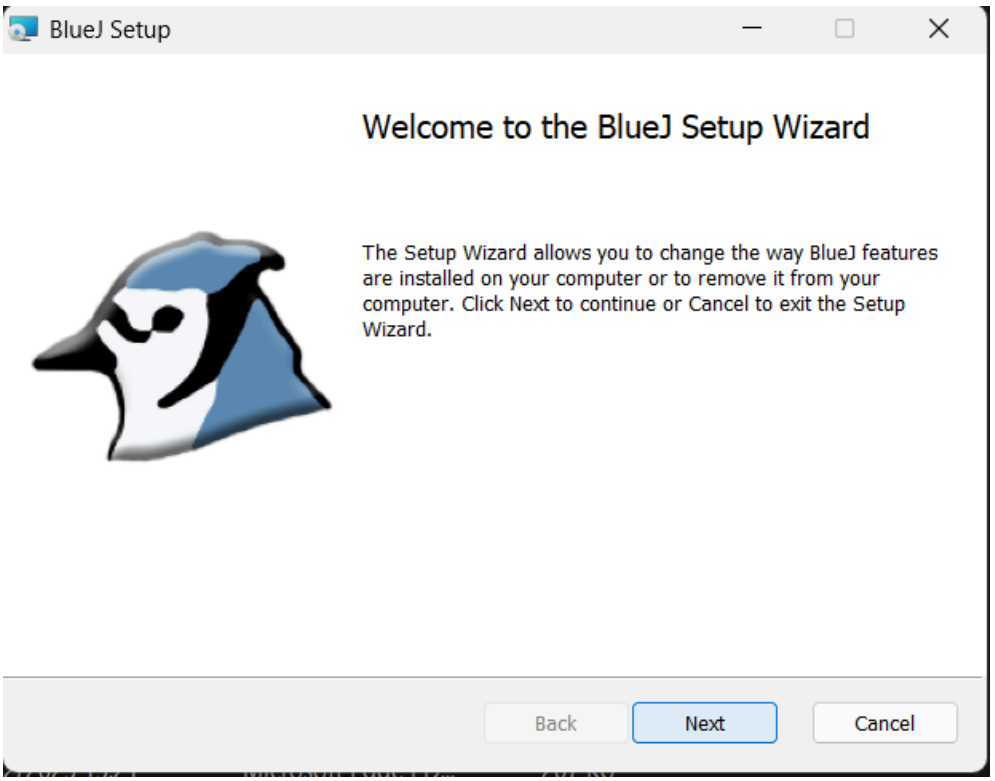
Tutoriel BlueJ : Premiers pas ! 🚀

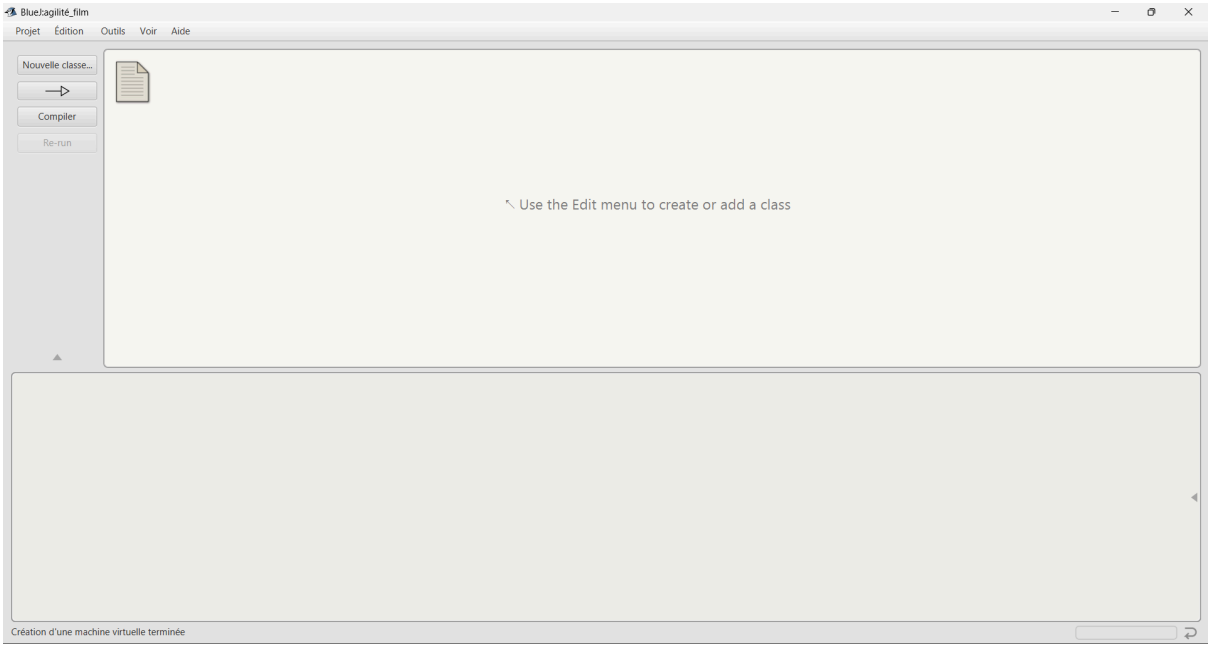
Installation 🛠️

Pour commencer, rendez-vous sur la page web BlueJ.org et installez le sur la plateforme de votre choix. **C'est parti !** ➡️

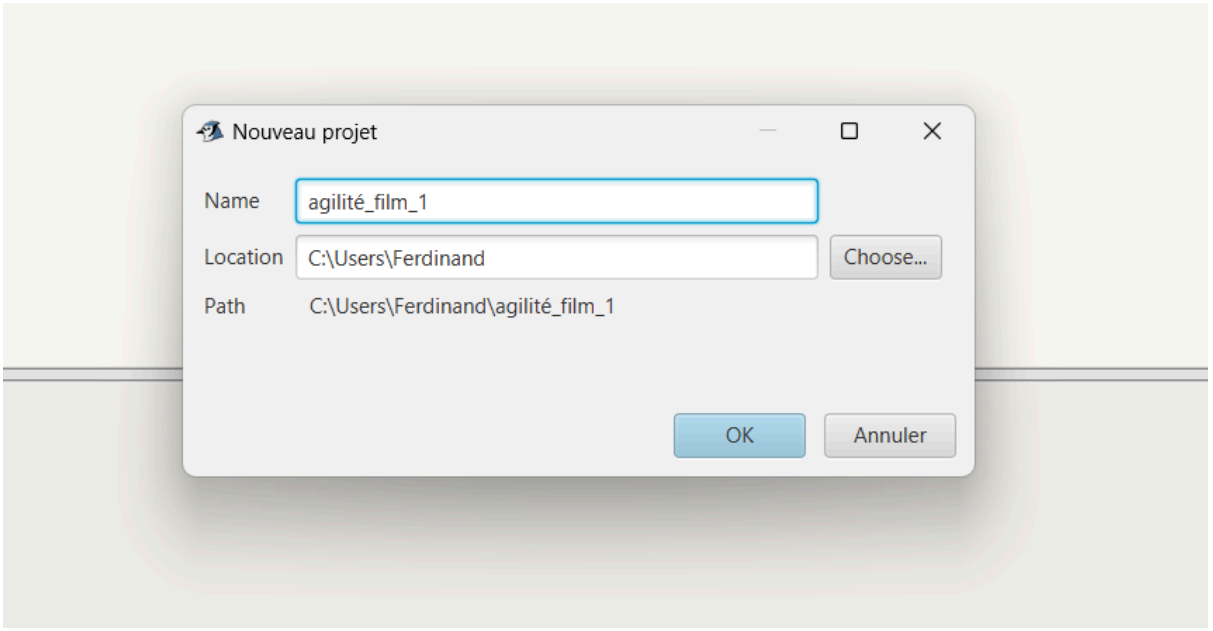


Une fois BlueJ installé, il vous suffit de finir l'installation grâce au Setup Wizard en cliquant sur Next.

Votre premier projet sur BlueJ ✨

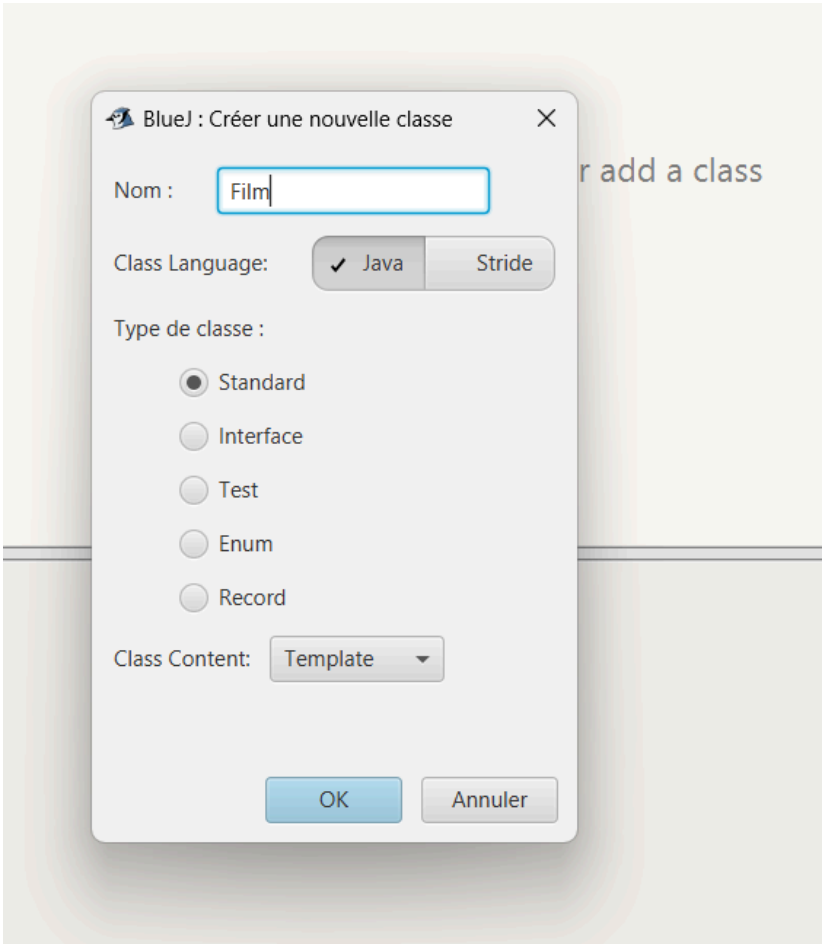


Ouvrez l'application. Pour créer votre premier projet, il vous suffit de cliquer sur le bouton projet en haut à gauche et de sélectionner "nouveau projet".



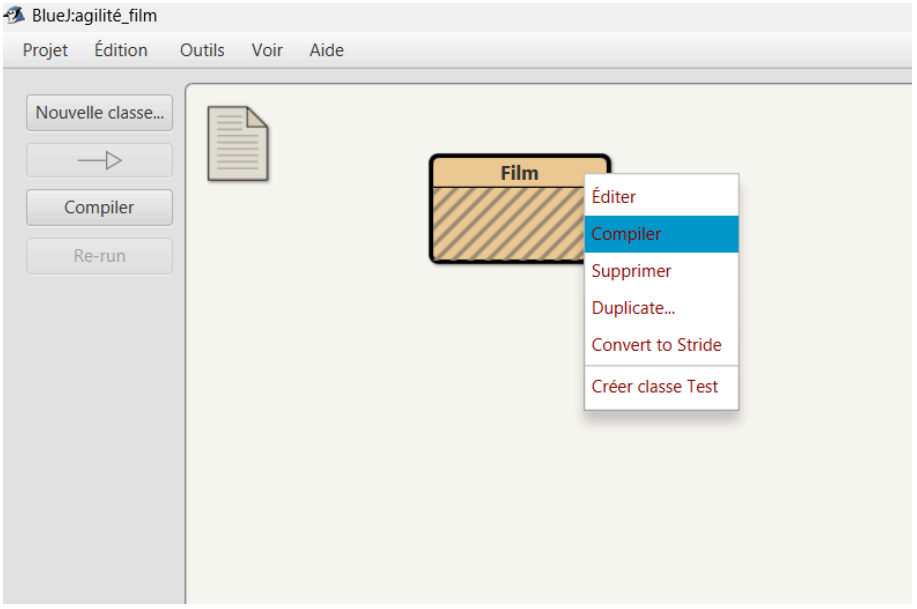
Via cette fenêtre, vous pouvez choisir le nom de votre projet ainsi que l’endroit dans lequel il sera localisé.

Votre première Classe : le cœur de votre projet ! ❤️



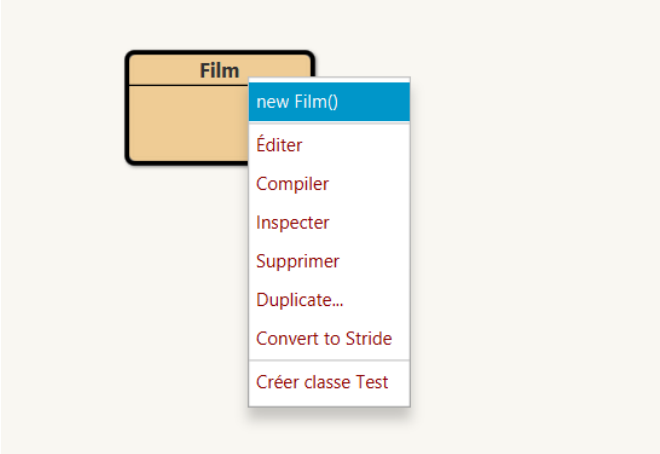
Pour créer la première classe de votre projet, c'est-à-dire un type d'objets, il vous suffira de cliquer sur le bouton "Nouvelle classe" dans le bandeau sur la gauche puis cette fenêtre apparaîtra. Vous pourrez choisir votre langage ainsi que le type de votre classe.

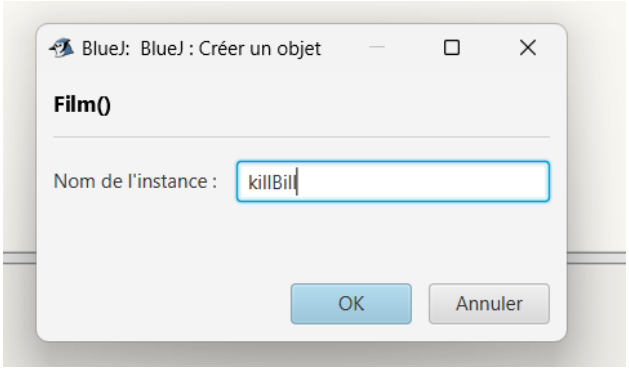
Les Classes : la boîte à outils pour vos Films ! 🎬



Vous venez de créer votre première classe, elle servira de template pour créer tous vos films préférés ! Elle correspond en quelque sorte à une boîte dans laquelle vous rangerez vos films. Elle vient d'apparaître sur l'écran d'accueil. En effectuant un clic droit, plusieurs actions s'offrent à vous. Nous allons dans cette étape compiler notre classe, c'est-à-dire ouvrir notre boîte (qui sera vide).

C'est l'heure de créer votre premier Film ! 🎥





Une fois notre boîte ouverte, nous voulons y ranger notre premier film. Pour cela, cliquez sur “new Film()” puis choisissez le nom du premier film à ranger dans notre boîte.
On peut le voir dans la partie du bas :

Premier film créé, bravo ! 🥳

Donnez de la personnalité à votre film : ajoutez des Attributs ! 🌟

```java
import java.util.Date;

public class Film
{
    private int duree;
    private String dateSortie;
    private Categorie categorie;

    public Film()
    {
        // initialisation des variables d'instance
        duree = 0;
        dateSortie = "2001/01/01";
    }

    public int getDuree()
    {
        return this.duree;
    }

    public void setDuree(int duree)
    {
        this.duree = duree;
    }

    public String getDateSortie()
    {
        return this.dateSortie;
    }

    public void setDateSortie(String dateSortie)
    {
        this.dateSortie = dateSortie;
    }
```

En cliquant sur “Éditer” on peut accéder au code de la classe. Ce n’est pas la partie la plus importante à comprendre, il faut simplement respecter la syntaxe Java pour ajouter des attributs et méthodes.

Pour pouvoir mieux décrire nos objets “Films”, on décide d’ajouter deux caractéristiques : la durée et la date de sortie. On peut voir l’ajout de ces deux caractéristiques ici :

```java
public class Film
{
    private int duree;
    private String dateSortie;
```

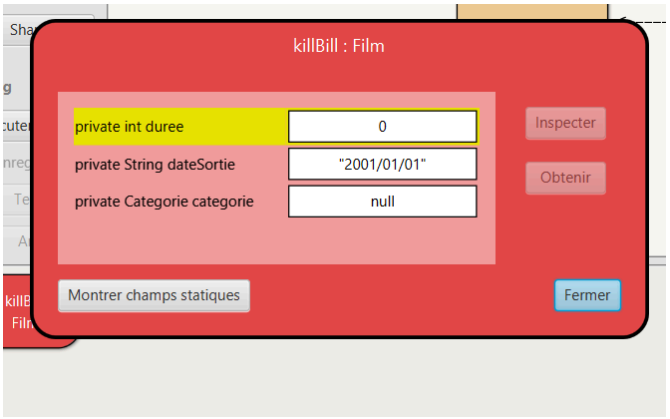
Ensuite, on aimerait pouvoir accéder et modifier ces valeurs, alors on crée des “méthodes”, comme ceci :

```
public int getDuree()
{
    return this.duree;
}

public void setDuree(int duree)
{
    this.duree = duree;
}
```

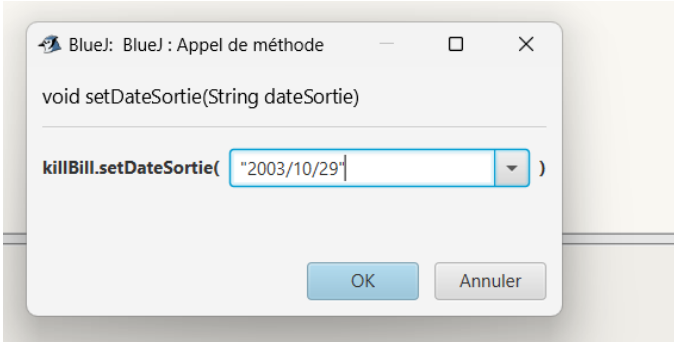
La première fonction getDuree() permet de récupérer la durée, et la seconde permet de la modifier.

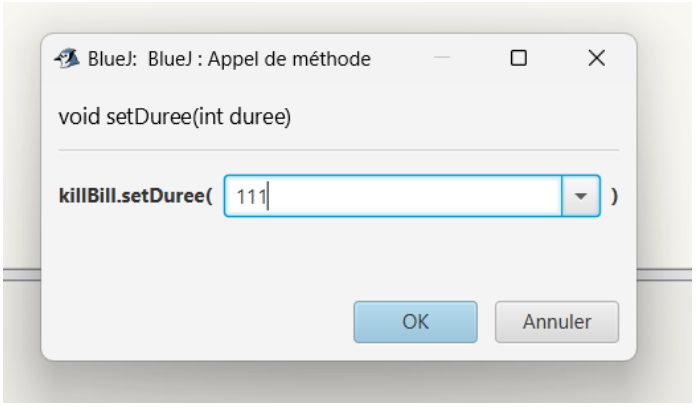
Testons un peu tout ça ! (Nouvelles modifications) ✅

Afin d'essayer ces nouveautés, créons à nouveau un film ! Pour cet exemple, j'ai encore créé le film "Kill Bill". En cliquant dessus et sur "Inspecter", on voit les valeurs par défaut quand on crée un film :

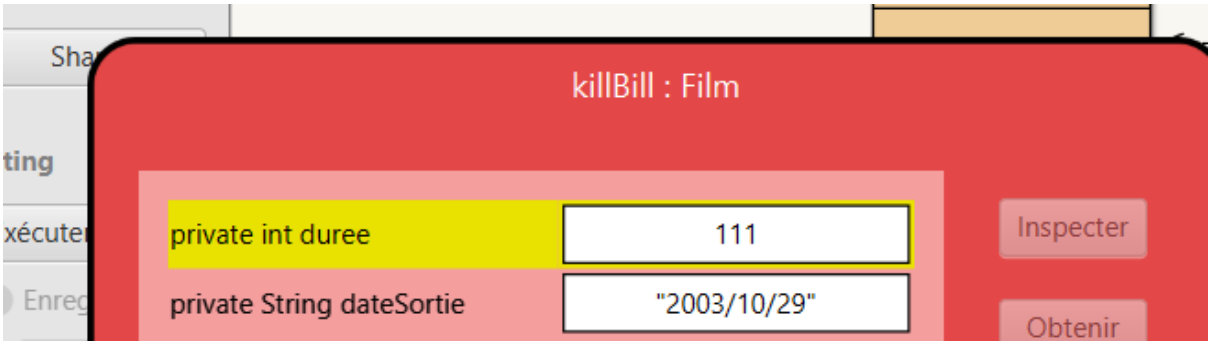


Mais Kill Bill dure plus longtemps que 0 minutes et n'est pas sorti en 2001… nous allons alors utiliser les méthodes que nous venons de coder.





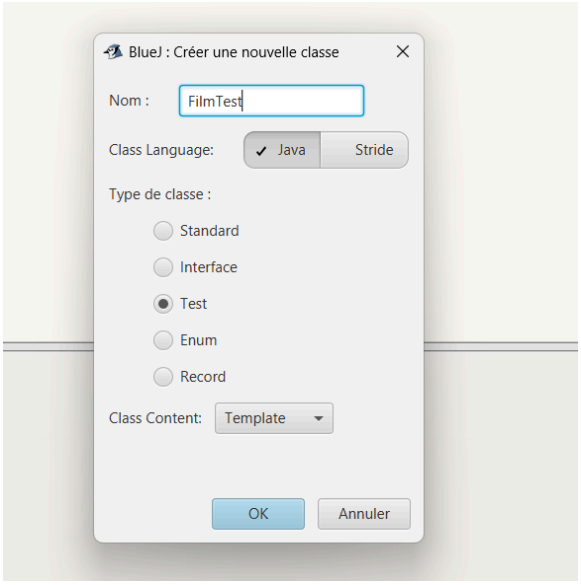
Pour que notre film ait les bonnes caractéristiques, nous allons utiliser les "méthodes" qui sont nos outils de modifications pour renseigner la bonne date de sortie ainsi que la bonne durée pour notre film.



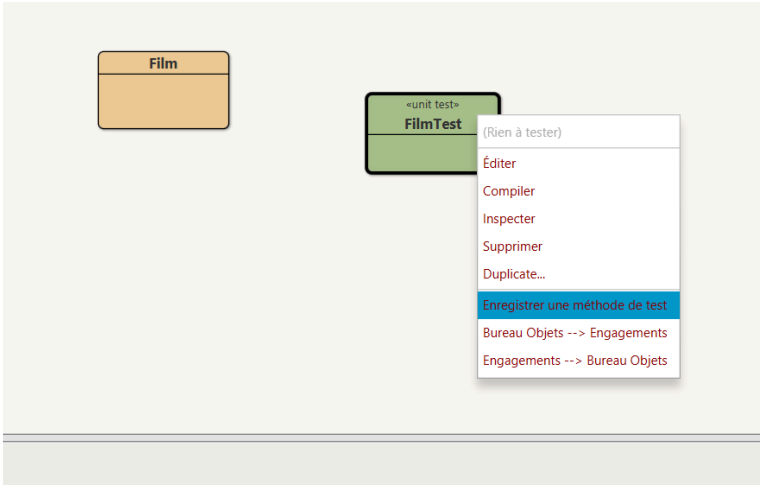
Une fois nos modifications appliquées, nous pouvons les consulter en cliquant sur le film et nous pouvons observer que les informations sont maintenant correctes. **Super !**

Passez à la vitesse supérieure avec une Classe de Test automatique ! 🤖

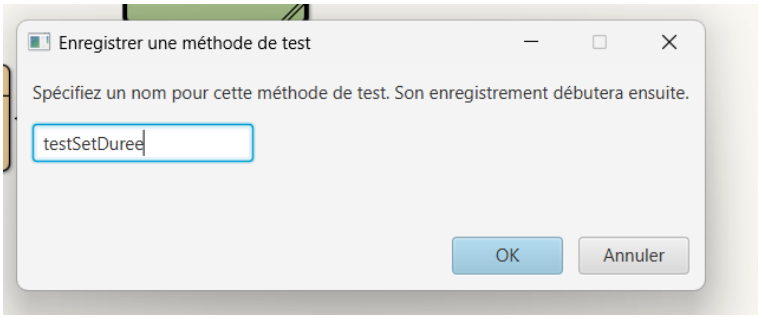
Le test manuel a bien fonctionné mais BlueJ permet d'automatiser ce genre d'actions. Créons une nouvelle classe, comme tout à l'heure pour les Films, mais cette fois, utilisons le type de classe "Test" dont le nom révèle la fonction.



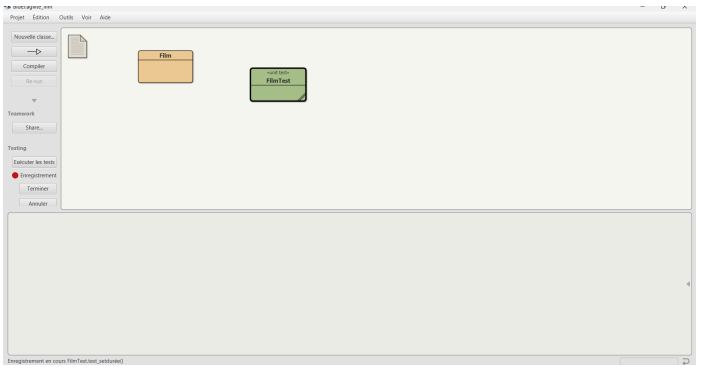
Cela ajoute une case dans la partie du haut de l'application. Nous allons montrer à BlueJ les actions à réaliser pour faire un test et il va les convertir en code qu'il pourra exécuter tout seul par la suite. En faisant un clic droit sur la nouvelle classe de Test, on peut choisir l'option "Enregistrer une méthode de test".



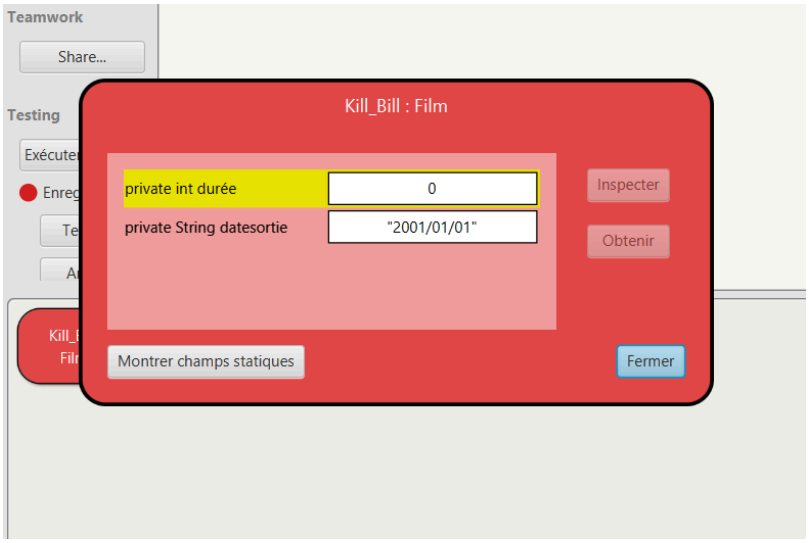
Ici nous allons choisir de tester l'attribut "duree" d'un film. J'appelle donc la méthode : "testSetDuree".



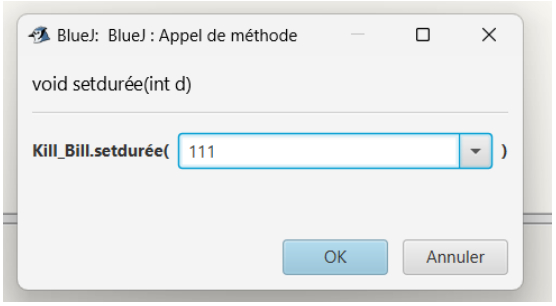
Après avoir cliqué sur "OK", un point rouge apparaît à gauche. Cela signifie que BlueJ enregistre toutes nos actions !



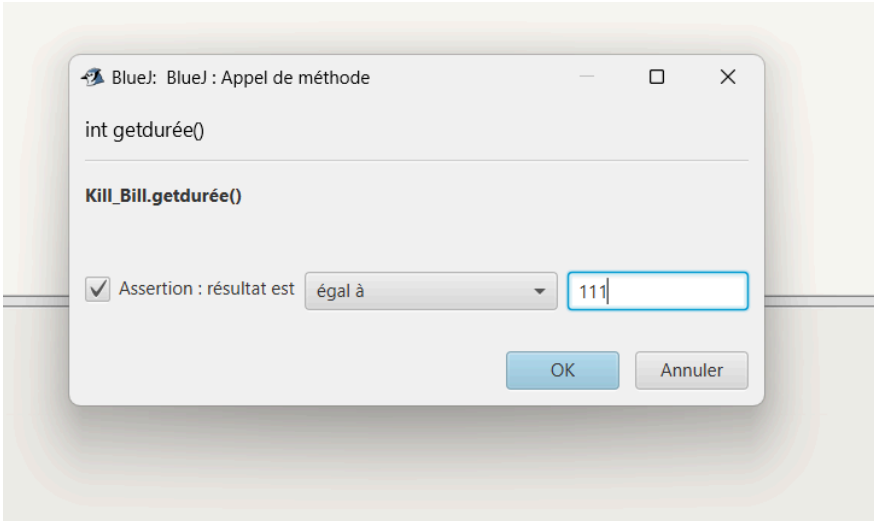
Recommençons nos actions de tout à l'heure : on crée un film, et on l'inspecte.



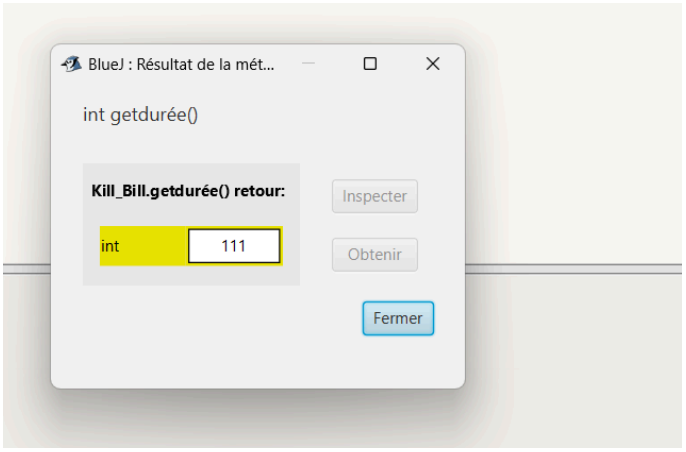
On voit bien les valeurs par défaut.
J'utilise la méthode "setDuree" et je mets la durée du premier volume de Kill Bill : 111 minutes.



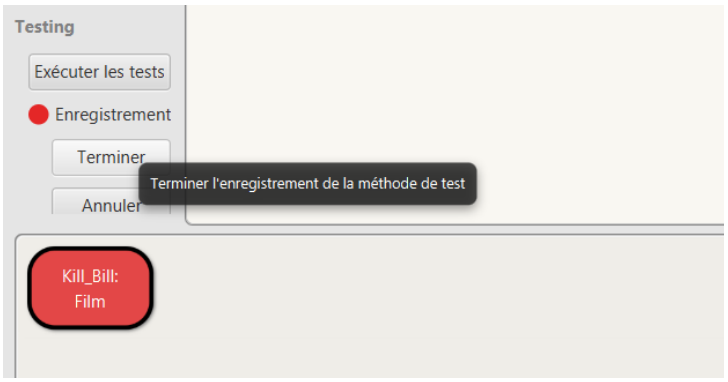
Maintenant, en lançant la méthode "getdurée()" pour vérifier que la modification a été prise en compte, une nouvelle option apparaît : **l'assertion**. On doit vérifier à quoi le résultat est égal, donc ici on attend la valeur 111 minutes.



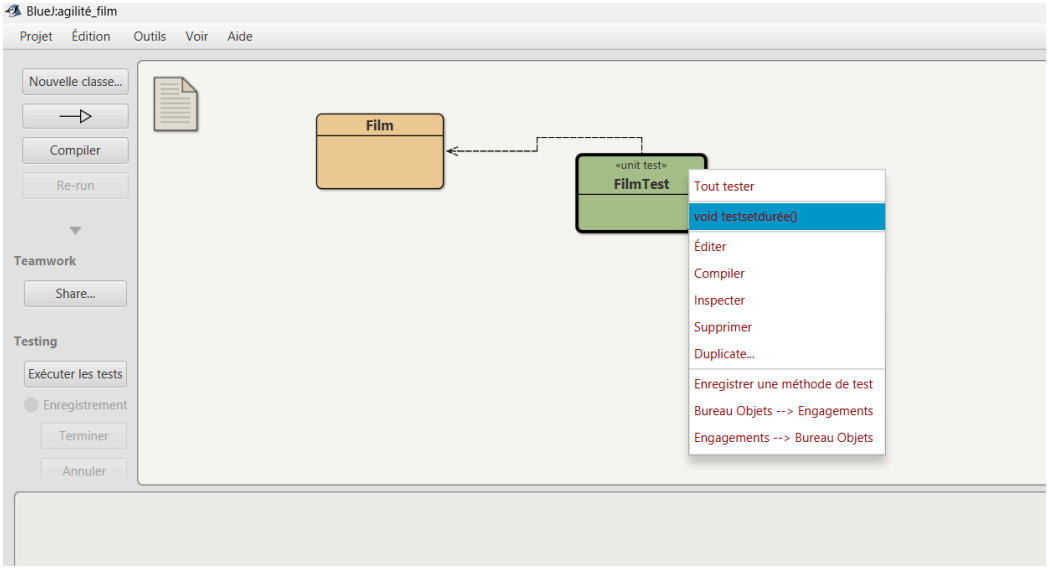
En cliquant sur "OK" on voit l'écran suivant où la valeur retournée est bien 111, **bravo !** 🎊

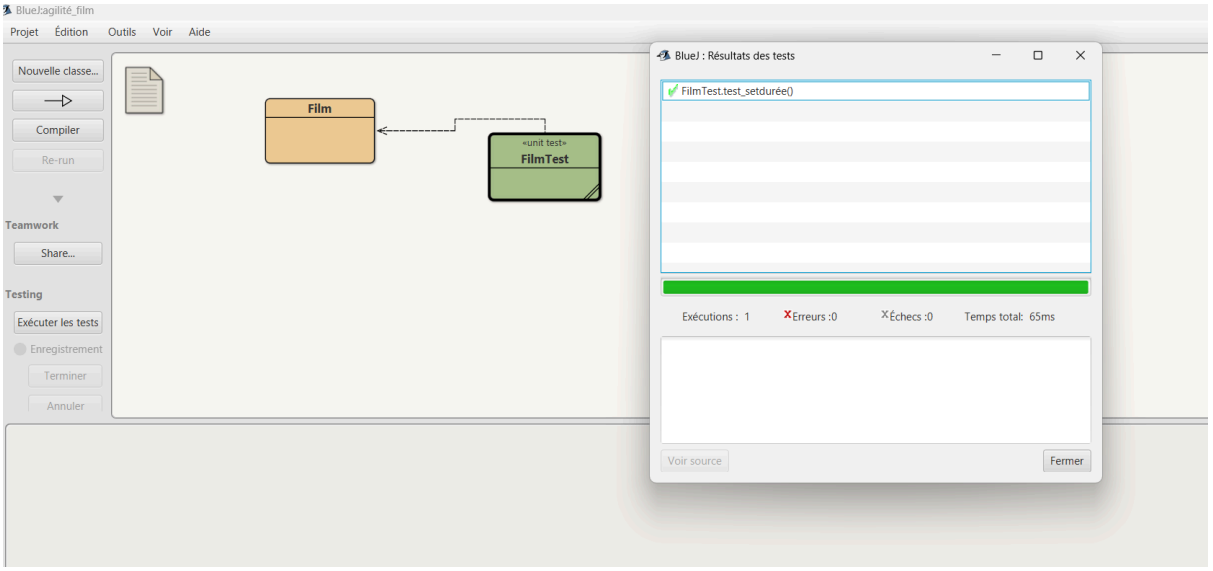


On peut alors terminer l'enregistrement en cliquant tout simplement sur "Terminer".



Maintenant, testons ce test automatique ! Pour le déclencher, il y a plusieurs options mais la plus simple reste d'appuyer sur le bouton à gauche : Exécuter les tests.





Après avoir exécuté notre test, nous pouvons voir son résultat et dans notre cas, on peut voir une barre verte qui signifie que tout est bon et que notre test est bien passé.
Félicitations ! 🎉🏆

Pour aller plus loin : Essayez de créer un test similaire pour l'attribut datesortie ! De cette manière toutes les méthodes de la classe Film seront testées.

NB : Comprendre le code du test automatique
En éditant la classe FilmTest, on voit les méthodes suivantes :

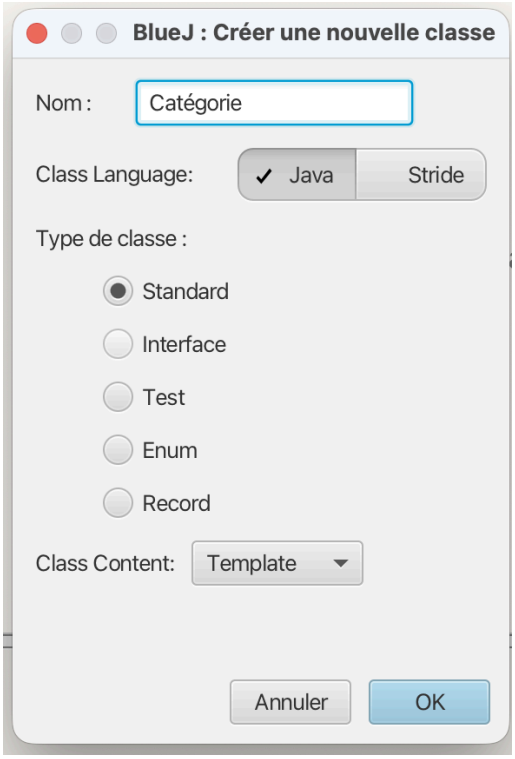
```java

    @Test
    public void testSetDuree()
    {
        killBill.setDuree(111);
        assertEquals(111, killBill.getDuree());
    }

    @Test
    public void testSetDateSortie()
    {
        killBill.setDateSortie("2003/10/29");
        assertEquals("2003/10/29", killBill.getDateSortie());
    }

```

Le test est en fait assez simple : on instancie un film, on utilise le set, puis on teste le get.

Ajoutez une nouvelle classe : Catégorie ➕

Parfois, les attributs peuvent être les mêmes pour plusieurs films et méritent d'avoir leur propre classe. Par exemple, un réalisateur a plusieurs caractéristiques (son âge, ses récompenses…) et plusieurs films différents ont le même réalisateur. On va donc créer une autre classe, une autre "boîte" et l'attribut du film fera référence à un objet de cette boîte, donc à une certaine catégorie.

Commençons avec la classe "Catégorie" pour pouvoir trier les films selon leur genre.



```
1
2  public class Categorie
3  {
4      private String nom;
5
6      public Categorie()
7      {
8          nom = null;
9      }
10
11     public String getNom()
12     {
13         return this.nom;
14     }
15
16     public void setNom(String nom)
17     {
18         this.nom = nom;
19     }
20 }
```

Voici le code qui définit notre catégorie. Notre catégorie aura évidemment un nom et nous avons défini des méthodes pour le récupérer et le modifier.

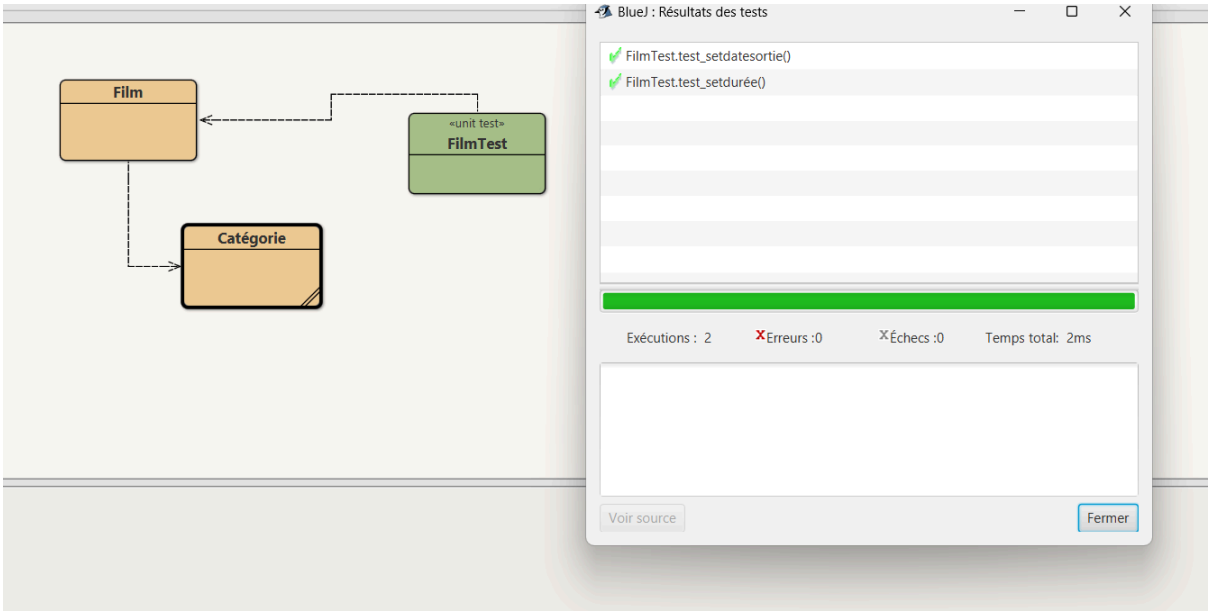
Liaison entre nos deux “boîtes” Film et catégorie : 🔗

```
36     public void setCategorie(Categorie categorie)
37     {
38         this.categorie = categorie;
39     }
40
41     public Categorie getCategorie()
42     {
43         return this.categorie;
44     }
45
46     public String description()
47     {
48         return "Je suis un film de catégorie " + this.categorie.getNom() + " et de durée " + this.duree + " sorti en " + this.dateSortie;
49     }
50
51 }
```

Grâce à ce code, nous allons associer nos deux classes pour qu’elles communiquent entre elles. En effet, nous voulons savoir à quelle catégorie appartient un film par exemple. Pour cela, nous avons écrit des méthodes dans notre classe Film pour récupérer et modifier une catégorie pour un film.
La méthode la plus importante ici est description() : elle permet d’afficher des informations sur le film, en utilisant notamment la méthode getnom() qui est propre à l’attribut Catégorie.

On aimerait maintenant mettre en place un test automatique de cette méthode, comme tout à l’heure.

Vérification de l’état des tests : 🔎

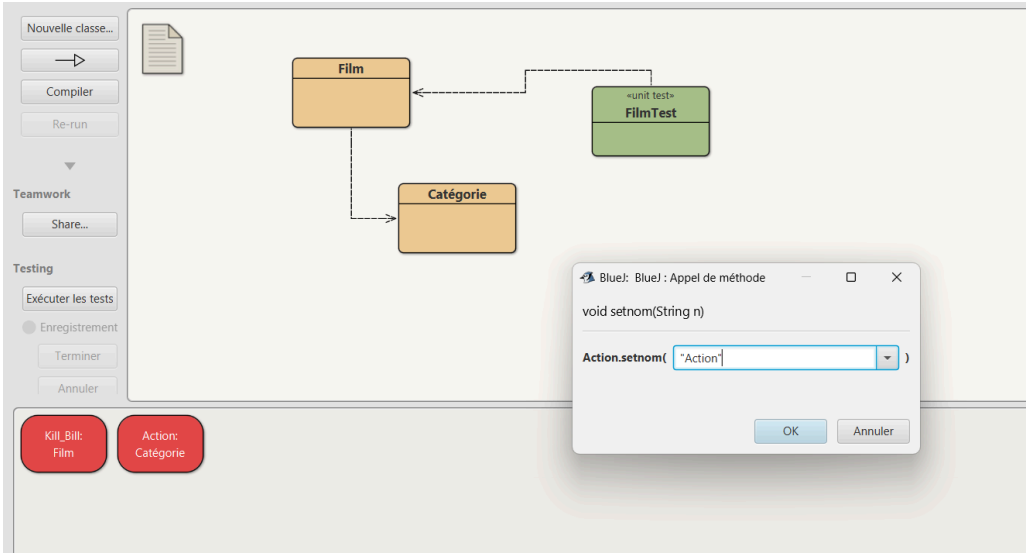


Lorsque l’on effectue des modifications comme la création d’une nouvelle classe, il est important de vérifier que nos anciens tests fonctionnent toujours.

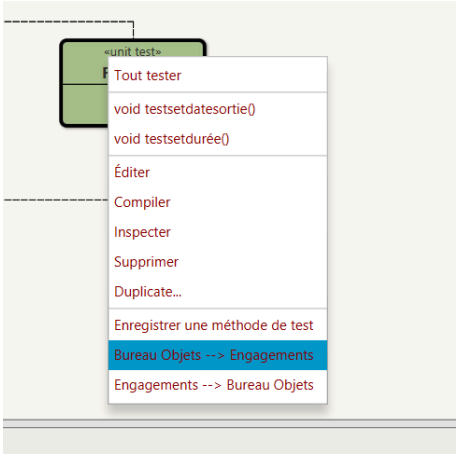
Création d’une “fixture” : 🏗️

Pour faire ce test, nous allons créer une instance de Film et une instance de Catégorie.
Le film est Kill Bill et nous effectuons deux actions avec ces instances : la première c’est de renommer le nom de la catégorie en “Action”, la seconde c’est de modifier dans Kill_Bill l’attribut catégorie pour qu’il pointe vers la catégorie qu’on vient de créer.

Nous avons donc deux objets reliés entre eux. Il faudrait que le test de notre méthode recrée cette situation avant de se lancer.



Nous allons donc utiliser l'option : " Bureau des objets → Engagement". En fait le bureau des objets est la partie basse de l'interface où l'on voit nos objets, alors que les engagements sont la "fixture", la partie "Setup" qui se lance automatiquement au début des tests.

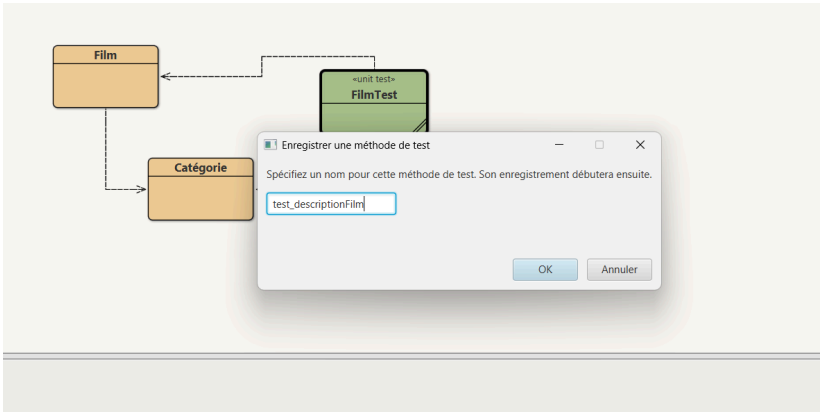


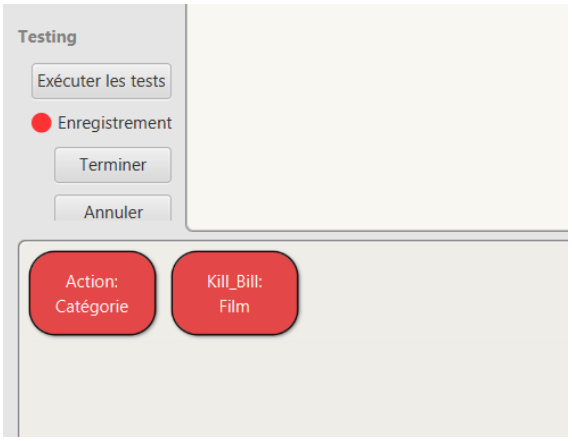
En cliquant ici, le code suivant est automatiquement généré :

```
52      /**
53       * Met en place les engagements.
54       *
55       * Méthode appelée avant chaque appel de méthode de test.
56       */
57      @BeforeEach
58      public void setUp() // throws java.lang.Exception
59      {
60          killBill = new Film();
61          action = new Categorie();
62          action.setNom("Action");
63          killBill.setCategorie(action);
64      }
65
```

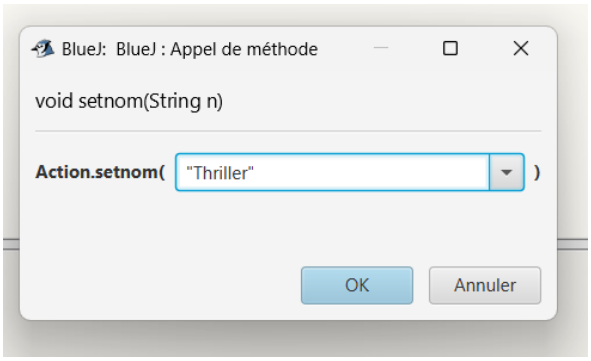
La méthode SetUp permet de coder automatiquement les créations de films et de catégories que nous avons réalisées en quelques clics juste avant, afin de préparer chaque test.

C'est l'heure de l'enregistrement du test ! Comme tout à l'heure, clic droit sur FilmTest et puis enregistrons une nouvelle méthode :



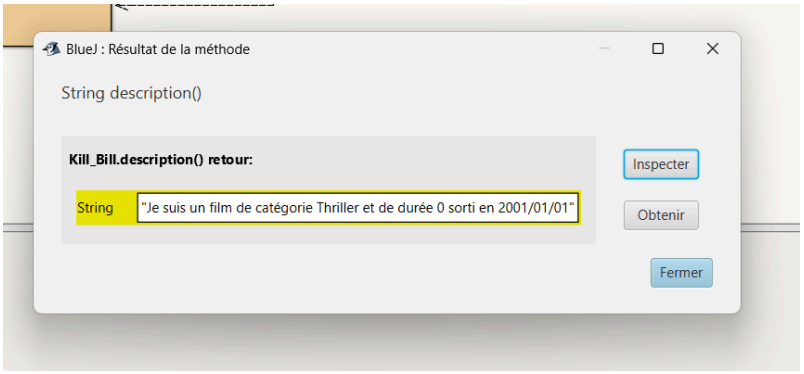


Et la magie s'opère, les instances sont déjà créées ! Afin de répliquer l'environnement de test, l'enregistrement lance déjà le Setup. Il ne reste qu'à tester la méthode description().



Modifions le nom de la catégorie pour vérifier. On passe d'action à thriller.
Ensuite, on va tester la méthode description en précisant le résultat souhaité :
"Je suis un film de catégorie Thriller et de durée 0 sorti en 2001/01/01".

Résultat : 💡

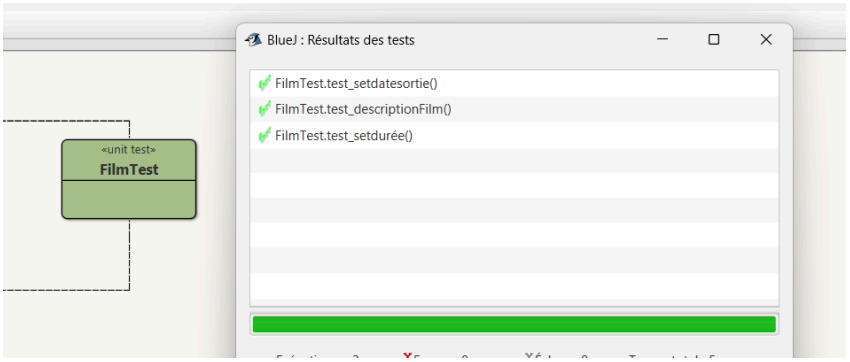


Nous pouvons observer que dans notre description, la valeur de la catégorie associée à notre film a bien été récupérée ainsi que la durée du film et sa date de sortie.
Le test est concluant ! On termine l'enregistrement.

Exécution des nouveaux tests : ▶️

L'enregistrement terminé, on peut voir le code suivant ajouté dans la partie FilmTest :

```
@Test
public void testDescriptionFilm()
{
    action.setNom("Thriller");
    assertEquals("Je suis un film de catégorie Thriller et de durée 0 sorti en 2001/01/01", killBill.description());
}
```



Comme dans la première partie, nous cliquons sur exécution des tests afin de vérifier si les fonctionnalités que nous venons d'ajouter marchent bien. La barre verte nous indique que tout est bon !

```
    */
    @BeforeEach
    public void setUp() // throws java.lang.Exception
    {
        killBill = new Film();
        action = new Categorie();
        action.setNom("Action");
        killBill.setCategorie(action);
    }

    @AfterEach
    public void tearDown() // throws java.lang.Exception
    {
        //Libérez ici les ressources engagées par setUp()
    }

    @Test
    public void testSetDuree()
    {
        killBill.setDuree(111);
        assertEquals(111, killBill.getDuree());
    }

    @Test
    public void testSetDateSortie()
    {
        killBill.setDateSortie("2003/10/29");
        assertEquals("2003/10/29", killBill.getDateSortie());
    }

    @Test
    public void testDescriptionFilm()
    {
        action.setNom("Thriller");
        assertEquals("Je suis un film de catégorie Thriller et de durée 0 sorti en 2001/01/01", killBill.description());
    }
}
```

Maintenant qu'on a ajouté le setup, on a en fait automatisé la création du film Kill_Bill. Ce n'est plus nécessaire d'avoir cette étape au début des deux autres tests. Libre à vous de les supprimer pour optimiser le code !