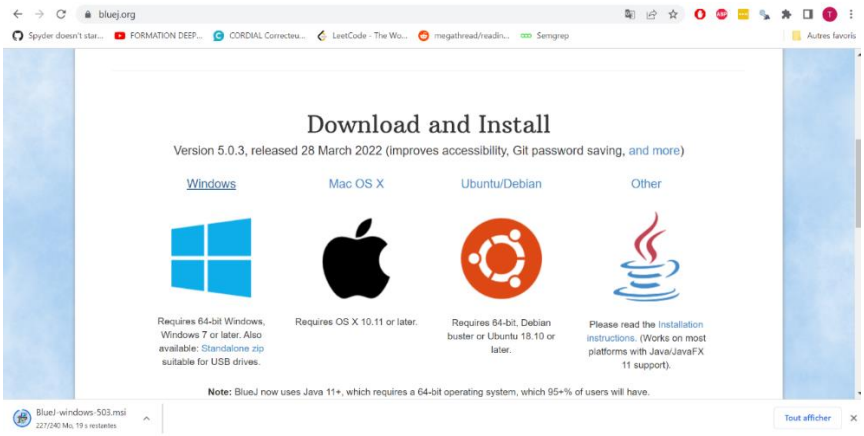
TP1 : Méthodes agile et test unitaire

Thibault Bougerol, Bastien Lesouef

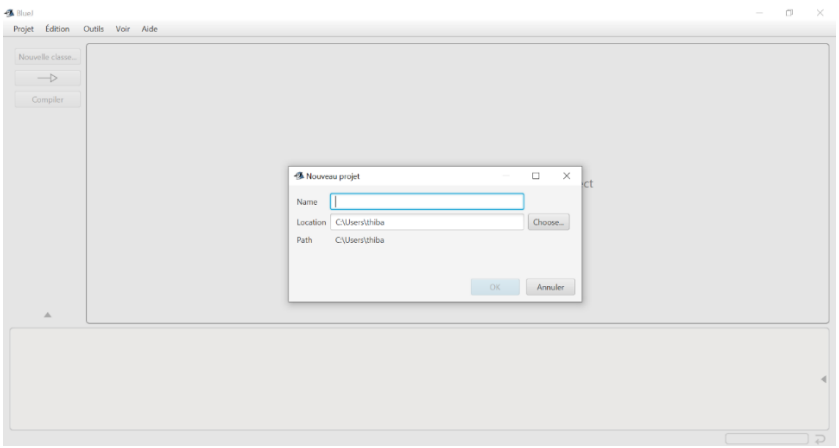
Nous présentons dans ce document sous forme de tutoriel une introduction aux classes à travers l'aventure d'un robot que nous appellerons Bobo. On va utiliser BlueJ qui est un IDE Java qui permet une représentation graphique des classes. Vous pouvez télécharger Bluej en cliquant sur ce lien :

https://www.bluej.org/

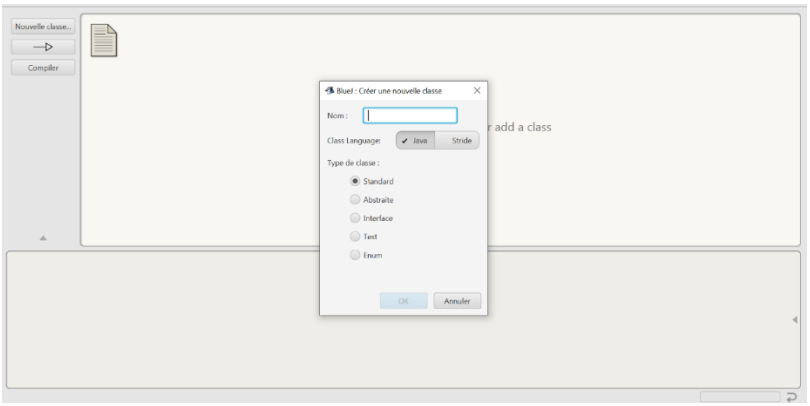
choisissez la version supporter par votre système d'exploitation comme ci-dessous



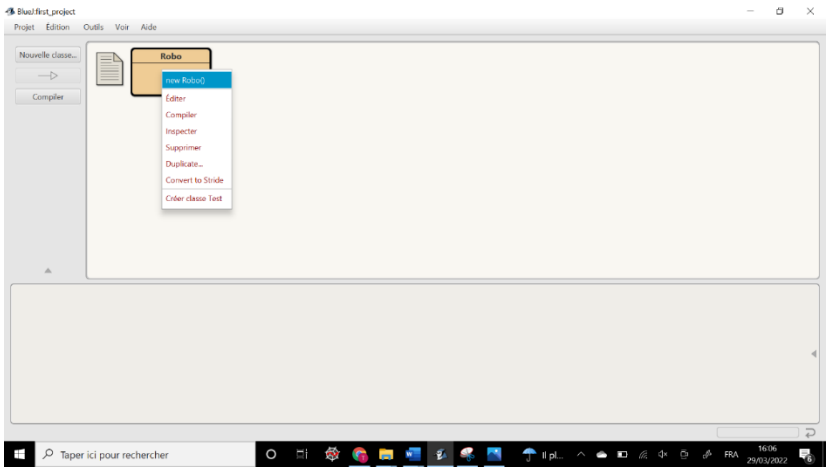
Une fois Bluej installé, cliquez sur l'onglet Projet puis créer un nouveau projet. Une fenêtre s'ouvre et vous demande un nom de projet ainsi que la location du répertoire d'installation.



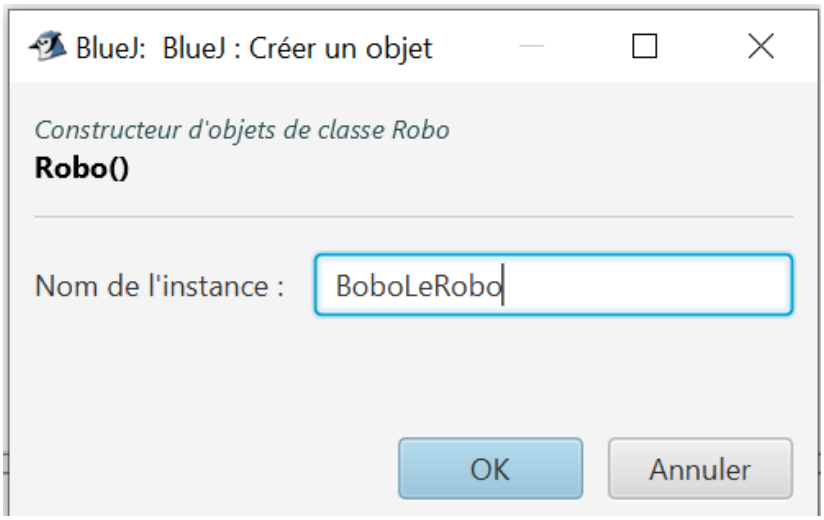
Nous allons créer notre première classe que nous appelons « LeRobo » en cliquant sur l'onglet « Nouvelle classe » en haut à droite comme ci-dessous. BlueJ nous permet de choisir different type de classe comme les classes standard ou bien les interface. Ici nous commençons par créer une classe standard en Java.



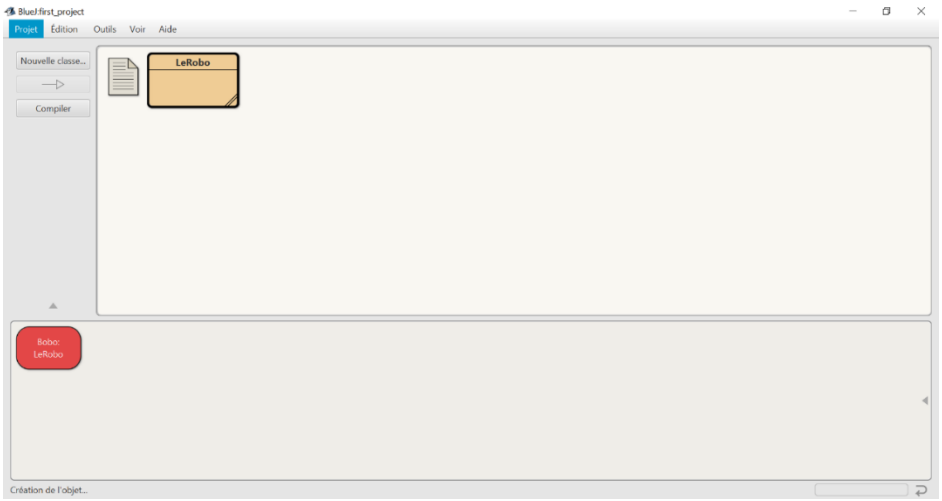
Une fois cette classe créé on peut créer notre première instance de classe en faisant clique droit sur Robo et new Robo().



Voici BoboLeRobo qui est une instance de notre classe Robo. Pour le moment cette classe est vide est ne fait pas grand-chose mais avec un peu d'huile de coude nous allons donner vie à ce robot.



En bas à gauche se trouve la représentation graphique de notre instance de classe.



Pour donner vie à Bobo on peut commencer par ajouter quelques lignes Java dans la classe Robo. On ouvre le contenu de la classe et double cliquant sur la classe LeRobo. Une fenêtre s'ouvre avec le contenu de la classe entre accolade « {} ». On qualifie notre robot à l'aide de trois variables privés qui seront modifiable uniquement avec l'appelle de méthodes. Premièrement nous stockeront sont nom sous forme de string avec la variable « name ». Sont niveau de vie est représenter par son niveau d'huile qui est une variable de type entier qui sera compris entre 0 et 100. Si le niveau d'huile tombe à zero, notre robot seras bon pour la casse… Enfin s'il est vivant ou non (en fonction de son niveau d'huile) est défini par le booléen « is_alive ».

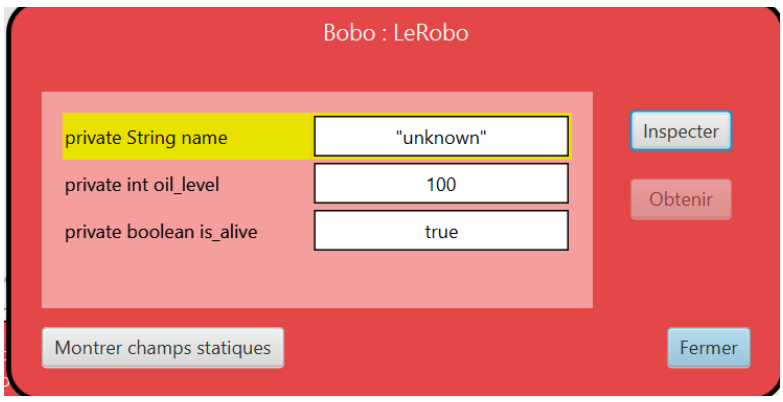
```
public class LeRobo
{
    // variables d'instance - remplacez l'exemple qui suit par le
    private String name;
    private int oil_level;
    private boolean is_alive;

    /**
     * Constructeur d'objets de classe LeRobo
     */
    public LeRobo()
    {
        // initialisation des variables d'instance
        this.oil_level = 100;
        this.name = "unknown";
        this.is_alive = true;
    }
```

En dessous des variables d'instance se trouve le constructeur. Il est identifiable des méthodes de classe car c'est la seule fonction qui porte le même nom que la classe elle-même. Lors de la création d'un objet Robo, le contenu du code se trouvant dans le constructeur est exécuter. Lorsque l'on créer un robot on souhaite deux choses. Premièrement on lui donne un niveau d'huile égale à 100. C'est la taille de son réservoir il ne

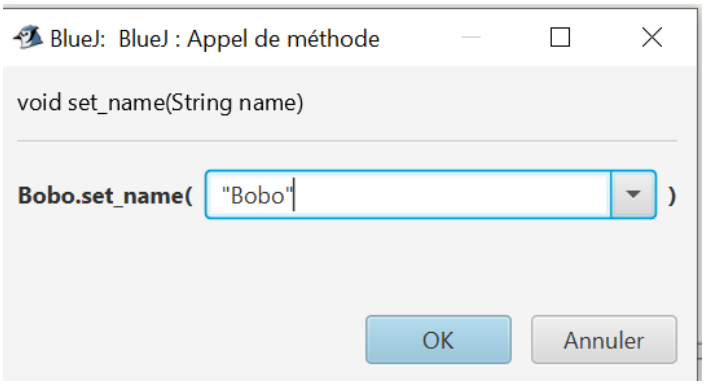
pourras pas en avoir davantage. Puisque qui est fabriqué et que son réservoir est rempli d'huile il devient vivant, nous initialisons la variable is_alive comme vrais. Pour le moment on ne peut avoir aucune interaction avec notre robot. Notre robot est vivant et en pleine santé mais malheureusement il ne peut pas encore se déplacer… Afin d'interagir avec lui on utilise ce que l'on appelle des méthodes de classe. On commence par créer trois méthodes. Une lui permet de changer son nom (il en sera ravi) et deux autres qui interagirons directement sur son niveau d'huile. Ces trois méthodes ne retournent rien mais permettent de changer les instances de notre classe. Add_oil prend comme valeur oil qui est un entier et incrémente le niveau d'huile de notre robo seulement si l'ajout de cette quantité ne dépasse pas la limite maximale de 100 du réservoir.

```
    public void add_oil(int oil)
    {
        // Oil level cannot exceed 100
        if (this.oil_level + oil > 100) {
            this.oil_level = 100;
        }else {
             this.oil_level += oil;
        }
    }

    public void set_name(String name){
        this.name = name;
    }

    public void set_oil_level(int oil){
        if (oil > 0){
            this.oil_level = oil;
            this.is_alive = true;
        }else if(oil == 0){
            this.oil_level = oil;
            this.is_alive = false;
        }
    }
}
```

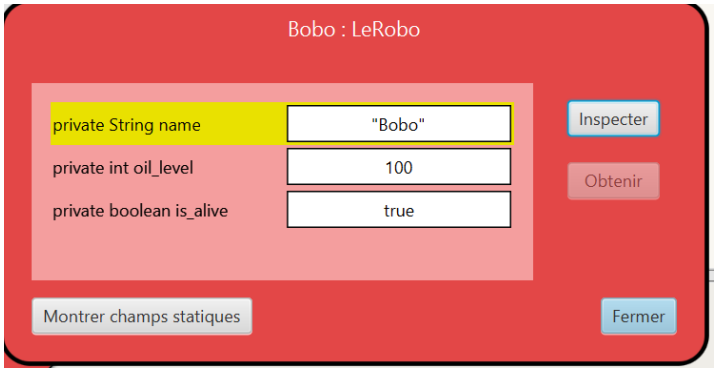
Afin de visualiser l'utilité des méthodes on commence par créer un nouveau robot. En double cliquant sur l'instance en bas à droite une fenêtre rouge s'ouvre et nous montre le contenu des variables d'instances. Pour le moment on remarque que notre robot est bien vivant, pleins d'huile mais malheureusement il ne possède pas encore de nom ni de personnalité…



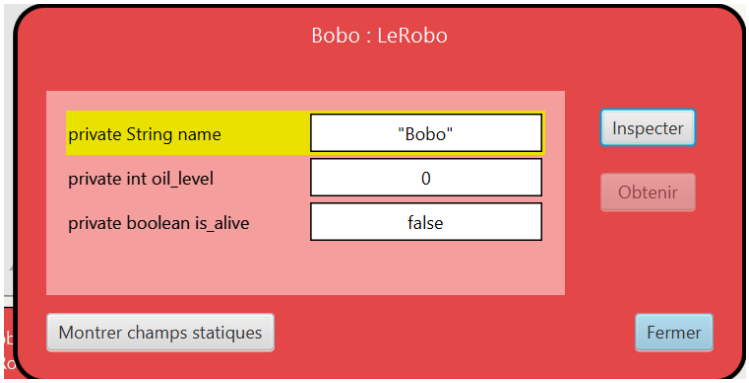
On commence par le renommé Bobo (et il nous en remercie).



En ouvrant une nouvelle fois l'explorateur de classe on voit que la variable « name » a bien changé en Bobo.



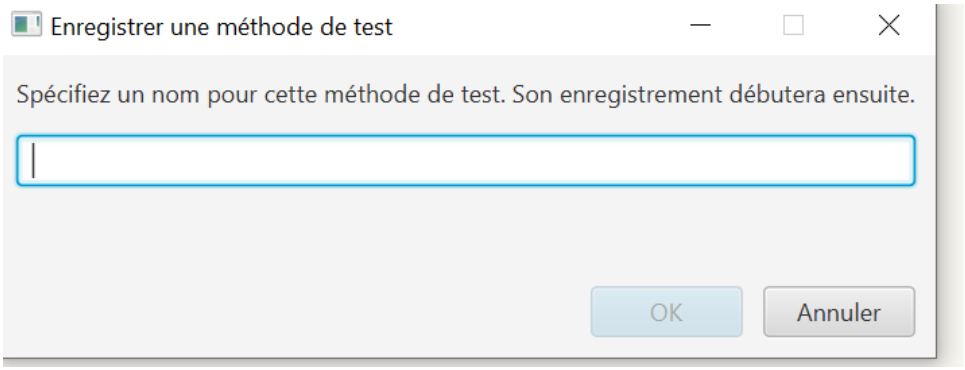
Toujours avec un appelle de méthod on change le niveau d'huile de Bobo à zero. Apres recompilation on voit que bobo n'a malheuresement pas survecu à cette vidange.



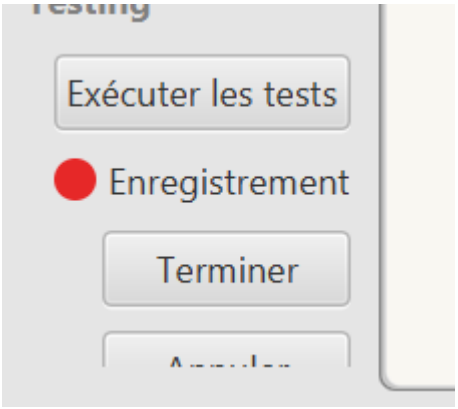
Jusqu'à présent nous avons développé en testant partiellement notre code mais nous ne somme pas à l'abri d'erreur. Plutôt que d'appeler manuellement chacune des méthodes de classes en faisant varier les paramètres en en vérifiant les inputs nous allons automatiser le processus. On créer une nouvelle classe Java mais cette fois ce ne seras pas une classe standard mais une classe Test.

En faisant un clic droit sur la classe test, on peut enregistrer une méthode de test :

On donne un nom à cette méthode et on l'enregistre en faisant les actions que l'on veut :



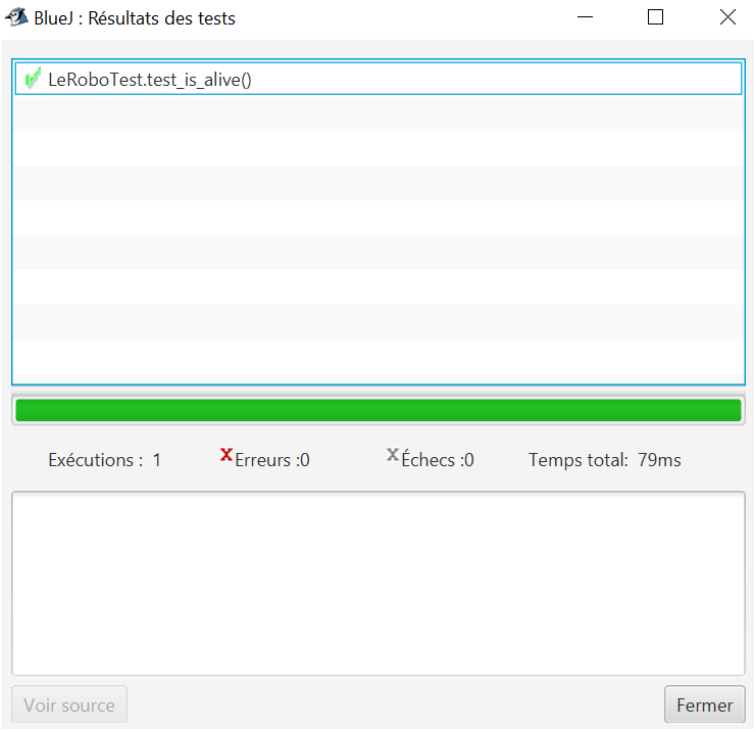
Il est facile de vérifier que l'on est bien en train d'enregistrer :



Une fois l'enregistrement terminé, on peut rajouter le assertEquals que l'on veut à la fin du test ou le faire directement en appelant la méthode et en mettant le résultat attendu.

```
@Test
public void test_is_alive()
{
    LeRobo Bobo = new LeRobo();
    Bobo.set_oil_level(0);
    assertEquals(Bobo.is_alive(), false);
}
```

Et lancer le test :

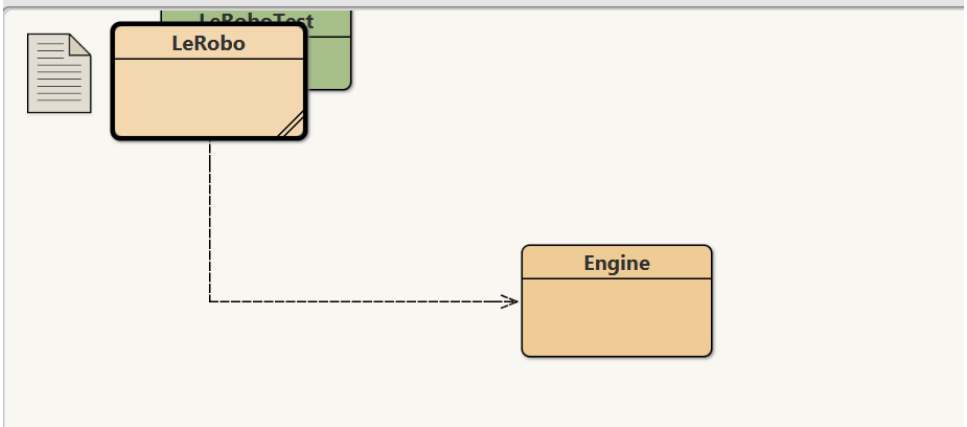


On créer une classe Engine qui représente un moteur et on rajoute un engine dans notre classe LeRobo. Maintenant, nous aurons un état “ready_to_walk” qui dépendra du moteur et du robot

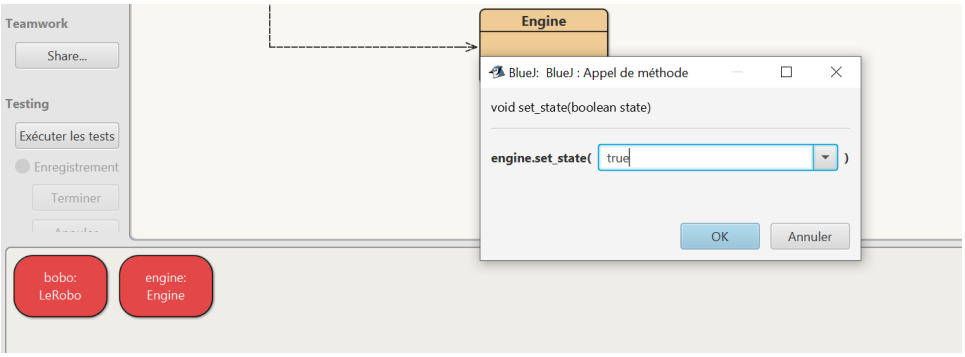
```
 */
public class LeRobo
{
    // variables d'instance - remplacez l'exemple qui suit par le vôt
    private String name;
    private int oil_level;
    private boolean is_alive;
    private Engine engine;
```

```
    public boolean ready_to_walk(){
        if (this.is_alive == true || this.engine.get_state() == true){
        return true;
        } else {
            return false;
        }
    }
}
```

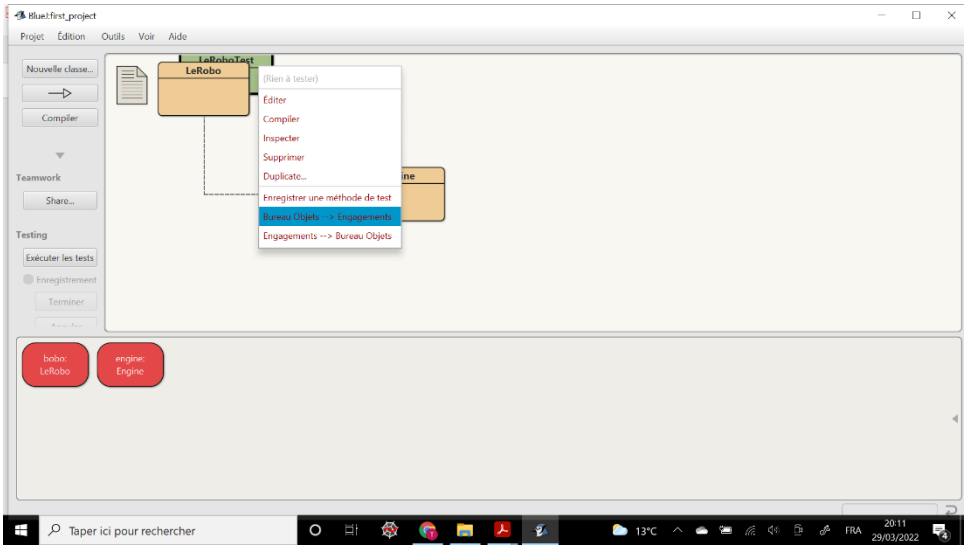
Cette flèche nous indique que l'association est bien faite et unidirectionnelle :



Pour finir, on créé un robot et un engine et on met le state de l'engine sur true :



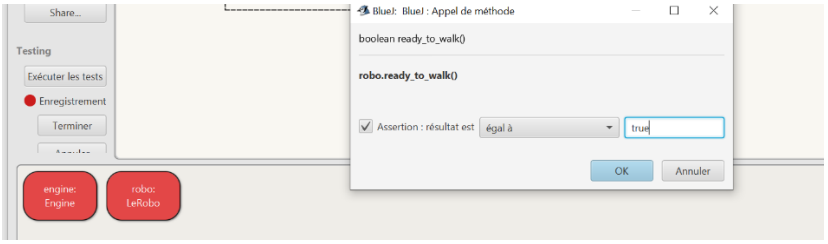
Puis nous créons notre fixture :



Cela créé le setup de notre classe test pour que l'on n'ait pas à le refaire à chaque fois si l'on veut tester sur ces instantes en particulier :

```
 */
@BeforeEach
public void setUp() // throws java.lang.Exception
{
    robo = new LeRobo();
    engine = new Engine();
    engine.set_state(true);
    robo.get_engine(engine);
}
```

Ici, ce que l'on fait, c'est que l'on créé un robo et un engine. Ensuite, on met le state de l'engine sur true et on l'associe à notre robot. Enfin, on va lancer un dernier test qui vérifie que le robot est capable de marcher (car il est vivant et que son moteur fonctionne) :



Le test se lance correctement :