TP N°1 – Agilité – Les tests unitaires

HADI Ismaïl & JOLIMAN Iris

Première partie : BlueJ

La programmation orientée objet est un outil très répandu aujourd'hui dans le développement logiciel. Elle permet de représenter des problématiques très variées à travers des **objets** et des **classes**.

Pour comprendre son fonctionnement, nous avons décidé de baser ce tutoriel pour illustrer la vie des NACs (Nouveaux Animaux de Compagnie).

Avant de commencer à étudier la partie technique, une phase d'analyse est ESSENTIELLE en programmation orientée objet. De plus, la schématisation est aussi importante. En ce sens, l'outil BlueJ, que nous allons utiliser au sein de ce tutoriel, permettra d'allier la schématisation et l'implémentation.

Installation de BlueJ

Comme indiqué précédemment, installer BlueJ est un prérequis pour ce tutoriel car car c'est l'outil qui nous permettra de visualiser et manipuler nos objets et nos classes.

Vous pourrez le télécharger et l'installer en cliquant sur le lien suivant : http://www.bluej.org/

Création d'un nouveau projet

En ouvrant BlueJ, il est nécessaire de créer un nouveau projet. Pour cela on réalise les actions suivantes : Projet > Nouveau projet…

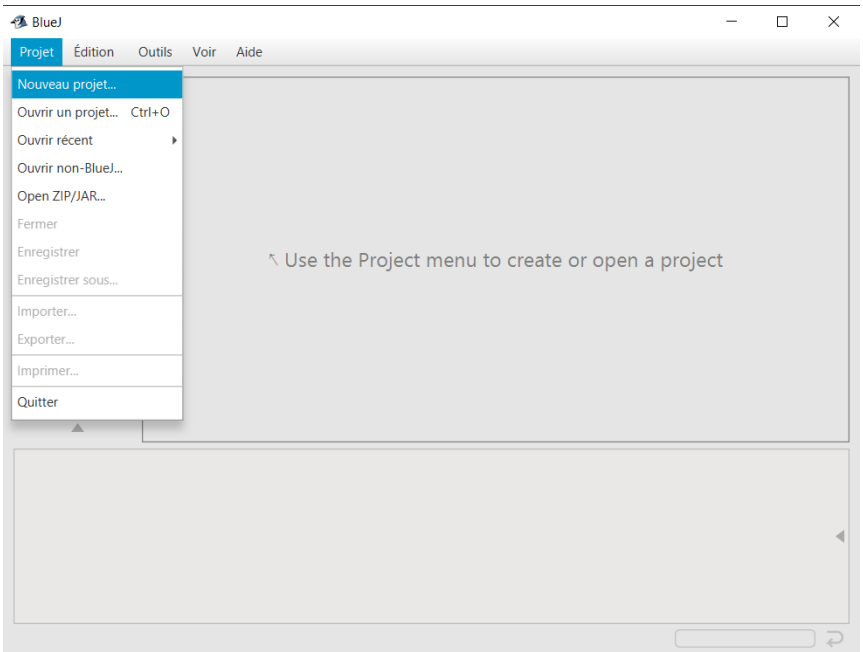


Figure 1: Création d'un nouveau projet dans BlueJ

Par la suite, vous pouvez choisir le nom de votre choix ainsi que le répertoire dans lequel vous voulez retrouver le projet.

Nous voulions garder le suspens pour l'instant concernant le thème de notre projet… Veuillez donc ne pas porter trop d'attention au nom choisi 😊.

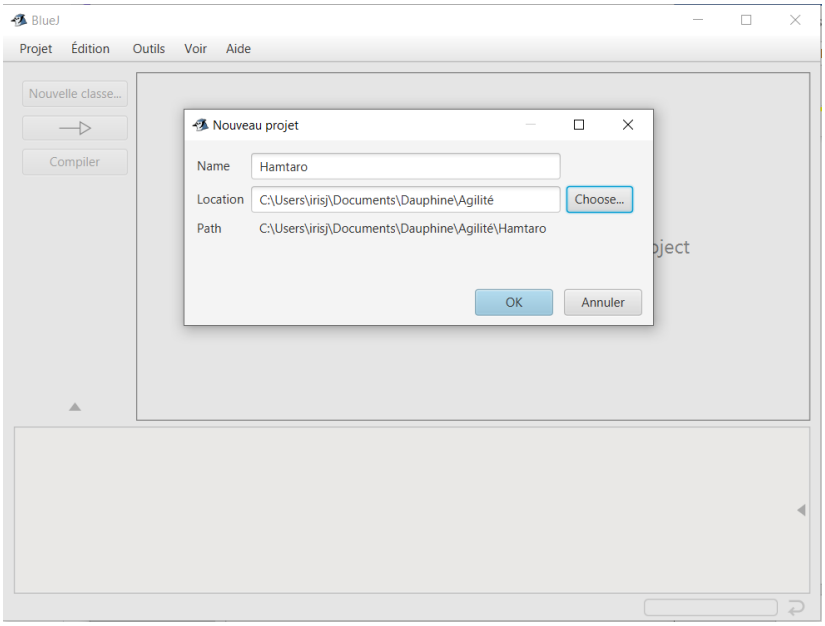


Figure 2: Nommer le projet

A l'issue de la création du projet, vous devriez obtenir une fenêtre similaire à celle ci-dessous avec le nom de votre projet en haut à gauche.

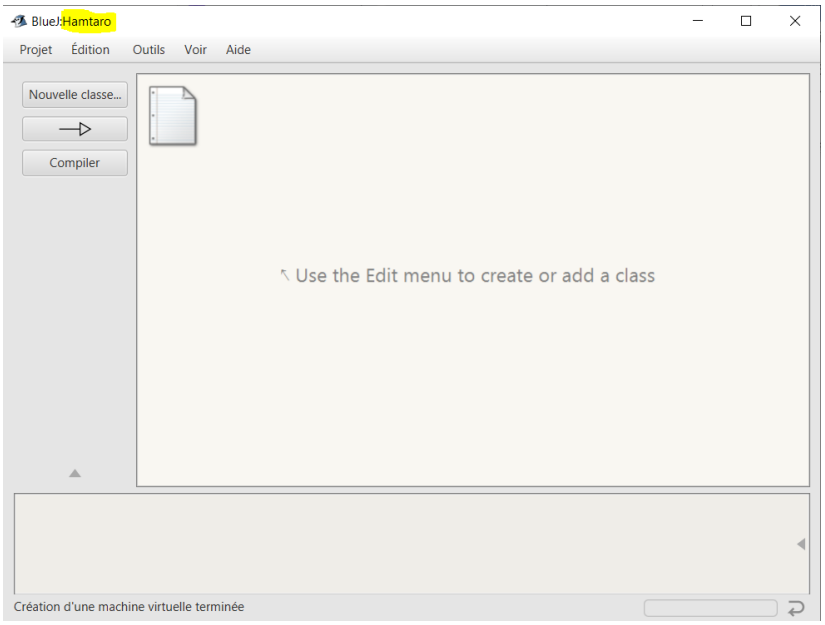


Figure 3: Fenêtre après la création de projet

Pour revenir à notre modélisation, quel est donc notre besoin ?

C'est l'heure de mettre fin au suspens qui vous tiraillait surement. Nous allons représenter la vie et les aventures d'un petit hamster très célèbre pour tous les enfants des années 2000 (ça commence à faire vieux…) : Hamtaro.

Pour ceux qui ne connaissent pas, voici pour la culture : https://www.youtube.com/watch?v=sWti4NjPTYo

Pour adopter cet animal de compagnie qui en fait craquer plus d'un, il faut pouvoir trouver un moyen de modéliser son espèce dans notre projet. En programmation orientée objet, cela se matérialise par la création d'une classe (description d'un type) que nous appellerons NAC.

Hamtaro sors tous les jours lorsque son maître est au travail ou à l'école pour retrouver tous ses amis Hamster dans le club des Ham-Hams où on retrouve Amiral, Bijou et Chapo par exemple. Tous ces petits hamsters y compris notre favori Hamtaro seront des instances de la classe NAC (donc des objets en programmation orientée objet). Nous pourrions également créer la pire angoisse d'Hamtaro Serpentar le serpent qui serait également une instance de la classe NAC.

Création de la classe « fétiche » : NAC

Pour créer notre nouvelle classe NAC, nous devons cliquer sur Nouvelle classe… et la fenêtre suivante s'affiche. Pour le moment, nous nous intéressons uniquement au type standard.

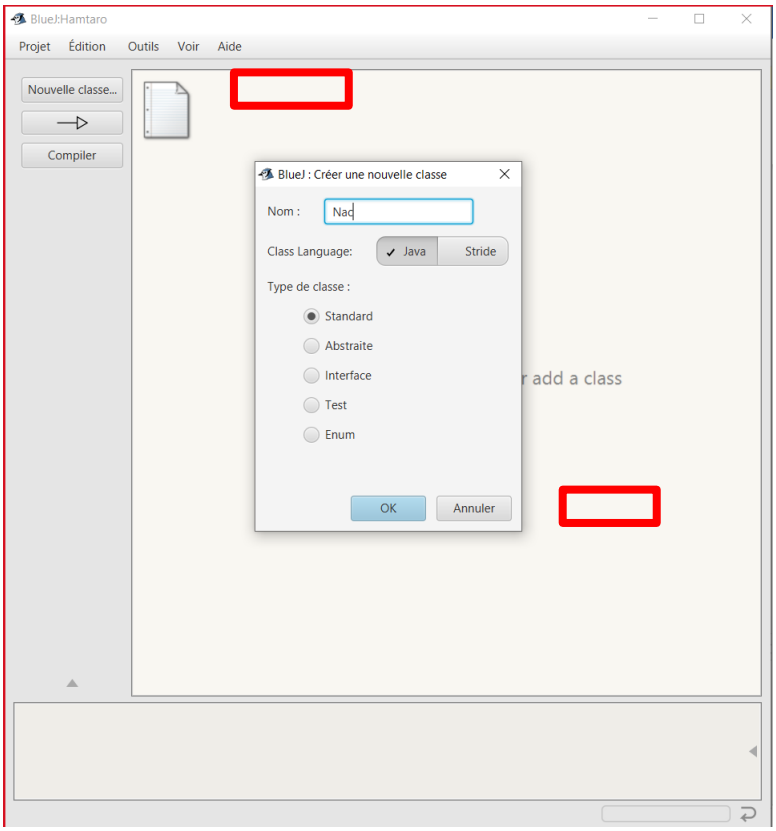


Figure 4: Définition de notre nouvelle classe (type standard)

La compilation consiste à vérifier la cohérence du code présent au sein de la classe compilée. C'est au niveau de cette opération que les erreurs sont détectées.

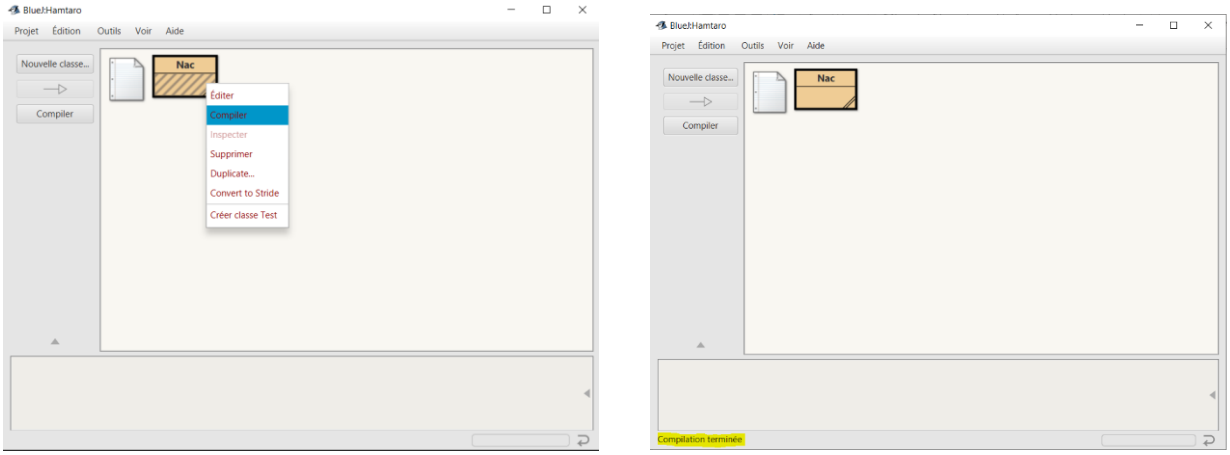
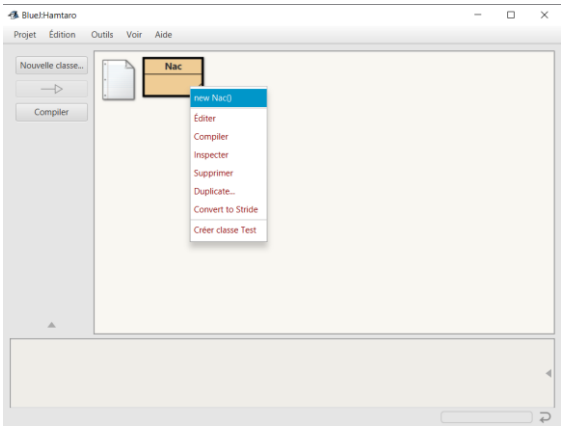


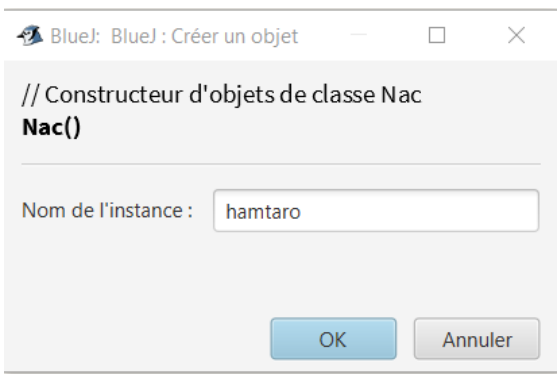
Figure 5: Classe NAC avant et après compilation

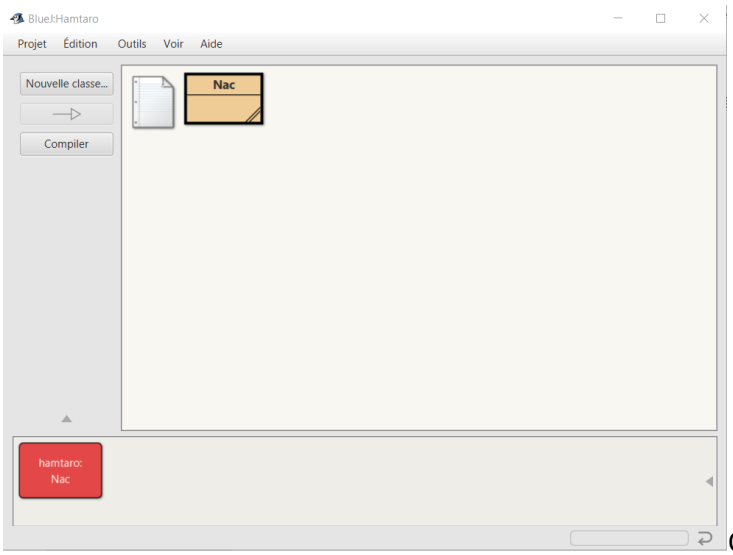
Avant compilation, on remarque que la classe est grisée. C'est cette modélisation sur BlueJ qui indique à l'utilisateur qu'une compilation est nécessaire pour pourvoir utiliser la classe (avant ça, on bloque la naissance de notre Hamtaro préféré … quel dommage).
Visuellement, vous pouvez constater que le quadrillage gris a disparu après la compilation. Voilà une bonne chose de faite.

Tu vas donner vie à un Hamster ! (c'est magique)

Maintenant que notre classe est prête à l'emploi, nous allons pouvoir créer autant d'animaux de compagnies (bon pas un zoo quand même) que voulus: commençons tout d'abord par notre hamster Hamtaro !
Pour cela, il suffit d'effectuer un clic droit sur la classe Nac (après compilation) et de sélectionner le champ New Nac(). Une pop-up apparait ensuite nous permettant de donner le nom souhaité à notre nouveau compagnon.
En tant que futur as de la programmation orientée objet, il est à noter que le nom d'une classe commence par une MAJUSCULE et celui d'une instance par une minuscule. Dans notre cas, hamtaro est une instance de la classe Nac (tant pis pour le français de toute façon on se rappelle plus de rien depuis le CP).







On peut voir ici qu'Hamtaro a bien été créé (dans la partie rouge)

Cependant, hamtaro aimerait bien posséder un nom. Et même si il aime faire la Java (haha), il a besoin de beaucoup d'heure de sommeil. Ce sont des attributs que nous souhaiterions lui donner.

Pour se faire, nous ajoutons deux attributs dans la classe (nom et nbHeureSommeil). Afin de rendre le code robuste, nous créons des accesseurs (méthodes get et set) afin de récupérer ou modifier ces valeurs. Voici le code implémentant ces modifications :

```
public class Nac
{
    // variables d'instance - remplacez l'exemple qui suit par le vôtre
    private String nom;
    private int nbrHeureSommeil;

    /**
     * Constructeur d'objets de classe Nac
     */
    public Nac(String nom_, int nbrHeureSommeil_)
    {
        // initialisation des variables d'instance
        this.nom = nom_;
        this.nbrHeureSommeil = nbrHeureSommeil_;
    }

    public String getNom()
    {
        return(this.nom);
    }

    public void setNom(String nom_)
    {
        this.nom = nom_;
    }
```

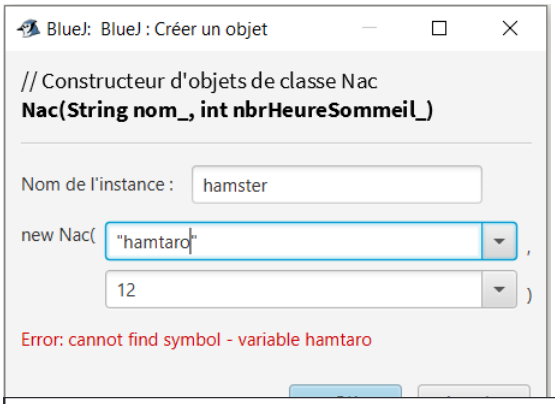
```
    public void setNom(String nom_)
    {
        this.nom = nom_;
    }

    public int getNbrDodo()
    {
        return(this.nbrHeureSommeil);
    }

    public void setNbrDodo(int nbrHeureSommeil_)
    {
        this.nbrHeureSommeil = nbrHeureSommeil_;
    }

    public String presentation()
    {
        return "Je m'appelle " + this.nom + " et je dors " + this.nbrHeureSommeil + "heures.";
    }
}
```

Par ailleurs, une méthode nouvellement créée permet désormais de présenter votre animal ! Et oui en programmation orientée objet on peut tout faire … Même faire parler les animaux.
Pour effectuer ces différentes méthodes, il suffit de faire un clic droit sur l'instance de la classe que vous devrez créer à nouveau grâce au constructeur (en bas à gauche en rouge) et le tour est joué ! Vous n'avez qu'à entrer les valeurs souhaitées et vous obtiendrez le résultat escompté.



Nous avons par ailleurs modifié le constructeur de la classe (nous ne sommes pas Dieu mais pouvons tout de même mettre au monde nos animaux). Ainsi, notre animal peut être baptisé avant de voir le jour.

Figure 6: Création d'un nouvel hamtaro avec de nouveaux attributs

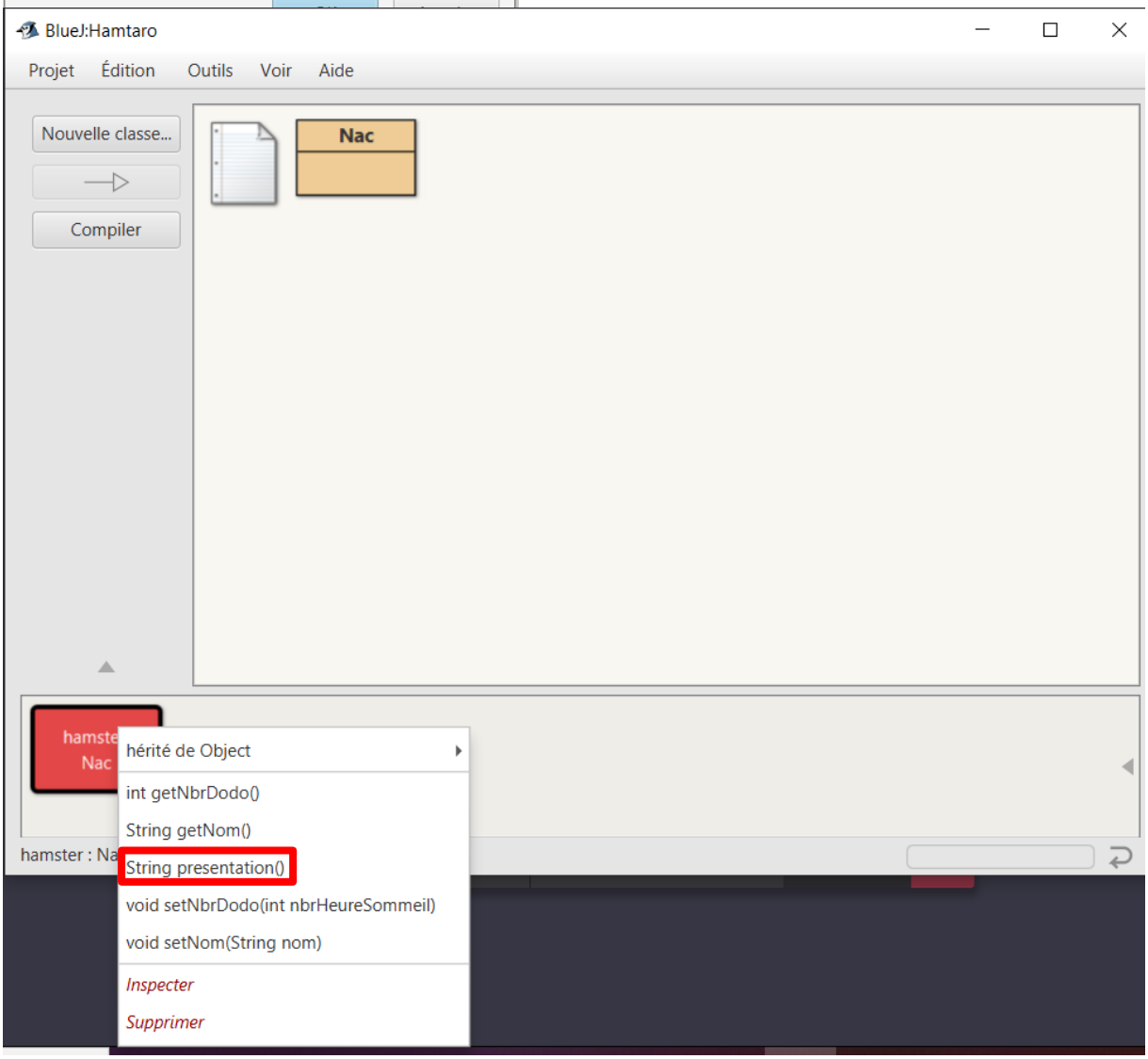


Figure 7: Lancement de la fonction presentation()

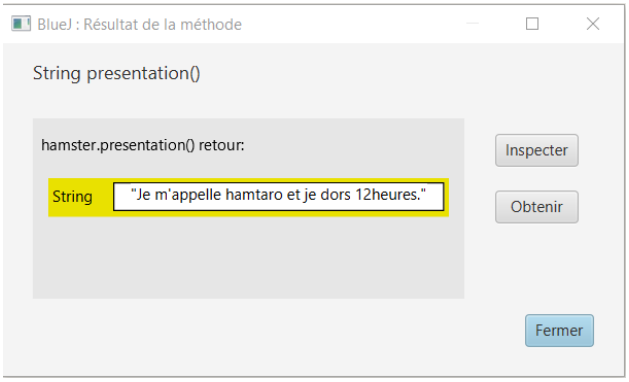


Figure 9: Retour de la fonction presentation

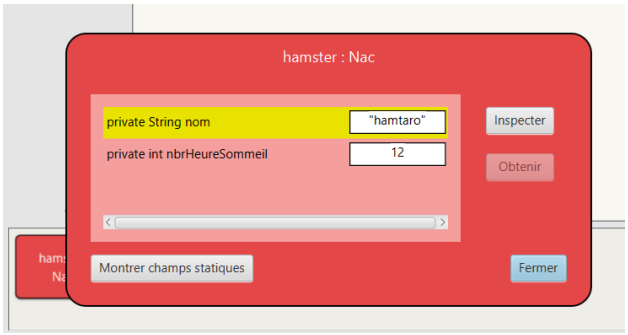


Figure 8: Vérifications en cliquant sur "Inspecter"

Hamtaro se présente-t-il vraiment bien ?

Même si nous pouvons faire énormément de choses en programmation orientée objet, la règle primordiale est de tester notre code. On ne voudrait quand même pas qu'Hamtaro se retrouve avec une queue de poisson…

Pour tester les nouvelles fonctionnalités de notre œuvre, il est primordial de créer une nouvelle classe, que l'on nommer NacTest. Pour cela, il suffit d'effectuer un clic droit sur la classe Nac et de sélectionner "Créer classe Test". Une classe verte, nommée "NacTest" apparaît alors derrière la première.

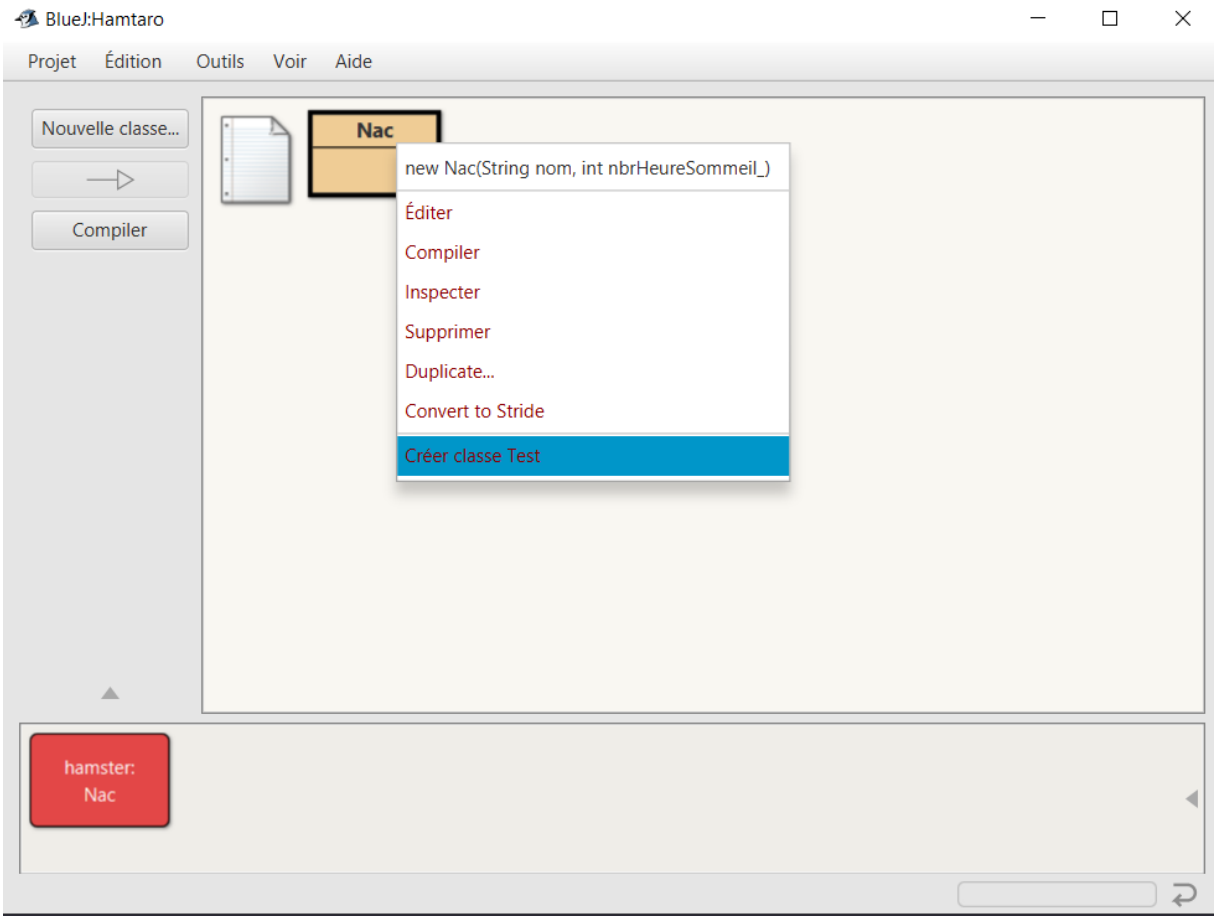


Figure 10: Création de NacTest

Vous l'aurez compris, coder en Java demande avant tout de la patience ! Pour tester une méthode créée (nous prendrons pour exemple la méthode présentation), un clic droit sur la classe NacTest vous permettra d'effectuer l'action "Enregistrer une méthode de test"

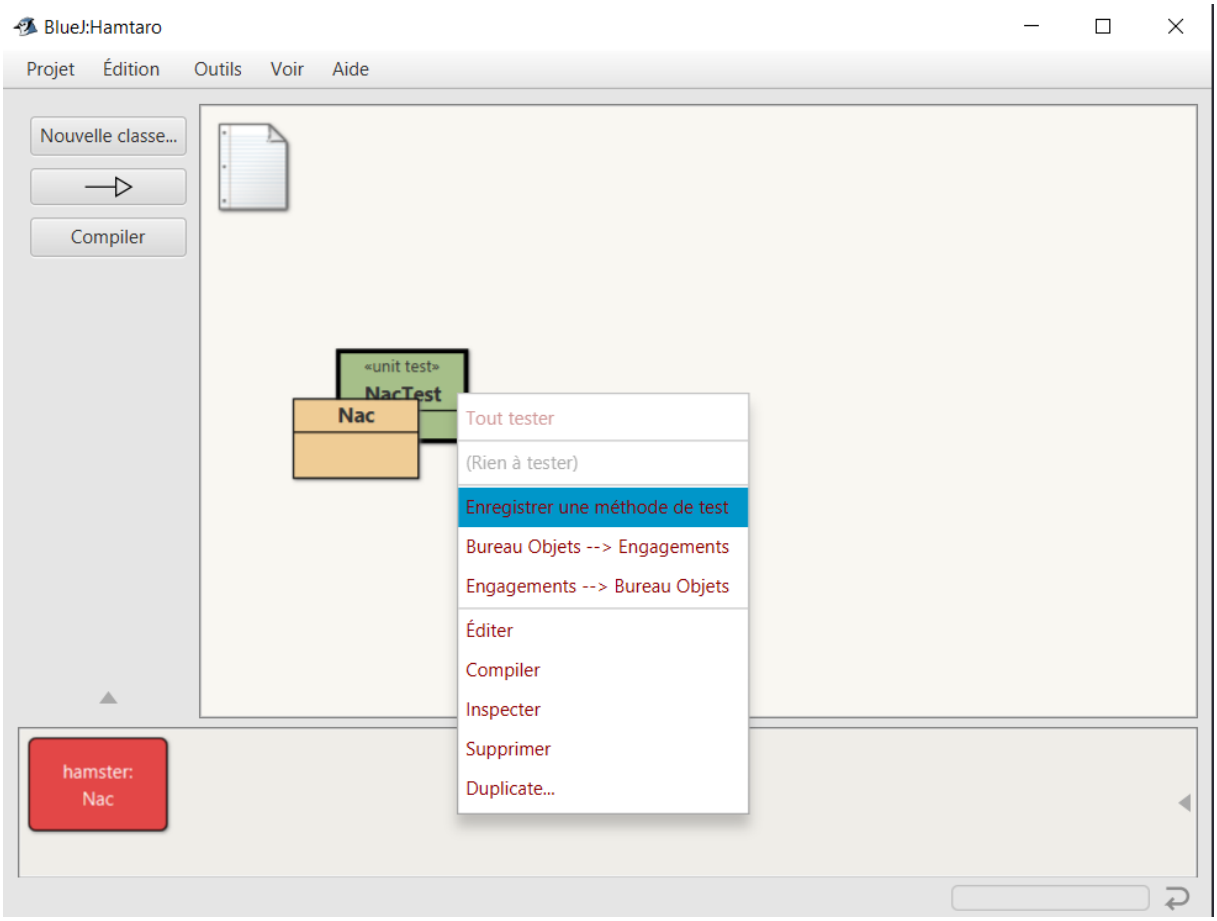


Figure 11: Enregistrer une méthode de test

Dès lors, une pop-up s'affiche vous demandant le nom de la méthode de test. Il est de coutume d'un codeur qui se respecte (oui toi lisant ce tutoriel) de nommer la méthode comme suit: nom de la méthode testée (ici présentation) + Test à la fin ce qui nous donne (fuusiioon) presentationTest (eh oui tout le monde n'en est pas capable soyez fiers !)

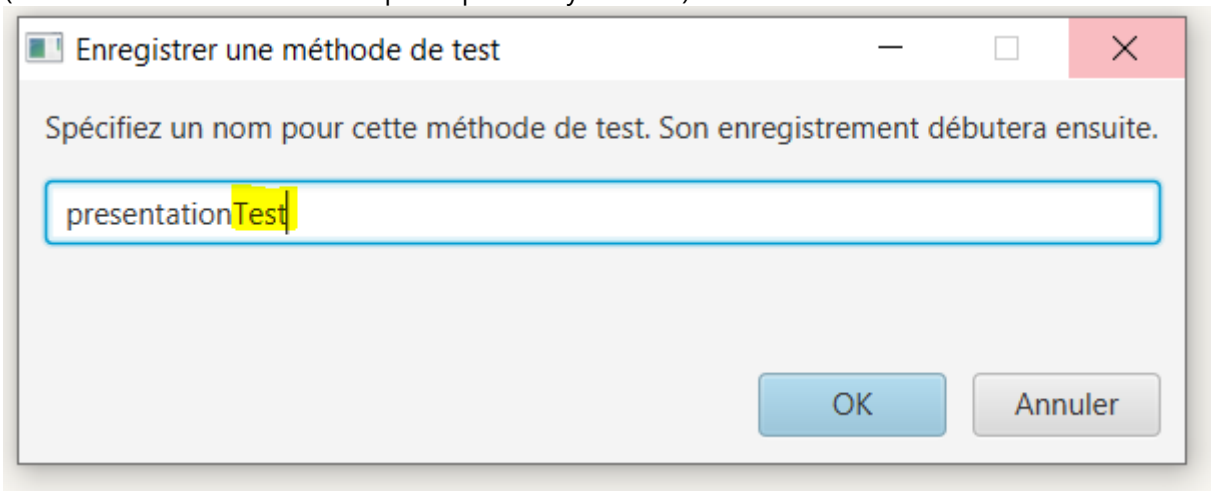
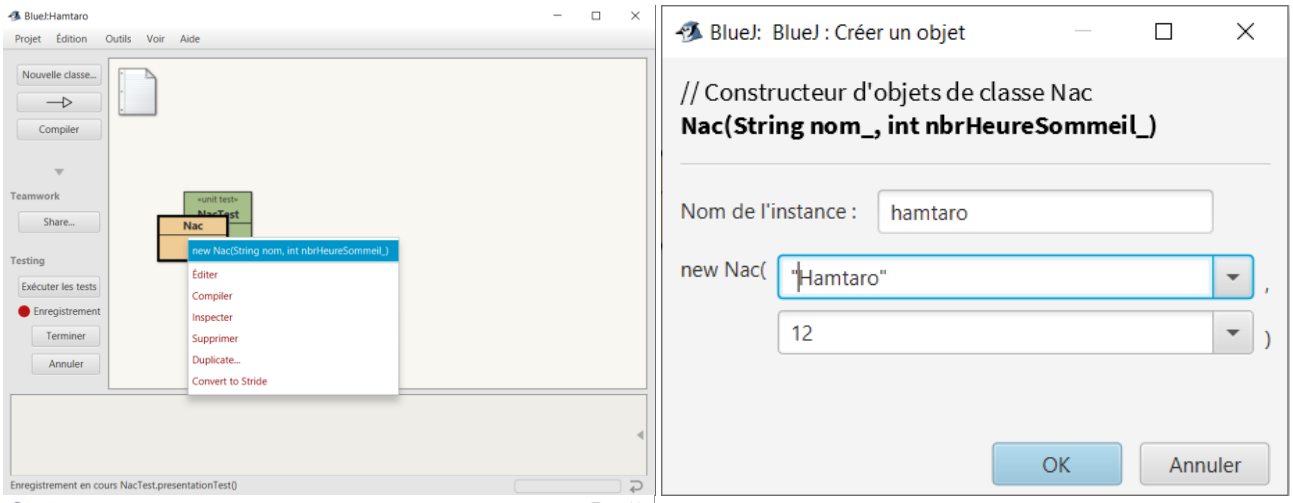
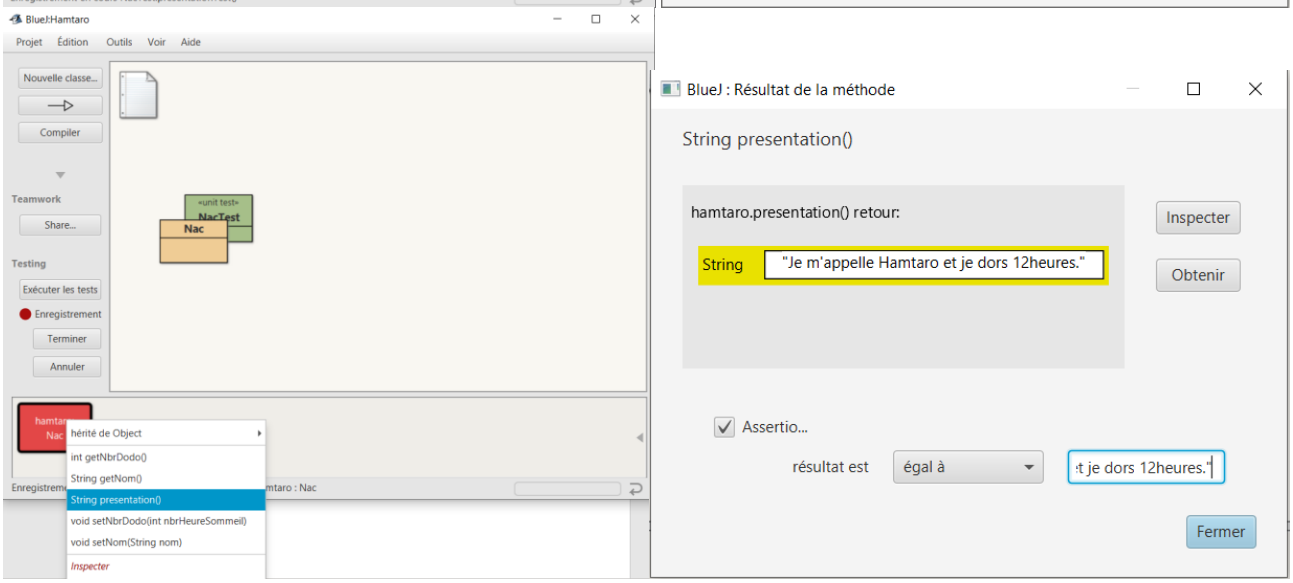
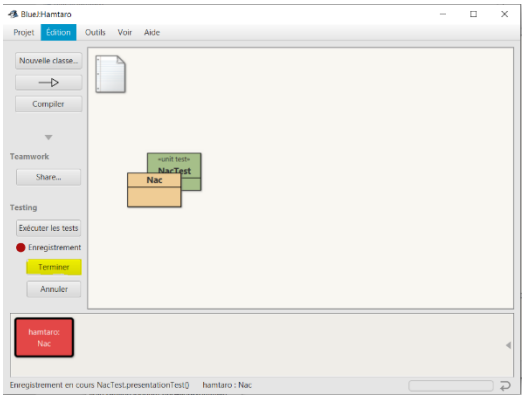


Figure 12: PopUp intergalactique de fusion (bon ok.. c'est juste un nom)

Vous constaterez dès la validation qu'un bouton rouge "enregistrement" apparaît sur la gauche de votre écran et matérialise le début des tests. Pour tester notre fonction présentation, il suffit de créer un nouvel objet puis de cliquer (Hé oui toujours le droit) sur cet objet créé en rouge et enfin de sélectionner la méthode présentation.
Un petit effort de réflexion vous est demandé (Ohhhh non…). En effet, c'est le moment d'être aussi intelligent que votre programme et d'imaginer le résultat attendu. Celui-ci sera à entrer au niveau du champ "résultat est" et permettra à l'outil de comparer le résultat attendu et celui renvoyé par la méthode. Une fois la fenêtre fermée, vous pouvez arrêter l'enregistrement en cliquant sur le bouton "terminer" à gauche de votre écran.







Si vous êtes curieux, cliquez double sur la nouvelle classe NacTest et vous devriez voir le code suivant apparaître ! (Waaaaaaaaaaa)

```
@Test
public void presentationTest()
{
    Nac hamtaro = new Nac("Hamtaro", 12);
    assertEquals("Je m'appelle Hamtaro et je dors 12heures.", hamtaro.presentation());
}
```

Pour afficher le résultat de la comparaison (promis c'est la dernière étape), cliquez sur le bouton "exécuter les tests". Tadaaaa, si le programme fonctionne (et si non tant pis je ne recommencerai pas toutes ces étapes), un checkpoint vert apparaît à côté de la méthode vous signifiant le bon fonctionnement. Vous rentrez maintenant dans la cour des grands.

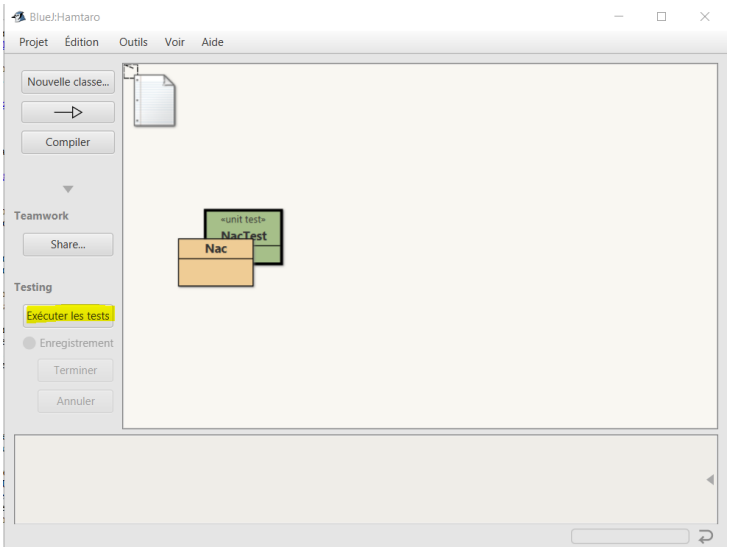


Figure 13: Exécution des tests

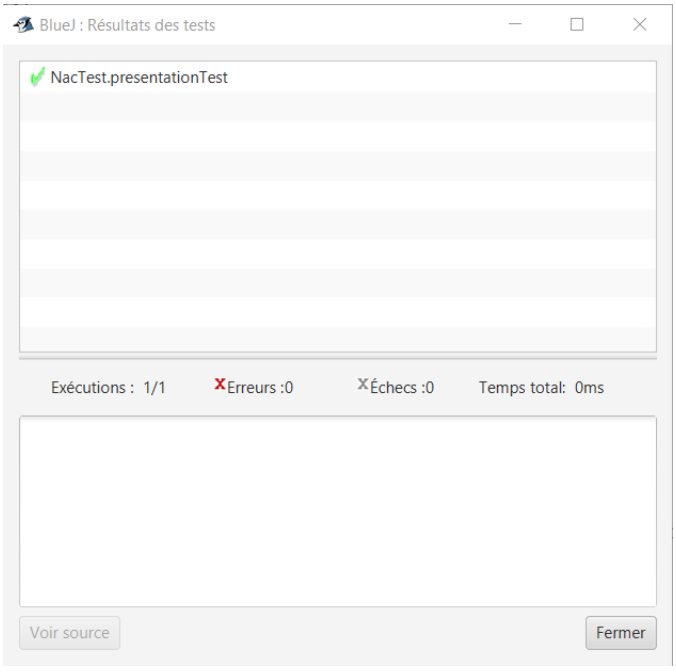


Figure 14: La barre est verte pas de rouge à l'horizon tout est bon

A présent, permettons à Hamtaro de connaître son maître (eh oui uniquement pour pimenter les choses). Pour ce faire, nous créons une classe "Maître" caractérisée par un nom, un âge ainsi qu'un animal. Pour pallier à la solitude du maître au travail ou à l'école, nous lui proposons de pouvoir promener son hamster. (Triviale pour quelqu'un de votre envergure)

Voici l'implémentation de cette classe. L'idéal serait de ne pas regarder ! On connait tous la technique de la feuille qui cache un peu mais qui glisse de ligne en ligne… 😊

```
public class Maitre
{
    // variables d'instance - remplacez l'exemple qui suit par le vôtre
    private int age;
    private String nom;
    private Nac animal;

    /**
     * Constructeur d'objets de classe Maitre
     */
    public Maitre()
    {
        // initialisation des variables d'instance
        age = 0;
        nom = "Laura";
    }

    public int getAge(){
        return(this.age);
    }
    public void setAge(int number){
        this.age = number;
    }
```

```
public String getNom(){
    return(this.nom);
}
public void setAge(String name){
    this.nom = name;
}
public Nac getAnimal(){
    return(this.animal);
}
public void setAnimal(Nac nc){
    this.animal = nc;
}
public String promener()
{
    // Insérez votre code ici
    return "*Arrive dans son jardin* Hey, je suis " + this.nom + "... Hoooo reviens ici "+ animal.getNom();
}
```

Figure 15: Implémentation de la classe Maitre

Nous devons par la suite compiler le tout, mais je ne devrais même plus le dire … puis exécuter notre méthode promener().

Nous allons donc créer Laura et Hamtaro. Les images ci-dessous décrivent la procédure à faire pour lier les deux petits cœurs solitaires. Leur création n'est plus un mystère pour vous alors je vous épargne les répétitions.

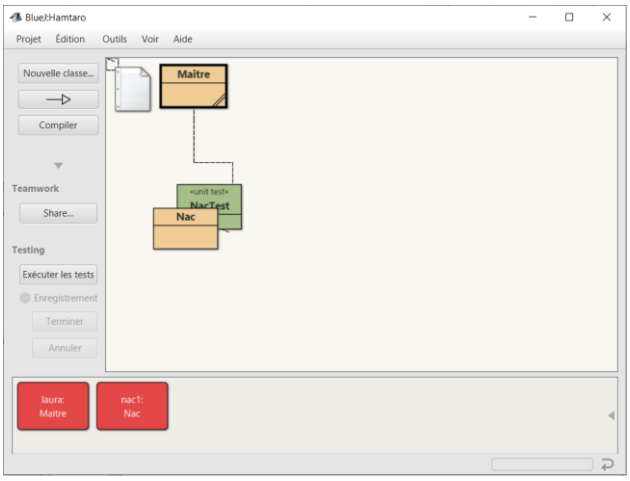


Figure 17: Création des deux instances

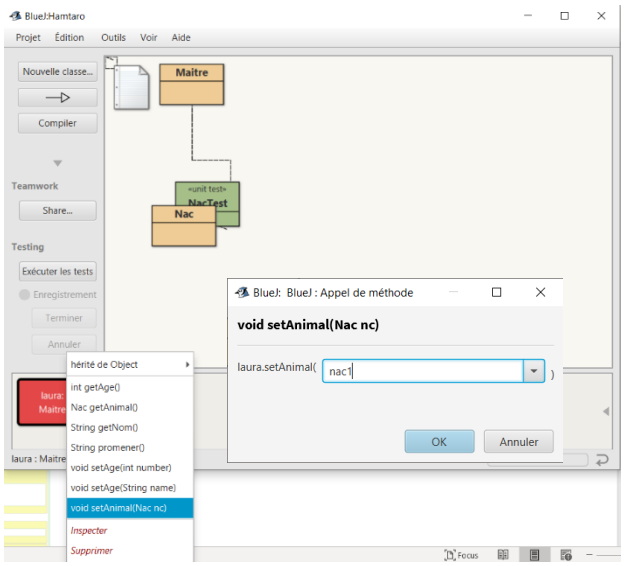


Figure 16: Associer Hamtaro à sa maîtresse Laura

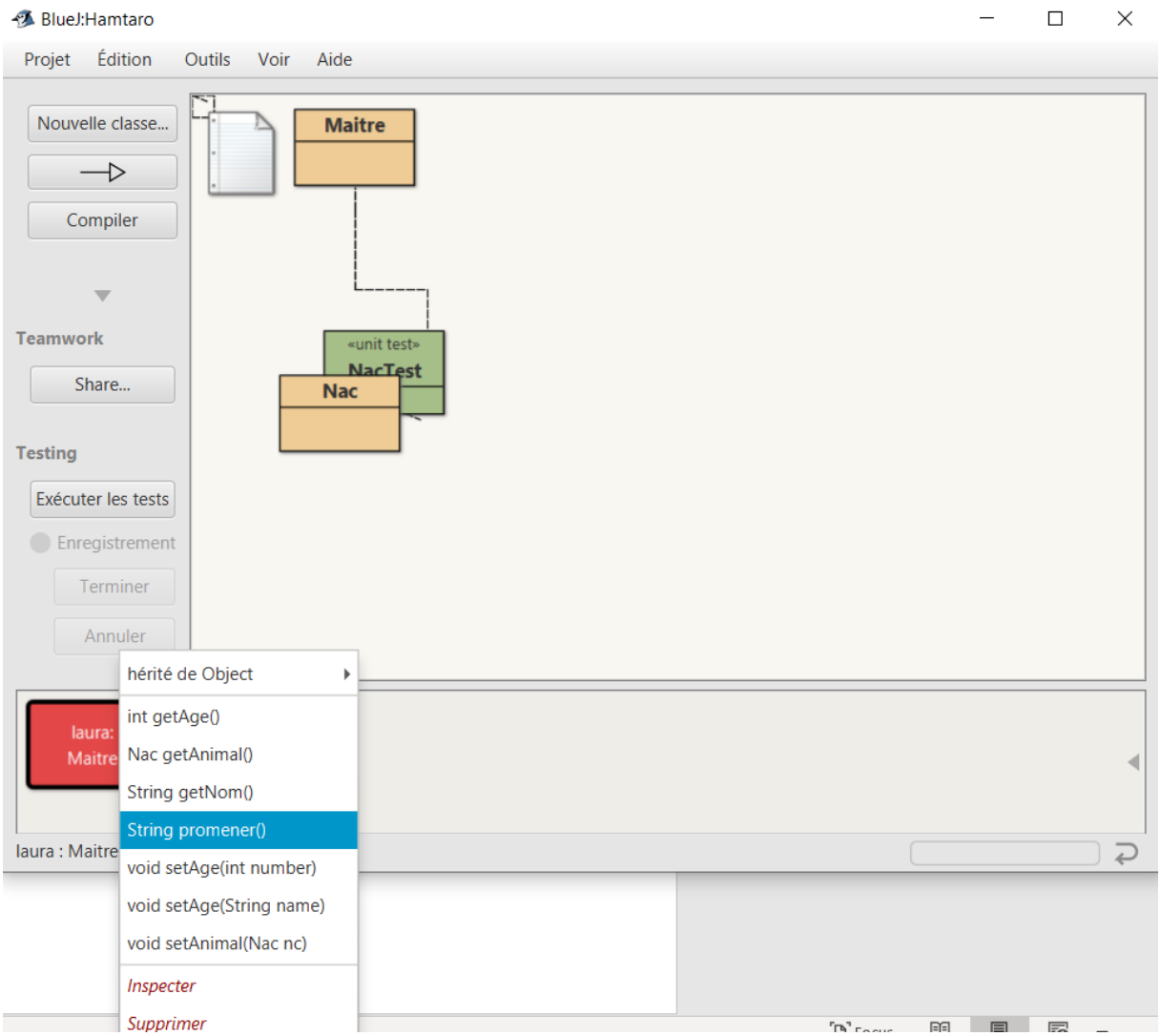
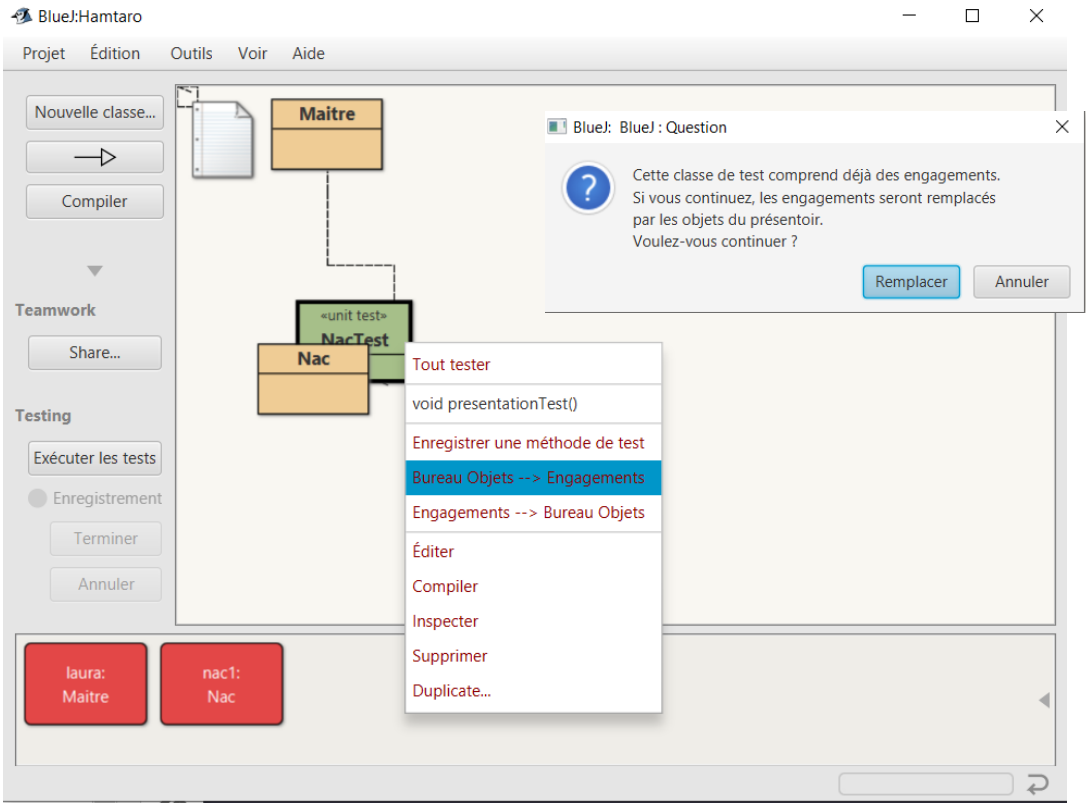


Figure 18: Débuter la promenade (espérons que tout se passe bien mais ....)

Figure 19: Résultat .. CATASTROPHE

Encore des tests ... toujours des tests

Le test de cette méthode (que vous maîtrisez si bien à présent) requiert une subtilité préalable: l'enregistrement des instances dans la classe test. ("C'est bon il m'a perdu je m'en vais...")



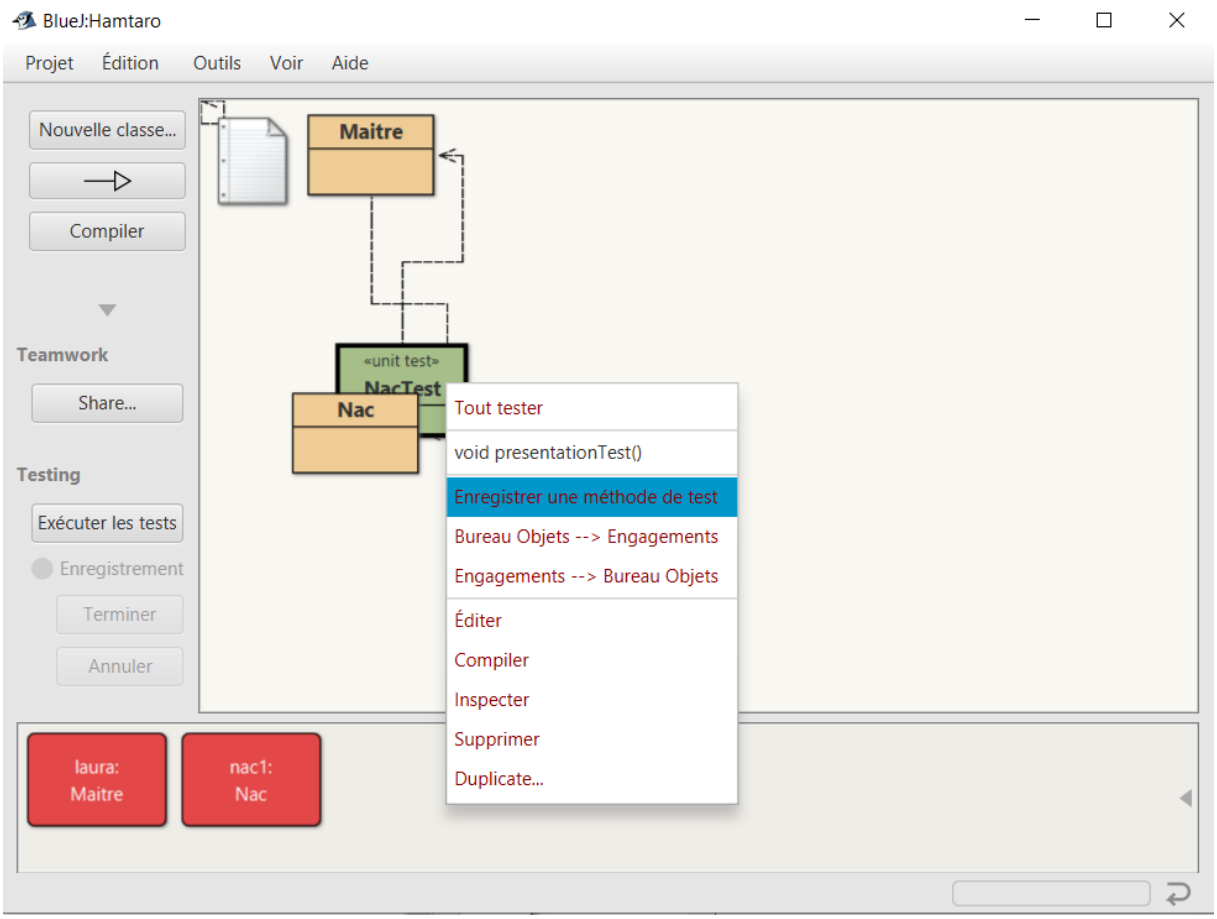
Je m'explique: lorsque vous allez tester la méthode, il faudra préalablement avoir créé un objet (correspondant à l'instance d'une classe) de chaque classe. Ensuite, en un clic (en vérité deux mais le premier ne compte pas) vous pourrez stocker l'animal et son maître dans la classe de test en faisant clic droit sur la classe de test > "Bureau Objets --> Engagements"

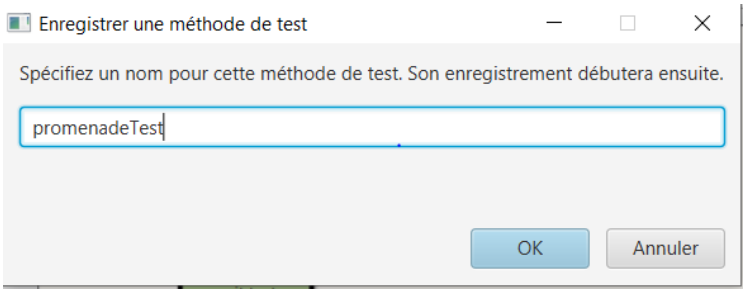
En remplaçant les données présentes par celles utiles au test, il ne vous reste plus qu'à reprendre l'étape de test comme mentionné ci-dessus et confirmer leur bonne exécution.

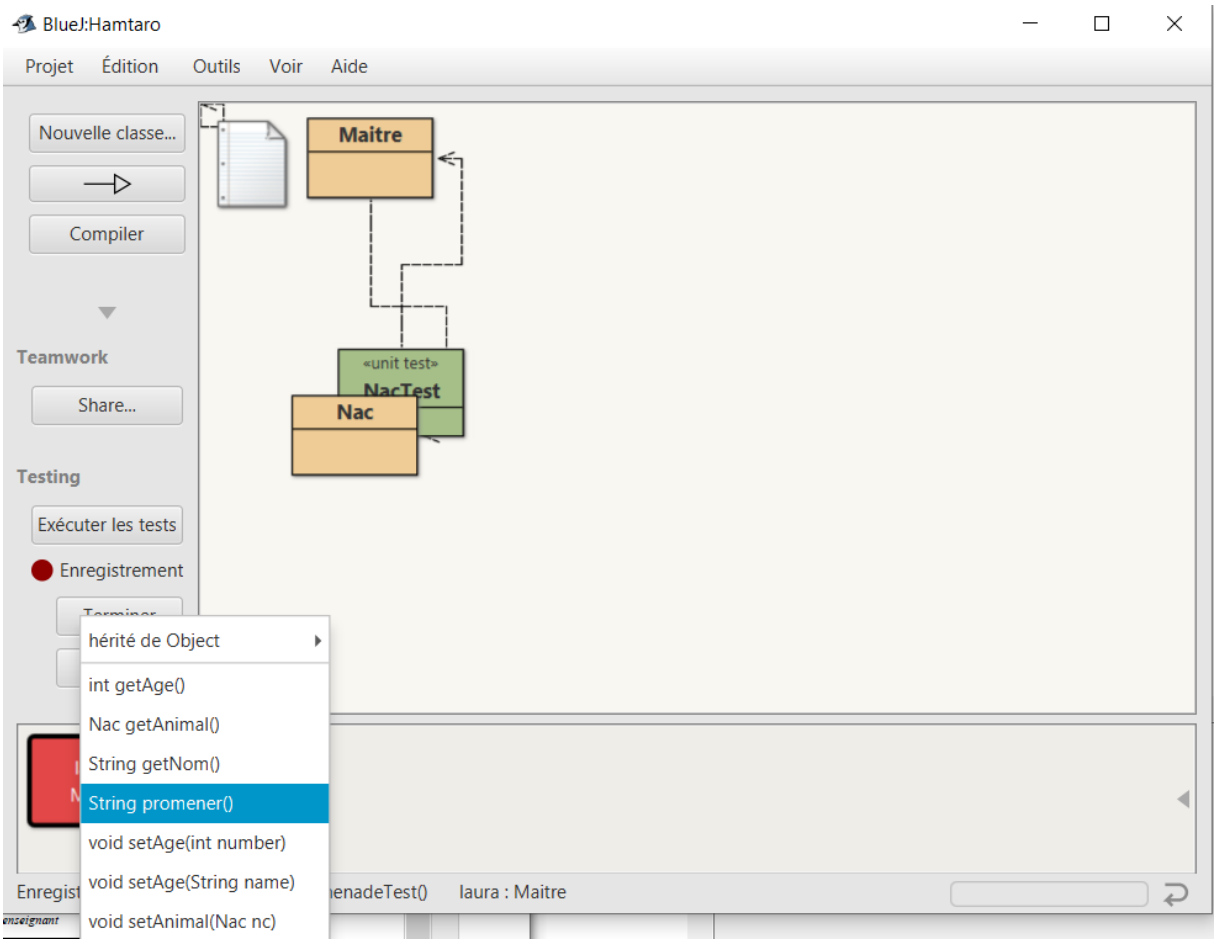
```java
@Before
public void setUp() // throws java.lang.Exception
{
    laura = new Maitre();
    nac1 = new Nac("Hamtaro", 12);
    laura.setAnimal(nac1);
}
```

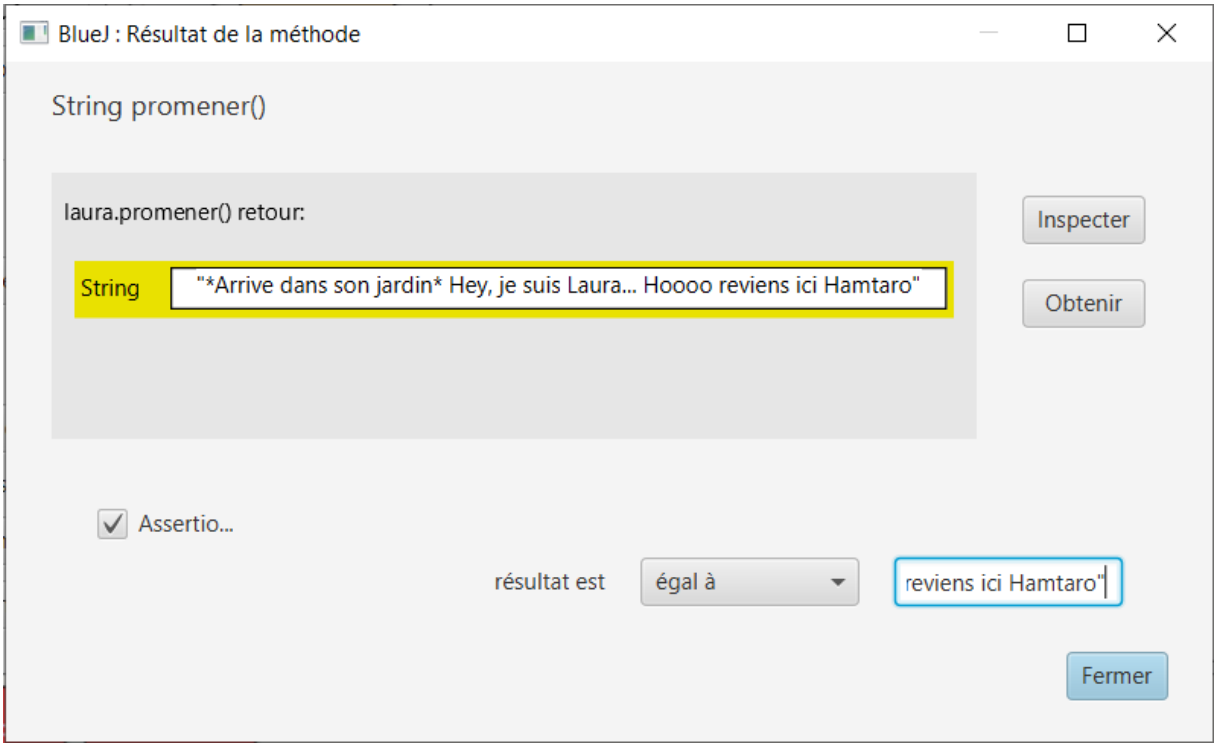
En reprenant le même principe pour la création du test de la promenade vous devriez à nouveau apercevoir non plus une mais deux barres vertes !

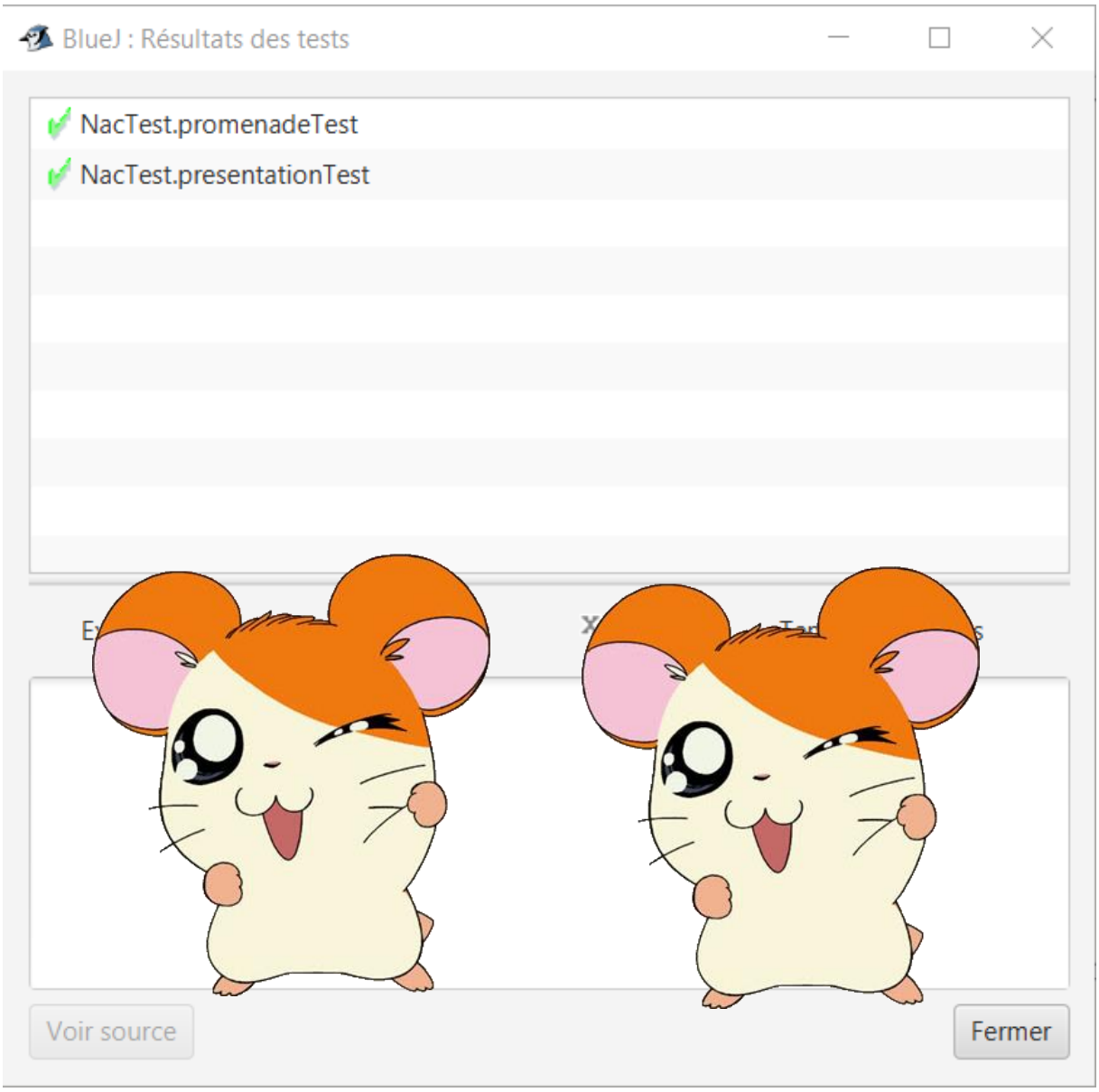
Suivez les captures suivantes :






BlueJ:Hamtaro
Projet
Édition
Outils
Voir
Aide
Nouvelle classe...
Compiler
Teamwork
Share...
Testing
Exécuter les tests
Enregistrement
Maitre
«unit test»
NacTest
Nac
hérité de Object
int getAge()
Nac getAnimal()
String getNom()
String promener()
void setAge(int number)
void setAge(String name)
void setAnimal(Nac nc)
laura : Maitre


BlueJ : Résultat de la méthode
String promener()
laura.promener() retour:
String
"*Arrive dans son jardin* Hey, je suis Laura... Hoooo reviens ici Hamtaro"
Inspecter
Obtenir
Assertio...
résultat est
égal à
reviens ici Hamtaro"
Fermer



A bientôt pour de nouvelles aventures ….