TP N°1 – Agilité – Les tests unitaires

*HADI Ismaïl & JOLIMAN Iris*

Table des matières

[Première partie : BlueJ 2](#_Toc38982084)

[*Installation de BlueJ 2*](#_Toc38982085)

[*Création d’un nouveau projet 2*](#_Toc38982086)

[*Création de la classe « fétiche » : NAC 4*](#_Toc38982087)

[*Tu vas donner vie à un Hamster ! (c’est magique) 5*](#_Toc38982088)

[*Hamtaro se présente-t-il vraiment bien ? 8*](#_Toc38982089)

[*Encore des tests … toujours des tests 14*](#_Toc38982090)

# Première partie : BlueJ

Q1. télécharger 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).

L’outil BlueJ, que nous allons utiliser au sein de ce tutoriel, permettra d’allier la schématisation à l’implémentation technique.

## Installation de BlueJ

Q2. Installer BlueJ sur la machine

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

Q3. Créer un nouveau projet

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

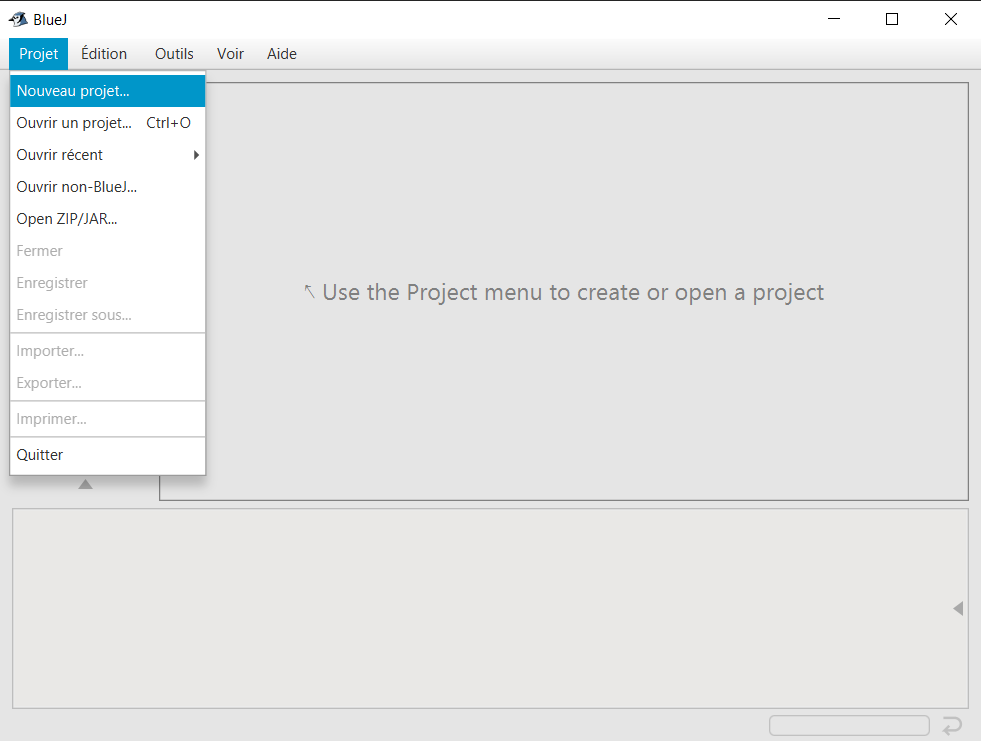


Figure 1: Création d'un nouveau projet dans BlueJ

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 😊.

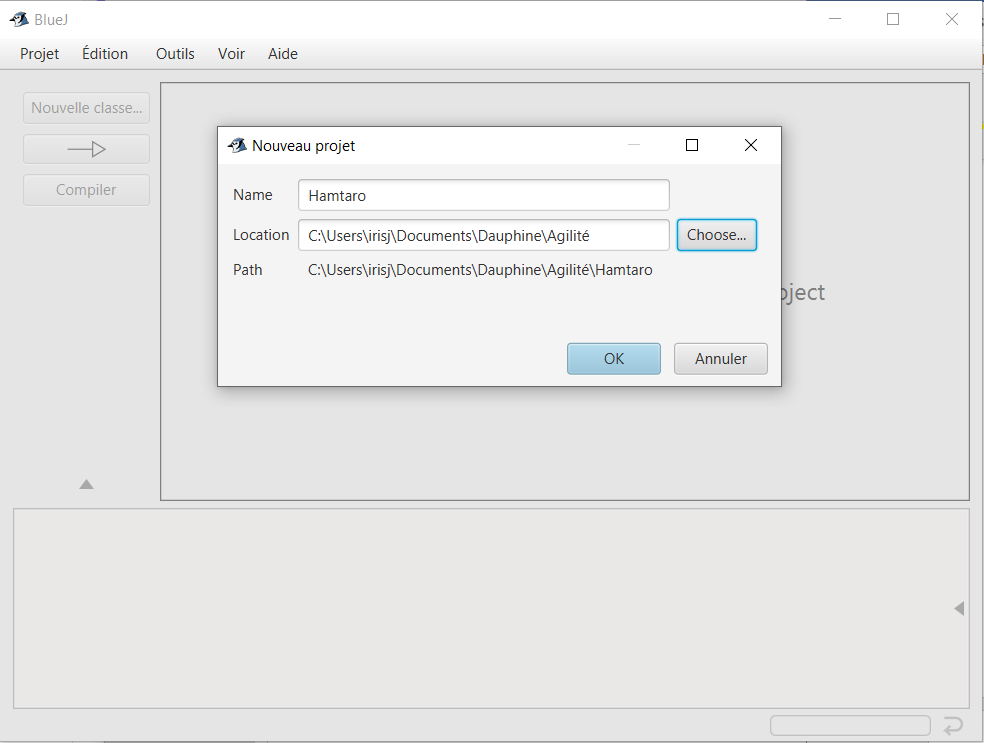


Figure 2: Nommer le projet

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.

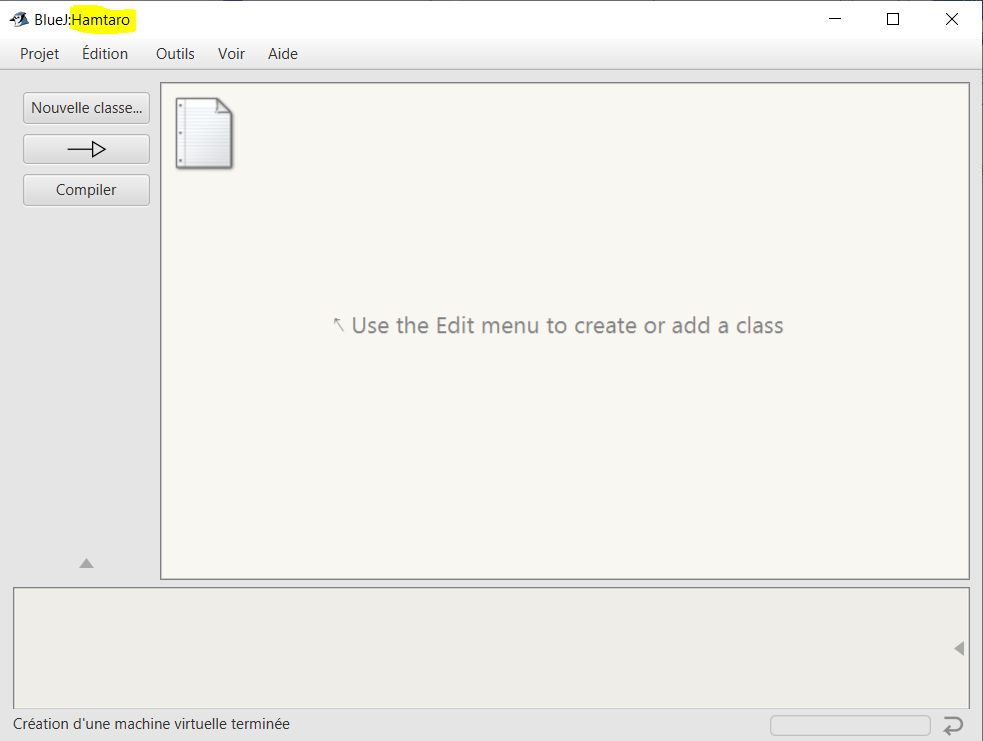


Figure 3: Fenêtre après la création de projet

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 sort 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

Q4. Créer la classe fétiche

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.

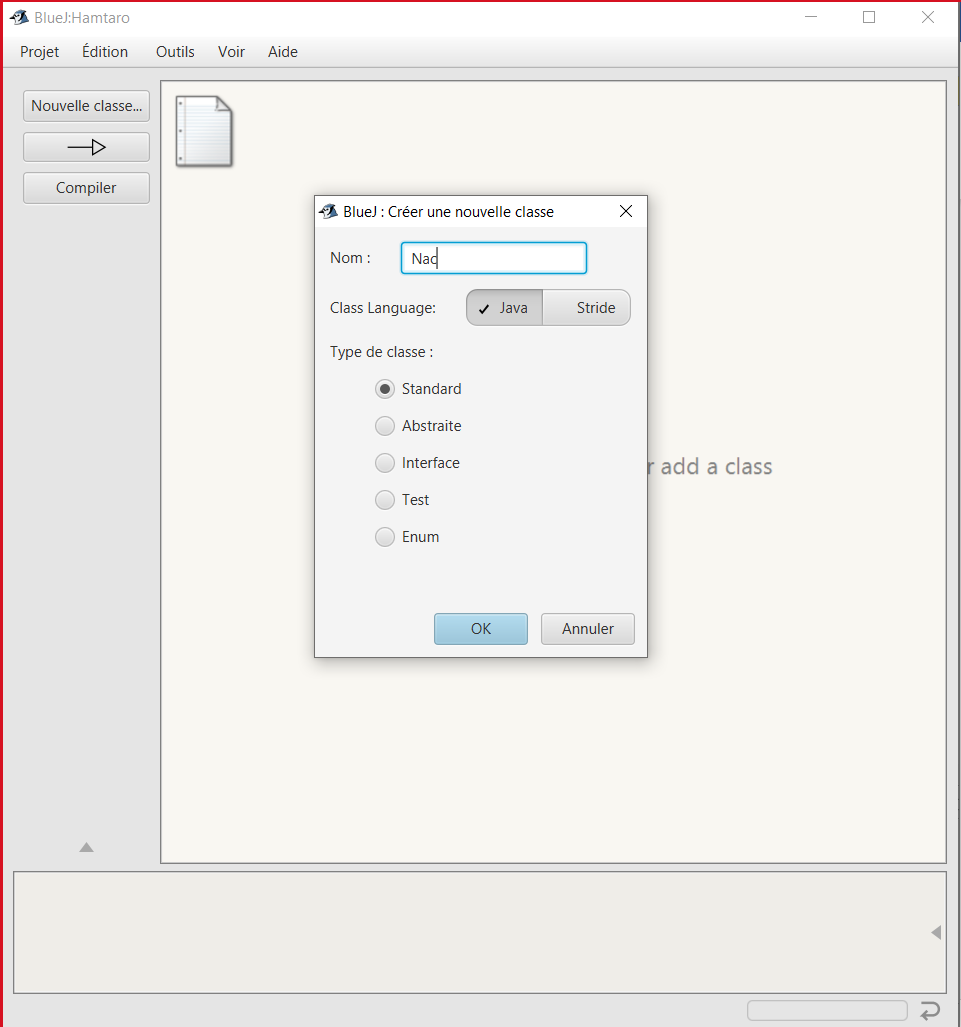


Figure 4: Définition de notre nouvelle classe (type standard)

Q5. Compiler la classe

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.



[Cette photo](https://en.wikipedia.org/wiki/File:Information.svg) par Auteur inconnu est soumise à la licence [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/)

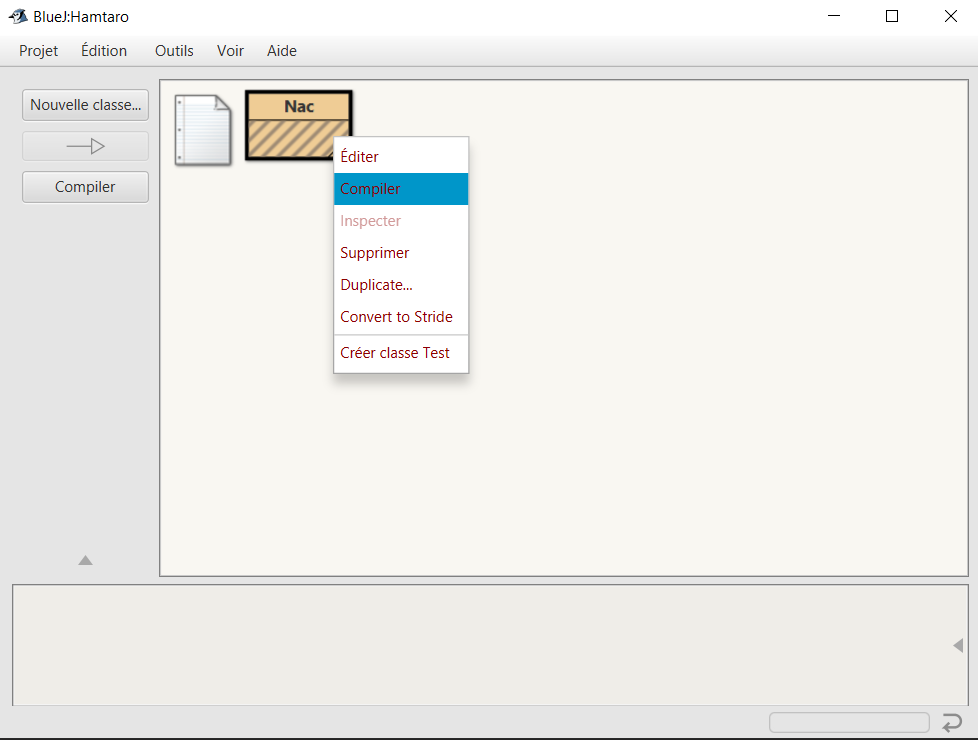
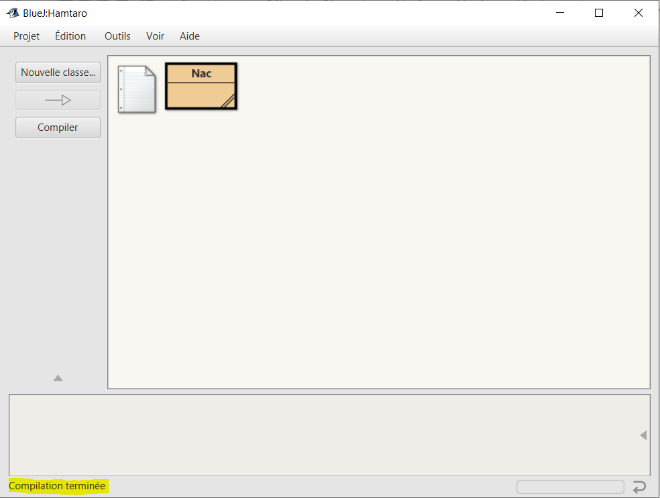


Figure 5: Classe NAC avant et après compilation

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)

Q6. Instancier la classe

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).

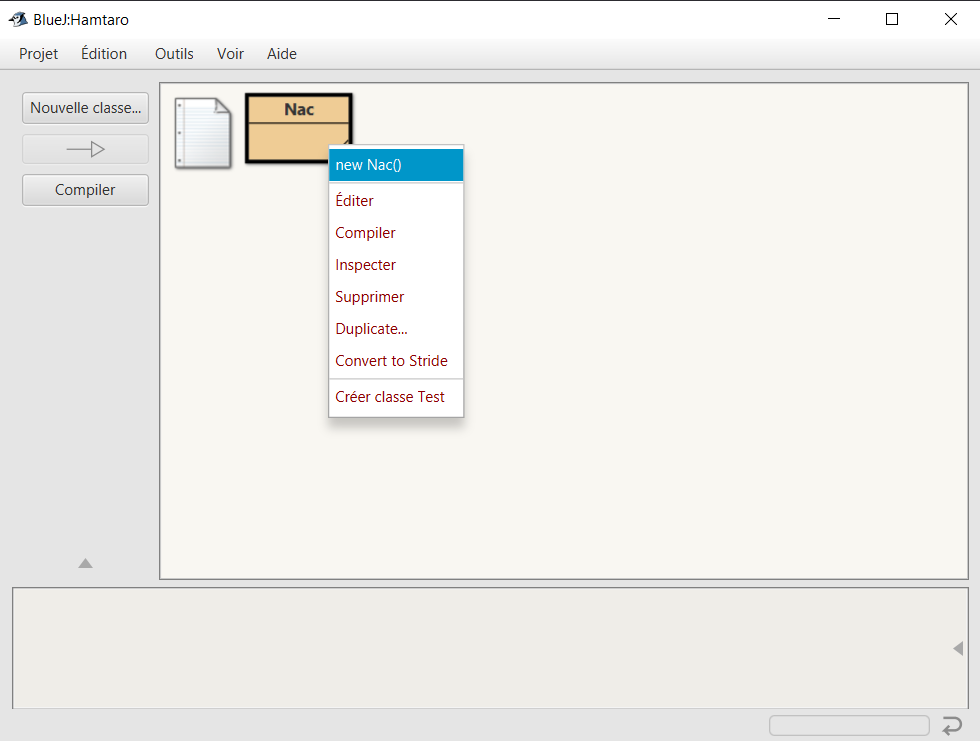
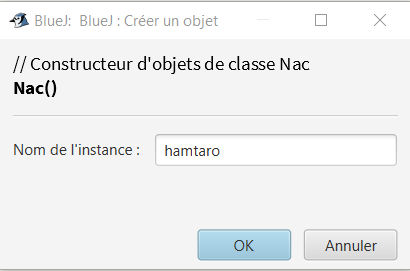
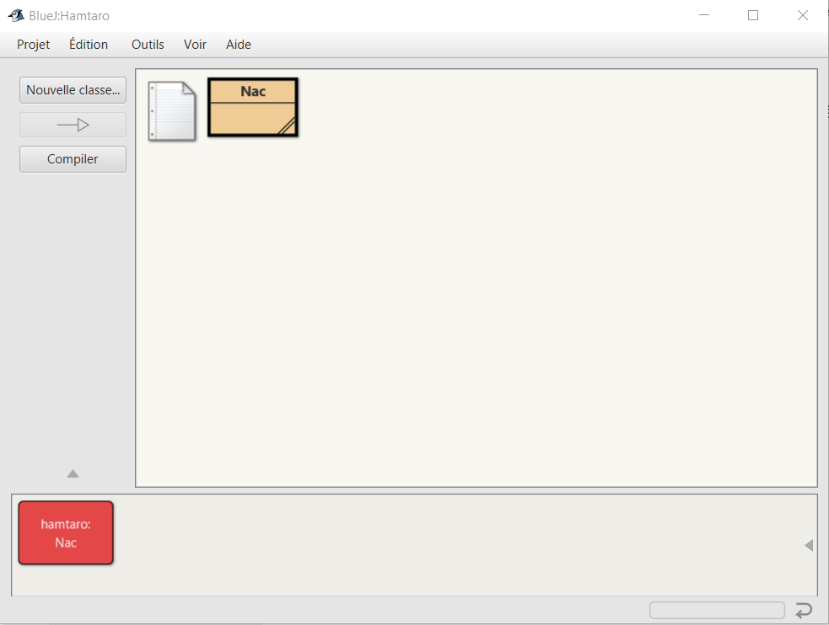


Figure 6: Création de l'instance hamtaro

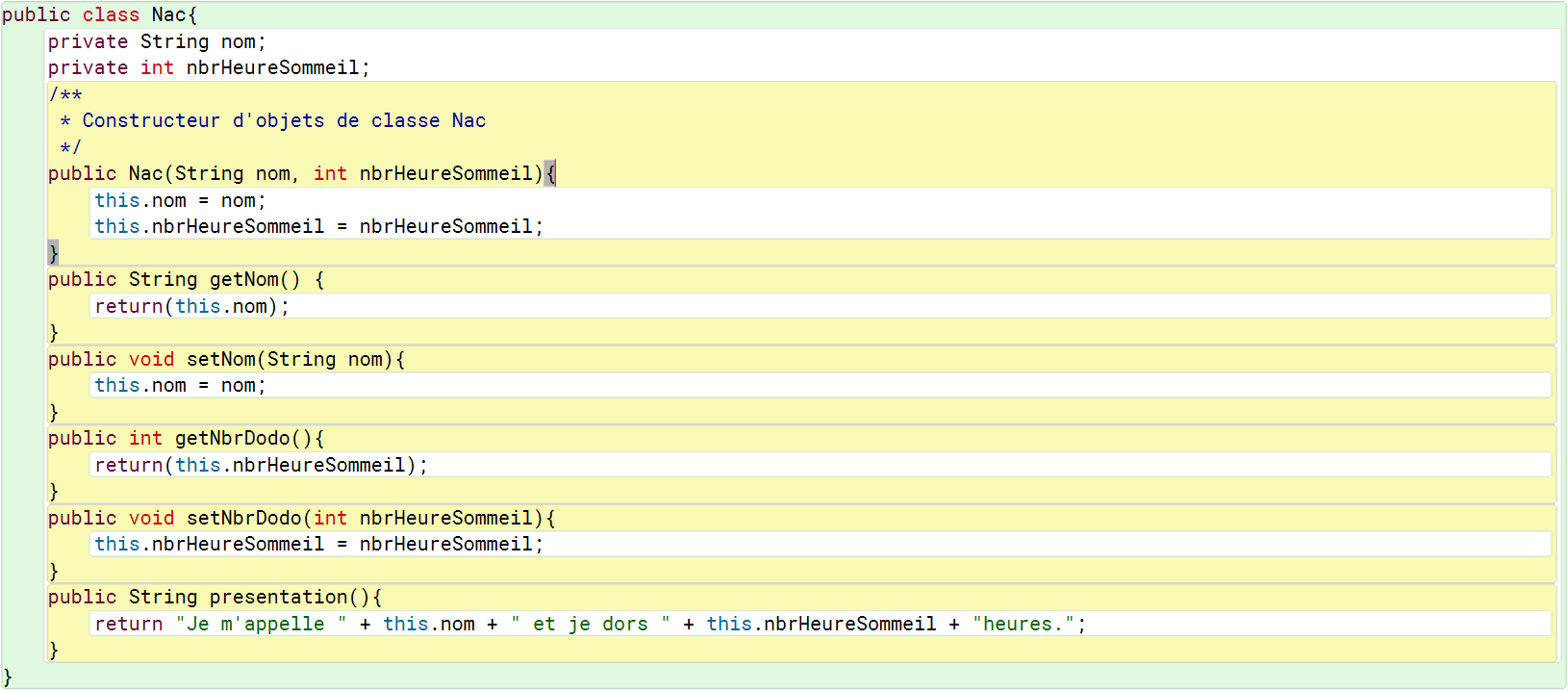
On peut voir ici qu’Hamtaro a bien été créé (dans la partie rouge)

Q7. Ajouter deux attributs avec accesseurs et une méthode qui les manipulent

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 :



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.

Q8. Instancier à nouveau

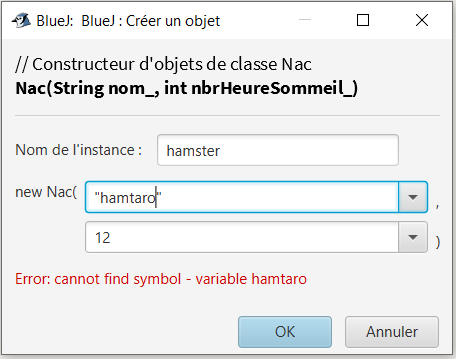
Nous avons par ailleurs modifié le constructeur de la classe (nous ne sommes pas Dieu mais pouvons tout de même mettre au monde nos animaux). Ainsi, notre animal peut être baptisé avant de voir le jour.

Figure 7: Création d'un nouvel hamtaro avec de nouveaux attributs

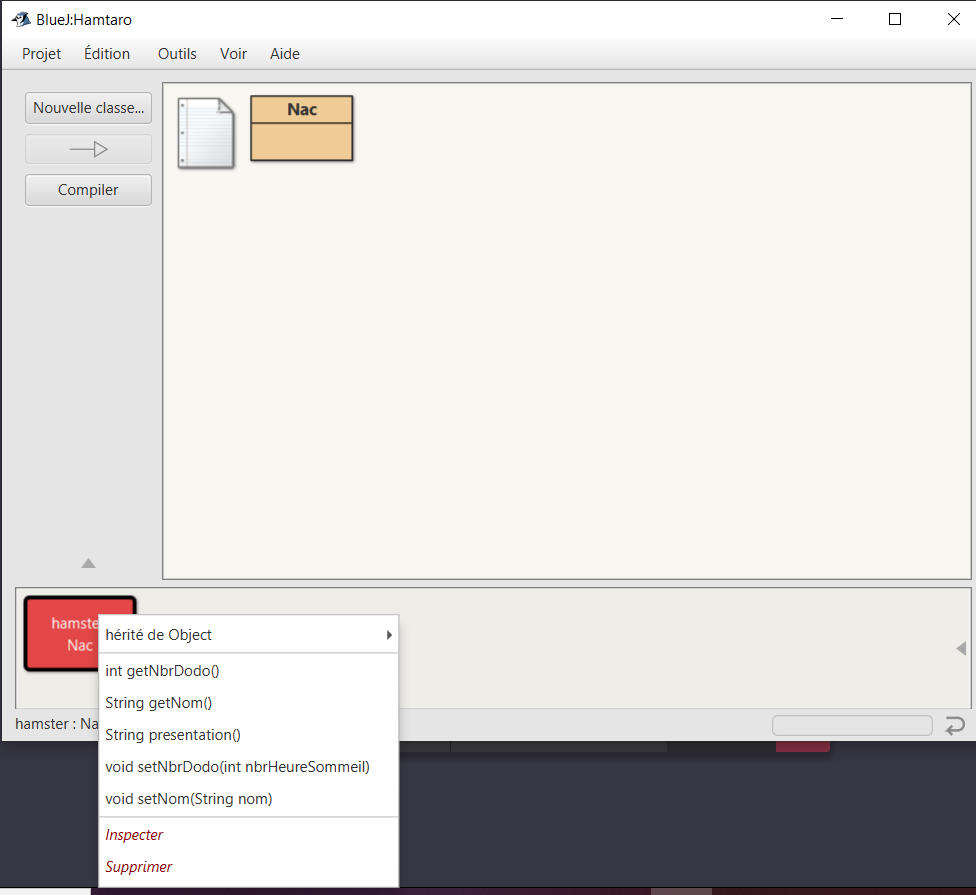


Figure 8: Lancement de la fonction presentation()

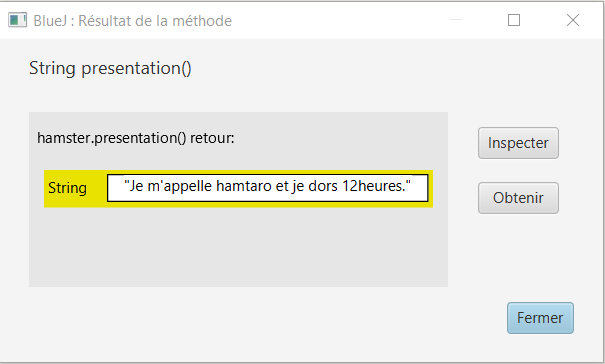
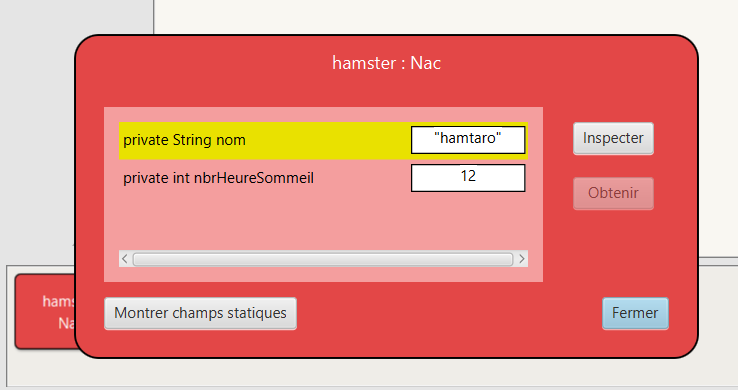


Figure 9: Vérifications en cliquant sur "Inspecter"

Figure 10: Retour de la fonction présentation

## Hamtaro se présente-t-il vraiment bien ?

Q9. Tester unitairement la classe et montrer la barre verte.

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.

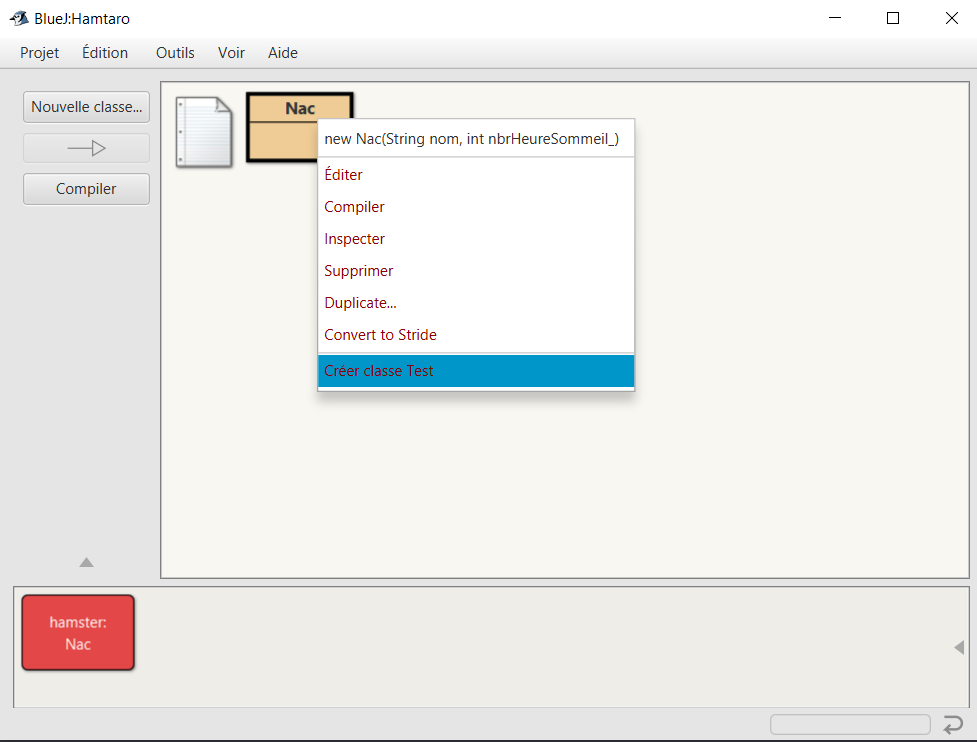


Figure 11: Création de NacTest

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"

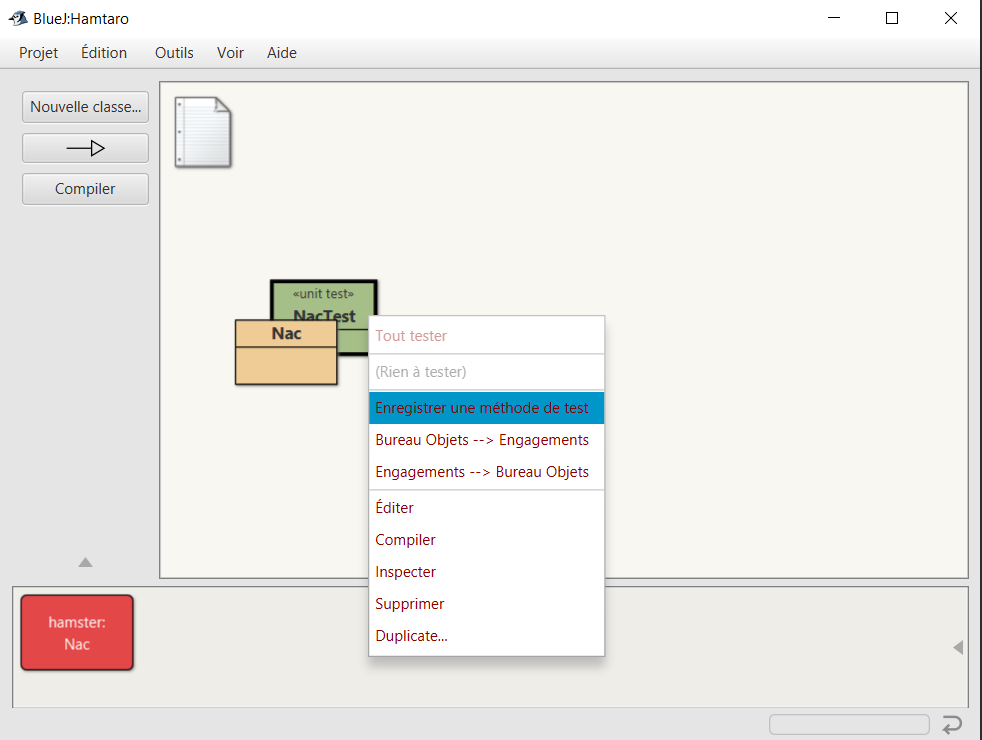


Figure 12: Enregistrer une méthode de test

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 !)

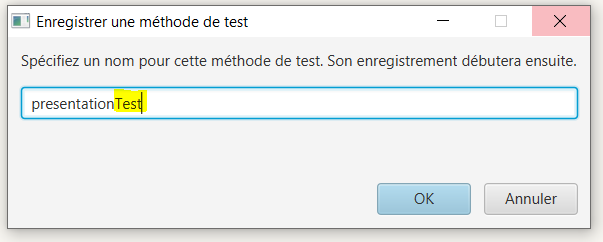
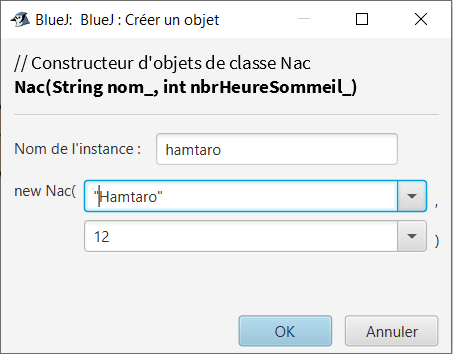
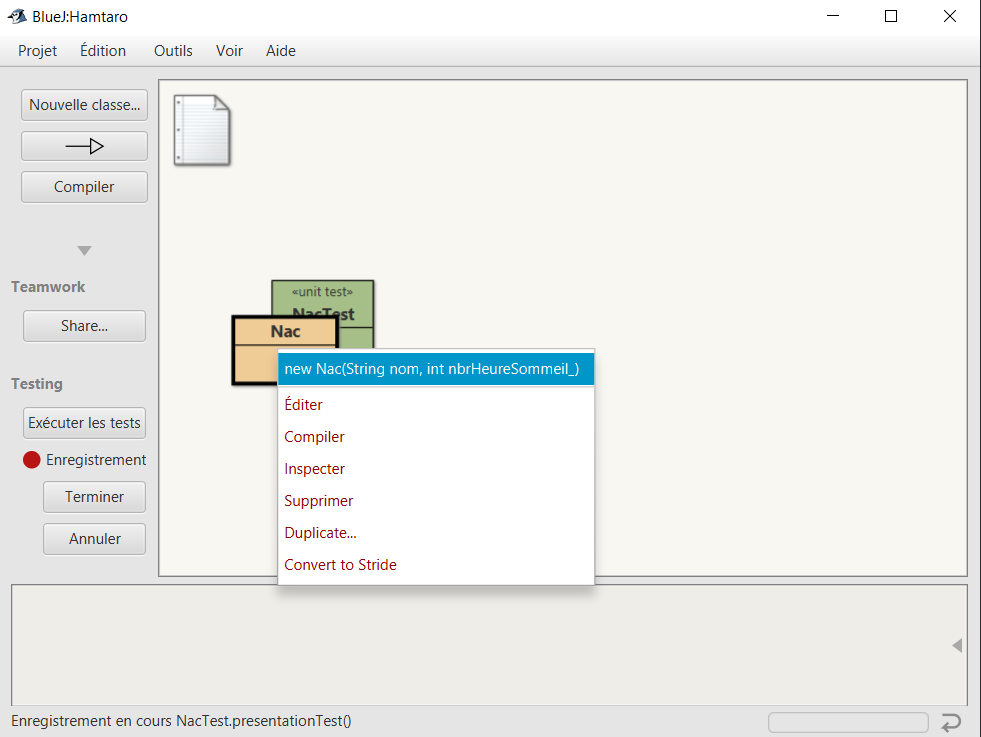
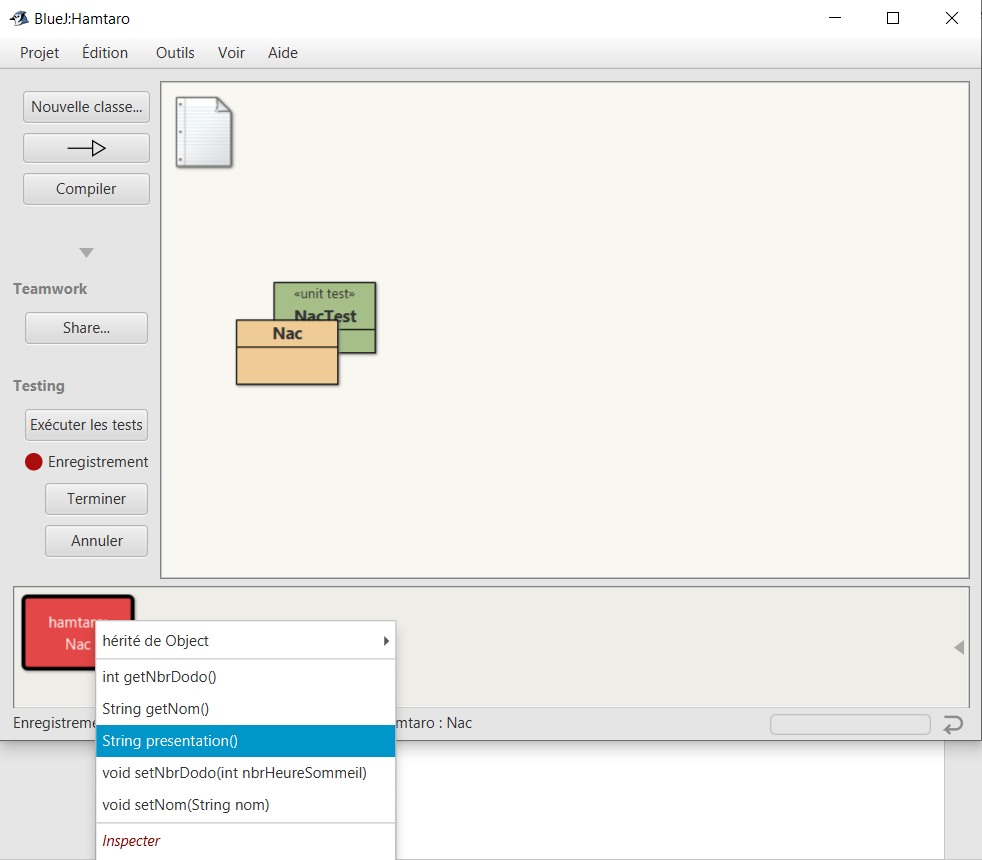


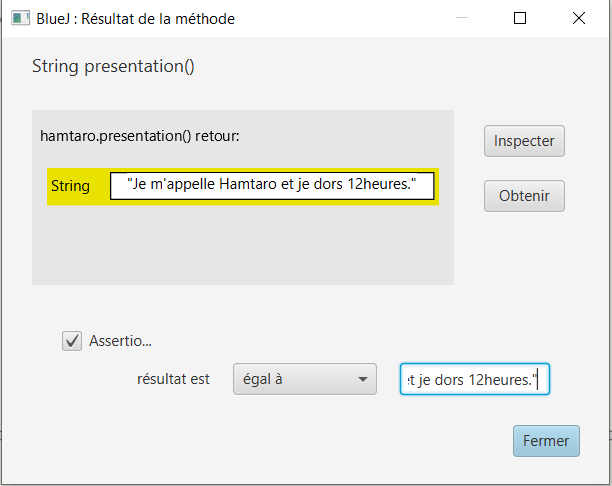
Figure 13: PopUp intergalactique de fusion (bon ok.. c'est juste un nom)

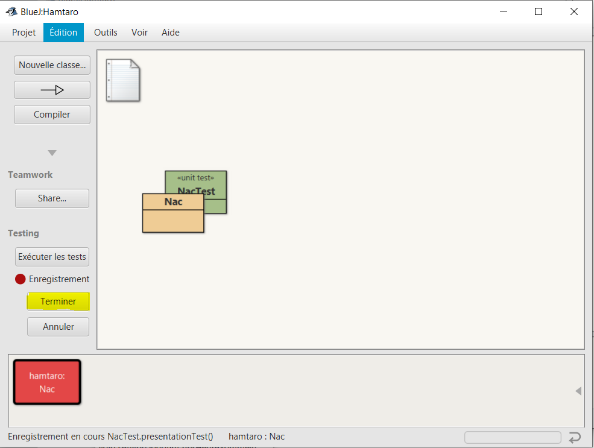
Vous constaterez dès la validation qu'un bouton rouge "enregistrement" apparaît sur la gauche de votre écran et matérialise le début des tests. Pour tester notre fonction présentation, il suffit de créer un nouvel objet puis de cliquer (Hé oui toujours le droit)  sur cet objet créé en rouge et enfin 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.

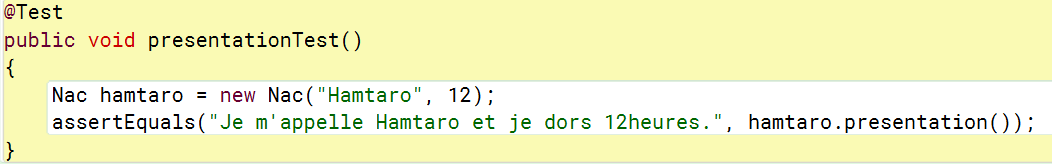








Si vous êtes curieux, cliquez double sur la nouvelle classe NacTest et vous devriez voir le code suivant apparaître ! (Waaaaaaaaaaaa)



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.

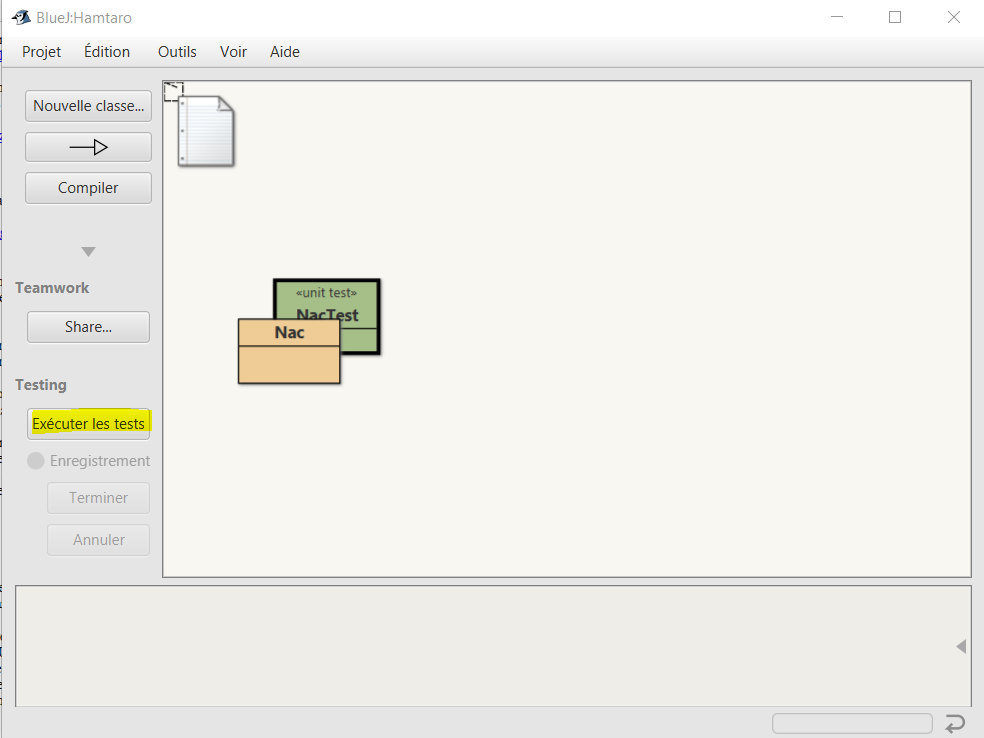


Figure 14: Exécution des tests

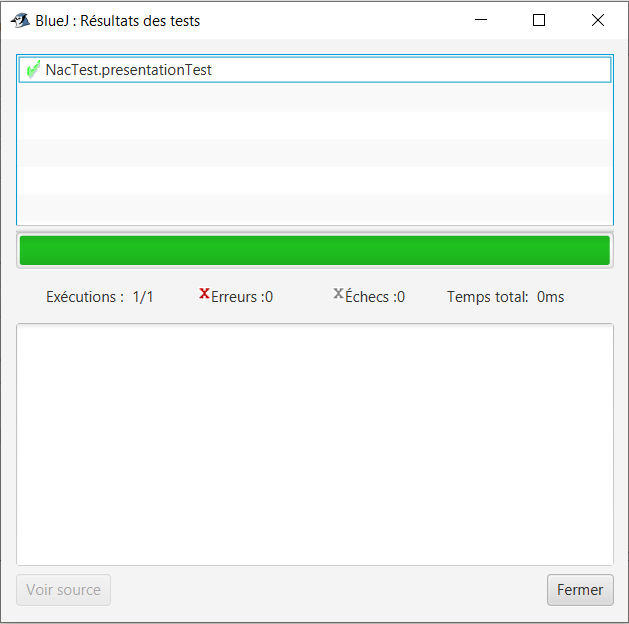


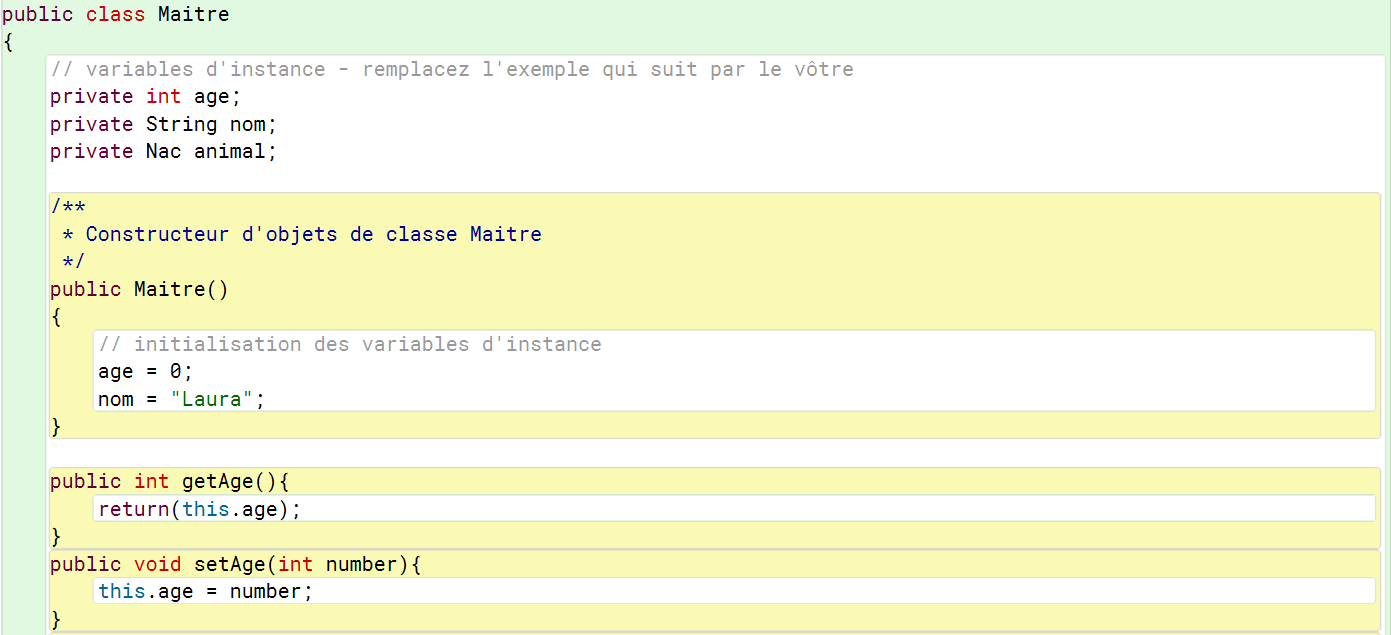
Figure 15: La barre est verte pas de rouge à l'horizon tout est bon

Q10. Ajouter une seconde classe et associer la à la classe fétiche avec une multiplicité 0..1 à 0...1

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)

Q11. Ajouter une méthode qui collabore avec la classe fétiche

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… 😊



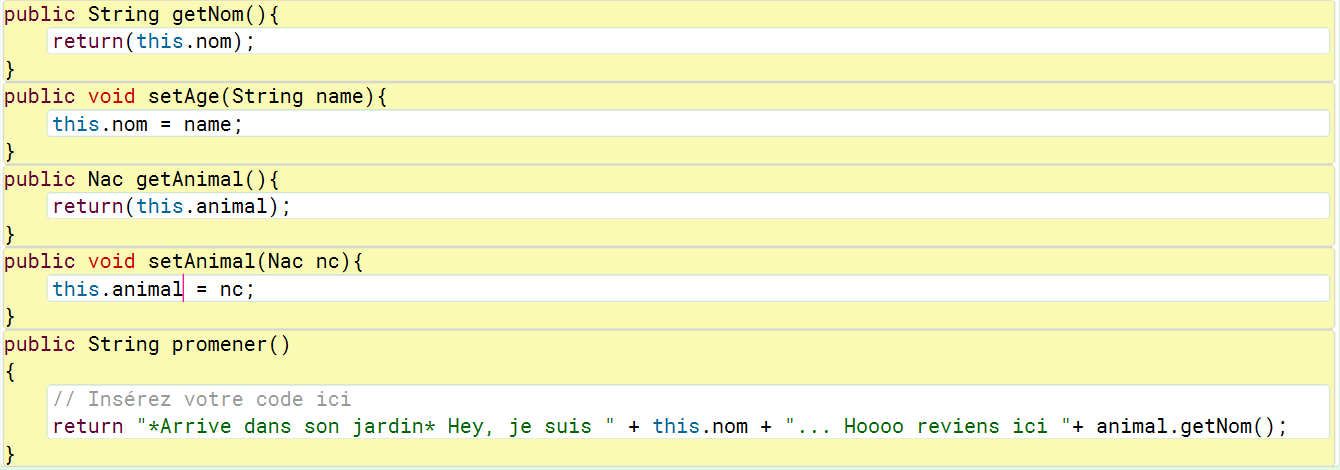


Figure 16: Implémentation de la classe Maitre

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.

Q12. Instancier les classes et relier les objets, les sauvegarder dans la ficture d’une classe test

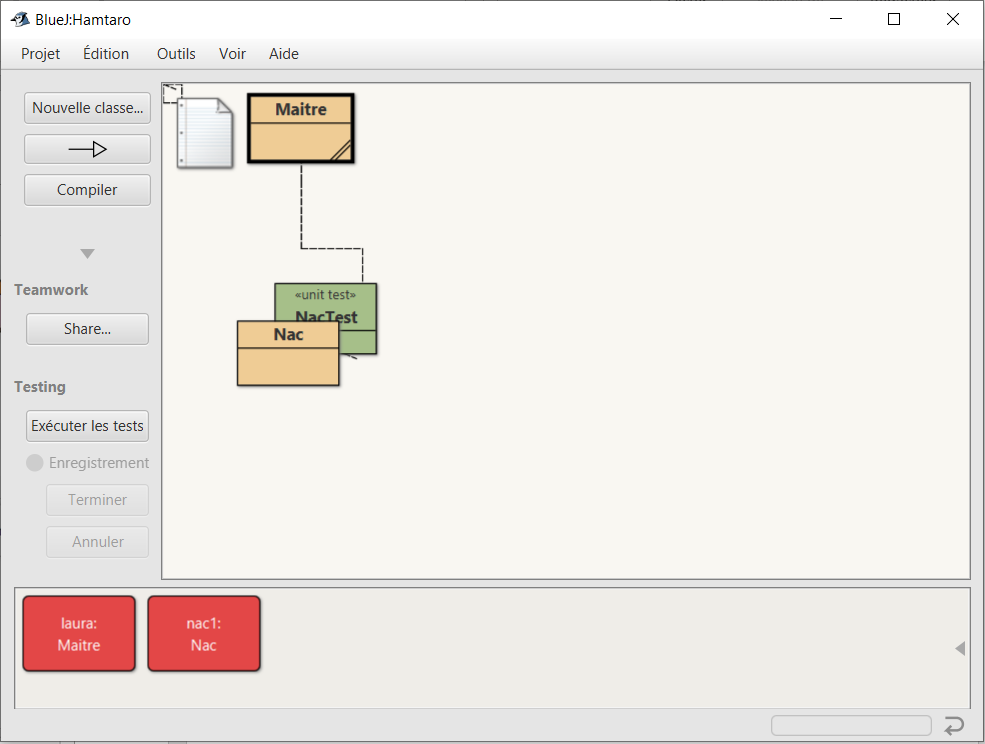
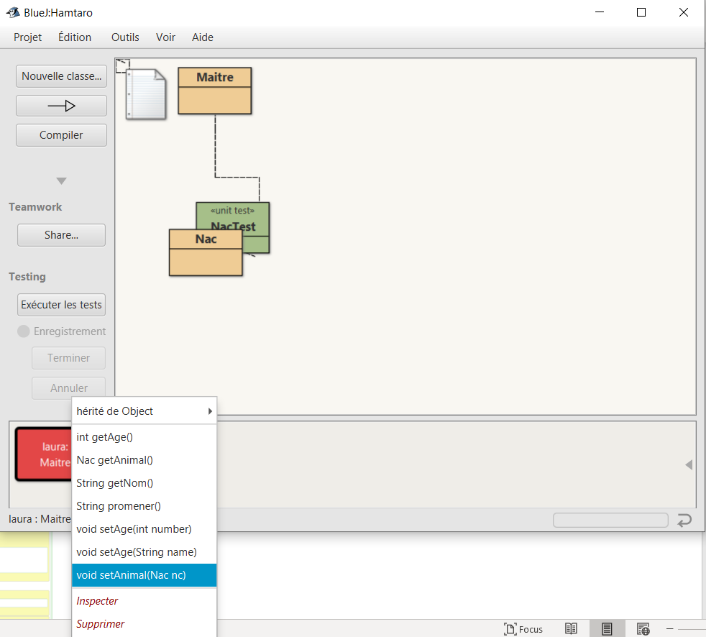
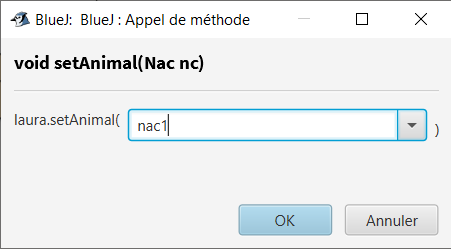


Figure 17: Associer Hamtaro à sa maîtresse Laura

Figure 18: Création des deux instances

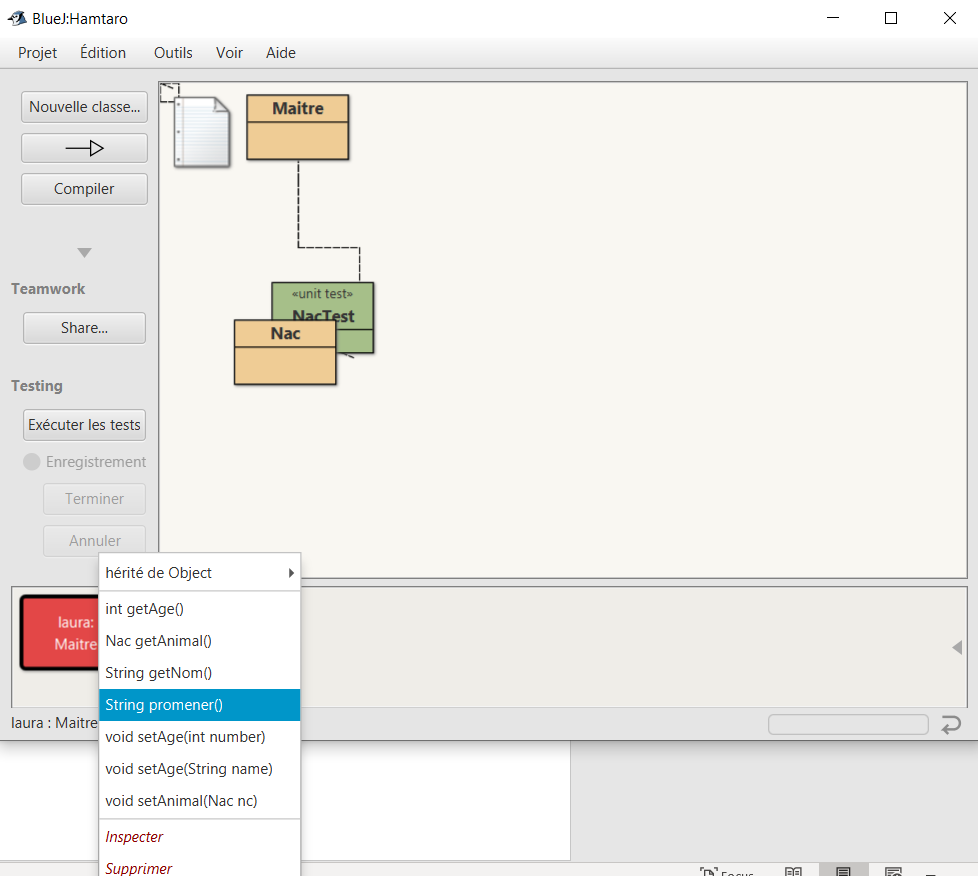


Figure 19: Débuter la promenade (espérons que tout se passe bien mais ....)

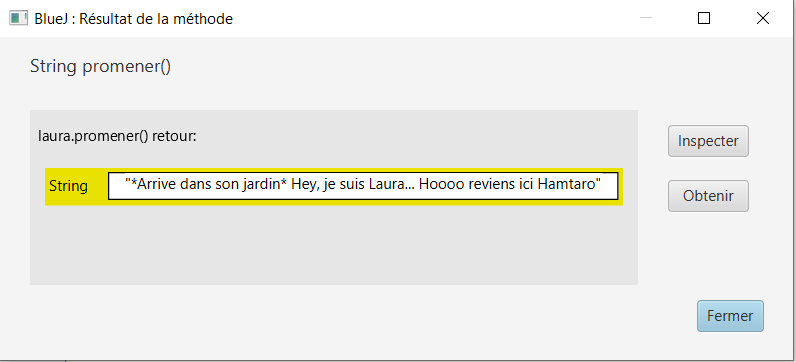
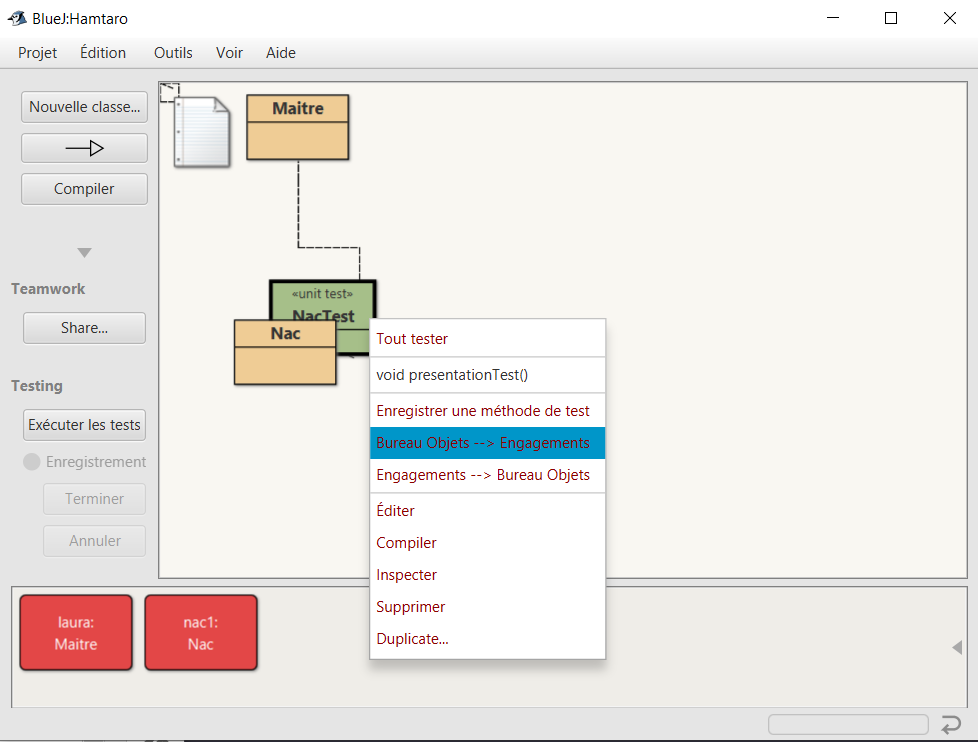
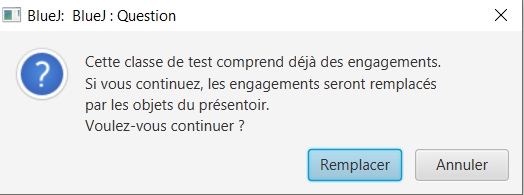


Figure 20: Résultat .. CATASTROPHE

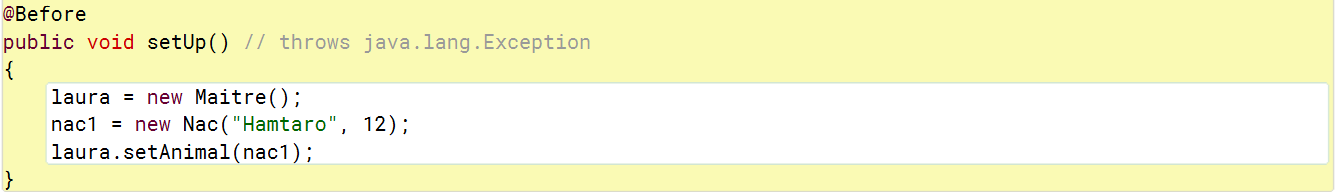
## Encore des tests … toujours des tests

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"

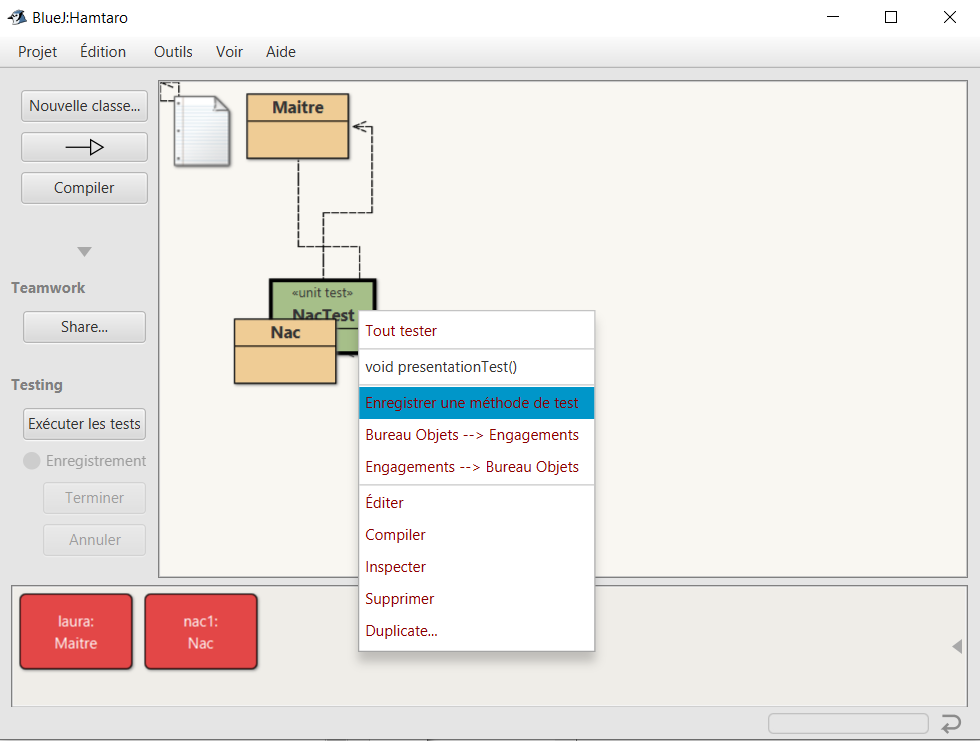
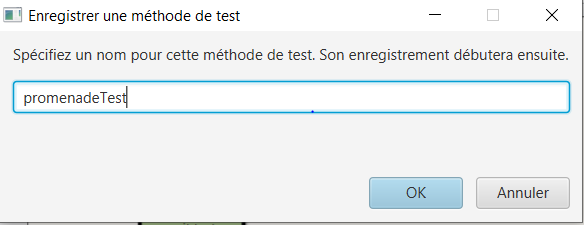
En remplaçant les données présentes par celles utiles au test, il ne vous reste plus qu'à reprendre l'étape de test comme mentionné ci-dessus et confirmer leur bonne exécution.

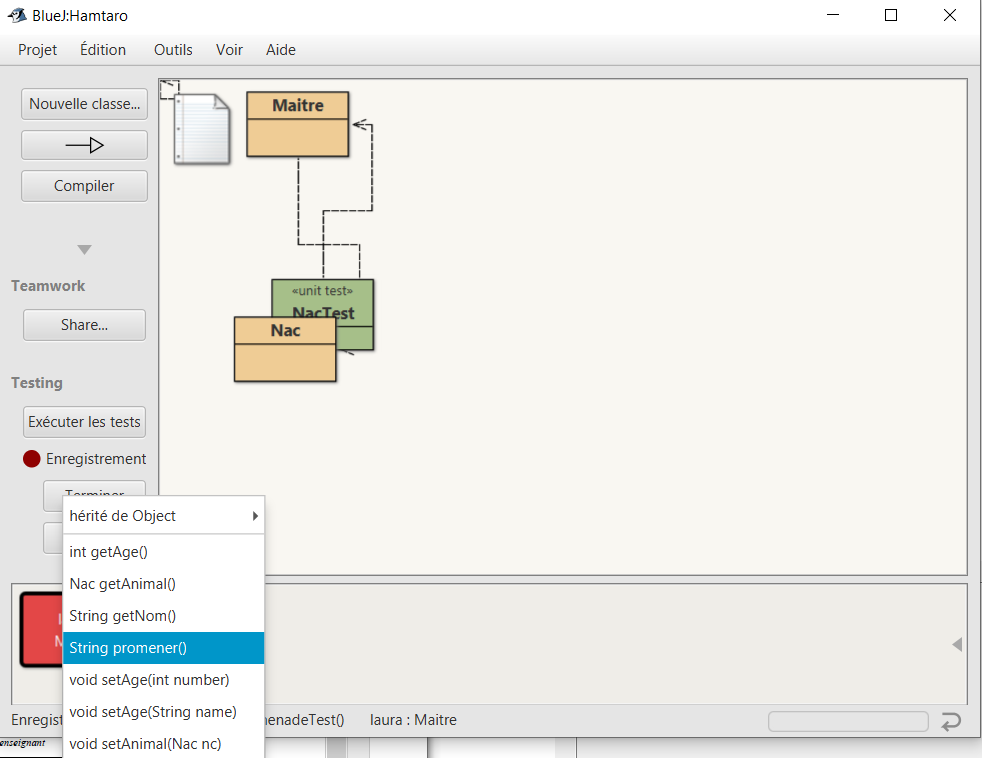


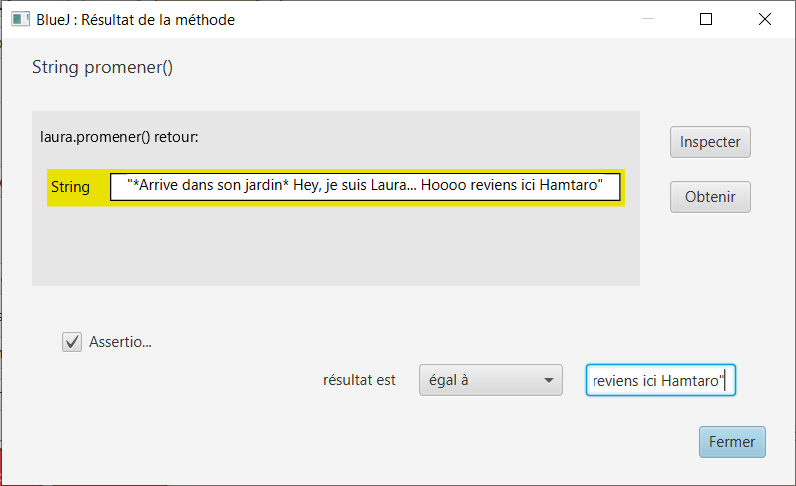
En reprenant le même principe pour la création du test de la promenade vous devriez à nouveau apercevoir non plus une mais deux barres vertes !

Q13. Créer interactivement une méthode de test qui utilise la fixture et montrer le résultat de la barre

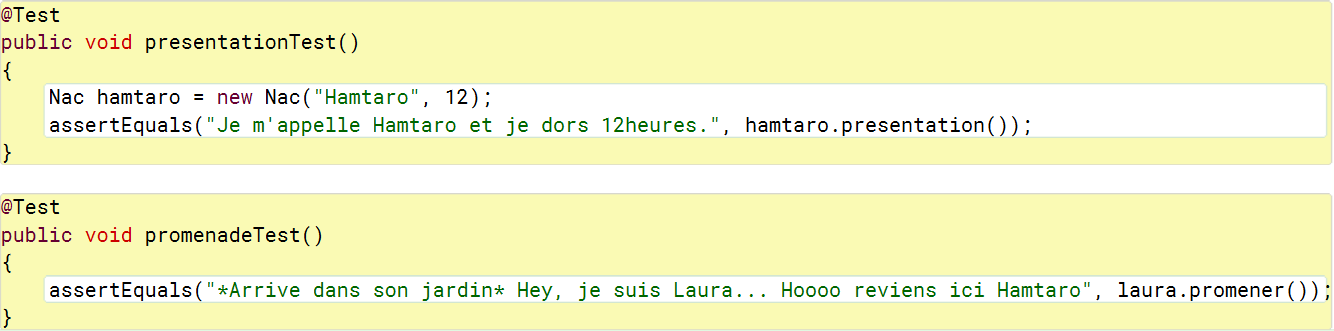
Suivez les captures suivantes :

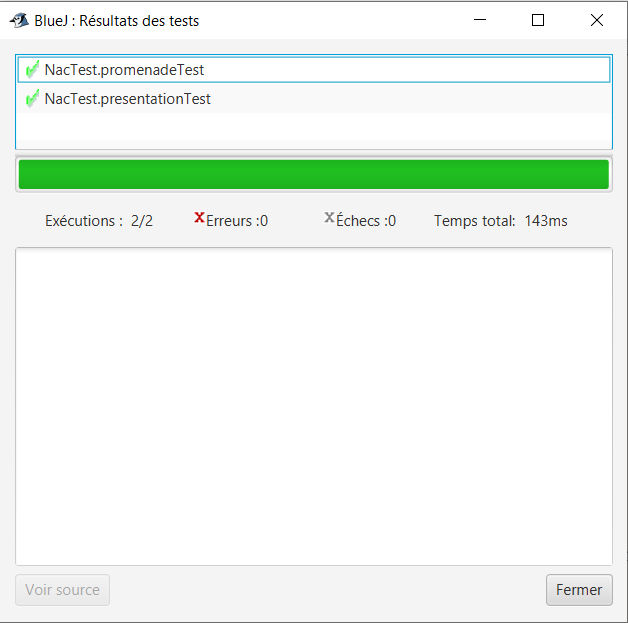






Voici le code généré par les manipulations précédentes :



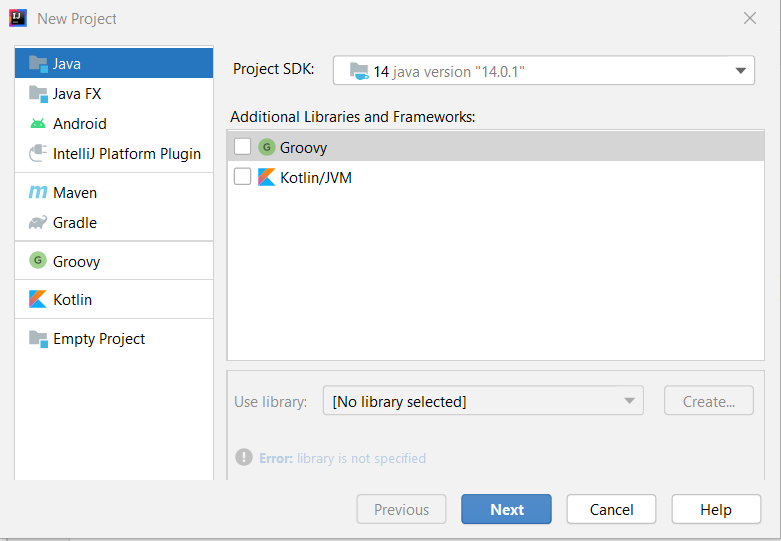


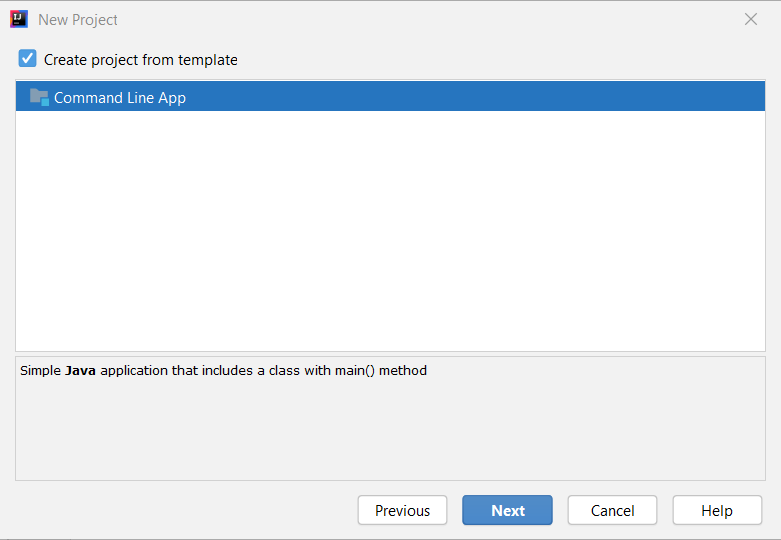
A bientôt pour de nouvelles aventures ….

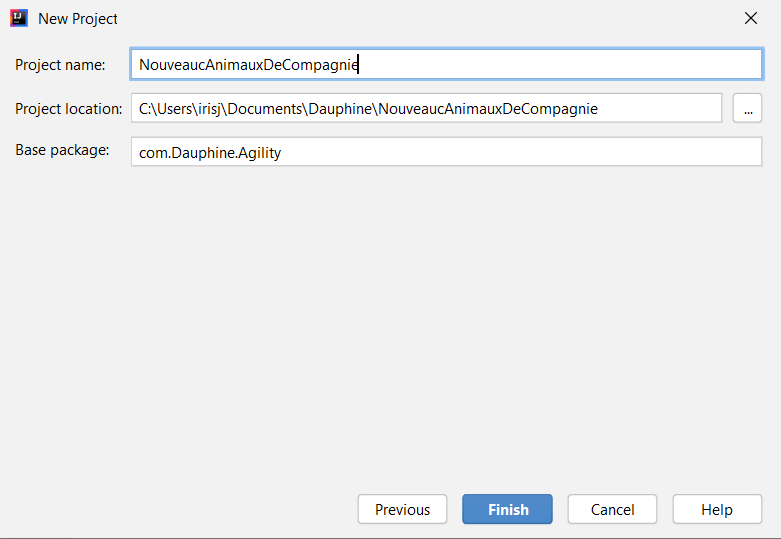
# Seconde partie : Eclipse et jUnit

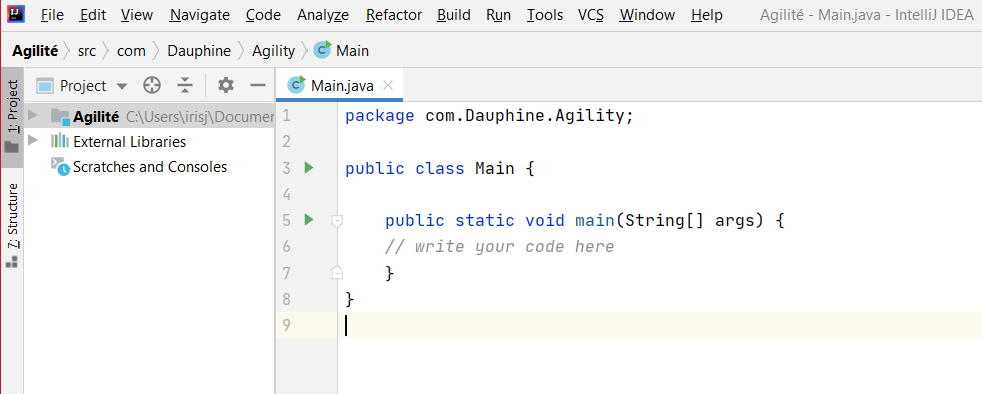
## Création d’un projet avec IntelliJ

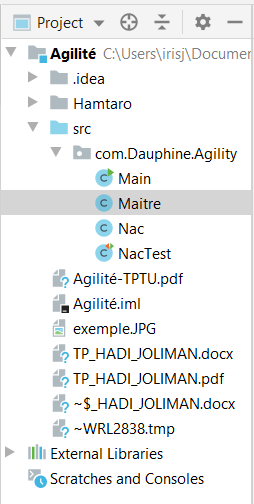
Q14. Créer un projet Eclipse

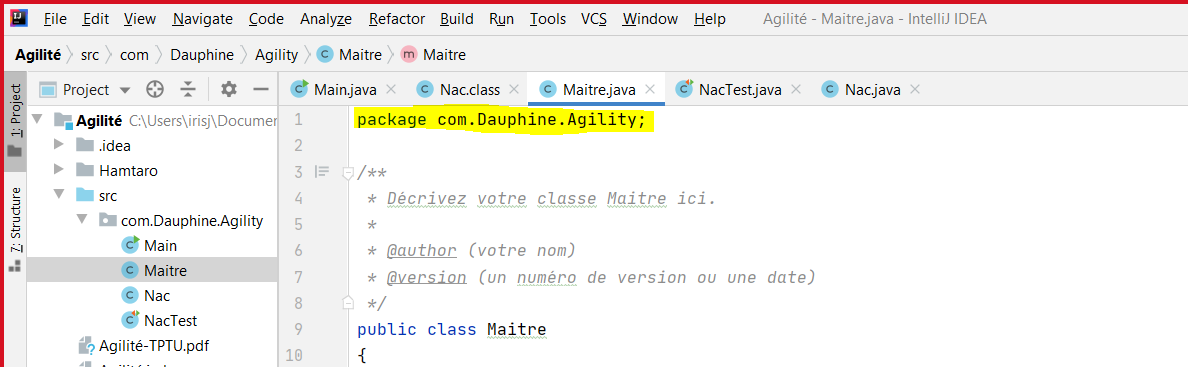












package com.Dauphine.Agility;  
  
import java.util.ArrayList;  
  
public class Maitre  
{  
 *// \*\*\*\*\*\* ATTRIBUTS \*\*\*\*\*\* //* private int age;  
 private String nom;  
 private ArrayList<Nac> animaux = new ArrayList<Nac>(); *// Un maître possède 0 ou plusieurs NAC  
  
 // \*\*\*\*\*\* CONSTRUCTEURS \*\*\*\*\*\* //  
  
 // Constructeur vide* public Maitre() {  
 this.age = 0;  
 this.nom = "Laura";  
 }  
 *// Ce constructeur permet de créer un maître ne possédant pas de nac* public Maitre(int age, String nom) {  
 this.age = age;  
 this.nom = nom;  
 }  
  
 *// Ce constructeur permet de créer un maître possédant des Nacs* public Maitre(int age, String nom, ArrayList<Nac> animal) {  
 this.age = age;  
 this.nom = nom;  
 this.animaux = animal;  
 for (Nac puppy : this.animaux) {  
 Maitre puppyMaitre = puppy.getMaitre();  
 if(puppyMaitre != null) {  
 if (!puppy.getMaitre().equals(this)) {  
 puppy.setMaitre(this);  
 }  
 }  
 else{  
 puppy.setMaitre(this);  
 }  
 }  
 }  
  
 *// \*\*\*\*\*\* ACCESSEURS \*\*\*\*\*\* //* public int getAge(){ return(this.age); }  
 public void setAge(int number){  
 this.age = number;  
 }  
 public String getNom(){  
 return(this.nom);  
 }  
 public void setNom(String name){  
 this.nom = name;  
 }  
 public ArrayList<Nac> getAnimal(){  
 return(this.animaux);  
 }  
 public void setAnimal(ArrayList<Nac> nc){  
 if(nc != null) {  
 this.animaux = nc;  
 for (Nac puppy : this.animaux) {  
 if (!puppy.getMaitre().equals(this)) {  
 puppy.setMaitre(this);  
 }  
 }  
 }  
 }  
  
 *// \*\*\*\*\*\* METHODES \*\*\*\*\*\* //* @Override  
 public String toString(){  
 return "Je suis : " + this.nom + " et j'ai " + this.age + " ans.";  
 }  
  
 @Override  
 public boolean equals(Object obj){  
 boolean retour = false;  
 if (obj!= null && (obj.getClass().equals(this.getClass()))){  
 if (obj instanceof Maitre){  
 Maitre maitre = (Maitre)obj;  
 if(checkAnimalExistance(maitre) && checkAnimalExistance(this)){  
 retour = this.nom.equals(maitre.getNom()) && this.age == maitre.getAge();  
 }  
 else{  
 if(checkAnimalExistance(maitre) || checkAnimalExistance(this)){  
 retour = false;  
 }  
 else{  
 retour = this.nom.equals(maitre.getNom()) &&  
 this.age == maitre.getAge() && this.animaux.equals(maitre.getAnimal());  
 }  
 }  
 }  
  
 }  
 return(retour);  
 }  
  
 private boolean checkAnimalExistance(Maitre maitre) {  
 return maitre.getAnimal().size() == 0;  
 }  
  
 *// Cette fonction ajoute un animal à la liste déjà existante* public void addNac(Nac puppy) {  
 if(! this.isNacInAnimals(puppy)) {  
 this.animaux.add(puppy);  
 }  
 }  
  
 public boolean isNacInAnimals(Nac puppy){  
 for (Nac mypuppy:this.animaux) {  
 if(puppy.equals(mypuppy)){  
 return(true);  
 }  
 }  
 return(false);  
 }  
  
 *// Préciser qu'on doit modifier la méthode promener dans le rapport* public String promener() {  
 String promenade = "Pour cette promenade, " + this.nom + " vous présente :";  
 for (Nac puppy:this.animaux) {  
 promenade = promenade + '\n' + '-' + puppy.presentation();  
 }  
 return(promenade);  
 }  
}

package com.Dauphine.Agility;  
  
  
import java.util.ArrayList;  
  
*// La classe Nac représente un Nouvel Animal de Compagnie*public class Nac{  
  
 *// \*\*\*\*\*\* ATTRIBUTS \*\*\*\*\*\* //* private String nom;  
 private int nbrHeureSommeil;  
 private Maitre maitre; *// Un animal possède 0 ou 1 maître  
  
  
 // \*\*\*\*\*\* CONSTRUCTEURS \*\*\*\*\*\* //  
  
 // Constructeur vide* public Nac(){  
 this.nom = "animal sans nom";  
 this.nbrHeureSommeil = 0;  
 this.maitre = null;  
 }  
 *// Ce constructeur permet de créer un animal ne possédant pas de maître à sa création* public Nac(String nom, int nbrHeureSommeil){  
 this.nom = nom;  
 this.nbrHeureSommeil = nbrHeureSommeil;  
 this.maitre = null;  
 }  
  
 *// Ce constructeur permet de créer un animal possédant un maître à sa création* public Nac(String nom, int nbrHeureSommeil, Maitre maitre){  
 this.nom = nom;  
 this.nbrHeureSommeil = nbrHeureSommeil;  
 this.maitre = maitre;  
 maitre.addNac(this);  
 }  
  
  
 *// \*\*\*\*\*\* ACCESSEURS \*\*\*\*\*\* //* public String getNom() {  
 return(this.nom);  
 }  
 public void setNom(String nom){  
 this.nom = nom;  
 }  
 public int getNbrDodo(){  
 return(this.nbrHeureSommeil);  
 }  
 public void setNbrDodo(int nbrHeureSommeil){  
 this.nbrHeureSommeil = nbrHeureSommeil;  
 }  
 public Maitre getMaitre(){  
 return(this.maitre);  
 }  
 public void setMaitre(Maitre maitre){  
 this.maitre = maitre;  
 maitre.addNac(this);  
 }  
  
 *// \*\*\*\*\*\* METHODES \*\*\*\*\*\* //* @Override  
 public String toString(){  
 String retour = "Je suis : " + this.nom + " et j'ai besoin de " + this.nbrHeureSommeil + " heures de sommeil.";  
 if(this.maitre != null){  
 retour = retour + "\n Mon maître s'appelle : " + this.maitre.getNom();  
 }  
 return(retour);  
 }  
  
 @Override  
 public boolean equals(Object obj){  
 boolean retour = false;  
 if (obj!= null && (obj.getClass().equals(this.getClass()))){  
 if (obj instanceof Nac){  
 Nac puppy = (Nac)obj;  
 if(puppy.getMaitre() == null && this.maitre == null){  
 retour = this.nom.equals(puppy.getNom()) && this.nbrHeureSommeil == puppy.getNbrDodo();  
 }  
 else {  
 if(puppy.getMaitre() == null || this.maitre == null){  
 retour = false;  
 }  
 else {  
 retour = this.nom.equals(puppy.getNom()) &&  
 this.nbrHeureSommeil == puppy.getNbrDodo() && this.maitre.equals(puppy.getMaitre());  
 }  
 }  
 }  
 }  
 return(retour);  
 }  
  
 public String presentation(){  
 return "Je m'appelle " + this.nom + " et je dors " + this.nbrHeureSommeil + " heures.";  
 }  
}

package com.Dauphine.Agility;  
  
import static org.junit.Assert.\*;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
import java.util.ArrayList;  
  
public class MaitreTest {  
  
 public MaitreTest() {  
 }  
 */\*\*  
 \* Méthode appelée avant chaque appel de méthode de test.  
 \*/* @Before  
 public void setUp() *// throws java.lang.Exception* {  
 }  
  
 */\*\*  
 \* Supprime les engagements  
 \* Méthode appelée après chaque appel de méthode de test.  
 \*/* @After  
 public void tearDown() *// throws java.lang.Exception* {  
 *//Libérez ici les ressources engagées par setUp()* }  
  
 @Test  
 public void constructeurVideTest() {  
 Maitre m1 = new Maitre();  
 *assertEquals*(m1.getNom(), "Laura");  
 *assertEquals*(m1.getAge(), 0);  
 *assertEquals*(m1.getAnimal(), new ArrayList<Nac>());  
 }  
  
 @Test  
 public void constructeur2argumentsTest() {  
 Maitre m1 = new Maitre(10, "Laye");  
 *assertEquals*(m1.getNom(), "Laye");  
 *assertEquals*(m1.getAge(), 10);  
 *assertEquals*(m1.getAnimal(), new ArrayList<Nac>());  
 }  
  
 @Test  
 public void constructeur3argumentsTest() {  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
 ArrayList<Nac> myNacs = new ArrayList<Nac>();  
 myNacs.add(n1);  
 myNacs.add(n2);  
 myNacs.add(n3);  
 Maitre m1 = new Maitre(10, "Laye", myNacs);  
 *assertEquals*(m1.getNom(), "Laye");  
 *assertEquals*(m1.getAge(), 10);  
 *assertEquals*(m1.getAnimal(), myNacs);  
 for (Nac puppy:m1.getAnimal()) {  
 *assertEquals*(m1, puppy.getMaitre());  
 }  
 Maitre m2 = new Maitre(10, "Simona");  
 Nac n4 = new Nac("Hamtara", 11, m1);  
 ArrayList<Nac> myNacsZam = new ArrayList<Nac>();  
 myNacsZam.add(n4);  
 Maitre m3 = new Maitre(60,"Zam", myNacsZam);  
 *assertEquals*(m3.getNom(), "Zam");  
 *assertEquals*(m3.getAge(), 60);  
 *assertEquals*(myNacsZam, m3.getAnimal());  
 for (Nac puppy:m3.getAnimal()) {  
 *assertEquals*(m3, puppy.getMaitre());  
 }  
 }  
  
 @Test  
 public void getNomTest() {  
 Maitre m1 = new Maitre(10, "Laye");  
 *assertEquals*("Laye", m1.getNom());  
 }  
  
 @Test  
 public void getAgeTest() {  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
 ArrayList<Nac> myNacs = new ArrayList<Nac>();  
 myNacs.add(n1);  
 myNacs.add(n2);  
 myNacs.add(n3);  
 Maitre m1 = new Maitre(10, "Laye",myNacs);  
 *assertEquals*(10, m1.getAge());  
 }  
  
 @Test  
 public void getAnimals() {  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
 ArrayList<Nac> myNacs = new ArrayList<Nac>();  
 myNacs.add(n1);  
 myNacs.add(n2);  
 myNacs.add(n3);  
 Maitre m1 = new Maitre(10, "Laye", myNacs);  
 *assertEquals*(myNacs, m1.getAnimal());  
 }  
  
 @Test  
 public void setNomTest() {  
 Maitre m1 = new Maitre(10, "Laye");  
 m1.setNom("Bijou");  
 *assertEquals*("Bijou", m1.getNom());  
 }  
  
 @Test  
 public void setAge() {  
 Maitre m1 = new Maitre(10, "Laye");  
 m1.setAge(20);  
 *assertEquals*(20, m1.getAge());  
 }  
  
 @Test  
 public void setAnimalTest() {  
 Maitre m2 = new Maitre(10, "Laye");  
 Nac n1 = new Nac("Hamtaro", 12,m2);  
 Nac n2 = new Nac("Hamtara", 11,m2);  
 Nac n3 = new Nac("Hamtaru", 10,m2);  
  
 Nac n5 = new Nac("amtaro", 2,m2);  
 Nac n6 = new Nac("amtara", 1,m2);  
 Nac n7 = new Nac("amtaru", 0,m2);  
  
 ArrayList<Nac> myNacs = new ArrayList<Nac>();  
 ArrayList<Nac> myNacsNew = new ArrayList<Nac>();  
 myNacs.add(n1);  
 myNacs.add(n2);  
 myNacs.add(n3);  
 myNacsNew.add(n5);  
 myNacsNew.add(n6);  
 myNacsNew.add(n7);  
 Maitre m1 = new Maitre(10, "Laye", myNacs);  
  
 m1.setAnimal(myNacsNew);  
 *assertEquals*(myNacsNew, m1.getAnimal());  
 }  
  
 @Test  
 public void toStringTest(){  
 Maitre maitre = new Maitre(8, "Laura");  
 *assertEquals*(maitre.toString(), "Je suis : Laura et j'ai 8 ans.");  
 }  
  
 @Test  
 public void equalsTest(){  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
  
 Nac n5 = new Nac("amtaro", 2);  
 Nac n6 = new Nac("amtara", 1);  
 Nac n7 = new Nac("amtaru", 0);  
  
 ArrayList<Nac> myNacs1 = new ArrayList<Nac>();  
 ArrayList<Nac> myNacs2 = new ArrayList<Nac>();  
 myNacs1.add(n1);  
 myNacs1.add(n2);  
 myNacs1.add(n3);  
 myNacs2.add(n5);  
 myNacs2.add(n6);  
 myNacs2.add(n7);  
  
 Maitre maitre0 = new Maitre(8, "Laura");  
 Maitre maitre1 = new Maitre(10, "Laurent",myNacs2);  
 Maitre maitre2 = new Maitre(10, "Laurent",myNacs1);  
 Maitre maitre3 = new Maitre(10, "Laurent",myNacs1);  
 Maitre maitre4 = new Maitre(8, "Phil");  
  
 *assertEquals*(true, maitre3.equals(maitre2));  
 *assertEquals*(false, maitre0.equals(maitre1));  
 *assertEquals*(false, maitre1.equals(maitre2));  
 *assertEquals*(false, maitre0.equals(maitre4));  
 }  
  
 @Test  
 public void promenadeTest()  
 {  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
 ArrayList<Nac> myNacs1 = new ArrayList<Nac>();  
 myNacs1.add(n1);  
 myNacs1.add(n2);  
 myNacs1.add(n3);  
 Maitre maitre2 = new Maitre(10, "Laurent",myNacs1);  
 *assertEquals*("Pour cette promenade, Laurent vous présente :\n" +  
 "-Je m'appelle Hamtaro et je dors 12 heures.\n" +  
 "-Je m'appelle Hamtara et je dors 11 heures.\n" +  
 "-Je m'appelle Hamtaru et je dors 10 heures.", maitre2.promener());  
 }  
  
 @Test  
 public void addNacTest(){  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
  
 Nac n4 = new Nac("NewAnimal", 4);  
  
 ArrayList<Nac> myNacs = new ArrayList<Nac>();  
 ArrayList<Nac> myNacsNew = new ArrayList<Nac>();  
 ArrayList<Nac> myNacsOne = new ArrayList<Nac>();  
 myNacs.add(n1);  
 myNacs.add(n2);  
 myNacs.add(n3);  
 myNacsNew = myNacs;  
 myNacsNew.add(n4);  
 myNacsOne.add(n4);  
 Maitre maitreWithoutAnimal = new Maitre(10, "Laurent");  
 maitreWithoutAnimal.addNac(n4);  
 Maitre maitreWithAnimal = new Maitre(8, "Laura", myNacs);  
 maitreWithAnimal.addNac(n4);  
 *assertEquals*(myNacsNew, maitreWithAnimal.getAnimal());  
 *assertEquals*(myNacsOne, maitreWithoutAnimal.getAnimal());  
 for (Nac puppy:maitreWithAnimal.getAnimal()) {  
 *assertEquals*(maitreWithAnimal, puppy.getMaitre());  
 }  
 }  
  
 @Test  
 public void isNacInAnimalsTest(){  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 11);  
 Nac n3 = new Nac("Hamtaru", 10);  
  
 Nac n4 = new Nac("NewAnimal", 4);  
  
 ArrayList<Nac> myNacs = new ArrayList<Nac>();  
 myNacs.add(n1);  
 myNacs.add(n2);  
 myNacs.add(n3);  
 Maitre maitreWithAnimal = new Maitre(8, "Laura", myNacs);  
 *assertEquals*(true, maitreWithAnimal.isNacInAnimals(n3));  
 *assertEquals*(false, maitreWithAnimal.isNacInAnimals(n4));  
 }  
}

package com.Dauphine.Agility;  
  
import static org.junit.Assert.\*;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
  
*/\*\*  
 \* Classe-test NacTest.  
 \* Les classes-test sont documentées ici :  
 \* http://junit.sourceforge.net/javadoc/junit/framework/TestCase.html  
 \* et sont basées sur le document Š 2002 Robert A. Ballance intitulé  
 \* "JUnit: Unit Testing Framework".  
 \*/*public class NacTest  
{  
 *// Définissez ici les variables d'instance nécessaires à vos engagements;  
 // Vous pouvez également les saisir automatiquement du présentoir  
  
 /\*\*  
 \* Constructeur de la classe-test NacTest  
 \*/* public NacTest()  
 {  
 }  
  
 */\*\*  
 \* Méthode appelée avant chaque appel de méthode de test.  
 \*/* @Before  
 public void setUp() *// throws java.lang.Exception* {  
 }  
  
 */\*\*  
 \* Supprime les engagements  
 \* Méthode appelée après chaque appel de méthode de test.  
 \*/* @After  
 public void tearDown() *// throws java.lang.Exception* {  
 *//Libérez ici les ressources engagées par setUp()* }  
  
   
  
 @Test  
 public void constructeurVideTest(){  
 Nac nac1 = new Nac();  
 *assertEquals*(nac1.getNom(),"animal sans nom");  
 *assertEquals*(nac1.getNbrDodo(), 0);  
 *assertEquals*(nac1.getMaitre(), null);  
 }  
  
 @Test  
 public void constructeur2argumentsTest(){  
 Nac nac2 = new Nac("Hamtaro",12);  
 *assertEquals*(nac2.getNom(),"Hamtaro");  
 *assertEquals*(nac2.getNbrDodo(), 12);  
 *assertEquals*(nac2.getMaitre(), null);  
 }  
  
 @Test  
 public void constructeur3argumentsTest(){  
 Maitre maitre3 = new Maitre(8, "Laura");  
 Nac nac3 = new Nac("Hamtaro",12, maitre3);  
 *assertEquals*(nac3.getNom(),"Hamtaro");  
 *assertEquals*(nac3.getNbrDodo(), 12);  
 *assertEquals*(nac3.getMaitre(), maitre3);  
 }  
  
 @Test  
 public void getNomTest(){  
 Nac n = new Nac("Hamtaro", 12);  
 *assertEquals*("Hamtaro", n.getNom());  
 }  
  
 @Test  
 public void getNbrHeureSommeilTest(){  
 Nac n = new Nac("Hamtaro", 12);  
 *assertEquals*(12, n.getNbrDodo());  
 }  
  
 @Test  
 public void getMaitreTest(){  
 Maitre maitre = new Maitre(8, "Laura");  
 Nac n = new Nac("Hamtaro", 12, maitre);  
 *assertEquals*(maitre, n.getMaitre());  
 }  
  
 @Test  
 public void setNomTest(){  
 Nac n = new Nac("Hamtaro", 12);  
 n.setNom("Bijou");  
 *assertEquals*("Bijou", n.getNom());  
 }  
  
 @Test  
 public void setNbrHeureSommeilTest(){  
 Nac n = new Nac("Hamtaro", 12);  
 n.setNbrDodo(20);  
 *assertEquals*(20, n.getNbrDodo());  
 }  
  
 @Test  
 public void setMaitreTest(){  
 Maitre maitre = new Maitre(8, "Laura");  
 Maitre maitre2 = new Maitre(8, "Zam");  
 Nac n = new Nac("Hamtaro", 12, maitre);  
 n.setMaitre(maitre2);  
 *assertEquals*(maitre2, n.getMaitre());  
 }  
  
 @Test  
 public void equalsTest(){  
 Maitre maitre = new Maitre(8, "Laura");  
 Maitre maitre1 = new Maitre(10, "Laurent");  
 Nac n1 = new Nac("Hamtaro", 12, maitre);  
 Nac n2 = new Nac("Hamtaro", 12, maitre);  
 Nac n3 = new Nac("Hamtaro", 12, maitre1);  
 Nac n4 = new Nac("Hamtaro", 12);  
 Nac n5 = new Nac("Hamtaro", 12);  
 *assertEquals*(true, n4.equals(n5));  
 *assertEquals*(true, n1.equals(n2));  
 *assertEquals*(false, n3.equals(n2));  
 *assertEquals*(false, n2.equals(n4));  
 *assertEquals*(false, n4.equals(n2));  
 }  
  
 @Test  
 public void toStringTest(){  
 Maitre maitre = new Maitre(8, "Laura");  
 Nac n1 = new Nac("Hamtaro", 12);  
 Nac n2 = new Nac("Hamtara", 10, maitre);  
 *assertEquals*(n1.toString(), "Je suis : Hamtaro et j'ai besoin de 12 heures de sommeil.");  
 *assertEquals*(n2.toString(), "Je suis : Hamtara et j'ai besoin de 10 heures de sommeil.\n" +  
 " Mon maître s'appelle : Laura");  
  
 }  
 @Test  
 public void presentationTest()  
 {  
 Nac hamtaro = new Nac("Hamtaro", 12);  
 Maitre maitre = new Maitre(8, "Laura");  
 *assertEquals*("Je m'appelle Hamtaro et je dors 12 heures.", hamtaro.presentation());  
 }  
  
  
  
  
}

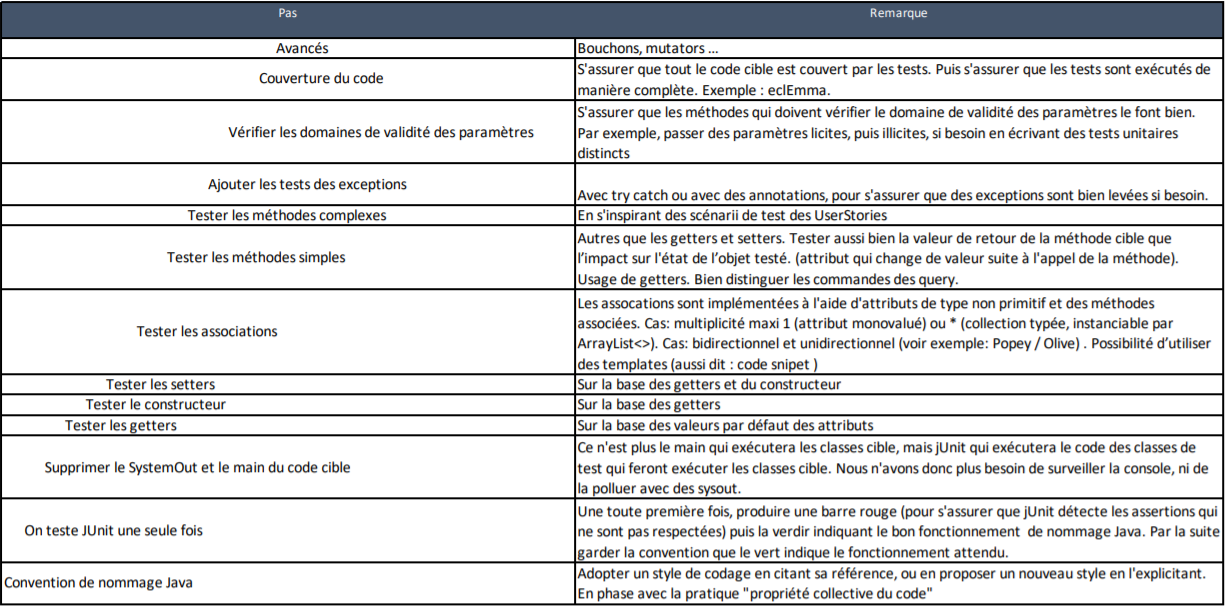


Figure 21: Escalade de tests

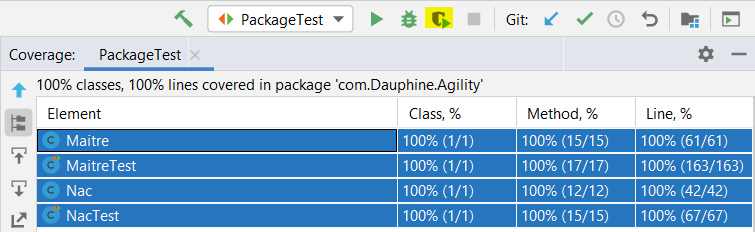


Figure 22: Pourcentage de couverture du code

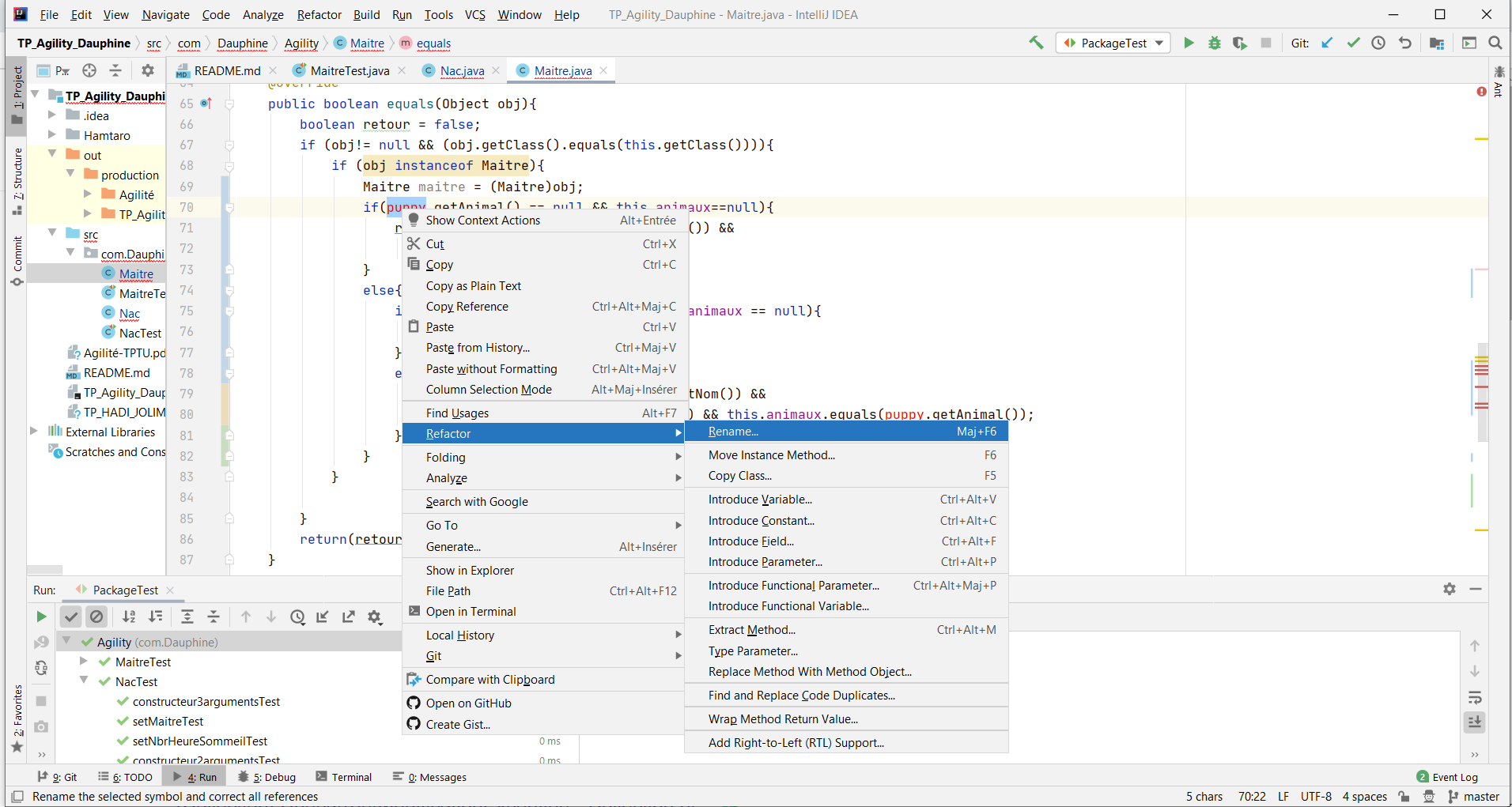


Figure 23: Refactoring (renaming)

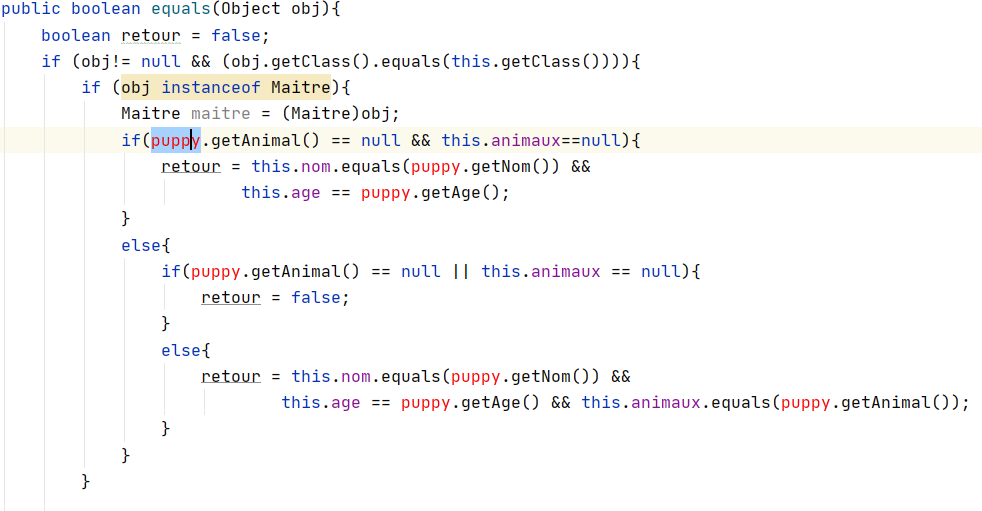


Figure 24: Avant refactoring

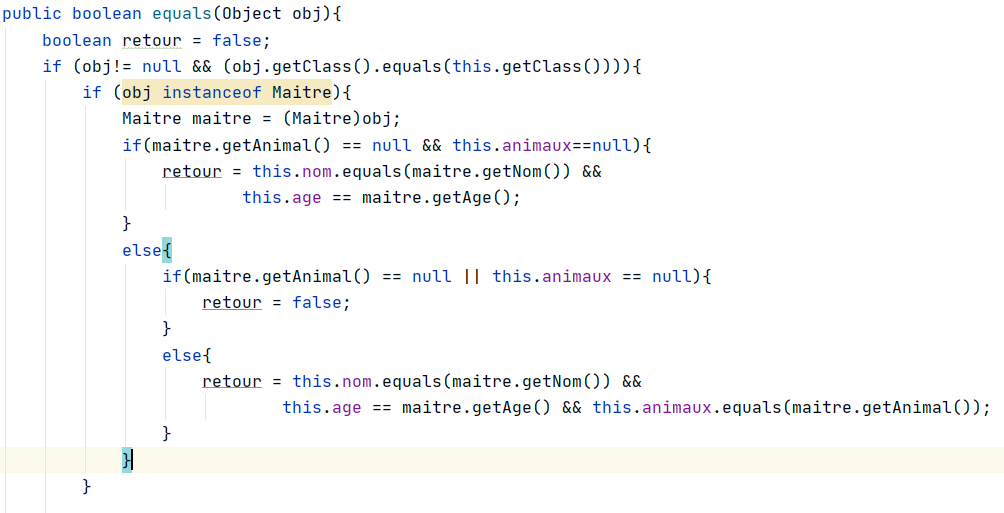


Figure 25: Après refactoring

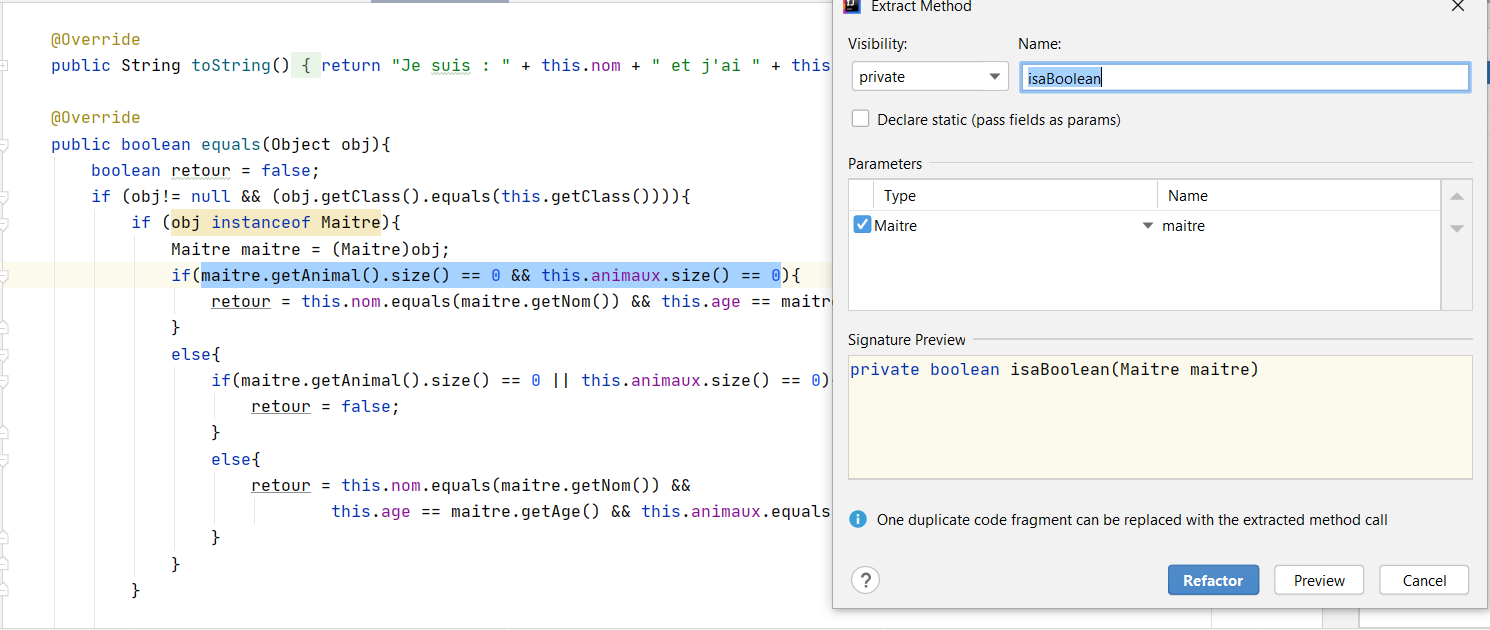
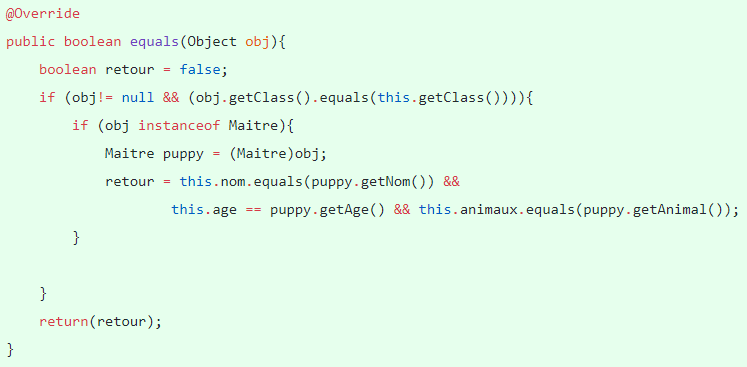
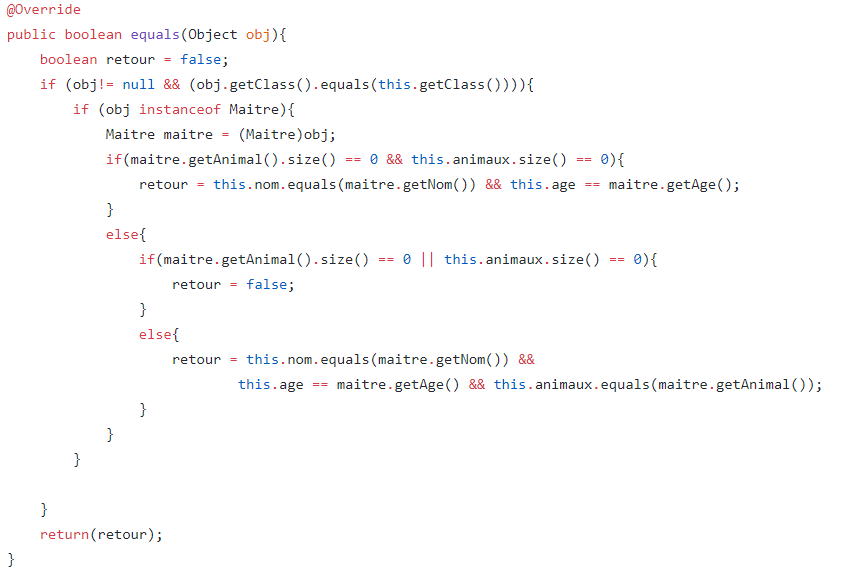
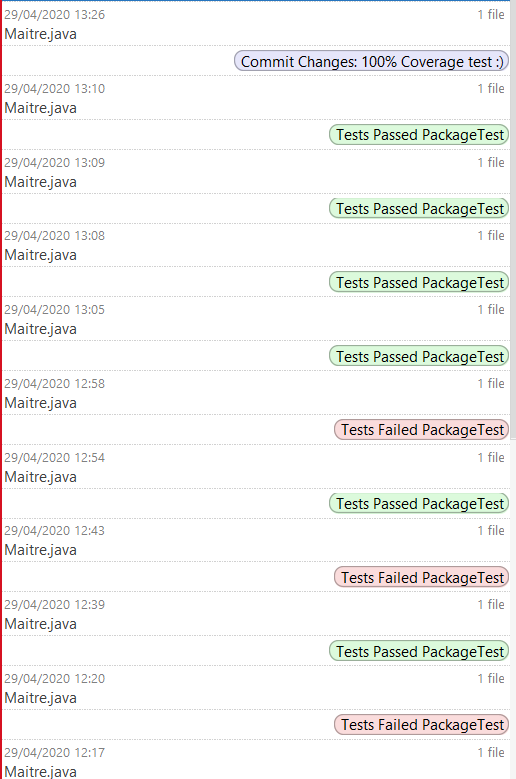


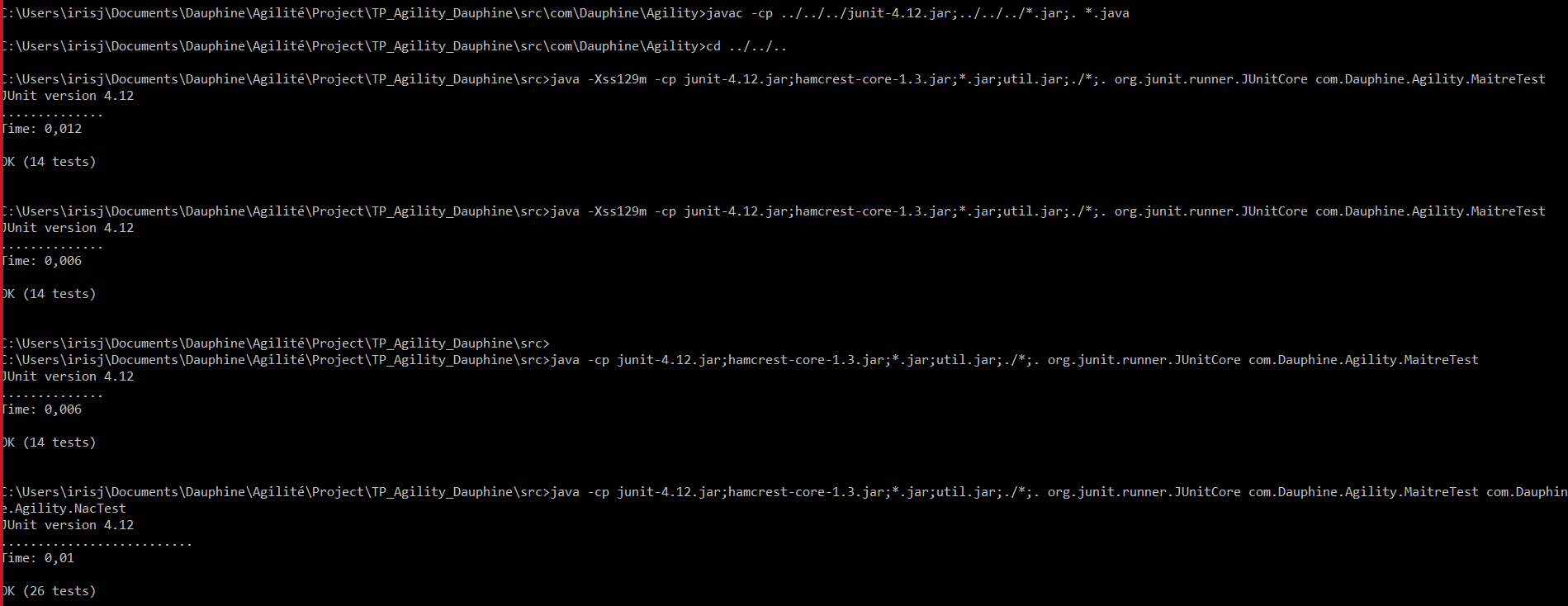
Figure 26: Extract method











Loi de Murphy : « Si cela peut mal se passer, cela arrivera »