Introduction

Ce rapport détaille le processus complet de création, d'instanciation et de test d'une application Java simple à l'aide de l'environnement de développement BlueJ. Il présente également l'interaction entre deux classes (`animal` et `Boost`).

Étapes de Développement

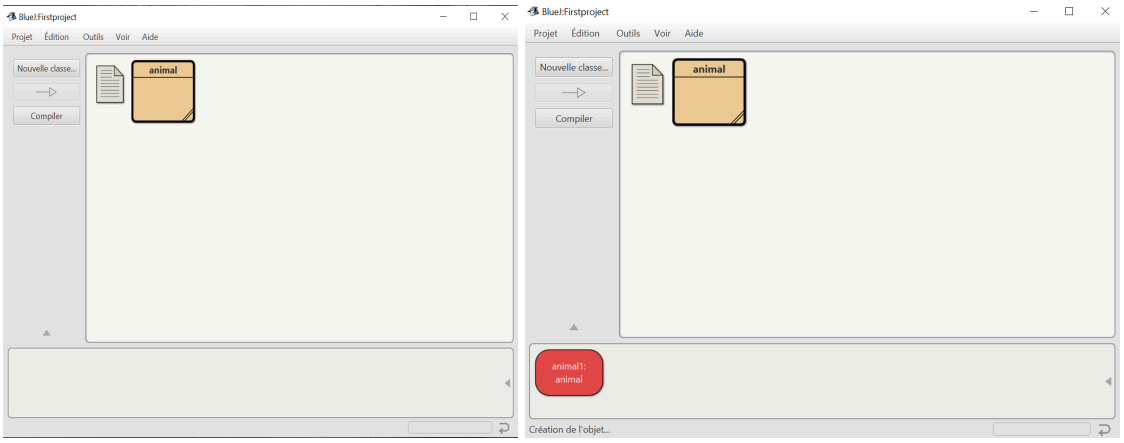
Création du Projet

Le projet nommé « FirstProject » a été initialement créé via BlueJ, un environnement graphique qui permet une gestion visuelle simple des projets Java.

Création de la Classe « Animal »

Une classe intitulée `animal` a été définie en premier lieu. Cette classe représente notre classe « fétiche », identifiée de manière unique dans le cadre du projet.

Après la création de la classe `animal`, elle a été compilée avec succès afin de s'assurer de l'absence d'erreurs syntaxiques. Suite à cela, un premier objet nommé `animal1` a été instancié, permettant ainsi de visualiser l'objet directement dans l'environnement BlueJ.



Définition des Attributs et Méthodes

La classe `animal` a été enrichie par trois attributs principaux :

- `nom` (String)
- `age` (int)

`energie` (int, initialement 100)

```java
// variables d'instance - remplacez l'exemple qui suit par le vôtre
private String nom;
private int age ;
private int energie = 100;

/**
 * Constructeur d'objets de classe animal
 */
public animal(String n,int a)
{
    nom = n;
    age = a;
}
```

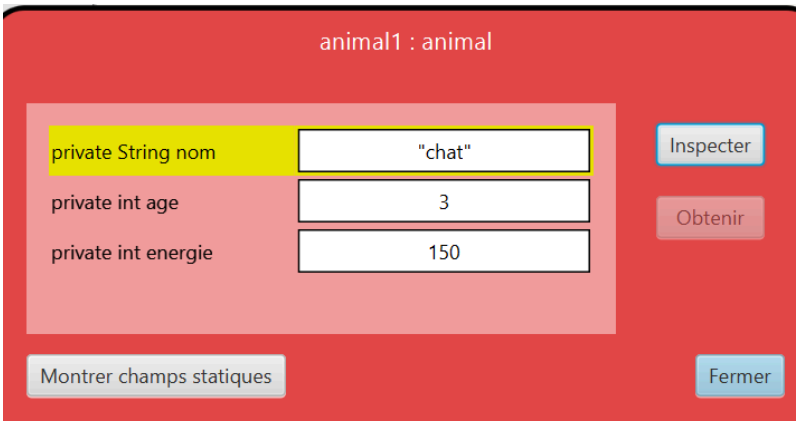
Des méthodes spécifiques ont été ajoutées pour interagir avec ces attributs :

- `afficherEnergie()` : affiche l'énergie actuelle.
- `nourrir(Boost boost)` : augmente l'énergie selon un objet `Boost`.
- `attaquer(animal an)` : diminue l'énergie d'un autre animal.
- `getEnergie()` : accesseur retournant l'énergie courante.

```java
public void afficherEnergie() {
  System.out.println("Energie =" + this.energie);
}
public void nourrir(Boost boost){
    energie += boost.Boost_energie();
}
public void attaquer(animal an){
    an.energie-=50;
}
public int getEnergie(){
    return this.energie;
}
```

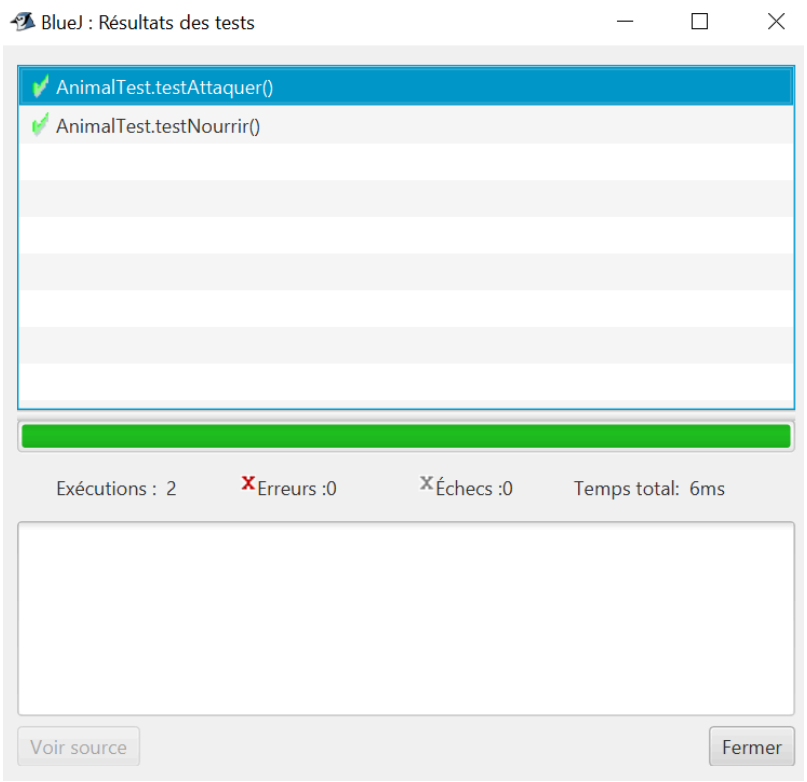
Exécution Interactive

Une instanciation interactive de la méthode `nourrir` a permis d'observer directement son effet sur l'état interne de l'objet. Suite à l'exécution, l'énergie initiale de l'animal (100) a été modifiée (150), confirmant ainsi le fonctionnement correct de cette méthode.



Tests Unitaires et Validation

Des tests unitaires (`testAttaquer()` et `testNourrir()`) ont été créés à l'aide de l'environnement de test de BlueJ. L'exécution réussie de ces tests, signalée par la barre verte, a validé le comportement attendu des méthodes implémentées.



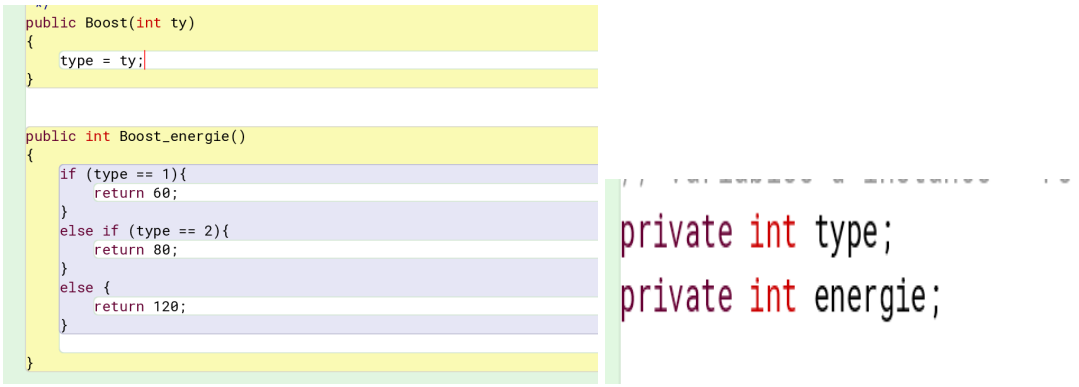
Ajout d'une Classe Associée « Boost »

La seconde classe `Boost` a été créée et associée de façon unidirectionnelle à la classe `animal`, avec une multiplicité de 0..1 à 0..1. Cette classe contient un attribut unique :

- `type` (int)

Cette classe possède une méthode clé :

- `Boost_energie()` : retourne la quantité d'énergie ajoutée selon le type du boost (60, 80 ou 120).



La classe `animal` collabore avec la classe `Boost` via la méthode `nourrir(Boost boost)`, qui utilise directement la valeur retournée par la méthode `Boost_energie()`.

Utilisation des Fixtures dans les Tests Unitaires

Pour garantir la reproductibilité et la cohérence des tests, une fixture (`setUp()`) a été définie, instanciant systématiquement des objets avant chaque test :

- Deux objets de type `animal` : `animal1` et `animal2`
- Un objet de type `Boost` : `boost1`

Les tests utilisant cette fixture ont tous abouti à un résultat positif, confirmé par l'apparition systématique d'une barre verte.

```java
@BeforeEach
public void setUp() {
    animal1 = new animal("papicha", 3);
    boost1 = new Boost(2);
    animal2 = new animal("son", 2);
}

@AfterEach
public void tearDown() {
    animal1 = null;
    boost1 = null;
    animal2 = null;
}

@Test
public void testNourrir() {
    animal1.nourrir(boost1);
    assertEquals(180, animal1.getEnergie());
}

@Test
public void testAttaquer() {
    animal1.attaquer(animal2);
    assertEquals(50, animal2.getEnergie());
}
```