Fondamentaux du développeur

TP - Maven, JavaDoc, et Tests unitaires JUnit

Philippe Roussille

3iL INGENIEURS
</les sciences informatiques>

1 Introduction

Dans ce TP, vous allez :

- Installer et utiliser **Maven** dans votre machine virtuelle Linux.
- Créer un petit projet structuré : `tp-canards`.
- Compléter un code Java fourni contenant
  - de la **JavaDoc à écrire**,
  - des **bouts manquants**,
  - et quelques **erreurs** à corriger.
- Écrire et exécuter des **tests unitaires JUnit**.
- Générer enfin la **documentation JavaDoc**.

2 Partie 1 - Installer Maven dans la VM

Maven n'est **pas installé de base** sur votre machine virtuelle.

2.1 Étape 1 : Vérifier la version de Java

```
java -version
```

Vous devez obtenir une version **Java 11** ou supérieure.

2.2 Étape 2 : Installation

```
sudo apt update
sudo apt install maven
```

2.3 Étape 3 : Vérification

```
mvn -version
```

Vous devez voir apparaître :

```
Apache Maven 3.x.x
Java version: 11.x.x
```

3 Partie 2 - Création du projet Maven

Dans votre dossier de TP :

```
mvn archetype:generate -DgroupId=com.example \
  -DartifactId=tp-canards \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```

Cette commande génère :

```
tp-canards/
 ├── pom.xml
 ├── src/main/java/com/example/App.java
 └── src/test/java/com/example/AppTest.java
```

Supprimez `App.java` et `AppTest.java` : nous allons les remplacer.

4 Partie 3 - Mise en place du code du TP

Créez le package `duck` :

```
src/main/java/duck/
src/test/java/duck/
```

Placez-y les fichiers suivants.

5 Fichier 1 - `Canard.java`

Attention : ce fichier comporte volontairement des erreurs étranges. À vous de les détecter avant de tester !

```java
package duck;

public class Canard {

    private String couleur;
    private int flottabilite;
    private boolean humeur;

    /**
     * Crée un canard (TODO: JavaDoc complète)
     */
    public Canard(String couleur, int flottabilite, boolean humeur) {
```

```
        if (flottabilite < 0)
            throw new IllegalArgumentException("Flottabilité négative.");

        // ERREUR : inversion étrange de paramètres
        this.couleur = couleur.toUpperCase(); // devrait peut-être
        ↪  conserver la casse ?
        this.humeur = flottabilite > 10;
        this.flottabilite = flottabilite;
    }

    /** TODO : documenter */
    public boolean flotte() {
        return flottabilite > 0;
    }

    /** TODO : documenter */
    public String couine() {
        if (!humeur) return "Coin coin !";
        return "... (silence bougon)";
    }

    /** TODO : documenter */
    public void changerHumeur() {
        humeur = !humeur;
    }

    public String getCouleur() { return couleur; }
    public int getFlottabilite() { return flottabilite; }
    public boolean getHumeur() { return humeur; }
}
```

6 Travail demandé sur `Canard.java`

1. **Compléter la JavaDoc** de la classe et de chaque méthode.
2. **Identifier les erreurs anormales**.
3. Les **corriger proprement**.
4. Ajouter une précondition optionnelle : la couleur **ne doit pas être `null`**.

7 Fichier 2 - `ParcCanards.java`

```
package duck;

import java.util.ArrayList;
import java.util.List;

public class ParcCanards {

    private List<Canard> canards = new ArrayList<>();
```

```java
    /** TODO : documenter */
    public void ajouter(Canard c) {
        canards.add(c);
    }

    /** TODO : documenter */
    public int compterCanardsFlottants() {
        int n = 0;
        for (Canard c : canards) {
            if (c.flotte()) n++;
        }
        return n;
    }

    /** TODO : documenter */
    public boolean estVide() {
        return canards.size() == 0;
    }

    /** TODO : documenter */
    public List<Canard> getCanards() {
        return canards;
    }
}
```

8 Partie 4 - Fichier `pom.xml`

Voici le `pom.xml` minimal à utiliser :

```xml
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>tp-canards</artifactId>
  <version>1.0</version>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
      <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter</artifactId>
          <version>5.9.3</version>
          <scope>test</scope>
      </dependency>
  </dependencies>
```

```
</project>
```

9 Partie 5 - Tests unitaires JUnit (fournis et à compléter)

Créez `src/test/java/duck/CanardTest.java`.

```java
package duck;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class CanardTest {

    @Test
    public void testConstructeurValide() {
        Canard c = new Canard("jaune", 5, true);
        assertEquals("jaune", c.getCouleur());
        assertEquals(5, c.getFlottabilite());
        assertTrue(c.getHumeur());
    }

    @Test
    public void testFlotte() {
        Canard c = new Canard("rouge", 1, false);
        assertTrue(c.flotte());
        Canard c2 = new Canard("bleu", 0, true);
        assertFalse(c2.flotte());
    }

    @Test
    public void testCouine() {
        Canard c = new Canard("vert", 5, true);
        assertEquals("Coin coin !", c.couine());
        c.changerHumeur();
        assertEquals("... (silence bougon)", c.couine());
    }
}
```

10 Tests pour ParcCanards

Créez `src/test/java/duck/ParcCanardsTest.java`.

```java
package duck;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class ParcCanardsTest {
```

```java
    @Test
    public void testAjout() {
        ParcCanards p = new ParcCanards();
        assertTrue(p.estVide());
        p.ajouter(new Canard("vert", 3, true));
        assertFalse(p.estVide());
        assertEquals(1, p.getCanards().size());
    }

    @Test
    public void testFlottants() {
        ParcCanards p = new ParcCanards();
        p.ajouter(new Canard("jaune", 0, true));
        p.ajouter(new Canard("rouge", 5, false));
        assertEquals(1, p.compterCanardsFlottants());
    }
}
```

11 Partie 6 - Compilation & exécution

11.1 Compiler

```
mvn compile
```

11.2 Lancer les tests

```
mvn test
```

Vous devez obtenir :

```
Tests run: 5, Failures: 0, Errors: 0
```

12 Partie 7 - Générer la JavaDoc

```
mvn javadoc:javadoc
```

Puis ouvrir dans votre VM :

```
target/site/apidocs/index.html
```

Vérifier que :

— vos commentaires apparaissent,
— aucune balise n'est cassée,
— les erreurs corrigées sont cohérentes avec la doc.

13 Questions ouvertes (à rendre)

1. Quelles étaient les erreurs logiques dans `Canard.java` ?
2. Pourquoi les tests fournis permettent-ils de les détecter ?
3. Proposer un test supplémentaire qui aurait pu détecter un autre bug.