L03F\_TESTS UNITAIRES JUNIT

SOLUTIONNAIRE

Table des matières

[1 Ouverture du projet L03F\_Joueur 2](#_Toc148183684)

[1.1 Exploration du projet 2](#_Toc148183685)

[2 Tests JUnit 3](#_Toc148183686)

[2.1 Mise en place 3](#_Toc148183687)

[2.2 Annotations avant/après les tests 3](#_Toc148183688)

[2.3 Méthodes de tests 3](#_Toc148183689)

[2.3.1 Créez les tests suivants: 3](#_Toc148183690)

[3 Questions 5](#_Toc148183691)

[4 SOLUTION 5](#_Toc148183692)

# Ouverture du projet L03F\_Joueur

1. Récupérez sur Léa le fichier **C34\_L03F\_Joueur\_Etudiants.zip**.
2. Extrayez le contenu dans votre répertoire de projets pour le cours.
3. Ouvrez le projet dans IntelliJ IDEA.

## Exploration du projet

Le projet représente un personnage dans un jeu. Les champs de la classe **Joueur** sont les suivants:

|  |
| --- |
| private String alias; *// Nom d'affichage du joueur* *private int* credits; *// Monnaie du jeu, peut améliorer l'arme* *private int* gemmes; *// Autre monnaie du jeu, peut améliorer l'armure* *private* String nomArme; *// Nom de l'arme du joueur* *private int* niveauArme; *// Niveau actuel de l'arme du joueur* *private* String nomArmure; *// Nom de l'armure du joueur* *private int* niveauArmure; *// Niveau actuel de l'armure du joueur* |

Il y a des méthodes d’accès et de mutation pour tous les champs.

Les mécaniques explorées dans le projet sont l’amélioration du niveau de l’arme et de l’armure.

Une **arme** peut être améliorée d’un niveau avec des **crédits** selon la formule suivante:

* Coût en crédit = 0.4 \* (niveau actuel de l’arme)2

Une **armure** peut être améliorée d’un niveau avec des **gemmes** selon la formule suivante:

* Coût en gemmes = 0.3 \* (niveau actuel de l’armure)2

4 méthodes sont mises en place pour gérer les améliorations:

**coutAmeliorerArme**: reçoit un niveau d’arme et retourne le coût d’amélioration du prochain niveau en crédits

**coutAmeliorerArmure**: reçoit un niveau d’armure et retourne le coût d’amélioration du prochain niveau en gemmes

**ameliorerArme**:

* Sans paramètre
* Vérifie si le joueur a assez de crédits pour améliorer l’arme au prochain niveau
* Si oui, augmente le niveau de l’arme de 1, déduits le coût des crédits et retourne **true**
* Si non, aucun changement n’est fait et retourne **false**

**ameliorerArmure**:

* Sans paramètre
* Vérifie si le joueur a assez de gemmes pour améliorer l’armure au prochain niveau
* Si oui, augmente le niveau de l’armure de 1, déduits le coût des gemmes et retourne **true**
* Si non, aucun changement n’est fait et retourne **false**

Dans ce laboratoire, on créé des tests pour valider les comportements attendus de ces méthodes

# Tests JUnit

## Mise en place

1. Dans le projet, ajoutez les **librairies** (dépendances) requise pour utiliser **JUnit 5.8.1**.
2. À la racine du projet au même niveau que src, créez un répertoire **test**
3. Marquez le répertoire test comme racine source de tests
4. Dans le répertoire **test**, créez une classe **JoueurTest**
5. Dans la classe **JoueurTest**, ajoutez les imports requis pour:
   1. Utiliser l’annotation **@Test**
   2. Utiliser les annotations **@BeforeEach**, **@AfterEach**, **@BeforeALL**, **@BeforeEach**
   3. Utiliser les méthodes **assertEquals**
   4. Utiliser les méthodes de la classe **Joueur**
6. Dans la classe **JoueurTest**, déclarez un champ qui représentera un objet Joueur, mais sans créer l’objet (pas de new)

## Annotations avant/après les tests

1. Ajoutez des méthodes avec les annotations @BeforeAll, @AfterAll et @AfterEach, qui ne font qu’afficher un message de votre choix.
2. À la suite de l’annotation **@BeforeEach**, créez une méthode **initJoueur()** qui:

* Créé un nouvel objet Joueur en lui passant des valeurs pour tous les paramètres requis pour faire les tests.

## Méthodes de tests

Pour chacun des tests, vous devez utiliser l’objet Joueur qui est un champ et qui est initialisé avant chaque test (@BeforeEach). Si vous avez besoin de modifier une des valeurs initiales pour un test donné, utilisez les méthodes de mutation du joueur.

### Créez les tests suivants:

1. Test simple qui vérifie que la méthode **getCredits** retourne la bonne valeur
2. Test pour vérifier que **coutAmeliorerArmure** retourne le résultat attendu (essayer un niveau positif, niveau = 0 et un niveau négatif)
3. Tester la valeur de retour de **ameliorerArme** (**2** tests, un qui devrait retourner **true** et un qui devrait retourner **false**)
4. Testez les changements attendus des champs du joueur suite à une tentative d’amélioration de l’arme - **ameliorerArme** : (**3** tests, 1 qui vérifie que la bonne quantité de crédits est déduite, 1 qui vérifie que le niveau est bien amélioré de 1 et 1 qui confirme que les crédits n’ont pas changés s'il n’y en avait pas assez)

# Questions

1. On pourrait répéter tous ces tests pour l’armure. Quels autres tests pouvez-vous imaginer?

|  |
| --- |
| Plusieurs tests pourraient être imaginés :  Tous les getters, tous les setters, des valeurs de départ négatives, si il y a une limite à la longueur de l’alias etc. |

1. Dans les résultats, quelle est la différence entre **fail** et **error**?

|  |
| --- |
| Fail : un test n’a pas été exécuté ou a produit un résultat différent de celui attendu  Error : erreur à la compilation ou l’exécution de la classe de test |

# SOLUTION

Fichier de classe **JoueurTest**

|  |
| --- |
| import org.junit.jupiter.api.\*; import static org.junit.jupiter.api.Assertions.\*; import jeu.Joueur;  public class JoueurTest {  Joueur joueurTest;   @BeforeAll  public static void avantTousTests(){  System.*out*.println("=> DÉBUT DES TESTS <=");  }   @AfterEach  public void apresChaqueTest(){  System.*out*.println("=> FIN DU TEST <=");  }   @AfterAll  public static void apresTousTests(){  System.*out*.println("=> TOUS TESTS COMPLÉTÉS <=");  }   @BeforeEach  public void initJoueur(){  joueurTest = new Joueur("Skywalker42", 1000, 1000, "Épée", 40, "Plastron Vibranium", 60);  }   @Test *// Test 1.* public void testGetCredits(){  *assertEquals*(1000, joueurTest.getCredits());  }   @Test *// Test 2.a* public void testCoutAmeliorerArmureNiveauPositif(){  *assertEquals*(30, joueurTest.coutAmeliorerArmure(10));  }   @Test *// Test 2.b* public void testCoutAmeliorerArmureNiveauNul(){  *assertEquals*(0, joueurTest.coutAmeliorerArmure(0));  }   @Test *// Test 2.c* public void testCoutAmeliorerArmureNiveauNegatif(){  *assertEquals*(30, joueurTest.coutAmeliorerArmure(-10));  }   @Test *// Non demandé, tests d'armes comme ceux d'armure - a* public void testCoutAmeliorerArmeNiveauPositif(){  *assertEquals*(160, joueurTest.coutAmeliorerArme(20));  }   @Test *// Non demandé, tests d'armes comme ceux d'armure - b* public void testCoutAmeliorerArmeNiveauNul(){  *assertEquals*(0, joueurTest.coutAmeliorerArme(0));  }   @Test *// Non demandé, tests d'armes comme ceux d'armure - c* public void testCoutAmeliorerArmeNiveauNegatif(){  *assertEquals*(160, joueurTest.coutAmeliorerArme(-20));  }   @Test *// Test 3.a* public void testAmeliorerArmeTrue(){  *assertEquals*(true, joueurTest.ameliorerArme());  }   @Test *// Test 3.b* public void testAmeliorerArmeFalse(){  joueurTest.setCredits(100);  *assertEquals*(false, joueurTest.ameliorerArme());  }   @Test *// Test 4.a* public void testAmeliorerArmeCreditsOK(){  joueurTest.ameliorerArme();  *assertEquals*(360, joueurTest.getCredits());  }   @Test *// Test 4.b* public void testAmeliorerArmeNiveauOK(){  joueurTest.ameliorerArme();  *assertEquals*(41, joueurTest.getNiveauArme());  }   @Test *// Test 4.c* public void testAmeliorerArmeInvalide(){  joueurTest.setNiveauArme(100);  joueurTest.ameliorerArme();  *assertEquals*(1000, joueurTest.getCredits());  } } |