# Code Source

/\*

\* TestJoueur.java 23/05/2021

\* IUT de Rodez, pas de droit d'auteur

\*/

**package** othello.tests;

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

**import** org.junit.jupiter.api.AfterEach;

**import** org.junit.jupiter.api.BeforeEach;

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

**import** othello.caracteristiques.Couleur;

**import** othello.caracteristiques.Position;

**import** othello.joueur.Joueur;

**import** othello.Plateau;

**import** othello.Pion;

/\*\*

\* Tests unitaires de la classe Joueur

\* **@author** Théo Michellon, Maxime Moskalyk, Yohann May David Simonin, Simon

\* Launay

\*

\*/

**class** TestJoueur {

/\*\*

\* Test de joueur valide

\* Jeu de données pour les méthodes

\*/

**public** **final** Joueur[] VALIDES = {

**new** Joueur("CRAZYMat", Couleur.***BLANC***),

**new** Joueur("Ghalhyus", Couleur.***NOIR***),

**new** Joueur("Guepaz", Couleur.***NOIR***),

**new** Joueur("Mausaille", Couleur.***BLANC***),

**new** Joueur("moska", Couleur.***BLANC***)

};

/\*\*

\* Jeu de données pour la construction de plateau

\*/

**final** Pion[]ENSEMBLE1 = {

**new** Pion(**new** Position(2, 3), Couleur.***NOIR***),

**new** Pion(**new** Position(3, 3), Couleur.***NOIR***),

**new** Pion(**new** Position(3, 4), Couleur.***NOIR***),

**new** Pion(**new** Position(4, 3), Couleur.***NOIR***),

**new** Pion(**new** Position(4, 4), Couleur.***BLANC***)

};

/\*\*

\* Jeu de données pour la construction de plateau

\*/

**final** Pion[] ENSEMBLE2 = {

**new** Pion(**new** Position(2, 3), Couleur.***BLANC***),

**new** Pion(**new** Position(3, 3), Couleur.***BLANC***),

**new** Pion(**new** Position(3, 4), Couleur.***BLANC***),

**new** Pion(**new** Position(4, 2), Couleur.***BLANC***),

**new** Pion(**new** Position(4, 3), Couleur.***BLANC***),

**new** Pion(**new** Position(4, 4), Couleur.***BLANC***)

};

/\*\*

\* Jeu de données pour la méthode getScore(Plateau)

\*/

**final** Plateau[] DAMIERS = {

**new** Plateau(),

**new** Plateau(ENSEMBLE1),

**new** Plateau(ENSEMBLE2)

};

/\*\*

\* Taille du jeu de données VALIDES

\*/

**public** **final** **int** TAILLE = VALIDES.length;

@BeforeEach

**void** setUp() **throws** Exception {

}

@AfterEach

**void** tearDown() **throws** Exception {

}

/\*\*

\* Test de la méthode getPseudo()

\*/

@Test

**void** testGetPseudo() {

/\* jeu de données avec les valeurs attendues \*/

**final** String[] ATTENDUS= {

"CRAZYMat",

"Ghalhyus",

"Guepaz",

"Mausaille",

"moska"

};

**for** (**int** jdd = 0; jdd < TAILLE ; jdd++) {

*assertEquals*(ATTENDUS[jdd], VALIDES[jdd].getPseudo());

}

}

/\*\*

\* Test de la méthode getCouleur()

\*/

@Test

**void** testGetCouleur() {

/\* jeu de données avec les valeurs attendues \*/

**final** Couleur[] ATTENDUS = {

Couleur.***BLANC***, Couleur.***NOIR***, Couleur.***NOIR***,

Couleur.***BLANC***, Couleur.***BLANC***

};

**for** (**int** jdd = 0; jdd < TAILLE ; jdd++) {

*assertEquals*(ATTENDUS[jdd], VALIDES[jdd].getCouleur());

}

}

/\*\*

\* Test de la méthode PeutJouer

\*/

@Test

**void** testPeutJouer() {

*assertTrue*(VALIDES[0].peutJouer(DAMIERS[0]));

*assertTrue*(VALIDES[1].peutJouer(DAMIERS[0]));

*assertTrue*(VALIDES[2].peutJouer(DAMIERS[1]));

*assertTrue*(VALIDES[3].peutJouer(DAMIERS[1]));

*assertFalse*(VALIDES[4].peutJouer(DAMIERS[2]));

}

/\*\*

\* Test de la méthode getScore(Plateau)

\*/

@Test

**void** testGetScore() {

/\* jeu de données avec les valeurs attendues \*/

**final** **int**[] ATTENDUS = { 2, 2, 4, 1, 6 };

*assertEquals*(ATTENDUS[0], VALIDES[0].getScore(DAMIERS[0]));

*assertEquals*(ATTENDUS[1], VALIDES[1].getScore(DAMIERS[0]));

*assertEquals*(ATTENDUS[2], VALIDES[2].getScore(DAMIERS[1]));

*assertEquals*(ATTENDUS[3], VALIDES[3].getScore(DAMIERS[1]));

*assertEquals*(ATTENDUS[4], VALIDES[4].getScore(DAMIERS[2]));

}

/\*\*

\* Test de la méthode toString()

\*/

@Test

**void** testToString() {

/\* jeu de données avec les valeurs attendues \*/

**final** String[] ATTENDUS = {

"CRAZYMat",

"Ghalhyus",

"Guepaz",

"Mausaille",

"moska"

};

**for** (**int** jdd = 0; jdd < TAILLE ; jdd++) {

*assertEquals*(ATTENDUS[jdd], VALIDES[jdd].toString());

}

}

/\*\*

\* Test de la méthode Equals()

\*/

@Test

**void** testEqualsObject() {

/\* jeu de donnée copie de VALIDES \*/

**final** Joueur[] COPIES\_VALIDES = {

**new** Joueur("CRAZYMat", Couleur.***BLANC***),

**new** Joueur("Ghalhyus", Couleur.***NOIR***),

**new** Joueur("Guepaz", Couleur.***NOIR***),

**new** Joueur("Mausaille", Couleur.***BLANC***),

**new** Joueur("moska", Couleur.***BLANC***)

};

**for**(**int** jdd = 0 ; jdd < TAILLE ; jdd++) {

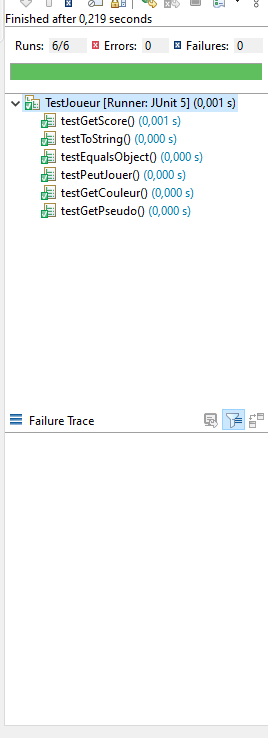
*assertEquals*(COPIES\_VALIDES[jdd], VALIDES[jdd]);

}

}

}

# Résultats



# Explications

J’ai donc tout d’abord créé un jeu de donné contenant des joueurs valides et servant en même temps de test pour le seul constructeur de la classe. Ensuite j’ai instancié deux tableaux de Pion afin de construire les différents plateau mes tests, j’ai donc ensuite créé mes plateaux. J’ai finalement créé une constante pour pouvoirs avoir la taille de mon jeu de données valides, sans avoir a appelé la méthode Arrays.length.

1. Joueur(String, Couleur)

Comme dit précédemment, les tests valides pour le constructeur ont été effectué lors de l’initialisation des jeux de données. Il ne me restait donc plus qu’à tester les joueurs invalides. J’ai donc créé deux jeux de données, un pour les pseudos et un autre pour les couleurs. Dans le premier j’ai mis les 4 combinaisons d’échec possible avec Couleur et un pseudo valide pour vérifier que ça échoue bien avec seulement une couleur invalide. Pour les couleurs j’ai donc mis deux couleurs valides, pour vérifier que ça échouait bien à cause du pseudo, puis que des Couleurs invalides pour le reste des combinaisons possibles. J’ai donc juste regardé que les constructeurs renvoyaient bien une exception, sinon je forçais l’échec.

1. getPseudo

Pour le test de getPseudo j’ai juste créé un jeu de données contenant tous les pseudos de mes joueurs valides, puis j’ai juste vérifié que les valeurs récupéré étaient les mêmes.

1. getCouleur

Le principe est le même que pour getPseudo, je créer un tableau contenant les valeurs attendues et je vérifie qu’elles correspondent bien avec celles qui sont récupérées.

1. getScore

Encore une fois, le principe est le même que précédemment, sauf que cette fois la méthode est aussi lié au plateau. J’ai donc créé un tableau avec le nombre de pion de la même couleur que le joueur et j’ai vérifié que les nombres correspondaient. C’est pour avoir des résultats plus variés que j’ai créé plusieurs plateaux.

1. peutJouer

Comme la méthode ne fait que renvoyer un booléen si le joueur peut jouer, j’ai juste vérifié que cela correspondait bien avec les valeurs attendues. Encore une fois les plusieurs tableaux nous permettent d’avoir des résultats variés.

1. toString

Le principe est le même que pour les getters, je vérifie que les valeurs récupérées soient les mêmes que celles attendus.

1. equals

Pour ce dernier test c’est très simple, j’ai créé une copie du tableau VALIDES, puis j’ai vérifié que chaque élément était bien égal à sa copie.