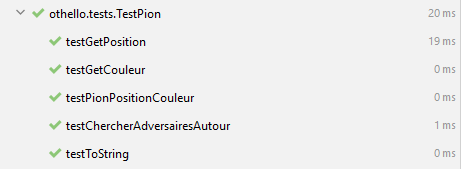
# Code Source

/\*  
 \* TestPion.java 27 Avril 2021  
 \* IUT Rodez, pas de droits  
 \*/

package othello.tests;  
  
import junit.framework.Assert;  
import junit.framework.TestCase;  
import othello.Pion;  
import othello.Plateau;  
import othello.caracteristiques.Couleur;  
import othello.caracteristiques.Position;  
  
import java.util.LinkedList;  
import java.util.List;  
  
public class TestPion extends TestCase {  
  
 public void tearDown() throws Exception {  
 }  
  
 /\*\* Pions valides pour realiser des tests \*/  
 static final Pion[] PIONS\_VALIDES = {  
 new Pion(new Position(7, 7), Couleur.NOIR),  
 new Pion(new Position(0, 0),Couleur.BLANC),  
 new Pion(new Position(3, 4),Couleur.NOIR),  
 new Pion(new Position(4, 3),Couleur.BLANC),  
 new Pion(new Position(3, 3),Couleur.NOIR),  
 new Pion(new Position(4, 4),Couleur.BLANC),  
 new Pion(new Position(4, 5),Couleur.NOIR),  
 new Pion(new Position(4, 7),Couleur.BLANC),  
 new Pion(new Position(2, 1),Couleur.NOIR)  
 };  
  
 /\*\* Tableau contenant des positions invalides \*/  
 static final Position[] POSITIONS\_INVALIDES = {  
 new Position(-1, -1),  
 new Position(-1, 5),  
 new Position(4, -5),  
 new Position(9, 9),  
 new Position(10, 5),  
 new Position(8, 12),  
 new Position(74, -87),  
 new Position(80, 32)  
 };  
  
 /\*\*  
 \* Tests unitaires du constructeur Pion qui prend en paramètre :  
 \* une position et une couleur  
 \*/  
 public void testPionPositionCouleur() {  
 new Pion(new Position(7, 7), Couleur.NOIR);  
 new Pion(new Position(0, 0), Couleur.BLANC);  
 new Pion(new Position(3, 4), Couleur.NOIR);  
 new Pion(new Position(4, 3), Couleur.BLANC);  
 new Pion(new Position(4, 4), Couleur.NOIR);  
 new Pion(new Position(3, 3), Couleur.BLANC);  
 new Pion(new Position(4, 5), Couleur.NOIR);  
 new Pion(new Position(4, 7), Couleur.BLANC);  
 new Pion(new Position(2, 1), Couleur.NOIR);  
  
 /\* test avec des dates invalides \*/  
 for (Position positionsInvalide : POSITIONS\_INVALIDES) {  
 try {  
 new Pion(positionsInvalide, Couleur.NOIR);  
 fail();  
 } catch (IndexOutOfBoundsException echec) {  
 }  
 }  
 }  
  
 /\*\*  
 \* Tests unitaires de getPosition()  
 \*/  
 public void testGetPosition() {  
 final Position[] ATTENDU = {  
 new Position(7, 7),  
 new Position(0, 0),  
 new Position(3, 4),  
 new Position(4, 3),  
 new Position(3, 3),  
 new Position(4, 4),  
 new Position(4, 5),  
 new Position(4, 7),  
 new Position(2, 1)  
 };  
  
 for (int i = 0; i < 8; i++) {  
 Assert.assertEquals(ATTENDU[i], PIONS\_VALIDES[i].getPosition());  
 }  
 }

/\*\*  
 \* Tests unitaires de getCouleur()  
 \*/  
 public void testGetCouleur() {  
 final Couleur[] ATTENDU = {  
 Couleur.NOIR, Couleur.BLANC, Couleur.NOIR, Couleur.BLANC,  
 Couleur.NOIR, Couleur.BLANC, Couleur.NOIR, Couleur.BLANC,  
 Couleur.NOIR  
 };  
  
 for (int i = 0; i < 9; i++) {  
 Assert.assertEquals(ATTENDU[i], PIONS\_VALIDES[i].getCouleur());  
 }  
 }  
  
 /\*\*  
 \* Tests unitaires de chercherAdversairesAutour()  
 \*/  
 public void testChercherAdversairesAutour() {  
 List<Pion> listePion = new LinkedList<>();  
 Plateau damier = new Plateau();  
 int compteur;  
 Pion[] adversaireAttendu = {  
 // Premiere possibilite  
 new Pion(new Position(3, 3), Couleur.BLANC),  
 // Deuxieme possibilite  
 new Pion(new Position(4, 4), Couleur.BLANC)  
 };  
  
 Pion[] pionATester = {  
 // Premiere possibilite  
 new Pion(new Position(3, 2), Couleur.NOIR),  
 new Pion(new Position(2, 3), Couleur.NOIR),  
 // Deuxieme possibilite  
 new Pion(new Position(5, 4), Couleur.NOIR),  
 new Pion(new Position(4, 5), Couleur.NOIR)  
 };  
  
  
 compteur = 0;  
 for (int i = 0; i < 2; i++) {  
 for (int j = 0; j < 2; j++) {  
 listePion = pionATester[compteur]  
 .chercherAdversairesAutour(damier);  
 compteur++;  
 for (Pion aTester : listePion) {  
 Assert.assertEquals(adversaireAttendu[i].getPosition(),  
 aTester.getPosition());  
 }  
 }  
 }  
  
  
  
 }  
  
 /\*\*  
 \* Tests unitaires de toString()  
 \*/  
 public void testToString() {  
 final String[] ATTENDU = {  
 "x", "o", "x", "o", "x", "o", "x", "o", "x"  
 };  
  
 for (int i = 0; i < 9; i++) {  
 Assert.assertEquals(ATTENDU[i], PIONS\_VALIDES[i].toString());  
 }  
 }  
}

# Résultats



# Explications

Nous avons tout d’abord commencé par créer un jeu de données sur des pions valides et invalides pour effectuer des tests dessus par la suite.

1. Pion(Position, Couleur)

Pour tester le constructeur nous avons commencé par créer des pions valides. Puis pour vérifier que les pions invalides étaient bien invalides nous avons utilisé un try catch. Afin de l’utiliser correctement nous avons mit celui-ci dans un for each utilisant le tableau contenant les différents pions invalides. Pour ce faire nous avons à chaque itération essayé de créer un pion invalide, si il était créé alors il y a une erreur, sinon, si le pion n’était pas créé par le constructeur le test est validé.

1. getPosition

Afin de réaliser les tests pour obtenir la position d’un pion nous avons utilisé un for each sur les pions valides, à chaque itération nous vérifions la validité entre la méthode getPosition sur les pions valides et un tableau initié dans le test contenant toutes les positions attendues.

1. getCouleur

Afin de réaliser les tests pour vérifier le bon fonctionnement de getPosition nous avons utilisé la même méthode que pour getPosition. Nous avons créé un tableau contenant toutes les couleurs attendues puis nous avons à l’aide d’un for vérifié que chaque couleur obtenu à l’aide de la méthode getCouleur sur les pions valides correspond à celle attendu.

1. chercherAdversairesAutour

Pour tester la méthode chercherAdversairesAutour à était un peu plus difficile. Nous avons commencé par créer un tableau contenant les pions attendus lors de l’utilisation de la méthode, un tableau sur les différents pions à tester. Ensuite nous avons décidé de réaliser une boucle qui pour chaque pions à tester vérifie que les pions trouvées correspondent bien à ceux attendues.

1. toString

Afin de réaliser les tests du toString nous avons utilisé la même méthode que les getters. Nous avons créer un tableau contenant les toString attendus puis à l’aide d’un for each nous avons vérifié que chaque toString sur les pions valides renvoies bien le toString attendu.