NBA codere Tests

1º de grado superior
Desarrollo de aplicaciones multiplataforma
Entornos de desarrollo

Juan Antonio Perez Beltran

Víctor Páez Anguita

Sandro Tejero Delgado

Sergio Cosano Jiménez

Rafa Lucena Valle

Índice:

UsuarioTest:

El Test “existe_usuario” comprueba si el usuario existe al crearlo. Devolverá el mismo nombre o una excepción.

```
@Test
void existe_usuario() {
    try {
        assertEquals("nombre1", usuario1.getName());
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

El Test “asignar_nombre” comprueba si el usuario tiene el nombre al asignarlo. Devolverá el nombre o una excepción.

```
@Test
void asignar_nombre() {
    try {
        usuario1.setName("nombre2");
        assertEquals("nombre2", usuario1.getName());
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

El Test “anadir_usuario” comprueba si al añadir un “jugador” se añade al “Array de Jugadores” o devuelve una excepción.

```
@Test
void anadir_usuario() {
    assertThrows("Jugador no añadido", usuario1.AnadirUsuario("Jugador1"));
}
```

El Test “sistema_de_apuestas” comprueba si al añadir un “equipo” se añade al “Sistema de Apuestas” o devuelve una excepción.

```
@Test
void sistema_de_apuestas() {
    assertThrows("Equipo no añadido", usuario1.sistemadeaupestas(equipo1));
}
```

DineroTest:

El Test “obtener_puntos” comprueba si los puntos que tiene el usuario son correctos.

```
@Test
void obtener_puntos() {
    assertEquals(100, dinero1.getpuntos());
}
```

El Test “asignar_puntos” comprueba si asigna los puntos son correctos.

```
@Test
void asignar_puntos() {
    dinero1.setpuntos(200);
    assertEquals(200, dinero1.getpuntos());
}
```

El Test “anadir_puntos” comprueba si la suma de los puntos es correcta.

```
@Test
void anadir_puntos() {
    dinero1.acumularPuntos(100);
    assertEquals(200, dinero1.getpuntos());
}
```

El Test “restar_puntos” comprueba si la resta de los puntos es correcta.

```
@Test
void restar_puntos() {
    dinero1.acumularPuntos(-100);
    assertEquals(0, dinero1.getpuntos());
}
```

El Test “no_anadir_puntos” comprueba si al no añadir ningún punto devuelve el mismo resultado.

```
@Test
void no_anadir_puntos() {
    dinero1.acumularPuntos(0);
    assertEquals(100, dinero1.getpuntos());
}
```

JugadorTest:

El primer test realizado a la clase jugador tendrá relación con la función que funcionara para anotar puntos para un equipo. Básicamente comprueba que al insertar la estadística de tiro del jugador si esta es valida para marcar puntos.

```java
import static org.junit.jupiter.api.Assertions.*;

class JugadorTest {

    //static Jugador jugador = new jugador("prueba","50","50","50","50","50","50");

    @Test
    void testTiraCanasta(int tiro) {
        boolean result=false;
        if(tiro>0) {
            assertEquals(result, true);
        }
    }

```

El siguiente test es para comprobar que la función de pasarBalon sea valida según la estadística del jugador que reciba para sumar probabilidad a la función de canasta.

```java

    @Test
    void testPasarBalon(int pase) {
        boolean result=false;
        if(pase>0) {
            assertEquals(result, true);
        }
    }

```

El siguiente test comprobará que la función falta valide que al recibir la estadística del jugador determine si se hace falta o no.

```java

    @Test
    void testFalta(int probabilidad) {
        boolean result=false;
        if(probabilidad>60) {
            assertEquals(result, true);
        }
    }

```

El siguiente comprobará que la estadísticas del jugador resten probabilidad a la función de canasta.

```java

    @Test
    void testDefender(int ataque, int defensa) {
        boolean result=false;
        if(defensa>ataque) {
            assertEquals(result, true);
        }
    }


```

MercadoTest:

El siguiente test de mercado comprobará que de la colección de jugadores que reciba existan los jugadores para poder ficharlos.

```
10
11      @Test
12      void testFichaJugador(jugadores) {
13          assertEquals(jugadores, null);
14      }
15
```

El siguiente test comprobará que de la colección de jugadores que reciba existan los jugadores para poder comprarlos.

```
16      @Test
17      void testVendeJugador(jugadores) {
18          assertEquals(jugadores, null);
19      }
20  }
21
```

Partido Test:

Gracias al uso de la función “assert” se ha podido crear varios test unitarios que comprueben de distintas maneras la validez del futuro código de la esta clase (Partido).

1º test_2Equipos():

```
@Test
void test_2equipos(){

    assertEquals(Equipos.lenght, 2);
}
```

Este test realiza la acción de comprobar si en el partido hay dos equipos recibiendo el tamaño del array del partido.

2º test_EqualEquipo():

```
@Test
void test_EqualEquipo() {

    assertNotEquals(Equipo[0], Equipo[1]);

}
```

Este test comprueba que los dos equipos no son iguales mediante el equals de la función de assert

3º test_TiempoPartido():

```
@Test
void test_TiempoPartido() {
    assertArrayEquals(ArrayList<Equipo>(e1), ArrayList<Equipo>(e1,e2));

}
```

Compruba el tiempo del partido.

Estadística Test :

```java
package modelo;

import static org.junit.Assert.assertTrue;
import org.hamcrest.core.IsNot;
import org.junit.Test;

class prueba_sistema {
    ;

    @Test
    void Noneg_ganados(int partidos_ganados) {
        assertTrue("El numero es negativo", partidos_ganados>0);
    }
    @Test
    void Noneg_perdidos( int partidos_perdidos) {
        assertTrue("El numero es negativo", partidos_perdidos>0);
    }

    @Test
    void Noneg_tiros(int tiros) {
        assertTrue("El numero es negativo", tiros>0);
    }
    @Test
    void Noneg_pases(int pases) {
        assertTrue("El numero es negativo", pases>0);
    }
    @Test
    void Noneg_bloqueos(int bloqueos) {
        assertTrue("El numero es negativo", bloqueos>0);
    }
    @Test
    void guarda_estadisticas (int tiros , int pases , int bloqueos , int puntos) {
        assertTrue(Estadistica.media(pases, bloqueos, tiros ,puntos));
    }


    }
```

Los test “Noneg” se encargan de asegurarse de que los números no sean negativos , como de los tiros , pases , bloqueos , partidos ganados , partidos perdidos … etc , para asegurarse que en cualquier cuenta no haya error por introducirse datos erróneos .

Y el test guarda_estadística se encarga de asegurarse de que los datos están en el formato correspondiente .