 ****

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

**CARRERA DE INGENIERÍA EN SISTEMAS E INFORMÁTICA**

**Programación Móvil**

**NRC:** 6112

**Tema: Pruebas Unitarias**

**Nombres:**

* Martínez Carlos
* Monteros Paula

**SANGOLQUÍ – ECUADOR**

**MAYO – SEPTIEMBRE 2020**

**Clase CalculadoraTest**

/\*  
 \* ESPE - DCC - PROGRAMACIÓN MÓVIL  
 \* Sistema: Calculadora\_MVP  
 \* Creado 08/06/2020  
 \*  
 \* Los contenidos de este archivo son propiedad privada y estan protegidos por  
 \* la licencia BSD  
 \*  
 \* Se puede utilizar, reproducir o copiar el contenido de este archivo.  
 \*/  
package com.example.calculadora\_mvp;  
  
import android.graphics.Path;  
  
import org.junit.Before;  
import org.junit.Test;  
  
import static org.junit.Assert.*assertEquals*;  
import static org.junit.Assert.*assertNotNull*;  
  
*/\*\*  
 \* Clase para las pruebas unitarias que se realizarán en la aplicación.  
 \*  
 \** ***@author*** *Carlos Martínez  
 \** ***@author*** *Paula Monteros  
 \*/*public class CalculadoraTest {  
  
 private Operacion operacion;  
 private Numero numero;  
 private Numero numero1;  
 private Numero numero2;  
  
 */\*\*  
 \* Método set para instanciar la variables u objetos que se usarán  
 \* en la spruebas unitarias. \*  
 \*/* @Before  
 public void setUp() throws Exception {  
 operacion = new Operacion();  
 numero = new Numero(0.0);  
 numero1 = new Numero(0.0);  
 numero2 = new Numero(0.0);  
 }  
  
 */\*\*  
 \* Prueba unitaria para verificar que el objeto no sea NULL.  
 \*/* @Test  
 public void testNotNull(){  
 *assertNotNull*(operacion);  
 }  
  
 */\*\*  
 \* Prueba Unitaria para la operación Suma entre 2 números.  
 \*/* @Test  
 public void testSuma() {  
 numero.setValor(3.0);numero1.setValor(1.0);numero2.setValor(2.0);  
 *assertEquals*(numero.getValor(),operacion.sumar(numero1,numero2).getValor());  
 numero.setValor(-5.0);numero1.setValor(-10.0);numero2.setValor(5.0);  
 *assertEquals*(numero.getValor(),operacion.sumar(numero1,numero2).getValor());  
 numero.setValor(4.28);numero1.setValor(2.14);numero2.setValor(2.14);  
 *assertEquals*(numero.getValor(),operacion.sumar(numero1,numero2).getValor());  
 numero.setValor(666000.0);numero1.setValor(-123456.0);numero2.setValor(789456.0);  
 *assertEquals*(numero.getValor(),operacion.sumar(numero1,numero2).getValor());  
  
 numero.setValor(-3.0);numero1.setValor(-3.0);numero2.setValor(-9.0);  
 *assertEquals*(numero.getValor(),operacion.modulo(numero1,numero2).getValor());  
 }  
  
 */\*\*  
 \* Prueba Unitaria para la operación Resta entre 2 números.  
 \*/* @Test  
 public void testResta() {  
 numero.setValor(-1.0);numero1.setValor(1.0);numero2.setValor(2.0);  
 *assertEquals*(numero.getValor(),operacion.restar(numero1,numero2).getValor());  
 numero.setValor(-15.0);numero1.setValor(-10.0);numero2.setValor(5.0);  
 *assertEquals*(numero.getValor(),operacion.restar(numero1,numero2).getValor());  
 numero.setValor(0.0);numero1.setValor(2.14);numero2.setValor(2.14);  
 *assertEquals*(numero.getValor(),operacion.restar(numero1,numero2).getValor());  
 numero.setValor(-912912.0);numero1.setValor(-123456.0);numero2.setValor(789456.0);  
 *assertEquals*(numero.getValor(),operacion.restar(numero1,numero2).getValor());  
 }  
  
 */\*\*  
 \* Prueba Unitaria para la operación Multiplicación entre 2 números.  
 \*/* @Test  
 public void testMultiplicacion() {  
 numero.setValor(2.0);numero1.setValor(1.0);numero2.setValor(2.0);  
 *assertEquals*(numero.getValor(),operacion.multiplicacion(numero1,numero2).getValor());  
 numero.setValor(-50.0);numero1.setValor(-10.0);numero2.setValor(5.0);  
 *assertEquals*(numero.getValor(),operacion.multiplicacion(numero1,numero2).getValor());  
 numero.setValor(4.5796);numero1.setValor(2.14);numero2.setValor(2.14);  
 *assertEquals*(numero.getValor(),operacion.multiplicacion(numero1,numero2).getValor());  
 numero.setValor(-9.7463079936E10);numero1.setValor(-123456.0);numero2.setValor(789456.0);  
 *assertEquals*(numero.getValor(),operacion.multiplicacion(numero1,numero2).getValor());  
 }  
  
 */\*\*  
 \* Prueba Unitaria para la operación División entre 2 números.  
 \*/* @Test  
 public void testDivision() {  
 numero.setValor(0.5);numero1.setValor(1.0);numero2.setValor(2.0);  
 *assertEquals*(numero.getValor(),operacion.division(numero1,numero2).getValor());  
 numero.setValor(-2.0);numero1.setValor(-10.0);numero2.setValor(5.0);  
 *assertEquals*(numero.getValor(),operacion.division(numero1,numero2).getValor());  
 numero.setValor(1.0);numero1.setValor(2.14);numero2.setValor(2.14);  
 *assertEquals*(numero.getValor(),operacion.division(numero1,numero2).getValor());  
 numero.setValor(-0.1563811029367058);numero1.setValor(-123456.0);numero2.setValor(789456.0);  
 *assertEquals*(numero.getValor(),operacion.division(numero1,numero2).getValor());  
 }  
  
 */\*\*  
 \* Prueba Unitaria para la operación exponencial.  
 \*/* @Test  
 public void testExponencial(){  
 numero.setValor(0.1111111111111111);numero1.setValor(3.0);numero2.setValor(-2.0);  
 *assertEquals*(numero.getValor(),operacion.exponencial(numero1,numero2).getValor());  
 numero.setValor(9.0);numero1.setValor(-3.0);numero2.setValor(2.0);  
 *assertEquals*(numero.getValor(),operacion.exponencial(numero1,numero2).getValor());  
 numero.setValor(1.0);numero1.setValor(3.0);numero2.setValor(0.0);  
 *assertEquals*(numero.getValor(),operacion.exponencial(numero1,numero2).getValor());  
 numero.setValor(-27.0);numero1.setValor(-3.0);numero2.setValor(3.0);  
 *assertEquals*(numero.getValor(),operacion.exponencial(numero1,numero2).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la operación factorial de un número.  
 \*/* @Test  
 public void testFactorial(){  
 numero.setValor(6.0);numero1.setValor(3.0);  
 *assertEquals*(numero.getValor(),operacion.factorial(numero1).getValor());  
 numero.setValor(2.0);numero1.setValor(2.0);  
 *assertEquals*(numero.getValor(),operacion.factorial(numero1).getValor());  
 numero.setValor(40320.0);numero1.setValor(8.0);  
 *assertEquals*(numero.getValor(),operacion.factorial(numero1).getValor());  
 numero.setValor(1.0);numero1.setValor(1.0);  
 *assertEquals*(numero.getValor(),operacion.factorial(numero1).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la operación raíz cuadrada de un número.  
 \*/* @Test  
 public void testRaiz(){  
 numero.setValor(1.5892857142857142);numero1.setValor(2.5);  
 *assertEquals*(numero.getValor(),operacion.raiz(numero1).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la operación MOD entre dos números.  
 \*/* @Test  
 public void testMod(){  
 numero.setValor(2.0);numero1.setValor(-8.0);numero2.setValor(5.0);  
 *assertEquals*(numero.getValor(),operacion.modulo(numero1,numero2).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la operación logaritmo de un número.  
 \*/* @Test  
 public void testLog(){  
 numero.setValor(1.7185016888); numero1.setValor(52.3);  
 *assertEquals*(numero.getValor(),operacion.logBase10(numero1).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la operación logaritmo natural de un número.  
 \*/* @Test  
 public void testLn(){  
 numero.setValor(2.484906649788);numero1.setValor(12.0);  
 *assertEquals*(numero.getValor(),operacion.logNatural(numero1).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la función seno.  
 \*/* @Test  
 public void testSeno(){  
 numero.setValor(0.8660254037844385);numero1.setValor(60.0);  
 *assertEquals*(numero.getValor(),operacion.*seno*(numero1).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la función coseno.  
 \*/* @Test  
 public void testCoseno(){  
 numero.setValor(0.500000433432915);numero1.setValor(60.0);  
 *assertEquals*(numero.getValor(),operacion.*serieTaylor*(numero1).getValor());  
 }  
  
 */\*\*  
 \* Prueba unitaria para la transformación de decimal a binario.  
 \*/* @Test  
 public void testBinario(){  
 numero1.setValor(850.0);  
 *assertEquals*("00000001101010010",operacion.*decimalABinario*(numero1));  
 }  
  
 */\*\*  
 \* Prueba unitaria para la transformación de decimal a octal.  
 \*/* @Test  
 public void testOctal(){  
 numero1.setValor(850.0);  
 *assertEquals*("001522",operacion.*decimalAOctal*(numero1));  
 }  
  
 */\*\*  
 \* Prueba unitaria para la transformación de decimal a hexadecimal.  
 \*/* @Test  
 public void testDecimal(){  
 numero1.setValor(5963.0);  
 *assertEquals*("0174B",operacion.*decimalAHexadecimal*(numero1));  
 }  
  
 */\*\*  
 \* Prueba unitaria para el resultado de la operación realizada por la polaca inversa.  
 \*/* @Test  
 public void testPol(){  
 PrefijoInfijo p = new PrefijoInfijo();  
 EvaluarResultado e = new EvaluarResultado();  
  
 *assertEquals*("",p.*Infijo2PosfijoTxt*("2 + 3 - √ 2 \* -3"));  
 *assertEquals*("0.7573595",e.*Postfijo2resulTxt*("2 3 + 2 √ -3 \* -"));  
  
 *assertEquals*("",p.*Infijo2PosfijoTxt*("5\*-8+4^-2"));  
 *assertEquals*("-37.9375",p.*Infijo2PosfijoTxt*("5\*-8+4^-2+√4.1"));  
  
 *assertEquals*("2 +",p.*Infijo2PosfijoTxt*("2 \* 4!"));  
 *assertEquals*("-39.9375",e.*Postfijo2resulTxt*("2 4 ! \*"));  
  
 *assertEquals*("5 -8 \* 4 -2 ^ 3 ! \* +",p.*Infijo2PosfijoTxt*("5\*-8+4^-2\*3!"));  
 *assertEquals*("-39.625",e.*Postfijo2resulTxt*("5 -8 \* 4 -2 ^ 3 ! \* +"));  
  
 *assertEquals*("5 4 ! √ \*",p.*Infijo2PosfijoTxt*("5\*√4!"));  
 *assertEquals*("24.494898",e.*Postfijo2resulTxt*("5 4 ! √ \*"));  
  
 *assertEquals*("",p.*Infijo2PosfijoTxt*("8 % -3 \*2"));  
 *assertEquals*("-2.0",e.*Postfijo2resulTxt*("8 -3 % 2 \*"));  
  
 *assertEquals*("3 8 10 l √ \* -",p.*Infijo2PosfijoTxt*("3-8\*√log10"));  
 *assertEquals*("-5.0",e.*Postfijo2resulTxt*("3 8 10 l √ \* -"));  
  
 *assertEquals*("3 4 % -1 / 6 ! +",p.*Infijo2PosfijoTxt*("3%4/(-1+6!)"));  
 *assertEquals*("-5.0",e.*Postfijo2resulTxt*("3 4 % -1 / 6 ! +"));  
  
 *assertEquals*("60 23.6 9 \* + 0.98 3 / - 96.3 l 0.666 l \* +",p.*Infijo2PosfijoTxt*("60+23.6\*9-0.98/3+log96.3\*log0.666"));  
 *assertEquals*("-5.0",e.*Postfijo2resulTxt*("60 23.6 9 \* + 0.98 3 / - 96.3 l 0.666 l \* +"));  
 }  
  
}

**Resultado de las Pruebas Unitarias Ejecutadas**

