CALCULADORA DE FRACCIONES

A clase Fraction é un obxecto que almacena información sobre unha fracción (numerador e denominador). Ten diversos constructores e métodos implementando o comportamento das fraccións.

ATRIBUTOS

Dúas variables enteiras privadas **enteiras** (int) que almacenen o numerador e o denominador.
Podes consultar Información sobre control de acceso para coñecer máis sobre o control de acceso.

CONSTRUCTORES

- Constructor de dous parámetros ***int*** que permita inicializar o numerador e o denominador, e que lance unha ***IllegalArgumentException*** si o denominador é 0. Si o denominador é negativo, debe mover o signo ao numerador de xeito apropiado
- Constructor con un parámetro ***int*** que cree unha fracción de ese valor poñendo o denominador a 1.
- Constructor sen argumentos que inicializa o numerador a 0 e o denominador a 1 (fracción "nula", de valor total 0).

Exemplos:

Fraction f=new Fraction (2/-3); *almacenaría -2 no numerador e 3 no denominador.*
Fraction f=new Fraction (-2/-3); *almacenaría 2 no numerador e 3 no denominador.*
Fraction f=new Fraction(8); *almacenaría 8 no numerador e 3 no denominador*

MÉTODOS

Método	Parámetros	Devolve	Descripción
getNumerator()	ningún	int	devolve o valor do numerador
getDenominator()	ningún	int	devolve o valor do denominador
toString()	ningún	String	devolve a cadea "numerador/denominador"
toDouble()	ningún	double	devolve o valor double resultado de dividir o numerador entre o denominador
add()	Fraction other	Fraction	devolve unha **nova** Fraction que é a suma entre esta Fraction e other
subtract()	Fraction other	Fraction	devolve unha **nova** Fraction que é a o resultado de restarlle a esta Fraction a Fraction other
multipy()	Fraction other	Fraction	devolve unha **nova** Fraction que é a multiplicación entre esta Fraction e other
divide()	Fraction other	Fraction	devolve unha **nova** Fraction que é o resultado de dividir esta Fraction entre other. **Lanza IllegalArgumentException** si other é a fracción nula.
equals()	Object obj	boolean	devolve **true** si obj é unha Fraction equivalente a esta.
toLowestTerms()	ningún	Fraction	devolve unha **nova** Fraction que é o resultado de simplificar esta.
gcd()	int a, int b	int	método **estático** que devolve o máximo común divisior entre a e b

Se pide elaborar unha clase JUnit FractionTest que comprobe o correcto funcionamento de esta clase.

SOLUCIÓN

```
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class FractionTest {

        @Test( expected = IllegalArgumentException.class )
        public void shouldLaunchIllegalArgumentException() throws
IllegalArgumentException {
                new Fraction(1,0);
        }

        @Test
        public void shouldSwitchSign() {
                Fraction f=new Fraction(3,-1);
                int n=f.getNumerator();
                int d=f.getDenominator();
                assertEquals("Numerator must be -3",-3,n);
                assertEquals("Denominator must be 1",1,d);
        }

        @Test
        public void denominatorIsOne() {
                Fraction f=new Fraction(12);
                assertEquals("Denominator must be 1",1,f.getDenominator());
                assertEquals("Numerator must be 12",12,f.getNumerator());
        }

        @Test
        public void isNullFraction() {
                Fraction f=new Fraction();
                int n=f.getNumerator();
                int d=f.getDenominator();
                assertTrue("Numerator must be 0 and Denominator must be 1",
(n==0)&&(d==1));
        }

        @Test
        public void getCorrectNumeratorAndDenominator() {
                Fraction f=new Fraction(23,17);
                assertEquals("Numerator must be 23 ",23,f.getNumerator());
                assertEquals("Denominator must be 17 ",17,f.getDenominator());
        }

        @Test
        public void isCorrectString() {
                String s=new Fraction(12,4).toString();
                // assertTrue("Fraction is 12/4",s.equals("12/4"));
                assertEquals("Fraction is 12/4","12/4",s);
        }

        @Test
        public void testFractionValue() {
                Fraction f=new Fraction(3,2);
                assertEquals("Double value Must be
1.5",1.5,f.toDouble(),0.000000001);
        }
```

```
	/**
		As especificacións da clase indican que o método multiply debe
		crear unha NOVA fracción con valor o resultado da multiplación.
		Polo tanto, o valor das fraccións orixinais non PODE variar.
	*/
	@Test
	public void testMultiply() {
		Fraction a=new Fraction(2,3);
		Fraction b=new Fraction(3,2);
		Fraction c=a.multiply(b);

		// Verificamos o resultado
		assertTrue("Multiply must be 6/6",
(c.getNumerator()==6)&&(c.getDenominator()==6));

		// Verificamos que as fraccions orixinais non cambian
		assertTrue("A fraccion orixinal a e 2/3",
(a.getNumerator()==2)&&(a.getDenominator()==3));
		assertTrue("A fraccion orixinal a e 3/2",
(b.getNumerator()==3)&&(b.getDenominator()==2));
	}

	@Test
	public void testDivide() {
		Fraction f=new Fraction(5,3);
		Fraction f1=new Fraction(5,3);
		Fraction g=new Fraction(4,6);
		Fraction g1=new Fraction(4,6);
		Fraction s=g.divide(f);
		Fraction r=new Fraction(12,30);
		assertEquals("Divide Result must be equivalent to -6/6",r,s);
		assertEquals("O primeiro operando non cambiou ",f,f1);
		assertEquals("O segundo operando non cambiou ",g,g1);
	}

	@Test
	public void testAdd() {
		Fraction f=new Fraction(5,3);
		Fraction f1=new Fraction(5,3);
		Fraction g=new Fraction(4,6);
		Fraction g1=new Fraction(4,6);
		Fraction s=f.add(g);
		Fraction r=new Fraction(14,6);
		assertEquals("Add Result must be equivalent to 14/6",r,s);
		assertEquals("O primeiro operando non cambiou ",f,f1);
		assertEquals("O segundo operando non cambiou ",g,g1);
	}


	@Test
	public void testSubstract() {
		Fraction f=new Fraction(5,3);
		Fraction f1=new Fraction(5,3);
		Fraction g=new Fraction(4,6);
		Fraction g1=new Fraction(4,6);
		Fraction s=g.subtract(f);
		Fraction r=new Fraction(-6,6);
		assertEquals("Substract Result must be equivalent to -6/6",r,s);
		assertEquals("O primeiro operando non cambiou ",f,f1);
		assertEquals("O segundo operando non cambiou ",g,g1);
	}
```

```
	@Test
	public void testEquals() {
		Fraction a=new Fraction(3,5);
		Fraction b=new Fraction(15,25);
		assertTrue("3/5 e equivalente a 15/25",a.equals(b));
	}


	@Test
	public void testSimplify() {
		Fraction a=new Fraction(15,25).toLowestTerms();
		assertTrue("Simplify is 3/5",
(a.getNumerator()==3)&&(a.getDenominator()==5));

		a=new Fraction(17,25).toLowestTerms();
		assertTrue("Simplify is 17/25",
(a.getNumerator()==17)&&(a.getDenominator()==25));

	}

	@Test
	public void testGcd() {
		int gcd=Fraction.gcd(5,3);
		assertEquals("GCD[5,3]=1",1,gcd);
		gcd=Fraction.gcd(2,4);
		assertEquals("GCD[2,4]=2",2,gcd);
	}

	public static void main(String[] args) {
		Result result = JUnitCore.runClasses(FractionTest.class);
		System.out.println("Realizados "+result.getRunCount()+" tests en
"+result.getRunTime()+"ms");
		for (Failure failure : result.getFailures()) {
			System.out.println(failure.toString());
		}
	}
}
```