Ecuaciones

Enunciado

Desarrollar las clases necesarias para representar una operación matemática contemplando las operaciones básicas (suma, resta, multiplicación y división). La estructura debe permitir componer expresiones con cantidades arbitrariamente grandes de parámetros y anidamientos.

Las expresiones sólo se podrán construir a través de código en un lenguaje o pseudocódigo, utilizando las interfaces de las clases que fueran necesarias. Se requiere que la función sepa calcular su valor o devolver una excepción adecuada.

La expresión deberá ser capaz de mostrarse a sí misma en forma de texto, incluyendo paréntesis u otros símbolos donde fuera necesario, de forma que se pueda leer la expresión original. No es necesario preocuparse por evitar paréntesis superfluos. Es decir, vale mostrar (2 * 5) + 10.

Manejo de variables. La expresión podrá contener una o más variables. En el momento de solicitar el cálculo, se deberá indicar que valor tienen las variables.

Diagrama de clases

Diagrama de objetos de (((7 + x) * (2 - y)) / z)

Diagrama de objetos de (((4 * 3) * 2) + a)

Código de los métodos mas importantes

Mostrar la representación de la ecuación

#Numerica

```
public String getRepresent(){
	return  getNumero().toString();
}
```

#Variable

```
public String getRepresent(){
	return  getLetra().toString();
}
```

#Binaria

```
public String getRepresent(){
	return "(" + elem1.getRepresent() + " " +
		this.getSimbolo() + " " +
		elem2.getRepresent() +  ")";
}
```

#Suma

```
@Override
public String getSimbolo() {
	return "+";
}
```

#Resta

```
@Override
public String getSimbolo() {
	return "-";
}
```

#Multiplicacion

```
@Override
public String getSimbolo() {
	return "*";
}
```

#Division

```
@Override
public String getSimbolo() {
	return "/";
}
```

Calcular el valor de la ecuación

#Numerica

```
public int getValor(Map<Character, Integer> variables){
	return getNumero();
}
```

#Variable

```
public int getValor(Map<Character, Integer> variables){
	Integer valor = variables.get(getLetra());
	if(valor == null){
		throw new FaltaVariableException(getLetra());
	}
	return valor;
}
```

#Suma

```
@Override
public int getValor(Map<Character, Integer> variables) {
	return this.getElem1().getValor(variables) +
	this.getElem2().getValor(variables);
}
```

#Resta

```
@Override
public int getValor(Map<Character, Integer> variables) {
	return this.getElem1().getValor(variables) -
	this.getElem2().getValor(variables);
}
```

#Multiplicacion

```
@Override
public int getValor(Map<Character, Integer> variables) {
	return this.getElem1().getValor(variables) *
	this.getElem2().getValor(variables);
}
```

#Division

```
@Override
public int getValor(Map<Character, Integer> variables) {
	return this.getElem1().getValor(variables) /
	this.getElem2().getValor(variables);
}
```

NOTA: Que pasaria si se trata de hacer una division por cero en el metodo getValor de la clase Division?
Se lanzaría una ArithmeticException con el mensaje “/ by zero”, al ser una excepcion no chequeada y bastante descriptiva del problema, no nos vemos en la necesidad de tratarla ni envolverla en ninguna otra.

Apéndice de tests

A continuación realizaremos los tests que garantizaran el correcto funcionamiento de nuestras clases de dominio

Nota: Para el testeo utilizaremos JUnit versión 4

Numerica

Que debemos probar en esta clase?

- Que el valor del a expresión sea el numero asignado
- Que la representación del a expresión concuerde con el numero asignado

```
public class NumericaTest {

	private Expresion num;
	private Map<Character, Integer> mapa =
		new HashMap<Character, Integer>();

	@Before
	public void setUp() throws Exception {
		num = new Numerica(7);
	}

	@Test
	public void testValor() throws Exception {
		int val = num.getValor(mapa);
		assertEquals(val, 7);
	}

	@Test
	public void testRepresent() throws Exception {
		String rep = num.getRepresent();
		assertEquals(rep, "7");
	}
}
```

Variable

Que debemos probar en esta clase?

- Que el valor del a expresión sea la indicada en el mapa de parametros
- Que la representación de la expresión concuerde con la letra asignada a la misma
- Que al tratar de calcular el valor sin pasar la variable en el mapa de parámetros se lance una FaltaVariableException

```
public class VariableTest {

	private Expresion exp;
	private Map<Character, Integer> mapa;

	@Before
	public void setUp() throws Exception {
		exp = new Variable('a');
		mapa = new HashMap<Character, Integer>();
	}

	@Test
	public void testValor() throws Exception {
		mapa.put('a', 4);
		int val = exp.getValor(mapa);
		assertEquals(val, 4);
	}

	@Test
	public void testRepresent() throws Exception {
		String rep = exp.getRepresent();
		assertEquals(rep, "a");
	}

	/**
	 * Este metodo tiene que lanzar FaltaVariableException
	 * si la lanza da verde, sino falla
	 */
	@Test(expected = FaltaVariableException.class)
	public void testFaltaVariableException() throws Exception {
		mapa = new HashMap<Character, Integer>();
		exp.getValor(mapa);
	}
```

Suma

Que debemos probar en esta clase?

- Que el valor del a expresión sea la suma de sus 2 elementos
- Que la representación de la expresión sea (ELEM1 + ELEM2)

```
public class SumaTest {
	private Expresion exp;
	private Map<Character, Integer> mapa;

	@Before
	public void setUp() throws Exception {
		exp = new Suma(new Numerica (2), new Numerica (5));
		mapa = new HashMap<Character, Integer>();
	}

	@Test
	public void testValor() throws Exception {
		int val = exp.getValor(mapa);
		assertEquals(val, 7);
	}

	@Test
	public void testRepresent() throws Exception {
		String rep = exp.getRepresent();
		assertEquals(rep, "(2 + 5)");
	}
}
```

Resta

Que debemos probar en esta clase?

- Que el valor del a expresión sea la resta de sus 2 elementos
- Que la representación de la expresión sea (ELEM1 - ELEM2)

```
public class RestaTest {
	private Expresion exp;
	private Map<Character, Integer> mapa;

	@Before
	public void setUp() throws Exception {
		exp = new Resta(new Numerica (4), new Numerica (1));
		mapa = new HashMap<Character, Integer>();
	}

	@Test
	public void testValor() throws Exception {
		int val = exp.getValor(mapa);
		assertEquals(val, 3);
	}

	@Test
	public void testRepresent() throws Exception {
		String rep = exp.getRepresent();
		assertEquals(rep, "(4 - 1)");
	}
}
```

Multiplicación

Que debemos probar en esta clase?

- Que el valor del a expresión sea la multiplicación de sus 2 elementos
- Que la representación de la expresión sea (ELEM1 * ELEM2)

```
public class MultiplicacionTest {
	private Expresion exp;
	private Map<Character, Integer> mapa;

	@Before
	public void setUp() throws Exception {
		exp = new Multiplicacion(new Numerica (8), new Numerica
(5));
		mapa = new HashMap<Character, Integer>();
	}

	@Test
	public void testValor() throws Exception {
		int val = exp.getValor(mapa);
		assertEquals(val, 40);
	}

	@Test
	public void testRepresent() throws Exception {
		String rep = exp.getRepresent();
		assertEquals(rep, "(8 * 5)");
	}
}
```

División

Que debemos probar en esta clase?

- Que el valor del a expresión sea la división de sus 2 elementos
- Que la representación de la expresión sea (ELEM1 / ELEM2)
- Que al realizar una division por 0 se lance una ArithmeticException

```
public class DivisionTest {
	private Expresion exp;
	private Map<Character, Integer> mapa;

	@Before
	public void setUp() throws Exception {
		exp = new Division(new Numerica (8), new Numerica (2));
		mapa = new HashMap<Character, Integer>();
	}

	@Test
	public void testValor() throws Exception {
		int val = exp.getValor(mapa);
		assertEquals(val, 4);
	}

	@Test
	public void testRepresent() throws Exception {
		String rep = exp.getRepresent();
		assertEquals(rep, "(8 / 2)");
	}

	@Test(expected = ArithmeticException.class)
	public void testDivisionBy0() throws Exception {
		exp = new Division(new Numerica (8), new Numerica (0));
		exp.getValor(mapa);
	}

}
```