Ejercicios propuestos pruebas

1. Realizar todos los test de prueba necesarios para tener una cobertura del 100% en las clases Subscripcion y OperadorArimetico. Comprobarlo utilizando la herramienta EclEmma.

SubscripcionTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class SubscripcionTest {
	@Test
	public void testprecioPorMes() {
		double esperado = 100;
		Subscripcion s = new Subscripcion(200, 2);
		double resultado = s.precioPorMes();
		assertEquals(esperado, resultado, 0);
	}

	@Test
	public void testprecioPorMes2() {
		double esperado = 67;
		Subscripcion s = new Subscripcion(200, 3);
		double resultado = s.precioPorMes();
		assertEquals(esperado, resultado, 0);
	}

	@Test
	public void testprecioPorMes3() {
		double esperado = 0;
		Subscripcion s = new Subscripcion(0, 1);
		double resultado = s.precioPorMes();
		assertEquals(esperado, resultado, 0);
	}

	@Test
	public void testprecioPorMes4() {
		double esperado = 0;
		Subscripcion s = new Subscripcion(1, 0);
		double resultado = s.precioPorMes();
		assertEquals(esperado, resultado, 0);
	}

	@Test
	public void testprecioPorMes5() {
		Subscripcion a = new Subscripcion(1, 2);
		a.cancel();
		assertTrue(true);
	}
}
```

OperadorAritmeticoTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class OperadorAritmeticoTest {
	@Test
	public void suma() {
		int esperado=8;
		int a = 5;
		int b = 3;
		int suma = OperadorAritmetico.suma(a, b);
		assertEquals(esperado, suma);
	}

	@Test
	public void divisionConCero() {
		int a = 8;
		int b = 0;
		int division;
		try {
			division = OperadorAritmetico.division(a, b);
			fail("Debería haber entrado al catch");
		} catch (Exception e) {
			assertTrue(true);
		}
	}

	@Test
	public void division() {
		int esperado=2;
		int a = 8;
		int b = 4;
		int division;
		try {
			division = OperadorAritmetico.division(a, b);
			assertEquals(esperado, division);
		} catch (Exception e) {
			fail();
		}
	}
}
```

2. Implementar una clase **Pila** donde para poder introducir los elementos deban **cumplir alguna condición**. Cada uno debe proponer su propia clase Pila. Ejemplos de Pilas:

- Una pila que solo incluya números de 3 cifras mayores que cero.
- Una pila que solo incluya números múltiplos de 5.
- Una pila que sólo incluya Personas mayores de edad. 3.

Las operaciones que al menos debe tener la Pila son: **push**, **pop**, **isEmpty** y **top**.
Se propone realizar todos los test de prueba necesarios para tener una cobertura del 100% sobre la Pila creada.

Solución:

Vamos a definir una Pila donde los elementos deban ser enteros comprendidos entre 2 y 20. Su implementación es la siguiente:

```
public class Pila {
	private Stack<Integer> pila = new Stack<Integer>();

	// Añade el número solo si es mayor que 2 y menor que 20
	public void push (Integer num) {
		if (num > 2 && num < 20)
			pila.push(num);
	}

	/*
	 * Si la pila está vacía devolvemos null, sino se devuelve el
último número añadido a la pila sacándolo de la pila
	 */
	public Integer pop() {
		if (pila.isEmpty())
		//	System.out.println("null");
			return null;
		 else
			return pila.pop();
	}

	 //Devuelve true si la pila está vacía, false caso contrario
	public boolean isEmpty() {
		return pila.isEmpty();
	}

	/*
	 * Si la pila está vacía devolvemos null, sino se devuelve el
primer número añadido a la pila sin sacarlo
	 */
	public Integer top() {
		if (pila.isEmpty())
			return null;
		  else
			return pila.peek();
	}}
```

Las Tablas de clases de equivalencia y la obtención de los casos de prueba para cada método son las siguientes:

PUSH

```
public void push (Integer num) {
	if (num > 2 && num < 20)
		pila.push(num);
}
```

CONDICIÓN DE ENTRADA	CLASE DE EQ. VÁLIDA	CLASE DE EQ. INVÁLIDA
La pila está vacía y el elemento cumple la condición	Pila $\in \emptyset$ y num $\in [3,19]$ 1	------
La pila está vacía y el elemento no cumple la condición	Pila $\in \emptyset$ y num <3 2 Pila $\in \emptyset$ y num >19 3	------
La pila no está vacía y el elemento cumple la condición	Pila $\in Z$ y num $\in [3,19]$ 4	------
La pila no está vacía y el elemento no cumple la condición	Pila $\in Z$ y num <3 5 Pila $\in Z$ y num >19 6	------
El dato es de tipo Integer	num $\in Z$ 7	num = null 8

VALOR DE ENTRADA Y ESTADO	CLASE DE EQ. CUBIERTA	RESULTADO
[], 10	1,7	[10]
[], 1	2,7	[]
[], 30	3,7	[]
[5,8], 10	4,7	[5,8,10]
[5,8], 1	5,7	[5,8]
[5,8], 30	6,7	[5,8]
null	8	Exception

POP:

```
public Integer pop() {
		if (pila.isEmpty()) {
			return null;
		} else {
			return pila.pop();
		}
	}
```

CONDICIÓN DE ENTRADA	CLASE DE EQ. VÁLIDA	CLASE DE EQ. INVÁLIDA
La pila está vacía	Pila $\in \emptyset$ 1	
La pila no está vacía	Pila $\in \mathbb{Z}$ 2	

VALOR DE ENTRADA Y ESTADO	CLASE DE EQ. CUBIERTA	RESULTADO
[], -	1	null
[5,8], -	2	8

ISEMPTY:

```
public boolean isEmpty() {
		return pila.isEmpty();
	}
```

CONDICIÓN DE ENTRADA	CLASE DE EQ. VÁLIDA	CLASE DE EQ. INVÁLIDA
La pila está vacía	Pila $\in \emptyset$ 1	
La pila no está vacía	Pila $\in \mathbb{Z}$ 2	

VALOR DE ENTRADA Y ESTADO	CLASE DE EQ. CUBIERTA	RESULTADO
[], -	1	true
[5,8], -	2	false

TOP:

```
public Integer top() {
	if (pila.isEmpty())
		return null;
	  else
		return pila.peek();
	}
```

CONDICIÓN DE ENTRADA	CLASE DE EQ. VÁLIDA	CLASE DE EQ. INVÁLIDA
La pila está vacía	Pila $\in \emptyset$ 1	
La pila no está vacía	Pila $\in \mathbb{Z}$ 2	

VALOR DE ENTRADA Y ESTADO	CLASE DE EQ. CUBIERTA	RESULTADO
[], -	1	null
[5,8], -	2	8

PilaTest.java

```
import org.junit.Test;
import junit.framework.TestCase;

public class PilaTest extends TestCase {
	protected Pila p;

	@Override
	protected void setUp() {
		try {
		super.setUp();
		p = new Pila();
		p.push(5);
		p.push(8);
		} catch (Exception e) {

		}
	}

	/*
	 * No añade elemento (menor que 2) a una pila con 2 elementos
previos
	 */
	@Test
	public void testPushNoAnade1() {
		//Pila p = new Pila();
		p.push(1);
		assertTrue(p.cuantos()==2);

	}

	/*
	 * No añade elemento (mayor que 20) a una pila con 2 elementos
previos
	 */
	@Test
	public void testPushNoAnade2() {
		//Pila p = new Pila();
		p.push(30);
		assertTrue(p.cuantos()==2);

	}

	/*
	 * Añade elemento a una pila con 2 elementos previos
	 */
	@Test
	public void testPushAnade() {
		p.push(10);
		assertTrue(p.cuantos()==3);
```

```
	}

	/*
	 * No añade elemento (menor que 2) a una lista con 0 elementos
previos
	 */
	@Test
	public void testPushNoAnadeVacio1() {
		Pila p = new Pila();
		p.push(1);
		assertTrue(p.isEmpty());
	}

	/*
	 * No añade elemento (mayor que 20) a una lista con 0 elementos
previos
	 */
	@Test
	public void testPushNoAnadeVacio2() {
		Pila p = new Pila();
		p.push(30);
		assertTrue(p.isEmpty());
	}

	/*
	 * Añade elemento a una lista con 0 elementos previos
	 */
	@Test
	public void testPushAnadeVacio() {
		Pila p = new Pila();
		p.push(10);
		assertTrue(p.cuantos()==1);
	}


	/*
	 * Ocurre una excepción cuando no entra null como parámetro
	 */
	@Test
	public void testPushNull () {
		try {
			p.push(null);
			fail();
		} catch (Exception e) {
			assertTrue(true);
		}
	}

	/*
	 * Extrae último elemento de la pila no vacía
	 */
	@Test
```

```
	public void testPop() {
		Integer esperado = 8;
		Integer tmp = p.pop();
//		assertEquals(tmp, esperado);
		assertTrue(tmp.equals(esperado) && p.cuantos() == 1);
	}

	/*
	 * Extrae último elemento de la pila vacía, es decir, devuelve
null
	 */
	@Test
	public void testPopVacio() {
		Pila p = new Pila();
		Integer esperado = null;
		Integer tmp = p.pop();
	//	assertEquals(tmp, esperado);
//		System.out.println(tmp);
		assertTrue(tmp==esperado && p.isEmpty());
	}

	/*
	 * La pila no está vacía
	 */
	@Test
	public void testIsEmpty() {
		assertTrue(!p.isEmpty());
	}

	/*
	 * La pila está vacía
	 */
	@Test
	public void testIsEmptyVacio() {
		Pila p = new Pila();
		assertTrue(p.isEmpty());
	}

	/*
	 * Muestra (sin extraer) último elemento de la pila no vacía
	 */
	@Test
	public void testTop() {
		Integer esperado = 8;
		int tamanoEsperado = 2;
	//	assertEquals(p.top(),esperado);
		assertTrue(p.top().equals(esperado) && p.cuantos() ==
tamanoEsperado);
	}

	/*
	 * Muestra (sin extraer) último elemento de la pila vacía, que
```

```
es null
	 */
	@Test
	public void testTopVacio() {
		Pila p = new Pila();
		Integer esperado = null;
		int tamanoEsperado = 0;
		assertTrue(p.top()==esperado && p.cuantos() ==
tamanoEsperado);
	}

}
```