Solution MathBean JUnit

```
package item6_bean_JUnit5.test;

import static org.junit.Assert.assertEquals;import static
org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;import static
org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.AfterAll;import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import item3_5_recursif_mieux.DEMO2_CalculFactorielle;
import item6_bean.MathBean;

class MathBeanTest {

static MathBean bean; //objet visible de tt les méthodes static

@BeforeAll
static void setUpBeforeClass() throws Exception {
        bean = new MathBean();//bean stateless
}

@AfterAll
static void tearDownAfterClass() throws Exception {
        bean = null;
}

@BeforeEach
void setUp() throws Exception {
}

@AfterEach
void tearDown() throws Exception {
}

//      @Test
//      void testFactorielle() {
//              fail("Not yet implemented");
//      }
@Test //test unitaire
public void testFactorielle1() {
        assertFalse("La factorielle de zero vaut 1",bean.factorielle(0) != 1);
}
@Test public void testFactorielle2() {
        assertTrue("La factorielle de 1 vaut 1",bean.factorielle(1) == 1);
}
@Test public void testFactorielle3() {
        assertEquals("La factorielle de 3 vaut 6",bean.factorielle(3), 6);
}
@Test public void testFactorielle5() {
        assertEquals("La factorielle de 14 vaut
87178291200L",bean.factorielle(14), 87178291200L);
}
```

```
//"La factorielle de 21 ne peut pas être calculé"
//@Test
//public void testFactorielle11() {
//    assertThrows(IllegalArgumentException.class,
//            ()->{DEMO2_CalculFactorielle.factorielle(21);},
//            "La factorielle de 21 ne peut pas être calculé" );
//}
@Test
public void testFactorielle11() {
    assertThrows(IllegalArgumentException.class,
            ()->{bean.factorielle(21);},
            "La factorielle de 21 ne peut pas être calculé" );
}
//"La factorielle de -1 ne peut pas être calculé (hors limite)"
@Test
public void testFactorielle12() {
    assertThrows(IllegalArgumentException.class,
            ()->{bean.factorielle(-1);},
            "La factorielle de -1 ne peut pas être calculé (hors limite)" );
}

//"La factorielle de 77 ne peut pas être calculé"
@Test
public void testFactorielle13() {
    assertThrows(IllegalArgumentException.class,
            ()->{bean.factorielle(77);},
            "La factorielle de 77 ne peut pas être calculé" );
}

////////////////////////////////////////////////////
////////////////////////////////////////////////////
//@Test
//void testDivision17() {
//      fail("Not yet implemented");
//}

@Test public void testDivision17_1() {
assertEquals("La division de 17 par 2 vaut 8", 8, bean.division17("2"));
//fail("Not yet implemented");
}
@Test public void testDivision17_2() {
assertTrue("La division de 17 par 8 vaut 2",  bean.division17("8") == 2);
//fail("Not yet implemented");
}
@Test public void testDivision17_3() {
assertFalse("La division de 17 par 3 vaut 5",  bean.division17("3") != 5);
//fail("Not yet implemented");
}

//@Test(expected=IllegalArgumentException.class)
//public void testDivision17_4() {
```