```java
package fr.manulep.entrainement;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertThrows;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;

/**
 * Training : condition, loop, array
 *
 * notes to read assert : - use of JUnit 5 - the first parameter is "what you
 * must have as a result" - the second parameter is the call of the method
 *
 * @author manulep
 *
 */

@DisplayName("Series 1")
class Series1Test

{

    /**
     * Hello World
     */
    @Test
    void hello() {
        assertEquals("Hello World", Series1.helloWorld(null), "null name");
        assertEquals("Hello World", Series1.helloWorld(""), "blank name");
        assertEquals("Hello Manu", Series1.helloWorld("Manu"), "Manu name");
    }

    /**
     * removes null values from an array
     */
    @Test
    public void RemoveNullElements() {
        String arrayIn[] = { "a", "b", null, null, "false", "null" };
        String arrayExpected[] = { "a", "b", "false", "null" };
        assertArrayEquals(arrayExpected, Series1.removeNullElements(arrayIn));

        String arrayIn2[] = { null, null };
        String arrayExpected2[] = {};
        assertArrayEquals(arrayExpected2, Series1.removeNullElements(arrayIn2));
    }
```

```java
/**
     * adds an element to the beginning of an array
     */
    @Test
    public void addElementToBeginning() {
        int arrayIn[] = { 2, 3, 4, 5 };
        int arrayExpected[] = { 1, 2, 3, 4, 5 };
        assertArrayEquals(arrayExpected, Series1.addElementToBeginning(arrayIn, 1));

        int arrayIn2[] = {};
        int arrayExpected2[] = { 1 };
        assertArrayEquals(arrayExpected2, Series1.addElementToBeginning(arrayIn2, 1));
    }

    /**
     * takes all elements except the first 3
     */
    @Test
    public void allElementsExceptFirstThree() {

        int arrayIn[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
        int arrayExpected[] = { 4, 5, 6, 7, 8 };
        assertArrayEquals(arrayExpected, Series1.allElementsExceptFirstThree(arrayIn));

        int arrayIn2[] = { 12, 15 };
        int arrayExpected2[] = {};
        assertArrayEquals(arrayExpected2, Series1.allElementsExceptFirstThree(arrayIn2));

        assertThrows(NullPointerException.class, () -> {
            Series1.allElementsExceptFirstThree(null);
        });
    }

    /**
     * gets the first half of a string
     */
    @Test
    public void getFirstHalf() {
        assertEquals("dra", Series1.getFirstHalf("dragon"));
        assertEquals("sna", Series1.getFirstHalf("snake"));
    }

    /**
     * selects elements starting with a
     */
    @Test
    void selectElementsStartingWithA() {
        String[] fruits = { "bananas", "apples", "pears", "avocados" };
        String[] fruitsStartWithA = { "apples", "avocados" };
        assertArrayEquals(fruitsStartWithA, Series1.selectElementsStartingWithA(fruits));
    }
```

```java
/**
     * selects elements starting with a vowel
     */
    @Test
    void selectElementsStartingWithVowel() {
        String arrayIn[] = { "john", "david", "omar", "fred", null, "idris", "angela" };
        String arrayExpected[] = { "omar", "idris", "angela" };
        assertArrayEquals(arrayExpected, Series1.selectElementsStartingWithVowel(arrayIn))
;
    }

    /**
     * reverses the order of each element in an array
     */
    @Test
    public void reverseOrderInArray() {
        String arrayIn[] = { "dog", "monkey", "elephant", "kayak" };
        String arrayExpected[] = { "kayak", "elephant", "monkey", "dog" };
        assertArrayEquals(arrayExpected, Series1.reverseOrderInArray(arrayIn));
    }

    /**
     * Insert element in array's middle
     */
    @Test
    public void insertElementInTheMiddleOfAnArray() {
        assertArrayEquals(new int[] { 2, 3, 2, 0, -7, 4, 11, 6, 8 },
                Series1.insertElementInTheMiddleOfAnArray(new int[] { 2, 3, 2, 0, 4, 11, 6
, 8 }, -7));

        assertArrayEquals(new int[] { 9, 8, 3, 11 },
                Series1.insertElementInTheMiddleOfAnArray(new int[] { 9, 3, 11 }, 8));

        assertArrayEquals(new int[] { 2, 4, 3 }, Series1.insertElementInTheMiddleOfAnArray
(new int[] { 2, 3 }, 4));

        assertArrayEquals(new int[] { 13 }, Series1.insertElementInTheMiddleOfAnArray(new
int[] {}, 13));
    }

    /**
     * returns the shortest word
     */
    @Test
    public void shortestWord() {
        String text = "winter is coming";
        assertEquals("is", Series1.shortestWord(text));
    }
```

```java
/**
     * removes capital letters from a string
     */
    @Test
    public void removeCapitals() {
        String text = "Hello Kitty";
        assertEquals("ello itty", Series1.removeCapitals(text));
    }

    @Test
    public void addingTwoNumbers() {
        assertEquals(30, Series1.addingTwoNumbers(10, 20));
    }

    @Test
    public void addingThreeNumbers() {
        assertEquals(60, Series1.addingThreeNumbers(10, 20, 30));
    }

    @Test
    public void addingSeveralNumbers() {
        assertEquals(30, Series1.addingSeveralNumbers(10, 20));
        assertEquals(60, Series1.addingSeveralNumbers(10, 20, 30));
        assertEquals(7936, Series1.addingSeveralNumbers(256, 512, 1024, 2048, 4096));
    }

    /**
     * makes numbers negative
     */
    @Test
    public void makeNegative() {
        assertEquals((float) -4.52, Series1.makeNegative((float) 4.52), 0);
        assertEquals((float) -8, Series1.makeNegative((float) -8), 0);
    }

    /**
     * checks a string for special characters
     */
    @Test
    public void checkForSpecialCharacters() {
        assertFalse(Series1.checkForSpecialCharacters("joker"));
        assertTrue(Series1.checkForSpecialCharacters("ABC@dsklfj!"));
        assertTrue(Series1.checkForSpecialCharacters("#special"));
        assertFalse(Series1.checkForSpecialCharacters("finish!"));
        assertFalse(Series1.checkForSpecialCharacters("code"));
        assertTrue(Series1.checkForSpecialCharacters("%code%"));
    }
    /**
     * checks start with consonant
     */
    @Test
    public void checkIfStringStartsWithConsonant() {

        assertEquals(true, Series1.checkIfStringStartsWithConsonant("qwerty"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("assert"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("Art"));
        assertEquals(true, Series1.checkIfStringStartsWithConsonant("Fola"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("Olaf"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("Export"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("unity"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("Unit"));
        assertEquals(false, Series1.checkIfStringStartsWithConsonant("END"));
    }

    @Test
    public void getDomainName() {
        assertEquals("makersacademy", Series1.getDomainName("spike@makersacademy.com"));
        assertEquals("ssh.makersacademy", Series1.getDomainName("spike@ssh.makersacademy.c
om"));
    }

    /**
     * for each letter, find its position in the alphabet
     */
    @Test
    public void letterPosition() {
        int[] result = { 8, 5, 12, 12, 15, 11, 9, 20, 20, 25 };
        assertArrayEquals(result, Series1.letterPosition("HelloKitty"));
    }

    /**
     * is peer ?
     */
    @Test
    public void checkIfPeer() {
        assertEquals(true, Series1.isPeer(10));
        assertEquals(false, Series1.isPeer(21));
    }


}
```