



Ya que mi código no llegaba a obtener el 100% de cobertura por culpa de no hacer tests para los métodos getters, le pregunté para saber su opinión al respecto. Al igual que él, opinaba que no eran necesarios pero cierto que es son útiles para la documentación.




¿Hay que probar los metodos getters con pruebas junit?




No es necesario realizar pruebas JUnit para los métodos getter, ya que estos solo devuelven valores y no realizan ninguna acción compleja. Sin embargo, es común escribir pruebas para ellos como una forma de asegurarse de que los valores se estén almacenando y recuperando correctamente. También puede ser útil para documentar la funcionalidad de la clase y mejorar la cobertura de pruebas en general.

A continuación decidí preguntar por si se le ocurrían pruebas junit que yo ignoraba. Alguna como las 2,7,8 las veía poco útiles pero la idea de la 4 (en nuestro caso edades negativas) me pareció buena y decidí implementarla.



Ideas of junit test to a function of an array as parameter



Here are a few ideas for JUnit tests for a function that takes an array as a parameter:

1. Test the function with an empty array: In this test, you would pass an empty array to the function and check that the function returns the expected result when given an empty array.
2. Test the function with an array of the minimum allowed size: This test would check the behavior of the function when given an array of the smallest size allowed by the function's parameters.
3. Test the function with an array of the maximum allowed size: This test would check the behavior of the function when given an array of the largest size allowed by the function's parameters.
4. Test the function with an array containing negative numbers: This test would check how the function handles negative numbers in the array.
5. Test the function with an array containing duplicates: This test would check how the function handles duplicate values in the array.
6. Test the function with an array containing a mix of positive and negative numbers: This test would check how the function handles arrays with a mix of positive and negative numbers.
7. Test the function with an array sorted in ascending order: This test would check how the function behaves when given an array that is sorted in ascending order.
8. Test the function with an array sorted in descending order: This test would check how the function behaves when given an array that is sorted in descending order.
9. Test the function with an array containing a mix of different data types: This test would check

Además, una vez realizado los test que creí conveniente, decidí pasarle el código de mi Person.java para ver que pruebas junit me realizaba. Para mi sorpresa generó algunas de las que ya poseía pero ninguna nueva.

```
class PersonTest {

    @Test
    void testAverageAgePerGender_WithNullList_ShouldThrowException() {
        List<Person> persons = null;
        Exception exception = assertThrows(NullPointerException.class, () -> {
            Person.averageAgePerGender(persons);
        });
        assertEquals("Error, null as persons", exception.getMessage());
    }

    @Test
    void testAverageAgePerGender_WithEmptyList_ShouldReturnZeroArray() {
        List<Person> persons = new ArrayList<>();
        double[] result = Person.averageAgePerGender(persons);
        assertEquals(new double[] {0, 0}, result);
    }

    @Test
    void testAverageAgePerGender_WithAllMalePersons_ShouldReturnCorrectResult() {
        List<Person> persons = new ArrayList<>();
        persons.add(new Person("John", 30, "Male"));
        persons.add(new Person("Jack", 40, "Male"));
        persons.add(new Person("Jim", 35, "Male"));
        double[] result = Person.averageAgePerGender(persons);
        assertEquals(new double[] {35, 0}, result, 0.0001);
    }

    @Test
    void testAverageAgePerGender_WithAllFemalePersons_ShouldReturnCorrectResult() {
        List<Person> persons = new ArrayList<>();
        persons.add(new Person("Jane", 25, "Female"));
        persons.add(new Person("Jill", 28, "Female"));
        persons.add(new Person("Joan", 32, "Female"));
        double[] result = Person.averageAgePerGender(persons);
        assertEquals(new double[] {0, 28.666666}, result, 0.0001);
    }

    @Test
    void testAverageAgePerGender_WithMixedGenders_ShouldReturnCorrectResult() {
        List<Person> persons = new ArrayList<>();
        persons.add(new Person("John", 30, "Male"));
        persons.add(new Person("Jack", 40, "Male"));
        persons.add(new Person("Jim", 35, "Male"));
        persons.add(new Person("Jane", 25, "Female"));
        persons.add(new Person("Jill", 28, "Female"));
        persons.add(new Person("Joan", 32, "Female"));
        double[] result = Person.averageAgePerGender(persons);
        assertEquals(new double[] {35, 28.666666}, result, 0.0001);
    }
}
```