

Uso de ChatGPT en el ejercicio 1

Mejora de la calidad del código:

Pregunta

How the following code may be improved?:

```
public double[] averageAgePerGender(List<Person> persons) { double maleAgeSum = 0, femaleAgeSum = 0; int maleCount = 0, femaleCount = 0; for (Person person : persons) { if (person.gender().equals("Male")) { maleAgeSum += person.age(); maleCount++; } else if (person.gender().equals("Female")) { femaleAgeSum += person.age(); femaleCount++; } } return new double[]{maleAgeSum / maleCount, femaleAgeSum / femaleCount}; }
```

Respuesta

Here are some ways the code can be improved:

1. Add null checks: The code should include checks to make sure that `persons` is not null, and that the `gender` and `age` methods return non-null values.
2. Use a more meaningful variable name: Instead of using `maleAgeSum` and `femaleAgeSum`, you could use `maleSum` and `femaleSum` to make the code more readable.
3. Check for divide-by-zero: If the number of male or female persons is 0, the code will result in a divide-by-zero error. To avoid this, you should add checks to make sure that the `maleCount` and `femaleCount` variables are greater than 0 before performing the division.

Mejora de la documentación

Pregunta

Complete the documentation of the previous code given this as basis:

```
/* * Computes the average age of male and female persons in a list and returns the result in as array of two elements * (the first element is the male mean age and the second one is the female mean age) * * @param persons * @return */
```

Respuesta

```
/**
```

```
* Computes the average age of male and female persons in a list and returns the result as an array of two elements.
```

```
* The first element is the mean age of males and the second one is the mean age of females.
```

```
*
```

```
* @param persons A list of persons to compute the average age for. It must not be null.
```

```
* @return An array of two elements where the first element is the mean age of males and the second one is the mean age of females.
```

```
* @throws IllegalArgumentException if the `persons` list is null.
```

```
*/
```

```
public double[] averageAgePerGender(@NotNull List<Person> persons) {
```

```
    ...
```

```
}
```

Creación de casos de prueba

Pregunta

Create some test cases in JUnit5 for the previous code

Respuesta

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

```
import java.util.Arrays;
```

```
import java.util.Collections;
```

```
import java.util.List;
```

```
import static org.junit.jupiter.api.Assertions.assertArrayEquals;
```

```
import static org.junit.jupiter.api.Assertions.assertThrows;
```

```
class AverageAgePerGenderTest {
```

```
private AverageAgePerGender classUnderTest = new AverageAgePerGender();
```

```
@Test
```

```
void averageAgePerGender_nullList_throwsException() {  
    assertThrows(IllegalArgumentException.class, () ->  
classUnderTest.averageAgePerGender(null));  
}
```

```
@Test
```

```
void averageAgePerGender_emptyList_returnsZeroArray() {  
    double[] result = classUnderTest.averageAgePerGender(Collections.emptyList());  
    assertEquals(new double[]{0, 0}, result);  
}
```

```
@Test
```

```
void averageAgePerGender_allMales_returnsMaleAverage() {  
    List<Person> persons = Arrays.asList(  
        new Person("John", "Male", 20),  
        new Person("Jim", "Male", 25),  
        new Person("Jack", "Male", 30)  
    );  
    double[] result = classUnderTest.averageAgePerGender(persons);  
    assertEquals(new double[]{25, 0}, result);  
}
```

```
@Test
```

```
void averageAgePerGender_allFemales_returnsFemaleAverage() {  
    List<Person> persons = Arrays.asList(  
        new Person("Jane", "Female", 20),  
        new Person("Jill", "Female", 25),  
        new Person("Janet", "Female", 30)  
    );  
}
```

```
);  
double[] result = classUnderTest.averageAgePerGender(persons);  
assertArrayEquals(new double[]{0, 25}, result);  
}
```

@Test

```
void averageAgePerGender_mixedGenders_returnsBothAverages() {  
    List<Person> persons = Arrays.asList(  
        new Person("John", "Male", 20),  
        new Person("Jane", "Female", 25),  
        new Person("Jim", "Male", 30),  
        new Person("Janet", "Female", 35)  
    );  
    double[] result = classUnderTest.averageAgePerGender(persons);  
    assertArrayEquals(new double[]{25, 30}, result);  
}  
}
```

Añadiendo documentación a los casos de prueba

Pregunta

Add documentation for the following tests: @Test void

```
averageAgePerGender_nullList_throwsException() { assertThrows(IllegalArgumentException.class,
() -> Person.averageAgePerGender(null)); } @Test void
averageAgePerGender_emptyList_returnsZeroArray() { assertEquals(new double[]{0, 0},
Person.averageAgePerGender(Collections.emptyList())); } @Test void
averageAgePerGender_allMales_returnsMaleAverage() { List<Person> persons = Arrays.asList( new
Person("John", 20, "Male"), new Person("Jim", 25, "Male"), new Person("Jack", 30, "Male") );
assertEquals(new double[]{25, 0}, Person.averageAgePerGender(persons)); } @Test void
averageAgePerGender_allFemales_returnsFemaleAverage() { List<Person> persons =
Arrays.asList( new Person("Jane", 20, "Female"), new Person("Jill", 25, "Female"), new
Person("Janet", 30, "Female") ); assertEquals(new double[]{0, 25},
Person.averageAgePerGender(persons)); } @Test void
averageAgePerGender_mixedGenders_returnsBothAverages() { List<Person> persons =
Arrays.asList( new Person("John", 20, "Male"), new Person("Jane", 25, "Female"), new
Person("Jim", 30, "Male"), new Person("Janet", 35, "Female") ); assertEquals(new double[]
{25, 30}, Person.averageAgePerGender(persons)); } @Test void
averageAgePerGender_nullPersonInTheList_throwsException() { List<Person> persons =
Arrays.asList( new Person("John", 20, "Male"), null, new Person("Jim", 30, "Male"), new
Person("Janet", 35, "Female") ); assertThrows(IllegalArgumentException.class, () ->
Person.averageAgePerGender(persons)); }
```

Respuesta

```
/**
```

```
 * Test case for the scenario when the input list is null.
```

```
 * It is expected to throw an IllegalArgumentException.
```

```
*/
```

```
@Test
```

```
void averageAgePerGender_nullList_throwsException() {
```

```
    assertThrows(IllegalArgumentException.class, () -> Person.averageAgePerGender(null));
```

```
}
```

```
/**
```

```
 * Test case for the scenario when the input list is empty.
```

```
 * It is expected to return an array of two zeros.
```

```
*/
```

```
@Test
```

```
void averageAgePerGender_emptyList_returnsZeroArray() {
```

```
    assertEquals(new double[]{0, 0}, Person.averageAgePerGender(Collections.emptyList()));
```

```
}
```

```
/**
```

```
 * Test case for the scenario when all the persons in the input list are male.
```

```
 * It is expected to return an array with the average age of all males as the first element and zero  
as the second element.
```

```
*/
```

```
@Test
```

```
void averageAgePerGender_allMales_returnsMaleAverage() {
```

```
    List<Person> persons = Arrays.asList(
```

```
        new Person("John", 20, "Male"),
```

```
        new Person("Jim", 25, "Male"),
```

```
        new Person("Jack", 30, "Male")
```

```
    );
```

```
    assertEquals(new double[]{25, 0}, Person.averageAgePerGender(persons));
```

```
}
```

```
/**
```

```
 * Test case for the scenario when all the persons in the input list are female.
```

```
 * It is expected to return an array with zero as the first element and the average age of all  
females as the second element.
```

```
*/
```

```
@Test
```

```
void averageAgePerGender_allFemales_returnsFemaleAverage() {
```

```
    List<Person> persons = Arrays.asList(
```

```
        new Person("Jane", 20, "Female"),
```

```
        new Person("Jill", 25, "Female"),
```

```

        new Person("Janet", 30, "Female")

    );

    assertEquals(new double[]{0, 25}, Person.averageAgePerGender(persons));
}

/**
 * Test case for the scenario when the input list has a mix of male and female persons.
 * It is expected to return an array with the average age of all males as the first element and the
 * average age of all females as the second element.
 */
@Test
void averageAgePerGender_mixedGenders_returnsBothAverages() {

    List<Person> persons = Arrays.asList(
        new Person("John", 20, "Male"),
        new Person("Jane", 25, "Female"),
        new Person("Jim", 30, "Male"),
        new Person("Janet", 35, "Female")
    );

    assertEquals(new double[]{25, 30}, Person.averageAgePerGender(persons));
}

/**
 * Test case for the scenario when the input list contains a null person.
 * It is expected to throw an IllegalArgumentException.
 */
@Test
void averageAgePerGender_nullPersonInTheList_throwsException() {

    List<Person> persons = Arrays.asList(
        new Person("John", 20, "Male"),
        null,

```

```
        new Person("Jim", 30, "Male"),
        new Person("Janet", 35, "Female")
    );
    assertThrows(IllegalArgumentException.class, () -> Person.averageAgePerGender(persons));
}
```

Reflexiones sobre el uso de ChatGPT en la práctica

Pese a que se trata de una gran herramienta que me ha sido de gran utilidad en el desarrollo de la práctica, se debe seguir siendo crítico con sus respuestas ya que, en algunos casos estas contienen pequeños errores, incoherencias o están incompletas. Es por eso por lo que se debe evitar copiar ciegamente las respuesta que nos da esta IA.