TDD Ejercicio N° 2

UNAHUR - POO2 - 2022

Profesor: Juárez, Andrés

Grupo: Les Trainee

Inicialmente, planteamos el caso de la suma entre 3 y 5

```
@Test
public void sumaDosNumeros(){
    int result = cal.sumar(3,5);
    Assert.assertEquals(8,result);
}
```

Para ello, definimos nuestro Objeto Calculadora el cual posee una función. Donde al recibir 3 y 5, retorna 8. De esta forma, el primer test pasará exitosamente.

```
public int sumar(int numero, int numero2){
    return 8;
}
```

Luego, se agrega otro caso a cumplir como por ejemplo lo es la suma de dos números menores a cero.

```
@Test
public void sumaDosNumerosMenoresACero(){
    int result = cal.sumar(-1,-1);
    Assert.assertEquals(-2,result);
}
```

Para lograr esto, modificamos un poco el código inicial. De esta forma, ahora tenemos el siguiente código desarrollado:

```
public int sumar(int numero, int numeroDos){
    return numero + numeroDos;
}
```

Podemos observar que luego de una iteración, se pasó de tener un valor hardcodeado (para poder cumplir con el caso base) a contemplar algunos otros.

Avanzando un poco más, se plantea el hecho de que la calculadora también pueda sumar arrays, entre ellos uno nulo.

```
@Test
public void sumaArrayNull(){
    //Si no tiene elementos
    int[] arr ={};
    int result = cal.sumarArray(arr);
    Assert.assertEquals(0,result);
}
```

El código para este caso base se define de la siguiente manera:

```
public int sumarArray(int[] arr){
    return 0;
}
```

Por último y con el objetivo de poder sumar el contenido de distintos arreglos, se agregan los siguientes tests unitarios:

```
public void sumaArray0(){
    // Cuando un elemento sea 0
    int[] arr ={0};
    int result = cal.sumarArray(arr);
    Assert.assertEquals(0,result);
}

@Test
public void sumaArray1(){
    //Cuando hay un solo elemento
    int[] arr = {1};
    int result = cal.sumarArray(arr);
    Assert.assertEquals(1,result);
}

@Test
public void sumaArray123(){
    //Varios
    int[] arr = {1,2,3};
    int result = cal.sumarArray(arr);
    Assert.assertEquals(6,result);
}
@Test
public void sumaArrayVarios(){
    //Funcionando
    int[] arr = {10,20,25,50,115};
    int result = cal.sumarArray(arr);
    Assert.assertEquals(220,result);
}
```

Para lograr implementar estos requerimientos se modifica el código de la función sumarArray(ArrayParam).

```
public int sumarArray(int[] arr){
    int result = 0;
    if (!estaVacio(arr)){
        for (int i = 0; i < arr.length; i++) {
            result+= arr[i];
        }
    }
    return result;
}
```

Con esta modificación, ya podemos procesar mas de algunos simples casos particulares de arreglos.

Conclusión:

Puede observarse que partiendo de casos bases y luego de aplicarse cambios requeridos por tests introducidos en posteriores iteraciones, llegamos a lograr refinar y construir un código robusto, eficiente y que implementa exitosamente múltiples valores input.