Rodrigo Perela Posada ITT-IAA

Práctica: Introducción a pruebas de SW (testing)

1. NumZero

1. El bucle del for empieza por un número erróneo, debería ser i = 0, no i = 1 ya que entonces el rimer elemento delarray no es tenido en cuenta.

2. Utilizando los test es imposible no ejecutar dicho código erróneo porque directamente lo llamamos en la función. Si se ejecuta directamente el programa se podría conseguir si hacemos que falle antes, por ejemplo, no introduciendo argumentos.

3. No es posible hacerlo, el error siempre estará porque la variable “i” está mal

4.

```
@Test public void zeroMidElement()
{
    int arr[] = {7, 0, 2};
    assertEquals("Zero in first element", 1, NumZero.numZero(arr));
}
```

Cualquier array que no empiece por cero y que tenga solamente un elemento igual a cero (y de dimensión 2 o más).

5. El primero de los estados erroneos ocurre cuando empieza el for:
   - Se llama a la función numZero
   - count se iguala a 0
   - entra a la llamada de if
   - mira el valor inicial de i que es 1. AQUI ocurre el error

6. OK ✓

2. FindLast

1. El fallo vuelve a estar en el bucle del for, termina antes de comprobar el último valor. Se puede arreglar poniendo i >= 0 en la condición del bucle.

2. Utilizando los test es imposible no ejecutar dicho código erróneo porque directamente lo llamamos en la función. Si se ejecuta directamente el programa se podría conseguir si hacemos que falle antes, por ejemplo, no introduciendo argumentos.

3. No es posible hacerlo, el error siempre estará porque la variable “i” está mal

4. No hay forma de definirte un array que no de failura ya que podrias poner el número buscado en la segunda posición para que lo encontrase el “último” pero entonces te devolvería un 1 en vez de un 0.

5. El primero de los estados erróneos ocurre cuando se llega a la segunda posición del array, que ahí el for acaba dejando el primer término sin comprobar.

6. OK ✓

3. LastZero

1. El fallo está en el bucle for, debería empezar por el final del array y no por el principio ya que como está planteado encuentra el primer cero, no el último. Se arreglaría haciendo el bucle regresivo, que empezara por el final.

2. Utilizando los test es imposible no ejecutar dicho código erróneo porque directamente lo llamamos en la función. Si se ejecuta directamente el programa se podría conseguir si hacemos que falle antes, por ejemplo, no introduciendo argumentos.

3. No es posible hacerlo, el error siempre estará porque el bucle for está mal planteado.

4.

```
    @Test public void oneZero()
{
    int arr[] = {2, 1, 0};
    assertEquals("Multiple zeroes: should find last one", 2,
                 LastZero.lastZero(arr));
}
```

5. El primero de los estados erroneos ocurre cuando empieza el for:
   - Se llama a la función lastZero
   - entra a la llamada de for
   - mira el valor inicial de i que es 0. AQUI ocurre el error

6. OK ✓

4. CountPositive

1. El fallo se encuentra en la condición para ser positivo. Cuenta como positivo si es “mayor o igual” que cero, cosa que es falsa ya que el cero no es positivo. Se arreglaría eliminando el simbolo “=” de dicha comparación.

2. Utilizando los test es imposible no ejecutar dicho código erróneo porque directamente lo llamamos en la función. Si se ejecuta directamente el programa se podría conseguir si hacemos que falle antes, por ejemplo, no introduciendo argumentos.

3. No es posible hacerlo, el error siempre estará porque la comparación está mal planteada.

4.

```
    @Test public void arrayNoContainsZeroes()
{
    int arr[] = {-4, 2, -1, 2};
    assertEquals("Array contains zeroes", 2,
                 CountPositive.countPositive(arr));
}
```

5. El primero de los estados erroneos ocurre cuando empieza el for:
   - Se llama a la función lastZero
   - entra a la llamada de for
   - entra a la comparación del if
   - mira si el valor es mayor o igual que 0. AQUI ocurre el error

6. OK ✓

5. OddOrPos

1. El fallo se encuentra en la condición del if para ser impar o positivo. Da por comprobado que es positivo cuando el resto de dividir el número por dos da como resultado 1, pero eso solo ocurre cuando el número es positivo ya que cuando es negativo el resto es -1. Se podría arreglar cambiando la condición para que compruebe que no es par (`!(x[i]%2 == 0)`, ya que así te ahorras el posible problema de que sea un número negativo ya que lo igualas con 0.

2. Utilizando los test es imposible no ejecutar dicho código erróneo porque directamente lo llamamos en la función. Si se ejecuta directamente el programa se podría conseguir si hacemos que falle antes, por ejemplo, no introduciendo argumentos.

3. No es posible hacerlo, el error siempre estará porque la comparación está mal planteada.

4.

```
@Test public void positiveOddNumbers()
{
    int arr[] = {3, -2, 0, 1, 4};
    assertEquals("Negative odd numbers in array", 3, OddOrPos.oddOrPos(arr));
}
```

5. El primero de los estados erroneos ocurre cuando empieza el for:
   - Se llama a la función oddOrPos
   - entra a la llamada de for
   - entra a la comparación del if
   - mira si el valor del resto de dicha división da 1. AQUI ocurre el error

6. OK ✓