Ejercicios: Introducción a pruebas de SW (testing)

Cayetana Gómez Casado
ITT + IAA

NumZero:

1. 6. Falla al ejecutar NumZero puesto que a continuación llama a NumZero buscando que haya un número de ceros exacto, en este caso 1. Pero encuentra 0.

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

Se debe que al ejecutar el siguiente código no se tiene en cuenta la primera posición del array sino que se comienza a comparar por la segunda, de modo que si el '0' se encuentra en la primera posición nunca lo comparará. Para ello hago el siguiente cambio, "i=0":

```
    public static int numZero (int[] x)
{
    int count = 0;

    for (int i = 0; i < x.length; i++)
    {
       if (x[i] == 0) count++;
    }
    return count;
}
```

2. Con los test es imposible de evitar esa parte del código, al estar ejecutando directamente esa parte del código en el test. Pero sin los test tenemos la opción de que no llegue a llamarlo si lo ejecutamos sin argumentos.

3. No es posible que haya fallo y no error en el código, puesto que en la variable i hay un error y se ejecuta esa parte del código.

4. Cualquier array que no contenga un 0 en su primera posición. {2,4,0} Esto va a hacer que aunque haya un fallo y un error de consistencia en la variable 'i' no provoque un failure.

5. Error: llama a numZero(arr), que al entrar al for comienza por la segunda posición del array y no por la primera, luego comnpara en el if (la primera se queda sin comparar) y devuelve la variable 'count' con el número de 0s que encuentra y compara con un 1.

FindLast:

1. 6. Falla al ejecutar FindLast puesto que a continuación llama a findLast(arr, y) buscando que el valor de y dentro del array y devolviéndote la posición de éste. A continuación, compara esa posición con 0 buscando que esté en la primera del array:

```
    @Test public void lastOccurrenceInFirstElement()
{
    int arr[] = {2, 3, 5};
    int y = 2;
    assertEquals("Last occurence in first element", 0, FindLast.findLast(arr,
                y));
}
```

Se debe que al ejecutar el siguiente código no se tiene en cuenta la primera posición del array sino que se para en la comparación en la segunda posición dejando sin comparar la primera , de modo que si el '2' se encuentra en la primera posición nunca lo comparará. Para ello hago el siguiente cambio, "i>=0":

```
    public static int findLast (int[] x, int y)
{
    // As the example in the book points out, this loop should end at 0.
    for (int i=x.length-1; i >= 0; i--)
    {
        if (x[i] == y)
        {
            return i;
        }
    }
    return -1;
}
```

2. Con los test es imposible de evitar esa parte del código, al estar ejecutando directamente esa parte del código en el test. Pero sin los test tenemos la opción de que no llegue a llamarlo si lo ejecutamos sin argumentos.

3. No es posible que haya fallo y no error en el código, puesto que en la variable i hay un error y se ejecuta esa parte del código.

4. No es posible que no haya failure puesto que nunca llega a la primera posición del array para comparar por lo que al devolver la posición del valor y en el array nunca será 0 que es la posición dada en el test.

5. Error: llama a findLast(arr, y), que al entrar al for comienza por la última posición del array y no llega a la primera sino que se queda en la segunda, luego comnpara en el if el valor del array con la 'y' (la primera se queda sin comparar) y devuelve la variable 'i' con la posición donde lo encuentra y compara con un 0.

LastZero:

1. 6. Falla al ejecutar LastZero puesto que a continuación llama a lastZero(arr) buscando el valor de 0 y devolviéndo la posición del último 0 encontrado. Compara esa posición con 2 buscando que el último 0 esté en la tercera posición del array:

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

Se debe que al ejecutar el siguiente código se empieza a comparar por el inicio por lo que te devolverá la primera posición donde encuentre un 0 por lo que si sólo hay un 0 será correcta la función pero en el momento que haya más no me devolverá el último valor, por ello puedo hacer que el for empiece por el final del array. Así siempre me volverá la última posición:

```
   public static int lastZero (int[] x)
{
   for (int i = x.length-1; i >= 0; i--)
   {
      if (x[i] == 0)
      {
         return i;
      }
   }
   return -1;
}
```

2. Con los test es imposible de evitar esa parte del código, al estar ejecutando directamente esa parte del código en el test. Pero sin los test tenemos la opción de que no llegue a llamarlo si lo ejecutamos sin argumentos.

3. Es posible que haya fallo y no error en el código, puesto que no hay un error en el código ya que itera bien el bucle sin saltarse ninguna posición, lo único que se queda con la primera posición y no la segunda.

4. Si tomamos como error que el for recorra de manera inversa y no obtenga el resultado final entonces si es posible que no haya failure si solo hay un 0 en la tercera posición del array, por ejemplo: {2,4,0}

5. Error: llama a lastZero(arr), que al entrar al for comienza por la primera posición del array y va comparando los valores del array con 0 cuando lo encuentra devuelve la posición, por lo que la posición la compara con la introducida en el test y a no ser que sólo haya uno en esa posición la comparación última será errónea.

CountPositive:

1. 6. Falla al ejecutar CountPositive puesto que a continuación llama a countPositive(arr) buscando los valores positivos del array y devolviéndo el número de valores positivos encontrado. Compara ese número con 2:

```
   @Test public void arrayContainsZeroes()
{
   int arr[] = {-4, 2, 0, 2};
   assertEquals("Array contains zeroes", 2,
               CountPositive.countPositive(arr));
}
```

Se debe que al ejecutar el siguiente código tiene en cuenta el numero 0 como positivo, por ello encuentra tres positivos en el array y no dos como espera. La función será válida siempre y cuando no hay un 0 entre los números que se encuentren en el array:

```
   public static int countPositive (int[] x)
{
   int count = 0;

   for (int i=0; i < x.length; i++)
   {
      if (x[i] > 0)
      {
         count++;
      }
   }
   return count;
}
```

2. Con los test es imposible de evitar esa parte del código, al estar ejecutando directamente esa parte del código en el test. Pero sin los test tenemos la opción de que no llegue a llamarlo si lo ejecutamos sin argumentos.

3. No es posible que haya fallo y no error en el código, puesto que en la variable está tomando como valor posible el 0: “x[i]>=0” hay un error y se ejecuta esa parte del código.

4. Cualquier array que no contenga un 0. {2,4,-2,-4} Esto va a hacer que aunque haya un fallo y un error de consistencia en la variable en la declaración comentada en el apartado anterior no provoque un failure.

5. Error: llama a countPositive(arr), que al entrar al for comienza por la primera posición del array y va comparando los valores del array con 0 cuando lo encuentra un número igual a 0 o superior aumenta el contador y luego compara el contador con el número de positivos que espera.

OddOrPos:

1. 6. Falla al ejecutar OddOrPos puesto que a continuación llama a oddOrPos(arr) buscando los valores positivos o impares del array y devolviéndo el número de valores encontrado. Compara ese número con 3:

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

Se debe que al ejecutar el siguiente código no tiene en cuenta los números negativos impares, por ello encuentra sólo dos con las características requeridas en el array y no tres como espera. La función será válida siempre y cuando no haya impares negativos entre los números que se encuentren en el array:

```
   public static int oddOrPos (int[] x)
{  // Effects:  if x is null throw NullPointerException
   // else return the number of elements in x that
   //      are either odd or positive (or both)
   int count = 0;

   for (int i = 0; i < x.length; i++)
   {
      if (Math.abs(x[i]%2) == 1 || x[i] > 0)
      {
         count++;
      }
   }
   return count;
}
```

2. Con los test es imposible de evitar esa parte del código, al estar ejecutando directamente esa parte del código en el test. Pero sin los test tenemos la opción de que no llegue a llamarlo si lo ejecutamos sin argumentos.

3. No es posible que haya fallo y no error en el código, puesto que en la declaración está tomando como valor posible el 0: “x[i]%2 ==1” hay un error al no tener en cuenta el valor absoluto puesto que los negativos darán ‘-1’ y no ‘1’ y se ejecuta esa parte del código.

4. Cualquier array que no contenga números impares negativos. {2,4,-2,-4} Esto va a hacer que aunque haya un fallo y un error de consistencia en la declaración comentada en el apartado anterior no provoque un failure.

5. Error: llama a oddOrPos(arr), que al entrar al for comienza por la primera posición del array y va comparando los valores del array con 0 y viendo su módulo cuando algunas de las comparaciones son verdaderas aumenta el contador y luego compara el contador con el número que espera.