Ángela Vargas Alba

Práctica: Introducción a pruebas de SW (testing)

FindLast.java

1.- La condición del bucle es errónea: for (int i=x.length-1; i>0; i--)

Si no se añade i>=0, nunca tendrá en cuenta el caso de x[0], que será la primera posición del array introducido, de manera que si el número buscado se encuentra en la posición 0, no lo encontrará y devolverá -1 en lugar de 0 que sería el valor correcto.

2.- La única opción de que no se ejecute el código con el fallo es introduciendo un array incorrecto en formato, o no introduciendo los argumentos correctos. De esta manera no se ejecutará la función *findLast* que es donde se encuentra el código que tiene el fallo.

3.- No es posible porque siempre se producirá un error en el estado en la última vuelta del bucle cuando no lea la primera posición del array. Por tanto, introduciendo cualquier array válido se ejecutará el código que tiene el fallo y en la última vuelta del bucle se generará un error en el estado porque no leerá la primera posición.

4.- Este caso de prueba se genera un error de estado cuando no lee la primera posición del array, pero no provocará una disfunción puesto que el número buscado no se encuentra en la primera posición del array.

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

5.- El primer estado erróneo será cuando i=0 y se encuentre en la línea 2. Entonces no entrará por el bucle y no estará teniendo en cuenta la primera posición del array. Para el caso de prueba anterior x[i]=2 y por tanto no se producirá una disfunción porque no cumpliría la condición de la línea 3.

```
1. public static int findLast (int[] x, int y) {
2.       for (int i=x.length-1; i > 0; i--)  {
3.                if (x[i] == y)  {
4.                    return i;
5.                }
6.       }
7.       return -1;
8. }
```

LastZero.java

1.- El programa fallará si existen múltiples ceros en el array de entrada. La función *lastZero* retorna la posición del primer cero encontrado, y no recorre todo el array previamente. Solución: dejar que la función lea todo el array previamente y retornar el valor cuando haya terminado.

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

2.- Cualquier array en el que no se incluya ningún cero. De esta manera no entrará nunca por la condición errónea y no ejecutará el código que tiene el fallo.

```
 @Test public void anyZeroes() {
     int arr[] = {2, 1, 3};
     assertEquals("Any zeroes: not should find zeroes", -1, LastZero.lastZero(arr));
   }
```

3.- Cualquier array en el que se incluya un único cero. En este caso sí que entrará por la condición que tiene el error y, por tanto, se ejecutará el fallo del código, pero no provocará un estado de error porque cuando encuentre el primer cero también será el último y será un estado correcto.

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

4.- No es posible porque para que se provoque un error en el estado es necesario que existan múltiples ceros en el array, si no, el estado será correcto porque siempre devolverá el último, y único, cero. Si introducimos múltiples ceros para generar error en el estado, se producirá inevitablemente una disfunción.

CountPositive.java

1.- El programa fallará si se introducen ceros en el array de entrada, ya que los contará como números positivos y retornará un valor erróneo. Solución: no tener en cuenta los ceros como elementos positivos, sustituimos x[i] >= 0.

```
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.- La única opción de que no se ejecute el código con el fallo es introduciendo un array incorrecto en formato, o no introduciendo el número de argumentos correcto. De esta manera no se ejecutará la función *countPositive* que es donde se encuentra el código que tiene el fallo.

3.- Introduciendo un array que sólo contenga números positivos. De esta manera se ejecutará el código que tiene el fallo, pero no supondrá un estado de error ya que no interpretará ningún cero como positivo, porque no hay.

```
@Test public void arrayNotContainsZeroes() {
    int arr[] = {-4, 2, -3, 2};
    assertEquals("Array not contains zeroes", 2, CountPositive.countPositive(arr));
  }
```

4.- No es posible, ya que para generar un error en el estado es necesario incluir en el array de entrada algún cero y que lo interprete como positivo, por lo que se estaría generando una disfunción.

OddOrder.java

1.- El fallo del programa se encuentra en que no tiene en cuenta los números que son negativos pero sí son impares. El fallo en el código se encuentra en la función *oddOrPos*:

if (x[i]%2 == 1 || x[i] > 0) → De esta manera se comprueba aquellos números cuyo múltiplo sea 1, es decir los impares positivos y los que simplemente son impares. Para tener en cuenta también los impares negativos:

```
if (x[i]%2 == 1 || x[i] > 0 || x[i]%2 == -1)
```

2.- De nuevo la única manera de que el fallo no se ejecutara sería no ejecutando esta función. Por tanto, sólo serían los casos de prueba en el que los argumentos fueran incorrectos.

3.- Sería posible que ejecutando el fallo del código no se generara un error en el estado si introdujéramos un array que contuviera únicamente valores positivos e impares y valores negativos, pero pares. De esta manera no se interpretaría mal un número impar negativo nunca.

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

4.- De nuevo tampoco es posible porque para generar un error de estado es necesario que en el array se encuentren números negativos e impares y, entonces, se generaría una disfunción en el programa.