Ejercicios Java Test SW

Alumna: Rocío Díaz Fernández

*NOTA: Todos los test están en el punto 6 de casa pregunta.

Programa 1: countPositive.java

1. El fallo sería que cuenta los 0's como números positivos y lo correcto sería que no.

```java
public static int countPositive (int[] x)
{
  int count = 0;
  for (int i=0; i < x.length; i++)
    {
      if (x[i] >= 0 && x[i]!= 0)
      {
        count++;
      }
    }
    return count;
}
```

2. El caso de que no se ejecute el fallo es que el array esté inicializado y que no tenga elementos. Por ejemplo:

```java
int arr[] = {};
```

```java
int arr[] = {2, 5,1,-6};
```

   En estos casos darán por su salida 0. Como no entra en el for el fallo no es ejecutado.

3. Con el siguiente ejemplo ejecutamos el fallo pero no produce ni un error de estado ni una disfunción:

```java
public static void main(String args[])
  {
      int arr[] = {2, 4, -9,5,-4};
      int y = countPositive.countPositive (arr);
      System.out.println(y);
  }
```

   Con lo que la salida es 3.

4. **No es posible.** Porque siempre vamos a obtener una disfunción cuando haya error. Con el siguiente ejemplo se ejecuta el fallo y obtenemos un error de estado y acabamos teniendo una disfunción:

```java
public static void main(String args[])
  {
      int arr[] = {0, 4, 0,5,-4};
      int y = countPositive.countPositive (arr);
      System.out.println(y);
  }
```

Con lo que obtenemos:

```
untPositive$ javac countPositive.java
untPositive$ java countPositive
```

Este resultado:

```
roci
4
```

Esto estaría mal. Debería sacar 2.

5. En el caso anterior, cuando entra al for a comprobar si los números son positivos o negativos, java va a decir siempre que 0 es positivo si no le decimos lo contrario.
   Cuando está en la posición 0 del array el número es el cero y lo toma como positivo por lo que entra en el if y suma a contador 1. En la segunda posición que es 4, lo hace correctamente, ejecuta el fallo pero no crea un error interno.

6. Capturas del los test creados:

```
@Test
public void testForNullArray()
{
  int arr[] = null;

   try {
      countPositive.countPositive (arr);
   } catch (NullPointerException e) {
      return;
   }
   fail ("NullPointerException expected");
}
```

```
@Test
public void testCorrectCount()
{
   int arr[] = {2, 5,1,-6};
   assertEquals("Count of positive", 3, countPositive.countPositive (arr));
}
```

```
@Test
public void testZeroPositive()
{
   int arr[] = {0, 4, 0,5,-4};
   assertEquals("Count of positive", 2, countPositive.countPositive (arr));
}
```

```
@Test
public void testCorrectCount2()
{
   int arr[] = {};
   assertEquals("Count of positive", 0, countPositive.countPositive (arr));
}
```

Programa 2: lastZero.java

1. El fallo es que encuentra la posición del primer cero que se encuentre el for empezando por la izquierda (empieza en la i = 0). En el caso de que haya dos 0's solo muestra la posición del primero y NO la del último.
   Una proposición de arreglo sería empezar a buscar en el array por la derecha.
   El código quedaría de la siguiente manera:

```java
/**
  * Find LAST index of zero
  *
  * @param x array to search
  * @return index of last 0 in x; -1 if absent
  * @throws NullPointerException if x is null
  */
public static int lastZero (int[] x)
{
  for (int i = x.length-1; i >= 0; i--)
  {
    if (x[i] == 0)
    {
      return i;
    }
  }
  return -1;
```

2. No es posible porque siempre que el for esté dando vueltas se va a estar ejecutando la sentencia en la que hemos detectado y arreglado el fallo.

3. Un caso en el que se ejecuta el fallo pero no produce un error de estado es cuando el array está vacío con lo que no se ejecuta la sentencia for.

4. Siempre que tengamos más de un 0 va a dar una disfunción, pero cuando solo hay uno tenemos un error de estado pero sin disfunción porque la variable del i empieza en 0 y no en x.length. Test en el apartado 6.
5. NC

6. Capturas de los test:

```java
@Test
public void testForNullArray()
{
  int arr[] = null;

   try {
      lastZero.lastZero (arr);
   } catch (NullPointerException e) {
      return;
   }
   fail ("NullPointerException expected");
}
```

```java
@Test
public void testCorrectSearch()
{
   int arr[] = {2, 5,0,-6};
   assertEquals("Count of positive", 2, lastZero.lastZero (arr));
}
```

```java
@Test
public void testNoNumbers()
{
   int arr[] = {};
   assertEquals("Count of positive", -1, lastZero.lastZero (arr));
}
```

```java
@Test
public void testNoZeros()
{
   int arr[] = {4,5,-4};
   assertEquals("Count of positive", -1, lastZero.lastZero (arr));
}
```

```java
@Test
public void test2Zeros()
{
   int arr[] = {4,0, 5,-4,0 };
   assertEquals("Count of positive", 4, lastZero.lastZero (arr));
}
```

*NOTA: Error al ir haciendo copia/pega de todos los test. NO he cambiado el mensaje de test.

Programa 3: findLast.java

1. El fallo es que el bucle for no comprueba la posición 0 del array. Con lo que si el **único** numero que concuerde con el index está en esa posición 0, no lo va a detectar.
   Podemos corregir el fallo de esta manera:

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

   Añadiendo un "=" al " >".

2. Es imposible, siempre se está produciendo el fallo.
3. En el caso de que se ejecute la línea pero de que no hay error de estado es cualquiera que no haga una disfunción, es decir, siempre que no esté el numero que se busca en la primera posición. (Test en el punto 6.)
4. En este caso si que podemos decir que en el caso de que haya un index que esté repetido en el array, en la posición 0 y en otra, habrá error interno por la ejecución del fallo pero no habrá disfunción porque habrá encontrado el último de la posición.
   Ejemplo:

```
public static void main(String args[])
  {
      int arr[] = {5,4,0,5,2,1};
      int z = 5
```

   Sacará por su salida la posición 3 y es CORRECTO.
5. Como he dicho antes, a causa de que el programa busca la última posición en la que está el index no vamos a detectar que ahí hay un fallo.
   El error viene causado porque cuando llega al final del for y comprueba que 0 no es mayor que 0, no entra al for, cuando si que debería.

6. Capturas de lo test hechos para este ejercicio:

```
@Test
public void testForNullArray()
{
  int arr[] = null;
  int z = 4;

   try {
      findLast.findLast (arr,z);
   } catch (NullPointerException e) {
      return;
   }
   fail ("NullPointerException expected");
}
```

```java
@Test
public void testCorrectSearch()
{
   int arr[] = {2, 5,0,-6};
   int z = 4;
   assertEquals("The good resul", -1, findLast.findLast (arr, z));
}
```

```java
@Test
public void testNoNumbers()
{
   int arr[] = {};
   int z = 4;
   assertEquals("No numbers", -1, findLast.findLast (arr,z));
}
```

```java
@Test
public void testNoIndexInArray()
{
    int z = 6;
    int arr[] = {4,5,-4};
    assertEquals("No found", -1, findLast.findLast (arr,z));
}
```

```java
@Test
public void testFirstElement()
{
    int z = 6;
    int arr[] = {6,5,-4};
    assertEquals("First element of array", 0, findLast.findLast (arr,z));
}
```

```java
@Test
public void testLastElement()
{
    int z = -4;
    int arr[] = {6,5,-4};
    assertEquals("Last element of array", 2, findLast.findLast (arr,z));
}
```

```java
@Test
public void test2Index()
{
    int z = 0;
    int arr[] = {4,0, 5,4,0 };
    assertEquals("More than one", 4, findLast.findLast (arr,z));
}
```

Programa 4: oddOrPos.java

1. El fallo de este programa es que no tiene en cuenta los números negativos impares. Al tener la expresión de “Impares o Positivos” tiene que tener en cuenta también los impares negativos. En la línea en la que se comprueba si el módulo del número es 1, debería comprobarse si es también -1.
   Imagen del código corregido:

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

2. Siempre va a ejecutar la línea en la que está el error mientras el for esté haciendo su función. MENOS cuando hay un array sin elementos que no pasa al if. (Test en el punto 6)

3. En el caso de que no haya error en el estado interno de las variables es cuando no hay ningún número negativo impar:

```
@Test
public void testNoNumOddNeg()
{
    int arr[] = {4,0, 5,-4,0 };
    assertEquals("Count of positive", 2, oddOrPos.oddOrPos (arr));
}
```

4. Imposible. Siempre que tengamos valores internos erróneos vamos a tener una anomalía en la salida.
5. NC
6. Capturas de los test:

```
@Test
public void testForNullArray()
{
  int arr[] = null;
   try {
      oddOrPos.oddOrPos (arr);
   } catch (NullPointerException e) {
      return;
   }
   fail ("NullPointerException expected");
}
```

```java
@Test
public void testCorrectSearch()
{
   int arr[] = {2, 5,0};
   assertEquals("Count of positive and odd", 2, oddOrPos.oddOrPos (arr));
}
```

```java
@Test
public void testNoNumbers()
{
   int arr[] = {};
   assertEquals("Count of positive", 0, oddOrPos.oddOrPos (arr));
}
```

```java
@Test
public void testOddNegatives()
{
    int arr[] = {-3, -1, -5,-7};
    assertEquals("Count of positive", 4, oddOrPos.oddOrPos (arr));
}
```

```java
@Test
public void testNoNumOddNeg()
{
    int arr[] = {4,0, 5,-4,0 };
    assertEquals("Count of positive", 2, oddOrPos.oddOrPos (arr));
}
```

*NOTA: Error al hacer el copia/pega de los test. NO he cambiado el comentario.