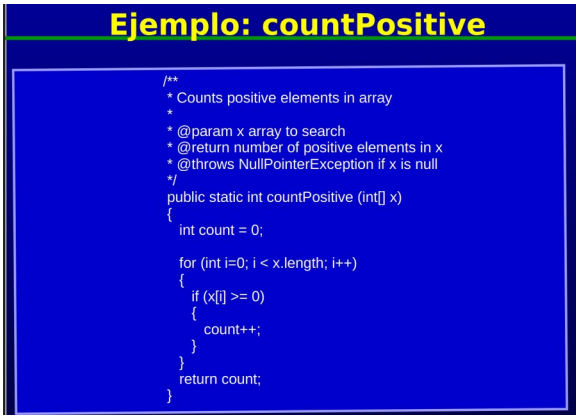
EJERCICIOS TEMA TEST SW:
ROCÍO NAVARRO VILLARINO

1. countPositive:

1. **Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.**
   No he encontrado fallo en este código.



2. lastZero:

1. **Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.**
   En este caso el fallo va a ser que va a devolver la posición del primer 0, no del último. Ya que cuando encuentra un 0 hace return de su posición, sin comprobar si es el último.
   Una posible solución es: poner que la i = x.length -1. De esta forma si estaría resuelto el problema. Devuelve la posición del primer 0 que encuentra por la cola de la lista.
   ```
   for (int i = x.length-1; i >= 0; i--)
   ```

2. **Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué. Para cada caso de prueba indica los datos usados en la prueba, el resultado esperado y el resultado obtenido.**
   La línea 12 (comienzo del for), siempre se ejecuta. No hay ejemplo que no atraviese el for, incluso cuando la lista está vacía.

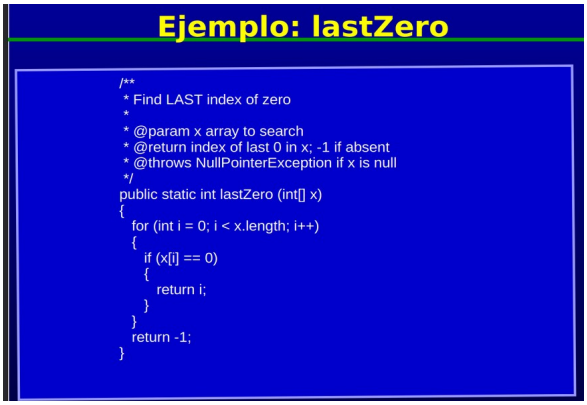
3. **Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.**
   No es posible que se ejecute el código de error y no se produzca un fallo en las variables/contador.

4. **Si es posible, proporciona un caso de prueba que provoque un error en el estado, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.**
   X = [1 2 0]. Resultado = 2. El resultado obtenido es el correcto. Las variables según el código están mal, porque el contador no debería comenzar en 0.

5. **Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalladamente todo el estado (todas las variables, incluyendo el contador de programa).**
   Al comienzo: i = 0 (x[0] = 1)
   i = 1 (x[1] = 2)
   i = 2 (x[2] = 0) → return i = 2.

6. **Ejecuta en programas Java el código corregido y pruébalo con los casos de prueba creados en apartados anteriores**

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



3. findLast:

1. **Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.**
   El fallo se encuentra en la línea del bucle for. No se llega a evaluar la última posición de la lista. Por ello, cuando en el test se da: x[0] = y, el resultado que devuelve no será nunca 0, porque no ha llegado.
2. **Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué. Para cada caso de prueba indica los datos usados en la prueba, el resultado esperado y el resultado obtenido.**
   La línea 13 (comienzo del for), siempre se ejecuta. No hay ejemplo que no atraviese el for, incluso cuando la lista está vacía, se ejecuta esta línea.

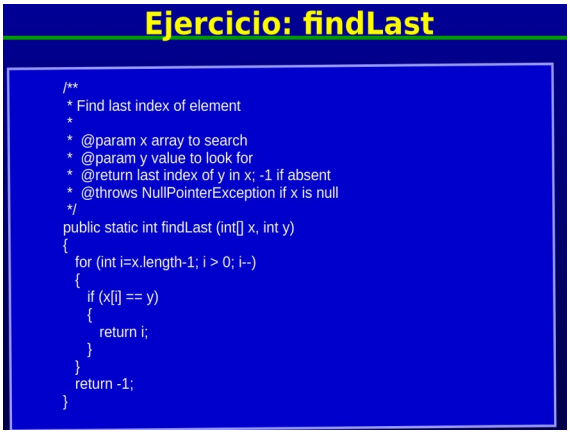
3. **Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.**
   No es posible dar un test de prueba que ejecute el fallo y no provoque error en el estado. Debido a que en este caso, la variable contador, no llega a ser 0.

4. **Si es posible, proporciona un caso de prueba que provoque un error en el estado, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.**
   Con el test list = {2,2,0}, y la letra y = 3. El resultado es -1.
   En este caso, el contador no llega nunca a la posición 0, por tanto, se produce error en el estado.
   El contador i comienza en 2, y avanza hasta i = 1. Y finaliza el bucle.
   X[2] !=3
   x[1] != 3
   Return = -1.

5. **Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalladamente todo el estado (todas las variables, incluyendo el contador de programa).**
   Al comienzo: i = 2 (x[2] = 2 != 3)
   i = 1 (x[1] = 2 =! 3)
   return i = -1.
   El primero de los casos erróneos es la primera vez que pasamos por el bucle, donde se dice que se seguirá ejecutando el bucle hasta que i > 0. Cuando debería repetirse hasta i >= 0.

6. **Ejecuta en programas Java el código corregido y pruébalo con los casos de prueba creados en apartados anteriores**

```
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;
	}
```



4. oddOrPos:

1. **Explica el fallo (qué está mal en el código). Describe de manera precisa el fallo y propón una modificación al código que lo arregle.**
   En este caso, el número 0 se considera par y no se le considera como positivo. Además cuando el número a evaluar es impar y negativo (como el -3), no se le considera impar.
   Por tanto, hay errores al evaluar el número 0 y al evaluar números impares negativos.
   Esto ocurre en la sentencia del if, dentro del bucle for.
   Una posibles solución será:
   Evaluar el número, según su valor absoluto e introducir el igual para denominar al 0 como número positivo.
   `if (Math.abs(x[i])%2 == 1 || x[i] >= 0)`

2. **Proporciona, si ello es posible, un caso de prueba que no ejecute el código que tiene el fallo. Si no es posible, explica por qué. Para cada caso de prueba indica los datos usados en la prueba, el resultado esperado y el resultado obtenido.**
   Como el fallo se da en el if dentro del bucle, puede haber algún caso en el que no entremos en el bucle, y por tanto, no ejecutemos la sentencia if.
   En este caso, lo más seguro es que al evaluar la lista en el bucle, se comprueba que es null, y salta excepción, sin ejecutar las sentencias dentro del bucle.
   X = []
   Resultado = Se eleva la excepción NullPointerException

3. **Si es posible, proporciona un caso de prueba que ejecuta el fallo que hay en el código, pero que no provoque un error en el estado. Si no se puede, explica por qué.**
   En este caso, sí podemos tener un test que ejecute el fallo y que no de error de estado, porque las variables (i, count) no se ven afectadas.
   Prueba:
   X = {1,-2,3}
   En el inicio:
   count = 0; i = 0
   x[0] = 1 → count = 1,
   count = 1; i = 1
   x[1] = -2 → count = 1,
   count = 1; i = 2
   x[2] = 3 → count = 2,
   return 2.

4. **Si es posible, proporciona un caso de prueba que provoque un error en el estado, pero que no acabe provocando una disfunción en el comportamiento del programa. No olvides que el contador de programa forma parte, junto a las variables, del estado del programa. Si no es posible, explica por qué.**
   En este caso, no podemos proporcionar un caso que de error de estado, pero que no provoque disfunciones. Esto se debe a que la variables i, no podemos modificarla con las entradas, y si la variable count da error (error de estado), entonces se va a dar disfunción, porque es el valor que se devuelve.

5. **Para el caso de prueba del anterior apartado, describe el primero de los estados erróneos. Describe detalladamente todo el estado (todas las variables, incluyendo el contador de programa).**

6. **Ejecuta en programas Java el código corregido y pruébalo con los casos de prueba creados en apartados anteriores**

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