

**PRÁCTICA CAJA NEGRA**

IWM41

**─**

Grupo 8:

Omar Piñeiro Parada

Violeta Macías de Miguel

Javier Gómez de Lucio

Javier Herrero Sánchez

Alejandro Del río Álvarez

# Índice

[**Índice**](#_nhq21d5sol1k) **2**

[**Test addAtPos(T element, int p)**](#_k5d1mr6o07pl) **2**

[**Test addFirst(T element)**](#_13y0we36qid) **5**

[**Test addNTimes (T element, int n)**](#_1zi00mhd9ra2) **6**

[**Test addLast (T element)**](#_nnig5ixsuivt) **7**

[**Test removeLastTest()**](#_w1eompulgdlx) **8**

[**Test removeLastTest(T elem)**](#_l11mwi2on3nz) **9**

[**Test size()**](#_cvzeraekuits) **10**

[**Test toString()**](#_48t19w157etp) **10**

[**Test reverse()**](#_o4itd0blo4dj) **11**

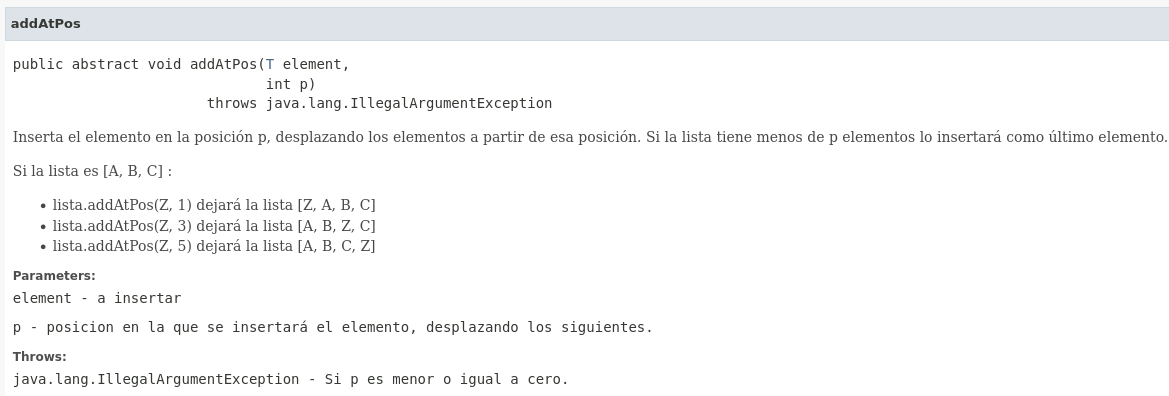
[**TestGetAtPos**](#_dzc46knkcys0) **12**

[**TestIndexOf**](#_6cld7wa6mo67) **13**

[**TestIsSublist**](#_wfmkupa051c4) **14**

# 

# Test addAtPos(T element, int p)



Descripción funcionalidad método addAtPos.

Para los tests se han identificado las siguientes clases de prueba:

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | A <= element <= Z | element < A  element > Z |
| p | p >= 1 && p <= tamaño  p > tamaño | p <= 0 |

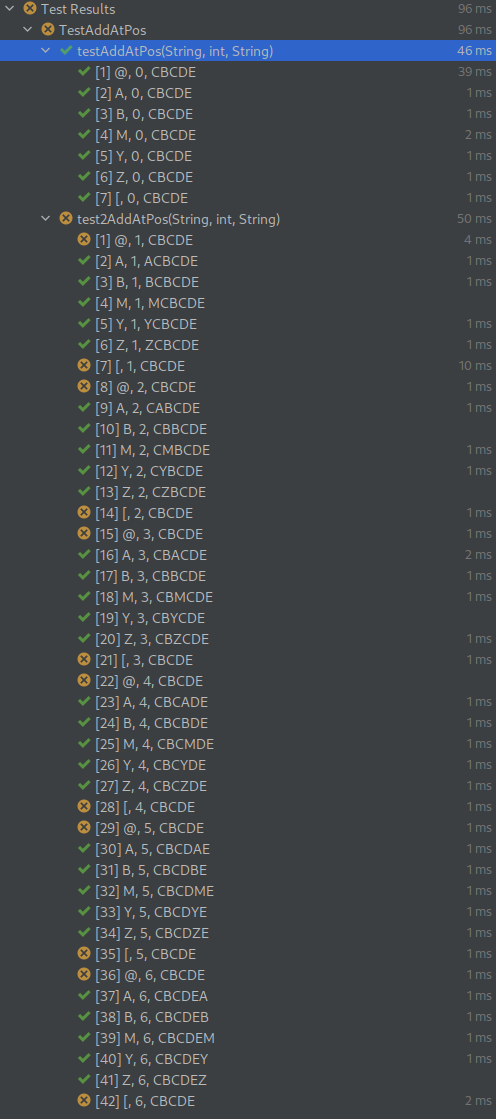
element {“@”, “A”, “B”, “M”, “Y”, “Z”, “[“}

p {0, 1, 2, tamaño / 2, tamaño - 1, tamaño, tamaño + 1}

Debido a la gran cantidad de casos de prueba (7²), se ha determinado que es necesario el empleo de CSV. Para ello, se ha usado el siguiente patrón para construirlo:

element, p, resultado

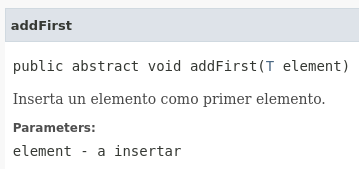
Ejemplo: "@, 1, CBCDE",



Como se puede observar, los tests que fallan son los que usan elementos que no son letras. Por ejemplo, en el resultado del test (@, 1, CBCDE) se comprueba que también se aceptan símbolos. Esto incumple la funcionalidad expuesta en el enunciado de solo aceptar letras mayúsculas.



# Test addFirst(T element)



Descripción funcionalidad método addFirst.

Para los tests se han identificado las siguientes clases de prueba:

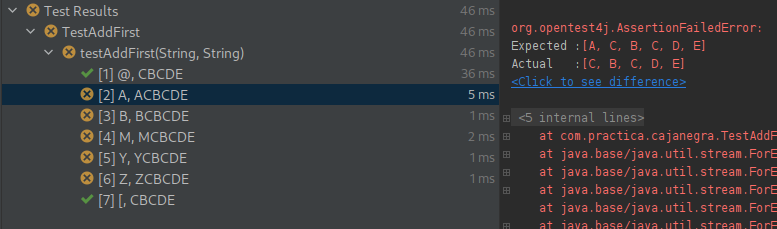
| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | A <= element <= Z | element < A  element > Z |

element {“@”, “A”, “B”, “M”, “Y”, “Z”, “[“}

Se ha usado un CSV para agrupar todos los casos de prueba (7). La estructura del mismo es:

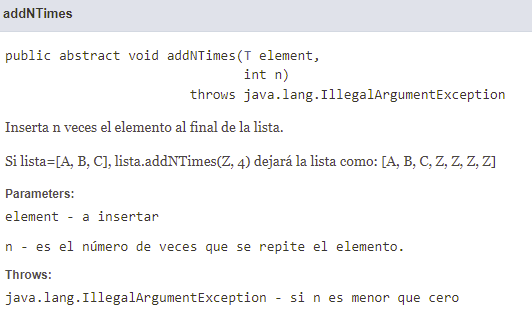
element, resultado

Ejemplo: “@, CBCDE”,



Como se ve en los tests, fallan los que añaden una letra mayúscula debido a que el método no cumple su función de añadir una letra al principio. Es por esto que se ejecutan satisfactoriamente los tests con símbolos ya que no los tiene que añadir.

# Test addNTimes (T element, int n)



Para los test, se han identificado los siguientes casos de prueba:

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | A <= element <= Z | element < A  element > Z |
| n | 0 <= n | n < 0 |

Por lo tanto, se han implementado los siguientes tests:

**1**: añadir un elemento válido un número de veces mayor que 0. Debería añadirlos correctamente.

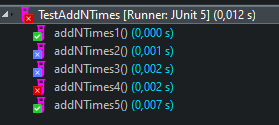
**2**: añadir un símbolo “@” un número de veces mayor que 0. Debería no añadir nada.

**3**: añadir un símbolo “ [ ” un número de veces mayor que 0. Debería no añadir nada.

**4**: añadir un elemento válido un número de veces igual a 0. Debería no añadir nada.

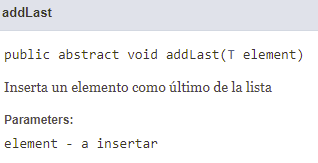
**5**: añadir un elemento válido un número de veces menor que 0. Debería lanzar una Excepción.

Resultados**:**



Observamos que las pruebas número 2 y 3 fallan, ya que el método añade a la lista los símbolos, cuando no debería ser posible según el enunciado. La prueba número 4 también falla porque con n=0, el método lanza una excepción como si fuera un número no válido, cuando en la descripción del método se entiende que lo es. Las pruebas 1 y 5 funcionan correctamente.

# Test addLast (T element)



Para los test, se han identificado los siguientes casos de prueba:

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | A <= element <= Z | element < A  element > Z |

Por lo tanto, se han implementado los siguientes tests:

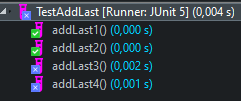
**1**: añadir un elemento válido. Debería añadir el elemento correctamente

**2**: añadir un elemento válido y repetido. Debería añadir el elemento correctamente.

**3**: añadir un símbolo “@” no válido. No debería añadir nada.

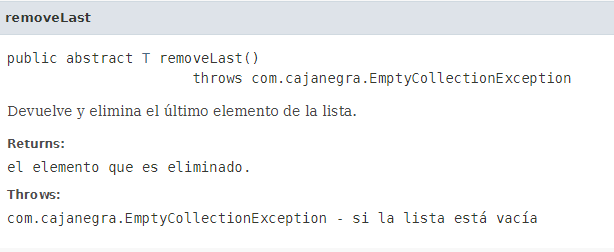
**4**: añadir un símbolo “ [ ” no válido. No debería añadir nada.

Resultados:



Observamos que las pruebas 1 y 2 funcionan de manera adecuada. Sin embargo, las pruebas 3 y 4 no producen el resultado esperado, ya que el método consigue añadir los símbolos sin problema, cosa que no debería hacer.

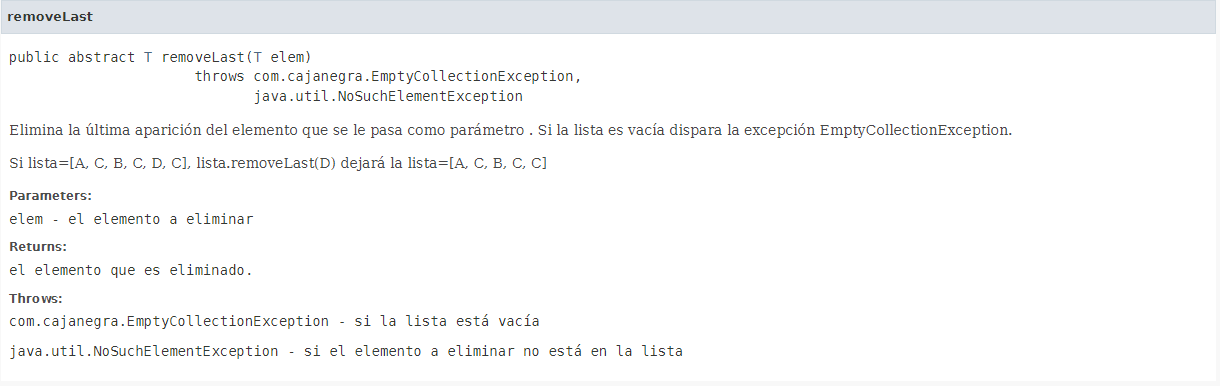
# Test removeLastTest()



Para los test se han identificado los siguientes casos de prueba

* lista vacía
* lista de un elemento
* lista de 2 o más elementos

# Test removeLastTest(T elem)



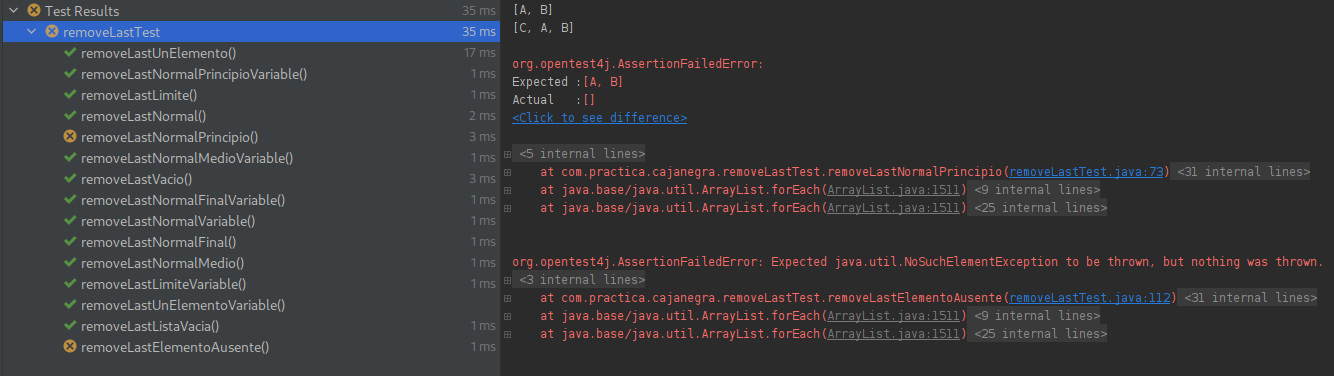
Para los test se han identificado los siguientes casos de prueba

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | A <= element <= Z para todo element que esté presente en la lista | element < A  element > Z y los elementos que no estén en la lista |

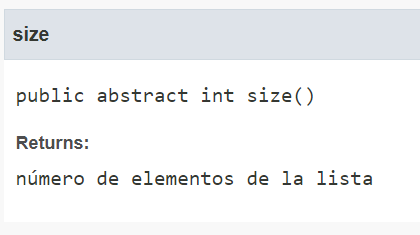
element {“A”,”B”,”C”,”D”}

* lista vacía
* lista de 1 elemento
* lista de 2 o más elementos

Como se puede ver fallan dos test, esto es debido a que si se pretende eliminar el primer elemento de una lista de por lo menos dos elementos se eliminará en su lugar la lista entera, el otro test que falla es el que prueba si se lanza la excepción



# Test size()



Descripción funcionalidad método size().

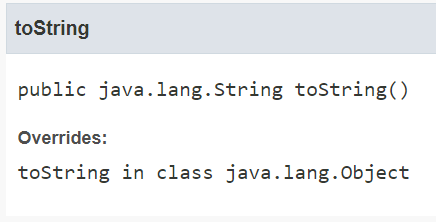
Para los test se han identificado los siguientes casos de prueba

* lista vacía (“”)
* lista de un elemento("A")
* lista de 2 o más elementos("A", "B", "C")



Los test se pasan satisfactoriamente.

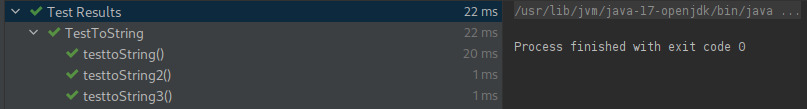
# Test toString()



Descripción funcionalidad método toString().

Para los test se han identificado los siguientes casos de prueba

* lista vacía (“”)
* lista de un elemento("A")
* lista de 2 o más elementos("A", "B", "C")



Los test se pasan satisfactoriamente.

# Test reverse()

Descripción funcionalidad método reverse().

Para los test se han identificado los siguientes casos de prueba

* lista vacía (“”)
* lista de un elemento("A")
* lista de 2 o más elementos("A", "B", "C")



Los test se pasan satisfactoriamente.

## 

# TestGetAtPos()

Esta prueba devuelve el elemento dada una posición pasada por parámetro, si el número no es uno de los índices de la lista, entonces lanza una excepción *IllegalArgumentException.* Como se puede ver en las diferentes capturas, todos los test realizados han sido satisfactorios.

Los casos de prueba identificados para este método son:

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | element > 0  element <= Tamaño de la lista | element <= 0  element > Tamaño de la lista |







# TestIndexOf()

Este método debería devolver el índice de la posición de la lista en la que se encuentra un elemento pasado por parámetro.

En este caso hemos realizado las pruebas con el primer elemento, el último elemento y de un elemento entre medias. Como se puede observar, en la captura, estas pruebas, se han superado satisfactoriamente. También hemos incluido la comprobación de la excepción *NoSuchElementException.* Esta excepción es ejecutada cuando el elemento que se ha pasado por parámetro no pertenece al conjunto de elementos incluidos en la lista. Como se observa en la captura también lo ha superado satisfactoriamente.

Los casos de prueba identificados para este método son:

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | element = Algún elemento de la lista  A <= element <= Z | element != Algún elemento de la lista |



# TestIsSublist()

Este método devuelve un entero, indicando el índice de la posición del inicio de la lista en el caso de que sea una sublista, en caso contrario, devuelve -1.

Como se puede ver en las capturas las pruebas realizadas son una lista de 2 elementos que es una sublista de otra lista de 3 elementos. La segunda prueba es una lista de 2 elementos que no es una sublista de la lista de 1 elemento, luego el valor esperado de esta prueba sería -1. La tercera prueba se trata de dos listas iguales de 1 elemento. La cuarta prueba se basa en comprobar si una lista vacía contiene alguna sublista. Y la quinta prueba es una una sublista de una lista, que empieza en la posición número 2.

Los casos de prueba identificados en este caso serían los siguientes:

| Entrada | Clases válidas | Clases inválidas |
| --- | --- | --- |
| element | element = Una sublista contenida en la lista | element != Una sublista contenida en la lista |

Como se puede observar en las capturas, todos los test han sido completados satisfactoriamente.

