INFORME PRÁCTICA 4B: PRUEBAS DE SOFTWARE

1. Introducción

En este documento se pretende detallar los resultados del proceso de verificación y validación de pruebas de la clase dada ListaOrdenada, comprobar el funcionamiento correcto de los diversos métodos de la clase dada y que así se cumplan con los requisitos funcionales y comportamiento correcto en diferentes casos de uso.

- Fases
  - Definición de los casos de prueba: se vieron los valores límites de las distintas clases y se han determinado valores con los que han diseñado las pruebas de caja negra y caja blanca.
  - Creación de las pruebas: confirmando con la estructura del código si estas son posibles para los casos creados.
  - Comprobación de las pruebas: Revisión de los valores esperados y corrección de errores con respecto al desarrollo anterior.
- Técnicas de pruebas aplicadas
  - Pruebas de caja negra: revisión del funcionamiento general de los métodos.
  - Pruebas de caja blanca: comprobar funcionamiento interno según el código de los métodos.
- Herramientas utilizadas
  - Junit, Maven y Git (GitHub).
- Criterios de cobertura elegidos
  - Cobertura de sentencias: verificar que todas las lineas de código sean corridas por lo menos una vez.
  - Cobertura de decisión: comprobar que cada estructura de control ha sido evaluada en sus posibles casos.
  - Cobertura de bucles: evalúa la ejecución de los bucles de código, comprobando su funcionamiento para distintos casos.
- Herramientas utilizadas para cada tipo de prueba
  - Caja negra: Junit, Maven y Excel
  - Caja blanca: Junit, Maven y Debugger de VS code.

2. Proceso de pruebas de caja negra de la clase ListaOrdenada

Métodos	Parametros	Clases Validas	Clases no Validas
get	index	Índice dentro del rango de la lista	< 0 Índice fuera del rango de la lista
	lista	Con elementos	Vacía
add	elemento	!= null	null
	lista	Vacía Con elementos	
remove	index	Índice dentro del rango de la lista	< 0

			Índice fuera del rango de la lista
	lista	Con elementos	Vacía
size	lista	Vacía Con elementos	
clear	lista	Vacía Con elementos	

Valores:

- **Índice dentro del rango de la lista**: 0
- **Con elementos**: [0, 1, 2, 3, 4]
- **!= null**: 4
- **Vacía**: []
- **< 0**: -1
- **Índice fuera del rango de la lista**: 5
- **null**: null

Métodos	Casos de prueba validos	Casos de prueba no validos
get	(0, [0,1,2,3,4]): 0 ([0,1,2,3,4])	(0, []): IndexOutOfBoundsException (-2, [0,1,2,3,4]): IndexOutOfBoundsException (5, [0,1,2,3,4]): IndexOutOfBoundsException
add	(1, []): ([1]) (2, [1]): ([1,2])	(null, [1]): NullPointerException
remove	(0, [0,1,2]): ([1,2])	(0, []): IndexOutOfBoundsException (-2, [0,1,2]): IndexOutOfBoundsException (5, [0,1,2]): IndexOutOfBoundsException
size	([0,1,2,3,4]): 5 ([]): 0	
clear	([0,1,2,3,4]): ([]) ([]): ([])	

3. Proceso de pruebas de caja blanca de la clase ListaOrdenada

Se han subrayado los casos de prueba añadidos respecto al caso anterior.

Cobertura de condición/decisión:

Métodos	Clase de equivalencia	Casos de prueba	Valor esperado
get	Lista vacía	0, []	IndexOutOfBoundsException
	Índice negativo	-2, [0,1,2,3,4]	IndexOutOfBoundsException
	Índice fuera de rango	5, [0,1,2,3,4]	IndexOutOfBoundsException
	Índice dentro de rango	0, [0,1,2,3,4] 4, [0,1,2,3,4]	0, [0,1,2,3,4] 4, [0,1,2,3,4]
remove	Lista vacía	0, []	IndexOutOfBoundsException
	Índice negativo	-2, [0,1,2,3,4]	IndexOutOfBoundsException
	Índice fuera de rango	5, [0,1,2,3,4]	IndexOutOfBoundsException
	Índice dentro de rango	0, [0,1,2,3,4] 3, [1,2,3,4]	0, [1,2,3,4] 3, [1,2,3]

Cobertura de condición:

Métodos	Clase de equivalencia	Casos de prueba	Valor esperado
add	Lista vacía	1, []	[1]
	Lista con elementos	2, [1]	[1,2]
	Elemento null	null, [1]	NullPointerException

Cobertura de bucle:

Métodos	Clase de equivalencia	Casos de prueba	Valor esperado
size	Lista vacía (tamaño = 0)	[]	0, []
	Lista con elementos (tamaño > 0)	[0,1,2,3,4]	5, [0,1,2,3,4]
clear	Lista vacía (tamaño = 0)	[]	[]
	Lista con elemento (tamaño = 1)	[0]	[]
	Lista con elementos (tamaño > 1)	[0,1,2,3,4]	[]

Cobertura alcanzada:

- Cobertura de condición/decisión: 100% con todas las condiciones de control y expresiones booleanas son evaluadas en sus distintos casos verdaderos y falsos.
- Cobertura de bucles: 100%, los bucles se han ejecutado con valores límites y múltiples casos.
- Cobertura de sentencias: 100% al correr todas las lineas entre las pruebas creadas.

Haciendo uso de los distintos casos de prueba, encontramos y corregimos los siguientes errores:

1. El método add() añadía elementos de forma que la lista estuviera ordenada de mayor a menor, en vez de menor a mayor. Fuimos notificado de este error gracias a que la clase de equivalencia “Lista con elementos” del mismo método no retornaba el valor esperado. Ubicamos la causa en un uso erróneo del método compareTo() en el método add(), y lo corregimos.
2. El método clear() nunca borraba el primer elemento, sin importar el número de elementos que tuviese la lista. Nos dimos cuenta de este error debido a que las clases de equivalencia “Lista con elemento (tamaño = 1)” y “Lista con elementos (tamaño > 1)” no retornaban el valor esperado. Encontramos que la razón era que el bucle de borrado en clear() no empezaba en la primera posición de la lista, sino en la segunda (i = 1), por lo que lo arreglamos para que lo hiciera.
3. Aún arreglado el anterior error, el método clear() seguía sin borrar todos los elementos, borrando solo la mitad. Nos avisó de esto la clase de equivalencia “Lista con elementos (tamaño > 1)” al no retornar el valor esperado. Averiguamos porque revisando el código y dándonos cuenta de que el bucle for estaba mal construido para el borrado de lista, ya que todas las posiciones de los elementos se mueven a la

izquierda tras que se borre un elemento, y el bucle avanza a la derecha por cada iteración. Lo solucionamos haciendo que el bucle for comience desde la última posición e iteré hacia la primera.

Inicialmente, el método clear solo tenía su primer y último caso de prueba (“Lista vacía (tamaño = 0)” y “Lista con elementos (tamaño > 0)”), pero al tratar de encontrar la solución al segundo error que arreglamos, decidimos añadir “Lista con elemento (tamaño = 1)” con el fin de alcanzar la cobertura deseada.