Alddrin Damian Venegas Cordero
Estructura de Datos
Facultad de Ciencias Técnicas
Ingeniería en Tecnologías de la Información

Parte 1: Identificación de Casos Borde

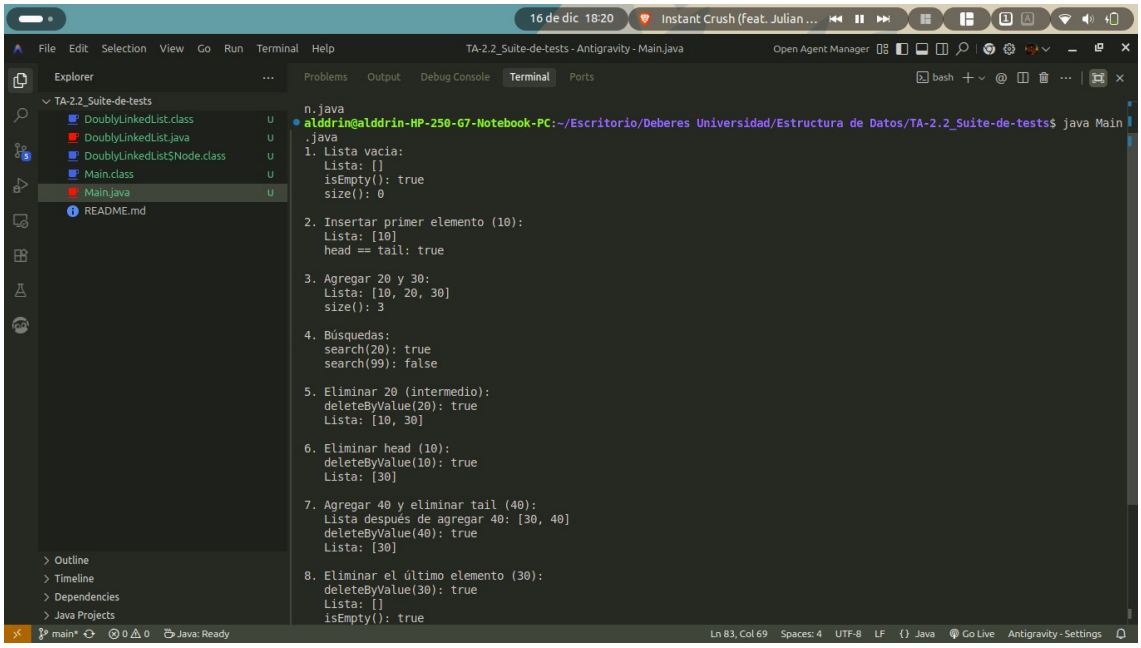
#	Caso Borde	Justificación de Criticidad
1	Lista vacía (head == null, tail == null)	Muchas operaciones ya sea eliminar, buscar, invertir, deben manejar null sin lanzar excepciones. Errores aquí rompen la robustez básica.
2	Lista con un solo elemento (head == tail, prev/next == null)	Operaciones como eliminar o invertir deben actualizar correctamente head/tail a null, y mantener punteros consistentes.
3	Eliminar head (primer elemento)	Debe actualizar head al siguiente nodo y establecer su prev = null. Crítico para mantener integridad bidireccional.
4	Eliminar tail (último elemento)	Debe actualizar tail al anterior nodo y establecer su next = null. Evita punteros inválidos al final.
5	Eliminar único elemento (lista pasa de 1 a 0 elementos)	Combinación de eliminar head/tail: debe dejar head = tail = null. Caso común de error en implementaciones.
6	Insertar en lista vacía	Debe crear el primer nodo y establecer head = tail correctamente, con prev/next = null.
7	Buscar elemento inexistente	Debe retornar null o false sin errores, incluso en lista vacía o con múltiples elementos.
8	Invertir lista vacía o con 1 elemento	Lista vacía debe permanecer vacía con 1 elemento debe quedar igual. Verifica que no se rompan punteros en casos triviales.

Parte 2: Diseño de Casos de Prueba

ID	Precondición	Acción	Resultado Esperado	Postcondición o verificación de punteros
TC-DLL-001	Lista vacía	addLast(10)	Lista: [10], retorna true	head == tail, head.prev == null, tail.next == null
TC-DLL-002	Lista vacía	deleteByValue(10)	Retorna false, lista permanece vacía	head == null, tail == null
TC-DLL-003	Lista vacía	search(5)	Retorna false	head == null, tail == null
TC-DLL-004	Lista vacía	reverse()	Lista permanece vacía	head == null, tail == null

ID	Precondición	Acción	Resultado Esperado	Postcondición o verificación de punteros
TC-DLL-005	Lista con un elemento: [10]	deleteByValue(10)	Retorna true, lista vacía	head == null, tail == null
TC-DLL-006	Lista con un elemento: [10]	reverse()	Lista: [10]	head == tail, prev/next == null
TC-DLL-007	Lista: [10, 20, 30]	deleteByValue(10) // eliminar head	Retorna true, lista: [20, 30]	head.value=20, head.prev==null, tail.next==null
TC-DLL-008	Lista: [10, 20, 30]	deleteByValue(30) // eliminar tail	Retorna true, lista: [10, 20]	tail.value=20, tail.next==null, head.prev==null
TC-DLL-009	Lista: [10, 20, 30]	deleteByValue(20) // eliminar medio	Retorna true, lista: [10, 30]	10.next=30, 30.prev=10, punteros consistentes
TC-DLL-010	Lista: [10, 20, 30]	search(99)	Retorna false	Lista sin cambios
TC-DLL-011	Lista: [10, 20, 30]	reverse()	Lista: [30, 20, 10]	head=30 (prev=null), tail=10 (next=null), punteros invertidos correctamente
TC-DLL-012	Lista con un elemento: [5]	addLast(15)	Lista: [5, 15]	head=5 (prev=null), tail=15 (next=null), 5.next=15, 15.prev=5

Parte 4: Reporte de Cobertura



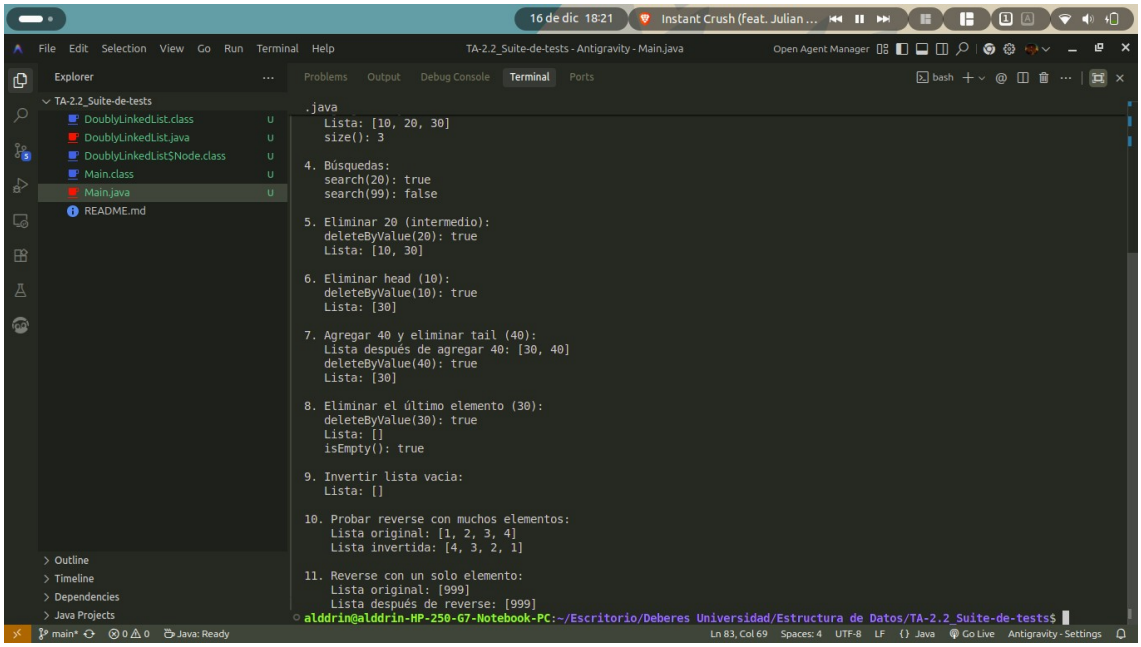


Tabla de ejecución de pruebas:

Métrica	Cantidad
Total pruebas	11
Pruebas pasadas	11
Pruebas fallidas	0

Todas las pruebas definidas en DoublyLinkedListTest.java se ejecutaron correctamente utilizando la implementación de DoublyLinkedList.java. No se produjeron excepciones inesperadas como NullPointerException en ningún caso borde.

¿Qué operaciones están cubiertas?

Operación / Escenario	Cubierto	Correspondencia en Main
Lista vacía (estado inicial, isEmpty, size)	Sí	Caso 1 y 9
Inserción en lista vacía (`addLast`)	Sí	Caso 2
Inserción en lista no vacía	Sí	Caso 3 y 7
Búsqueda de elemento existente	Sí	Caso 4
Búsqueda de elemento inexistente	Sí	Caso 4
Eliminación de nodo intermedio	Sí	Caso 5
Eliminación del head	Sí	Caso 6
Eliminación del tail	Sí	Caso 7
Eliminación del único elemento	Sí	Caso 8
Inversión de lista vacía	Sí	Caso 9

Operación / Escenario	Cubierto	Correspondencia en Main
Inversión de lista con múltiples elementos	Sí	Caso 10
Inversión de lista con un solo elemento	Sí	Caso 11
Verificación implícita de punteros y consistencia	Sí	En todos los casos

Identificar Gaps: ¿Qué casos faltan por probar?

Eliminación en lista vacía - No se intenta deleteByValue() sobre lista vacía, eso debería retornar false sin error.

Búsqueda en lista vacía - No se ejecuta search() cuando la lista está vacía.

Inserción múltiple y reverse en lista con 2 elementos - Solo se prueba con 1 y 4 elementos aunque el de 4 cubre el caso general.

Valores duplicados - No se prueba comportamiento con elementos repetidos como [10, 20, 10].

Operaciones por índice - No existen ni se prueban métodos como addFirst, get(index), removeAt(index).

Pruebas de estrés - No se insertan cientos o miles de elementos.

Propuestas de Mejora para Aumentar Cobertura

Agregar al Main.java dos casos más:

- Intentar deleteByValue(99) en lista vacía.
- Intentar search(5) en lista vacía, esto elevaría la demostración a 13 casos y cubriría los gaps menores.
- Agregar método addFirst a la lista y probarlo en Main.java o en tests.