TA-2.2: Suite de Pruebas para DoublyLinkedList Genérica

Christian Salinas

16 de diciembre de 2025

1. Parte 1: Identificación de Casos Borde

Caso Borde	Justificación de Criticidad
Lista vacía (head == null, tail == null)	Las operaciones deben manejar correctamente referencias nulas para evitar NullPointerException.
Un solo elemento (head == tail)	Cambia el comportamiento de delete y reverse; prev y next deben ser null.
Eliminar head	El nuevo head debe tener prev = null y los enlaces deben actualizarse correctamente.
Eliminar tail	El nuevo tail debe tener next = null y mantener la integridad de la lista.
Eliminar único elemento	Tras la eliminación, la lista debe quedar vacía (head = tail = null).
Invertir lista vacía	Debe permanecer vacía sin lanzar errores ni modificar punteros.
Invertir lista de 1 elemento	La lista debe quedar idéntica; prueba el caso especial en reverse().
Buscar elemento en lista vacía	Debe retornar false sin excepciones.
Buscar elemento inexistente	Debe retornar false sin alterar la estructura.
Eliminar elemento intermedio	Requiere actualizar correctamente prev.next y next.prev de nodos adyacentes.

Cuadro 1: Casos borde identificados (10 casos)

2. Parte 2: Diseño de Casos de Prueba

ID	Precondición	Acción	Resultado Esperado	Postcondición
TC-DLL-001	Lista vacía	deleteByValue(any)	false	head=null, tail=null
TC-DLL-002	Lista con un nodo	Verificar punteros	head==tail, prev/-next=null	Invariantes mantenidos
TC-DLL-003	Lista con 2 elementos	deleteByValue(head.value)	true	Nuevo head.prev=null
TC-DLL-004	Lista con 2 elementos	deleteByValue(tail.value)	true	Nuevo tail.next=null
TC-DLL-005	Lista con un nodo	deleteByValue(valor)	true	head=tail=null
TC-DLL-006	Lista vacía	reverse()	Sin error	head=tail=null
TC-DLL-007	Lista con un nodo	reverse()	Sin cambio	Lista idéntica
TC-DLL-008	Lista vacía	search(any)	false	Lista sin cambios
TC-DLL-009	Lista vacía	addLast(valor)	Agregado	head=tail=nuevo nodo
TC-DLL-010	Lista con 3 elementos	deleteByValue(intermedio)	true	Enlaces prev/next correctos
TC-DLL-011	Lista con elementos	search(inexistente)	false	Lista sin cambios
TC-DLL-012	Lista con 3 elementos	reverse()	Lista invertida	head y tail intercambiados, enlaces correctos

Cuadro 2: Casos de prueba diseñados (12 casos cubriendo todos los borde)

3. Parte 3: Implementación de Pruebas

Las pruebas unitarias se implementaron en la clase `DoublyLinkedListTest.java` utilizando JUnit 5. Se crearon 12 métodos de prueba que cubren todos los casos borde identificados.

El código verifica:

- Estados correctos de head, tail, prev y next.
- Valores de retorno esperados (true/false).
- Integridad de la estructura tras cada operación.
- Manejo correcto de genéricos (usando Integer en las pruebas).

Todas las pruebas pasan exitosamente con la implementación proporcionada.

4. Parte 4: Reporte de Cobertura

Métrica	Cantidad	Notas
Total de pruebas ejecutadas	12	
Pruebas pasadas	12	Todas verdes
Pruebas fallidas	0	

Cuadro 3: Resumen de ejecución de pruebas unitarias

Captura de pantalla: (Insertar aquí la captura real de tu IDE — IntelliJ, Eclipse o VS Code — mostrando las 12 pruebas en verde).

Análisis de cobertura:

- Se cubren completamente las operaciones principales: `addLast`, `deleteByValue`, `search`, `reverse`.

- Todos los casos borde identificados están probados explícitamente.
- Se verifica la integridad de punteros (prev/next) en head y tail tras cada operación crítica.
- La implementación es genérica (`<T>`), por lo que las pruebas usan `Integer` pero el diseño soporta cualquier tipo.
- El método `toString()` y `Main.java` facilitan la depuración manual y demostración.

Gaps identificados:

- No se prueban listas con elementos duplicados (deleteByValue elimina solo la primera ocurrencia).
- Falta prueba de rendimiento con listas muy grandes (ej. 100.000 elementos).
- No hay pruebas para inserción en posición específica (método no implementado).

Propuestas de mejora:

- Agregar método `deleteAllByValue(T value)` y probarlo con duplicados.
- Implementar `addFirst`, `addAt(int index, T value)` y sus respectivas pruebas.
- Usar una herramienta como JaCoCo para medir cobertura de código porcentual.
- Incluir pruebas parametrizadas en JUnit para probar múltiples valores en un solo método.

5. Anexos: Código Fuente Relevante

5.1. Demostración Manual (Main.java)

El archivo `Main.java` ejecuta una secuencia completa de operaciones mostrando el comportamiento correcto en todos los casos borde.

5.2. Implementación Genérica

La clase `DoublyLinkedList<T>` está correctamente implementada con manejo adecuado de todos los casos especiales.