Kevin Michael Girón Chuquirima
Estructura de datos
Facultad de Ciencias Técnicas
Ingeniería en Tecnologías de la Información

Tema: Suite de tests (casos borde)
Fecha: 16/12/2025
Descripción: Esta actividad de trabajo autónomo te permitirá diseñar y ejecutar una suite completa de pruebas para validar el comportamiento de una lista doblemente enlazada. Te enfocarás en casos borde (edge cases), pruebas de caja negra, y cobertura de escenarios críticos que garanticen la robustez de la implementación.

Contexto:
Las pruebas de casos borde son fundamentales para detectar errores que solo aparecen en situaciones límite (lista vacía, un solo elemento, eliminar head/tail). Una suite de pruebas bien diseñada valida que la implementación de DoublyLinkedList maneje correctamente todos los escenarios posibles, no solo los casos comunes.

Parte 1: Identificación de Casos Borde:

Tabla con 8 casos borde identificados y justificación de criticidad.

	Caso Borde	Descripción	Justificación de criticidad
1	Lista vacía	La lista no contiene nodos (head == null, tail == null)	Es el estado inicial. Muchas implementaciones fallan aquí con NullPointerException.
2	Lista con un solo elemento	head == tail, prev == null, next == null	Valida consistencia mínima de punteros. Error común al eliminar o invertir.
3	Insertar en lista vacía	Agregar el primer nodo	Define correctamente head y tail. Si falla, todo lo demás falla.
4	Eliminar head	Eliminar el primer nodo de una lista con ≥2 nodos	Debe actualizar head y limpiar prev. Error típico: head.prev no nulo.
5	Eliminar tail	Eliminar el último nodo de una lista con ≥2 nodos	Debe actualizar tail y limpiar next. Error común: tail.next no nulo.
6	Eliminar elemento intermedio	Eliminar un nodo que no es head ni tail	Valida la reconexión correcta de prev y next.
7	Buscar elemento inexistente	Buscar un valor que no está en la lista	Debe manejarse sin errores y retornar null o -1.
8	Invertir lista vacía o de 1 elemento	Ejecutar reverse() sobre casos límite	No debe modificar el estado ni causar errores.

Kevin Michael Girón Chuquirima
Estructura de datos
Facultad de Ciencias Técnicas
Ingeniería en Tecnologías de la Información

Parte 2: Diseño de Casos de Prueba

Tabla con 10+ casos de prueba con ID, precondición, acción, resultado esperado y postcondición.

ID	Precondición	Acción	Resultado esperado	Postcondición
TC-DLL-001	Lista vacía	delete(10)	No cambia la lista	head == null, tail == null
TC-DLL-002	Lista vacía	insert(5)	Nodo creado	head == tail, prev == null, next == null
TC-DLL-003	Lista con [10]	delete(10)	Lista vacía	head == null, tail == null
TC-DLL-004	Lista con [10]	reverse()	Lista sin cambios	head == tail
TC-DLL-005	Lista con [1,2,3]	delete(1)	Head eliminado	head.prev == null
TC-DLL-006	Lista con [1,2,3]	delete(3)	Tail eliminado	tail.next == null
TC-DLL-007	Lista con [1,2,3]	delete(2)	Nodo intermedio eliminado	Enlaces correctos entre 1 y 3
TC-DLL-008	Lista vacía	search(5)	No encontrado	Retorna -1 o null
TC-DLL-009	Lista con [1]	search(1)	Encontrado	Retorna posición válida
TC-DLL-010	Lista vacía	reverse()	Lista vacía	Sin errores

Parte 3: Implementación de Pruebas

DoublyLinkedListTest.java 1 | DoublyLinkedList.java

Casos_borde > src > DoublyLinkedListTest.java > DoublyLinkedListTest > main(String[])

```java
public class DoublyLinkedListTest {

    private static DoublyLinkedList<Integer> list;

    Run | Debug
    public static void main(String[] args) {
        testEmptyList();
        testInsertOnEmptyList();
        testDeleteSingleElement();
        testDeleteHead();
        testDeleteTail();
        testDeleteMiddle();
        testSearchNotFound();
        testSearchFound();
        testReverseEmpty();
        testReverseSingle();

        System.out.println(x: "Todas las pruebas pasaron correctamente");
    }

    static void setUp() {
        list = new DoublyLinkedList<>();
    }

    static void testEmptyList() {
        setUp();
        assertNull(list.getHead(), msg: "Head debería ser null");
```

Calle Agustín Carrión Palacios entre Av. Salvador Bustamante Celi y Beethoven. Sector Jipiro.
Loja-Ecuador
Telf.: (593-7) 258 4567
admisiones@uide.edu.ec
Apartado Postal:
www.uide.edu.ec

Kevin Michael Girón Chuquirima
Estructura de datos
Facultad de Ciencias Técnicas
Ingeniería en Tecnologías de la Información

```
        assertNull(list.getTail(), msg: "Tail debería ser null");
    }

    static void testInsertOnEmptyList() {
        setUp();
        list.insert(10);
        assertEquals(list.getHead(), list.getTail(), msg: "Head y Tail deben ser iguales");
        assertNull(list.getHead().prev, msg: "Prev debería ser null");
        assertNull(list.getHead().next, msg: "Next debería ser null");
    }

    static void testDeleteSingleElement() {
        setUp();
        list.insert(5);
        list.delete(5);
        assertNull(list.getHead(), msg: "Head debería ser null");
        assertNull(list.getTail(), msg: "Tail debería ser null");
    }

    static void testDeleteHead() {
        setUp();
        list.insert(1);
        list.insert(2);
        list.delete(1);
        assertNull(list.getHead().prev, msg: "Prev del head debería ser null");
    }

    static void testDeleteTail() {
        setUp();
        list.insert(1);
        list.insert(2);
        list.delete(2);
        assertNull(list.getTail().next, msg: "Next del tail debería ser null");
    }

    static void testDeleteMiddle() {
        setUp();
        list.insert(1);
        list.insert(2);
        list.insert(3);
        list.delete(2);
        assertEquals(Integer.valueOf(i: 3), list.getHead().next.data,
        msg: "El siguiente del head debería ser 3");
    }

    static void testSearchNotFound() {
        setUp();
        assertEquals(-1, list.search(99), msg: "Elemento no debería encontrarse");
    static void testSearchFound() {
        setUp();
        list.insert(7);
        assertEquals(expected: 0, list.search(7), msg: "Elemento debería encontrarse en índice 0");
    }

    static void testReverseEmpty() {
        setUp();
        list.reverse();
        assertNull(list.getHead(), msg: "Head debería ser null");
        assertNull(list.getTail(), msg: "Tail debería ser null");
    }

    static void testReverseSingle() {
        setUp();
        list.insert(4);
        list.reverse();
        assertEquals(list.getHead(), list.getTail(), msg: "Head y Tail deben ser iguales");
    }


    static void assertNull(Object obj, String msg) {
        if (obj != null) {
            throw new RuntimeException("Fallo: " + msg);
        }
    }
    static void assertEquals(Object expected, Object actual, String msg) {
        if (expected == null && actual == null) return;
        if (expected != null && expected.equals(actual)) return;
        throw new RuntimeException(
            "Fallo: " + msg + " | esperado=" + expected + ", actual=" + actual
        );
    }

    static void assertEquals(int expected, int actual, String msg) {
        if (expected != actual) {
            throw new RuntimeException(
                "Fallo: " + msg + " | esperado=" + expected + ", actual=" + actual
            );
        }
    }
}
```

Calle Agustín Carrión Palacios entre Av. Salvador Bustamante Celi y Beethoven. Sector Jipiro.
Loja-Ecuador
Telf.: (593-7) 258 4567
admisiones@uide.edu.ec
Apartado Postal:
www.uide.edu.ec

Resultados:

```
C:\Users\Kevin\Documents\Casos_borde\src> javac *.java
C:\Users\Kevin\Documents\Casos_borde\src> java DoublyLinkedListTest
Todas las pruebas pasaron correctamente
```

- testEmptyList verifica que una lista recién creada no tenga nodos, por lo que tanto la cabeza (head) como la cola (tail) deben ser nulas.
- testInsertOnEmptyList comprueba que al insertar un único elemento, la cabeza y la cola apunten al mismo nodo y que este no tenga referencias anteriores ni siguientes.
- testDeleteSingleElement valida que al eliminar el único elemento de la lista, esta vuelva a quedar vacía.
- testDeleteHead y testDeleteTail evalúan la correcta eliminación del primer y último elemento respectivamente, asegurando que los enlaces se actualicen correctamente.
- testDeleteMiddle comprueba la eliminación de un nodo intermedio, verificando que los enlaces entre los nodos restantes se mantengan coherentes.
- testSearchNotFound y testSearchFound validan el método de búsqueda, comprobando tanto el caso en que el elemento no existe como cuando sí está presente.
- testReverseEmpty y testReverseSingle aseguran que la operación de invertir la lista no genere errores cuando la lista está vacía o contiene un solo elemento.

Parte 4: Reporte de Cobertura

Resumen de ejecución

Total de pruebas	Pasadas	Fallidas
10	10	0

Análisis de cobertura

Operaciones cubiertas:

- Inserción
- Eliminación (head, tail, intermedio, único)
- Búsqueda
- Inversión
- Manejo de lista vacía

Cobertura lógica: Alta
Cobertura de casos borde: Completa según especificación
Gaps identificados:

- Inserción en posiciones específicas
- Manejo de valores duplicados
- Pruebas de estrés (listas grandes)
- Verificación de integridad tras múltiples operaciones consecutivas

Propuestas de mejora:

- Agregar pruebas de rendimiento
- Incluir pruebas con tipos genéricos distintos
- Medir cobertura con JaCoCo
- Automatizar pruebas en CI (GitHub Actions)

Referencias:

Karumanchi, N. (2020). Data Structures and Algorithms Made Easy in Java (pp. 92–109). CareerMonk Publications.

JUnit 5 User Guide. (s. f.). JUnit Documentation. Recuperado de la documentación oficial de JUnit 5.

Presentación Semana 7. Listas Doblemente Enlazadas: Casos Borde.

Presentación Semana 6. Pruebas de Caja Negra. Investigación revisada correspondiente a la actividad TA-2.1.