JUnit Testing

The main advantages of using JUnit are that it provides a clear structure for test scripts, it's easy to work with, and it helps ensure that the code is working as expected. It's also helpful in the process of debugging and can be used effectively in conjunction with build tools such as Maven or Gradle.

JUnit tests allow you to write code faster, which increases quality. They make it possible to ensure that your code is working as expected, that it handles failures gracefully, and that it remains intact after alterations.

JUnit is a widely used testing framework for the Java programming language. It plays a crucial role in the development of test-driven development methodologies by providing an easy-to-use set of assertion methods to check the correctness of the code.

The main advantages of using JUnit are that it provides a clear structure for test scripts, is easy to work with, and helps ensure that the code is working as expected. It is also helpful in the process of debugging and can be effectively used in conjunction with build tools such as Maven or Gradle.

JUnit tests enable faster code writing, leading to increased quality. They make it possible to ensure that the code works as expected, handles failures gracefully, and remains intact after alterations.

Linked List System

```
public class LinkedList {
    private Node head;

    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public LinkedList() {
        this.head = null;
    }

    public void add(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            current.next = newNode;
        }
    }

    public boolean contains(int data) {
        Node current = head;
        while (current != null) {
            if (current.data == data) {
                return true;
            }
            current = current.next;
        }
        return false;
    }

    public boolean remove(int data) {
        if (head == null) {
            return false;
        }

        if (head.data == data) {
            head = head.next;
            return true;
        }

        Node current = head;
        while (current.next != null && current.next.data != data
            current = current.next;
        }

        if (current.next == null) {
            return false;
        }

        current.next = current.next.next;
        return true;
    }

    public int size() {
        int count = 0;
        Node current = head;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }
}
```

Test Class

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class LinkedListTest {
    private LinkedList list;

    @BeforeEach
    public void setUp() {
        list = new LinkedList();
    }

    @Test
    public void testAdd() {
        list.add(1);
        list.add(2);
        list.add(3);
        assertEquals(3, list.size());
    }

    @Test
    public void testContains() {
        list.add(1);
        list.add(2);
        list.add(3);
        assertTrue(list.contains(2));
        assertFalse(list.contains(4));
    }

    @Test
    public void testRemove() {
        list.add(1);
        list.add(2);
        list.add(3);
        assertTrue(list.remove(2));
        assertFalse(list.contains(2));
        assertEquals(2, list.size());

        assertFalse(list.remove(4));
        assertEquals(2, list.size());
    }

    @Test
    public void testSize() {
        assertEquals(0, list.size());
        list.add(1);
        assertEquals(1, list.size());
        list.add(2);
        assertEquals(2, list.size());
    }
}
```

Stack System

```
public class Stack {
    private Node top;
    private static class Node {
        int data;
        Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public Stack() {
        this.top = null;
    }

    public void push(int data) {
        Node newNode = new Node(data);
        newNode.next = top;
        top = newNode;
    }

    public int pop() {
        if (top == null) {
            throw new IllegalStateException("Stack is empty");
        }
        int data = top.data;
        top = top.next;
        return data;
    }

    public int peek() {
        if (top == null) {
            throw new IllegalStateException("Stack is empty");
        }
        return top.data;
    }
    public boolean isEmpty() {
        return top == null;
    }

    public int size() {
        int count = 0;
        Node current = top;
        while (current != null) {
            count++;
            current = current.next;
        }
        return count;
    }
}
```

Test Class

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class StackTest {
    private Stack stack;

    @BeforeEach
    public void setUp() {
        stack = new Stack();
    }

    @Test
    public void testPush() {
        stack.push(1);
        stack.push(2);
        stack.push(3);
        assertEquals(3, stack.size());
    }

    @Test
    public void testPop() {
        stack.push(1);
        stack.push(2);
        stack.push(3);
        assertEquals(3, stack.pop());
        assertEquals(2, stack.pop());
        assertEquals(1, stack.pop());
        assertTrue(stack.isEmpty());
    }

    @Test
    public void testPeek() {
        stack.push(1);
        stack.push(2);
        assertEquals(2, stack.peek());
        assertEquals(2, stack.size());
    }

    @Test
    public void testIsEmpty() {
        assertTrue(stack.isEmpty());
        stack.push(1);
        assertFalse(stack.isEmpty());
    }

    @Test
    public void testSize() {
        assertEquals(0, stack.size());
        stack.push(1);
        assertEquals(1, stack.size());
        stack.push(2);
        assertEquals(2, stack.size());
    }

    @Test
    public void testPopEmptyStack() {
        assertThrows(IllegalStateException.class, () -> {
            stack.pop();
        });
    }

    @Test
    public void testPeekEmptyStack() {
        assertThrows(IllegalStateException.class, () -> {
            stack.peek();
        });
    }
}
```