# Lesson-4 – Class Notes, JUnit Basics and Testing in IntelliJ IDEA

**When to use Iteration :**

- If memory and performance are the priority
- Loops for mathematical operations (Factorial, Fibonacci, Prime Checking)
- Dynamic Programming(Will discuss in Algorithms)

**When to use Recursion: (**Will get knowledge in Algorithms course)

- Natural recursion structure exists.
- Tree Traversals (Binary Trees, Graph DFS)
- Divide & Conquer Algorithms (Merge Sort, Quick Sort, etc.,)
- Combinatorial Problems (Backtracking, N-Queens)

**What is Unit Test, JUnit and why use Junit?**

- A **Unit test** is a type of software test that focuses on verifying the correctness of a **single unit of code** (such as a method or function) in isolation.
- JUnit is a widely used **unit testing framework for Java** that helps in writing and running automated tests.
- Supports **automated testing** to detect issues early and reduce manual errors.
- Plays a key role in **Test-Driven Development (TDD)** by enabling tests to be written before the actual implementation.
- Ensures **code reliability, correctness, and maintainability** by catching defects at the unit level.
- Reduces debugging time by providing **clear test reports** on what works and what fails.
- Enhances team collaboration by ensuring **consistent and reusable** test cases for code verification.

## Important JUnit Annotations

| Annotation | Description |
|---|---|
| `@Test` | Marks a method as a test case. |
| `@BeforeEach` | Runs before each test case. Used for setup. |
| `@AfterEach` | Runs after each test case. Used for cleanup. |
| `@BeforeAll` | Runs once before all tests (static method). |
| `@AfterAll` | Runs once after all tests (static method). |
| `@Disabled` | Skips a test case temporarily. |

### Assertion Methods

| Method | Purpose |
|---|---|
| `assertEquals(expected, actual)` | Checks if values are equal. |
| `assertTrue(condition)` | Passes if the condition is true. |
| `assertFalse(condition)` | Passes if the condition is false. |
| `assertNotNull(object)` | Ensures the object is not null. |

# Writing a Simple JUnit Test

## Example 1

### Step 1: Create a Class `MyClass.java`

```
public class MyClass {
    public static int fact(int num) {
        if(num == 0 || num == 1)
            return 1;
        else
            return num * fact(num - 1);
    }
    public static int Sum(int n) {
        if (n == 1)
            return 1;
        else
            return Sum(n-1) + n;

    }
}
```

### Step 2: Create a Test Class MyClassTest.`java`

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

public class MyClassTest {
@Test
    public void test1(){
    int act = MyClass.fact(3);
    int exp = 6;
    assertEquals(exp,act);
}
@Test
    public void test2(){
    int act = MyClass.Sum(4);
    int exp = 10;
    assertEquals(exp,act);
}
}
```

- **Run the test**: Right-click on the test class and select **Run** '`MyClassTest.java`'.

---

# Using `@BeforeEach` and `@AfterEach` with Multiple Test Cases

## Example 2

### Step 1: Create a `BankAccount` Class

```
public class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            throw new IllegalArgumentException("Insufficient balance");
        }
    }

    public double getBalance() {
        return balance;
    }
}
```

### Step 2: Write JUnit 5 Test Class `BankAccountTest.java`

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class BankAccountTest {
    private BankAccount account;

    @BeforeEach
    void setUp() {
        System.out.println("Setting up a new BankAccount with $100
balance.");
        account = new BankAccount(100); // Initialize before each test
    }

    @AfterEach
    void tearDown() {
```

```
        System.out.println("Test completed. Cleaning up...");
    }

@Test
    void testDeposit() {
        account.deposit(50);
        assertEquals(150, account.getBalance(), "Balance should be 150 after
deposit of 50");
    }

@Test
    void testWithdraw() {
        account.withdraw(40);
        // Before executing this test-case balance changed to 100
        assertEquals(60, account.getBalance(), "Balance should be 60 after
                    withdrawal of 40");
    }
      // Check this part after Lesson-12 Exception Handling
}
```

---

# Running JUnit Tests in IntelliJ IDEA

- **Right-click on the test class → Run '`BankAccountTest.java`'.**

---