**JUNIT:**

1) **Factorial Test :**

```
public class MathUtil {
   public static int factorial(int n) {
      if (n < 0) throw new IllegalArgumentException("Negative not allowed");
      int result = 1;
      for (int i = 2; i <= n; i++) result *= i;
      return result;
   }
}
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MathUtilTest {
   @Test
   void testFactorialValid() {
      assertEquals(120, MathUtil.factorial(5));
      assertEquals(1, MathUtil.factorial(0));
   }

   @Test
   void testFactorialNegative() {
      assertThrows(IllegalArgumentException.class, () -> MathUtil.factorial(-1));
   }
}
```

---

2) **Reverse String Test:**

```
public class StringUtil {
   public static String reverse(String input) {
      if (input == null) return null;
      return new StringBuilder(input).reverse().toString();
   }
}
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class StringUtilTest {
   @Test
   void testReverseNormal() {
      assertEquals("cba", StringUtil.reverse("abc"));
   }

   @Test
   void testReverseEmpty() {
      assertEquals("", StringUtil.reverse(""));
   }

   @Test
   void testReverseNull() {
      assertNull(StringUtil.reverse(null));
   }
}
```

3) **Login Validation:**

```
public class AuthService {
    public static boolean validate(String username, String password) {
        return "admin".equals(username) && "1234".equals(password);
    }
}
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class AuthServiceTest {
    @Test
    void testValidLogin() {
        assertTrue(AuthService.validate("admin", "1234"));
    }

    @Test
    void testInvalidLogin() {
        assertFalse(AuthService.validate("admin", "wrong"));
        assertFalse(AuthService.validate("user", "1234"));
    }

    @Test
    void testEmptyAndNullLogin() {
        assertFalse(AuthService.validate("", ""));
        assertFalse(AuthService.validate(null, null));
    }
}
```

---

4) **StudentService with Mocked StudentRepository:**

```
class Student {
    int id;
    String name;
    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }
}
interface StudentRepository {
    Student findById(int id);
}
class StudentService {
    StudentRepository repo;

    StudentService(StudentRepository repo) {
        this.repo = repo;
    }

    String getStudentById(int id) {
        Student s = repo.findById(id);
        return s != null ? s.name : null;
    }
}
import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class StudentServiceTest {
    @Test
    void testGetStudent() {
        StudentRepository repo = mock(StudentRepository.class);
```

```
        when(repo.findById(1)).thenReturn(new Student(1, "Ravi"));

        StudentService service = new StudentService(repo);
        assertEquals("Ravi", service.getStudentById(1));
    }
}
```

---

**5) OrderService Calls PaymentService:**

```
interface PaymentService {
    void processPayment();
}
class OrderService {
    PaymentService payment;

    OrderService(PaymentService payment) {
        this.payment = payment;
    }

    void placeOrder() {
        payment.processPayment();
    }
}
import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;

public class OrderServiceTest {
    @Test
    void testPlaceOrder() {
        PaymentService payment = mock(PaymentService.class);
        OrderService order = new OrderService(payment);

        order.placeOrder();

        verify(payment, times(1)).processPayment();
    }
}
```

---

**6) Divide Method Exception:**

```
public class Calculator {
    public static int divide(int a, int b) {
        if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");
        return a / b;
    }
}
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {
    @Test
    void testDivideValid() {
        assertEquals(5, Calculator.divide(10, 2));
    }

    @Test
    void testDivideByZero() {
        assertThrows(IllegalArgumentException.class, () -> Calculator.divide(10, 0));
    }
}
```

7) **Full Banking System Test:**

```
class Account {
   int id;
   double balance;

   Account(int id, double balance) {
      this.id = id;
      this.balance = balance;
   }
}

interface AccountRepository {
   Account findById(int id);
   void update(Account acc);
}

interface NotificationService {
   void send(String msg);
}

class AccountService {
   AccountRepository repo;
   NotificationService notifier;

   AccountService(AccountRepository repo, NotificationService notifier) {
      this.repo = repo;
      this.notifier = notifier;
   }

   void transfer(int fromId, int toId, double amount) {
      Account from = repo.findById(fromId);
      Account to = repo.findById(toId);

      if (from == null || to == null)
         throw new RuntimeException("Account not found");

      if (from.balance < amount)
         throw new RuntimeException("Insufficient balance");

      from.balance -= amount;
      to.balance += amount;

      repo.update(from);
      repo.update(to);
      notifier.send("Transferred ₹" + amount + " from " + fromId + " to " + toId);
   }
}
```

**Test Class:**

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

public class AccountServiceTest {
   @Test
   void testSuccessfulTransfer() {
      AccountRepository repo = mock(AccountRepository.class);
      NotificationService notifier = mock(NotificationService.class);

      Account from = new Account(1, 1000);
      Account to = new Account(2, 500);
```

```
        when(repo.findById(1)).thenReturn(from);
        when(repo.findById(2)).thenReturn(to);

        AccountService service = new AccountService(repo, notifier);
        service.transfer(1, 2, 200);

        assertEquals(800, from.balance);
        assertEquals(700, to.balance);

        verify(repo, times(2)).update(any());
        verify(notifier).send("Transferred ₹200.0 from 1 to 2");
    }

    @Test
    void testInsufficientBalance() {
        AccountRepository repo = mock(AccountRepository.class);
        NotificationService notifier = mock(NotificationService.class);

        Account from = new Account(1, 100);
        Account to = new Account(2, 500);

        when(repo.findById(1)).thenReturn(from);
        when(repo.findById(2)).thenReturn(to);

        AccountService service = new AccountService(repo, notifier);

        assertThrows(RuntimeException.class, () -> service.transfer(1, 2, 200));
    }

    @Test
    void testAccountNotFound() {
        AccountRepository repo = mock(AccountRepository.class);
        NotificationService notifier = mock(NotificationService.class);

        when(repo.findById(1)).thenReturn(null);

        AccountService service = new AccountService(repo, notifier);

        assertThrows(RuntimeException.class, () -> service.transfer(1, 2, 100));
    }
}
```