**Setting Up JUnit**

**JUnit\_Basic Testing**

Vaishnavi

28/6/2025

JUnit is a Java testing framework used to write unit tests — small pieces of code that test individual methods in your program.

This Java project showcases the use of **JUnit 5** assertions to test the functionality of a BankAccount class, verifying deposit, withdrawal, balance handling, and validation logic through unit testing.

**Objective:**

* **Verify Core Functionalities:** Ensure correct behavior of account operations like deposit, withdrawal, and balance tracking.
* **Handle Invalid Scenarios:** Use assertions to confirm that exceptions are properly thrown for invalid operations (e.g., overdrafts, negative deposits).
* **Assess Logical Conditions:** Test conditions such as checking account owner details and evaluating wealth status (isRich() method).

**Implementation:**

### Create a New Java Project

**IntelliJ IDEA**: File → New → Project → Java

### Add JUnit to Your Project

#### By using ****Maven****: add the dependencies in pom.xml

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**Create a BankAccount Java Class and Test Class**

**BankAccount.java:**

public class BankAccount {

private String owner;

private double balance;

public BankAccount(String owner, double initialAmount) {

this.owner = owner;

this.balance = initialAmount;

}

public void deposit(double amount) {

if (amount <= 0) throw new IllegalArgumentException("Deposit must be positive");

balance += amount;

}

public void withdraw(double amount) {

if(amount>balance) throw new IllegalArgumentException("Insufficient funds");

balance -= amount;

}

public double getBalance() {

return balance;

}

public String getOwner() {

return owner;

}

public boolean isRich() {

return balance >= 1\_00\_000;

}

}

**BankAccountTest.java:**

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

import org.junit.jupiter.api.Test;

public class BankAccountTest {

@Test

public void testInitialBalance() {

BankAccount acc = new BankAccount("Karthi", 5000);

System.out.println("Initial balance: ₹" + acc.getBalance());

assertEquals(5000, acc.getBalance());

}

@Test

public void testDeposit() {

BankAccount acc = new BankAccount("Karthi", 2000);

acc.deposit(3000);

System.out.println("After deposit, balance: ₹" + acc.getBalance());

assertEquals(5000, acc.getBalance());

}

@Test

public void testWithdraw() {

BankAccount acc = new BankAccount("Karthi", 7000);

acc.withdraw(2000);

System.out.println("After withdrawal, balance: ₹" + acc.getBalance());

assertEquals(5000, acc.getBalance());

}

@Test

public void testOverWithdraw() {

BankAccount acc = new BankAccount("Karthi", 1000);

System.out.println("Trying to withdraw ₹2000 from ₹1000 balance");

assertThrows(IllegalArgumentException.class, () -> acc.withdraw(2000));

}

@Test

public void testNegativeDeposit() {

BankAccount acc = new BankAccount("Karthi", 1000);

System.out.println("Trying to deposit -100");

assertThrows(IllegalArgumentException.class, () -> acc.deposit(-100));

}

@Test

public void testOwnerName() {

BankAccount acc = new BankAccount("Karthi", 1000);

System.out.println("Account belongs to: " + acc.getOwner());

assertEquals("Karthi", acc.getOwner());

}

@Test

public void testMultipleDeposits() {

BankAccount acc = new BankAccount("Karthi", 1000);

acc.deposit(1000);

acc.deposit(2000);

acc.deposit(3000);

System.out.println("After multiple deposits, balance: ₹" + acc.getBalance());

assertEquals(7000, acc.getBalance());

}

@Test

public void testMultipleWithdrawals() {

BankAccount acc = new BankAccount("Karthi", 10000);

acc.withdraw(2000);

acc.withdraw(3000);

System.out.println("After multiple withdrawals, balance: ₹" + acc.getBalance());

assertEquals(5000, acc.getBalance());

}

@Test

public void testIsRichFalse() {

BankAccount acc = new BankAccount("Karthi", 90000);

System.out.println("Is account rich? " + acc.isRich());

assertFalse(acc.isRich());

}

@Test

public void testIsRichTrue() {

BankAccount acc = new BankAccount("Karthi", 100000);

System.out.println("Is account rich? " + acc.isRich());

assertTrue(acc.isRich());

}

@Test

public void testExactWithdraw() {

BankAccount acc = new BankAccount("Karthi", 5000);

acc.withdraw(5000);

System.out.println("Balance after exact withdrawal: ₹" + acc.getBalance());

assertEquals(0, acc.getBalance());

}

@Test

public void testBalanceNotNegative() {

BankAccount acc = new BankAccount("Karthi", 3000);

try {

acc.withdraw(5000);

} catch (Exception e) {

System.out.println("Exception caught: " + e.getMessage());

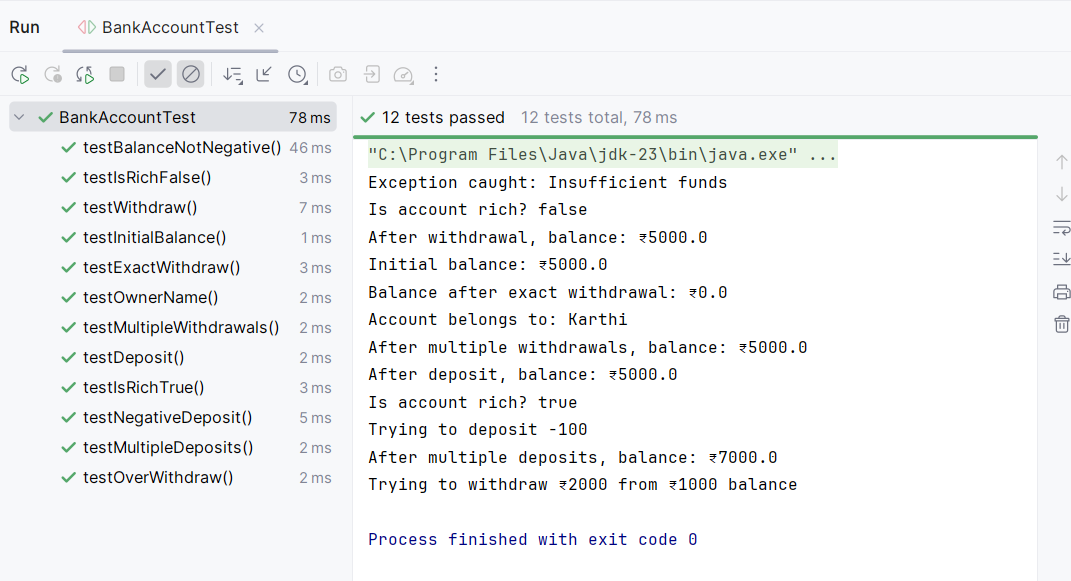
}

assertTrue(acc.getBalance() >= 0);

}

}

**Output:**

****