EX.NO.: 01
DATE: 16.06.2025

BANK ACCOUNT - JUNIT TESTING

AIM

To implement a simple Bank Account system in Java supporting initialization, deposit, withdrawal, and balance checking, along with JUnit 5 test cases to validate its correctness, including both passing and intentionally failing scenarios.

ALGORITHM

1. Define a `BankAccount` class with a private `balance` variable.
2. Initialize the account ensuring the initial balance is non-negative.
3. Implement a `deposit` method that adds a positive amount to the balance; throw an exception for invalid amounts.
4. Implement a `withdraw` method that deducts a positive amount from the balance; throw an exception if the amount is invalid or exceeds the balance.
5. Implement a `getBalance` method to return the current balance.
6. Write JUnit 5 test cases to verify:
7. Valid and invalid account initialization.
8. Valid and invalid deposit operations.
9. Valid and invalid withdrawal operations.
10. Proper exception handling.
11. Intentionally make some test cases fail by providing wrong expected values to observe JUnit’s failure reporting.

CODE AND OUTPUT

```java
public class BankAccount {
    private double balance;

    // Constructor
    public BankAccount(double initialBalance) {
        if (initialBalance < 0) {
            throw new IllegalArgumentException("Initial balance cannot be negative.");
        }
        this.balance = initialBalance;
    }

    // Deposit money
    public void deposit(double amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Deposit amount must be positive.");
        }
        balance += amount;
    }

    // Withdraw money
    public void withdraw(double amount) {
        if (amount <= 0) {
            throw new IllegalArgumentException("Withdrawal amount must be positive.");
        }
        if (amount > balance) {
            throw new IllegalArgumentException("Insufficient balance.");
        }
        balance -= amount;
    }

    // Get current balance
    public double getBalance() {
        return balance;
    }
}
```

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

class BankAccountTest {

    @Test
    void testValidInitialization() {
        BankAccount account = new BankAccount(100.0);
        assertEquals(100.0, account.getBalance());  // Should pass
    }

    @Test
    void testInvalidInitializationNegativeBalance() {
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            new BankAccount(-50.0);
        });
        assertEquals("Initial balance cannot be negative.", exception.getMessage());
// Should pass
    }

    @Test
    void testDepositValidAmount() {
        BankAccount account = new BankAccount(100.0);
        account.deposit(50.0);
        assertEquals(200.0, account.getBalance()); // ❌ Intentionally wrong expected
value; will FAIL
    }

    @Test
    void testDepositInvalidAmountZeroOrNegative() {
        BankAccount account = new BankAccount(100.0);
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            account.deposit(0);
        });
        assertEquals("Deposit amount must be positive.", exception.getMessage());  //
Should pass
```

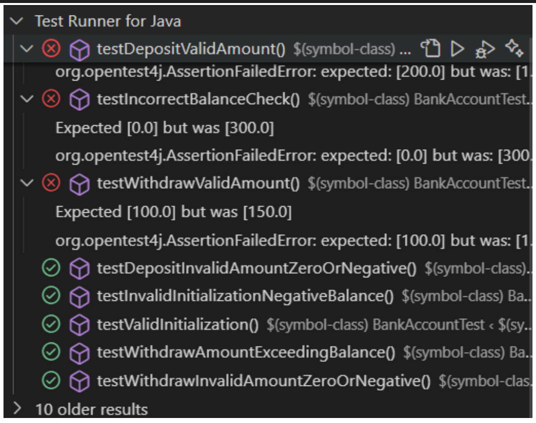
```java
    }

    @Test
    void testWithdrawValidAmount() {
        BankAccount account = new BankAccount(200.0);
        account.withdraw(50.0);
        assertEquals(100.0, account.getBalance()); // ❌ Intentionally wrong expected value; will FAIL
    }

    @Test
    void testWithdrawInvalidAmountZeroOrNegative() {
        BankAccount account = new BankAccount(100.0);
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            account.withdraw(-10);
        });
        assertEquals("Withdrawal amount must be positive.", exception.getMessage()); // Should pass
    }

    @Test
    void testWithdrawAmountExceedingBalance() {
        BankAccount account = new BankAccount(100.0);
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            account.withdraw(150.0);
        });
        assertEquals("Insufficient balance.", exception.getMessage());  // Should pass
    }

    // Extra failing test for demonstration
    @Test
    void testIncorrectBalanceCheck() {
        BankAccount account = new BankAccount(300.0);
        assertEquals(0.0, account.getBalance()); // ❌ This will FAIL since balance is 300
    }
}
```



```
Test Runner for Java
  ⊗ testDepositValidAmount() $(symbol-class) ...
      org.opentest4j.AssertionFailedError: expected: [200.0] but was: [1
  ⊗ testIncorrectBalanceCheck() $(symbol-class) BankAccountTest..
      Expected [0.0] but was [300.0]
      org.opentest4j.AssertionFailedError: expected: [0.0] but was: [300
  ⊗ testWithdrawValidAmount() $(symbol-class) BankAccountTest..
      Expected [100.0] but was [150.0]
      org.opentest4j.AssertionFailedError: expected: [100.0] but was: [1
  ✓ testDepositInvalidAmountZeroOrNegative() $(symbol-class)..
  ✓ testInvalidInitializationNegativeBalance() $(symbol-class) Ba..
  ✓ testValidInitialization() $(symbol-class) BankAccountTest ‹ $(sy..
  ✓ testWithdrawAmountExceedingBalance() $(symbol-class) Ba..
  ✓ testWithdrawInvalidAmountZeroOrNegative() $(symbol-clas
> 10 older results
```