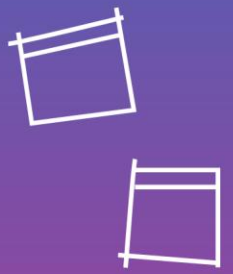
DBC
DIGITAL BUSINESS COMPANY®

VEM SER
DBC



Testes Unitários

Aula 1 – JUnit

Conteúdo do Módulo 4

- Testes Unitários (2 aula)
- MongoDB (3 aulas)
- TF6 (MongoDB + Testes)
- Apache Kafka (2 aulas e 1/2)
- Scheduler com SpringBoot (1/2 aulas)
- TF7 (Kafka + Scheduler)

Conteúdo do Módulo 4.2 Testes

- Testes Unitários
- JUnit
- Mockito

Conteúdo da Aula

- Testes Unitários
- JUnit
- Homework

Tipos de Testes

- Testes unitários
- Testes de integração
- Testes ponta à ponta
- Testes manuais
- Testes automatizados

Tipos de Testes

- **Testes unitários**
- Testes de integração
- Testes ponta à ponta
- Testes manuais
- Testes automatizados

Testes Unitários

https://www.devmedia.com.br/junit-tutorial/1432

Testes Unitários

- Se concentra na verificação da menor unidade do projeto de software

https://www.devmedia.com.br/junit-tutorial/1432

Testes Unitários

- Se concentra na verificação da menor unidade do projeto de software
- Geralmente são realizados de forma isolada do restante do sistema, visto que tem por objetivo assegurar a qualidade das unidades de forma individual e não o sistema como um todo.

https://www.devmedia.com.br/junit-tutorial/1432

Testes Unitários

- Se concentra na verificação da menor unidade do projeto de software
- Geralmente são realizados de forma isolada do restante do sistema, visto que tem por objetivo assegurar a qualidade das unidades de forma individual e não o sistema como um todo.
- Essa unidade pode ser identificada como um método, uma classe ou mesmo um objeto

https://www.devmedia.com.br/junit-tutorial/1432

Porque usar Testes Unitários?

Porque usar Testes Unitários?

- Previne contra o aparecimento de “BUG’S” oriundos de códigos mal escritos;

Porque usar Testes Unitários?

- Previne contra o aparecimento de “BUG’S” oriundos de códigos mal escritos;
- Código testado é mais confiável;

Porque usar Testes Unitários?

- Previne contra o aparecimento de “BUG’S” oriundos de códigos mal escritos;
- Código testado é mais confiável;
- Testa situações de sucesso e de falha;

Porque usar Testes Unitários?

- Previne contra o aparecimento de “BUG’S” oriundos de códigos mal escritos;
- Código testado é mais confiável;
- Testa situações de sucesso e de falha;
- Resulta em outras práticas XP como: Código coletivo, refatoração, integração contínua;

Porque usar Testes Unitários?

- Previne contra o aparecimento de “BUG’S” oriundos de códigos mal escritos;
- Código testado é mais confiável;
- Testa situações de sucesso e de falha;
- Resulta em outras práticas XP como: Código coletivo, refatoração, integração contínua;
- Gera e preserva um “conhecimento” sobre as regras de negócios do projeto.

Testes Unitários

- Adicionar dependência
- Criar classe e métodos de teste
- Descrever as validações a serem feitas (casos de teste)

Casos De Testes

- Conjunto de condições usadas para teste de software
- Garantir que os requisitos do software que foi construído sejam plenamente atendidos
- O caso de teste deve especificar os valores de entrada e os resultados esperados do processamento

Dependência

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

JUnit

```
@Test
public void deveTestarSeOValorEhPar(){

}
```

JUnit

```
@Test
public void deveTestarSeOValorEhPar(){
    //SETUP (CRIAÇÃO DAS VARIÁVEIS DA UNIDADE DE TESTES)
    int valorPar = 10;

}
```

JUnit

```
@Test
public void deveTestarSeOValorEhPar(){
    //SETUP (CRIAÇÃO DAS VARIÁVEIS DA UNIDADE DE TESTES)
    int valorPar = 10;

    //ACT (AÇÃO DO TESTE A SER EXECUTADO)
    boolean par = valorPar % 2 == 0;
}
```

JUnit

```java
@Test
public void deveTestarSeOValorEhPar(){
    //SETUP (CRIAÇÃO DAS VARIÁVEIS DA UNIDADE DE TESTES)
    int valorPar = 10;

    //ACT (AÇÃO DO TESTE A SER EXECUTADO)
    boolean par = valorPar % 2 == 0;

    //ASSERT (TESTE DA UNIDADE)
    assertTrue(par);
}
```

DBC
DIGITAL BUSINESS COMPANY®

show me your code;

VEM SER
DBC

Documentação

- https://junit.org/junit5/docs/current/user-guide/

O que eu devo testar?

O que eu devo testar?

- Regras de negócio (services)
- Classes utilitárias
- Códigos com lógica e complexidade

O que eu não devo testar?

O que eu não devo testar?

- Conexão com banco de dados
- Conexão com API
- Get/Set
- Controllers...

#Homework

- Criar pasta “modulo 4.2” no seu git
- Copiar o seu último projeto da conta corrente da aula de orientação à objetos
- Criar uma classe ContaTest.java no seu projeto e desenvolver os casos de testes do próximo slide

Casos De Testes

- **deveTestarSaqueContaCorrenteEVerificarSaldoComSucesso**
- **deveTestarSaqueContaCorrenteSemSaldo**: não dar certo o valor do saque (saque > saldo + ce)
- **deveTestarSaqueContaPoupancaEVerificarSaldoComSucesso**: deve creditar taxa antes
- **deveTestarSaqueContaPoupancaSemSaldo**: não dar certo o valor do saque (saque > saldo)
- **deveTestarSaqueContaPagamentoEVerificarSaldoComSucesso**
- **deveTestarSaqueContaPagamentoSemSaldo**: não dar certo o valor do saque (saque > saldo)
- **deveTestarTransferenciaEVerificarSaldoComSucesso**
- **deveTestarTransferenciaSemSaldo**: não dar certo o valor do saque (saque > saldo)
- **deveTestarDepositoEVerificarSaldoComSucesso**
- **deveTestarDepositoNegativo**