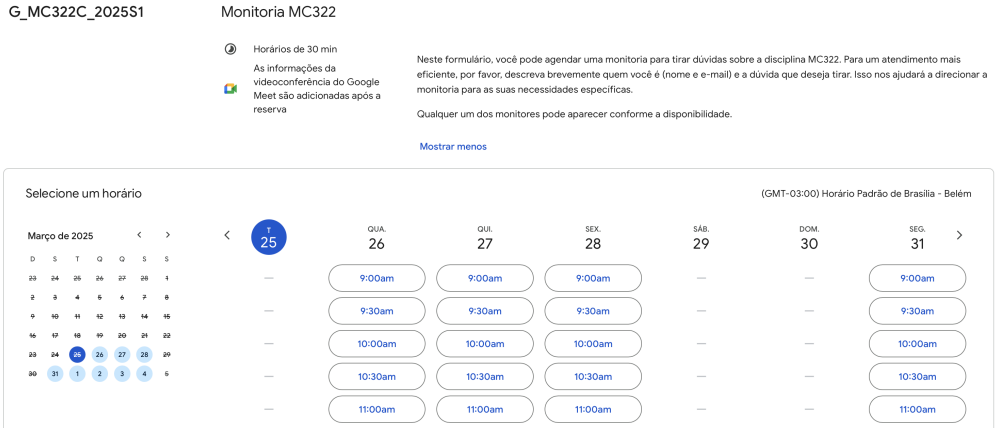
MC322 - Programação orientada a objetos
Lab 4
Monitoria e Testes

Instituto de Computação - UNICAMP

Monitoria

Marcar horário

Para agendar, acesse o link https://calendar.app.google/YbP1cktckoGLX8Yz5.



Testes

Diferentes Tipos de Testes em Java

Em Java, existem diferentes tipos de testes que podem ser realizados para garantir a qualidade do código. Aqui estão os principais tipos:

- **Testes Unitários**:
  - Focados em testar unidades individuais de código, como métodos e classes.
  - Geralmente utilizados com o framework JUnit.
  - Exemplo: Testar se um método de soma retorna o resultado correto.
- **Testes de Integração**:
  - Verificam a interação entre diferentes partes do sistema.
  - Testam se os componentes trabalham juntos conforme esperado.
  - Exemplo: Testar a interação entre a camada de persistência e a camada de serviço.
- **Testes Funcionais**:
  - Testam a funcionalidade do sistema sob condições normais de uso.
  - Avaliam o comportamento do sistema do ponto de vista do usuário final.
  - Exemplo: Testar se um formulário de cadastro envia dados corretamente.

Diferentes Tipos de Testes em Java

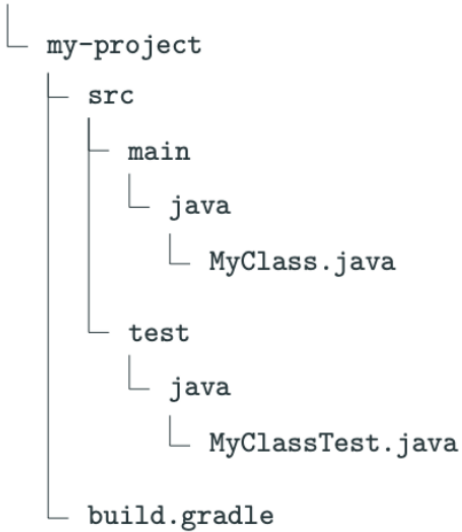
- **Testes de Aceitação**:
  - Realizados para verificar se o sistema atende aos requisitos do cliente ou usuário final.
  - Comumente realizados no final do ciclo de desenvolvimento.
  - Exemplo: Validar se todas as funcionalidades exigidas estão implementadas.
- **Testes de Regressão**:
  - Garantem que mudanças no código não introduzam erros em funcionalidades já existentes.
  - Importante após refatorações ou mudanças no sistema.
  - Exemplo: Verificar se a atualização de uma biblioteca não quebra funcionalidades anteriores.

Estrutura do Projeto

A estrutura típica de um projeto Java com Gradle para testes seria:

- `src/main/java`: código fonte principal.
- `src/test/java`: código fonte dos testes.

Estrutura do Projeto



O que são Assertions?

Assertions são uma maneira de verificar se o comportamento de um programa está correto durante a execução dos testes. As Assertions ajudam a verificar se os valores esperados e os valores reais correspondem, garantindo que o código esteja funcionando como esperado.

- As Assertions verificam se uma condição é verdadeira durante o teste.
- Se a condição for falsa, o teste falha.
- As Assertions ajudam a detectar problemas e falhas no código rapidamente.

Principais Métodos de Assertions

Há uma série de métodos para fazer as verificações de maneira simples e eficiente. Alguns dos mais utilizados são:

- `assertEquals(expected, actual)`: Verifica se os valores esperados e reais são iguais.
- `assertNotEquals(expected, actual)`: Verifica se os valores esperados e reais são diferentes.
- `assertTrue(condition)`: Verifica se a condição fornecida é verdadeira.
- `assertFalse(condition)`: Verifica se a condição fornecida é falsa.
- `assertNull(value)`: Verifica se o valor fornecido é `null`.
- `assertNotNull(value)`: Verifica se o valor fornecido não é `null`.

Teste Unitário - Exemplo de Código

Vamos criar uma classe simples em Java para exemplificar os testes.

```java
  public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}
```

Teste Unitário - Exemplo de Teste

Agora, vamos criar o teste para a classe `MyClass`.

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

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        assertEquals(5, calculator.add(2, 3));
    }

    @Test
    public void testSubtract() {
        Calculator calculator = new Calculator();
        assertEquals(1, calculator.subtract(3, 2));
    }
}
```

Teste de Integração - Exemplo de Código (User)

Vamos criar as classes que serão integradas em Java para exemplificar os testes.

```
  public class User {
    private String name;
    private String email;

    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }
}
```

Teste de Integração - Exemplo de Código (UserRepository)

```java
import java.util.ArrayList;
import java.util.List;

public class UserRepository {
    private List<User> users = new ArrayList<>();

    public void save(User user) {
        users.add(user);
    }

    public User findByEmail(String email) {
        for (User user : users) {
            if (user.getEmail().equals(email)) {
                return user;
            }
        }
        return null; // Retorna null se não encontrar
    }
}
```

Teste de Integração - Exemplo de Código (UserService)

```java
public class UserService {
    private UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User registerUser(String name, String email) {
        User user = new User(name, email);
        userRepository.save(user);
        return user;
    }
}
```

Teste de Integração - Exemplo de Teste

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

public class UserServiceIntegrationTest {

    @Test
    public void testUserRegistration() {
        // Criando inst ncias dos componentes
        UserRepository userRepository = new UserRepository();
        UserService userService = new UserService(userRepository);

        // Registrando um novo usuário
        User newUser = userService.registerUser("John Doe", "john.doe@example.com");

        // Verificando se o usuário foi salvo corretamente
        User foundUser = userRepository.findByEmail("john.doe@example.com");

        assertNotNull(foundUser, "O usuário deve ser encontrado no repositório");
        assertEquals("John Doe", foundUser.getName(), "O nome do usuário deve ser 'John Doe'");
        assertEquals("john.doe@example.com", foundUser.getEmail(), "O email do usuário deve ser
            'john.doe@example.com'");
    }
}
```

Teste Funcional - Exemplo de Código

```java
public class LoginService {

    // Método que simula o processo de login
    public boolean login(String username, String password) {
        // Usuário e senha fixos para simulação
        String validUsername = "user";
        String validPassword = "password123";

        // Verifica se o nome de usuário e a senha estão corretos
        return username.equals(validUsername) && password.equals(validPassword);
    }
}
```

Teste Funcional - Exemplo de Teste

```java
public class LoginServiceFunctionalTest {

    @Test
    public void testLoginSuccess() {
        // Criando uma inst ncia do LoginService
        LoginService loginService = new LoginService();

        // Simulando um login com credenciais válidas
        boolean loginResult = loginService.login("user", "password123");

        // Verificando se o login foi bem-sucedido
        assertTrue(loginResult, "O login deve ser bem-sucedido com as credenciais válidas");
    }

    @Test
    public void testLoginFailure() {
        // Criando uma inst ncia do LoginService
        LoginService loginService = new LoginService();

        // Simulando um login com credenciais inválidas
        boolean loginResult = loginService.login("user", "wrongpassword");

        // Verificando se o login falhou
        assertFalse(loginResult, "O login deve falhar com credenciais inválidas");
    }
}
```

Rodando os Testes

- Para rodar os testes, basta utilizar o comando do Gradle no terminal:

```
./gradlew test
```

  O Gradle irá compilar o projeto e rodar todos os testes na pasta `src/test/java`.

- **Se estiver utilizando o VScode:**
  1. Pressione `Ctrl` + `P` (`Cmd` + `P` no macOS).
  2. Irá aparecer uma linha de comando. Digite ">Gradle: Run a Gradle Build".
  3. Em seguida digite test
- Também pode-se abrir o arquivo `MyClassTest` e pressionar no botão de rodar teste manualmente.

Saída dos Testes

Após rodar o comando, você verá a saída no terminal, indicando se os testes passaram ou falharam.

Exemplo de saída:

```
> Task :test

MyClassTest > testSum() PASSED

BUILD SUCCESSFUL
```

Atividade - Objetivo

Você deve criar um sistema simples com as seguintes classes:

- **Produto**: Representa um produto com nome e preço.
- **Carrinho**: Contém métodos para adicionar produtos e calcular o total do carrinho.
- **CarrinhoService**: Um serviço que manipula o carrinho e interage com os produtos.

A tarefa é escrever os testes para garantir que o sistema funcione corretamente.

Atividade - Testes Unitários

Para os **Testes Unitários**, o objetivo é validar as funcionalidades individuais das classes do sistema, como:

- Verificar se a classe **Produto** está criando corretamente produtos com nome e preço.
- Testar os métodos de adição de produtos no **Carrinho**.
- Validar se o cálculo do total do carrinho funciona como esperado.

Exemplo: Testar se, ao adicionar dois produtos com preços 10.0 e 20.0, o total do carrinho seja 30.0.

Atividade - Testes de Integração

Para os **Testes de Integração**, o objetivo é garantir que as classes interajam corretamente:

- Verificar se a interação entre **CarrinhoService** e **Carrinho** funciona como esperado.
- Testar a adição de produtos através do **CarrinhoService** e validar se o total é calculado corretamente.

Exemplo: Ao adicionar dois produtos via **CarrinhoService**, o número total de produtos e o total calculado devem ser válidos.

Referências

- https://google.github.io/styleguide/javaguide.html
- https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

MC322 - Programação orientada a objetos
Lab 4
Monitoria e Testes

Instituto de Computação - UNICAMP