FIAP - TDS - Prof. Tiago Ferrer

Domain Driven Design

Semana 8 - 15/04/2024 : 19/04/2024

Uma introdução ao Teste Unitário

Introdução

- **Conceitos Básicos:** O que é teste unitário? Por que é importante?
- **Benefícios:** Melhorar a qualidade do código, encontrar bugs com antecedência, aumentar a confiança no código.

Configuração

- **Bibliotecas Necessárias:** JUnit 5, Mockito (para testes de simulação).
- **Configuração do Maven:** Adicionar dependências ao arquivo `pom.xml`.
- **Criando Classes de Teste:** Criar classes de teste com o sufixo “Test”.

Casos de Teste

- **Definição de Métodos de Teste:** Nomear e anotar métodos de teste com `@Test`.
- **Testes Esperados:** Definir testes para verificar comportamento esperado.
- **Falhas de Teste:** Lidar com falhas de teste e mensagens de erro.

Algumas Melhores Práticas

- **Nomeação de Teste:** Usar nomes de teste claros e descritivos.
- **Cobertura de Teste:** Escrever testes para cobrir o máximo possível de código.
- **Manutenção de Teste:** Manter os testes atualizados com as alterações de código.

Na Prática

1. Crie um projeto `maven`
2. Crie a classe `Pessoa`

```java
import java.time.LocalDate;

public class Pessoa {

    private String nome;

    private Integer anoNascimento;

    private String cpf;

    public Pessoa(String nome, Integer anoNascimento, String cpf) {
        this.nome = nome;
        this.anoNascimento = anoNascimento;
        this.cpf = cpf;
    }

    public Integer getIdade() {
        return LocalDate.now().getYear() - anoNascimento;
    }

    public boolean isMaiorIdade() {
        return getIdade() >= 18;
    }
}
```

3. Adicione a dependência do Junit

```xml
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.9.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

3. Crie a camada de teste

4. Crie um teste para validar quando `isMaiorIdade()` for verdadeiro.

5. Crie um teste para validar quando `isMaiorIdade()` for falso.

Herança

Herança é um conceito fundamental na programação orientada a objetos (OOP). Ela permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse). Isso promove reuso de código, reduzindo duplicação e melhorando a extensibilidade.

Sintaxe:

```
public class Subclass extends Superclass {
    // Código da subclasse
}
```

Diferença entre Visibilidade Protegida (protected) e Privada (private)

A diferença fundamental entre `protected` e `private` é o escopo de acesso:

- **Protected:** Os membros `protected` são acessíveis a subclasses e classes no mesmo pacote. Isso permite que as subclasses reutilizem a funcionalidade da superclasse enquanto protegem o acesso direto de classes externas.
- **Private:** Os membros `private` são acessíveis somente dentro da mesma classe. Eles não são herdados por subclasses ou acessíveis de outras classes, fornecendo o mais alto nível de encapsulamento.

A visibilidade `protected` permite que subclasses reutilizem a funcionalidade da superclasse, enquanto a visibilidade `private` protege o acesso direto a dados sensíveis ou internos. A compreensão dessas diferenças de visibilidade é essencial para projetar classes e hierarquias de herança eficazes.

Sobrescrita de Métodos

Quando uma subclasse possui um método com o mesmo nome e assinatura de outro em sua superclasse, esse método é sobrescrito. Isso permite que a subclasse forneça sua própria implementação do método.

Sintaxe:

```
class Subclass extends Superclass {
    @Override
    public void metodoSobrescrito() {
        // Implementação da subclasse
    }
}
```

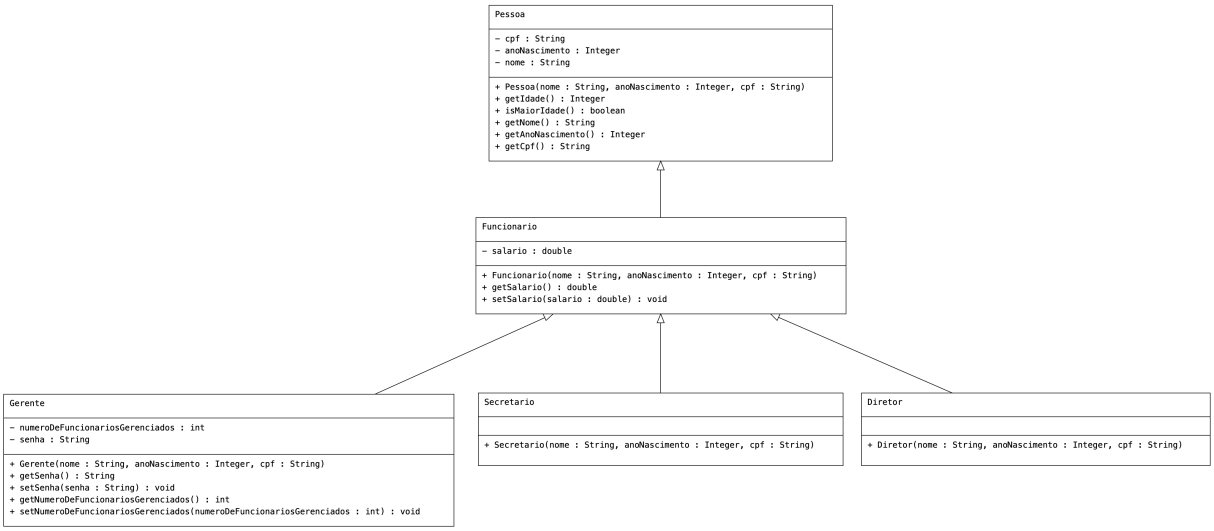
Invocações de Métodos

Uma subclasse pode invocar os métodos de sua superclasse usando a palavra-chave `super`. Isso é útil quando a subclasse precisa acessar a funcionalidade implementada na superclasse.

Sintaxe:

```
super.metodoDaSuperclasse();
```

Herança e UML



Exercício 1:

Crie uma classe `Veiculo` com as variáveis `marca` e `modelo`. Em seguida, crie uma subclasse `Carro` que herda de `Veiculo` e adicione uma variável `numeroDePortas`.

Exercício 2:

No código desenvolvido em sala, implemente o método `autenticar` da classe `Gerente`. Em seguida desenvolva o teste unitário para o caso de sucesso.