Testes de Software

Introdução

- Testes automatizados: programa que testa seu programa.
- Ideia: Testar de maneira rápida seu programa, além de encontrar possíveis falhas no seu código implementado.
- Testes de unidade: testa uma única unidade do nosso sistema
- Fluxo: Cenário -> Ação -> Validação

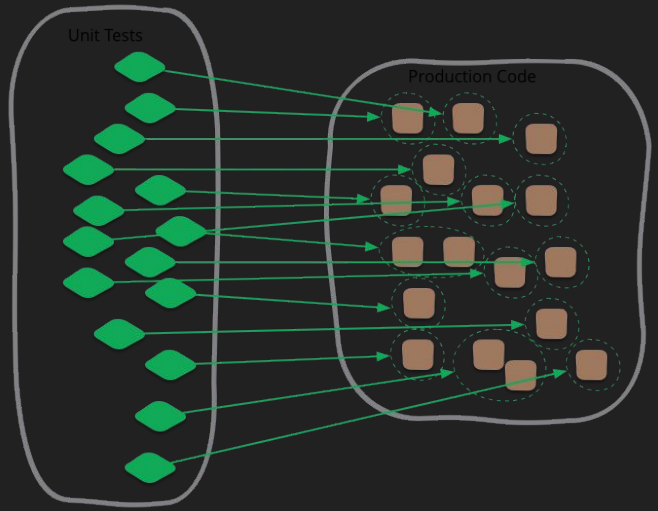
Testes de Unidade

- O testador escolhe entradas para explorar caminhos específicos e determina a saída apropriada;
- O objetivo do teste de unidade é examinar os componentes individuais;
- Verificar a funcionalidade, garantindo que o comportamento seja o esperado;
- O escopo de unidade é interpretativo;

Testes de Unidade

- Boa prática: o teste contenha a menor quantidade de código que executa uma tarefa autônoma;

- Escopos limitados e pequenos ajudam na detecção de erros.



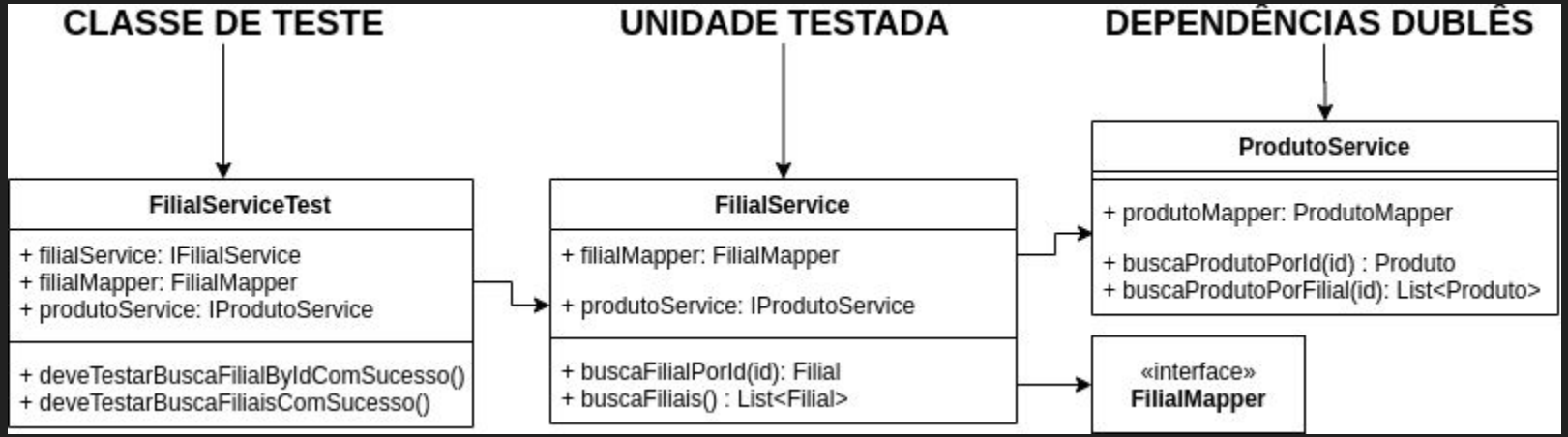
Testes de Unidade - Características

Isolado: isento de dependências externas, como bancos de dados, webservices;

Stateless: cada teste deve ter seus estados. Não se guarda estados de objetos;

Unitário: Deve testar apenas uma unidade, o resto deve ser tratado como dependência;

Testes de Unidade



Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

É um Framework de testes;

Anotação	Descrição
@Before	O método anotado será executado antes de cada método de teste na classe de teste.
@After	O método anotado será executado após cada método de teste na classe de teste.
@BeforeClass	O método anotado será executado antes de todos os métodos de teste na classe de teste

Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

É um Framework de testes;

Anotação	Descrição
@AfterClass	O método anotado será executado após todos os métodos de teste na classe de teste
@Test	É usado para marcar um método como teste junit
@Ignore	É usado para desabilitar ou ignorar uma classe de teste ou método do conjunto de testes

Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

- Ferramenta para asserções;
- Implementa asserções fluentes;
- Sintaxe Básica:

```
assertThat( instruction ).fluentMethodAssertion(expected);
```

- Sintaxe Básica para Tratamento de Exceções:

```
assertThatThrownBy( instruction ).fluentMethodAssertion(expected);
```

Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

- Framework de testes;
- Permite a criação de objetos de teste simulados em testes de unidade automatizados com a finalidade de desenvolvimento dirigido a teste (TDD) ou orientado a comportamento (BDD);
- Necessidade de garantir que os objetos executem os comportamentos que se esperam deles;

Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

- Permite que seja exercitado cada um desses comportamentos e verificado se ele executa como esperado, mesmo depois de ser alterado;
- Possibilita a “falsificação” de algumas dependências para que o objeto que está sendo testado tenha uma interação consistente com suas dependências externas;

Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

- **Ja**va **Co**de **Co**verage: biblioteca de cobertura de código java;
- Gera relatórios sobre métricas de testes do sistema.

Ferramentas Utilizadas

Junit 4

Assertj

Mockito

Jacoco

Sonarqube

- Ferramenta automática de revisão de código para detectar bugs, vulnerabilidades e code smells em seu código;
- Pode se integrar ao seu fluxo de trabalho existente para permitir a inspeção contínua de código em todas as ramificações do projeto e solicitações de pull;

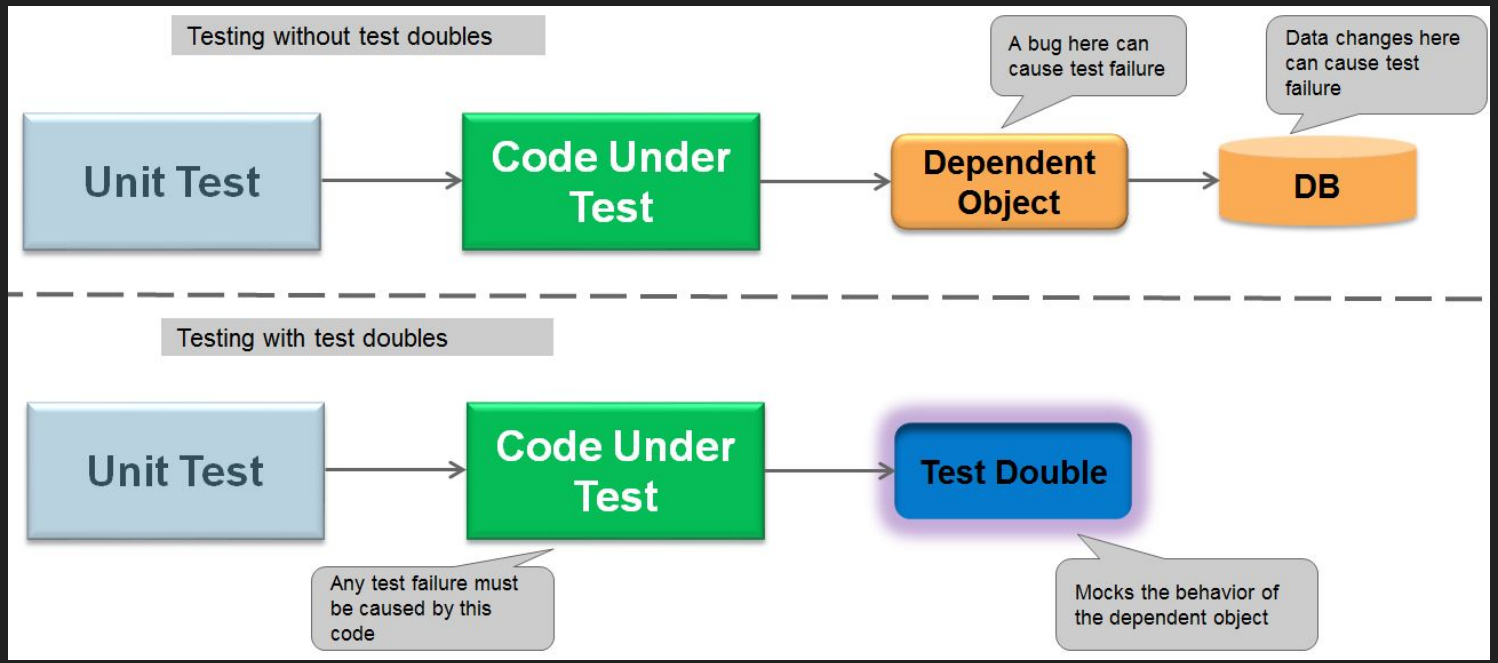
Metodologia Recomendada

TDD: Inversão do processo “tradicional” de desenvolvimento(os desenvolvedores, normalmente, primeiro implementam para depois testar).

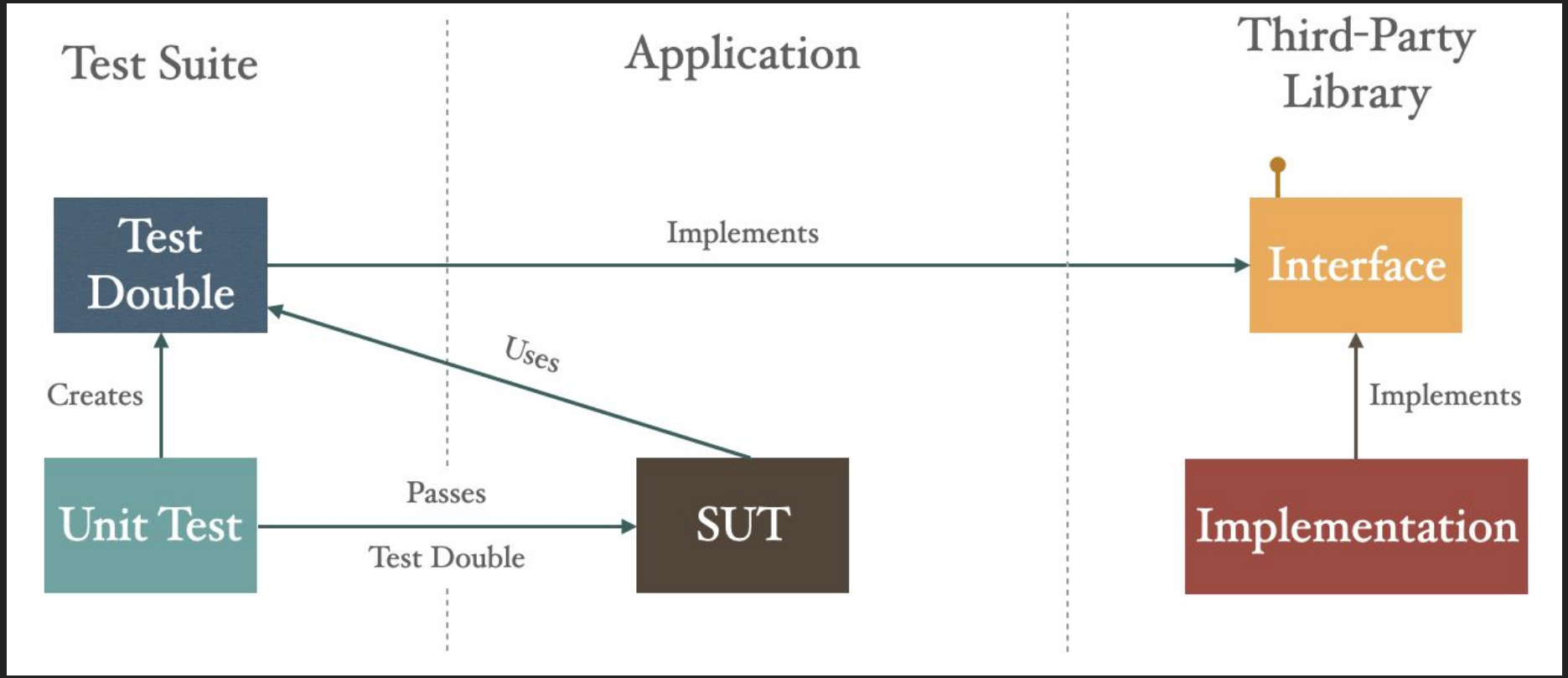
TDD: Desenvolvimento orientado a testes

Dublês de Teste

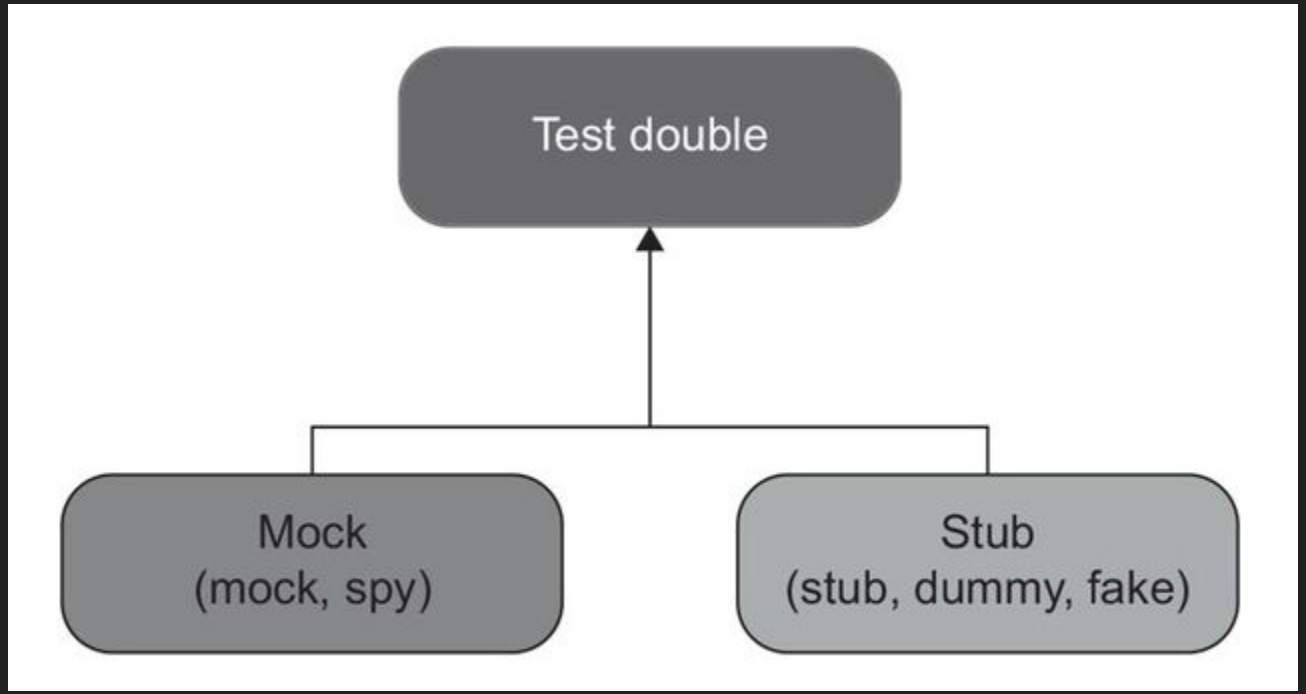
Simulam estados e comportamentos de objetos.



Dublês de Teste



Dublês de Teste



Dublês de Teste - Mocks

- Classes que simulam o comportamento de outras classes que interagem com dependências externas;

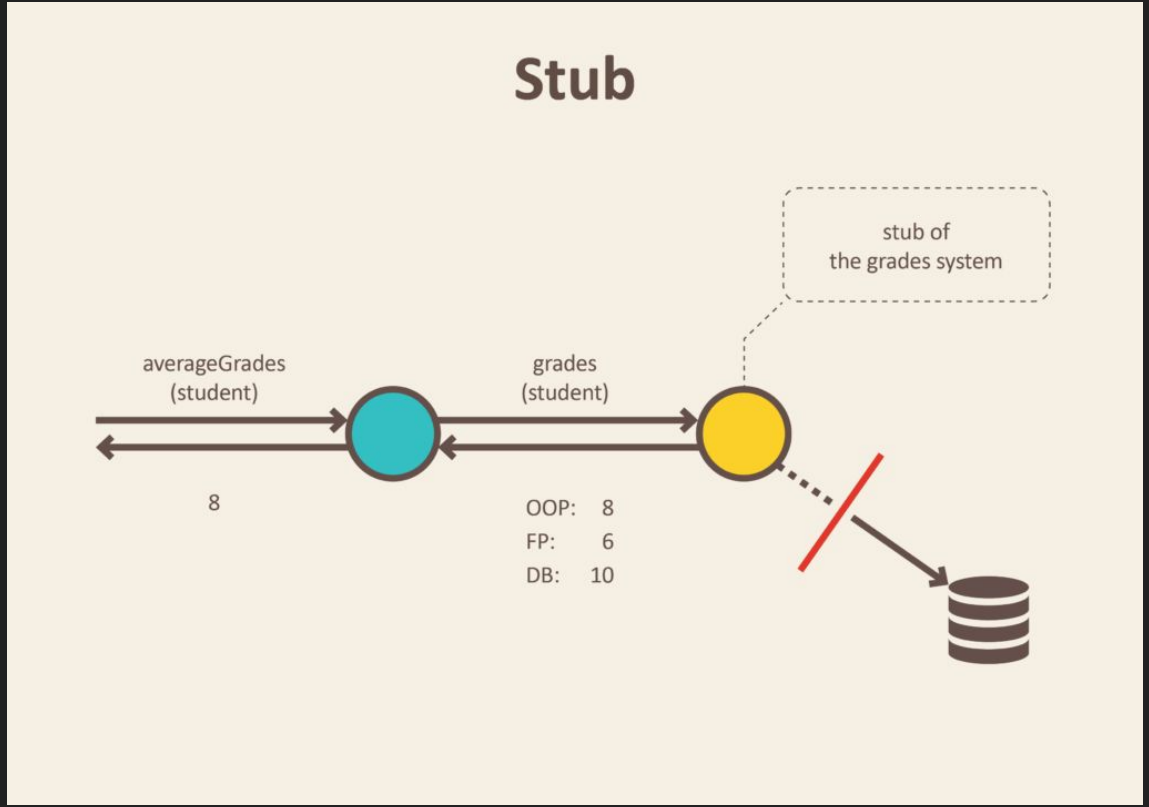
- Verificando se um ou mais métodos foram ou não chamados, a ordem de chamadas destes métodos, se esses métodos foram chamados com os argumentos certos, e quantas vezes foram chamados.

Dublês de Teste - Stubs

- Respostas prontas para as chamadas feitas durante o teste;

- Fornecem respostas prontas para nossas chamadas, eles ainda não têm nenhuma lógica, mas não irão gerar um erro, ao invés disso, eles retornam um valor pré-definido.

Dublês de Teste - Stubs



Dublês de Teste - Dummy

Objetos criados apenas para preencher parâmetros ou ‘compor o elenco’. Não são utilizados no fluxo principal de testes.

```java
public interface Logger {
    void append(String text);
}

@Service
@RequiredArgsConstructor
public class FilialService {
  private final Logger logger;

  public boolean createFilial(Filial filial)
  {
    logger.append("Creating filial for " );
    throw new UnsupportedOperationException();
  }
}
```

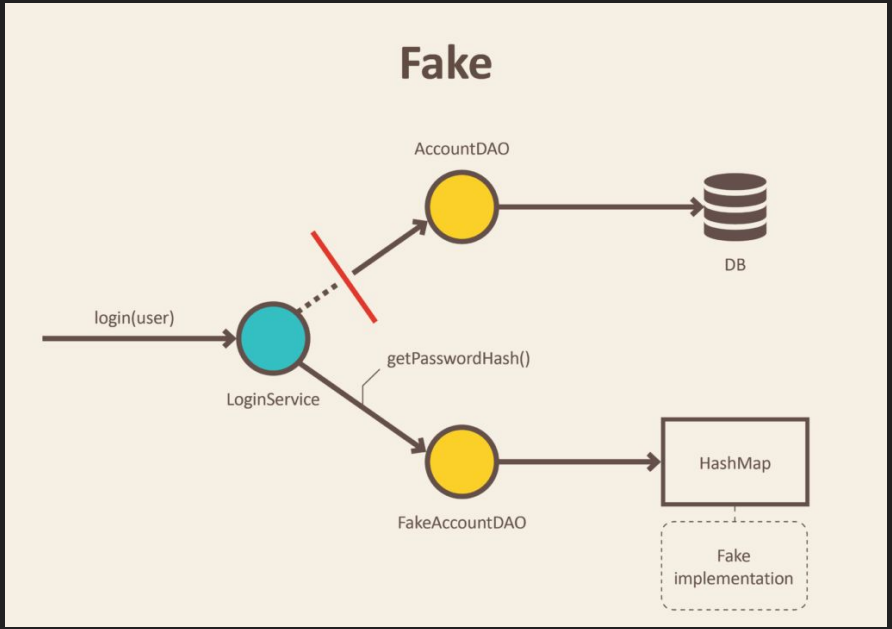
```java
public class LoggerDummy implements Logger {

    @Override
    public void append(String text) {}
}


LoggerDummy loggerDummy = new LoggerDummy();
FilialService = new FilialService(loggerDummy);
```

Dublês de Teste - Fakes

- Objetos que possuem implementação, porém com o objetivo de diminuir a complexidade e/ou tempo de execução de alguns processos



Dublês de Teste - Spies

- “Alguém que está infiltrado em seu SUT e está registrando todos os seus movimentos”;
- É uma denominação dada a um objeto que grava suas interações com outros objetos;
- Você usa spies quando não tem certeza sobre o que seu SUT vai chamar de seu colaborador, então você grava tudo e afirma se o spy chamou os dados desejados.

O que testar?

- Services: Afinal, representam a regra de negócio do sistema;
- Controllers: representam os pontos de entrada da aplicação que controlam como os services são chamados;
- Utils: Utilitários são importantes de serem testados!;
- Outros pacotes extras criados pelo programador.

O que está isento a testes?

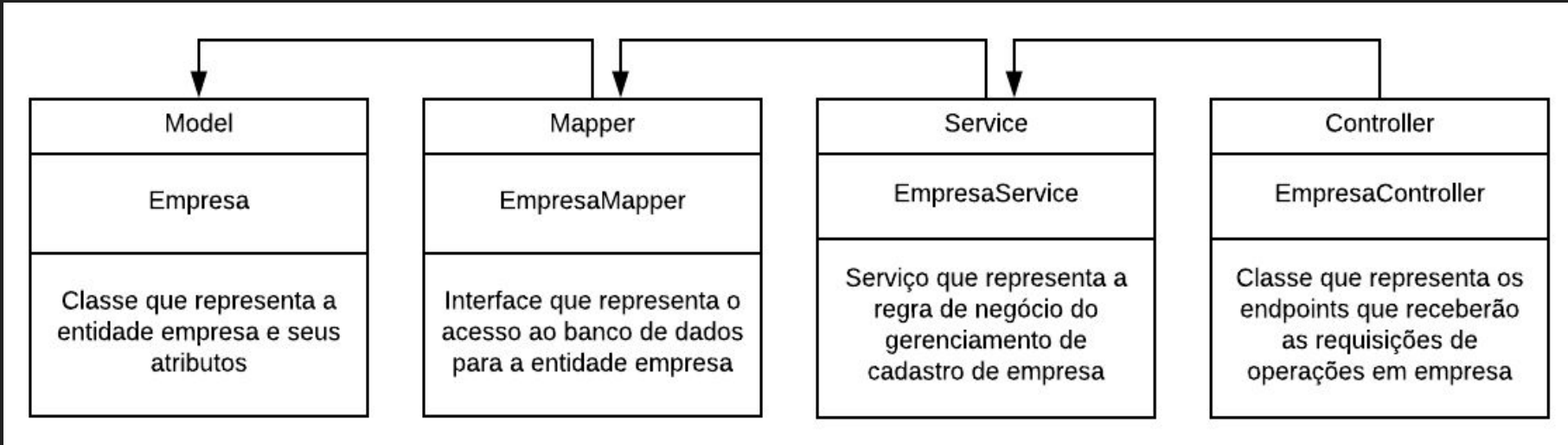
- Config: Afinal, são classes de configurações, que apenas devem ser consumidas;
- Mapper: Classes de acesso a dados não necessitam serem testadas;
- Model: Classe de entidades e modelos somente são preenchidos e utilizados, portanto não precisa testar!

Como Testar?

- Asserções;
- Verificação da quantidade de chamadas de determinado(s) método(s).

Fluxo Recomendado

Cenário: Gerenciamento de cadastro de Empresas



Escolhendo a unidade

Recomenda-se que se comece pelos services. Isso porque eles englobam a regra de negócio e, mesmo que nao haja banco de dados ou endpoints expostos, eles podem ser implementados isoladamente e testados porque são a regra principal do sistema!

Então como exemplo, escolhamos o EmpresaService para testá-lo.

Se iremos testar o EmpresaService, como ele depende do EmpresaMapper, este utilizando a entidade Empresa, necessitamos simular estados e comportamentos para as suas dependências e criar instâncias reais para a classe testada;

Situações

Situação 1:

Cenário: Objeto Empresa Alfa(cod 1)

Ação: cadastrar

Validação: Cadastrado Com sucesso

Situações

Situação 2:

Cenário: Objeto Empresa Beta(cod 2)

Ação: cadastrar

Validação: Cadastrado Com sucesso

Situações

Situação 2:

Cenário: Objeto Empresa Gama(cod 1)

Ação: cadastrar

Validação: Erro ao cadastrar por duplicidade de código

“A facilidade de se desenvolver um teste é proporcional a facilidade de lê-los”