# Testes de Software - Prof. Eiji Adachi

## Atividade Prática - Automatização de Testes e *Test-Driven Development*

### Criando um Projeto Maven

Inicialmente, vamos aprender o que é o **Maven** e como criar um projeto usando um archetype básico. O **Maven** é uma ferramenta de automação de build e gerenciamento de dependências amplamente utilizada em projetos Java. Ele facilita o processo de compilação, execução de testes e gerenciamento de bibliotecas externas de forma eficiente. Um **archetype**, por sua vez, é um modelo de projeto pré-configurado que define a estrutura básica de um projeto Maven, permitindo que você inicie rapidamente. Vamos usar o comando **mvn archetype:generate** com o archetype **maven-archetype-quickstart**, que cria um projeto Java simples e já preparado para você começar a codificar.

**Passo 0: Verificar se o Maven está instalado**

Primeiro, é necessário verificar se o Maven está instalado corretamente no seu sistema.

1. **Verificar a instalação do Maven**: Abra o terminal ou prompt de comando e execute o seguinte comando:

   ```
   mvn -v
   ```

   Se o Maven estiver instalado corretamente, você verá uma saída parecida com esta, mostrando a versão do Maven e detalhes sobre o ambiente Java:

   ```
   Apache Maven 3.8.4 (cecedd343002696d0abb50b32b541b8a6ba288b1)
   Maven home: /usr/local/apache-maven/apache-maven-3.8.4
   Java version: 11.0.10, vendor: Oracle Corporation, runtime:
   /usr/local/java/jdk-11.0.10
   Default locale: en_US, platform encoding: UTF-8
   OS name: "linux", version: "5.10.16", arch: "amd64", family: "unix"
   ```

2. **Se o Maven não estiver instalado**: Caso você receba uma mensagem de erro indicando que o Maven não foi encontrado, será necessário instalá-lo. Você pode seguir as instruções de instalação no site oficial do Maven:

   - Acesse o site: https://maven.apache.org/install.html

   Nesse endereço, você encontrará instruções detalhadas para baixar e instalar o Maven no Windows, macOS ou Linux.

Após verificar que o Maven está instalado corretamente, você pode prosseguir com a criação do projeto.

---

#### Passo 1: Executar o comando Maven

Abra o terminal ou prompt de comando no seu sistema e execute o seguinte comando:

```
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart -
DarchetypeVersion=1.5 -DinteractiveMode=true
```

- **mvn archetype:generate**: Este comando inicia o processo de criação do projeto a partir de um **archetype**, que é basicamente um template para projetos.
- **-DarchetypeArtifactId=maven-archetype-quickstart**: Especifica o archetype que será usado. O `maven-archetype-quickstart` é um dos archetypes mais simples e comuns para iniciar um projeto Java.
- **-DarchetypeVersion=1.5**: Especifica a versão do archetype que queremos usar.
- **-DinteractiveMode=true**: Define que o Maven irá solicitar os detalhes do projeto de forma interativa.

##### Passo 2: Preencher as informações do projeto

Após executar o comando, o Maven solicitará que você insira algumas informações sobre o projeto:

1. **groupId**: O identificador do grupo ou pacote principal do seu projeto. Isso geralmente segue o padrão de domínio reverso (ex: `com.example`). O `groupId` identifica de forma única a organização ou grupo que está desenvolvendo o projeto.

   ```
   Define value for property 'groupId': br.ufrn.imd
   ```

2. **artifactId**: O nome do artefato, ou seja, o nome do seu projeto. Este será o nome da pasta principal do projeto e o nome do artefato quando for empacotado.

   ```
   Define value for property 'artifactId': meu-projeto
   ```

3. **version**: A versão do projeto. Se você está começando, pode aceitar a versão padrão `1.0-SNAPSHOT`, que indica que o projeto está em desenvolvimento.

   ```
   Define value for property 'version' 1.0-SNAPSHOT: (pressione Enter
   para aceitar o padrão)
   ```

4. **package**: O pacote base para suas classes Java. Normalmente, o valor de `package` será o mesmo que o `groupId`. Ele define o nome do pacote onde o código Java será organizado.

   ```
   Define value for property 'package' br.ufrn.imd: (pressione Enter para
   aceitar o padrão)
   ```

Depois de preencher essas informações, o Maven criará automaticamente a estrutura do projeto.

#### Passo 3: Verificar a estrutura gerada

Depois que o processo terminar, você verá uma mensagem confirmando que o projeto foi criado com sucesso. A estrutura do projeto gerado será algo semelhante a esta:

```
meu-projeto/
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── br
    │           └── ufrn
    │               └── imd
    │                   └── App.java
    ├── test
    │   └── java
    │       └── br
    │           └── ufrn
    │               └── imd
    │                   └── App.java
```

- **pom.xml**: O arquivo de configuração do Maven. Ele define as dependências, plugins e informações do projeto.
- **src/main/java**: Diretório onde ficará o código-fonte do projeto.
- **src/test/java**: Diretório onde os testes ficam localizados.

#### Passo 4: Verificar se dependências do JUnit 5 foram acrescentadas

Verifique se as seguintes dependências estão no arquivo `pom.xml`:

```
<dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-params</artifactId>
      <scope>test</scope>
    </dependency>
```

#### Passo 5: Compilar e rodar o projeto

Agora que o projeto foi criado, você pode compilar o código com o seguinte comando:

```
mvn compile
```

Para rodar o projeto e verificar se tudo está funcionando, você pode executar a classe principal (gerada automaticamente) com o comando abaixo, usando o plugin `exec`:

```
mvn exec:java -Dexec.mainClass="br.ufrn.imd.App"
```

#### Passo 6: Rodar os testes

Para rodar os testes unitários (já configurados com uma classe de teste de exemplo), use o comando:

```
mvn test
```

Este comando executará todos os testes localizados no diretório `src/test/java` e exibirá os resultados.

### Passo 7: Importar o Projeto para um IDE (Eclipse, IntelliJ ou VSCode)

Após criar e configurar seu projeto Maven, você pode querer importar o projeto para um **IDE** (Integrated Development Environment) para facilitar o desenvolvimento. A seguir, veremos como importar o projeto para os três IDEs populares: **Eclipse**, **IntelliJ IDEA** e **VSCode**.

#### 7.1. Importar o Projeto para o Eclipse

1. **Abra o Eclipse** e vá até o menu **File > Import**.
2. No assistente de importação, selecione **Existing Maven Projects** e clique em **Next**.
3. Clique em **Browse** e navegue até o diretório raiz do seu projeto Maven (onde o arquivo `pom.xml` está localizado).
4. O Eclipse detectará automaticamente o projeto Maven e suas dependências. Selecione o projeto e clique em **Finish**.

   O Eclipse agora deve configurar e carregar o projeto Maven.

#### 7.2. Importar o Projeto para o IntelliJ IDEA

1. **Abra o IntelliJ IDEA** e vá até o menu **File > New > Project from Existing Sources**.
2. Navegue até a pasta do projeto que você criou e selecione o arquivo `pom.xml`.
3. O IntelliJ reconhecerá o projeto como um projeto Maven e exibirá uma janela para confirmar as configurações. Verifique se tudo está correto e clique em **OK**.
4. O IntelliJ IDEA irá baixar automaticamente as dependências e configurar o projeto para que você possa começar a trabalhar.

#### 7.3. Importar o Projeto para o VSCode

1. **Abra o VSCode** e instale a extensão **Maven for Java** (caso ainda não tenha feito isso). Você pode encontrar a extensão no marketplace do VSCode.
2. No VSCode, vá até **View > Command Palette** ou pressione `Ctrl + Shift + P`.
3. Na barra de comandos, digite **Maven: Add Maven Projects to Workspace**.
4. Navegue até a pasta do projeto e selecione o arquivo `pom.xml`.
5. O VSCode adicionará o projeto ao workspace, e a extensão do Maven baixará automaticamente as dependências necessárias.

---

## Enunciado de Implementação: Consolidação Parcial de Notas e Cálculo de Status de Aprovação

Você deverá implementar a funcionalidade de **consolidação parcial** das notas de uma turma de graduação, calculando o **status de aprovação** dos alunos com base no desempenho acadêmico e na frequência mínima exigida, de acordo com os seguintes critérios estabelecidos no regulamento:

---

## Regras de Aprovação:

**1. Critérios de Aprovação Direta:**

- O aluno é considerado **aprovado** se:
    - **Média parcial** nas unidades avaliativas for **igual ou superior a 6,0**.
    - **Nenhuma nota de unidade avaliativa** for **inferior a 4,0**.
    - **Frequência mínima** de 75% da carga horária for atingida.

  **Média final**: A média final do aluno aprovado será a **média parcial**, e o aluno estará dispensado da **avaliação de reposição**.

**2. Critérios para Avaliação de Reposição:**

- O aluno terá direito a realizar uma **avaliação de reposição** se:
    - Cumprir o critério de **frequência mínima de 75%**.
    - Sua **média parcial** for **igual ou superior a 3,0**.

  A **nota da avaliação de reposição** substituirá a menor nota entre as unidades avaliativas do aluno.

**3. Reprovação:**

- O aluno será **reprovado** se:
    - Sua **média parcial** for **inferior a 3,0**.
    - Não atingir a **frequência mínima de 75%**.
    - Não atender aos critérios para realizar a avaliação de reposição.

  **Média final**: O aluno reprovado terá sua **média final igual à média parcial**.

---

## Implementação

Você deverá implementar a funcionalidade de **consolidação parcial** das notas de uma turma de graduação, calculando o **status de aprovação** dos alunos com base no desempenho acadêmico e na frequência mínima exigida, utilizando as classes fornecidas. A seguir, você pode utilizar os arquivos disponibilizados para ajudar na implementação:

**Arquivos Disponibilizados:**

1. **Discente.java**: Representa os dados do aluno matriculado.
2. **Disciplina.java**: Representa as disciplinas e suas informações.
3. **Docente.java**: Representa o professor responsável pela disciplina/turma.
4. **Matricula.java**: Representa a matrícula de um aluno em uma disciplina, contendo as notas e frequência.
5. **StatusAprovacao.java**: Enumera os diferentes status de aprovação (REPF, REP, REC, APR, REPMF).
6. **Turma.java**: Representa uma turma com seus respectivos alunos matriculados.

Esses arquivos fornecem a base para a implementação da funcionalidade de consolidação das notas, e você deverá utilizá-los para implementar os métodos que calculam a média parcial, verificam o direito à reposição e definem o status final dos alunos.

**1. Classe `Aluno (Discente.java)`:**

Esta classe já contém os dados do aluno (discente), como nome, CPF, e outras informações pessoais. Utilize essa classe para referenciar os alunos matriculados em uma determinada turma.

**2. Classe `Matricula.java`:**

A classe de matrícula contém as notas e a frequência do aluno, e será fundamental para calcular a média parcial e verificar se o aluno atingiu os critérios de aprovação.

- **Notas**: Uma lista de notas referentes às unidades avaliativas.
- **Frequência**: A porcentagem de frequência do aluno (mínimo de 75%).

**3. Enum `StatusAprovacao (StatusAprovacao.java)`:**

Esta enumeração define os diferentes possíveis status de aprovação de um aluno:

- **REPF**: Reprovado por Faltas
- **REP**: Reprovado por Média
- **REC**: Em Recuperação (terá direito à avaliação de reposição)
- **APR**: Aprovado por Média
- **REPMF**: Reprovado por Média e Faltas

**4. Classe `Turma (Turma.java)`:**

A classe `Turma` representa uma turma de uma determinada disciplina e contém as informações necessárias para processar as matrículas dos alunos e calcular os status de aprovação. É nela que a consolidação dos resultados dos alunos será feita.

## Tarefas a Serem Implementadas:

**Consolidação parcial**:

- Implemente o método `consolidarParcialmente()` dentro da classe **Matricula**. Este método deve calcular a média das notas do aluno, verificar o critério de assiduidade e atualizar o atributo `status` da classe, de acordo com as regras estabelecidas acima.

**Teste da Consolidação parcial**:

- Implemente testes automatizados utilizando o JUnit 5. Seus testes devem utilizar testes parametrizados usando o formato CSV como entrada. A estrutura básica do seu teste deve ter formato similar ao teste abaixo:

```
// Arrange
Matricula matricula = new Matricula(discente, turma);
matricula.cadastrarFrequencia(frequencia);
matricula.cadastrarNota1(n1);
matricula.cadastrarNota2(n2);
matricula.cadastrarNota3(n3);

// Act
matricula.consolidarParcialmente();

// Assert
StatusAprovacao statusRetornado = matricula.getStatus();

Assertions.assertEquals(statusEsperado, statusRetornado);
```

- Implemente testes de exceções. Os métodos que registram notas e frequência devem ser alterados para lançar exceção `IllegalArgumentException` caso o valor recebido como parâmetro esteja fora do intervalo válido. As notas estão compreendidas entre 0 e 10 e a frequência entre 0 e 100. Em seguida, você deve implementar testes executáveis para verificar se os métodos de fato lança esta exceção.

**Utilizar *Test-Driven Development***:

- Implemente o método `calcularMediaParcial()` empregando o método TDD, seguindo o processo de criar um test cobrindo um trecho da especificação, implementar a funcionalidade correspondente, refatorar e seguir para a próxima parte da especificação. Lembre-se de rodar os testes continuamente para garantir que sua implementação está correta. Lembre-se também de realizar em passos pequenos.

## Entregáveis:

- O aluno deverá **implementar a funcionalidade** de consolidação parcial das notas e cálculo de status de aprovação.
- Deverá também implementar **testes automatizados** utilizando o JUnit 5, seguindo a abordagem de **Test-Driven Development (TDD)**.
- O arquivo `pom.xml` deverá ser alterado para que o campo `groupId` utilize o **nome completo** do aluno em formato **camelCase**, sem espaços e sem acentos, conforme o exemplo:

`joaoSilvaSantos`.

- O projeto deverá ser um arquivo .zip, que também deve ser nomeado com seu nome completo, conforme o formato definido no item anterior.

Itens obrigatórios:

- Implementação do método que faz a consolidação parcial.
- Implementação de testes executáveis usando testes parametrizados.
- Implementação de testes de exceção.
- Ao menos um caso de teste para cada possível Status.
- Seguir boas práticas de nomenclatura de classes e métodos e de estruturação de diretórios separados para código da aplicação e código de testes.

Referências:

- https://www.vogella.com/tutorials/JUnit/article.html
- https://www.baeldung.com/parameterized-tests-junit-5