FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS
CURSO: BANCO DE DADOS
DISCIPLINA: LABORATÓRIO DE DESENVOLVIMENTO EM BANCO DE DADOS VI
PROFESSOR: FABRÍCIO GALENDE MARQUES DE CARVALHO
ATIVIDADE PRÁTICA 01 – DESENVOLVIMENTO ORIENTADO A TESTES E FERRAMENTAS DE CONSTRUÇÃO

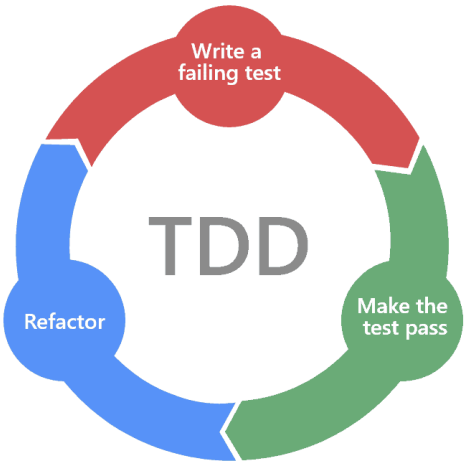
ATENÇÃO: O REPOSITÓRIO A SER CRIADO DEVERÁ SER MANTIDO ATIVO ATÉ PELO MENOS O INÍCIO DO PRIMEIRO SEMESTRE DE 2021 (PERÍODO DE REVISÃO DE NOTAS). OS HORÁRIOS DOS ENVIOS AOS REPOSITÓRIOS SERÃO CONSIDERADOS NO CUMPRIMENTO DOS PRAZOS DAS TAREFAS. NÃO "REINICIALIZE" SEU REPOSITÓRIO. APRENDA A USAR O GIT DE MODO ADEQUADO DE FORMA A NÃO PERDER O HISTÓRICO DE MODIFICAÇÕES E EVITAR QUE O PROFESSOR TENHA PROBLEMAS AO FAZER UMA ATUALIZAÇÃO OU AVALIAR SUA TAREFA (QUE PODERÁ SER ZERADA NO CASO DE PERDA DE HISTÓRICO).

OBJETIVO: Esse trabalho tem como objetivos ilustrar, na prática, o método de desenvolvimento orientado a testes (***TDD – Test Driven Development***) e, também, a utilização de ferramentas de build para obtenção de dependências, execução de sistemas e execução de casos de teste.

INTRODUÇÃO:

O desenvolvimento orientado a testes é um método de desenvolvimento de software que parte primeiramente da especificação de um conjunto de casos de teste para posterior codificação de uma funcionalidade e/ou sistema de software. Considera-se que, nesse caso, a especificação é dada pelo conjunto dos casos de teste.

A figura seguinte ilustra o ciclo de desenvolvimento TDD típico:



O ciclo começa na fase vermelha, onde um conjunto de testes executáveis são escritos, ou seja, deve ser possível fazer o build do sistema. Entretanto a funcionalidade não está disponível e, por isso, os testes escritos irão falhar. Um exemplo disso é simplesmente fazer com que uma asserção, que tem como um valor esperado X, receba forçadamente um valor Y. Dessa forma, os testes existem, são executáveis, mas falham!

A seguir, vai-se à fase verde. Nesse caso, uma funcionalidade **SIMPLIFICADA** é implementada de modo a fazer com que os casos de teste já especificados passem, ou seja, além de simplificada é **SUFICIENTE PARA PASSAR NOS TESTES**. Note que não se deve alterar o conjunto de especificações de casos de teste da fase vermelha para a fase verde, o que muda é que, ao invés

de forçar a falha com um valor Y (lembrando que o valor esperado do caso de teste vale X), utiliza-se uma versão simplificada do componente implementado (ex. um stub de método para um objeto que foi "mockado" e que só retorna corretamente certos valores ou que, por exemplo, não faz tratamento de exceções, etc.). Ao final dessa fase, os mesmos testes da fase vermelha devem ser executados e todos devem passar.

Na fase azul procede-se com a melhoria dos códigos do sistema (refatoração). Novamente sem alterar as especificações dos casos de teste. Exemplos de melhorias possíveis são: Tratamento de exceções que podem ser possíveis, mas que não estavam previstas na implementação original, otimização do código (redução do número de passos), melhoria de legibilidade, etc. Ao final dessa fase, os mesmos casos de teste da fase verde devem ser executados e devem passar.

Notar que uma decisão crítica no desenvolvimento orientado a testes consiste em se projetar as interfaces dos componentes que serão desenvolvidos, pois isso define o que será retornado e como os valores esperados e as entradas serão especificadas nos casos de teste executáveis.

ENUNCIADO:

1. Criar um repositório no GitHub, denominado **`lab_bd_vi`**. Nele, acrescentar um arquivo README.MD contendo seu nome, seu RA e um descritivo de cada uma das subpastas conforme as entregas da disciplina. Atentar também para a configuração do arquivo **`.gitignore`** para que arquivos de IDE, arquivos compilados, arquivos temporários, etc. não sejam inadequadamente acrescentados ao seu repositório.
2. No repositório criado, criar uma pasta chamada **`pratica_01`**. Colocar dentro dela todos os artefatos necessários à entrega dessa tarefa, incluindo um arquivo README.MD contendo o link para o vídeo explicativo da entrega, de no máximo 5 minutos (aproximadamente), onde você faz o build do sistema e executa uma rodada completa do TDD, mostrando e explicando seu código nos estados vermelho, verde e azul).
3. Para uma classe denominada **`Calculadora`**, que deverá operar sobre números de ponto flutuante ou números complexos, escolher uma operação e demonstrar a execução de uma rodada do ciclo TDD. Ao escolher a operação, observar se algum colega já não escolheu tal operação, se sim, escolher outra operação. Exemplos de operações elegíveis são: soma, subtração, multiplicação, raiz quadrada, raiz cúbica, raiz n-ésima, potência, logaritmo, fatorial, combinação, arranjo, etc. Notar que a escolha de uma operação determina completamente uma interface. Além disso, considera-se, nesse caso, que a operação para números de ponto flutuante é diferente de operação para números complexos (ex: método soma para complexo pode receber 2 argumentos, já para números complexos poderia receber 4, correspondentes às partes imaginárias e reais de cada número – dependendo de seu projeto). Logo, um aluno pode fazer uma soma para flutuante e outro pode fazer uma soma para números complexos.
4. Assegure-se de que seu repositório contém um único branch. E que os commits das fases vermelha, verde e azul estão claramente identificadas.
5. Grave um vídeo, com duração aproximada de 5 minutos, explicando sobre seu código nos estados vermelho, verde e azul e mostrando a execução dos casos de teste. Ilustre claramente a geração dos relatórios em HTML do sistema. Disponibilize este vídeo no YouTube e coloque o link a partir do REAME.MD interno à pasta **`pratica_01`**.

6. Utilize o Gradle 5 ou superior, linguagem de programação Java e o JUnit 5 para a execução dos casos de teste. Configure adequadamente o seu build script para usar somente as dependências apropriadas.
7. Disponibilize o link do seu repositório do GitHub para o professor. Certifique-se de que o professor, conseguirá:
    a. **Fazer a clonagem do seu repositório**
    b. **Executar o seu programa com o simples comando gradle run**
    c. **Executar os seus casos de teste com o simples comando gradle test**
    d. **Entender os comentários explicativos que devem estar presentes no seu código**

CRITÉRIOS DE PONTUAÇÃO: A grade seguinte ilustra como serão distribuídos os pontos da tarefa. Um critério adequado é aquele que atende a todos os itens no enunciado do trabalho e, também, a tabela com os critérios. Caso um único item citado nos critérios de avaliação ou no enunciado esteja incorreto, o aluno receberá 0 de nota no critério. Caso todos os itens estejam corretos, o aluno receberá 100% do total de pontos do critério.

Critério	**Percentual da Nota na Tarefa**
Vídeo explicativo claro, audível e acessível. Demonstração de domínio e entendimento apropriado do ciclo TDD. Código-fonte compatível com os requeridos no ciclo TDD em cada uma das etapas (vermelho, verde e azul).	50%
Utilização adequada do Git/GitHub. Repositório limpo, livre de arquivos não adequados (arquivos de cache do gradle, arquivos do IDE, arquivos binários, README.MD adequado e com), commits adequadamente comentados.	25%
Utilização apropriada da ferramenta de testes JUnit 5. Geração e visualização dos relatórios em todas as fases do ciclo TDD.	25%

OBS: Para demais tarefas que exijam entrega de código, apenas deverá ser acrescentada uma nova pasta à raiz do repositório e que conterá o código-fonte aplicável à entrega. O nome dessa pasta será especificado no enunciado da tarefa. O repositório não deve ser renomeado e nem os arquivos antigos excluídos. O uso apropriado do GitHub será considerado na avaliação. A não observação dessas instruções, penalizará o aluno em 50% da nota.