# Roteiro de exercícios sobre Projeto e implementação, disciplina Engenharia de Software e Sistemas

## Paulo Borba

## Centro de Informática

## Universidade Federal de Pernambuco

Adicione as respostas imediatamente após as perguntas, usando uma outra cor. Leia com calma e atenção as instruções, e reflita com cautela sobre as consequências de cada comando. As instruções abaixo assumem que toda a parte de instalação dos roteiros anteriores foram concluídas. É também assumido que todas as referência (manuais, tutoriais, etc.) mencionadas nos roteiros anteriores já foram consultadas, e que os alunos devem estar prontos para rapidamente consultá-las novamente durante o resto do curso, inclusive quando seguindo o roteiro abaixo.

## Aula 1

1. Leia o [material](http://domainlanguage.com/wp-content/uploads/2016/05/DDD_Reference_2015-03.pdf) (pelo menos as páginas 10 a 18) de referência sobre DDD e explique a diferença entre *entities* e *value objects*, e entre *factories* e *repositories*.
2. Considerando as necessidades dos cenários de cadastro e visualização de discrepâncias de auto-avaliação discutidos em roteiros anteriores, elabore o modelo DDD necessário para que o sistema possa contemplar a implementação das funcionalidades desse cenário. Foque apenas nesse aspecto, não em aspectos complementares (como turma, matrícula, etc.) que podem ter sido discutidos durante a aula. Cole a imagem do modelo logo abaixo.
3. Assuma que, no modelo DDD do teachingassistant, haja uma associação direta entre **Aluno** e **Avaliação**, no sentido de que um aluno contém um conjunto de avaliações (pares de meta, como "Escrever requisitos com qualidade", e conceito, como "MA" e "MPA"). Essa modelagem seria adequada em uma situação em que um aluno pode cursar várias disciplinas? Por que? Essa modelagem seria adequada em uma situação em que um aluno pode ter sido reprovado em uma disciplina? Por que?
4. Elabore o modelo DDD de um sistema de gerenciamento de tarefas como o de *issues* do GitHub, que considera tarefas com várias propriedades como descrição, responsáveis, tipo (*bug fix*, *feature*, etc.), e pontuação. Cada projeto agrupa tarefas em um conjunto de categorias (icebox, backlog, in progress, etc.), e tem também um grupo de *epics.* F

## Aula 2

1. Quais as vantagens de projetos que apresentam *low coupling* e *high cohesion*?
2. Do ponto de vista prático, qual a principal vantagem de seguir o princípio de *information hiding*?
3. Explique qual a relação do princípio *single responsibility* com *low coupling* e *high cohesion?*
4. Leia o artigo [*Feature Toggles*](https://www.martinfowler.com/articles/feature-toggles.html)até a seção "*Getting ready to release*". Para que serve a técnica proposta? Quais as vantagens e desvantagens da mesma em relação a possíveis alternativas?

## Aula 3

1. Leia as descrições dos padrões [*composite*](https://en.wikipedia.org/wiki/Composite_pattern)*,* [*strategy*](https://en.wikipedia.org/wiki/Strategy_pattern)*,* [*bridge*](https://en.wikipedia.org/wiki/Bridge_pattern).
   1. explique as diferenças e similaridades entre os padrões *composite* e *decorator*.
   2. em que situações seria interessante usar o *strategy* ao invés do *decorator*, e em que situações seria interessante usar uma combinação dos dois?
   3. como o padrão *bridge* poderia ser usado para tornar mais flexível a implementação da classe CadastroDeAlunos? Em que situações essa flexibilização seria útil?
   4. analise o segundo método **merge** na classe [**SemistructuredMerge**](https://github.com/guilhermejccavalcanti/jFSTMerge/blob/master/src/main/java/br/ufpe/cin/mergers/SemistructuredMerge.java), em especial o comando **for**. Quais dos seguintes padrões de projeto foram adotados nesse código:[*composite*](https://en.wikipedia.org/wiki/Composite_pattern)*,* [*strategy*](https://en.wikipedia.org/wiki/Strategy_pattern)*,* [*bridge*](https://en.wikipedia.org/wiki/Bridge_pattern)? Por que você acha que os desenvolvedores adotaram essa opção de projeto?
   5. em uma linguagem que tem suporte a funções de alta ordem, podemos implementar o *strategy* sem criar uma interface e classes que a implementam? Se sim, como? Se não, por que não é possível?
2. Altere o código do teachingassistant para que todos os testes discutidos no roteiro anterior sejam executados com sucesso ("O servidor só cadastra alunos" falha). Execute os testes e cole aqui o resultado da execução.

## Aula 4

1. **(Opcional)** Implemente a funcionalidade dos cenários de cadastro de auto-avaliação especificados por você no roteiro de requisitos.
   1. Cole logo abaixo o *link* do *commit* do seu repositório que contém essa implementação. Esse *commit* deve ter apenas as mudanças relativas a essa funcionalidade.
   2. Verifique se o seu projeto e implementação estão seguindo rigorosamente as recomendações para projeto e escrita de código com qualidade (mencionadas nos *slides* e no livro texto).
   3. Certifique-se de que os testes associados estejam passando, e que nenhum teste que passava antes esteja falhando após a realização da sua mudança. Execute os testes e cole logo abaixo a imagem do relatório da execução.
2. **(Opcional)** Implemente a funcionalidade dos cenários de visualização de discrepâncias de auto-avaliação especificados por você no roteiro de requisitos.
   1. Cole logo abaixo o *link* do *commit* do seu repositório que contém essa implementação. Esse *commit* deve ter apenas as mudanças relativas a essa funcionalidade.
   2. Verifique se o seu projeto e implementação estão seguindo rigorosamente as recomendações para projeto e escrita de código com qualidade (mencionadas nos *slides* e no livro texto).
   3. Certifique-se de que os testes associados estejam passando, e que nenhum teste que passava antes esteja falhando após a realização da sua mudança. Execute os testes e cole logo abaixo a imagem do relatório da execução.
3. **(Opcional)** Solicite, a um colega de turma, o *link* para a implementação de uma funcionalidade implementada por ele como resposta de um dos dois exercícios anteriores.
   1. Cole o *link* logo abaixo.
   2. Identifique, e liste logo abaixo, quais recomendações para projeto e escrita de código com qualidade (mencionadas nos *slides* e no livro texto) não foram seguidas na funcionalidade implementada pelo seu colega.
4. Defina, em conjunto com os outros membros da sua equipe, o modelo DDD do seu projeto:
   1. Apresente-o ao professor para revisão presencial, com todos os membros da equipe
   2. Cole abaixo a imagem do modelo após a revisão feita pelo professor
5. Crie, em conjunto com os outros membros da sua equipe (não de forma individual), uma versão inicial do seu projeto:
   1. Use como base a estrutura e configurações do projeto usado como exemplo na disciplina
   2. Adicione os esqueletos das implementações das classes presentes no DDD, incluindo a assinatura de métodos que façam parte da interface necessária para a realização independente de tarefas a serem realizadas por desenvolvedores diferentes da equipe
   3. Adicione arquivos HTML, e de outros tipos, que sejam necessários como interface entre as tarefas a serem realizadas de forma independente por desenvolvedores diferentes da equipe
   4. Crie um *pull request* (PR) com esse conteúdo e solicite, via chat, que o professor aceite esse PR antes de você começar a próxima questao
6. Implemente a funcionalidade associada aos cenários do seu projeto:
   1. Para cada cenário implementado, crie um *commit* (se necessário use *squash*) com a implementação da funcionalidade associada ao cenário
   2. Se necessário, crie um *commit* (se necessário use *squash*) com os ajustes necessários para que cada um dos testes de classe e serviço passem
   3. Crie pelo menos um *commit* (se necessário use *squash*) com o arredondamento da funcionalidade implementada (isto é, código que contemple funcionalidade não especificada pelos testes mas necessária para o bom funcionamento do sistema)
   4. Seguir rigorosamente as práticas sugeridas nos *slides* e livro, e o *checklist* associado
   5. Se necessário, acrescente *stubs* e *drivers* para simular dependências de funcionalidades sob responsabilidade de outros membros da sua equipe
   6. Se você não fez a sua parte do projeto, simplesmente avise ao professor; entregar ou tentar explicar algo que você não fez é fraude!
   7. Revisar o pull request enviado e verificar que nenhum arquivo ou mudança foi inadvertidamente incluído
      1. Só devem ser incluídos arquivos e mudanças relacionadas a sua funcionalidade, e que foram alterados durante a realização da sua atividade
      2. *Pull requests* **não** devem conter arquivos gerados ou de configuração particular de cada máquina ou IDE, como o conteúdo do diretório node\_modules, arquivos .js gerados a partir de .ts, etc. O mesmo vale para arquivos de log e similares. Não deve-se dar *commits* nesses arquivos. Use o .gitignore para evitar esse problema
      3. Evite pular linhas, adicionar espaços, etc. em partes do código que você não precisa modificar; cada mudança desse tipo precisa ser analisada por quem vai integrar a sua contribuição ao repositório principal
      4. Execute um *spell-checker* ou verificador gramatical para ter certeza que os cenários criados não contêm erros de inglês
      5. Tenha certeza de que todos os testes estejam passando na versão incluída no *pull request*, e que ele possa ser integrado sem conflitos
      6. A mensagem do *pull request* deve refletir com precisão o seu conteúdo.
7. Revisar o PR de um outro membro da sua equipe
   1. Por ordem alfabética, o primeiro membro da equipe revisa a entrega do segundo membro, o segundo membro revisa a entrega do terceiro, e assim por diante (se um dos membros não entregou a sua parte do projeto, revise a entrega do membro seguinte)
   2. Adicionar comentários no próprio github
      1. nas linhas dos arquivos que aparecem nos *commits* do *pull request*, não na área de discussão do *pull request*
      2. abrir a aba *files changed* do *pull request*, ir para o arquivo a ser revisado, passar o mouse sobre a linha que você gostaria de comentar, um símbolo de + irá aparecer, ao clicar nele você pode comentar o que há de errado com a linha
   3. Usar os slides das aulas como checklists da revisão (o revisor tem que verificar se todos os problemas e recomendações dos slides foram observados durante a realização do projeto)
8. Colar abaixo o *link* para o PR atualizado no repositório do professor com novos *commits* contendo
   1. Implementação dos quatro cenários. Cole abaixo *links* para o primeiro e último *commits* com contribuições para a implementação dos cenários.
   2. Cole abaixo *links* que apontam para comentários seus feitos em revisões do código de seus colegas