Tutorial do experimento:

O formulário é dividido em três seções. Primeiramente, preencha seu nome completo, e assinale a opção que expõe e concordância de utilização dos dados do formulário de forma confidencial. Posteriormente, assinale as opções na seção de perfil do participante para prosseguir na análise das ALPS otimizadas.

Na última seção, está descrito no título que será analisado. São quatro formulários diferentes, com intuito de obter uma análise subjetiva das arquiteturas otimizadas por meio da ferramenta OPLA-Tool. Portanto, verifique qual métrica será analisada, e identifique os arquivos nos diretórios disponibilizados. Os mesmos estão em ordem, e identificados de acordo com o campo Identificação *de ALPS* descrito no formulário. Após a análise subjetiva, atribua uma nota de 1-5 no campo *Nota* presente no formulário, e descreva brevemente uma justificativa para a nota. Abaixo são apresentados exemplos de justificativa.

Exemplos de Justificativa:

Análise de modularização de características:

Nota: 1	Características A e B pouco modularizadas
Nota: 5	Características totalmente modularizadas

Análise de acoplamento:

Nota: 1	Classes A, B e C são totalmente dependentes uma das outras
Nota: 5	Classes totalmente independentes

Análise de Trade-Off:

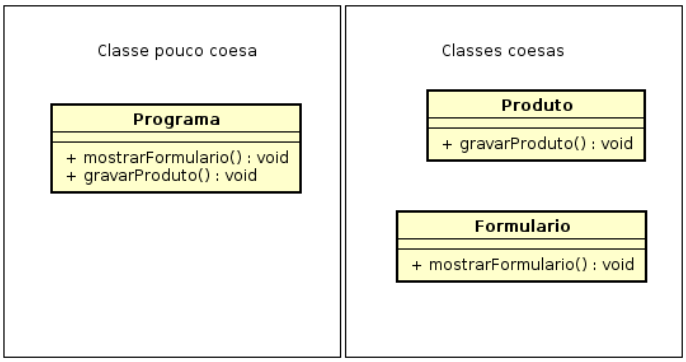
Nota: 1	Classes A, B e C poucos coesas e com alto acoplamento
Nota: 5	Classes com alta coesão e baixo acoplamento

Análise de Coesão:

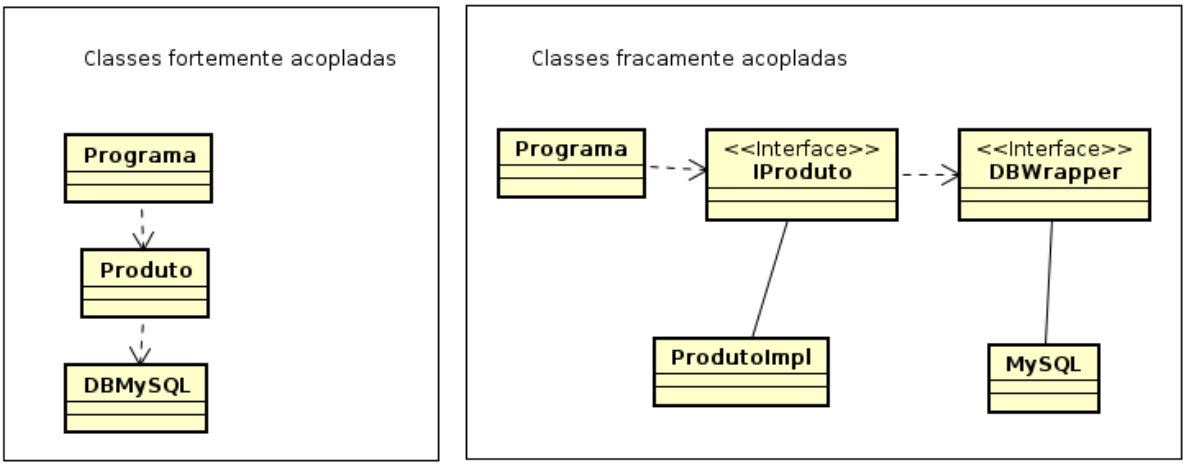
Nota: 1	Classes A, B e C com métodos repetidos
Nota: 5	Classes bem separadas

Se houver dúvidas sobre o que são as métricas que serão analisadas de acordo com cada formulário, utilize-se das informações abaixo.

Coesão: Introduzida por Robert C. Martin (2000), diz que uma classe deve ter apenas *uma* responsabilidade e realizá-la de maneira satisfatória. Ex:



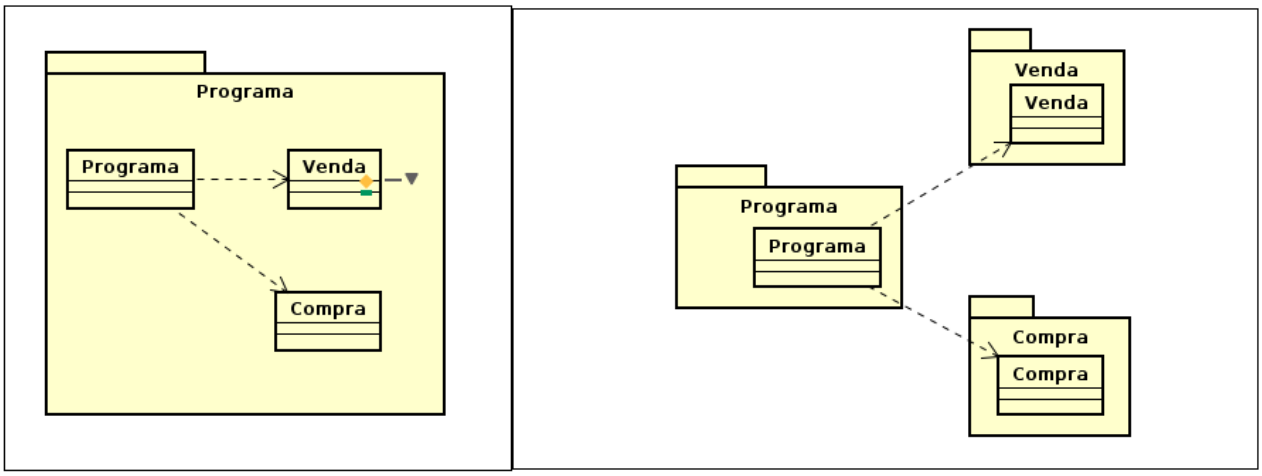
Observe que a classe Programa possui responsabilidades demais, o correto seria ter uma classe Formulário como o método de mostrar formulário e outra classe Produto para gravar o produto.



Acoplamento: Significa o quanto uma classe depende de outra para funcionar. Quanto maior tal dependência, dizemos que estas classes estão fortemente acopladas.

É possível observar que no primeiro quadro, as classes estão fortemente acopladas, pois se for necessário implementar uma conexão para Oracle, alterando a classe DBMySQL, poderia afetar a classe Produto, e implicaria até mesmo na coesão das classes. Por isto, o correto é ter uma interface Iproduto e uma DBWrapper, e utilizadas tais interfaces. Assim, teriamos uma implementação de Produto e uma implementação do MySQL, e quando fosse necessário implementar uma conexão Oracle, seria mais intuitivo, devido ao fato de só trocar a que está sendo utilizada. Além disso, poderia existir várias implementações de Produto e Banco de dados.

Modularização de características (Feature Driven): A ideia de dividir os programas em módulos, surgiu na década de 1960, devido as países desenvolvidos passarem por uma "crise de software". Com isto, ao passar dos anos surgiu o termo de Modularização de características, que é observar propriedades presentes na arquitetura, e agrupá-las em módulos. Para ficar mais claro, observe as classes abaixo.



Observe que no primeiro quadro, temos características misturadas, ou seja, um módulo com classes relacionadas ao Programa, Compra e Venda. No exemplo acima, são apenas 3 classes, mas imagine um sistema real de compra e venda, no qual existem centenas de classes relacionadas. Por isto, o correto é separar as características por pacotes, evitando classes misturadas.