**Verificação e Validação de Software I**

**Prof. Bernardo Copstein**

# Trabalho Final da Disciplina

## Enunciado geral

A figura apresenta o diagrama de classes parcial de um sistema de vendas pela internet. A classe *ServicoDeVendas* oferece alguns serviços usados pelos casos de uso do sistema. Eles são implementados pelos seguintes métodos:

- *void valida(List<ItemVenda> itens)*: verifica de um conjunto de itens de venda é válido no contexto do sistema. Gera uma *SistVendasException* caso alguma regra de validação seja violada. O conjunto de regras de validação utilizado varia conforme o horário do dia. Por isso o método se utiliza de uma classe “Factory” para obter o conjunto correto de regras.

*- Integer calculaSubtotal(List<ItemVenda> itens)*: retorna o somatório do valor de venda dos itens informados.

*- Integer calculaImpostos(List<ItemVenda> itens)*: retorna o total de imposto a pagar sobre a venda dos itens informados. Explora o padrão “strategy” de maneira a utilizar a estratégia de cálculo de impostos corrente.

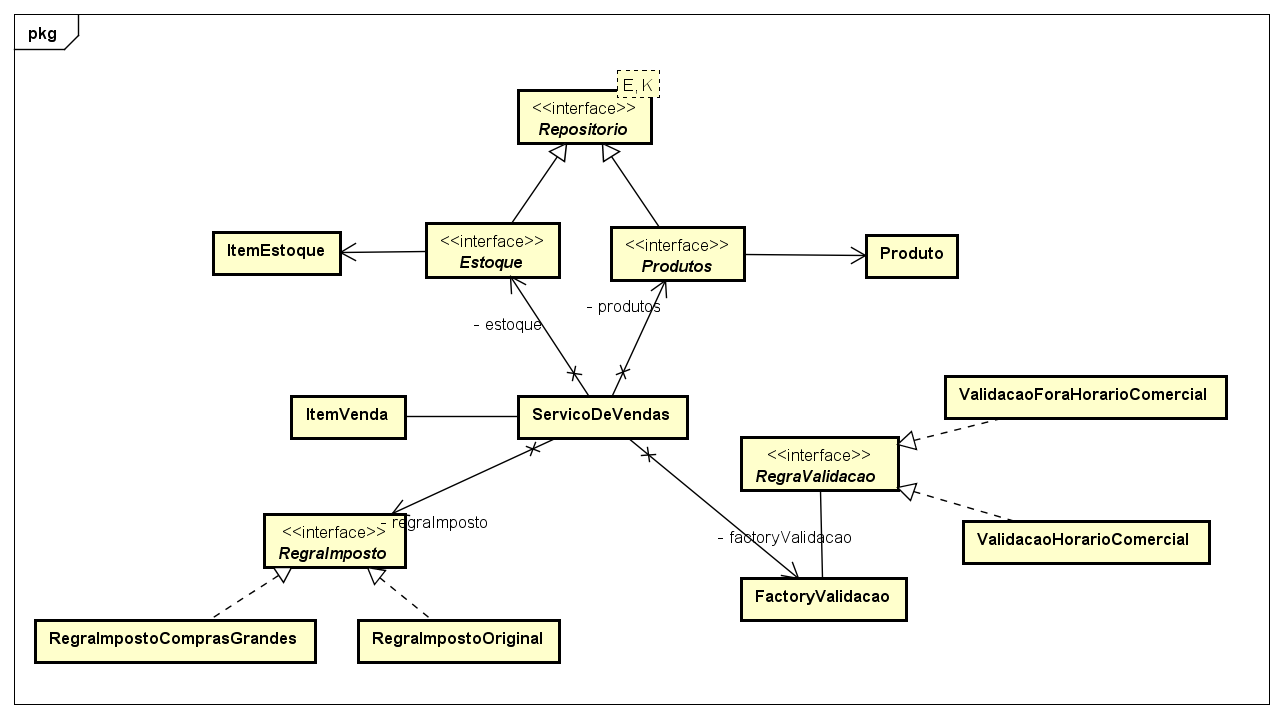
*- Integer calculaPrecoFinal(List<ItemVenda> itens)*: calcula o preço final de uma venda como sendo o somatório do subtotal com os impostos.

*- Integer[] todosValores(List<ItemVenda> itens)*: retorna um arranjo de inteiros com os seguintes valores: posição zero, subtotal; posição 1, impostos e posição 2, preço final.

As regras de validação são definidas a partir de um “factory” implementado pela classe *FactoryValidacao*. O construtor dessa classe recebe uma instância de *LocalTime* com o horário corrente como parâmetro. O método *RegraValidacao getRegraValidacao(Produtos produtos, Estoque estoque, List<ItemVenda> itens) throws SistVendasException* retorna uma instância de *RegraValidacao* de acordo com o horário corrente. Entre 8:00 e 18:00 inclusive, o método retorna uma instância de *ValidacaoHorarioComercial*. Fora desse horário retorna uma instância de *ValidaçãoForaHorarioComercial*. A interface *RegraValidacao* possui apenas um método: *void valida()*. No caso da validação em horário comercial o método gera uma exceção caso o conjunto de itens contenha algum código de produto não cadastrado no estoque ou não exista quantidade suficiente no estoque para atender o item de venda. No caso da validação fora do horário comercial, além destas duas verificações, verifica-se também se o valor de algum item de venda (quantidade \* preço de venda) não ultrapassa R$ 5000,00.

O cálculo dos impostos utiliza uma das duas implementações de *RegraImposto*: *RegraImpostoOriginal* e *RegraImpostoComprasGrandes*. A implementação que será usada é definida no código dependendo do cliente a que se destina o sistema. A interface RegraImposto define o método: *double calcular(List<ItemVenda> itens)*. A regra definida pela classe *RegraImpostoOriginal* calcula 10% sobre o somatório dos itens de venda. Já a regra definida pela classe *RegraImpostoComprasGrandes* define 10% para os 3 itens mais caros e 5% para os seguintes.

Este conjunto de classe depende ainda dos repositórios definidos pelas interfaces *Estoque* e *Produtos*. Os nomes dos métodos são auto explicativos. Solicite para o professor o esqueleto de todas as classes descritas aqui.



## Definição do trabalho

A partir desta especificação e do esqueleto das classes descritas aqui fornecido pelo professor, gere um conjunto de casos de teste para testar as mesmas conforme solicitado a seguir:

1. Gere um conjunto de casos de teste unitário. Gere casos de teste para todas as classes menos para as classes de repositório que não serão fornecidas. Identifique as estratégias utilizadas.
2. A partir dos casos de teste definidos no passo 1 gere os respectivos drivers de teste. Explore testes parametrizados sempre que possível. Explore “dubles” de maneira a testar apenas as classes alvo.
3. Solicite para o professor as classes implementadas (existem defeitos inseridos propositadamente)
4. Defina a ordem de integração das classes e, por consequência, os conjuntos de casos de teste de integração. Identifique as estratégias de geração de casos de teste utilizadas.
5. Implemente os drivers de teste de integração a partir dos casos de teste definidos no passo 3.
6. Execute os casos de teste; relate os defeitos encontrados; corrija os mesmos e repita o processo até não identificar mais defeitos.
7. Utilize o sonar cloud para avaliar a qualidade do código e a cobertura dos casos de teste. Resolva os problemas de qualidade e acrescente casos de teste para resolver os problemas encontrados.
8. Finalize o relatório com suas impressões sobre as técnicas e ferramentas utilizadas.