CES-28 Prova 1 - 2015

Sem consulta - individual - com computador - 3h

Obs.:
1. Qualquer dúvida de codificação Java só pode ser sanada com textos oficiais da Oracle ou JUnit.
2. Questões com itens diversos, favor identificar claramente pela letra que representa o item, para que eu saiba precisamente a que item corresponde a resposta dada!
3. Só pode implementar usando Eclipse ou outro ambiente Java as questões ou itens indicados com o rótulo **[IMPLEMENTAÇÃO]**!
4. Questoes obrigatorias devem ser resolvidas.
5- Codigo e arquivo texto (docx) com respostas podem ser submetidos como anexos em atividade no tidia. Use os numeros das questoes para identifica-las

1- Sistema de notas fiscais (Questão em 3 partes)

Obedecer as restricoes legais é critico. Uma auditoria examinará periodicamente todas as NF do sistema e falhas resultam em multa.

Conceitos (classes):

1. Nota Fiscal contem itens de venda.
2. NF contem informação do cliente.
3. Item de venda: Deve estar associado a um produto ou serviço pre-existente[1].
4. Cliente: dados de um cliente. contem CPF valido.
5. Cadastro: Acessa BD de clientes, possui metodos para encontrar clientes e para cadastrar novo cliente. Nao deve cadastrar cliente se o CPF for invalido.
6. VerificadorCPF: algoritmo que verifica se um CPF eh valido.

Restrições e Requisitos

1. Restrição legal: Nota fiscal não pode ter zero itens de venda. Deve ter 1 ou mais.

[1] na verdade deveriam haver cadastros de produtos e servicos tambem, mas omitiremos por brevidade

2. Restrição legal: Todo item de venda deve pertencer a exatamente uma nota fiscal.
3. Restrição legal: Todo item de venda se referirá a exatamente um produto ou serviço.
4. Restrição legal: Nota fiscal deve estar associada a exatamente um cliente pré-cadastrado.
5. Requisito do product owner[2]: deve ser facil de estender o sistema para especificar novas categorias de produtos e servicos no futuro, que ainda deverão ser associadas a um item de venda.
6. Requisito do product owner: uma vez criada uma NF, os seus itens de venda devem ser modificados, adicionados ou deletados apenas pelos metodos apropriados. Deve-se cuidar que não haja acesso de escrita inapropriado a lista de itens por outros meios.
7. Requisito do product owner: O objeto Cadastro é o único que pode criar Clientes, ou acessar o BD de clientes.

Atributos das Classes

Naturalmente há vários atributos para as classes:
Nota Fiscal: numero, valor (deve ser a soma dos valores dos itens), condições e data de entrega, alem dos dados do cliente.
Cliente: CPF, nome, endereco, telefone. Necessários para emitir a NF.
Item de venda: quantidade, desconto, condicoes e datas diferenciadas.
Produto: nome, preco/unidade, setor responsavel, informacoes sobre o processo e materias-primas, etc.
Servico: nome, preco/hora, setor responsavel, natureza (consultoria, treinamento, etc.)

[OBRIGATORIO] Parte I) Diagrama UML [2.0]

Defina o diagrama UML correspondente ao sistema, considerando as classes e requisitos/restrições acima (sem detalhar atributos). Para os seguintes relacionamentos, a notação UML deve indicar o tipo de relacionamento (associações, agregações, composições e heranças - vale criar classes auxiliares), e a multiplicidade das associações:
a) NF - Item
b) Item - Produto / Servico
c) NF - cadastro
d) Cadastro - ValidadorCPF

[IMPLEMENTAÇÃO] Parte II) Código [2.5]

Implemente um esqueleto de codigo com as associações deste sistema considerando:

Não precisa detalhar campos das classes, nem no papel, nem na UML, nem no codigo. Os campos existem para explicar a necessidade das classes para o aluno e ajudar a dar contexto. Pode implementar uma classe vazia ou apenas um campo simplificado representando os diversos dados da entidade quando escrever os outros campos seria apenas adicionar algumas atribuicoes. Por exemplo, a classe Cliente pode ter apenas um inteiro CPF, e omitir nome e endereco, ou um formato complexo para o CPF por brevidade.
É apenas necessário implementar

[2] product owner é o “dono” do sistema. Por exemplo, quem te contratou para desenvolver.

i) um esqueleto demonstrando as especificidades dos tipos de relacionamento (associacoes, agregações, composição, multiplicidades), como nos exemplos na aula, ou seja, apenas o código implícito ou arquitetural, para a parte II
ii) o minimo necessario para passar nos testes, para a parte III, ainda sem precisar detalhar campos.

[IMPLEMENTAÇÃO] Parte III) Testes [2.0]

Cadastro de Clientes, verificacao de CPF: Numa implementacao real, precisaria acessar um BD e mostrar uma janela ou widget para listar, ordenar, buscar e selecionar um cliente, ou executar um algoritmo no caso do CPF, mas aqui deve usar um test double: use mock, stub, spy, ou dummy, o minimo necessario para passar os testes, e não implemente o BD ou GUI. Se quiser pode criar tambem um test double para um BD de produtos/serviços.
AO USAR UM TEST DOUBLE, INDIQUE SE É UM MOCK, STUB, SPY, OU DUMMY
Implemente os seguintes testes:

a) cadastro de cliente (com CPF valido, precisa passar)
b) cadastro de cliente (com CPF invalido, precisa falhar)

c) cria NF com itens de compra, depois deleta itens e adiciona novos itens
verifica se os itens estao corretamente presentes na nota fiscal
verifica se o valor da nota eh correto
(ganha nota parcial mesmo se a implementacao nao for completa)

d) crie uma NF com APENAS um item de compra
troque esse item por outro item
verifique se o item e o valor esta correto depois da troca.
Lembre que de forma nenhuma o sistema pode permitir que exista uma NF vazia.

2-Uso de ferramentas xUnit [1.5]

Usamos o JUnit, e existem outras ferramentas similares. Teça 3 comentários justificados sobre estas ferramentas, onde estes comentários podem ser:

a. positivos: explicar uma vantagem de utilizar estas ferramentas. Não vale simplesmente dizer “é bom para testar”. Deve ser uma qualificação destes testes, ou uma consequencia positiva de se usar a ferramenta.
b. negativos: explicar um defeito, limitação ou risco de se utilizar a ferramenta. Por exemplo, um contexto ou tipo de teste para o qual a ferramenta não é apropriada.

OS COMENTÁRIOS NÃO PODEM SER OS 3 DO MESMO TIPO.

3-TDD e Repetiçoes: [1.5]

Joaozinho, ao implementar uma solução via TDD, começou implementando um caso trivial (um teste com entrada vazia, cuja implementação correspondente é apenas “return null”).
Ao seguir definindo mais testes e depois modificando a implementação com a transformação mais simples possivel para passar cada teste, o código adquiriu muitas repetições “copy-paste”, e longas estruturas switch-cases e cadeias if-elseif-elseif..., já que frequentemente a transformação mais simples para passar um teste era adicionar mais uma condição.
Comparando com a implementação da Mariazinha, que não seguiu TDD, vemos que a ideia geral de como resolver o problema é a mesma, mas no programa da Mariazinha a mesma logica foi implementada de forma mais curta e elegante, com loops, funções e fórmulas apropriadas que evitam as repetições e cadeias de condicionais do Joãozinho.
TDD realmente indica que, no passo de implementação do ciclo TDD, ao implementar uma nova funcionalidade, devemos realmente implementar a solução mais simples mesmo que não seja elegante ou eficiente. Então: O código do Joãozinho é realmente o que deve ser gerado pelo processo de TDD? **SIM** ( ) **NÃO** ( )
Se sim, explique qual o conceito e propósito do processo TDD nesse caso.
Se não, explique qual foi o erro de Joaozinho, em termos do processo TDD, utilizando a terminologia e considerando as 3 fases do ciclo do processo TDD: red->green->refactoring, ou seja, novo teste->nova implementação->refatoração.

4-Um Dalmatian é um Dog [1.5]

Dado que Dalmatian é subclasse de Dog, em Java não é permitido realizar a seguinte atribuição:

```
…
Dalmatian dalmatian = new Dog ( );
```

Onde a classe Dalmatian é subclasse da classe Dog.
Em termos de implementação, a razão é porque o tipo estático do objeto dalmatian é Dalmatian, e podem existir métodos em Dalmatian que não tenham sido implementados na classe do tipo dinâmico de dalmation, a saber, Dog. Na hipótese da atribuição da questão estar correta, seria perfeitamente legal enviar uma mensagem ao objeto dalmatian referente a método especificado apenas na classe Dalmatian – sem erro de compilação, pois o método é de classe estática Dalmatian –, mas não na classe Dog. Isso causaria um erro de tempo de execução!
Joãozinho esta especificando a nova linguagem JABBA (Joaozinho’s Amazingly Beautiful oBject lAnguage). Em JABBA, se pretende que o problema acima seja resolvido da seguinte forma:
Ao se chamar um metodo especifico da sub-classe, se o tipo dinâmico do objeto for a superclasse, o objeto será “promovido” à subclasse através de um construtor default. Por exemplo, se a proxima linha for:
dalmatian.countBlackSpots(); // only the class Dalmatian has this method, not Dog
o objeto será transformado em Dalmatian, com o numero de spots default (todo atributo de classe em JABBA tem um valor default).
Supondo que seja possivel implementar a linguagem dessa forma, isso é uma boa idéia conceitualmente em termos de OO? **SIM** ( ) **NÃO** ( ) ***Explique.***

5-Classe Abstrata [1.0]

Uma classe abstrata em Java pode não ter nenhum método abstrato? **SIM** ( ) **NÃO** ( )

Se sim, mostre com um exemplo simples porque isso seria útil. Senão, ilustre com um exemplo simples qual seria um problema (por exemplo, ser inútil, difícil de implementar, ou conceitualmente errado).

[OBRIGATÓRIA] 6-Polimorfica [2.0]

No trecho de código Java abaixo, claramente a classe SubClass estende a classe SuperClass. /Total 2.0/

a) Quais são as classes estática e dinâmica das variáveis supC, subC e supC nas linhas 01, 03 e 05, respectivamente? /0.5/

Linha	Variavel	Classe Estática	Classe Dinâmica
01	supC		
03	subC		
05	supC		

[b(i)] Nas linhas 02, 04 e 06, que valor a variável local x recebe ao final de cada atribuição? /0.5/

Linha	Variavel	Valor
01	supC	
03	subC	
05	supC	

[b(ii)] Explique, para cada uma dessas linhas, usando os conceitos de "classe estática", "classe dinâmica" e outros pertinentes, por que é assim e por que a atribuição compila corretamente? /0.5/

c) Examinando os métodos das duas classes, pode-se dizer que o princípio da substituição se preserva? Justifique sua resposta! /0.5/

```
public class SuperClass{
        public int add(int a, int b) { return a - b; }
        public int subtract(int a, int b) { return a + b; }
...
}
```

```
public class SubClass extends SuperClass{
        public int add(int a, int b) { return a + b; }
        public int subtract(int a, int b) { return a - b; }
...
}
```

Cliente de SuperClass e SubClass

```
...
SuperClass supC = new SuperClass( );        // linha 01
int x = supC.add(6,4);                      // linha 02
...
SubClass subC = new SubClass( );            // linha 03
x = subC.add(6, 4);                         // linha 04
...
supC = subC;                                // linha 05
x = supC.add(6,4);                          // linha 06
...
```