CES-28 Prova 3 - 2017

Sem consulta - individual - com computador - 3h

Obs.:

1. Qualquer dúvida de codificação Java só pode ser sanada com textos/sites oficiais da Oracle ou JUnit.
    a. Exceção são idiomas (ou 'macacos') da linguagem como sintaxe do método .equals(), ou sintaxe de set para percorrer collections, não relacionados ao exercício sendo resolvido. Nesse caso, podem procurar exemplos da sintaxe na web.
2. Sobre o uso do mockito, podem usar sites de ajuda online para procurar exemplos da sintaxe para os testes, e o próprio material da aula com pdfs, exemplos de codigo e labs, inclusive o seu código, mas sem usar código de outros alunos.
3. 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!
4. Só precisa implementar usando o Eclipse ou outro ambiente Java as questões ou itens indicados com o rótulo **[IMPLEMENTAÇÃO]**! Para as outras questões, você pode usar o Eclipse caso se sinta mais confortavel digitando os exemplos, mas não precisa de um código completo, executando. Basta incluir trechos de código no texto da resposta.
5. Submeter: a) Código completo e funcional da questão **[IMPLEMENTAÇÃO]; b)** arquivo PDF com respostas, código incluso no texto para as outras questões. Use os números das questões para identificá-las.
6. No caso de diagramas, vale usar qualquer editor de diagrama, e vale também desenhar no papel, tirar foto, e **incluir a foto no pdf dentro da resposta, não como anexo separado**. Atenção: use linhas grossas, garanta que a foto é legível!!!!

singleton builder , static factory, facade, template, composite, strategy, injection

Joãozinho programa Interpolação **[IMPLEMENTAÇÃO]**

O *package* InterpV0 inclui uma aplicação de interpolação numérica. Há duas classes que implementam métodos de interpolação (não precisa lembrar os detalhes de CCI22, basta lembrar o conceito de interpolação). E há outra classe MyInterpolationApp que realiza todo o trabalho. A proposta principal desta questão é transformar o package de Joãozinho em 3 *packages* Model, View e Presenter que implementam o padrão arquitetural MVP.

Deve incluir uma view funcional, mas que imprime no console, e com métodos que simulam entrada do usuário humano. Por exemplo, se o usuário humano deveria digitar um inteiro, basta haver um método set(int value). Quando a main() chamar este método, simulamos entrada de usuário.

Deve garantir que:

1. **[2 pt] O conceito de camadas seja seguido estritamente, e cada camada esteja em um package separado**.

Código

2. **[2 pt]** Que seja possível adicionar outras implementações da camada View, com as mesmas responsabilidades, e usar várias instâncias de Views diferentes ao mesmo tempo com a mesma instância de Presenter e Model, **sem necessitar mudar o código de Presenter ou Model**. .

Podem-se criar diversas subclasses view da superclasse IView e o código não precisara ser alterado pois presenter foi implementado usando a superclasse.
Como há atualização constante das variáveis não haverá problema em se instanciar várias vezes View.

3. **[2 pt] SUBQUESTÃO [IMPLEMENTAÇÃO]:** (esta parte envolve um padrão de projeto além do MVP). Seja possível implementar e escolher outros algoritmos de interpolação, **sem precisar mudar nada no código além de uma chamada de método para registrar o novo algoritmo.** *As camadas superiores apenas precisam escolher uma String correspondendo ao nome do método de interpolação desejado.*

Critério é satisfeito pois implementação foi feita com a superclasse `I_Interpolacao`. Assim, com a adição de novos métodos não sera necessária mudança no código, apenas para o reconhecimento da String.

[1 pt] Para cada uma das responsabilidades de MyInterpolationApp, indicadas com comentários no código e listadas abaixo, indique marcando uma colunas entre M, V ou P neste documento em qual camada deve ser incluída CADA responsabilidade. **DEVE CORRESPONDER AO SEU CÓDIGO**:

Obs item 1: a leitura é feita em V e a definição em M

	M	V	P
1. RESPONSABILITY: DEFINIR PONTO DE INTERPOLACAO (LEITURA ENTRADA DE USUARIO HUMANO)	X		
2. RESPONSABILITY: DEFINIR QUAL EH O ARQUIVO COM DADOS DE PONTOS DA FUNCAO (LEITURA ENTRADA DE USUARIO HUMANO)		X	
3. RESPONSABILITY: ABRIR E LER ARQUIVO DE DADOS	X		
4. RESPONSABILITY: IMPRIMIR RESULTADOS		X	
5. RESPONSABILITY: DADO O VALOR DE X, EFETIVAMENTE LER O ARQUIVO	X		
6. RESPONSABILITY: DADO O VALOR DE X, EFETIVAMENTE CHAMAR O CALCULO			X
7. RESPONSABILITY: CRIAR O OBJETO CORRESPONDENTE AO METODO DE INTERPOLACAO DESEJADO	X		
8. RESPONSABILIDADE: EFETIVAMENTE IMPLEMENTAR UM METODO DE INTERPOLACAO	x		

GRASP x SOLID

[1pt : 0.5 por princípio] Para a solução do exercício da interpolação, explique como a solução final promove 2 princípios GRASP ou SOLID (não vale os princípios que apenas definem menor acoplamento e separação de responsabilidades, High Coesion, Low Coupling, Single Responsability).

Liskov substitution: Classes podem ser substituídas por instancias de suas subclasses sem alterar o código, como pode-se perceber em Iview, I_Interpolacao que permitem implementação com suas subclasses sem alteração de código

Controller (grasp): O presenter do item anterior pode ser visto como controller atribuindo a responsabilides de manipular ações do sistema para uma classe que nao é interface do usuário.

Open/Closed principle (solid) : De acordo inclusive com o item 2 da questão 1 a nova implementação permite a inserção de novos métodos de interpolação(Open principle), ao mesmo tempo que não permite que usuário acesse métodos de interpolação diretamente (Closed principle)

DPs são tijolos para construir Frameworks

[2 pt: 2 * { a) [0.5] b [0.5] }]
Escolha **2 (dois)** DPs que ao serem aplicados como parte do código de um Framework, promovam:

a) o **reuso de código**

Strategy
A partir do uso de interfaces como superclasses pode-se reaproveitar métodos que façam usos da classes filha, basta implementar métodos utilizando a superclasse. Assim, caso deseje-se por exemplo, criar uma nova classe para ser usada no método, basta cria-la como subclasse da superclasse já existente.
Por exemplo:
Superclasse: Calcular frete
Subclasse: Calcular_frete_expresso, Calcular_frete_normal
Nova classe a ser adicionada sem alterar o código Calcular_frete_internacional

Dependency Injection
A partir do uso deste DP é possível definir a dependência entre classes no runtime.

Assim, caso deseje-se mudar a dependência de classes fazendo uso de novas classes, basta criar o método inject desta classe devidamente, fazendo reuso do código já existente.

b) a **separação de interesses** (separation of concerns), entre o código do framework e o código do programador-usuário do framework.

Observer- promove o não acesso de uma camada a outra, fazendo apenas os ajustes de estados, fazendo assim a separação de interesses

Builder pode promover o não acesso a métodos construtores por parte do usuário.

Explique conceitualmente como cada um 2 DPs promove os 2 conceitos a) e b). Vale usar diagramas UML na explicação, mas *deixe claro o que deve ser implementado pelo framework e o que deve ser implementado pelo programador-usuário do framework*.

Abusus non tollit Usum

Conceito	Consequência do Abuso do conceito Marque o número apropriado conforme lista abaixo
Singleton DP	1 2 3
Dependency Injection	1 2 3
Getters and Setters	1 2 3

1. Excessiva quantidade de código e classes auxiliares para inicializar objetos
2. Acoplamento excessivo e código difícil de entender devido à proliferação de Dependências e conflitos de nomes.
3. Confusão semântica dependendo da ordem de chamada de métodos, resultando em objetos com estado inválido.

a) **[0.5]** Associe cada conceito à consequência do seu abuso, marcando os números apropriados na a tabela acima, conforme a lista acima.

Singleton – 1
Dependency Injection- 3
Getters and Setters- 2

b) **[1]** Escolha Singleton ou Dependency Injection e explique a causa da consequência, explicando o contexto do abuso do conceito.

Com o uso excessivo de Singleton são necessárias muitas e longas classes que verifiquem se o objeto em questão já foi instanciado. Assim, há excesso de código para inicializar os objetos.

No caso do Dependency Injection muitas vezes com o intuito de se definir a classe que se acopla no run time pode-se causar uma confusão semântica tornando difícil de entender qual objeto e fato esta sofrendo inject

c) **[0.5]** Para o mesmo conceito escolhido em b), explique um contexto de uso apropriado, em que há razões claras para se utilizar o conceito sem incorrer nas consequências negativas.

No caso do singleton, por exemplo, quando se deseja ter apenas um acesso ao banco de dados, ou seja, que todos os dados sejam salvos e consultados no mesmo lugar, torna-se muito útil o uso de singleton