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!!!!

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**.
   R: Pensando nisso foi criado um package para as View's, Model's, Presenter's, Main O package Main foi implementado representando os inputs do Usuário caso o programa possuísse uma interface gráfica.
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**.
   R: Foi criada uma interface genérica de View a qual visa a implementação de um método print para o usuário e um set de Presenter. Isso já induz a utilização do DP Observer que utilizará notificações do Presenter para dar update nas View's – novas ou antigas. Dessa forma, implementando a interface InterfaceView e Observer, novas implementações de novas View's não dependerão da implementação dos Model's e Presenter's

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.*
   R: O Presenter implementado mediantes métodos get e set nao utiliza nomes de modelos liteirais na sua implementação. Dessa forma, ao implementar um novo modelo mediante o getMethod() a abstração do método será encaminhada para o método de calculo adequado.

[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**:

	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).
R: Um dos principais princípios SOLID afetados é o Denpendency Inversion Principle, pois utilizando a construção MVP tenhos que as dependências fazem referencia a abstrações e não a objetos concretos. Além disso, pode-se citar o Open/Closed principle, uma vez que as interfaces de Model's e View's e a utilizaçãoo do DP Observer, cria uma estrutura fechada para modificações e aberta para extensões de View e Models.

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

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*.
R: O primeiro Desing Pattern muitíssimo utilizado para o reuso de código e a separação de interesses é o Template Method, pois ele controi o esqueleto do código a ser implementado mediante o uso de classes abstratas e mediante métodos ou classes Hook ele projeta o que é necessário ser implementado para a utilização do método e separa do que é padrão da implementação. Dessa forma, encaminhando o usuário a so implementar o necessário, ele contribui para o reuso do código e da separação dos interesses.

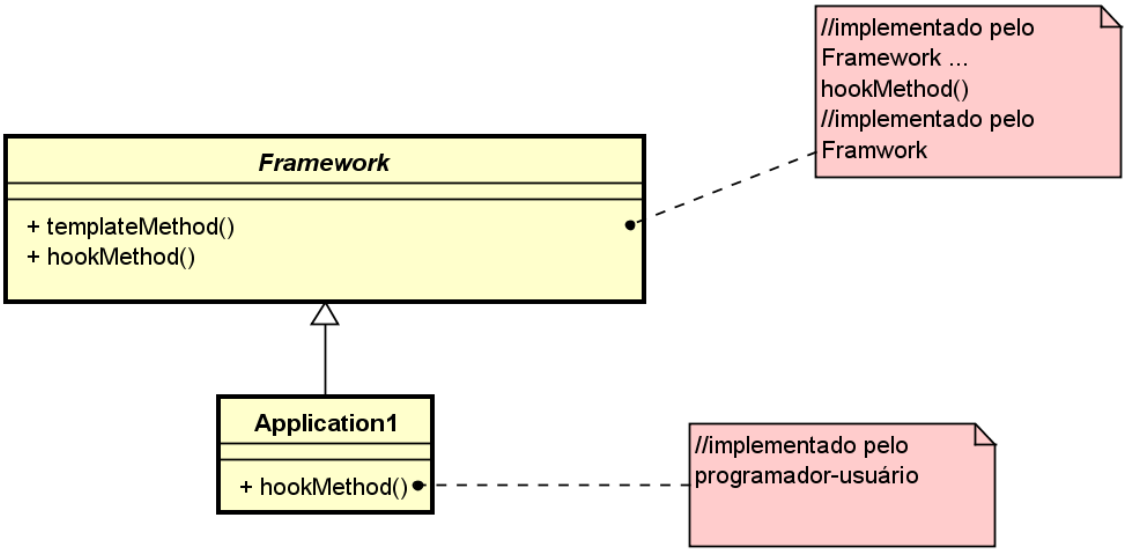

Figure 1 - UML Template Method

Por outro lado, o Desing Pattern Facade também é uma boa escolha quando deseja-se utilizar frameworks, pois ele garante que subsistemas sejam independentes e que a adição de subsistemas seja sem interferência no sistema como um todo. Dessa forma, garante o reuso do código mediante o compartilhamento de subsistemas e garante a separação de interesses pois cada subsistema pode estar associado a um parte diferente do projeto.

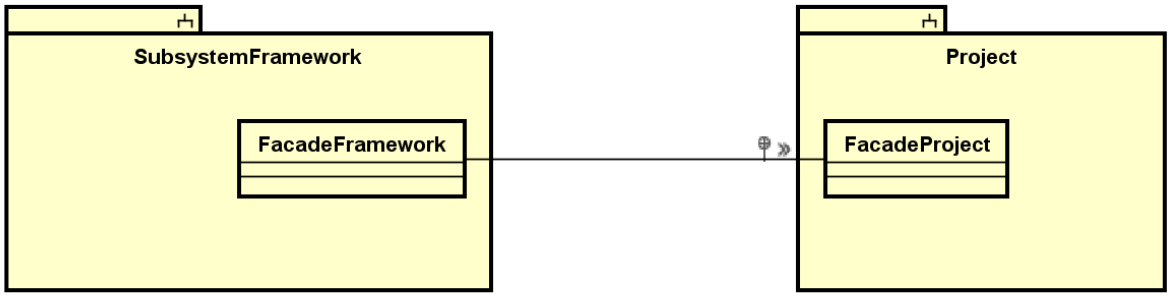

Figure 2 - UML Facade

Abusus non tollit Usum

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

b) **[1]** Escolha Singleton ou Dependency Injection e explique a causa da consequência, explicando o contexto do abuso do conceito.
R: Ao utilizar o Singleton, simploriamente falando, temos uma classe que gerencia todas as outras classes de um pacote. Isso é extremamente útil para casos de programas excessivamente grandes, pois mantem uma dependência entre pacotes mediante essa interação com o Singleton. No entanto, quando o conceito de separação de interesses é abusado, o programador cria diversos pacotes com poucas classes/logicas internas. Por exemplo, a cada 2 classes ele cria um novo pacote e um novo classe Singleton para essas 2 classes, em um programa com 200 classes não Singleton, teria-se 100 classes Singleton com 100 pacotes. Isto é, um abuso do DP Singleton.
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.
R: Repetindo o que foi dito anteriormente, em programas enormes, o Desing Pattern Singleton é praticamente indispensável, pois ele garante que o programa seja separado em grandes blocos de pacotes com uma lógica interna. Isso faz com que a extensão, manuteabilidade e o reuso do código sejam extremamente mais simples, pois a implementação de um pacote so depende dele próprio e a classe Singleton seria a responsável pela comunicação com os demais pacotes. Resumidamente, o DP Singleton bem utilizado consegue melhorar todos conceitos SOLID, por exemplo o Open/Closed e o Liskov Substitution principle que seriam um dos principais.