CES-28 Prova 3 - 2017

Sem consulta - individual - com computador - 3h

Aluno: Eduardo Henrique Ferreira Silva

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**.

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**.

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.*

   Observação: no item c), a chamada de método para registrar o novo algoritmo deve ser feita no construtor do Presenter. O local correto está indicado no código.

[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).

RESPOSTA:

A solução final promove o princípio *Controller* do GRASP, pois atribui à um tratador oficial (Presenter) a responsabilidade de tratar todos todos os eventos do sistema. Nesse caso, Model é somente o bando de dados de algoritmo e View é o pacote que contém todas as possíveis UI's e observadores do presenter.

Além disso, a implementação da interface iViews promove o *Open/Closed Principle*, do SOLID, pois possibilita faz com que todas as views sejam criadas obedecendo um conjunto predeterminado de regras que não devem ser mudadas (Closed), mas não impede que novas funcionalidades sejam implementadas em uma nova view (Open).

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*.

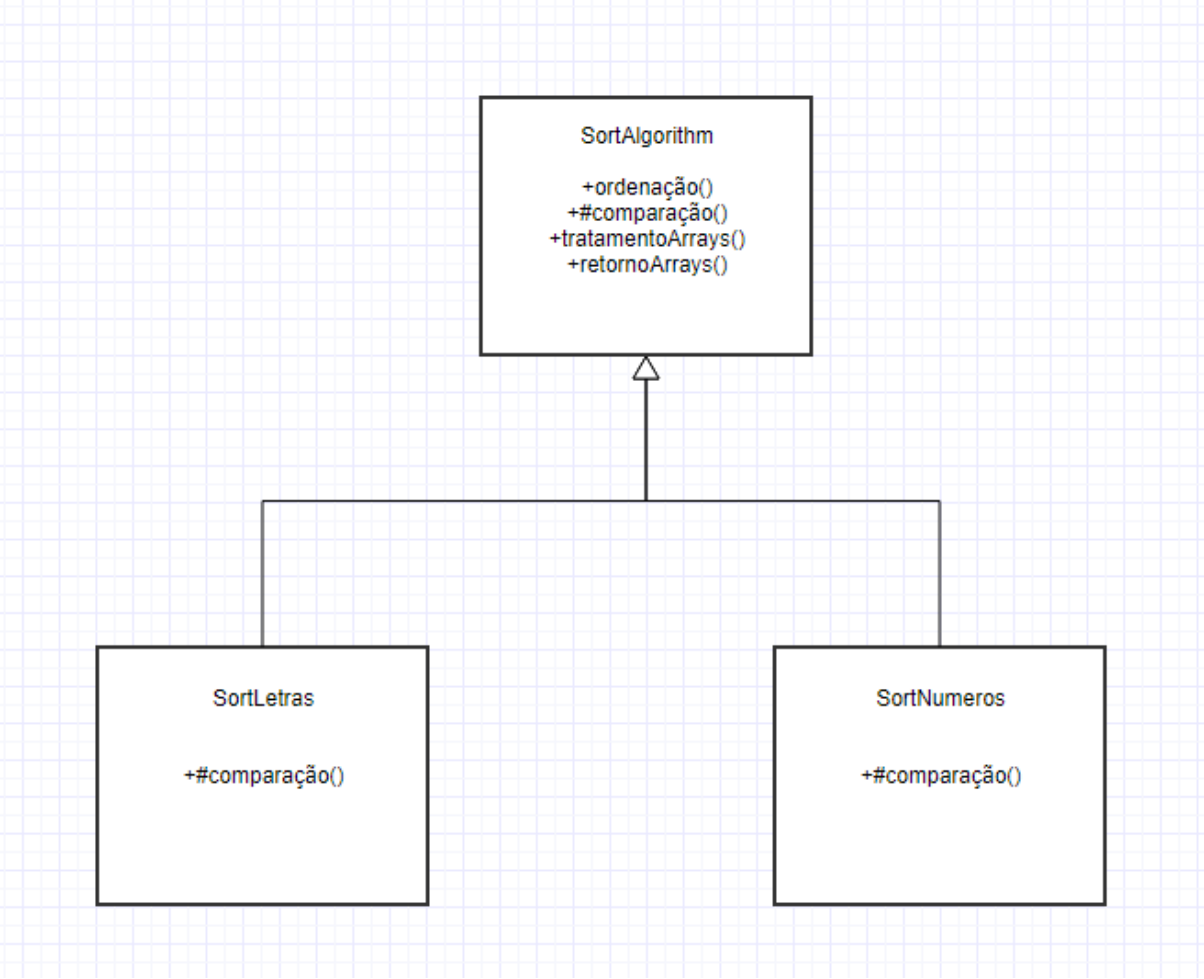
RESPOSTA:

Template Method:

a) Um Template Method promove o reuso de código pois ele permite que passos comuns a algumas atividades sejam programados uma só vez e, após isso, utilizados em cada uma das atividade.

Por exemplo, imaginemos que vamos construir um framework de ordenamento de arrays, em que construir todo o arcabouço para que o usuário faça sua ordenação somente definindo o modo de se comparar os itens. Assim, poderemos implementar todos os passos, como o tratamento e o retorno dos arrays, e a ordenação em si somente uma vez e cada usuário utilizara esses códigos uma vez que implemento a função de comparação.

b) Além disso, o template method possibilita a separação de interesses, pois o programador só precisa se preocupar com as etapas comuns do código e o usuário somente com a parte individual a cada caso. Por exemplo, no exemplo do framework de ordenamento que utilizamos, o programador do framework só precisa se preocupar com as etapas de ordenamento, processamento dos arrays e retorno dos arrays, enquanto o usuário precisa somente se preocupar com a implementação da comparação entre os objetos.



Facade:

a) O facade promove a reutilização de código pois permite uma interface simples para um conjunto de ferramentas que pode ser infinitamente complexo.

Suponhamos que exista um framework muito complexo que executa diversos tipos de tarefas. Sem uma fachada simplificada, um usuário que precisa utilizar somente uma pequena parte desse framework pode preferir reescrever as funcionalidades que ele necessitado que ter que lidar com toda a complexidade do framework. Com uma facade simples, é mais fácil para o usuário utilizar a framework do que refazer o que já foi implementado. Logo, o padrão facade promove a economia e o reuso de código, pois ele só precisara implementar as classes que utilizar das funcionalidades da framework e não programar as funcionalidades em sí, o que foi feito pelo programador do framework.

b) Ademais, o facade promove a separação de interesses entre o código do usuário e do programador pois o usuário não precisa se preocupar com a estrutura interna do código do framework, o que é de responsabilidade única do programador do framework. O usuário só precisa compreender e utilizar a facade nos seus códigos. Ademais, o programador do framework não precisa sobre como o usuário utilizara seus códigos, uma vez que a facade permite encapsular toda a complexidade e oferecer ao usuário somente aquilo que é seguro que ele use. Ele precisa, então, se preocupar somente com a implementação e organização do framework em sí, e em oferecer ao usuário, através da facade, somente aquilo que é seguro ser oferecido.

Abusus non tollit Usum

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

RESPOSTA:

Singleton:

Em alto nível, podemos considerar um singleton como sendo uma variável global: todo o código se utiliza de somente um objeto.

Pensemos no seguinte exemplo: as classes A, B e C estavam completamente desacopladas. Entretanto, todas elas agora terão como atributo um objeto de uma dada classe S, que é um singleton.

Observemos que agora as classes A,B e C estão completamente acopladas, dado que se qualquer uma que alterar seu objeto de S alterará os objetos de C das outras duas.

Logo, gerou-se acoplamento. Acoplamento esse que pode ser muito prejudicial ao código.

Ademais, suponhamos que A chamou se objeto de S de "_objetoS", B o chamou de "X" e C o chamou de "singleton". Observemos que isso piora a legibilidade do código, pois por mais que os três nomes sejam diferentes, eles se referem a um mesmo objeto.

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.

RESPOSTA:

Uma boa utilização de singleton, por exemplo, é quando estamos lidando com um objeto de instanciação muito custosa ou demorada. Assim, forçamos que a instanciação seja feita somente uma vez, evitando desperdício de poder computacional e/ou tempo.