

CES-28 Prova 3 - 2017

Sem consulta - individual - com computador - 3h

Dylan Nakandakari Sugimoto

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 código 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 confortável 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.**

Temos um pkg para model, presenter e View. O nosso model são as classes que implementam um método de interpolação e que implementam a interface já existente e que representa um método de interpolação. A nossa View é a classe que imprime algumas coisas no console e pega dados de entrada do usuário, nesse caso, vamos implementar a entrada de dados do usuário usando métodos setter como enunciado. E o Presenter faz esse meio de campo entre a View, que não tem mais um ponteiro do model, e o model.

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.****

Como usamos o DP Observer e a nossa view implementa uma interface, basta uma outra view implementar a Iview que simplesmente é a interface Observer do java.util, e criar um objeto Presenter para realizar o addObserver e chamar outros métodos do Presenter. O código do Presenter e do Model não precisam ser alterados. O DP Observer resolve esse problema de garantir a notificação de todos os Observers quando ocorre o cálculo do resultado da interpolação. Assim, é possível usar várias instâncias de View para o mesmo Presenter, e como o Presenter gerencia as chamadas para cálculo da interpolação (ou chamadas para os models) também é possível usar para as mesmas instâncias de models.

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.*

Escreveu-se um método em que o usuário pode criar o seu model que precisa implementar a interface do model, e depois em tempo de execução ele precisa instanciar a classe que ele criou e passar como parâmetro para a view do método defineNewInterpolationCalculate que delega para o Presenter registrar essa instancia como novo método de interpolação. Assim, basta o usuário passar o ponto de interpolação para a view que ela pede para o Presenter calcular a interpolação usando o novo algoritmo.

[1 pt] Para cada uma das responsabilidades de MyInterpolationApp, indicadas com comentários no código e listadas abaixo, indique marcando uma das 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)		V	
2. RESPONSABILITY: DEFINIR QUAL EH O ARQUIVO COM DADOS DE PONTOS DA FUNCAO (LEITURA ENTRADA DE USUARIO HUMANO)		V	
3. RESPONSABILITY: ABRIR E LER ARQUIVO DE DADOS			P
4. RESPONSABILITY: IMPRIMIR RESULTADOS		V	
5. RESPONSABILITY: DADO O VALOR DE X, EFETIVAMENTE LER O ARQUIVO			P
6. RESPONSABILITY: DADO O VALOR DE X, EFETIVAMENTE CHAMAR O CALCULO			P
7. RESPONSABILITY: CRIAR O OBJETO CORRESPONDENTE AO METODO DE INTERPOLACAO DESEJADO			P
8. RESPONSABILIDADE: EFETIVAMENTE IMPLEMENTAR UM METODO DE INTERPOLACAO	M		

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 Responsibility).

Temos o [Dependency Inversion](#) que é representado pela relação entre a View e o Presenter, o Presenter possui um ponteiro para uma abstração da View, assim o Presenter não depende de detalhes da View apenas da sua abstração, ou seja, abstração não depende de detalhes. E acabou que o Presenter também possui um ponteiro para uma abstração do model (ou para sua interface InterpolationMethod), assim o Presenter não depende dos detalhes do model também, sendo possível trabalhar com vários models diferentes. Isso facilitar a extensão, caso algum usuário programador quisesse estender para algum algoritmo de interpolação inexistente, ele apenas precisa implementar a interface, escolher

uma string para o Presenter identificá-la quando o View repassar a escolha do usuário e adicionar a linha em que o Presenter cria o novo model. Seria então o conceito de Open to extension and closed to modification, ou seja, é fácil estender sem precisar modificar.

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*.

O Hook class promovem reuso de código e separação de interesse entre framework e o código do usuário, pois o hook class é uma classe que define um algoritmo que vai ser usado em uma template class, por exemplo, que possui parte comuns e imutáveis e outras partes que pode ser variável, que é a parte do hook class. Assim, o usuário pode criar a sua hook class (concreta, herdando da classe abstrata definida pelo framework) e o framework possui a sua template class, e os códigos do framework e do usuário estão separados, mas de certa forma acoplados, porque o que o usuário programador escreveu vai ser usado em tempo de execução pelo framework.

Abusus non tollit Usum

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

Dependency Injection escreve muitas interfaces, cada classe pode implementar múltiplas interfaces ao mesmo tempo, e as classes que possuem agregações dependem de que outras classes inicie e passe as instâncias ficando assim com código em quantidade excessiva e necessita de classes auxiliares para inicializar objetos.

Singleton em excesso complica o entendimento do código porque várias variáveis com nomes diferentes vão apontar para a mesma instância podendo haver conflito de nome, e esse conflito pode resultar em objetos com estado inválido devido à confusão semântica.

Getter e Setter em abuso gera código em excesso, pois nem sempre precisa de getter ou setter, getter aumentam o acoplamento e o uso incorreto dos setter pode levar os objetos a estado inválido.

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

Na tentativa de deixar o código com baixo acoplamento e seguir a risca o conceito de Dependency Inversion, ou seja, deixar todo o código dependente apenas de abstrações, de forma que seja fácil estender sem precisar modificar, acaba por usar em excesso o DI e o projeto fica com código em excesso, pois não se sabe se algum dia realmente vai ser necessário essa maleabilidade ou flexibilidade da aplicação do DI.

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.

Desenvolvimento de software com recebimento de dados externos ou comunicação externa (a dispositivos de hardware, sensores ou banco de dados). Nesse caso, o uso de DI é bem adequado, pois o baixo acoplamento é necessário e, efetivamente, está se aproveitando dos benefícios da aplicação do DI.