

# 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 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.
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**.

View possui dois construtores, um sem argumento e outro com argumento que recebe outra view. Se o segundo for usado: `view1 = new View( view 2)`, então view 1 possuirá o mesmo presenter e model de view 2.

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.

Dentro de models há um hashmap que guarda o novo algoritmo instanciado.

[1 pt ] Para cada uma das responsabilidades de MyInterpolationApp, indicadas com comentários no código e listadas abaixo, indique marcando uma coluna 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 Responsibility).

- Grasp: **Especialista de informação:** Com a refatoração do código, foram separadas responsabilidades entre 3 classes, isto é, atribuindo maior especialização e removendo a existência de uma classe “faz-tudo”.

**Indireção:** A introdução de uma classe mediadora (o Presenter) respeita este padrão GRASP. Este padrão suporta low-coupling e torna o código mais reusável.

**Creational:** No código, este princípio foi respeitado, a classe criadora dos métodos de interpolação possui a informação necessária para fazê-lo. E necessita diretamente de sua existência.

- Solid: **Open/Closed Principle:** Como o código foi estruturado, assume-se que há uma operação padrão, viewer recebe a data, passa para o presenter ler e o presenter passa o dado construído para o model que executa os cálculos e retorna o valor final. Por isso, o método interpolate object é mantido como método abstrato, ele não deve ser modificado, se o programador quiser mudar o viewer, ele pode estender o viewer dando-lhe mais funcionalidades, porém este método nunca lhe estará acessível para modificação.

**ISP:** Como foi configurado o código, o cliente enxerga apenas uma classe e necessita executar apenas um método. Todos os outros métodos ou classes utilizadas no processo são escondidos deste.

## 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.

**Padrão Visitor:** O padrão visitor engloba toda uma lógica por trás, mas para um programador usuário basta-lhe apenas implementar novas classes visitor de acordo com o padrão definido pela interface, a lógica de visita é implementada pelo framework. Então este padrão a) reutiliza a lógica de visita, havendo aproveitamento de código e b) permite que o usuário programador seja responsável apenas pela codificação de novas classes visitor.

**Padrão Observer:** O padrão observer se trata de uma lógica reutilizável, cabe a apenas ao programador usuário estender a classe observador e observável de acordo com as suas necessidades, para citar exemplo do próprio Java que já possui este padrão implementado. Então este padrão a)reutiliza a lógica observador e observável, não há necessidade de implementar os detalhes da mecânica, o usuário programador só deve estender por exemplo o método update para que este execute as funções de seu desejo e por tanto satisfazendo b).

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.

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

**Singleton:** O Singleton acaba por ser uma instância global, ou seja, ele possui relações com todas as classes, qualquer classe pode chamar a sua existência sem a necessidade de este

ser passado como parâmetro. Se existe um singleton apenas no código e sua existência global faz sentido no campo das idéias este problema não causa grandes convoluções. Todavia se há várias classes que possuem a propriedade de ser singleton o acoplamento se torna excessivo, o diagrama UML se torna uma teia com várias setas apontando para os singletons. O seu uso deve ser obstinado à um objeto cuja presença é de fato global para todos os outros objetos.

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.

O livro do professor Guerra cita por exemplo o contexto de um jogo de xadrez em que o tabuleiro seria um singleton. Esta idéia faz total sentido pois há apenas um tabuleiro no jogo de xadrez e todos os outros objetos envolvidos num jogo de xadrez estão relacionados por meio deste tabuleiro.