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.
   1. 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:** Por favor, verificar as versões **InterpV1 e InterpV2, separadamente.** As duas versões foram desenvolvidas de forma independente a fim de tentar resolver o problema. Nenhuma substitui a outra. **Foram tentativas diferentes de solução.** Levar em conta a estrutura/divisão adotada, não o resultado final.

1. **[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**.
2. **[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.*

**[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**:

**R:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | M | V | P |
| 1. RESPONSABILITY: DEFINIR PONTO DE INTERPOLACAO (LEITURA ENTRADA DE USUARIO HUMANO) |  | X |  |
| 1. RESPONSABILITY: DEFINIR QUAL EH O ARQUIVO COM DADOS DE PONTOS DA FUNCAO (LEITURA ENTRADA DE USUARIO HUMANO) |  | X |  |
| 1. RESPONSABILITY: ABRIR E LER ARQUIVO DE DADOS |  |  | X |
| 1. RESPONSABILITY: IMPRIMIR RESULTADOS |  | X |  |
| 1. RESPONSABILITY: DADO O VALOR DE X, EFETIVAMENTE LER O ARQUIVO |  |  | X |
| 1. RESPONSABILITY: DADO O VALOR DE X, EFETIVAMENTE CHAMAR O CALCULO |  |  | X |
| 1. RESPONSABILITY: CRIAR O OBJETO CORRESPONDENTE AO METODO DE INTERPOLACAO DESEJADO |  |  | X |
| 1. 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:**

## **GRASP: Controller** – O Presenter implementado funciona como um Controller. Ou seja, aplicando o padrão MVP, delega-se a responsabilidade de manipular eventos do sistema para uma classe que não seja de interface do usuário, separando a logística de processamento (de negócios) da logística de exibição. Assim, atribui-se a responsabilidade do tratamento de um evento do sistema a uma classe que representa ou um controlador de fachada ou um controlador de caso de uso, o que comprova a obediência ao princípio Controller de GRASP.

**SOLID: Open/Closed Principle:** Com o emprego de Interfaces do padrão MVP, o projeto fica aberto a extensões, facilitando a inserção de um novo tipo de View (Interface com o Usuário), por exemplo, sem necessitar mudar o código de Presenter ou Model. Ademais, como o fato de o Presenter intermediar a relação entre View e Model contribui para o cumprimento do princípio de fechamento para modificações, obedecendo completamente o Open/Closed Principle de SOLID.

## 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:

1. o **reuso de código**

**R:**

Template Method

Dependency Injection

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

**R:**

Observer

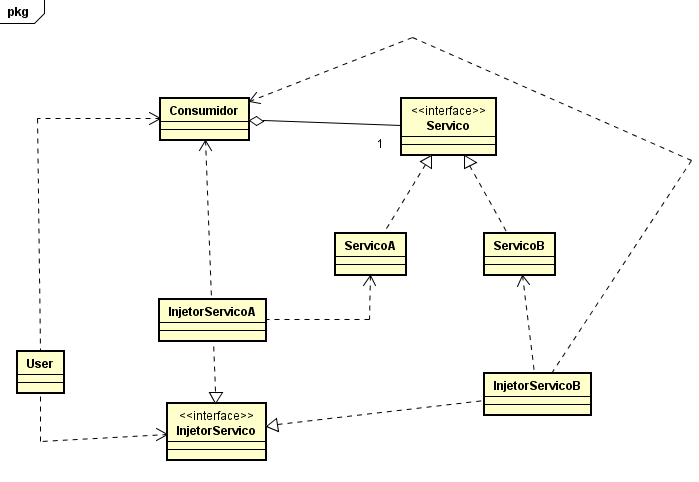
Façade

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:**

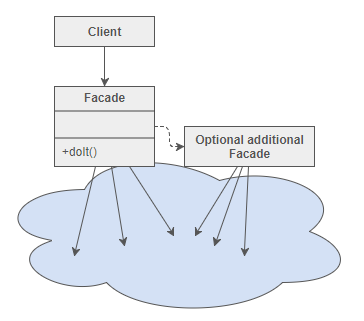
a) **Template Method:** O padrão de projetos Template Method define o esqueleto de um algoritmo em uma operação, delegando a “especialização” e o “preenchimento da forma” para as respectivas subclasses. Dessa forma, em um Framework, aplicando-se o padrão de projetos Template Method como parte de um código de framework, aproveita-se todo o código-base utilizado no “esqueleto” e apenas implementa-se as especializações a cada nova operação/utilização.

**Dependency Injection:** O padrão de projetos Dependency Injection permite o aproveitamento/reuso do código por adição de extensões, possibilitadas pelas interfaces dos “serviços” e dos respectivos “injetores”. Uma vez implementadas as interfaces, o acréscimo de um novo serviço genérico é bastante simples, conforme se pode comprovar no diagrama UML a seguir, feito no software ASTAH:



b) **Observer:** Como o padrão de projetos Observer pode ser intensamente explorado no padrão Model-View-Controller, ele contribui para a separação de responsabilidades entre as classes, uma vez que, as views, por exemplo podem ser vistas como empregos de Observer, a serem atualizadas sempre que o Subject (Observable) tiver seu estado alterado. Assim, tal padrão possibilita o desacoplamento entre a logística de processamento e a logística de exibição, contribuindo para que, em um Framework, o código do framework e o código do programador-usuário do framework tenham interesses convenientemente separados.

**Façade:** Oferece uma interface para um conjunto de interfaces em um subsistema. O padrão de projetos Façade define uma interface mais alto nível (“fachada”) que torna o subsistema mais fácil de usar. Como esse padrão permite englobar um subsistema mais complicado, convertendo em uma interface mais simples, ele é bastante útil para a finalidade de separação de interesses, como, por exemplo, em um Framework, entre o código do framework e o código do 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.

**R:**

**Feito na Tabela.**

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

**R:**

Singleton – **Consequência do Abuso:** Excessiva quantidade de código e classes auxiliares para inicializar objetos

**Causa:** Como o padrão de projetos Singleton utiliza o padrão de projetos Static Factory Method, é utilizada a implementação de “novos construtores”, a fim de se garantir uma instância única do objeto em questão, transformando o construtor padrão em private e inicializando dentro da própria classe do objeto em questão um objeto “null”. Ademais, dentro do “novo construtor”, impõe-se uma lógica para saber se já houve algum objeto daquela classe instanciado. Uma forma (mas não a única!) de se fazer isso é verificar tal condição: se foi instanciado (objeto diferente de null), retorna-se esse objeto; se não foi, instancia esse objeto pela primeira e única vez. Todo esse procedimento, quando utilizado em larga escala de forma abusiva, promove como consequência excessiva quantidade de código e classes auxiliares para inicializar objetos. Exemplos do uso de Singleton são vistos a seguir, para que os dois “efeitos colaterais” supracitados possam ser observados:

***Exemplo 1:***

*public final class Singleton {*

*private static volatile Singleton instance;*

*private Singleton() {}*

*public static Singleton getInstance(String value) {*

*if (instance == null) {*

*synchronized (Singleton.class) {*

*if (instance == null) {*

*instance = new Singleton();*

*}*

*}*

*}*

*return instance;*

*}*

*}*

***Exemplo 2:***

*public class Singleton {*

*private Singleton() {}*

*private static class SingletonHolder {*

*private static final Singleton INSTANCE = new Singleton();*

*}*

*public static Singleton getInstance() {*

*return SingletonHolder.INSTANCE;*

*}*

*}*

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:**

O padrão Singleton garante que uma classe tenha apenas uma instância e fornece um ponto de acesso global a essa instância. É importante ressaltar que o padrão de projeto Singleton deve ser considerado apenas se os três critérios seguintes forem satisfeitos:

* Há a necessidade de se assegurar que um objeto só vá ser instanciado uma única vez
* A inicialização “lazy” é desejável
* O acesso global não é fornecido de outra forma

Se a instância única, quando e como a inicialização ocorre, e o acesso global não são problemas, o Singleton não é suficientemente interessante.