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**.
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.*

Feito. Padrão Strategy

**[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 |  |
| 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).

Open/Closed Principle (SOLID) - O código é aberto para qualquer extensão de algoritmo de interpolação novo que queira ser adicionado aos models.

Polymorphism (GRASP) - Os models implementam uma interface de algoritmo de interpolação. Com isso, o código passa a depender apenas de implementação dessa interface que define o comportamento de todos os models usáveis no programa que podem ser empregados via polimorfismo.

## 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**
2. 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*.

Prototype - Em frameworks de desenvolvimento de jogos, é comum que existam vários objetos de uma mesma classe ativos em cena ao mesmo tempo, dividindo o mesmo código. O DP prototype otimiza esse processo de instanciação de objetos (muitas vezes complexos) em cena. Assim, o usuário apenas precisa criar a classe do objeto, instanciá-lo uma vez e salvar essa primeira instância como o protótipo. E em outros códigos do desenvolvimento do jogo, ao invés de instanciar o objeto a partir de sua classe (o que aliás, muitas game engines que usam esse DP não deixam o usuário fazer), o programador chama- uma função PrototypeFactory (onde podemos passar o objeto protótipo como argumento) que instancia o objeto a partir de uma cópia de seu protótipo já existente. A maneira como esse protótipo é instanciado e colocado no espaço do jogo é responsabilidade da engine.

Update Pattern - O update pattern é um outro DP utilizado em game engines (usado por Unity, Unreal e Game Maker por exemplo). Esse padrão faz com que cada game object do jogo tenha uma função update que é chamada cada vez que o jogo atualiza seu frame. A maneira como essa função é chamada e otimizada é responsabilidade da engine, mas ela obriga que os objetos do jogo sejam filhos de uma classe que implementa essa função update (geralmente uma classe game object) mesmo que essa função nada faça. Com essa imposição, a engine usa o mesmo código de chamada de update para todos os objetos existentes no jogo (bastando ter um tracker deles em algum lugar para poder invocar a funçao).

## 

## 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 associa uma instância de um objeto à declaração de sua classe, tornando-o um objeto de acesso global para qualquer outro objeto existente no programa. Ou seja, como a instância é um objeto, cria-se para cada classe que se relaciona com o singleton uma dependência direta. O mau uso pode gerar códigos de singletons cada vez maiores e com mais responsabilidades e cada vez mais dificuldade de se mudar códigos de classes do programa sem que tenha que se mexer (e poluir) ainda mais o código do singleton. Um bom exemplo de mau uso está na criação de objetos \_GameManager\_ em códigos de jogos. Geralmente isso tem a ver com pouco conhecimento de orientação a objetos e de devida derivação de responsabilidade em classes do jogo. Com isso, cria-se tal singleton que fica responsável por cada interação entre os objetos do jogo de modo que toda e qualquer informação trocada entre as classes passe antes por ele, assim, cria-se um código confuso, com muito acoplamento, dependência e responsabilidade excessiva na classe 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.

Um bom contexto para a utilização de singleton em jogos são objetos em que temos garantia de serem pouco mutáveis ao decorrer do projeto e que devem existir durante quase toda a execução do jogo. Um exemplo de utilização é um objeto gerenciador do audio do jogo, ou um AudioManager. Esse objeto tem uma responsabilidade bem definida: tocar os sons do jogo quando algum objeto qualquer (que tem acesso global ao audio manager) requisitar. Com isso, esse singleton apenas tem uma Hash Table com os audios e seus respectivos nomes, e uma função para tocar um audio a partir de um nome. Assim, basta que algum objeto que queira que um som seja tocado no momento de alguma ação, envie uma mensagem para o singleton: \_audiom = AudioManager.instance(); \_audiom .Play("ShotSound");