

CES-28 Exame - 2017

Com consulta - individual - com computador - 3h

Gabriel Adriano de Melo

QUESTÕES

- 1) **[1.5 PTOS]** De posse do problema apresentado na Parte I, mapeie 03 requisitos funcionais, os escrevendo por via de histórias de usuários (*user stories*), atentando para que os mesmos atentem para possuir as características do mnemônico INVEST (Independente, Negociável, Valiosa, Estimável, Pequena (Small) e Testável.
 - Enquanto um drone (ARP), envio minhas informações de posicionamento, para que o sistema de controle possa funcionar corretamente. **(1)**
 - Enquanto estação de controle (GCS), possuo um sistema de mapas, com informações enviadas da UTM-CTR, para que eu possa realizar a navegação com segurança. **(2)**
 - Enquanto sistema de controle e vigilância (UTM-CTR), compilo e gero um mapa síntese de informações (mapa de consciência situacional), para que eu possa enviar esse mapa síntese (**sa info**) para as GCS. **(4)**USERPOINTS em parênteses no final, em negrito.
- 2) **[0.5 PTOS]** Apresente (explique), como suas palavras, em cada uma das histórias de usuários os atributos contidos em INVEST são obtidos.
 - Independentes: as user stories são independentes umas das outras, no sentido de que a ordem na qual cada uma é implementada pode ser qualquer.
 - Negociável: As user stories são simples e podem ser reescritas e alteradas sem complicações.
 - Valiosa: Elas trazem valor ao projeto, satisfazem os requisitos básicos do projeto e não são supérfluas.
 - Estimável: Elas tem o seu tempo (carga de trabalho) estimadas por meio dos users-points.
 - Pequenas: São simples e não demandam muito tempo para serem concluídas (epics que deveriam ser divididos em user-stories menores).
 - Testável: O bom entendimento delas é suficiente para que os requisitos sejam testados.
- 3) **[1.0 PTOS]** Perceba que na descrição do UTM-CTR informa que ele deve ser implementado de forma única, ou seja, não pode existir mais de uma instância dele rodando por vez. Qual o padrão de projeto que resolve essa questão? Explique.

O padrão de projeto Singleton resolve esse problema. Ele garante que cada classe (nesse caso um sistema) tenha apenas uma instância, e também prove um ponto de acesso global a ela. No caso de ser um sistema, um servidor que todos conhecem faria o papel do Singleton, com um ponto de acesso global a ele e também com apenas um dele funcionando, isto é, apenas uma instância.

- 4) **[2.0 PTOS]** O problema apresentado na Parte I, claramente nos remete a uma série de padrões aprendidos no ano de 2017 em CES 28. Entretanto, de forma bem explícita podemos ver o padrão arquitetural MVP (*Model View Presenter*) e o Observer. Usando o problema, construa um diagrama de classe que apresente uma solução para o problema.

IMPORTANTE: Não foi considerado o canal **c2 msg**, conforme o indicado no texto.

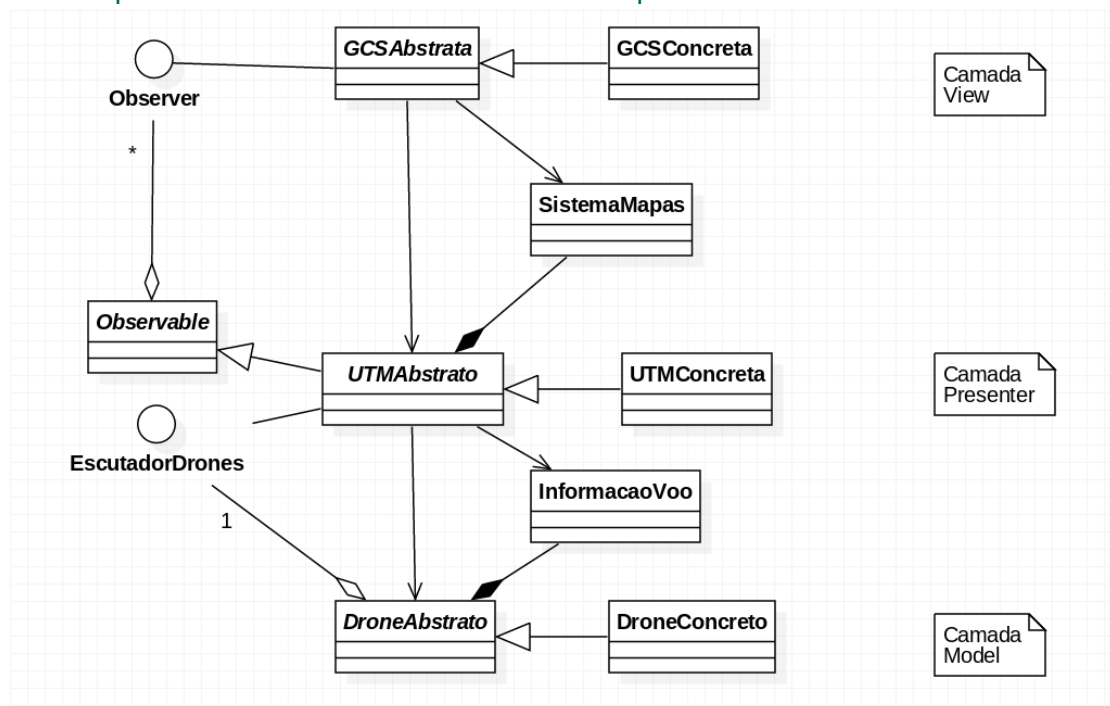
A GCS foi considerada a View, por fazer parte da camada superior, que faz a interface com o usuário.

O UTM foi considerado como Presenter, por ser responsável por conciliar a view (GCS) com o model (Drone). Ele é o middle-man.

O Drone foi considerado como Model, por ser a camada mais baixa e por armazenar e processar (simular) os dados.

Todas as camadas foram separadas seguindo o DIP (Dependency Inversion Principle), com abstrações sendo criadas entre as camadas.

As implementações concretas são fakes, no sentido de implementar apenas o mínimo para demonstrar a funcionalidade da arquitetura.



- 5) **[IMPLEMENTAÇÃO]** De posse do diagrama do exercício 4, usando linguagem Java implemente o mesmo. Construa uma classe principal (Main), que demonstre o correto funcionamento dos padrões usados em sua solução. Por exemplo, no caso do MVP deve ser apresentadas que as responsabilidades entre as entidades da arquitetura (model, view, controller) funcionam adequadamente.

- Obs1: Atente que não é necessário implementar nenhuma interface gráfica, onde na demonstração, deve-se imprimir na console, as mensagens de forma a entender o funcionamento do código gerado.
- Obs2: Para efeito do exemplo devem ser instanciados ao menos 3 instâncias de GCS, consequentemente 03 instâncias de Drones.

- c. Obs3: A solução deve tratar o problema apresentado na questão 3 (instância única do UTM-CTR).
- d. Salve todo o código gerado num pacote utm_v0.

IMPORTANTE: Não foi considerado o canal **c2 msg**, conforme o indicado no texto. Por isso o GCS não tem nenhum conhecimento dos Drones

CORREÇÃO:

- i. **[2.0 PTOS] – IMPLEMENTAR CORRETAMENTE A ARQUITETURA MVP**

A arquitetura MVP foi implementada com sucesso, de acordo com o especificado no diagrama UML da questão 4, desconsiderando-se o canal c2msg.

O model desconhece as camadas acima, tem acesso apenas a uma abstração que é o escutadorModel, para propagar eventos. O presenter só conhece o model (camada abaixo) e tem acesso a uma abstração (observer) da camada acima, para propagar eventos. A View conhece apenas o presenter, e não sabe nada do model.

- ii. **[1.0 PTOS] – IMPLEMENTAR UMA CLASSE MAIN QUE DEMONSTRE O MVP**

A classe Main foi implementada. Criando 3 instâncias de drones e 3 instância de CGS, além de utilizar o getInstance() da UTM, que é um Singleton. Observa-se as mensagens no console.

- iii. **[1.0 PTOS] – TRATAR O PROBLEMA DA QUESTÃO 5.C**

O problema foi resolvido por meio do DP Singleton.

- 6) **[IMPLEMENTAÇÃO]** Transforme seu código, de forma a substituir sua classe principal (Main) por testes projetados em JUNIT e MOCKITO.
 - a. Obs1: Lembre-se um teste, uma funcionalidade.
 - b. Obs2: Os nomes dos testes, devem lembrar o que está sendo testado.
 - c. Obs3: Em cada teste, deve-se ter um comentário (JAVADOC) do que está se fazendo.
 - d. Salve todo o código gerado num pacote utm_v1.

CORREÇÃO:

- i. **[1.0 PTOS] – USO CORRETO DO MOCKITO (NÃO USAR CLASSES FUNCIONAIS, QUANDO SE PODE MOCAR)**

Todos os testes foram de unidade, com classes mocadas.

- ii. **[2.0 PTOS] – ESTRATÉGIA CORRETA DE TESTES -> DEMONSTRAR EM TESTES SEPARADOS O MODEL, O VIEW, O CONTROLLER**

Os testes foram separados.

- iii. **[1.0 PTOS] – DEMONSTRAR EM TESTES A SOLUÇÃO DO PROBLEMA DA QUESTÃO 5.C**

A solução e o teste com o Singleton continua valendo. Não é possível instanciar mais de um objeto.