Fundação CECIERJ - **Vice Presidência de Educação Superior a Distância**

Curso de Tecnologia em Sistemas de Computação
Disciplina de Arquitetura e Projeto de Sistemas II
Gabarito – AD2 1° semestre de 2011.

Nome –

Observações:

1. Prova com consulta.

Atenção: Como a avaliação à distância é individual, caso sejam constatadas semelhanças entre provas de alunos distintos, será atribuída a nota ZERO a TODAS as provas envolvidas. As soluções para as questões podem ser buscadas por grupos de alunos, mas a redação final de cada prova tem que ser distinta. ALÉM DISSO, às questões desta AD respondidas de maneira muito semelhantes às respostas oriundas dos gabaritos já publicados de ADs de períodos anteriores, será atribuída a nota ZERO, incluindo também cópias diretas e sem sentido de tópicos dos slides das aulas.

Questão 1 [2 pontos]

Em um *sistema de ponto de venda*, deseja-se criar uma classe que representa o catálogo unificado dos produtos disponíveis em todas as lojas da rede. Para isso, esse catálogo deve ter um ponto global de acesso e ter uma única instância.

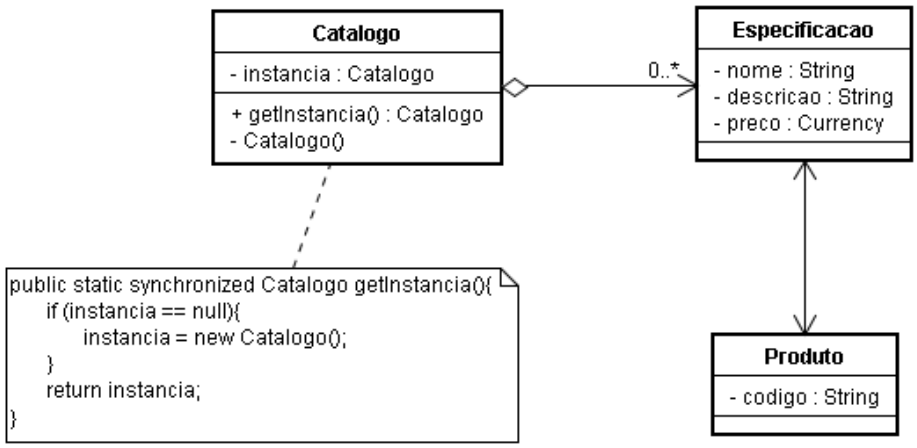
a) [1 ponto] Qual padrão de projeto deve ser utilizado nessa situação? Justifique.

b) [1 ponto] Desenhe o diagrama de classes que represente a solução com a adoção desse padrão de projeto.

Resposta:

a. O padrão de projeto *Singleton*. Sempre que é necessário ter somente uma única instância para uma classe e permitir que essa instância tenha um ponto global de acesso no sistema, o padrão *Singleton* é o indicado.

b.



Questão 2 [2 pontos]

Um dos requisitos de um sistema de locadora determina que "*os clientes da locadora podem informar o interesse em filmes que não estão disponíveis para aluguel naquele momento. Assim que os filmes estiverem disponíveis, os clientes interessados devem ser notificados*".

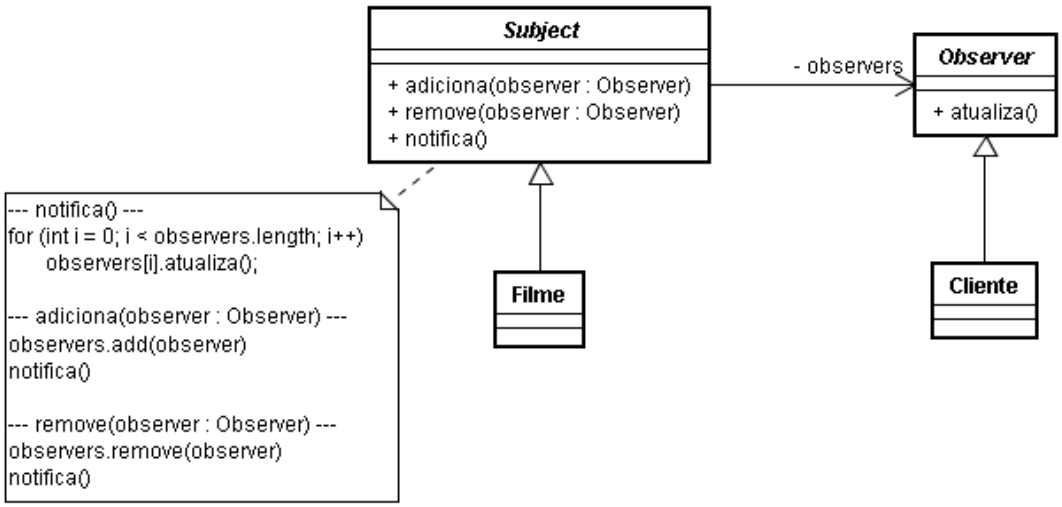
a) [1 ponto] Responda com suas palavras qual é o padrão GoF mais apropriado para auxiliar na implementação desse requisito. Justifique.

b) [1 ponto] Forneça um diagrama de classes que exiba a implementação deste requisito de acordo com o padrão GoF identificado.

Resposta:

a. Padrão *Observer*. Este padrão tem exatamente o propósito de permitir a criação de dependência de um para muitos via mecanismo de notificação de mudanças.

b.



Questão 3 [4 pontos]

No contexto do desenvolvimento de software tradicional, sabe-se que a utilização do conceito de interfaces ajuda na substituição de componentes.

a) [1 ponto] Explique o que são interfaces e quais são os seus tipos considerando um componente de cálculo de conversão de moedas.

b) [1 ponto] A partir do fato de que existem alguns problemas relacionados com a utilização do conceito de interfaces na substituição de componentes, como poderia ser feita a refatoração de um sistema que precisa ter partes substituídas, mas que não foi projetado para isso (ou seja, não usa componentes)?

c) [1 ponto] Como poderia ser feita a modificação de um serviço declarado público na interface?

d) [1 ponto] No que se refere ao escopo do projeto de um componente, o que é mais vantajoso ou interessante: *um componente com muitas funcionalidades ou com poucas funcionalidades*? Argumente a sua resposta.

Resposta:

a. Interfaces consistem em um conjunto de assinaturas de operações que podem ser invocadas por um cliente (desde que cada operação tenha sua semântica especificada) e visam determinar como o componente pode ser reutilizado e interconectado com outros componentes. As interfaces podem ser de negócio (providas ou requeridas) ou de configuração. As interfaces providas (*provided interfaces*) servem para fornecer serviços de um determinado componente para outros componentes que necessitam estes serviços. As interfaces requeridas (*required interfaces*) representam os serviços que um componente necessita para efetuar a sua função (cumprir o seu contrato) e são provenientes de outros componentes. Por fim, as interfaces de configuração (*configuration interfaces*) visam possibilitar a variabilidade do componente segundo alguns parâmetros pré-estabelecidos, que devem ser ajustados para que a funcionalidade desejada sobre o componente seja alcançada.

   Como exemplo, um componente de cálculo de conversão de moedas apresentaria: como *interface provida*, uma operação que resultasse no valor calculado após a conversão; como *interface requerida*, operações que fornecem uma tabela de conversão de valores padrão; e como *interface de configuração*, os diferentes tipos de moedas suportados pelo componente, ou formas de exibir o resultado (e.g., enviar por e-mail, imprimir na saída padrão ou exportar em algum formato).

b. Dado que as partes desse sistema não se comunicam por meio de interfaces (não foram utilizados componentes em seu projeto), a refatoração iniciaria pela identificação das partes principais do sistema e introdução de interfaces entre essas partes. A partir desse momento, cada uma dessas partes do sistema poderia ser aprimorada individualmente, testada e substituída sem que as demais partes fossem

afetadas. Em uma situação ideal, essas partes poderiam se tornar componentes, facilitando a manutenção futura.

c. Se um serviço é declarado como público na interface de uma classe, outras classes podem fazer uso do mesmo. Sendo assim, é necessário, inicialmente, comunicar aos possíveis usuários do serviço que uma modificação precisa ser feita. Nesse período, tanto a versão antiga do serviço quanto a nova devem estar disponíveis, para possibilitar uma migração gradativa dos clientes. Além disso, durante o período de migração, a operação antiga referente ao serviço deve chamar a sua nova operação, para não correr o risco de haver código duplicado, o que poderia gerar retrabalho de manutenção. Somente quando o período de migração se encerrar é que a versão antiga do serviço pode ser removida, restando somente a nova versão do mesmo.

d. Dependendo do **contexto do componente e/ou do projeto** e da **experiência do desenvolvedor e/ou organização**, por exemplo, diversas vantagens (e respectivas desvantagens justificadas sobre cada uma dessas vantagens) para cada uma das escolhas mencionadas poderiam ser apontadas. No entanto, *analisar o escopo de um componente requer uma análise de contexto e experiência*. Um componente de conversão de moeda para um sistema *web* de uma companhia aérea é bem delimitado em seu escopo, isto é, possui poucas funcionalidades (uma, no caso) e é pequeno em termos da medida de uma métrica como LoC (linhas de código). Por outro lado, um *framework* para desenvolvimento *web* como Struts ou JSF, com diversas funcionalidades e grande em termos da medida da métrica LoC, no contexto de um sistema nacional para integrar subsistemas de saúde, educação, receita, eleitoral etc., com milhões de usuários e com vários requisitos não funcionais como disponibilidade e confiabilidade, também é bem delimitado em seu escopo e, diante de outros componentes deste sistema (onde um subsistema pode ser um componente), pode parecer oferecer poucas funcionalidades e até ser pequeno em termos de própria medida mencionada, no contexto em que está inserido. Adicionalmente, um *framework* destes, no contexto do sistema *web* de uma companhia aérea, pode parecer grande em termos da medida em questão, e com várias funcionalidades. Assim, o vantajoso, ou interessante, é que o componente atenda às suas propriedades fundamentais (conforme visto em sala de aula), apresentando um escopo bem definido em seu projeto e construção.

Questão 4 [2 pontos]

Em relação ao processo de reutilização de software:

a) [1 ponto] Qual é a diferença entre “Desenvolvimento para Reutilização” e “Desenvolvimento com Reutilização”? Discuta e exemplifique com suas palavras.

b) [1 ponto] Que atividades estão envolvidas quando se desenvolve software a partir de uma biblioteca de artefatos reutilizáveis, do ponto de vista do consumidor?

Resposta:

a. O desenvolvimento *para* reutilização visa identificar, representar e implementar um conjunto de artefatos reutilizáveis, ao passo que o desenvolvimento *com* reutilização corresponde ao processo de se utilizar esses artefatos reutilizáveis para a construção de produtos de software específicos.

b. Ao desenvolver software a partir de uma biblioteca de artefatos reutilizáveis, um consumidor deve identificar, compreender, avaliar, selecionar, adaptar (se necessário) e integrar artefatos ao sistema em construção.