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 – AD1 2° semestre de 2008.

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 [1 ponto]

A partir do fato de que mudanças sempre acontecem em projetos de software, tanto com relação a requisitos, como pessoas e ambientes em que o software está inserido, cite e explique os principais tipos de manutenção que o software está sujeito a enfrentar.

R: Dentre os principais tipos de manutenção que o produto de software está sujeito, destacam-se a *manutenção adaptativa*, a *manutenção corretiva*, a *manutenção evolutiva/perfectiva* e a *manutenção preventiva*.

A manutenção adaptativa é realizada quando o produto de software precisa ser adaptado às novas tecnologias (hardware e software) implantadas no ambiente operacional ou mesmo mudanças no próprio ambiente (ex. legislação). As atividades de manutenção adaptativa têm origem em uma solicitação de melhoramento, seja pelos usuários do produto de software, seja por alterações no ambiente no qual ele está inserido.

A manutenção corretiva é realizada quando são corrigidos os defeitos não identificados durante a etapa de teste. Visa a consertar defeitos de um produto de software para que fique em conformidade com os requisitos sobre os quais foi desenvolvido. Estes defeitos

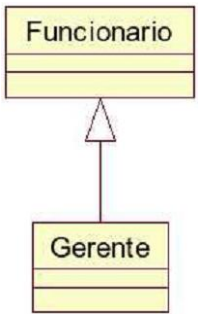
dizem respeito a quaisquer problemas com o hardware, com o software ou com a documentação.

A manutenção evolutiva/perfectiva é realizada quando o produto de software deve englobar novos requisitos ou melhorias. Isso acontece também quando se modifica o sistema, de modo a aumentar a qualidade do software ou de sua documentação, sem necessariamente modificar a sua funcionalidade. Nesta categoria, estão incluídos os esforços para aumentar a legibilidade do produto de software, para melhorar sua performance, para incrementar a reusabilidade, entre outros.

A manutenção preventiva é realizada quando o produto de software deve ser alterado para aumentar a sua manutenibilidade e/ou confiabilidade, sendo relativamente raro em ambientes de desenvolvimento, ou seja, a origem deste tipo de manutenção depende da maturidade da organização que desenvolve software. Modificações realizadas neste tipo de manutenção não afetam o comportamento funcional do produto de software.

Questão 2 [1 ponto]

Qual atitude pode ser tomada na estrutura abaixo caso não seja possível projetar as pré-condições dos métodos de `Gerente` com restrições iguais ou menores que as restrições das pré-condições dos métodos de `Funcionario`?



R: Dado que não seja possível projetar as pré-condições dos métodos de `Gerente` com restrições iguais ou menores que as restrições das pré-condições dos métodos de `Funcionário`, pode-se pensar que essa é herança questionável. Dessa forma, uma das atitudes que podem ser tomadas se refere à criação de uma nova classe (e.g., `Pessoa`), que reúna as características (atributos) e comportamentos (métodos) básicos e comuns ao pessoal empregado na organização na qual o sistema é utilizado, de forma que essa nova classe seja herdada pelas classes `Gerente` e `Funcionario`. Assim, cada subclasse definiria e implementaria os métodos que, devido aos problemas de contrato, inviabilizaram a herança em questão.

Por outro lado, observando-se as entidades envolvidas na estrutura acima, caso os métodos de `Gerente` necessitem de restrições maiores quanto às suas pré-condições, dada a herança a partir de `Funcionario`, pode ser preciso reavaliar o projeto da classe `Funcionário`: fazendo um paralelo com o domínio de aplicação que abriga essas entidades, um “gerente” tem mais privilégios do que um “funcionário” comum, inclusive

os destes. Considere um método `int calculaBonus (int avaliacao)`, em que um funcionário pode ser avaliado numa escala de 1 a 10 (espaço-estado da variável `avaliacao`), ao passo que um gerente poderia ser avaliado numa escala de -5 a 15. Caso a escala de avaliação para um gerente fosse reduzida para entre 3 e 9, provavelmente essa redução teria que ser considerada para um funcionário, visando manter a coerência e validade do domínio com relação à sua modelagem da aplicação; caso contrário, incorreria-se em problemas com contratos. Dessa forma, uma atitude a ser tomada corresponderia à verificação da classe `Funcionario` e de seus impactos na estrutura do sistema, a fim de realizar uma manutenção evolutiva e avaliar a existência de herança entre `Gerente` e `Funcionario`, caso Gerente cumpra as cláusulas contratuais.

Questão 3 [2 pontos]

Responda:

a) [1 ponto] Explique o que é um projeto por contrato e detalhe seu princípio de funcionamento. Apresente um exemplo;
b) [1 ponto] É possível definir um *framework* que apóie a utilização de invariantes, pré e pós-condição em linguagens que não dão suporte a esses recursos? Tente fazer um esboço e argumentar as vantagens e desvantagens do seu *framework*.

R: (a) Um projeto por contratos (*design by contract*) é uma abordagem utilizada para especificar as variações de espaço-estado possíveis para um método. Nesse caso, um contrato é descrito pela combinação da invariante de classe com as pré e pós-condições dos métodos, a fim de que a pré-condição seja satisfeita antes da execução do método e que a pós-condição seja satisfeita após a execução do método. Dessa forma, (i) antes da execução de cada método, a seguinte condição deve ser avaliada: `(invariantes de classe) and (pré-condições do método)`, devendo-se impedir a sua execução, caso a avaliação seja falsa, e existir uma condição de tratamento de exceções pelo sistema; e (ii) após a execução de cada método, a seguinte condição deve ser avaliada: `(invariantes de classe) and (pós-condições do método)`, de forma que o método seja reimplementado o sistema entre com uma condição de tratamento de erro caso a avaliação seja falsa. Como exemplo, pode-se recordar a criação de contrato para uma classe `Pilha`, com os atributos `itens` (coleção dos elementos da pilha) e `limite` (tamanho máximo da pilha), e para um método `pop()`: a invariante seria *a pilha tem no máximo o número de itens definidos pelo limite*, a pré-condição do método `pop()` seria *a pilha não está vazia* e a pós-condição do método `pop()` seria *o número de itens foi decrescido em uma unidade*.

(b) É possível criar um *framework* que apóie a utilização de invariantes, pré e pós-condição em linguagens que não dão suporte a esses recursos, de forma sejam definidos alguns métodos nesse *framework*, que possuam como parâmetros de entrada as invariantes e as pré e pós-condições a serem avaliadas, visando respeitar o contrato estabelecido. A classe Verificador poderia ser definida, com o objetivo de avaliar as condições do contrato de projeto das classes do sistema, relativas ao momento anterior e posterior à execução de cada método, respectivamente. Essa classe seria construída para uma dada classe do sistema, e seu construtor esperaria as invariantes da classe do sistema

como argumento. Além disso, teria dois métodos, cada um responsável por analisar uma informação do sistema referente à verificação do contrato (*strings* armazenadas na classes `PreCondicao` e `PosCondicao`, respectivamente), a qual é combinada com a invariante (definida na classe `Invariante`). Por exemplo, no caso do método `boolean avaliaAntes(PreCondicao precondicao)`, os dois conjuntos de condições que se referem às pré-condições e às invariantes do método do sistema, seriam submetidos a uma operação lógica *and* (conforme explicado na parte (a) desta questão). Para representar cada um desses conjuntos, três outras classes, `PreCondicoes`, `Invariantes` e `PosCondicoes`, são definidas, juntamente com métodos que avaliem cada uma das partes do contrato, conforme apresentado na Figura 1.

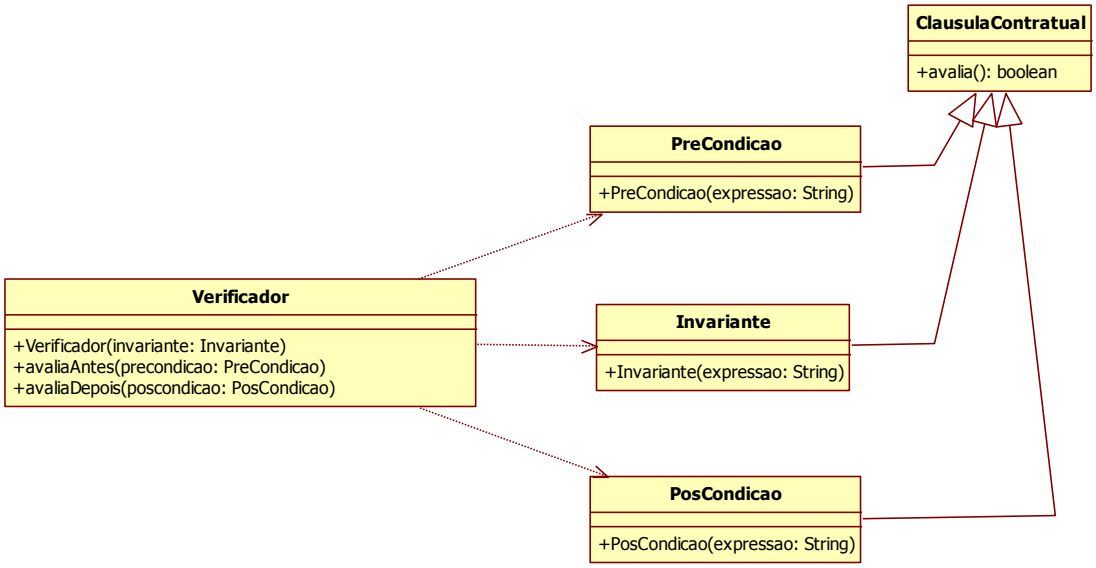


Figura 1 – Esboço do *framework* para verificação de contratos em sistemas

De maneira geral, a sintaxe para o conteúdo das variáveis `expressao` utilizaria símbolos e seria composta por dados, tais como o valor do atributo/variável a ser verificado, a expressão que representa a restrição ou condição para avaliação, o valor ou espaço-estado a partir do qual o atributo/variável deve ser comparado, sendo a sintaxe desses elementos previamente estabelecida na definição do *framework*. Dessa forma, caberia ao método avalia(), herdado de CláusulaContratual, comparar e retornar ao sistema uma resposta (do tipo *boolean*) a respeito de cada avaliação realizada, visando permitir a execução normal do sistema ou aplicar uma solução adequada, conforme explicado na parte (a) desta questão.

Uma das vantagens referentes à definição desse *framework* seria o fato de que a existência de um mecanismo genérico que verifique, para os métodos de um dado sistema, o cumprimento do contrato estabelecido, ou seja, se uma determinada condição é válida para a classe que contém esse método (invariante) e se as trocas de mensagens não ferem a confiabilidade do sistema (pré e pós-condições). Por outro lado, algumas desvantagens corresponderiam ao aumento da complexidade do sistema (e.g., novas

tecnologias e conceitos estarão presentes e demandarão maior tempo de aprendizagem), aos riscos inerentes à criação de uma dependência de mecanismos externos ao sistema (e.g., descontinuidade do projeto do *framework*) e ao maior recurso de processamento exigido pelo sistema, devido às trocas de mensagens com o *framework*.

Questão 4 [2 pontos]

Quais dimensões de estado abaixo você utilizaria para representar a classe `Retangulo` (Figura 2)? Quais são as vantagens e desvantagens da sua escolha? Existe alguma outra solução que contorna as desvantagens encontradas? Qual o espaço-estado de cada dimensão?

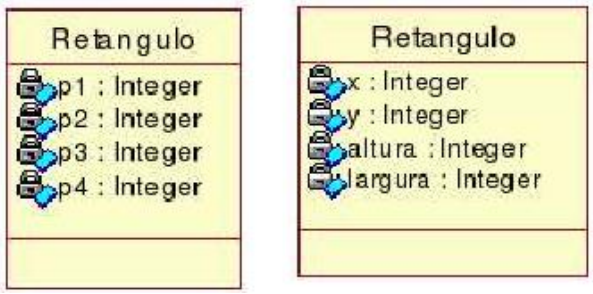


Figura 2 – Classe `Retangulo`

R: Apesar de não existir "a resposta correta", para representar a classe `Retangulo`, uma possível escolha de dimensões de estado seria a segunda opção, com os atributos *x*, *y*, *altura* e *largura*.

A vantagem estaria na melhor visualização e entendimento das propriedades de um retângulo, descrito em termos de um ponto inicial e sua largura e altura, além deste modelo suportar maior tolerância a falhas quando, por exemplo, um de seus métodos requeira a alteração da posição ou das dimensões do retângulo mediante a alteração de um de seus atributos (sem riscos de um objeto desse tipo não ser mais considerado um retângulo). Por outro lado, na primeira opção, existem quatro atributos referentes às coordenadas dos vértices do retângulo; caso ocorra uma alteração em um dos pontos, sem um devido cuidado, a classe pode deixar de representar um retângulo, ao se romper uma de suas propriedades geométricas (quatro ângulos internos de 90º e quatro lados com cada par de opostos de mesma medida).

Uma desvantagem estaria no fato de que tal modelagem suportaria apenas retângulos com pares de lados opostos paralelos aos eixos cartesianos X e Y. Considerando a primeira opção, por outro lado, não existiria esse problema, uma vez que o retângulo é descrito por seus quatro vértices, permitindo diferentes ângulos de rotação.

Uma solução para contornar a desvantagem existente na segunda opção estaria na inclusão de um novo atributo, *angulo*, referente ao ângulo de rotação dos lados correspondentes à largura do retângulo com relação ao eixo X.

Por fim, para a nova solução, o espaço-estado de cada dimensão seria *x*, *y* = [0..∞], *largura*, *altura* =]0.. ∞] e *angulo* = [0..90].

Questão 5 [2 pontos]

Em relação à Engenharia de Requisitos:

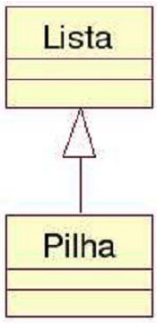
c) [1 ponto] Explique três de suas dificuldades, enfatizando possíveis impactos sobre o cliente (quando da utilização do software);
d) [1 ponto] Pesquise e discorra acerca de três técnicas de elicitação de requisitos.

R: (a) Dificuldades: (i) Comunicação com usuário: problemas de comunicação, ocasionados por ruídos no canal (meio) ou mesmo pela falta de compreensão do código utilizado (linguagem), são significativos para o entendimento dos requisitos a serem contemplados pelo projeto do sistema, o que pode resultar agregar funcionalidades erradas ou mesmo desnecessárias ao produto final; (ii) Mudanças constantes: solicitações realizadas pelos clientes são passíveis de alterações freqüentes, devido a mudanças gerenciais, de lógica de negócio, do arcabouço tecnológico, dentre outras, e isso requer uma postura flexível e dinâmica dos engenheiros de software; e (iii) Diferentes formas de representação: a falta de padronização de uso de uma forma para se representar o problema a ser resolvido e os requisitos a serem contemplados pelo sistema pode gerar distorções entre desenvolvedores e gerentes e entre gerentes e clientes, ocasionando tensões desnecessárias, por falta de planejamento.

(b) Técnicas: (i) Entrevista: é uma técnica bem simples, muito conhecida e usada, mas que deve ser conduzida de forma estruturada a fim de evitar se tornar muito informal – menos produtiva – e consistir apenas em uma conversa. Engenheiros de requisitos devem elaborar formulários com questões direcionadas para organizar as informações que o cliente fornecerá, sem perder detalhes importantes, além de buscar evitar retrabalho ou esforços futuros desnecessário; (ii) Reuniões de Brainstorming: consistem em reuniões de trocas de idéias onde várias questões são levantadas e colocadas em pauta, em um processo de várias reuniões desse tipo, começando do nível mais geral e prosseguindo-se para um detalhamento mais específico de idéias sobre a construção do sistema; e (iii) Prototipação: é uma técnica mais prática e bastante utilizada, na qual um entendimento inicial do sistema pode permitir a construção de um pequeno protótipo do sistema, a fim de exibir ao cliente o que foi assimilado *a priori* pelo engenheiro de requisitos acerca do que vai ser o sistema. Apesar da percepção positiva do cliente diante desta pequena versão do sistema, corre-se o risco de que ele entenda esse protótipo como um produto final e que se sinta completamente atendido em suas necessidades – o que deve ser evitado.

Questão 6 [1 ponto]

Você encontra algum problema no projeto abaixo? Se sim, mude a sua estrutura para que ele não tenha mais o problema detectado. Descreva métodos e atributos para a nova estrutura.



R: A implementação de herança entre `Lista` (superclasse) e `Pilha` (subclasse) tem alguns problemas: (i) existem métodos (comportamentos) de `Lista` que não se aplicam a `Pilha`, tais como `inserir(posicao : int)`; e (ii) os nomes das operações herdadas podem não ser muito corretos: o método `empilhar()` (da classe `Pilha`) não existe na classe `Lista` com esse nome, mas corresponderia ao método `inserir(0)` (no entanto, algumas linguagens, como *Eiffel*, permitem alterar o nome na herança). Dado que a herança permite o compartilhamento baseado em classes e que as características e comportamentos da classe `Pilha` se encontram na classe `Lista`, poderíamos adotar como solução a utilização da delegação, a qual permite o compartilhamento baseado em objetos. Dessa forma, este mecanismo permitiria o repasse da mensagem para o outro objeto. O objeto "delegador" do serviço, do tipo da classe `Pilha`, conteria uma referência para o objeto responsável pela execução, no caso aquele do tipo da `Lista`, podendo obter resultados dos métodos da classe `Lista` que a classe `Pilha` referencia, conforme apresentado na Figura 3. Ou seja, a implementação do método `empilhar()` conteria uma chamada para o `inserir(0)`, da classe `Lista`, e o método `desempilhar()`, uma chamada para o método `remover(0)`, a fim de se manter a disciplina que rege a estrutura de dados pilha: primeiro a chegar é o último a sair (*last in, first out*).

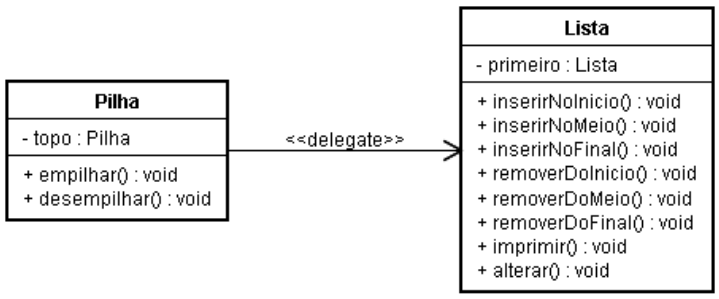


Figura 3 – Nova estrutura para o relacionamento entre as classes `Pilha` e `Lista`

Questão 7 [1 ponto]

Como é possível permitir que a classe `Noticia` (Figura 4) possa ser exibida na tela, na impressora ou no celular sem criar coesão de domínio misto e que facilite a criação de novos métodos de exibição? Tente encontrar uma solução e argumentar as vantagens e desvantagens da sua solução.

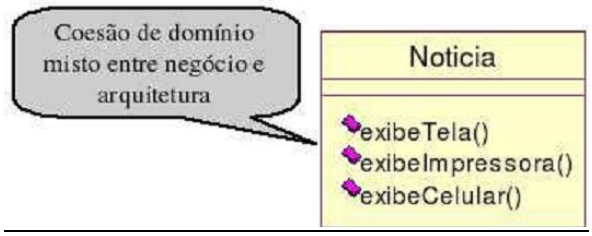


Figura 4 – Classe `Noticia`

R: É possível permitir que a classe `Noticia` possa ser exibida na tela, na impressora ou no celular sem criar coesão de domínio misto e de forma a facilitar a criação de novos métodos de exibição. Uma possível solução seria manter a classe `Noticia` no domínio de negócio, retirando dela métodos que se refiram ao domínio de arquitetura e criando um relacionamento de associação com uma nova classe abstrata `MeioDeExibicao`, a qual possuirá o método `exibir()` responsável por exibir uma notícia em algum meio de exibição disponível no sistema em questão. Esses meios seriam providos por meio de classes concretas do domínio de arquitetura, que possuem um relacionamento de herança com a classe `MeioDeExibicao` (do domínio de arquitetura) e que possuem implementações para o método `exibir()`: `Tela`, `Impressora` e `Celular`. Dessa forma, uma das vantagens é a possibilidade de exibição de notícias em diversos meios, bastando a implementação de uma nova classe que implemente a classe `MeioDeExibicao`, sem a necessidade de ser invasivo ao domínio de negócio, o que facilita a realização da evolução do sistema e a manutenção de sua organização estrutural. Uma possível desvantagem está no relacionamento direto entre as classes `Noticia` e `MeioDeExibicao`, o qual amarra uma classe de domínio de negócio a uma de domínio de arquitetura, sem a utilização de uma classe do domínio de sistema, por exemplo, que faça o relacionamento ("classe cola") e não impacte o sistema, caso esse relacionamento se torne desnecessário o futuro.

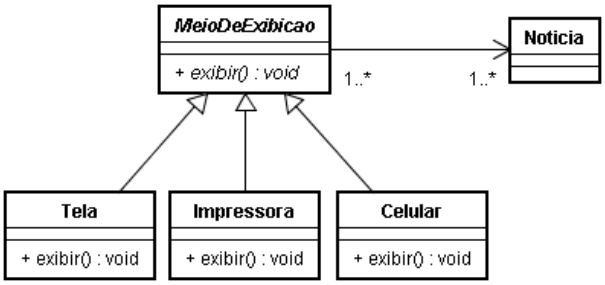


Figura 5 – Solução elaborada para o problema de coesão de domínio misto