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 1° 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.

---

Questão 1 [1 ponto]

Pesquise e discorra sobre as principais características dos seguintes processos de software:

(a) Cleanroom;
(b) RUP (Rational Unified Process);
(c) Scrum.

R: (a) Cleanroom: é uma metodologia de desenvolvimento de software considerada "pesada" pelos padrões da Engenharia de Software, mas difundida no desenvolvimento de grandes projetos corporativos. O processo é baseado no projeto apurado das funcionalidades, analisadas pelo método *peer-review* a fim de verificar se elas realizam realmente o que foram especificadas a fazer. Por analogia, podemos comparar esta metodologia com as "salas limpas" na fabricação de semicondutores, que eliminam a necessidade de se limpar *wafers* de silício pelo fato de que eles nunca começam sujos. O desenvolvimento Cleanroom minimiza a necessidade de depuração do programa, pois tenta evitar a introdução de erros no sistema. Os princípios básicos do processo Cleanroom são: (i) desenvolvimento de software baseado nos métodos formais; (ii) implementação incremental sob o controle estatístico de qualidade; e (iii) medição estatística dos testes.

(b) RUP (*Rational Unified Process*): é um processo proprietário criado pela Rational Software Corporation (adquirida pela IBM) que utiliza técnicas e práticas aprovadas comercialmente e que visa aumentar a produtividade dos projetos de software.

O RUP utiliza a orientação a objetos em sua concepção e é projetado e documentado por meio da notação UML (Unified Modeling Language) para ilustrar os processos em ação. É considerado pesado e preferencialmente aplicável a grandes equipes de desenvolvimento e a grandes projetos; porém, o fato de ser amplamente customizável torna-o passível de adaptação para projetos de qualquer escala. Para a gerência do projeto, o RUP provê uma solução disciplinada de como assinalar tarefas e responsabilidades dentro da organização sendo, por si só, um produto de software modular e automatizado. Define as seguintes linhas-mestras e esqueletos (*templates*) para os membros da equipe de um ciclo de produção: (i) gestão de requisitos; (ii) uso de arquitetura baseada em componente; (iii) uso de software de modelos visuais; (iv) verificação da qualidade do software; (v) gestão e controle de mudanças do software. Para capturar a dimensão do tempo de um projeto, o RUP o divide em quatro fases diferentes: concepção (ênfase no escopo do sistema), elaboração (ênfase na arquitetura), construção (ênfase no desenvolvimento) e transição (ênfase na implantação). As fases são objetivas, compostas por iterações (janelas de tempo com prazo definido) e geram artefatos que serão utilizados nas próximas fases e que documentam o projeto, permitindo melhor acompanhamento. Além disso, o RUP também se baseia nos 4 P's: pessoas, projeto, produto e processo.

(c) Scrum: é um processo ágil de desenvolvimento (iterativo e incremental) que pode ser aplicado a qualquer produto ou no gerenciamento e controle de qualquer atividade complexa, desenvolvido por Ken Schwaber e Mike Beedle na década de 90 baseando-se em experiências próprias no desenvolvimento de sistemas e processos. O Scrum contempla uma visão empírica baseada na teoria de controle de processos: parte do princípio que nem todas as características do produto são conhecidas na análise e que provavelmente os requisitos mudarão com o passar do tempo. Existem duas atividades principais: inspeção e adaptação. Como o processo não é definido, o gerente de projeto tem que inspecionar a execução diariamente, o que requer transparência, e fazer as adaptações necessárias com o passar do tempo. Divide-se o desenvolvimento em iterações (*sprints*) de até trinta dias. Equipes pequenas, de até dez pessoas, são formadas por projetistas, programadores, engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidades (os requisitos, em outras palavras) definidas no início de cada *sprint*. A equipe é responsável pelo desenvolvimento desta funcionalidade. Na Scrum existem reuniões de acompanhamento diárias. Nessas reuniões, que são preferencialmente de curta duração (aproximadamente quinze minutos), são discutidos pontos como o que foi feito desde a última reunião e o que precisa ser feito até a próxima. As dificuldades encontradas e os fatores de impedimento (*bottlenecks*) são identificados e resolvidos. O ciclo de vida da Scrum é baseado em três fases principais, divididas em sub-fases: pré-planejamento (*Pre-game phase*), desenvolvimento (*game phase*) e pós-planejamento (*post-game phase*).

Questão 2 [1 ponto]

Sobre conceitos de Orientação a Objetos, defina:

(a) classe;
(b) objeto;
(c) mensagens;

(d) herança;
(e) polimorfismo.

R: (a) Classe: representação computacional de entidades ou processos do mundo real. São compostas de atributos (características – informações) e métodos (comportamentos – processos) e instanciam objetos.

(b) Objeto: instanciação de uma classe. Possui um conjunto de serviços (interface) e sua implementação (estruturas de dados – atributos, e implementação de operações – métodos).

(c) Mensagens: paradigma de comunicação que visa à independência entre os objetos, uma vez que os dados de um objeto não podem ser manipulados ou vistos por outro.

(d) Herança: mecanismo que promove a reutilização de software por meio do reconhecimento da similaridade entre classes de objetos, formando uma hierarquia. Define uma relação do tipo "é um", onde uma classe compartilha a estrutura e o comportamento definidos em outras classes.

(e) Polimorfismo: propriedade derivada do fato de que objetos de diferentes classes podem reagir a uma mesma mensagem de forma diferente. Dessa forma, cada classe implementa um método específico para uma operação, possibilitando a definição de protocolos comuns.

Questão 3 [1 ponto]

Imagine que você disponha do Diagrama de Classes exibido na Figura 1, relativo ao Sistema de Gerenciamento de Hotel (SGH), e de um requisito desse sistema: *RF01 - O sistema deve calcular o valor a ser pago pelo hóspede quando este efetuar o check-out*. Faça a descrição deste caso de uso conforme o *template* da Tabela 1, considerando apenas os dados fornecidos sobre esse sistema. Além disso, modele o(s) Diagrama(s) de Seqüência da UML correspondente(s) a esse caso de uso.

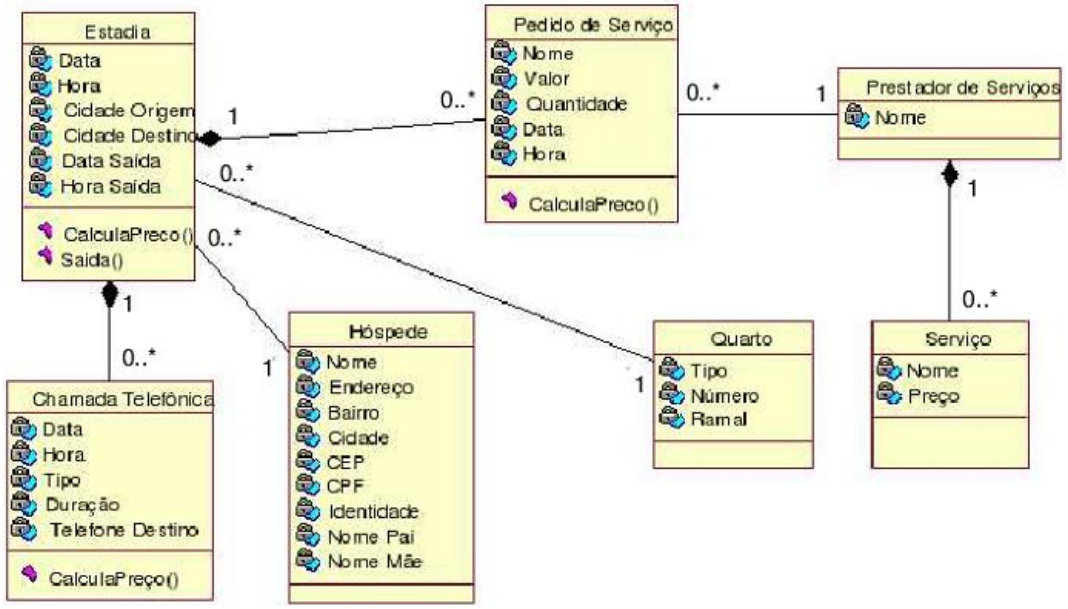



**Figura 1 – Diagrama de Classes do Sistema de Gerenciamento de Hotel (SGH)**

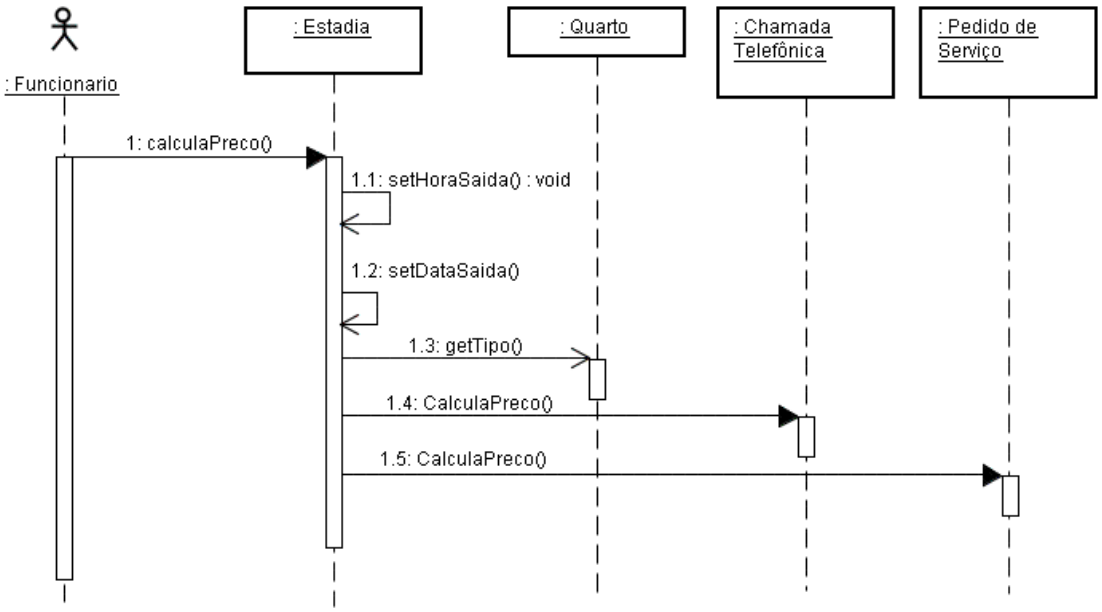
**Tabela 1 – Template para Descrição de Casos de Uso**

| | |
|---|---|
| **Nome:** | <definir o nome do caso de uso> |
| **Objetivo:** | <descrever o objetivo do caso de uso> |
| **Atores:** | <descrever os atores que interagem com o caso de uso> |
| **Pré-condições:** | <descrever as pré-condições a serem atendidas para que o caso de uso possa ser executado> |
| **Trigger:** | <definir que evento dispara a execução desse caso de uso> |
| **Fluxo Principal:** | <descrever o fluxo principal do caso de uso> |
| **Fluxo Alternativo:** | <descrever os fluxos alternativos do caso de uso, indicando que evento dispara cada um deles. Cada fluxo deve ser nomeado A1, A2, etc.> |
| **Extensões:** | <definir que extensões podem ser executadas> |
| **Pós-condições:** | <definir que produto ou resultado concreto o ator principal obterá ao final da execução do fluxo básico> |
| **Regras de negócio:** | <listar as regras de negócios que devem ser respeitadas na execução do caso de uso. Cada regra deve ser nomeada RN1, RN2, etc, e ser referenciada em algum fluxo do caso de uso (básico ou alternativo)> |

R: Para simplificar, um caso de uso "Efetuar *check-out*" inclui o caso de uso "Calcular despesas"; segue a descrição deste:

| | |
|---|---|
| **Nome:** | Calcular despesas. |
| **Objetivo:** | O sistema deve calcular o valor a ser pago pelo hóspede quando este efetuar o *check-out*. |
| **Atores:** | Funcionário. |
| **Pré-condições:** | O ator deve ter acionado o *check-out* para um cliente. |
| **Trigger:** | O ator aciona a opção calcular despesas. |
| **Fluxo Principal:** | 1. O ator aciona a opção calcular despesas.<br>2. O sistema solicita a data e a hora de saída. [A1]<br>3. O ator fornece as informações solicitadas.<br>4. O sistema efetua o cálculo do valor total a ser pago por meio do período de estadia, chamadas telefônicas e pedidos de serviço.<br>5. O sistema exibe ao ator o valor total a ser pago pelo hóspede. |
| **Fluxo Alternativo:** | [A1] Hora/data incoerentes.<br>1. O sistema exibe a mensagem "hora/data incorretos". Volta ao passo 2 do fluxo principal para repetir o processo. |
| **Extensões:** | Não há. |
| **Pós-condições:** | O valor calculado das despesas do hóspede é exibido com sucesso. |
| **Regras de negócio:** | RN01 – O cálculo das despesas totais é obtido por meio de despesas de estadia, chamadas telefônicas e pedidos de serviços. |

A seguir, o diagrama de seqüência correspondente ao caso de uso "Calcular despesas":




Questão 4 [1 ponto]

Considerando o Projeto Orientado a Objetos (POO), responda:

(a) O que é e quais são as suas características?
(b) Qual a sua importância?
(c) Qual o primeiro passo para se construir um POO?

R: (a) Projeto Orientado a Objetos (POO) consiste em um conjunto de artefatos proveniente do refinamento dos modelos gerados pela Análise OO ao realizar a descrição estática e dinâmica do sistema em diferentes níveis de abstração. Suas principais funções são atribuir responsabilidades às entidades do sistema e encontrar abstrações adequadas para representar as entidades do sistema, visando solucionar o problema descrito na análise e respeitando os requisitos funcionais e não funcionais do produto.

(b) A importância do POO reside no fato de possibilitar ao ser humano lidar com a complexidade e permitir a comunicação entre as pessoas envolvidas no desenvolvimento de software, além de apoiar a derivação de outros modelos e facilitar a codificação do sistema.

(c) O primeiro passo para se construir POO consiste na definição da arquitetura do sistema.

Questão 5 [1 ponto]

Considerando o Projeto Orientado a Objetos (POO) na UML, responda:

(a) O que é um mapa de navegação?
(b) Utilize um Diagrama de Transição de Estados da UML para modelar um mapa de navegação para a tela abaixo (Figura 2).

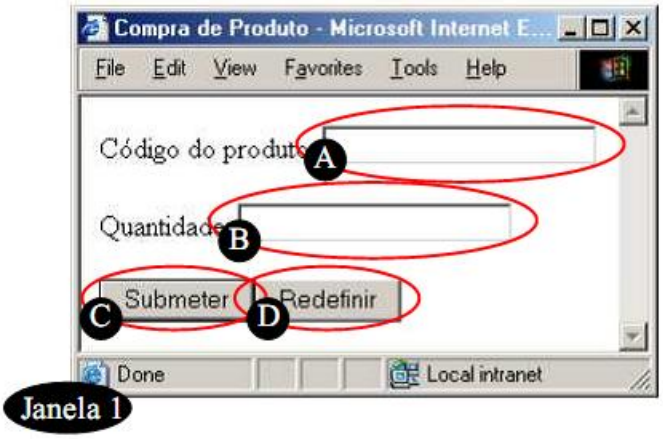
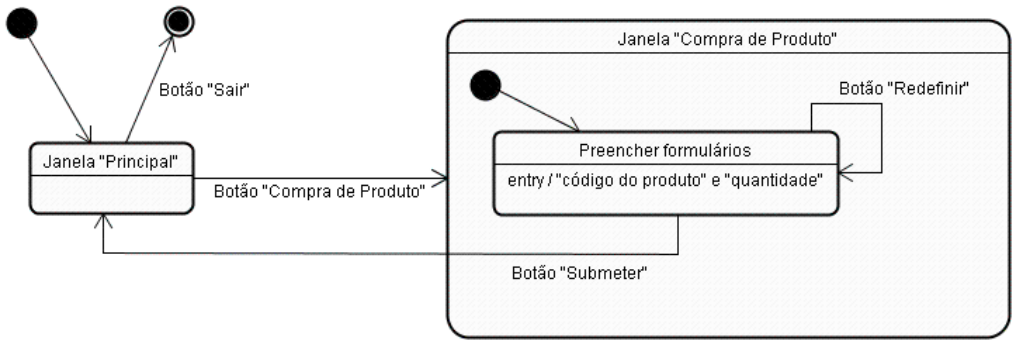



**Figura 2 – Tela Compra de Produto**

R: (a) Mapa de navegação, também conhecido como modelo de navegação, é um artefato produzido pelo Projeto Orientado a Objetos com o intuito de facilitar a visualização do sistema como um todo, uma vez que o número de maquetes de um sistema normalmente é significativo. Esse modelo pode ser construído por meio do diagrama de transição de estados da UML e utiliza nomenclatura de janelas e campos definida nas maquetes.

(b) Modelagem da tela da Figura 2 via Diagrama de Transição de Estados da UML (considerar somente o estado preencher formulários e as suas transições):




Questão 6 [1 ponto]

A classe `FaturaEnviavel` (Figura 3) tem, sem dúvida, coesão de domínio misto. Este fato representa algum problema para o sistema? Tente fornecer argumentos sobre o seu ponto de vista.

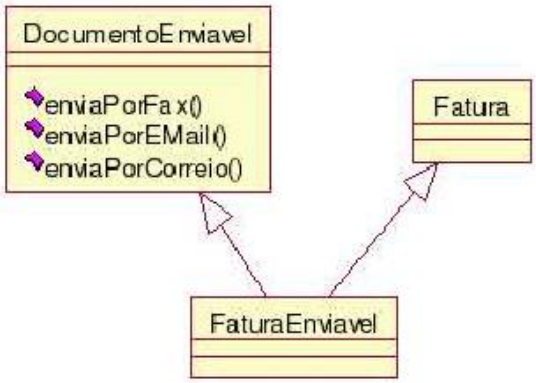
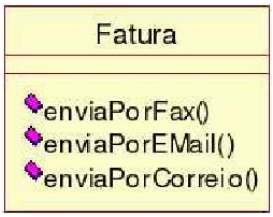



**Figura 3 – Parte do Diagrama de Classes de um subsistema de cobrança**

R: O fato da classe `FaturaEnviavel` possuir coesão de domínio misto não representa nenhum problema para o sistema, mesmo que a herança múltipla gerada não seja implementada "diretamente" por algumas linguagens de programação como em Java (mas existem formas de simula-la utilizando conceitos de interface e delegação). Na verdade, considerando a existência de quatro tipos de domínios de classes OO (aplicação, negócio, arquitetura e base), deve-se ressaltar que classes do domínio de negócio não devem ser dependentes de tecnologia. Isso aconteceria caso declarássemos uma classe `Fatura` com os métodos `enviaPorFax()`, `enviaPorEMail()` e `enviaPorCorreio()`, conforme a figura abaixo, o que faria com que tanto a classe do domínio quanto a tecnologia implementada nela fossem dificilmente reutilizáveis. Dessa forma, a criação da classe mista `FaturaEnviavel`, pertencente ao domínio de aplicação, é útil para misturar conceitos de domínios diferentes, cumprindo seu papel de classe "cola", de forma a não afetar as classes originais (classe `DocumentoEnviavel`, do domínio da arquitetura, e classe `Fatura`, do domínio do negócio).




Questão 7 [1 ponto]

Quais dimensões de estado abaixo você utilizaria para representar a classe `Retangulo` (Figura 4)? 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 4 – Classes `Retângulo`**

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 à 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 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.

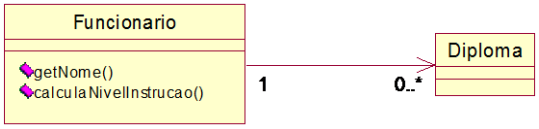
Uma solução para contornar a desvantagem existente na segunda opção estaria na inclusão de um novo atributo, *angulo*, referente ao ângulo de rotação dos lados correspondentes à largura do retângulo com relação ao eixo X.

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

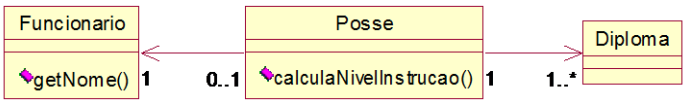
Questão 8 [1 ponto]

Cenário do problema: O sistema de controle de treinamentos de uma empresa precisa identificar, para cada funcionário da empresa, toda a sua formação até aquele momento, para poder lhe oferecer cursos do seu interesse e condizentes com o seu nível de instrução. Para isso, existe um método chamado calculaNivelInstrucao() que fornece o nível de instrução de um objeto da classe Funcionario (que é utilizada em outros sistemas da empresa) em função de objetos da classe Diploma, que representa os diplomas ou certificados obtidos pelo funcionário durante a sua carreira profissional. Analise os quatro modelos abaixo, citando para cada um deles as suas vantagens e desvantagens.

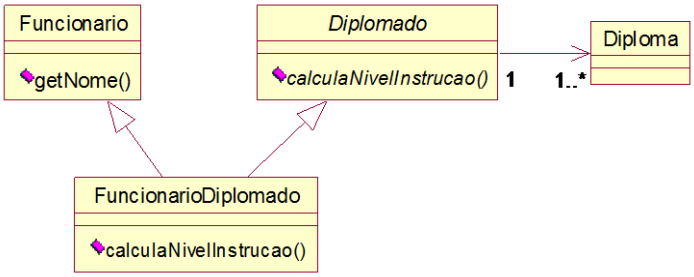
(a)



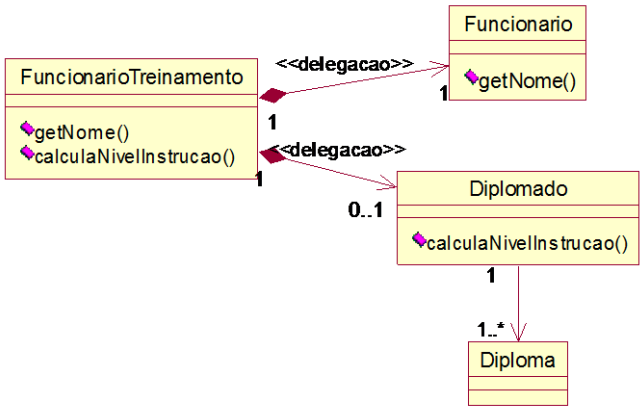

(b)




(c)




(d)




R: (a) Este modelo é bem simples e de fácil compreensão. Entretanto, insere-se na classe `Funcionario` (utilizada em outros sistemas da empresa) o método `calculaNivelInstrucao()`, que será utilizado apenas pelo sistema de controle de treinamentos, provocando acoplamento entre `Funcionario` e `Diploma` (uma classe do sistema controle de treinamentos). Dessa forma, deve-se tentar desacoplar decisões locais de decisões globais a fim de evitar comportamento intrusivo no sistema e deixar suas classes mais coesas.

(b) Neste modelo, buscou-se evitar o relacionamento direto entre as classes `Funcionario` e `Diploma` através da classe `Posse`, melhorando a questão do acoplamento indevido. Entretanto, não é possível, a partir da classe `Funcionario` obter a sua lista de diplomas `Diplomas`, pois não existe navegabilidade de `Funcionario` para `Posse`.

(c) O modelo apresentado melhora a situação de acoplamento e de relacionamento entre `Funcionario` e `Diploma` ao criar a classe `Diplomado`, que se relaciona com a classe `Diploma` por meio do método `calculaNivelInstrucao()`. A subclasse `FuncionarioDiplomado` constitui, nesse caso, o elo que contém funcionários que são diplomados. Um possível problema consiste na herança múltipla, quando ocorrer a tradução desse modelo para o modelo de implementação, devendo-se ocorrer ajustes dada a linguagem de programação escolhida (como em Java, por exemplo).

(d) O quarto modelo oferece boas características de projeto OO, uma vez que é criada uma classe `FuncionarioTreinamento` que representa um perfil (papel), composto pelas classes `Funcionario` e `Diplomado`. A herança permite o compartilhamento baseado em classes, enquanto que a delegação permite o compartilhamento baseado em objetos. Dessa forma, este mecanismo permite o repasse da mensagem para o outro objeto, uma vez que o objeto "delegador" do serviço, no caso aqueles da classe `FuncionarioTreinamento`, contém uma referência para o objeto responsável pela execução, no caso aqueles das classes `Funcionario` e `Diploma`, podendo obter resultados pelos dois métodos que a nova classe referencia. Um aspecto negativo desse modelo é o aumento de complexidade da solução. Contudo, essa complexidade extra pode ser compensada pelos ganhos de uma solução mais coesa e menos acoplada.

Questão 9 [1 ponto]

De acordo com os princípios de Projeto OO, você considera aceitável a criação de pré e pós-condições em métodos abstratos de uma classe? E quanto aos métodos de uma interface?

R: Sim, uma vez que a criação de pré e pós-condições em métodos abstratos de uma classe representa a sua documentação e qualquer classe que implemente esse método também deve respeitar a essa documentação, mantendo-se uma padronização na especificação das variações de espaço-estado possíveis para o método. Da mesma forma, essa padronização é entendida como aceitável para métodos de uma interface, garantindo, dessa forma, a metodologia do projeto por contrato.

Questão 10 [1 ponto]

Selecione uma heurística de cada tipo (classes e objetos, topologias, relacionamentos, herança e projeto OO físico) e explique com suas palavras a contribuição provida por cada heurística selecionada.

R: Heurística sobre classes e objetos: "Uma classe deve representar um e somente um elemento chave de abstração". A contribuição desta heurística se refere à manutenção do conceito de coesão em classe: uma classe que extrapole suas responsabilidades ou que seja pouco significativa para o domínio do sistema se torna vaga e, consequentemente, não retratam de forma realista os elementos do domínio, podendo acarretar em problemas futuros em manutenção do sistema.

Heurística sobre topologias: “Em sistemas com interface com usuário, as classes de modelo não devem depender das classes de interface”. Essa heurística contribui ao alertar para a questão de acoplamento entre camadas distintas do sistema e à geração de dependências entre elas (sobretudo quando classes de modelo dependem de classes de interface). Quando isso não acontece, modificações em uma camada tendem a não afetar modificações em outra camada, aumentando a característica de reutilização de partes do sistema.

Heurística sobre relacionamentos: “A maioria dos métodos de uma classe deve fazer uso da maioria dos atributos na maior parte do tempo”. Isso é importante para que a coesão de uma classe seja mantida, o que impacta diretamente na existência de um bom projeto que facilita o entendimento de suas entidades componentes. Além disso, isso significa que a classe é estruturada de forma a não necessitar divisão.

Heurística sobre herança: “As superclasses não devem conhecer as suas subclasses”. Isso mantém as hierarquias de herança do sistema organizadas, evitando que, com a criação de novas subclasses, a superclasse seja alterada, prejudicando o projeto do sistema.

Heurística sobre projeto OO físico: “Questões de projeto físico podem servir como o diferencial para tomada de decisões”. Esse fator contribui para que seja almejada a simplicidade de projetos de sistema, pois levanta aspectos como custo e simplicidade durante o processo de tomada de decisão acerca de projetos equivalentes.