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 2010.

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]

Ao refletir sobre o fato de que o processo de desenvolvimento de software requer respostas rápidas e flexíveis diante das constantes mudanças às quais ele é submetido no decorrer dos projetos realizados, observa-se que as metodologias ágeis emergiram e ganharam vários adeptos na década atual. Nesse sentido, escolha e explique uma das metodologias ágeis, apresentando os principais termos técnicos envolvidos e delineando os seus princípios e práticas.

Resposta:

O *Scrum* é uma metodologia[1] ágil para gestão e planejamento de projetos de desenvolvimento de software, desenvolvida por Ken Schwaber e Mike Beedle na década de 1990, com base em experiências na construção de sistemas e processos, a partir do reconhecimento de que o desenvolvimento de software é muito complexo para ser

[1] Existe uma confusão entre os termos *método*, *metodologia* e *processo* de desenvolvimento de software devido à inexistência de um consenso universal, o que se percebe pela divergência entre as definições apresentada por diferentes dicionários. Conforme o Dicionário Priberam da Língua Portuguesa, *método* é o processo racional que se segue para chegar a um fim, *metodologia* é a subdivisão da lógica que estuda os métodos técnicos e científicos e *processo* é a maneira de operar, de agir. Neste artigo, considera-se o *Scrum* como uma metodologia ágil para facilitar a gestão de projetos de desenvolvimento de software.

planejado corretamente desde o início. O *Scrum* contempla uma visão empírica baseada em aspectos teóricos de controle de processos, ou seja, parte do pressuposto de que nem todas as características do produto são conhecidas na análise e que os requisitos mudarão com o passar do tempo. Contempla, ainda, 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.

O desenvolvimento é dividido em iterações (*sprints*), de até trinta dias, e equipes pequenas, de até dez pessoas, reunindo projetistas, programadores, engenheiros e gerentes de qualidade – a lista de tarefas que o time (*team*) se compromete a fazer em um *sprint* é chamada de *sprint backlog*. As equipes trabalham em cima de funcionalidades (i.e., requisitos) definidas no início de cada *sprint* e cada equipe fica responsável pelo desenvolvimento destas funcionalidades, isto é, dos *product backlog items* (conjunto de requisitos priorizados, na forma de uma lista de itens, que devem ser desenvolvidos para o produto). Visando congregar a equipe, existem reuniões de acompanhamento diárias (*daily meetings*), preferencialmente de curta duração (aproximadamente quinze minutos), nas quais são discutidos três pontos: o que foi feito desde a última reunião (dia anterior), quais as dificuldades e os fatores de impedimento (*bottlenecks*) e o que precisa ser priorizado no dia que se inicia. Existem também as chamadas *retrospective meetings*, que consistem em reuniões realizadas ao final de cada *sprint* com o objetivo de relatar os pontos fortes e fracos deste *sprint*, bem como definir ações para manter os pontos fortes e eliminar ou reduzir os pontos fracos relatados. Nessa lista de reuniões, pode-se incluir também a *review meeting*, na qual o time mostra o que foi alcançado durante o *sprint*, tipicamente na forma de *demos* das novas funcionalidades. Por fim, a *sprint planning meeting* é uma reunião na qual todos os *stakeholders* envolvidos estão presentes para planejar a próxima *sprint*. O ciclo de vida do *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*).

Processos empíricos aceitam as falhas como conseqüência natural da produção e tentam torná-las visíveis e passíveis de correção, o mais cedo possível, para que a abordagem empírica se torne ideal para o ambiente de desenvolvimento de software. Nesses ambientes, as funcionalidades consideradas "terminadas" muitas vezes não estão suficientemente testadas e aceitas pelo cliente, fazendo-se necessárias atividades de inspeção e de adaptação. Além disso, deve-se considerar a existência de diferentes papéis (agentes independentes) no *Scrum*, com o intuito de atacar a complexidade do desenvolvimento e gerenciamento de projetos de software a partir da implantação de um controle descentralizado, capaz de lidar com contextos pouco previsíveis e de forma mais eficiente: (i) *product owner* (cliente): responsável pelo gerenciamento do retorno sobre o investimento do projeto, verificando se o produto entregue atende aos requisitos do projeto e priorizando as funcionalidades que devem ser entregues e aquelas de maior valor ao projeto; (ii) *Scrum team* (time): *stakeholders* envolvidos no desenvolvimento do projeto (e.g., analistas, desenvolvedores etc.), que almejam objetivos estabelecidos; (iii) *Scrum master*: responsável por guiar o time e agir como *broker*, intermediando

negociações entre o *product owner* e a equipe do projeto, além de ensinar e acompanhar a utilização do *Scrum*.

O *Scrum* realiza uma distinção entre os *stakeholders* responsáveis (*pigs*) e aqueles que estão apenas envolvidos, tais como clientes e executivos (*chickens*). Isso é importante para a visibilidade das responsabilidades sobre o projeto (e.g., da mesma forma que são atribuídas as responsabilidades para os *pigs*, o *Scrum* estabelece que todos os *chickens* não "perturbem" o time). Assim, todo e qualquer interesse de *stakeholders* não vinculados à equipe responsável é filtrado e levado em consideração apenas pelo *product owner* e pelo *Scrum master*, deixando o time livre para prosseguir com o projeto[2].

Questão 2 [7 pontos]

Considere a descrição simplificada do sistema GPT (Gestão de Passeios Turísticos)[3]:

> *O GPT gerencia os passeios turísticos de uma agência de turismo. Os passeios são de diversos tipos e variam na duração e nos pontos visitados. Para cada passeio, um veículo, um motorista e um guia são alocados e há um número máximo de participantes, dependendo do tipo de veículo alocado. O guia recebe o programa com o roteiro e a lista de participantes, antes do início do passeio. A reserva pode ser feita através do hotel, onde o turista estiver hospedado, ou pode ser feita particularmente; neste caso, o participante deve aguardar o veículo em um hotel a ser combinado com a agência. A reserva pode ser cancelada quando houver desistência e o cancelamento é aceito enquanto não for emitido o programa. Ao fazer a reserva, o interessado deve informar, além do nome e número do documento (RG ou passaporte), o telefone de contato e o hotel onde está hospedado ou onde estará aguardando. O pagamento é feito na reserva e, no caso da desistência no prazo, o valor é devolvido ao cliente.*
>
> *O GPT deve prever a emissão de relatórios gerenciais (relatório com passeios realizados no dia, com os dados relevantes). Os pontos turísticos são cadastrados, através da sua identificação, localização, descrição sucinta e duração da visita. Normalmente, os roteiros se encontram montados, mas pode-se elaborar um roteiro especial, se for solicitado por algum cliente. A elaboração de roteiros especiais e a alocação de recursos a um passeio são realizadas por um funcionário especialista em turismo. O GPT deve registrar as principais transações ocorridas. A alocação de recursos aos passeios não é feita automaticamente; no entanto, GPT deve fornecer os dados que dêem suporte a esta atividade. Na agência, devem ser instalados quiosques, onde os clientes podem consultar os dados dos passeios.*

a) [1 ponto] Desenhe um diagrama de casos de uso que satisfaça à esta descrição;

---

[2] Mais detalhes e referências podem ser obtidos em: SANTOS JUNIOR, A., SANTOS, R., 2009, "Aspectos Sociotécnicos do Desenvolvimento de Software Utilizando Scrum em um Caso Prático". In: *Anais do V Workshop Um Olhar Sociotécnico sobre a Engenharia de Software*, VIII Simpósio Brasileiro de Qualidade de Software, pp. 38-49, Ouro Preto, MG, Brasil.

[3] Fonte: SANTOS, R., COSTA, H., RESENDE, A., 2007, "Modelagem de Software usando UML", In: *Anais da VI Escola Regional de Informática de Minas Gerais*, *Mini-cursos*, pp. 1-30, Lavras, MG, Brasil.

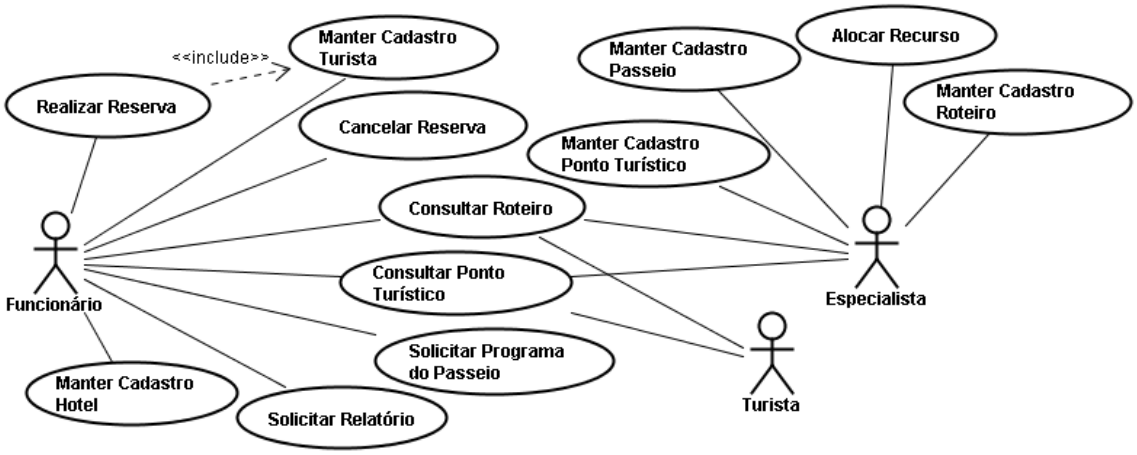
b) [1 ponto] Escolha um dos caso de uso identificados e realize a sua descrição, conforme o *template* da Tabela 1;
c) [2 pontos] A partir do caso de uso escolhido, construa o(s) Diagrama(s) de Seqüência e o(s) Diagrama(s) de Transição Estado correspondentes;
d) [1 ponto] Construa o Modelo de Classes conceitual (artefato resultante da fase *Análise Orientada a Objetos*), mantendo-o coerente com as entidades utilizadas.
e) [2 pontos] Explique detalhadamente o que grau de dependência, bem como suas variações, e relacione este conceito com os tipos de domínios de classes de um sistema. Além disso, calcule o grau de dependência de cada uma das classes apresentadas no Diagrama de Classes conceitual construído em (d).

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)>

Resposta:

a) Diagrama de Casos de Uso do GPT:

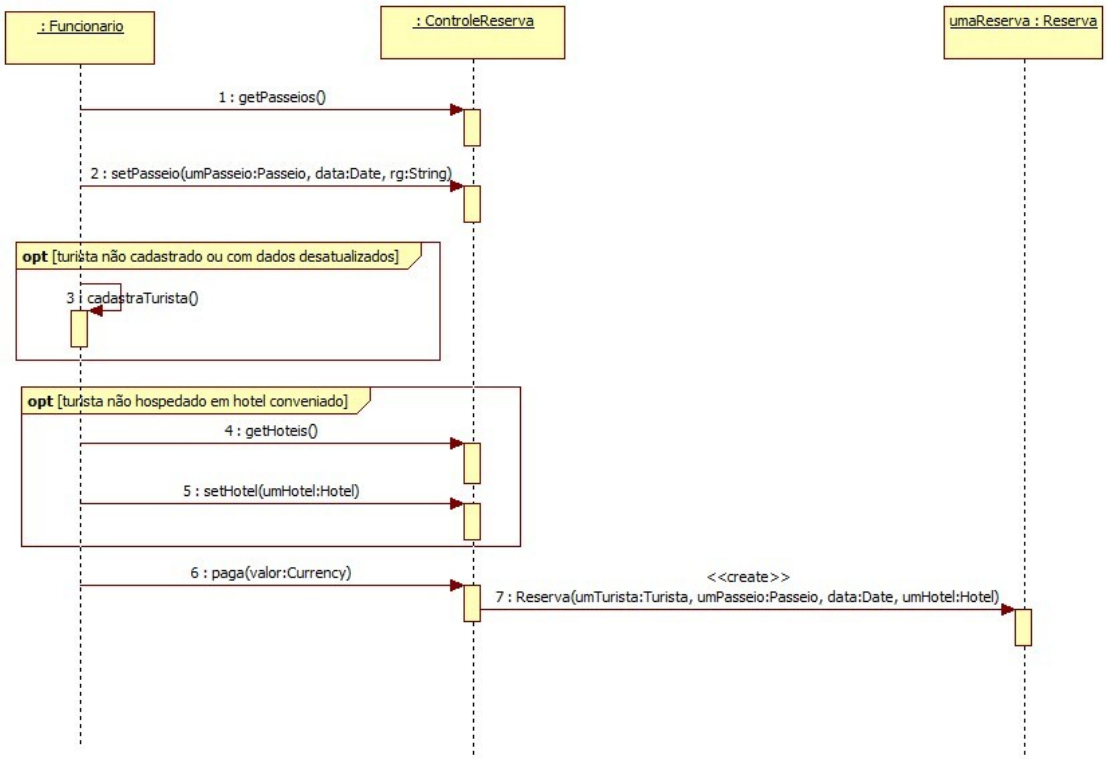


b) Descrição do Caso de Uso *Realizar Reserva*:

Nome:	Realizar Reserva
Objetivo:	O funcionário realiza a reserva solicitada pelo turista em um dos passeios disponíveis. O turista deve estar hospedado em um hotel ou combinar um hotel para aguardar o veículo do passeio. No ato da reserva, o funcionário deve solicitar as informações cadastrais do turista e solicitar o pagamento da reserva. O funcionário deve confirmar a reserva do turista ou informá-lo da impossibilidade de realizá-la.
Atores:	Funcionário
Pré-condições:	O passeio solicitado pelo turista deve estar previamente cadastrado no sistema.
***Trigger*:**	Um turista solicita um passeio.
Fluxo Principal:	1. O funcionário solicita os passeios disponíveis. 2. O sistema lista os passeios disponíveis com os respectivos preços. 3. O funcionário seleciona um passeio em uma data. 4. O sistema solicita os dados do turista. 5. O funcionário informa número do documento (RG ou passaporte) do turista. 6. O sistema exibe o cadastro do turista. 7. O sistema solicita confirmação do pagamento do passeio. 8. O funcionário confirma o pagamento do passeio. 9. O sistema registra a reserva do passeio.
Fluxo Alternativo:	[5a] O turista não está cadastrado no sistema. 1. O caso de uso “Manter Cadastro Turista” é acionado. 2. O fluxo retorna ao passo 6. [6a] O cadastro do turista está desatualizado. 1. O funcionário solicita a alteração do cadastro do turista. 2. O caso de uso “Manter Cadastro Turista” é acionado. 3. O fluxo retorna ao passo 7. [6b] O turista não está hospedado em um hotel. 1. O funcionário solicita a seleção de hotel. 2. O sistema exibe a lista de hotéis cadastrados. 3. O funcionário escolhe o hotel mais próximo da localidade desejada do turista 4. O fluxo retorna ao passo 7.
Extensões:	---
Pós-condições:	O turista tem a reserva para um passeio.
Regras de negócio:	RN01 – O funcionário deve ser capaz de realizar uma reserva em um passeio para um turista.

c) Diagramas do GPT relativos ao Caso de Uso *Realizar Reserva*:

a. Seqüência:



b. Transição de Estados:

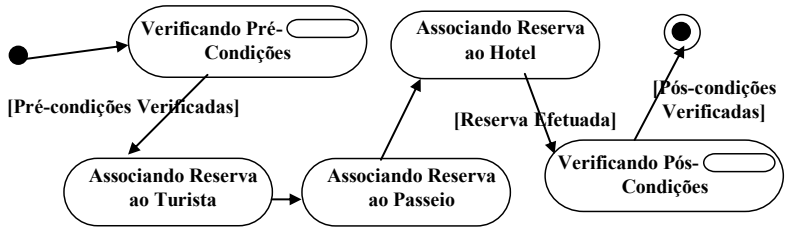


DIAGRAMA PRÉ-CONDIÇÕES

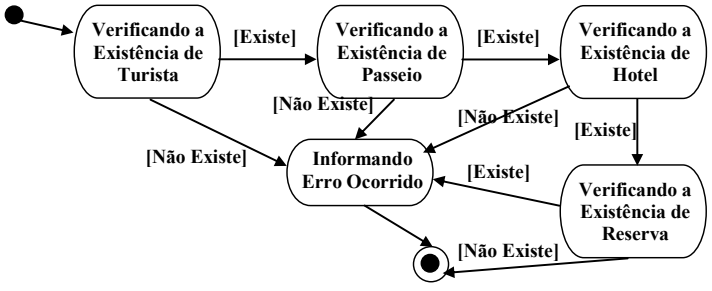
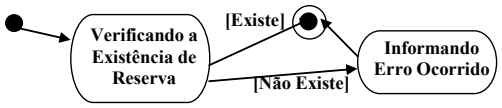
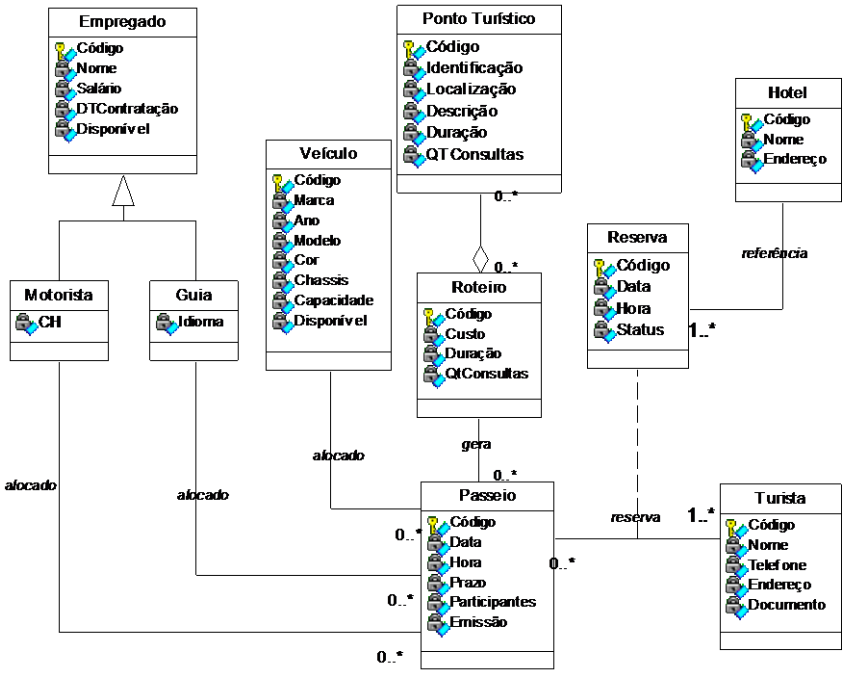


DIAGRAMA PÓS-CONDIÇÕES



d) Modelo de Classes Conceitual do GPT:



e) Grau de Dependência (GD) é uma métrica semelhante a *Fan-out*[4] de projeto estruturado e serve para verificar projetos orientados a objetos. Pode ser classificado em dois tipos: (i) *grau de dependência direto*, que indica quantas classes são referenciadas diretamente por uma determinada classe; e (iii) *grau de dependência indireto*, que indica quantas classes são referenciadas direta ou indiretamente (recursivamente) por uma determinada classe. Para identificar o grau de dependência, deve-se verificar se uma classe *A* referencia diretamente outra classe *B*, de maneira que *A* seja subclasse (ou tenha atributo, ou tenha parâmetro do método do tipo, ou tenha variáveis em métodos do tipo, ou ainda chame métodos que retornem valores do tipo) de *B*. Analisando os diferentes tipos de domínios de classes de um sistema, temos que classes de domínios mais altos (negócio e aplicação) tendem a ter alto grau de dependência indireto e as classes de domínios mais baixos (arquitetura e base) tendem a ter baixo grau de dependência indireto. Cálculo do grau de dependência para cada uma das classes:

CLASSES	GD Direto	GD Indireto
Empregado	0	0
Motorista	2	9
Guia	2	9
Passeio	5	9
Veículo	1	9
Roteiro	2	9

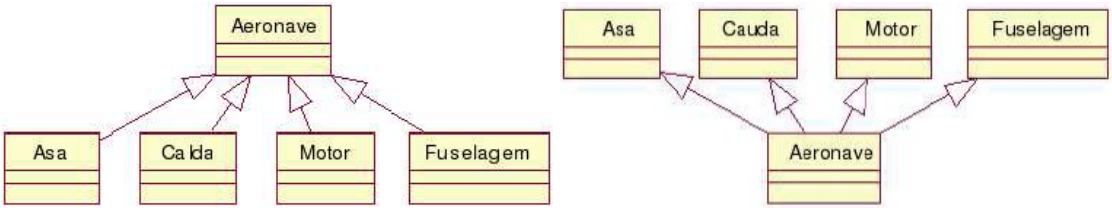
[4] Indica quantos módulos são acessados por um dado módulo.

Ponto Turístico	0	0
Turista	1	9
Reserva[5]	3	9
Hotel	1	9

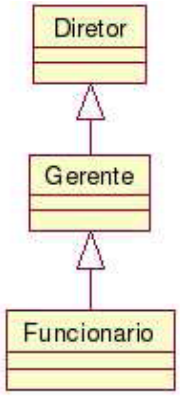
Questão 3 [2 pontos]

Para cada um dos modelos do projeto orientado a objetos desenhados abaixo, explique qual é o perigo detectado e associe-o diretamente com a heurística que apoia o projetista a evitá-lo:

a) [1 ponto]



b) [1 ponto]



Resposta:

a) *Perigo detectado nos modelos*: projetistas utilizam equivocadamente herança para mostrar que os sistemas são orientados a objeto (OO), confundindo os conceitos de herança e composição (que seria o correto, neste caso).
Heurística que apóia o projetista a evitar este perigo: "Faça perguntas à estrutura para verificar a sua corretude". *Raciocínio*: a subclasse "é um" tipo especial da superclasse? – ex.: a classe `Aeronave` não deve herdar da classe `Fuselagem`.

[5] Durante o *design*, uma classe de associação se torna uma classe entreposta entre as duas classes antes diretamente associadas, sendo estabelecidas associações entre a classe de associação e as outras duas classes com as devidas multiplicidades, ou seja, uma associação entre `Passeio` e `Reserva`, e outra entre `Reserva` e `Turista`.

b) *Perigo detectado no modelo*: confusão entre estrutura hierárquica organizacional e hierarquia de classes OO (no caso, as classes `Gerente` e `Diretor` herdariam da classe `Funcionario`).
Heurística que apóia o projetista a evitar este perigo: "Faça perguntas à estrutura para verificar a sua corretude". *Raciocínio*: a subclasse "é um" tipo especial da superclasse? – ex.: a classe `Funcionario` não deve herdar da classe `Gerente` e nem da classe `Diretor`, pois isso corresponderia a estender características e comportamentos destas duas classes para a classe `Funcionario`, o que não está de acordo com o que realmente o modelo deve representar. Ou seja, o correto é que um gerente e um diretor "são" funcionários.