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 da AD2 – 2° semestre de 2014.

Nome:
Polo:
Matrícula:

Observações:

1. Prova com consulta.

LER ATENTAMENTE AS INSTRUÇÕES A SEGUIR:

1. As respostas devem ser enviadas em um **único arquivo em formato exclusivamente .PDF, não compactado**. Além disso, o conteúdo deste arquivo deve **seguir exatamente o *template* das respostas**, caso exista. Caso não atenda a estes pontos, a **AD não será corrigida**. ADs enviadas no MODO RASCUNHO também **não serão corrigidas**. ADs MANUSCRITAS ou ESCANEADAS também **não serão corrigidas**.
2. 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.
3. Além disso, às questões desta AD respondidas de maneira muito semelhantes às respostas oriundas dos gabaritos já publicados de ADs e APs de períodos anteriores, **será atribuída a nota ZERO**, incluindo também cópias diretas, indiretas (semelhanças/paráfrases) ou sem sentido de tópicos dos slides das aulas. A AD é uma atividade de pesquisa (trabalho da disciplina) e deve ser elaborada como tal, não se atendo somente ao conteúdo dos slides das aulas.
4. Por fim, a pesquisa na Internet e em livros é estimulada, devendo ser referenciada na AD, mas as respostas devem ser construídas com as palavras do próprio aluno e atender diretamente ao que pede à questão, evitando respostas prolixas ou extensas. Às respostas copiadas ou semelhantes a soluções da Internet ou de livros, e/ou que não atendem (fora do escopo) ou excedem demasiadamente ao que pede a questão, **será atribuída a nota ZERO**.

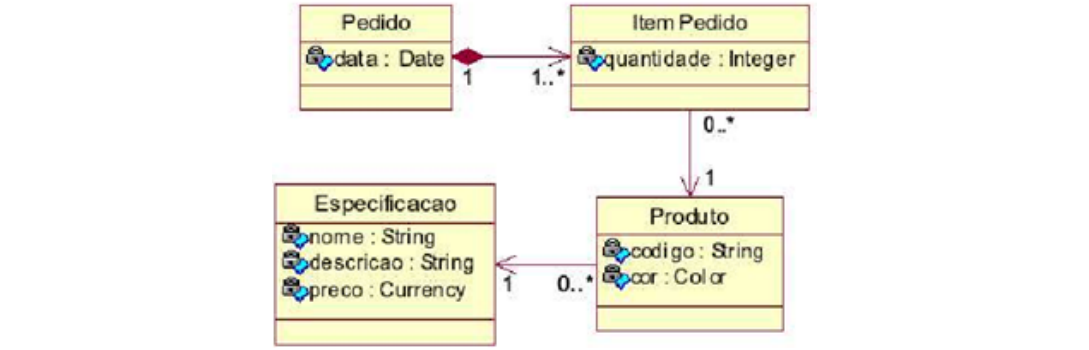
Questão 1 [2.5 pontos]

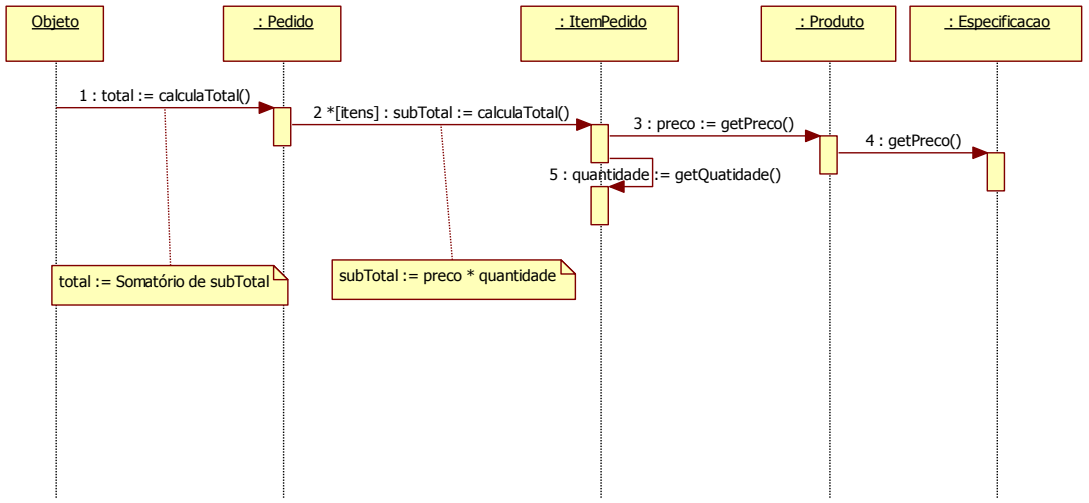
O padrão *Information Expert* visa atribuir a responsabilidade ao especialista, isto é, à classe que tem a informação necessária para satisfazer a responsabilidade.

a) [1 ponto] Descreva um exemplo, **diferente daquele visto em aula ou nos gabaritos das ADs passadas,** de uma situação onde esse padrão é útil.
b) [0.5 pontos] Quais são os benefícios alcançados com o uso desse padrão?
c) [1 ponto] Mostre como o padrão pode ser utilizado na situação descrita no item (a) através de um modelo de classes e um diagrama de sequência.
d)

Resposta:

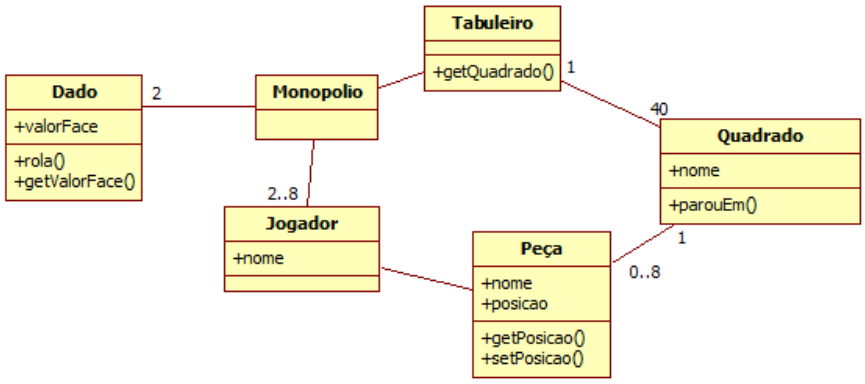
a) Em um sistema de PDV, o responsável pelo cálculo do total de um pedido deveria ser uma classe `Pedido`, considerando o padrão *Information Expert*, uma vez que deve-se atribuir a responsabilidade ao especialista, ou seja, à classe que tem a informação necessária para satisfazer a responsabilidade (no caso, o cálculo do total do pedido).
b) Entre os benefícios deste padrão, estão: (i) leva a projetos onde o objeto de software faz o que o objeto real faria; (ii) mantém o encapsulamento e não aumenta o acoplamento, pois utiliza informações próprias; e (iii) distribui o comportamento entre as classes do sistema, aumentando a coesão das mesmas.
c) Para o diagrama de classes apresentado abaixo, a aplicação do padrão supracitado para o cálculo do total de um pedido em um sistema de PDV pode ser visualizado pelo diagrama de sequência abaixo:





Questão 2 [5.0 pontos]

Baseando no jogo de tabuleiro *Monopólio/Banco imobiliário*, cujo diagrama de classe é apresentado abaixo, faça:



a) [1.5 pontos] Aplique o padrão de projeto *Polymorphism* para permitir diversos tipos de *quadrados* (vulgarmente conhecido como *casas*) no jogo. Faça um diagrama de sequencia para dois tipos diferentes de quadrados do jogo.

b) [0.5 pontos] Implemente o padrão de projeto *Creator* no *Monopólio*.

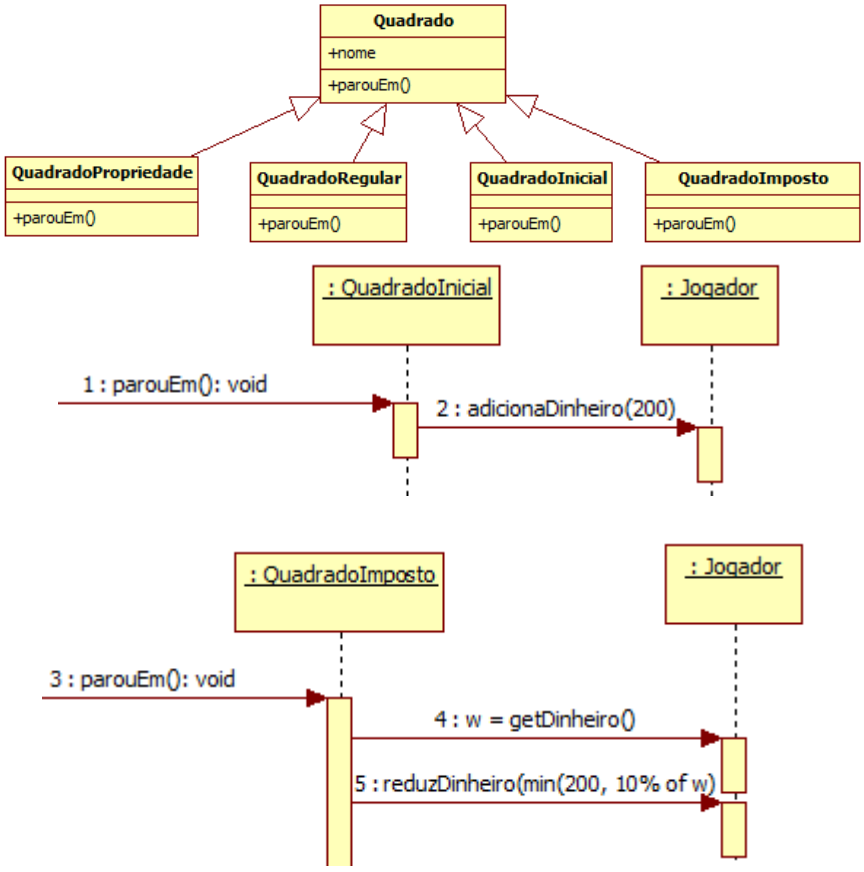
c) [0.5 pontos] Suponha que objetos precisam ser capazes de referenciar um *Quadrado* em particular, dado o seu nome. Qual classe consegue identificar o objeto *Quadrado* através do nome? Justifique citando um padrão GRASP.

d) [2.0 pontos] Desenhe o diagrama de sequencias que ilustre um turno de um jogador.

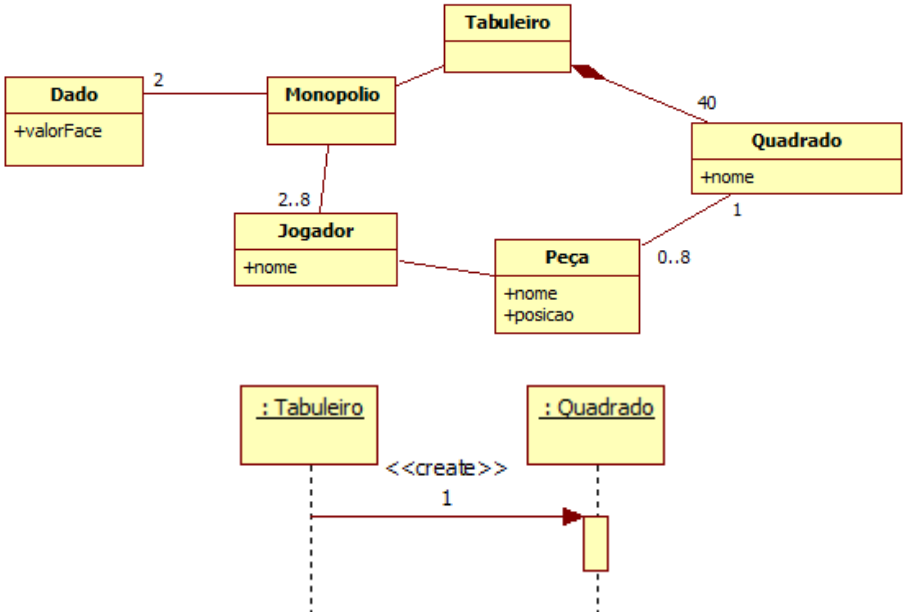
e) [0.5 pontos] Desenhe o diagrama de sequencias que ilustre o loop do jogo (*game loop*).

Resposta:

a.

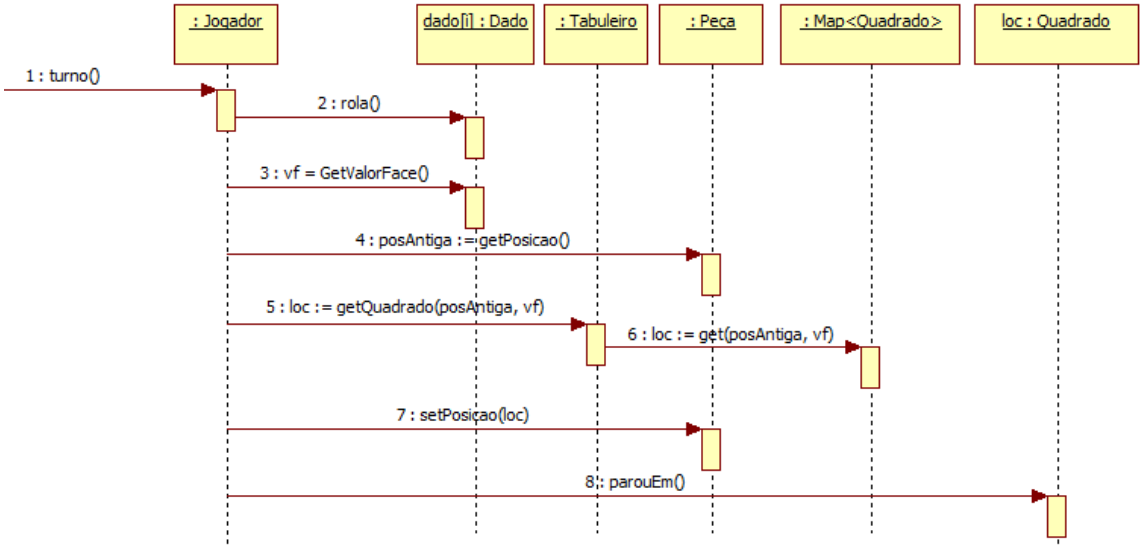


b.

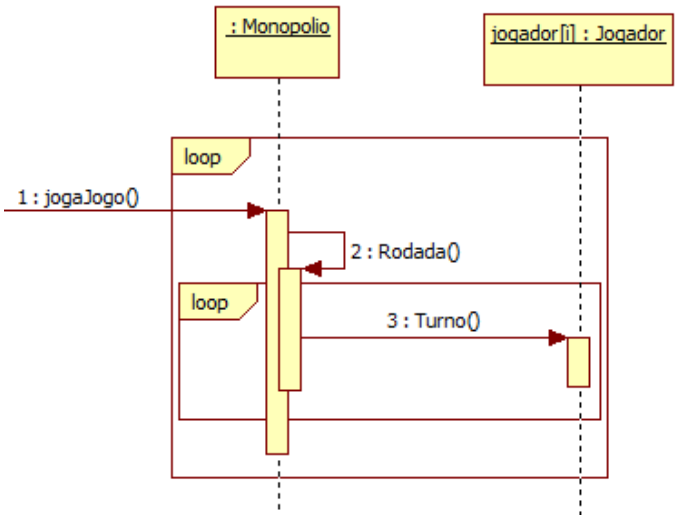


c. Como a classe Tabuleiro agrega todos os quadrados, então pelo padrão *Information Expert*, a classe Tabuleiro possui as informações necessárias para cumprir essa responsabilidade.

d.

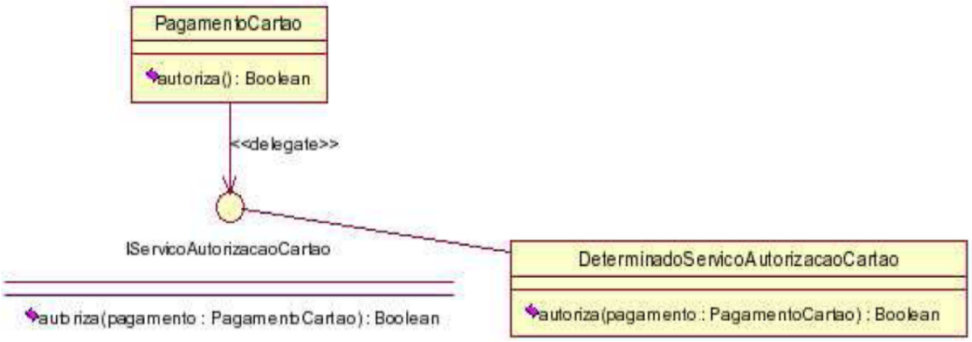


e.

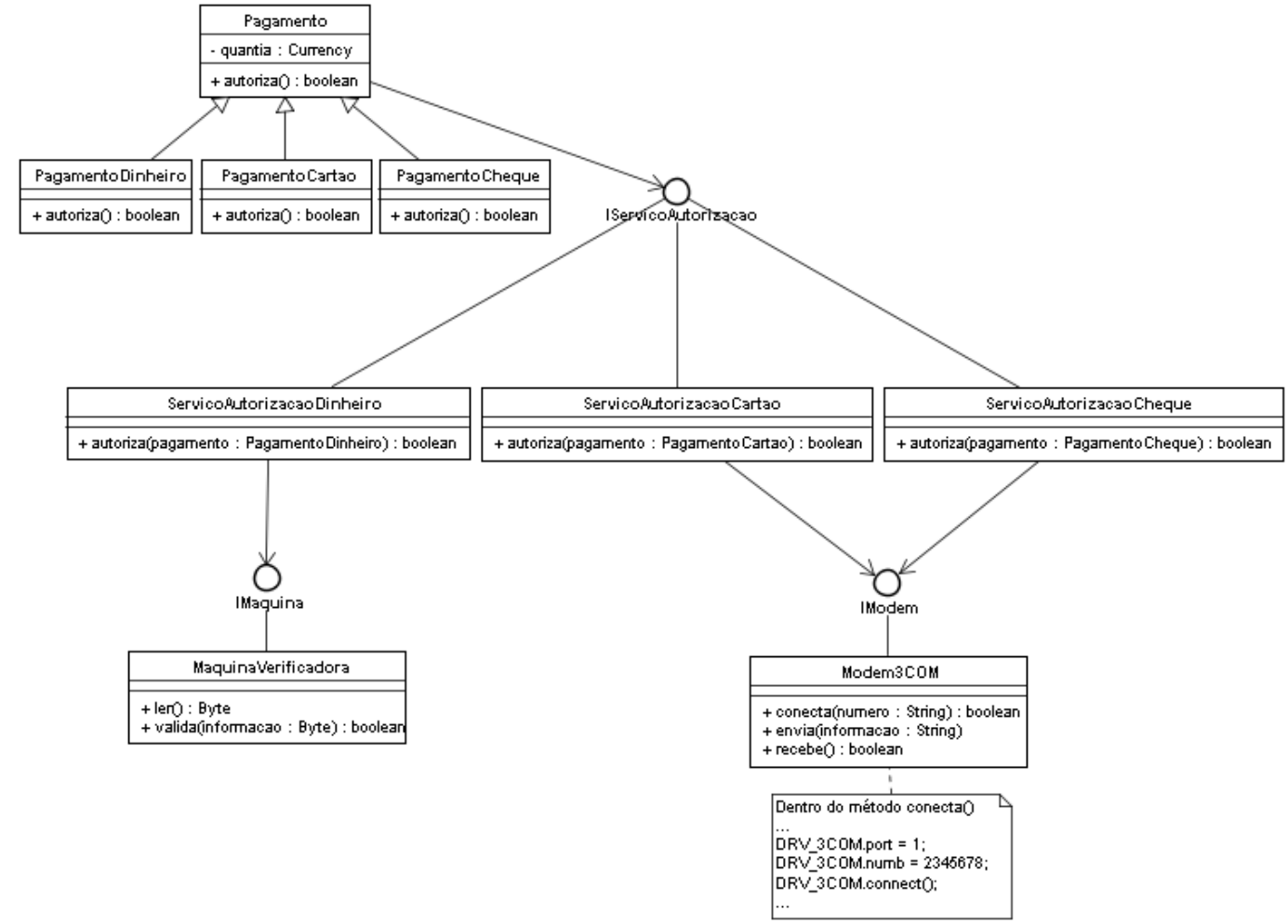


Questão 3 [1.0 pontos]

Evolua o modelo abaixo para contemplar o padrão *Indirection* para as três formas de pagamento *cartão*, *cheque* e *dinheiro*. Nesse caso, a indireção deve ser feita para o modem (cartão e cheque) e para a máquina de verificação de autenticidade de notas (dinheiro). Sobre o modelo gerado, perceba que um determinado serviço de autorização pode ser visto como um componente (interfaces providas e requeridas).

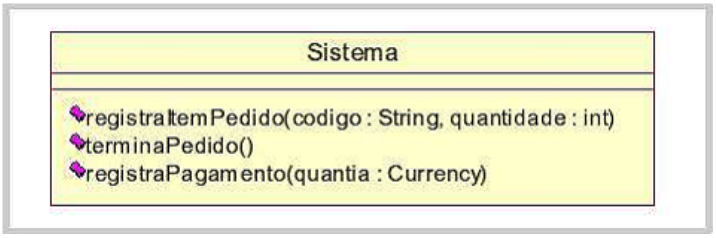


Resposta:



Questão 4 [1.5 pontos]

Para os eventos de sistemas listados abaixo, na classe virtual `Sistema` (vindos do caso de uso *Compra de itens*), defina uma atribuição de responsabilidades e argumente o motivo da sua escolha citando um padrão GRASP [0.5 pontos]. Além disso, elabore um diagrama de sequência para cada evento, explicitando quais classes recebem que requisições (provenientes das mensagens de sistema) e como essas classes (*ControleCompraItens*, *Catalogo*, *Pedido*, *ItemPedido* e *Pagamento)* devem estar estruturadas para tratá-las [1.0 pontos].

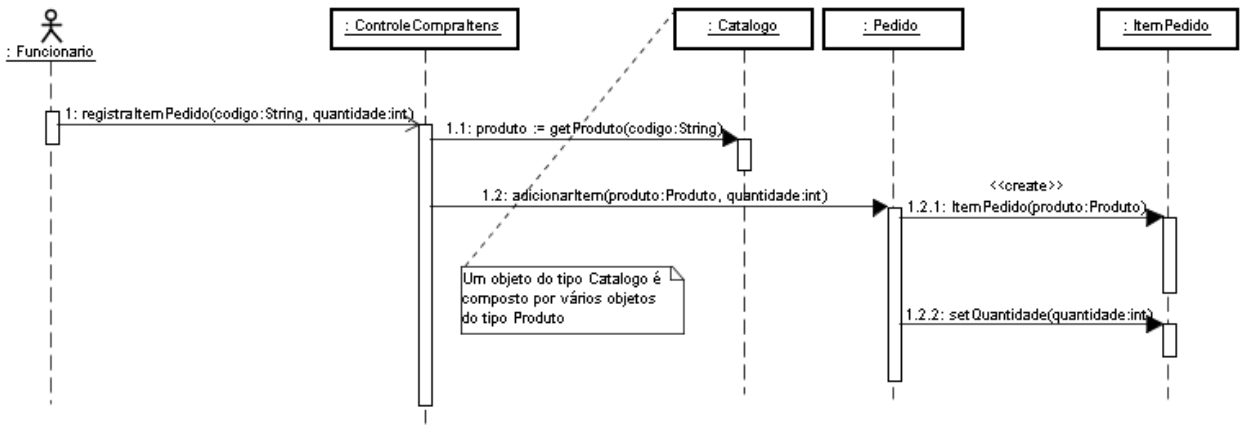


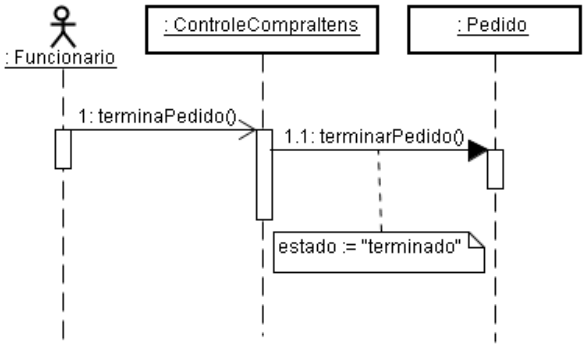
R: Sabe-se que os eventos de sistema estão associados às mensagens de sistema, que são geradas a partir dos passos dos casos de uso. Visando realizar uma atribuição de responsabilidades, ou seja, "quem" deve ser o responsável por tratar cada um desses eventos, utiliza-se o padrão *Controller*, desenvolvendo modelos de sequência que explicitem essa atribuição. No caso dos três eventos advindos do caso de uso *Compra de Itens* (apresentados pela classe virtual `Sistema`), iremos apresentar três modelos de sequência que atribuem a um representante do caso de uso em questão (tratador artificial), a classe `ControleCompraItens`, a responsabilidade por tratar as mensagens relacionadas a esses eventos.

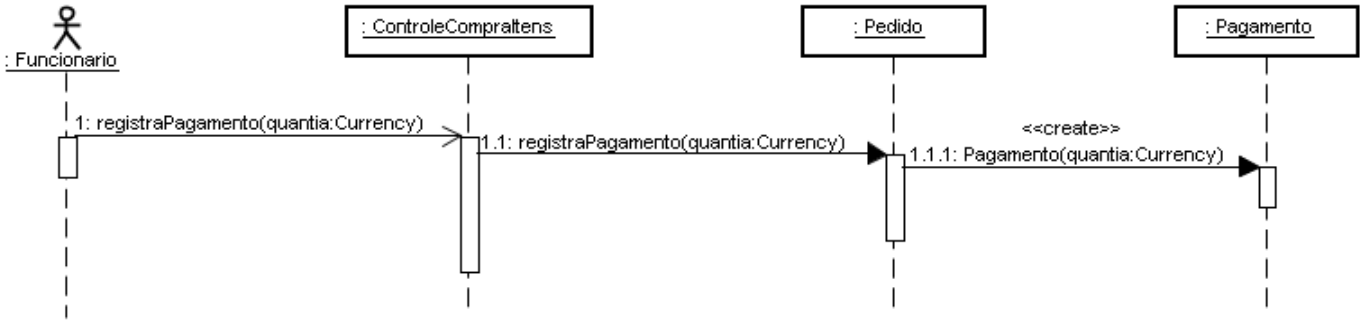
Como argumento geral para essa escolha, cita-se a separação *view-controller*, que facilita a reutilização de componentes específicos de negócio (lógica) diante de quaisquer alterações na interface com o usuário. Ou seja, delegar a responsabilidade de uma operação de sistema a um controlador apoia a reutilização da lógica em futuras aplicações e, como a lógica não está ligada à camada de interface, pode ser substituída por uma interface diferente.

Como argumento específico, relativo à escolha do controlador como o tratador artificial, ressalta-se que a utilização de uma única classe para todos os eventos de um caso de uso possibilita a manutenção de estado do caso de uso como atributos dessa classe. Os três eventos listados em `Sistema` constituem passos do caso de uso *Compra de Itens* e essa informação pode ser necessária para verificar se eles acontecem em uma sequência válida ou no caso de se desejar ser capaz de raciocinar sobre o estado corrente de atividade e operações dentro do caso de uso em andamento. Por exemplo, pode ser necessário garantir que a operação *registraPagamento* não possa ocorrer até que a operação *terminaPedido* tenha ocorrido. Se esse for o caso, a informação de estado deverá ser buscada em algum lugar; o controlador é uma boa escolha, especialmente se o mesmo controlador for utilizado em todo o caso de uso (o que é recomendável). Além disso, a utilização desse controlador pode ser útil para não sobrecarregar nenhuma classe, pois, como cada caso de uso tem sua própria classe, mesmo que o sistema tivesse 1000 casos de uso, nenhuma classe ficaria muito grande.

Os diagramas de sequência são:




: Funcionario
: ControleCompraItens
: Pedido
1: terminaPedido()
1.1: terminarPedido()
estado := "terminado"


: Funcionario
: ControleCompraItens
: Pedido
: Pagamento
1: registraPagamento(quantia:Currency)
1.1: registraPagamento(quantia:Currency)
<<create>>
1.1.1: Pagamento(quantia:Currency)