Projeto de Software

Padrões de Projeto e GRASP

Lesandro Ponciano

2024

Objetivos da Aula

- Apresentar o conceito de Projeto Guiado por Responsabilidades (PGR)
- Contextualizar o surgimento de Padrões de Projeto
- Contextualizar padrões GRASP e *Gang-of-Four*
- Apresentar os padrões GRASP

PGR

- Projeto guiado por responsabilidades (PGR)
- Uma forma de raciocinar sobre o projeto de objetos de software e também sobre componentes é em termos de
  - Responsabilidades
  - Papéis
  - Colaborações
- Objetos de software vistos como pessoas com responsabilidades, que colaboram com outras para fazer com que o trabalho seja feito
  - Comunidade de objetos responsáveis que colaboram entre si

Responsabilidade

- Responsabilidade é uma abstração do que os objetos fazem
- A UML define responsabilidade como "*Um contrato ou obrigação de um classificador*"
  - Obrigações de um objeto em termos do seu papel
- Responsabilidades são de dois tipos
  - Fazer
  - Saber

Responsabilidade de Fazer e Saber

- As responsabilidades de “fazer” de um objeto incluem
  - Fazer algo propriamente dito, como criar um objeto ou executar um cálculo
  - Iniciar uma ação em outros objetos
  - Controlar e coordenar atividades em outros objetos

- As responsabilidades de “saber” de um objeto incluem
  - Ter conhecimento sobre dados privados e encapsulados
  - Conhecer objetos relacionados
  - Ter conhecimento sobre coisas que ele pode derivar ou calcular

Colaboração

- Uma responsabilidade não é a mesma coisa que um método – é uma abstração – porém métodos satisfazem às responsabilidades

- PGR inclui a ideia de colaboração
  - Responsabilidades são implementadas por métodos que atuam sozinhos ou colaboram com outros métodos e objetos

Jargão de Projetistas/Desenvolvedores

- Jill: *Jack, para o subsistema de persistência, vamos expor os serviços como uma Fachada. Usaremos uma Fábrica Abstrata para mapeadores e Procuradores para materialização sob demanda*"
- Jack: "*Que diabos você acabou de dizer?!?*"
- Jill: "*Aqui, leia isto…*"

Padrões

- Padrões são repositório de princípios gerais e de soluções que guiam a criação de software

- Padrões
  - Descrição denominada de um problema e solução que pode ser aplicada em novos contextos

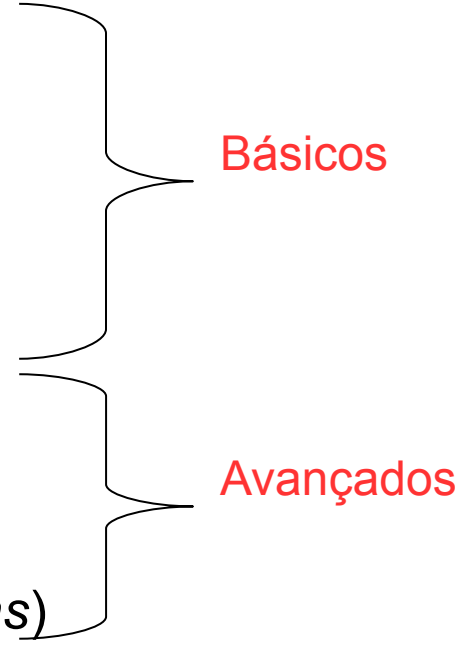
- Há diversos padrões
  - GRASP
  - Gang-of-Four (GoF)
    - 23 padrões de autoria de 4 pessoas
    - "*Design Patterns*", bíblia de padrões de projeto escrito por Gamma, Helm, Johnson e Vlissides

GRASP

- GRASP
  - *General Responsibility Assignment Software Patterns*
  - Diretrizes, guias, padrões para atribuir responsabilidade a classes e objetos
- Destaca a importância de compreender (*grasp*) os princípios para projetar software OO
- Define nove padrões

Aplicação de GRASP

- Padrões GRASP
  1) Criador (*Creator*)
  2) Especialista (*Information Expert*)
  3) Acoplamento Baixo (*Low coupling*)
  4) Controlador (*Controller*)
  5) Coesão Alta (*High Cohesion*)
  6) Polimorfismo (*Polymorphism*)
  7) Invenção Pura (*Pure Fabrication*)
  8) Indireção (*Indirection*)
  9) Variações Protegidas (*Protected Variations*)



Padrão Criador

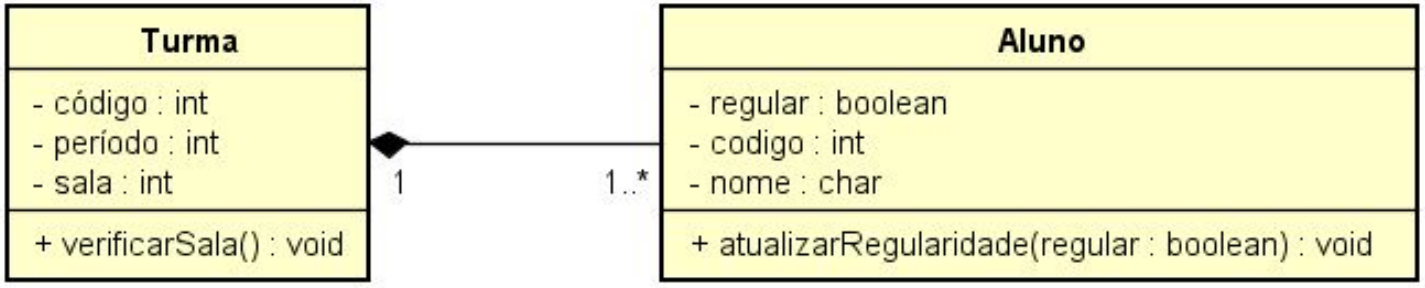
Problema: Quem deve ser responsável pela criação de uma nova instância de uma classe?

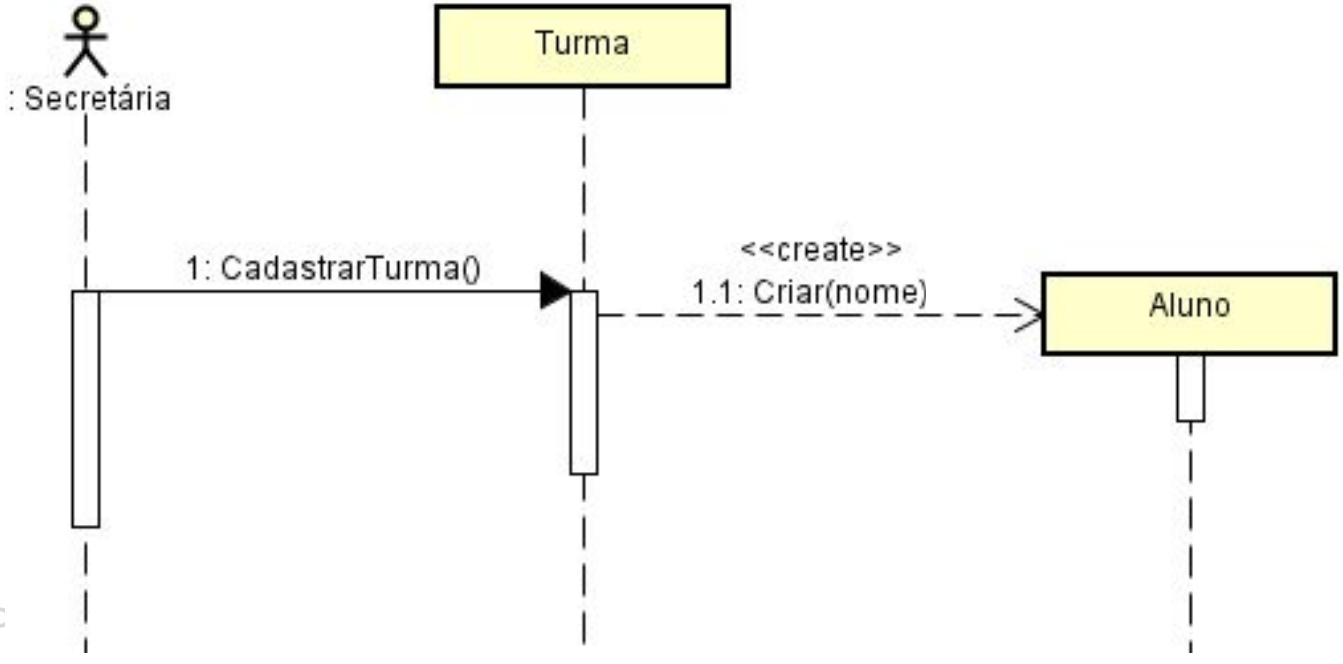
Solução: Classe *B* deve criar instância da classe *A* se pelo menos uma das seguintes condições for verdade (quanto mais melhor)

- *B* "contém" *A* ou *B* "agrega" *A* de modo composto
- *B* registra *A*
- *B* usa *A* de maneira muito próxima
- *B* tem dados iniciais de *A*, que serão repassados a *A* quando criada

- Diz-se que *B* é um *criador* de *A*

Modelo (Parcial)





Padrão Especialista

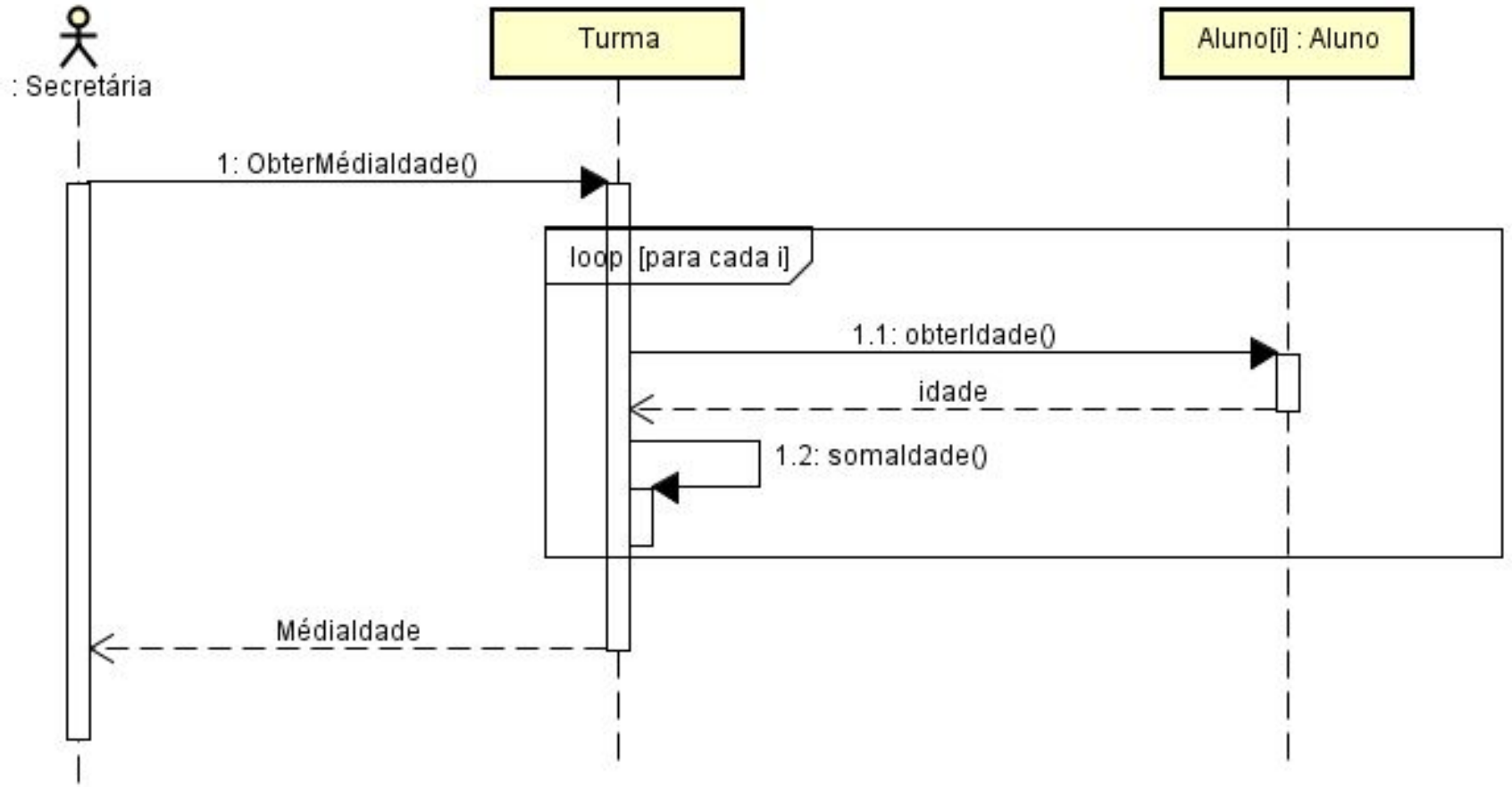
Problema: Qual o princípio geral de atribuição de responsabilidade a objetos?

Solução: Atribua responsabilidade ao especialista na informação – a classe que tem a informação necessária para satisfazer a responsabilidade

- Outros nomes
  - "quem sabe faz", "fazer por si", "colocar serviços com os atributos com os quais eles trabalham"
- Em alguns casos pode comprometer o acoplamento e a coesão

Exemplo

- Questão que enuncia a responsabilidade: *Quem deve ser responsável por conhecer a média de idade dos alunos na turma?*
- A resposta seria *Turma*? *Aluno*? Outra classe?
- Turma
  - Novo Método: *CalcularMédiaDeIdade*()
  - Novo Atributo: *mediaIdade*

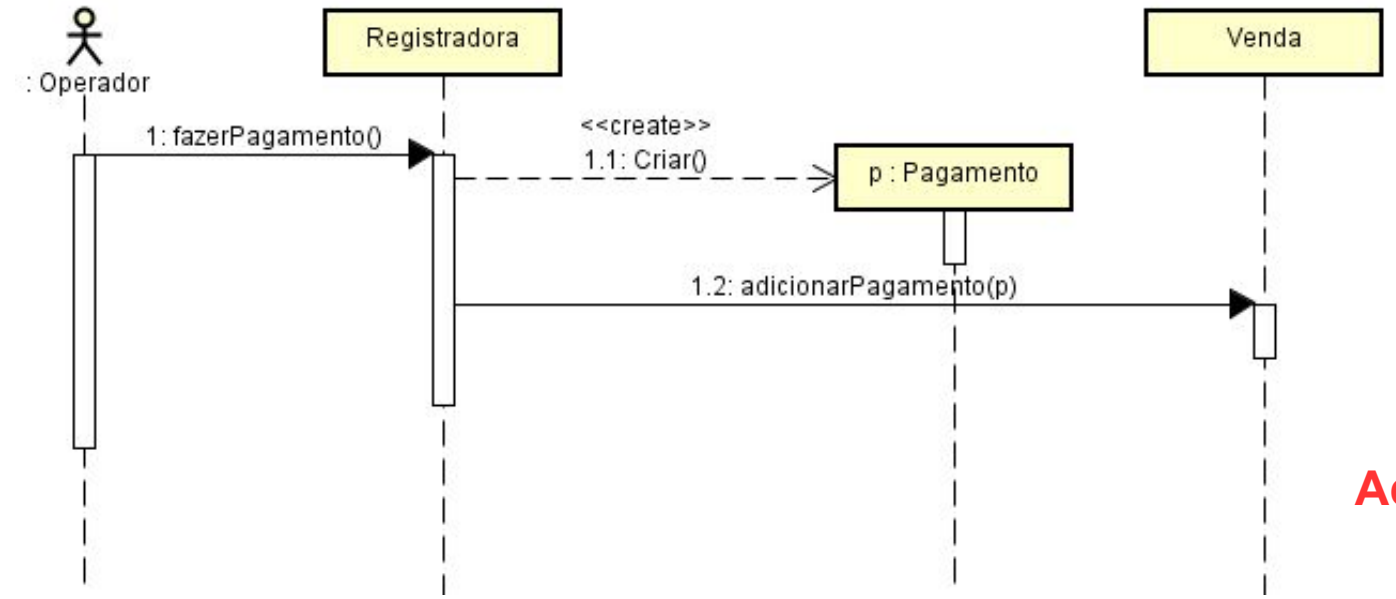

: Secretária
Turma
Aluno[i] : Aluno
1: ObterMédialdade()
loop [para cada i]
1.1: obterIdade()
idade
1.2: somaIdade()
Médialdade

Padrão Acoplamento Baixo

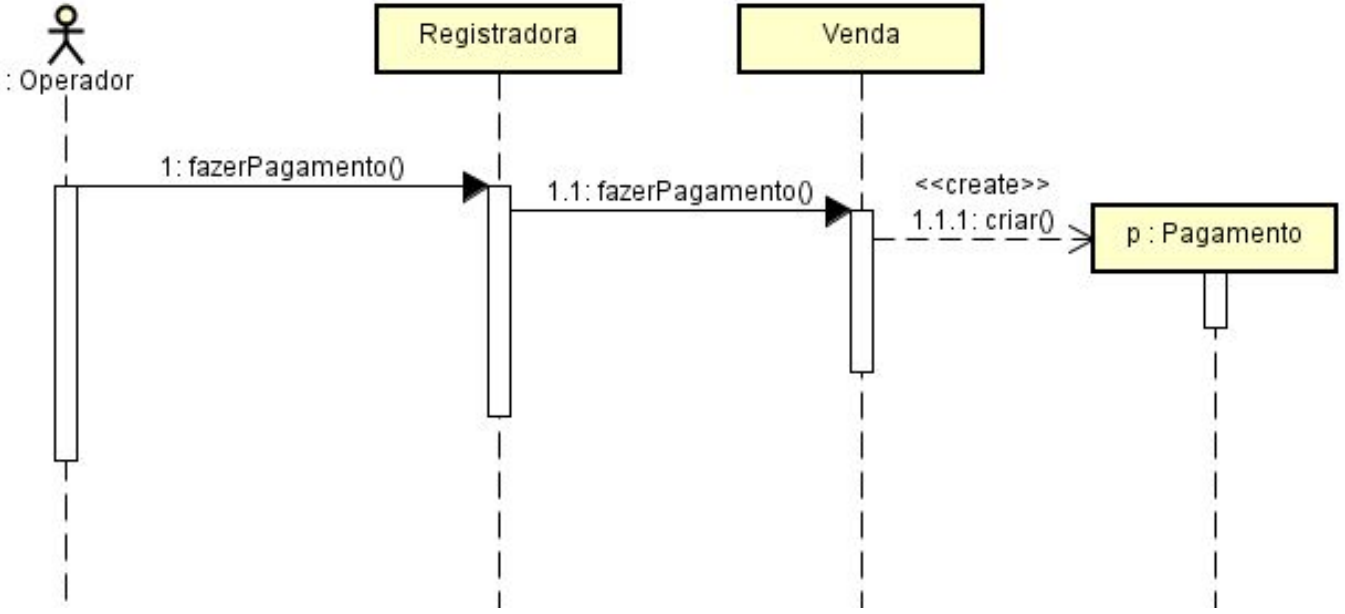
Problema: Como apoiar dependência baixa, baixo impacto de notificações e aumento de reuso?

Solução: Atribuir responsabilidade de modo que o acoplamento permaneça baixo. Use esse princípio para avaliar alternativas.

- Importante
  - Padrões diferentes podem sugerir soluções diferentes
  - Acoplamento Baixo não pode ser considerado isolado de outros princípios, tais como Especialista e Coesão Alta



Acoplamento alto



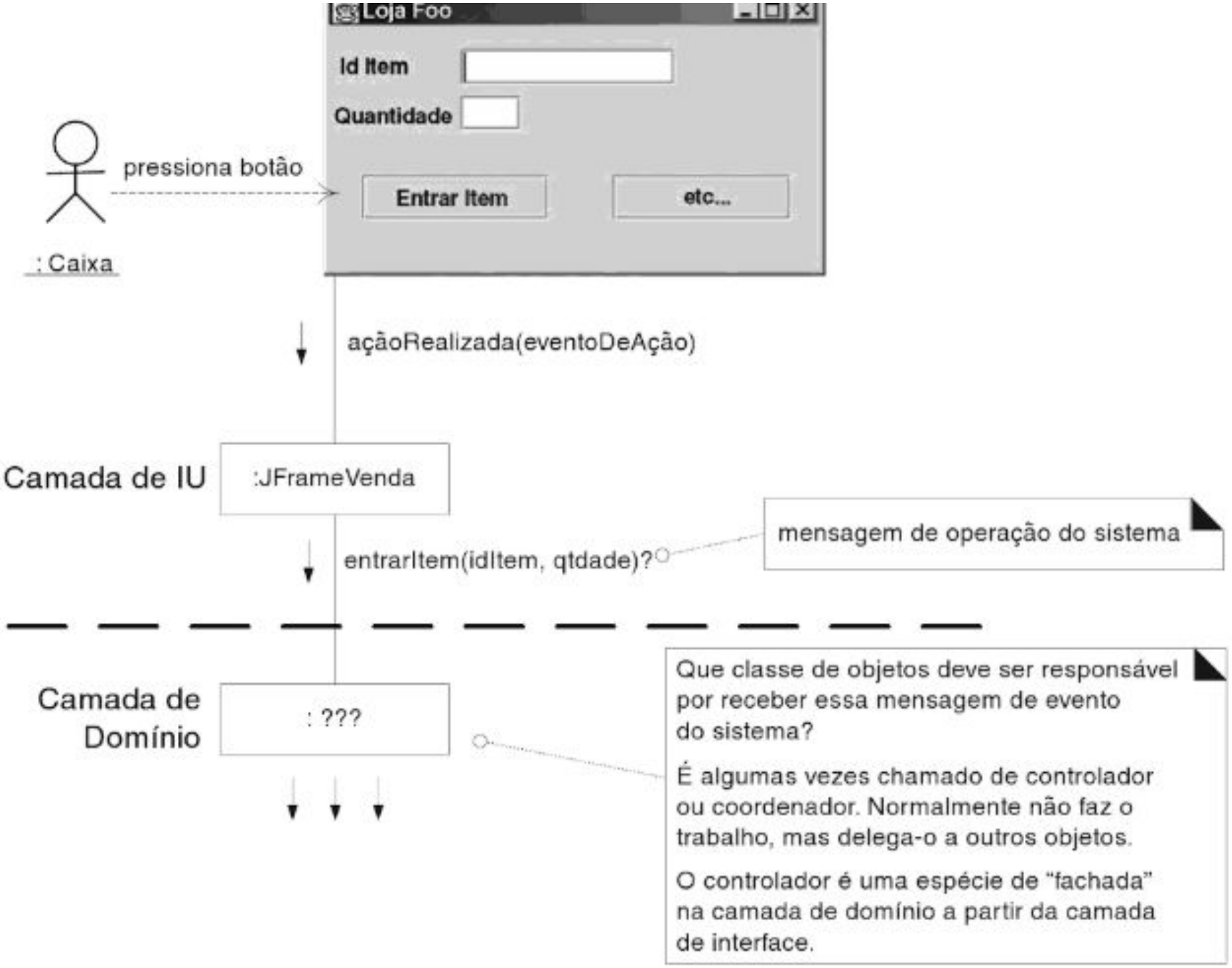
Acoplamento baixo

Padrão Controlador

Problema: Qual é o primeiro objeto, além da camada de Interface com Usuário (IU), que recebe e controla uma operação do sistema?

Solução: Atribua responsabilidade a uma classe que representa uma das seguintes escolhas

- Representa o "sistema" global, um objeto raiz, um dispositivo dentro do qual o software está sendo processado, ou um subsistema importante
- Representa um cenário de um caso de uso dentro do qual ocorre o evento do sistema, frequentemente denominado:
  - TratadorDo<NomeDoCasoDeUso>
  - CoordenadorDo<NomeDoCasoDeUso>
  - SessãoDo<NomeDoCasoDeUso>


Loja Foo
Id Item
Quantidade
Entrar Item
etc...
pressiona botão
: Caixa
açãoRealizada(eventoDeAção)
Camada de IU
:JFrameVenda
entrarItem(idItem, qtdade)?
mensagem de operação do sistema
Camada de Domínio
: ???
Que classe de objetos deve ser responsável por receber essa mensagem de evento do sistema?
É algumas vezes chamado de controlador ou coordenador. Normalmente não faz o trabalho, mas delega-o a outros objetos.
O controlador é uma espécie de "fachada" na camada de domínio a partir da camada de interface.

A considerar no Padrão Controlador

- É um corolário que a camada de UI não tem como responsabilidade o tratamento de eventos de sistema

- O padrão controlador é diferente do controlador em a arquitetura *Model-View-Controller* (MVC)
- A rigor, o controlador faz pouca tarefa, apenas delega a outros objetos o serviço que precisa ser feito

- Vantagens
  - Aumento das possibilidades de reutilização e de interfaces "plugáveis"
  - Oportunidade de raciocinar sobre o estado do caso de uso

Padrão Coesão Alta

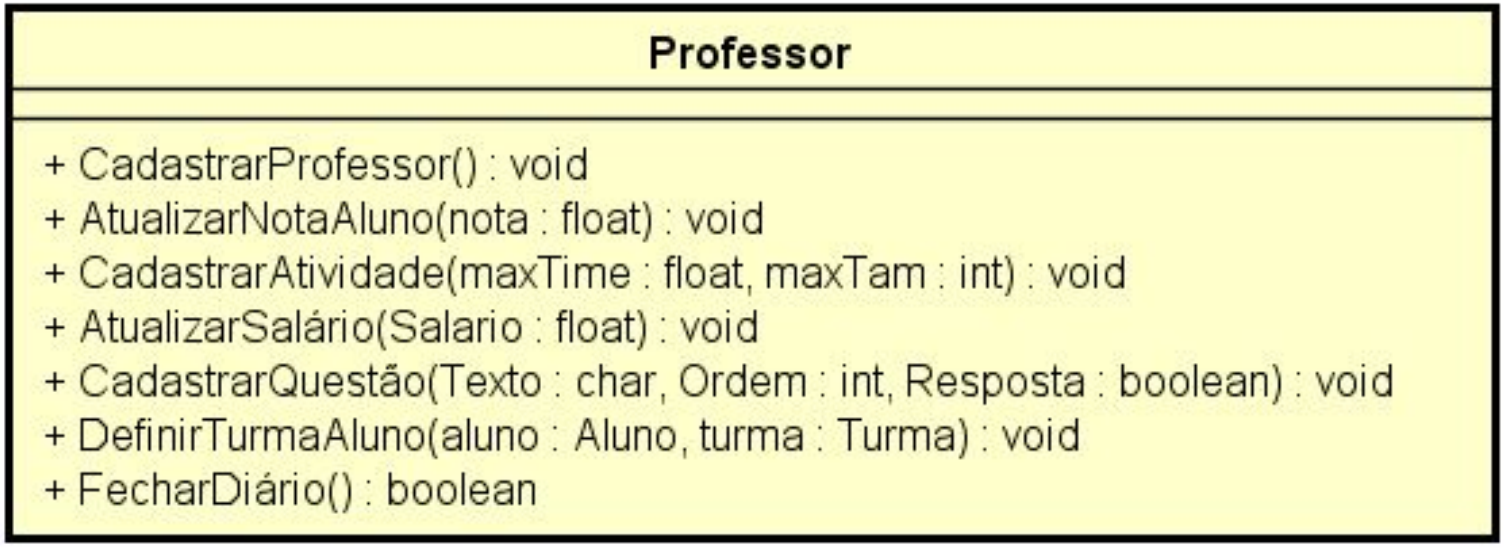
Problema: Como manter os objetos focados, inteligíveis e gerenciáveis e como efeito colateral apoiar o "Baixo Acoplamento"?

Solução: Atribuir uma responsabilidade de forma que a coesão permaneça alta. Usar isso para avaliar alternativas.

- Baixa coesão funcional indica que o objeto possui responsabilidades não relacionadas e executa grande volume de trabalho. Classes com baixa coesão:
  - São difíceis de compreender
  - São difíceis de reutilizar e manter
  - São delicadas; constantemente afetadas por modificações

Coesão Muito Baixa

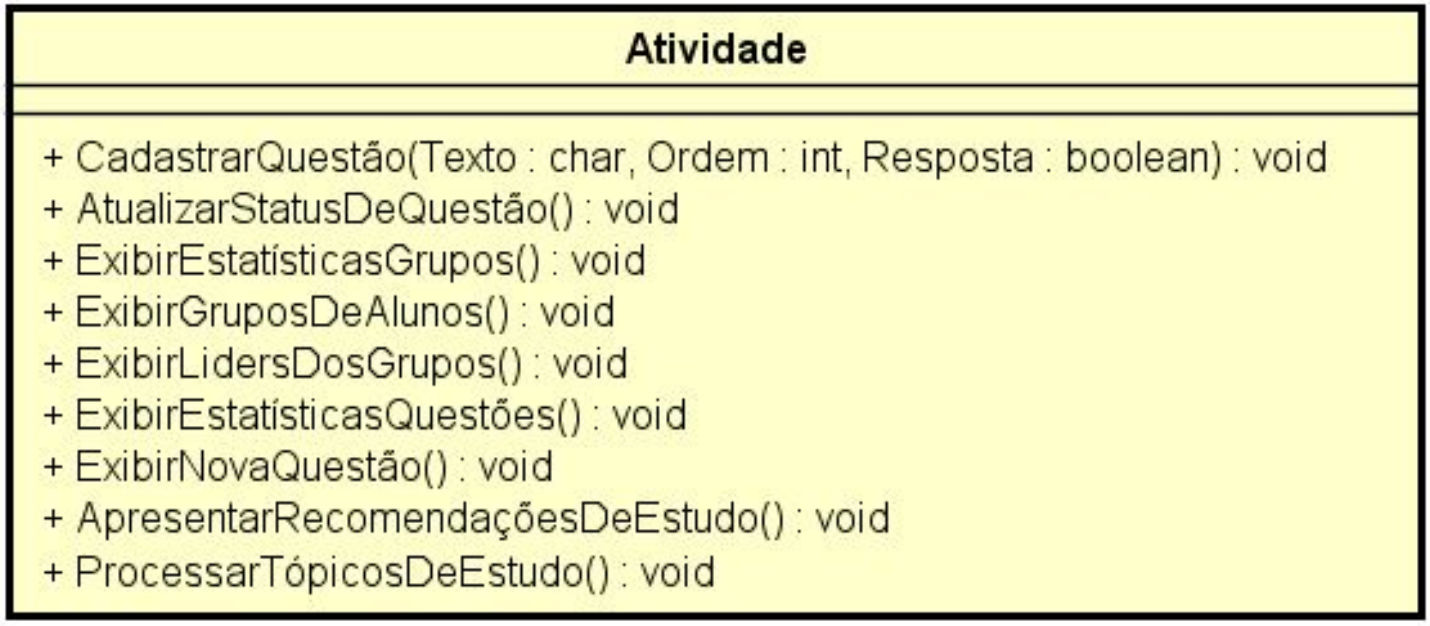
- Uma classe é a única responsável por muitas coisas em áreas funcionais muito diferentes



- Atributos omitidos

Coesão Baixa

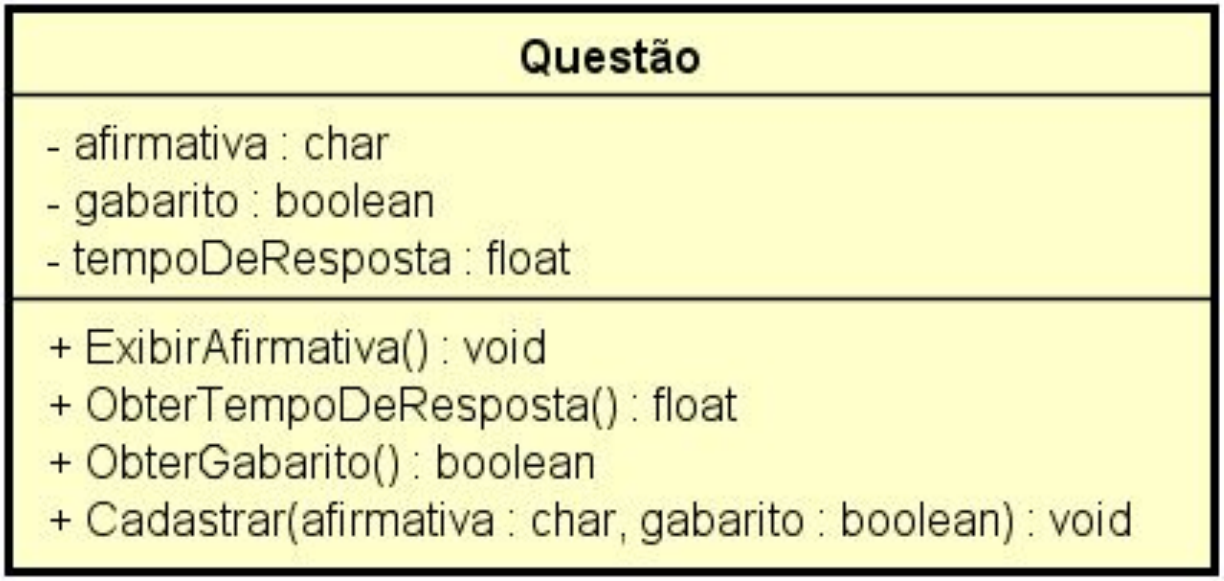
- Uma classe é a única responsável por uma tarefa complexa em uma área funcional



- Atributos omitidos

Coesão Alta

- Uma classe tem responsabilidades moderadas em uma área funcional e colabora com outras classes para realizar tarefas



Coesão Moderada

- Uma classe tem peso leve e responsabilidade exclusiva em algumas áreas logicamente relacionadas ao conceito da classe, mas não uma com as outras

Padrão Polimorfismo

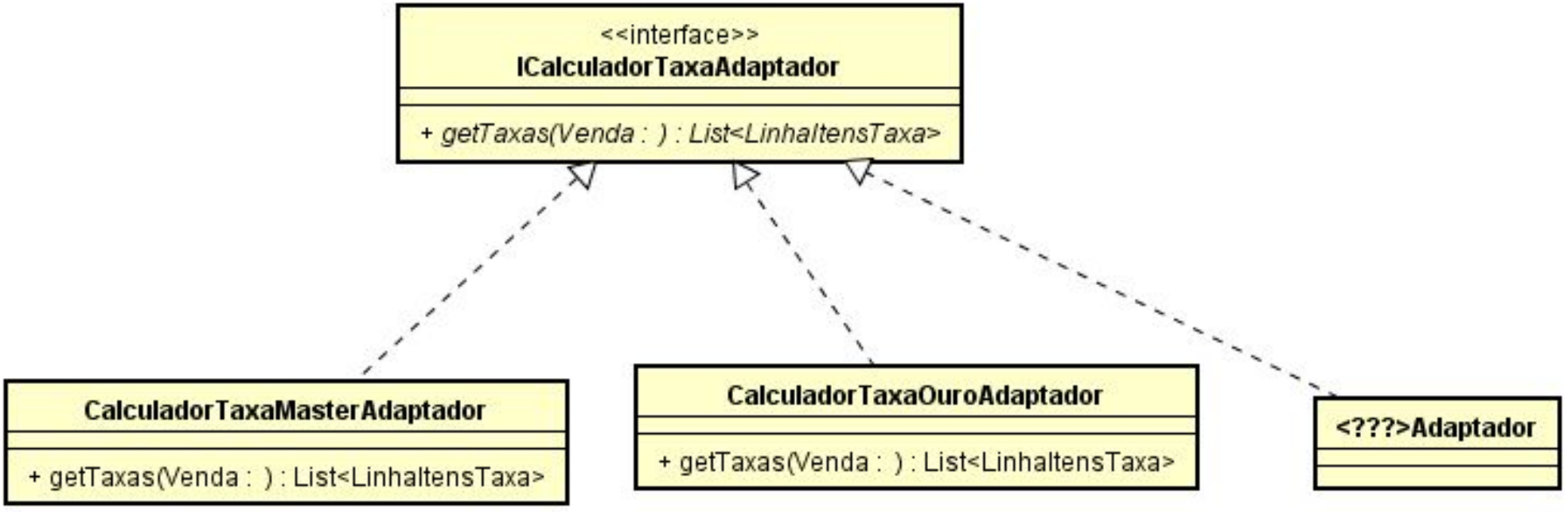
Problema: Como tratar alternativas com base no tipo? Como criar componentes de software interconectáveis?

- *If-else-then* ou *case* exigem mudanças (geralmente em vários lugares) caso surjam novos tipos
- Em vários componentes ligados, substituir um por outro sem afetar os demais

Solução: Usar operações polimórficas para implementar as responsabilidades quando alternativas ou comportamentos relacionados variam com o tipo

Corolário: Não teste o tipo de um objeto e não use lógica condicional para executar alternativas que variam com base no tipo

Exemplo (Fragmento)



Não muda nada que já está feito, só cria um tipo novo.

Padrão Invenção Pura

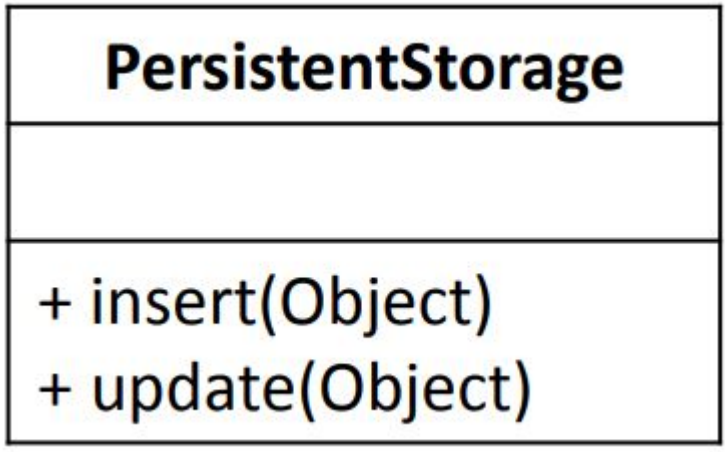
Problema: Qual objeto deve ter a responsabilidade quando não se quer violar a 'Coesão Alta' e o 'Acoplamento Baixo' ou outros objetivos, mas as soluções oferecidas pelo Especialista (por exemplo) não são apropriadas?

Solução: Atribuir um conjunto de responsabilidades altamente coeso a uma classe artificial ou de conveniência que não represente um conceito do domínio do problema

- Algo inventado para apoiar alta coesão e baixo acoplamento
- O projeto da invenção é muito limpo ou "puro"

Exemplo (Fragmento)

- Questão que enuncia a responsabilidade: *Qual a classe responsável apenas por salvar uma venda no banco de dados?*



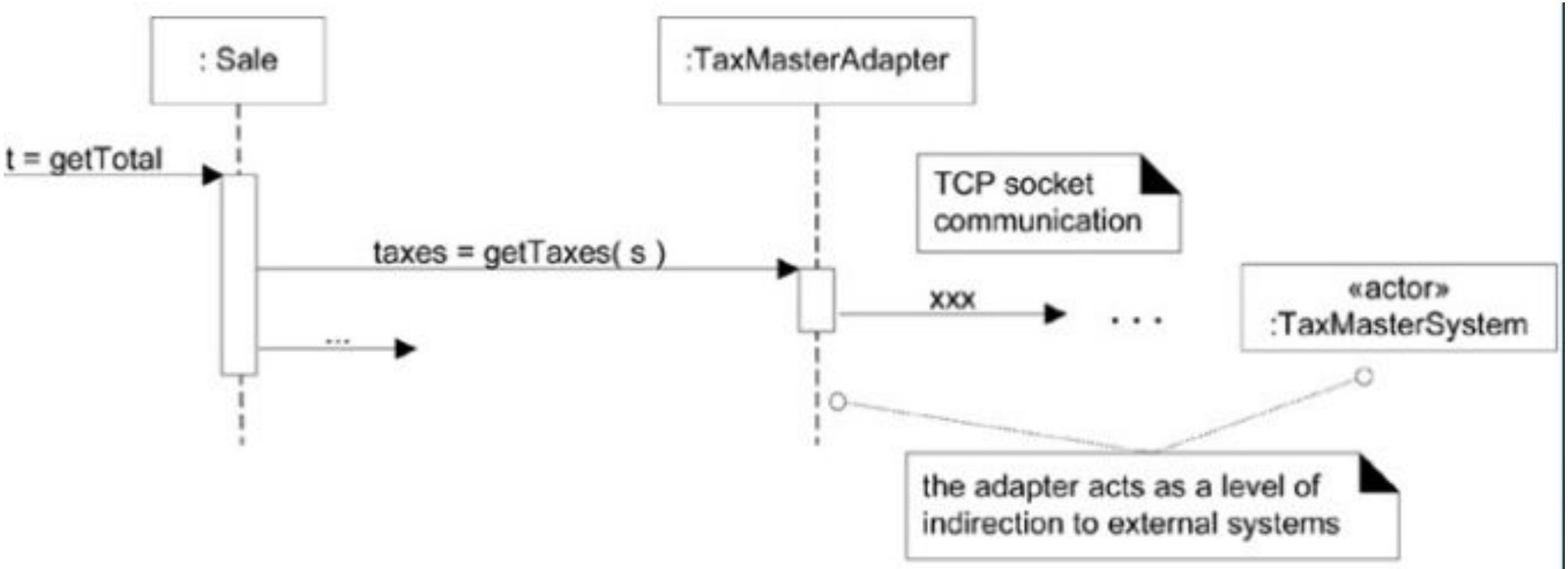
Padrão Indireção

- **Problema**: A quem devemos atribuir a responsabilidade de maneira a evitar o acoplamento direto entre dois (ou mais) objetos? Como desacoplar os objetos de modo que o 'Acoplamento Baixo' seja apoiado e o potencial de reúso permaneça mais alto?

- **Solução**: Atribuir a responsabilidade de ser o mediador entre outros componentes ou serviços a um objeto intermediário, para que eles não sejam diretamente acoplados

Exemplo (usando adaptadores)

- Manter o baixo acoplamento, através de delegação de responsabilidades a uma classe mediadora



Padrão: Variações Protegidas

- **Problema**: Como projetar objetos, subsistemas e sistemas de modo que as variações ou a instabilidade nesses elementos não tenham um impacto indesejado sobre outros elementos?

- **Solução**: Identificar pontos de variação ou instabilidade; atribuir responsabilidades para criar uma interface estável em torno deles
  - O termo "Interface" nesta frase tem com conceito mais amplo que em Programação Orientada a Objetos
  - Adaptadores, polimorfismo e interfaces

Exercício de Fixação

- Explique em que consiste cada um dos seguintes padrões
  - Criador
  - Especialista
  - Acoplamento Baixo
  - Controlador
  - Coesão Alta
  - Polimorfismo
  - Invenção Pura
  - Indireção
  - Variações Protegidas

Referências

LARMAN, C.; Utilizando UML e Padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo, Porto Alegre, Bookman, 2007. (Capítulo 11)

GUEDES, Gilleanes T. A. UML 2: uma abordagem prática. 2. ed. São Paulo: Novatec, c2011. 484 p. ISBN 9788575222812 (Capítulo 5-10)

BEZERRA, E.; Princípio de Análise e Projeto de Sistemas com UML, Rio de Janeiro, Elsevier, 2007.

Diagramas foram feitos usando a ferramenta Astah <http://astah.net/editions>

Projeto de Software

Prof. Dr. Lesandro Ponciano

https://orcid.org/0000-0002-5724-0094