Padrões GRASP

João Pedro Oliveira Batisteli

Projeto de Classes

- O que é projeto?

Projeto de Classes

- O que é projeto?
  - É uma das partes mais difíceis da programação.
  - Consiste em criar abstrações.
- Isto significa três coisas:
  - Quais classes devem ser **criadas**?
  - Quais **responsabilidades** (operações/métodos) devem ser assumidas por cada classe?
  - Quais são os **relacionamentos** entre tais classes e objetos dessas classes?

Projeto

- Criar boas abstrações é difícil e vem com experiência.

Porém, algumas regras básicas ajudarão a adquirir a experiência mais rapidamente

Exemplo: Decisões Oriundas de Projetos

- Qual é o problema deste código?

```
Classe VideoLocadora
fitas : Conjunto;
clienteCorrente : Cliente;

Método emprestaFita(fCodigo: String)
        fita : Fita;
        emprestimoCorrente : Emprestimo;
        item : ItemDeEmprestimo;

        fita := fitas.get(fCodigo);
        emprestimoCorrente := clienteCorrente.getEmprestimoCorrente();
        item := ItemDeEmprestimo.new();
        item.associaFita(fita);
        emprestimoCorrente.associaItem(item);
Fim Método;
Fim Classe.
```

Exemplo: Decisões Oriundas de Projetos

- Qual é o problema deste código?

Código com Responsabilidades concentradas

```
Classe VideoLocadora
fitas : Conjunto;
clienteCorrente : Cliente;

Método emprestaFita(fCodigo: String)
        fita : Fita;
        emprestimoCorrente : Emprestimo;
        item : ItemDeEmprestimo;

        fita := fitas.get(fCodigo);
        emprestimoCorrente := clienteCorrente.getEmprestimoCorrente();
        item := ItemDeEmprestimo.new();
        item.associaFita(fita);
        emprestimoCorrente.associaItem(item);
Fim Método;
Fim Classe.
```

Diagrama de Sequência: Responsabilidades Concentradas

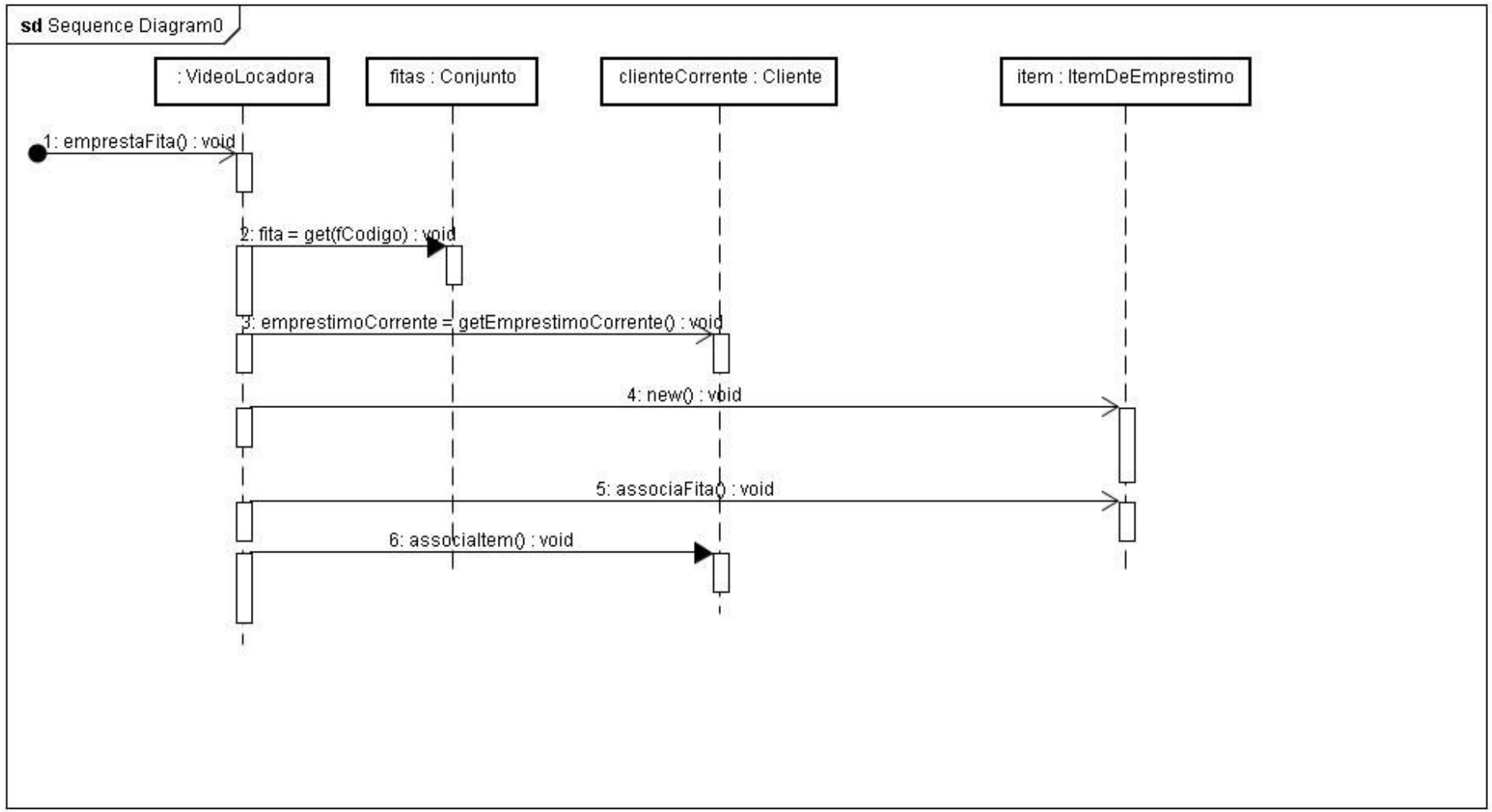
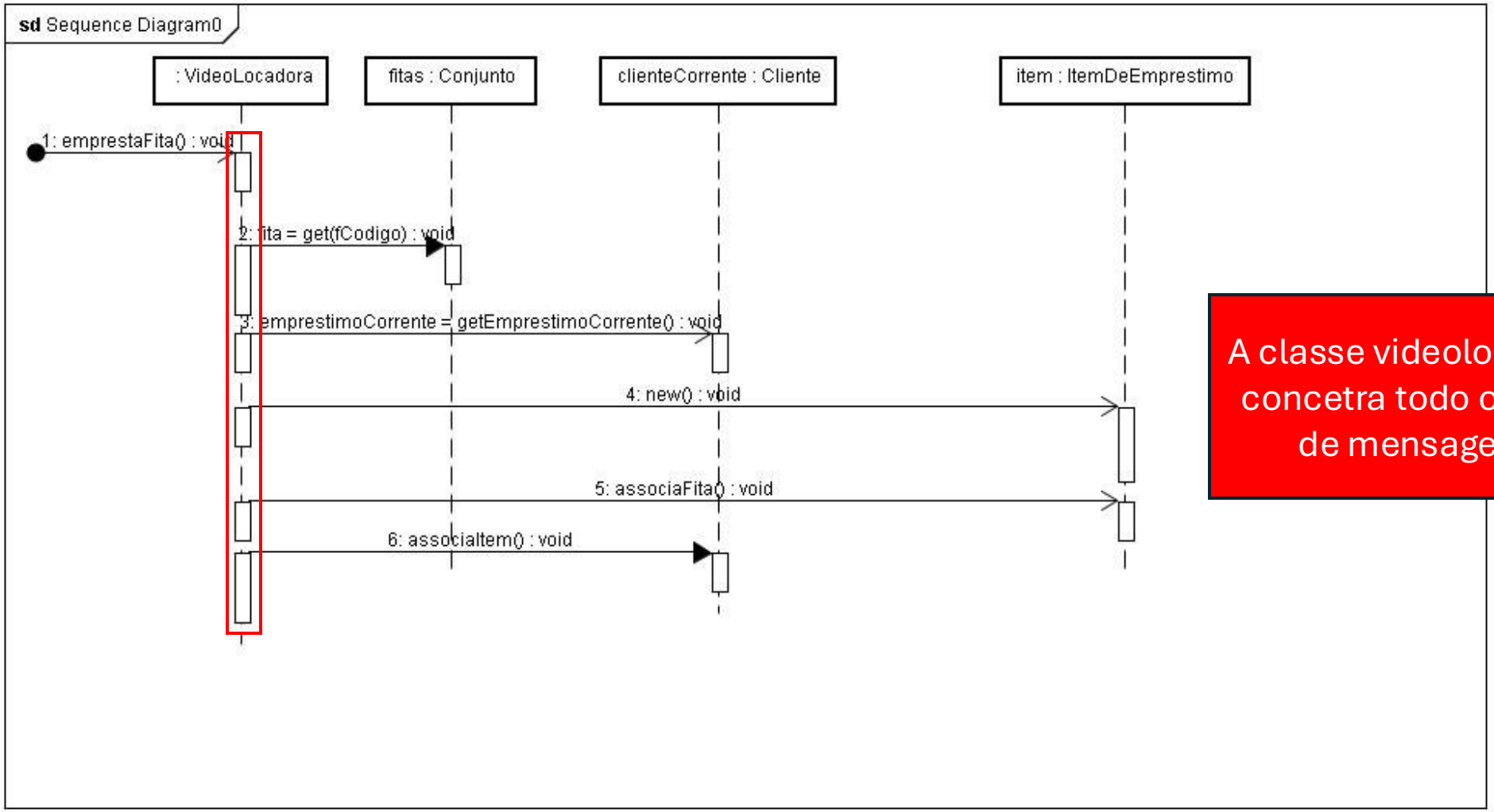


Diagrama de Sequência: Responsabilidades Concentradas



A classe videolocadora concetra todo o envio de mensagens

Exemplo: Decisões Oriundas de Projetos

- Qual é o problema deste código?

Código com Responsabilidades concentradas

```
Classe VideoLocadora
fitas : Conjunto;
clienteCorrente : Cliente;

Método emprestaFita(fCodigo: String)
        fita : Fita;
        emprestimoCorrente : Emprestimo;
        item : ItemDeEmprestimo;

        fita := fitas.get(fCodigo);
        emprestimoCorrente := clienteCorrente.getEmprestimoCorrente();
        item := ItemDeEmprestimo.new();
        item.associaFita(fita);
        emprestimoCorrente.associaItem(item);
Fim Método;
Fim Classe.
```

Qual a solução?

Responsabilidades Distribuídas

Classe VideoLocadora

fitas : Conjunto ;
clienteCorrente : Cliente;

Metodo emprestaFita(fCodigo : String);
    fita : Fita;

    fita := fitas.get(fCodigo);
    clienteCorrente.empresta(fita)
Fim Metodo;
Fim Classe.

Classe Cliente

emprestimoCorrente : Emprestimo;

Metodo empresta(fita : Fita);
    emprestimoCorrente.adiciona(fita);
Fim Metodo;
Fim Classe.

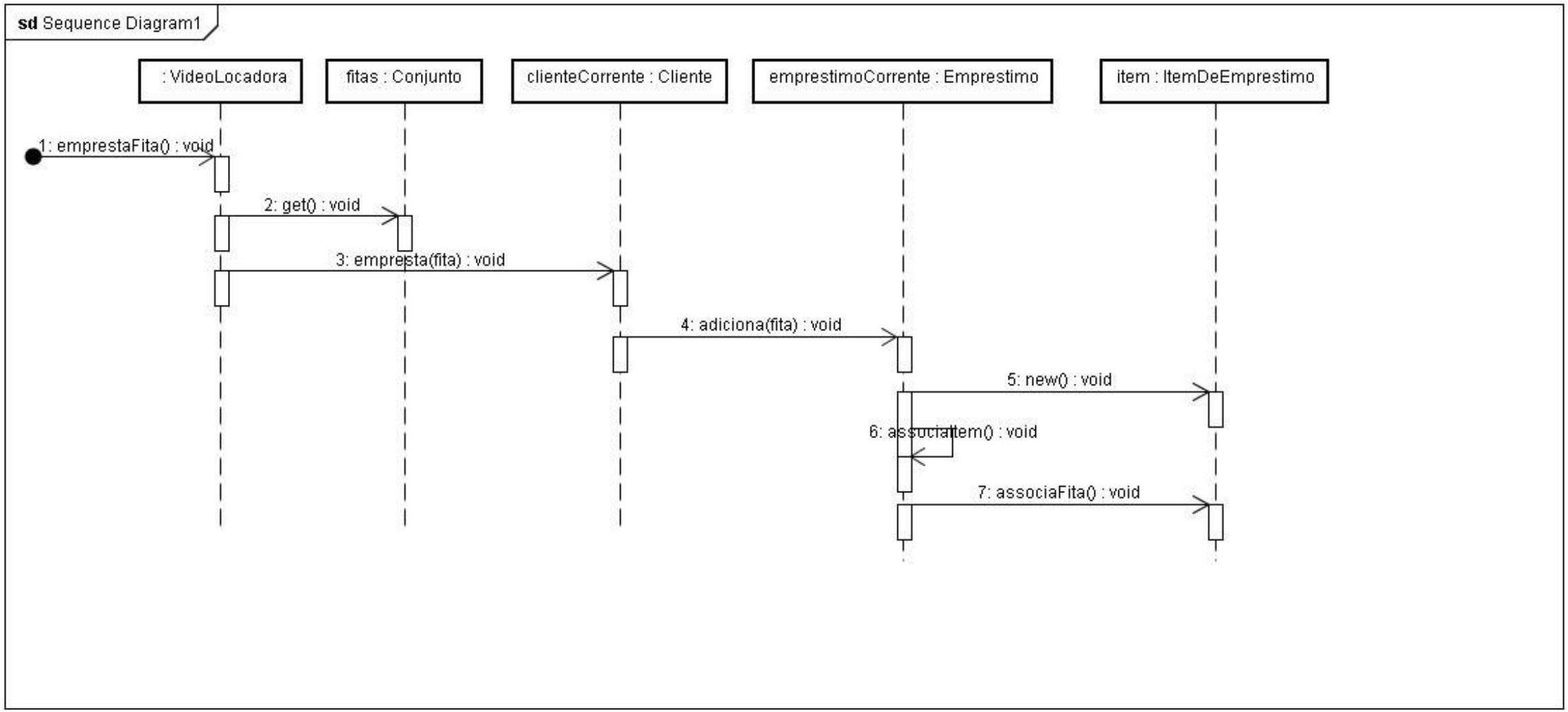
Classe Emprestimo

itens : Conjunto;

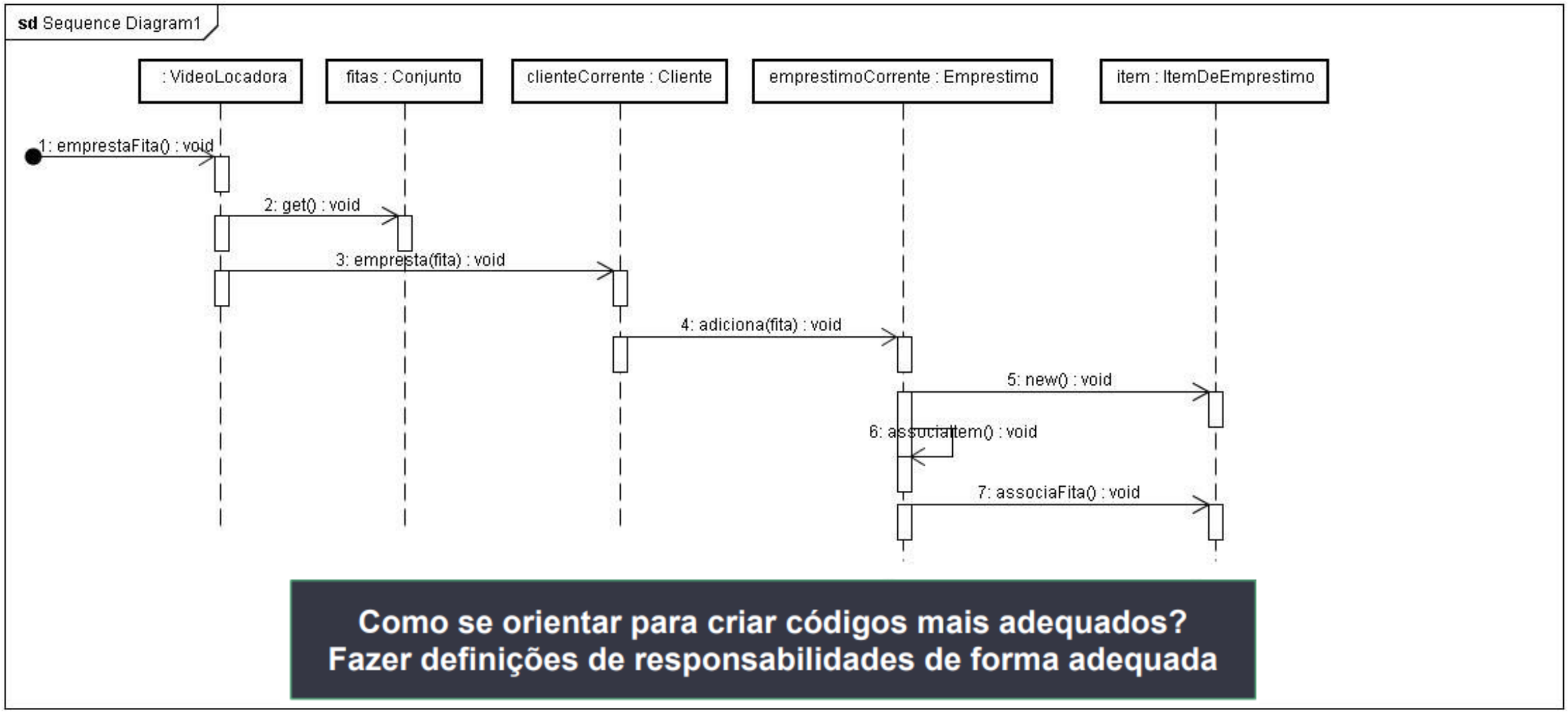
Metodo adiciona(fita : Fita);
    item : ItemDeEmprestimo;

    item := ItemDeEmprestimo.new();
    self.associaItem(item);
    item.associaFita(fita);
Fim Metodo;
Fim Classe.

Responsabilidades Distribuídas



Responsabilidades Distribuídas



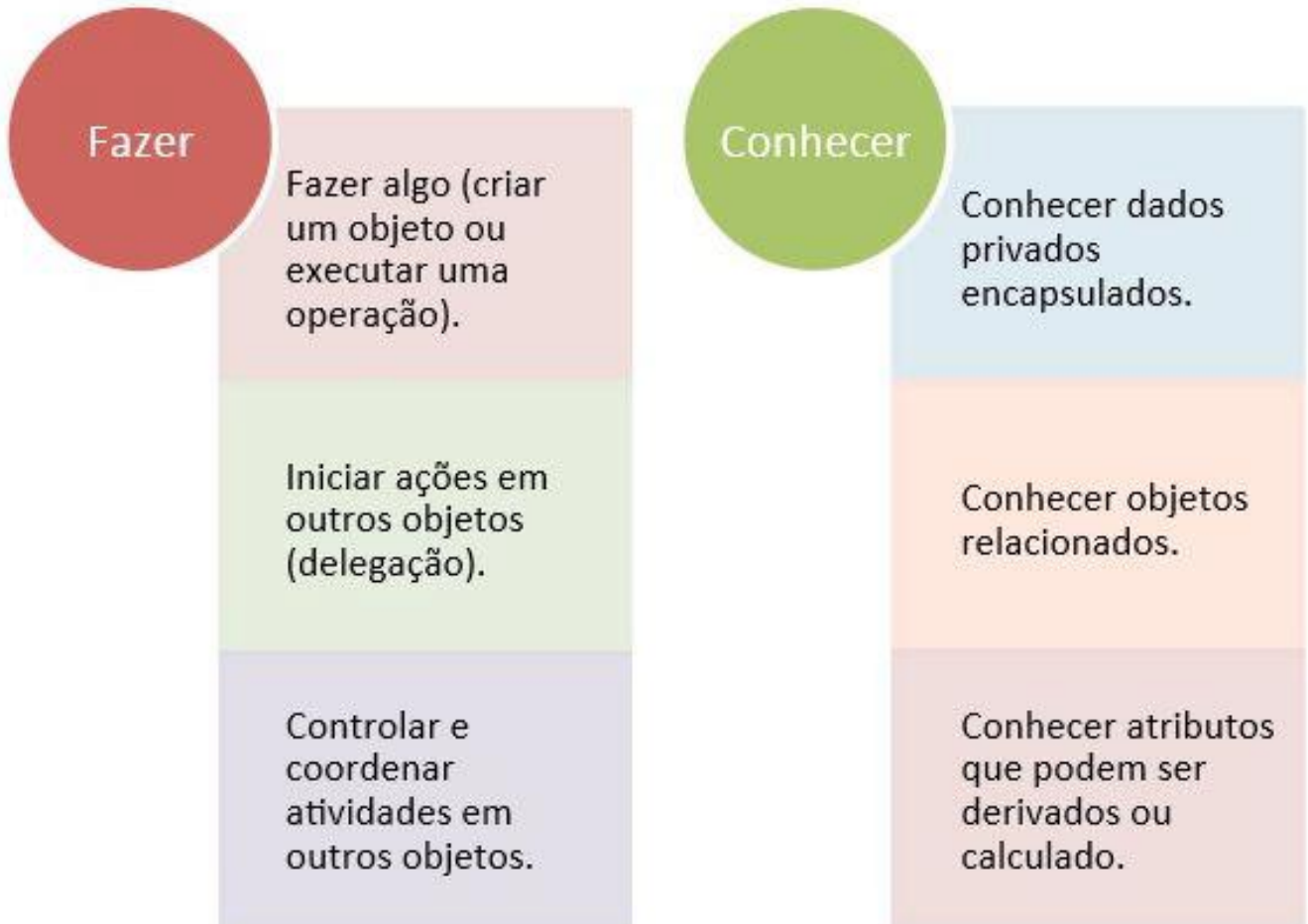
Como se orientar para criar códigos mais adequados?
Fazer definições de responsabilidades de forma adequada

Responsabilidades e Métodos

- Responsabilidade é “**um contrato ou obrigação de um tipo ou classe**” [Booch et al.’97]
  - Relacionada com o **comportamento** dos objetos.

- Responsabilidades são de dois tipos:
  - Fazer
  - Saber

Responsabilidades e Métodos



Responsabilidades e Métodos

- Responsabilidade são atribuídas aos objetos do sistema durante o projeto OO.
- Exemplos:
  - “Uma Venda é responsável por imprimir a si própria” **(de fazer)**
  - “Uma Venda é responsável por conhecer sua data” **(de conhecer)**
  - Uma Conta bancária tem a responsabilidade de logar as transações **(de fazer)**
  - Uma Conta bancária tem a responsabilidade de saber sua data de criação **(de conhecer)**

Responsabilidades e Métodos

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

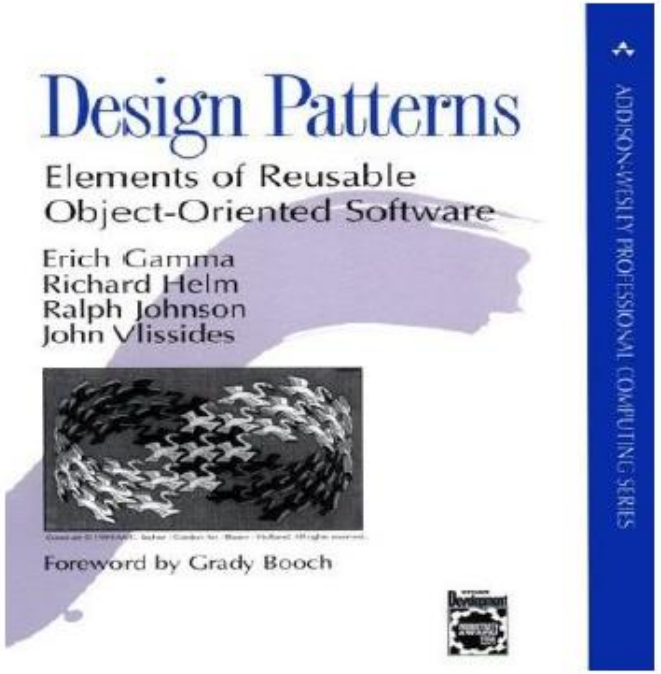
- **Exemplo :**
  - A classe **Venda** pode definir um **método específico** para cumprir a responsabilidade de **impressão**
  - Esse método, por sua vez, pode **precisar colaborar com outros objetos**, possivelmente enviando mensagens de impressão para cada um dos objetos Item-de-Venda associados à Venda.

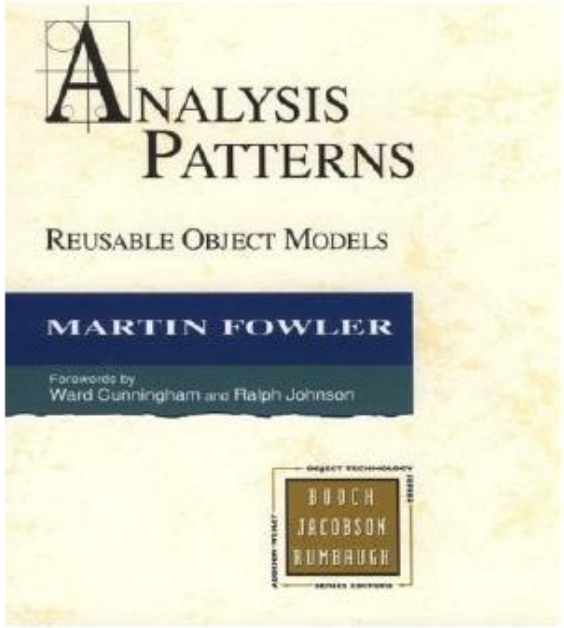
Padrões de Projeto

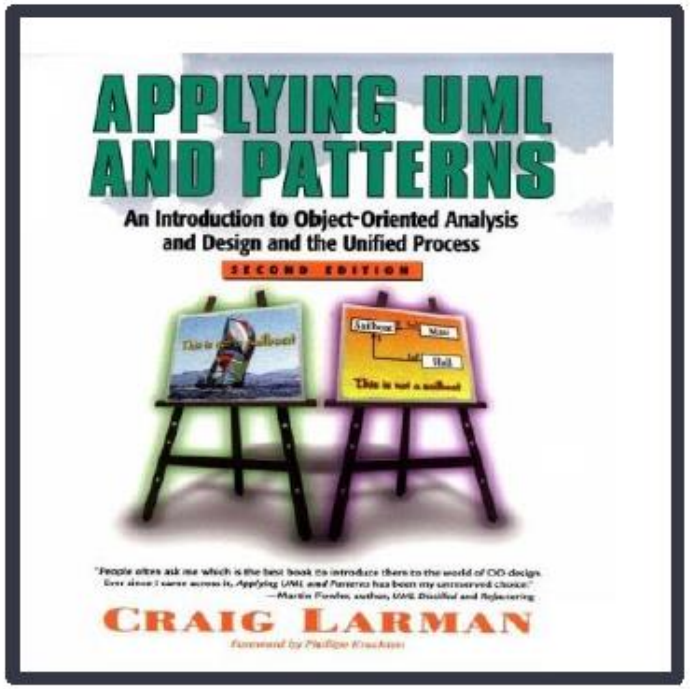
- Uma dúvida recorrente para projetistas OO/UML é sobre a alocação de métodos em classes:
  - Quais métodos criar?
  - Que classes devo ter?
  - Quais classes devem conter quais métodos?

Padrões de Projeto

- Padrões de projetos são utilizados para uniformizar estratégias para criação de estruturas orientadas a objetos, em nível de relacionamento entre classes e em nível de interação por mensagens.







Padrões de Projeto

- Expressam uma solução para um determinado problema em um determinado contexto incluindo:
  - Nome (facilita a abstração e comunicação)
  - Problema que ele resolve
  - Solução para o problema
  - Conselhos sobre sua aplicação em novas situações
  - Discussão sobre consequências de seu uso

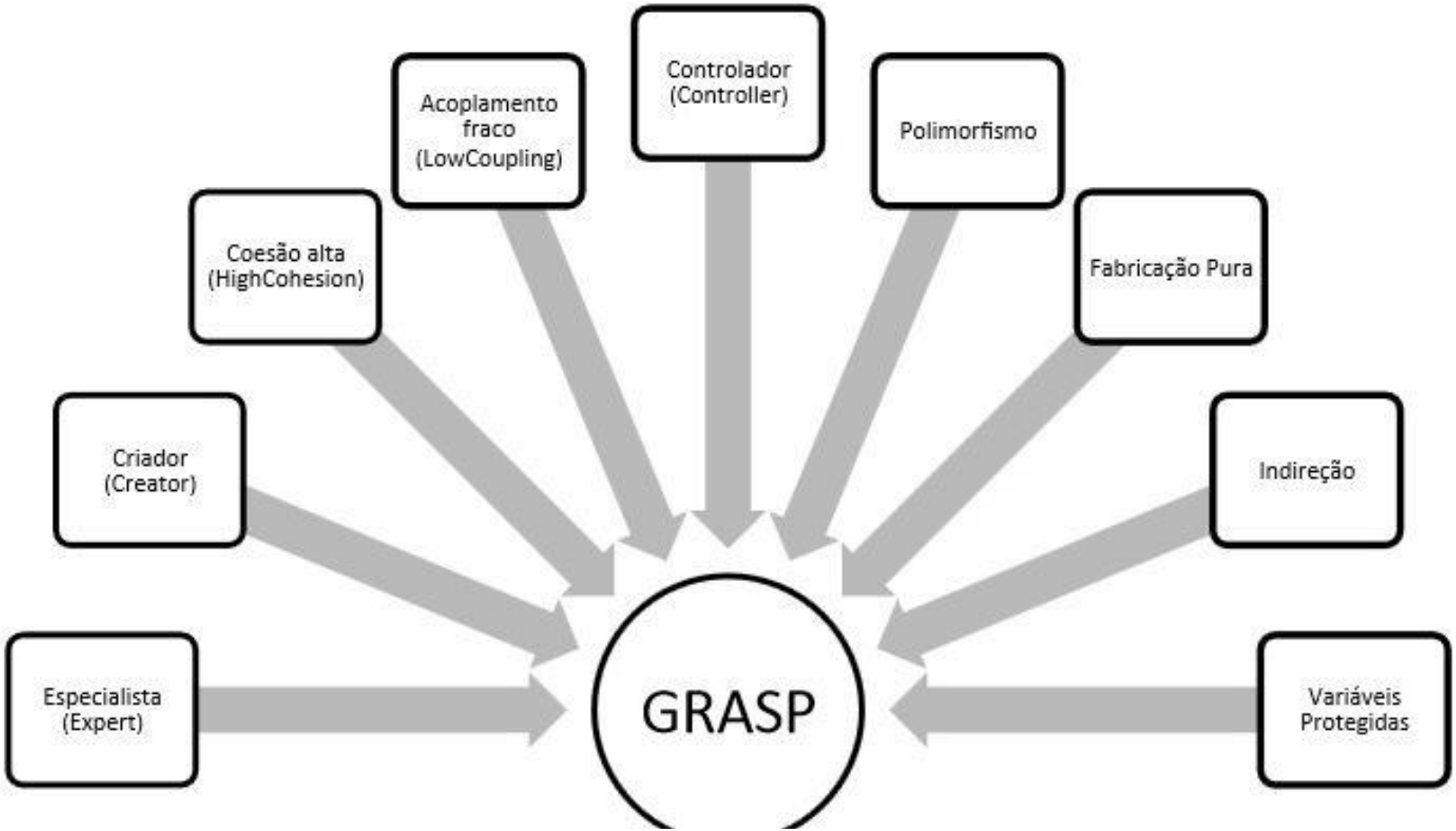
- Criados por desenvolvedores experientes

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.

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*)


Controlador
(Controller)
Acoplamento
fraco
(LowCoupling)
Polimorfismo
Coesão alta
(HighCohesion)
Fabricação Pura
Criador
(Creator)
Indireção
Especialista
(Expert)
GRASP
Variáveis
Protegidas

GRASP

- Padrões básicos
  - Information Expert
  - Creator
  - High Cohesion
  - Low Coupling
  - Controller
- Padrões avançados
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variations

- Padrões GRASP refletem práticas mais pontuais da aplicação de técnicas OO
- Padrões GoF exploram soluções mais específicas
- Padrões GRASP ocorrem na implementação de vários padrões GoF

Padrão Especialista

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

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"

Exemplo

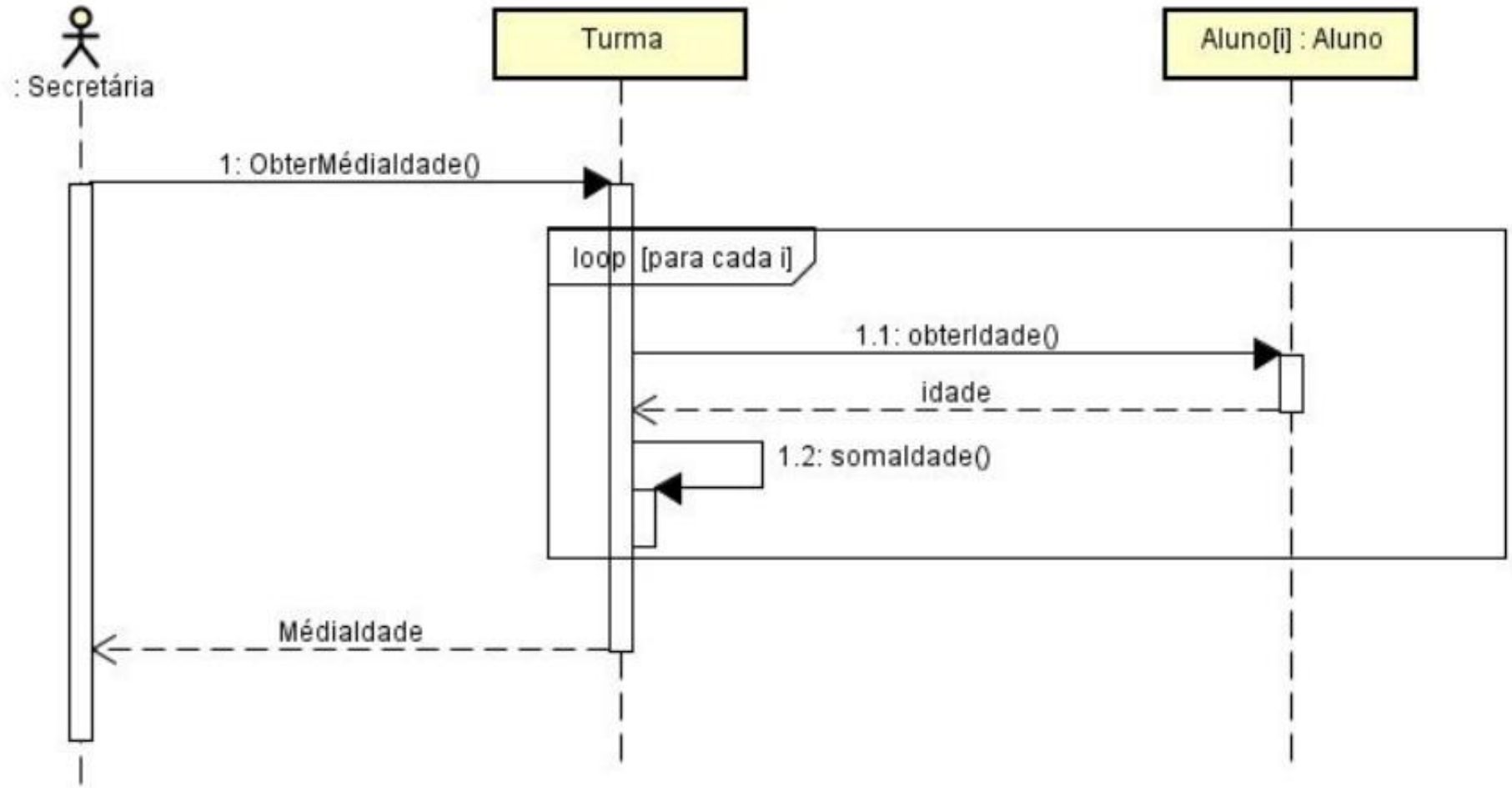
- Questão que enuncia a responsabilidade: *Quem deve ser responsável por conhecer a média de idade dos alunos na turma?*

- A resposta seria Turma? Aluno? Outra classe?

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: *medialdade*

Exemplo


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

Padrão Especialista

Benefícios:

- Mantém encapsulamento (baixo acoplamento)
- Comportamento é distribuído através das classes que tem a informação necessária para cumprir a responsabilidade (alta coesão)

Contra-indicações:

- Viola a separação dos principais interesses
  - Por exemplo: lógica e controle

Padrão Criador

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

Padrão Criador

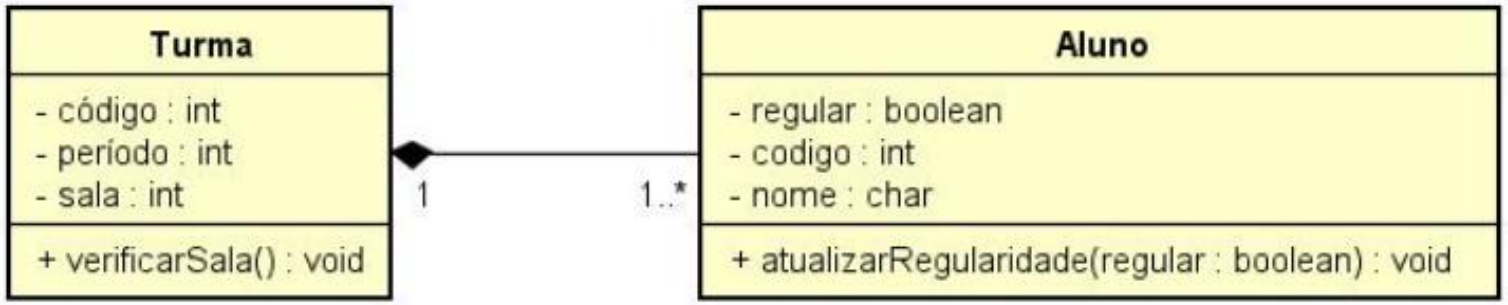
Problema: Quem deve ser responsável pela criação de uma nova instância de uma classe?

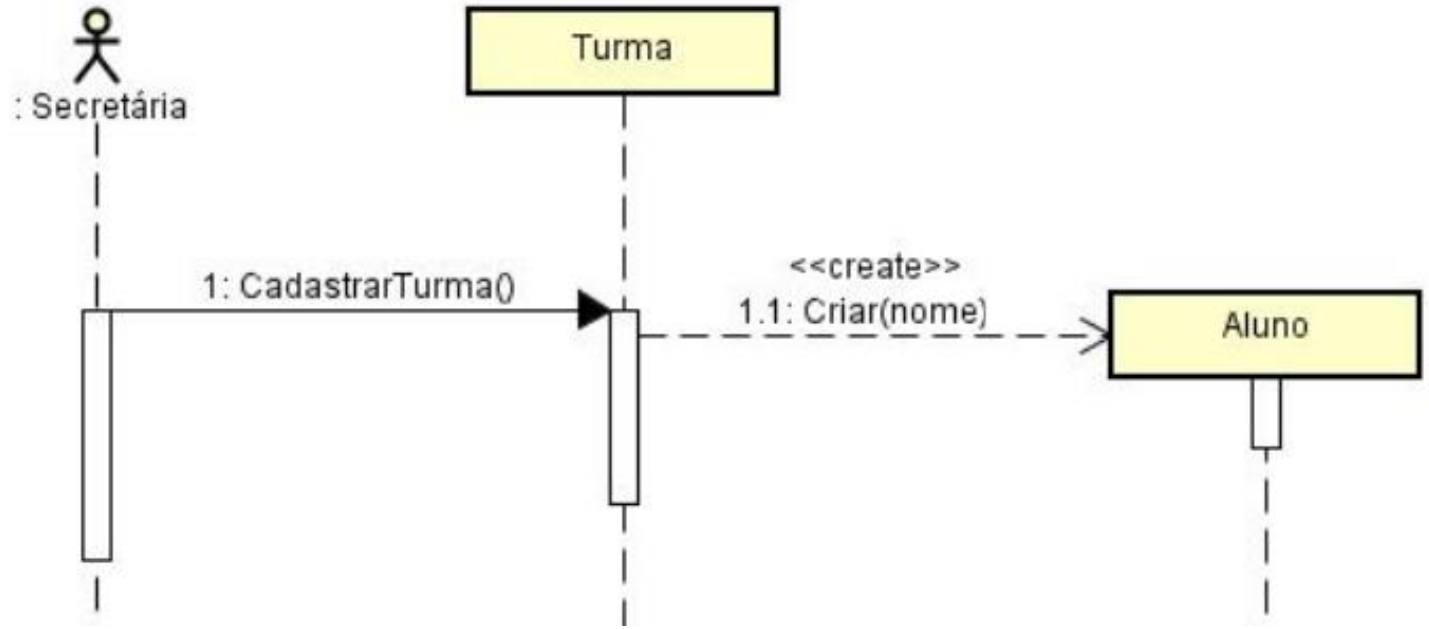
Solução: Classe **B** deve criar instância da classe **A** se pelo menos uma das seguintes condições for verdade (quanto mais melhor)

- 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 Criador

Contra-indicação:

- Não é indicado se a criação de um objeto for uma tarefa complexa
  - Delegar a criação a uma classe auxiliar chamada Fábrica

- Benefícios
  - Acoplamento fraco
    - Não aumenta o acoplamento pois provavelmente a classe criada já é visível à classe criadora devido às associações

Padrão Acoplamento Baixo

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

Relembrando: *O acoplamento é uma medida de quão fortemente uma classe está conectada, possui conhecimento ou depende de outra classe*

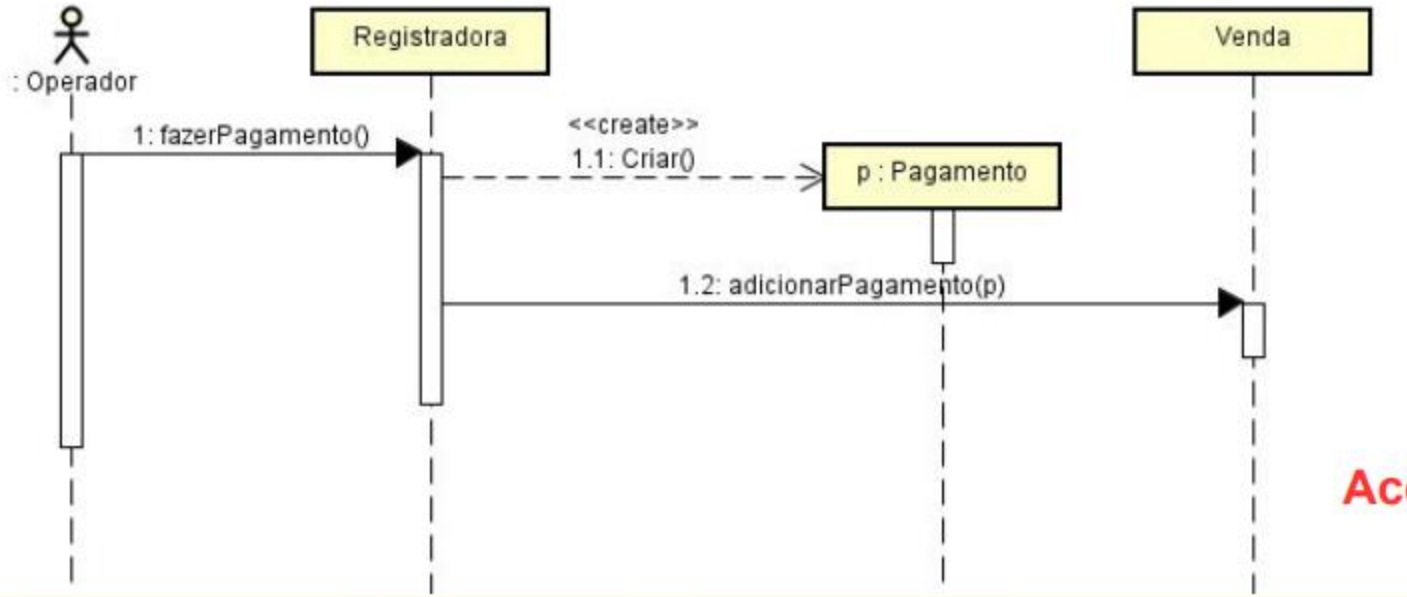
Padrão Acoplamento Baixo

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

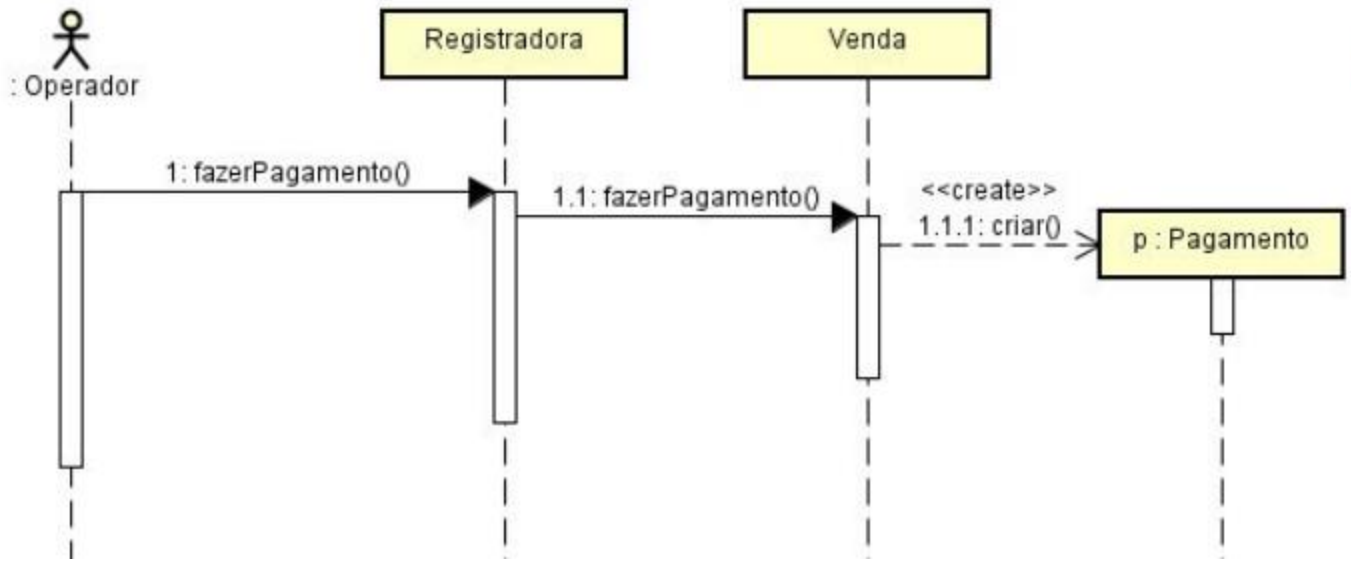
Solução: Atribuir responsabilidade de modo que o acoplamento permaneça baixo. Use esse princípio para avaliar alternativas.

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 Acoplamento Baixo

- A ideia é abstrata, mas o objetivo é: atribuir a responsabilidade de modo que o acoplamento (dependência entre classes) permaneça baixo.

Benefícios

- Responsabilidade de uma classe não é (ou é pouco) afetada por mudanças em outros componentes
- Responsabilidade de uma classe é mais simples de entender isoladamente
- Aumenta a chance de reutilização de uma classe

Padrão Controlador

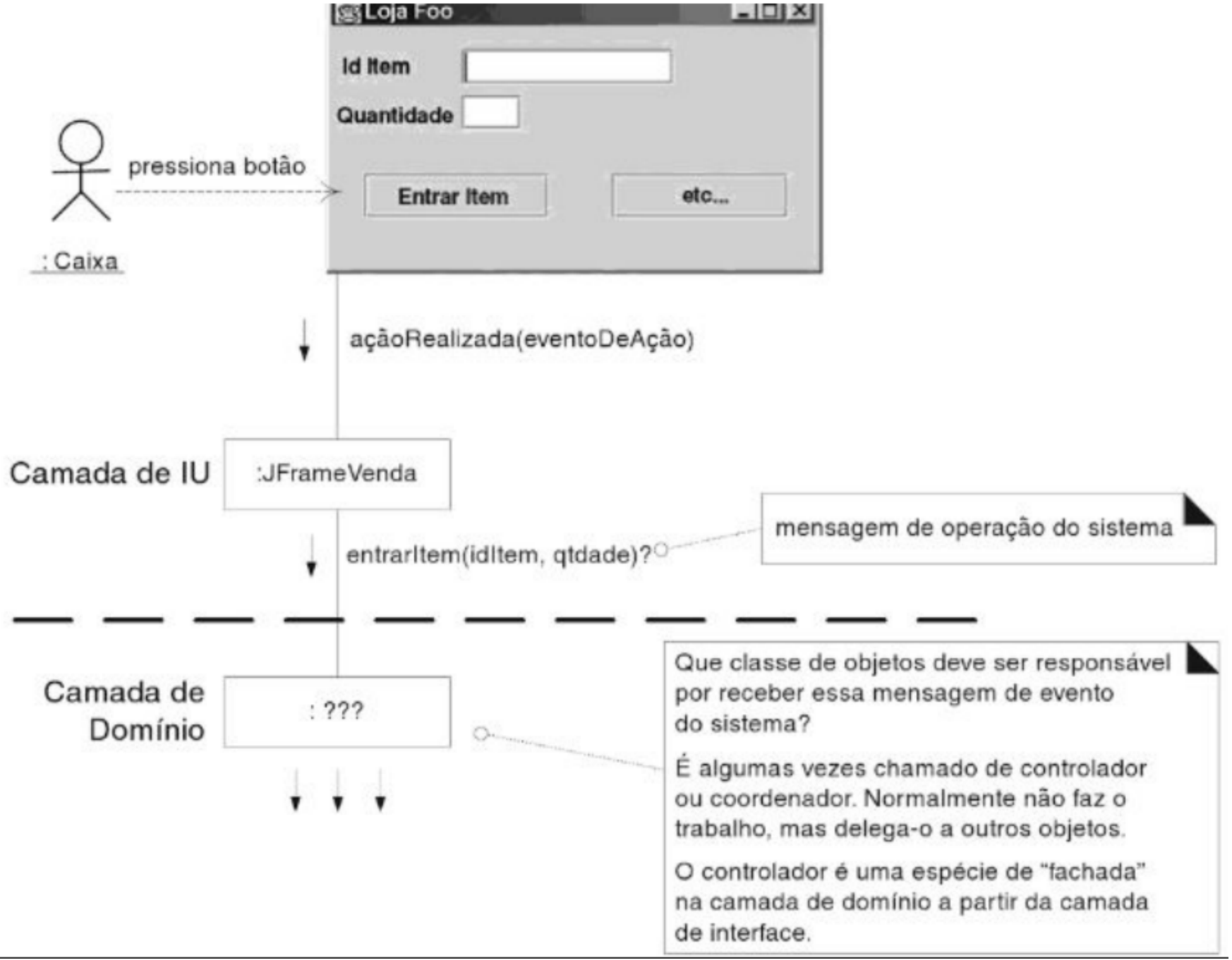
Problema: : Qual é o primeiro objeto, além da camada de Interface com Usuário (IU), que recebe e controla uma operação do sistema?

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.

Padrão Controlador – Outras regras

Atribuir a responsabilidade de tratar um evento do sistema para uma classe “controladora” representando:

- o sistema como um todo (facade controller)
- o negócio ou organização com um todo (facade controller)
- uma coisa ou papel de uma pessoa do mundo real envolvida diretamente com a tarefa (role controller)
- um “tratador” (handler) artificial para todos os eventos de um caso de uso (use-case controller)

Padrão Controlador

- 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"
    - Garante que a lógica da aplicação não seja tratada na camada de interface
  - Oportunidade de raciocinar sobre o estado do caso de uso
    - Garantir que as operações do sistema ocorram em uma sequência válida

Padrão Coesão Alta

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

Padrão Coesão Alta

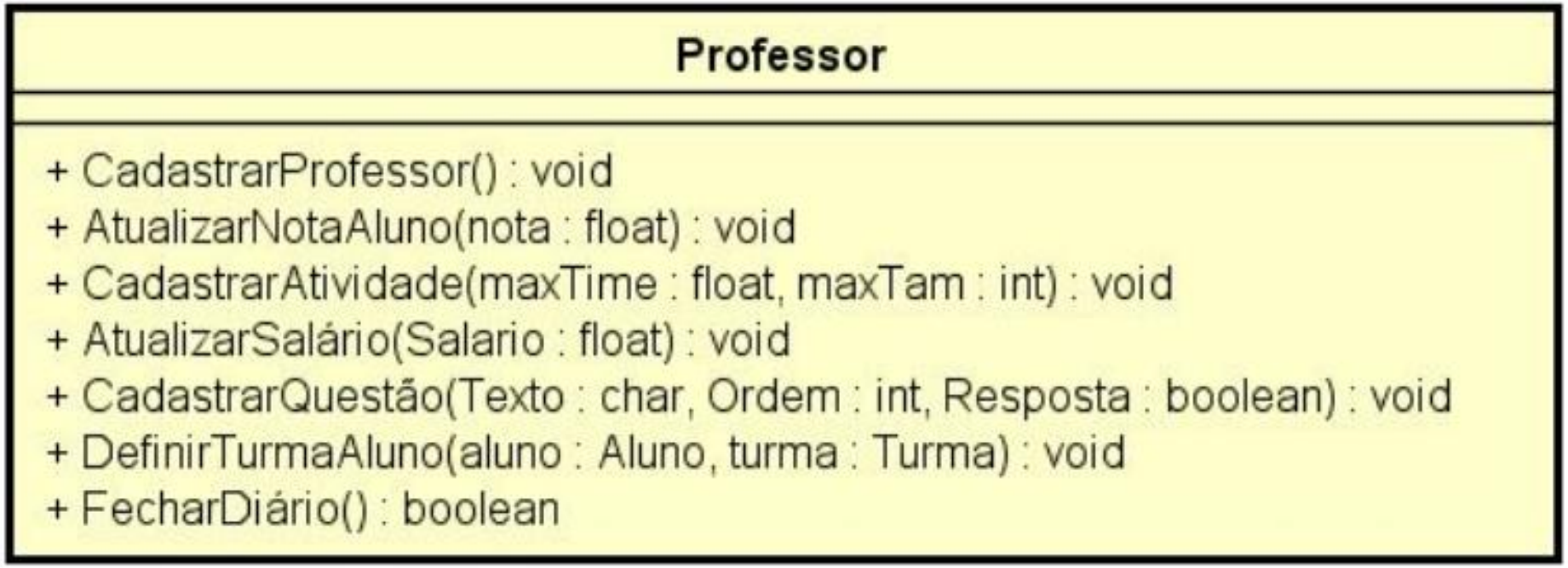
Problema: Como manter os objetos focados, inteligíveis e gerenciáveis e como efeito colateral apoiar o "Baixo Acoplamento"?

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

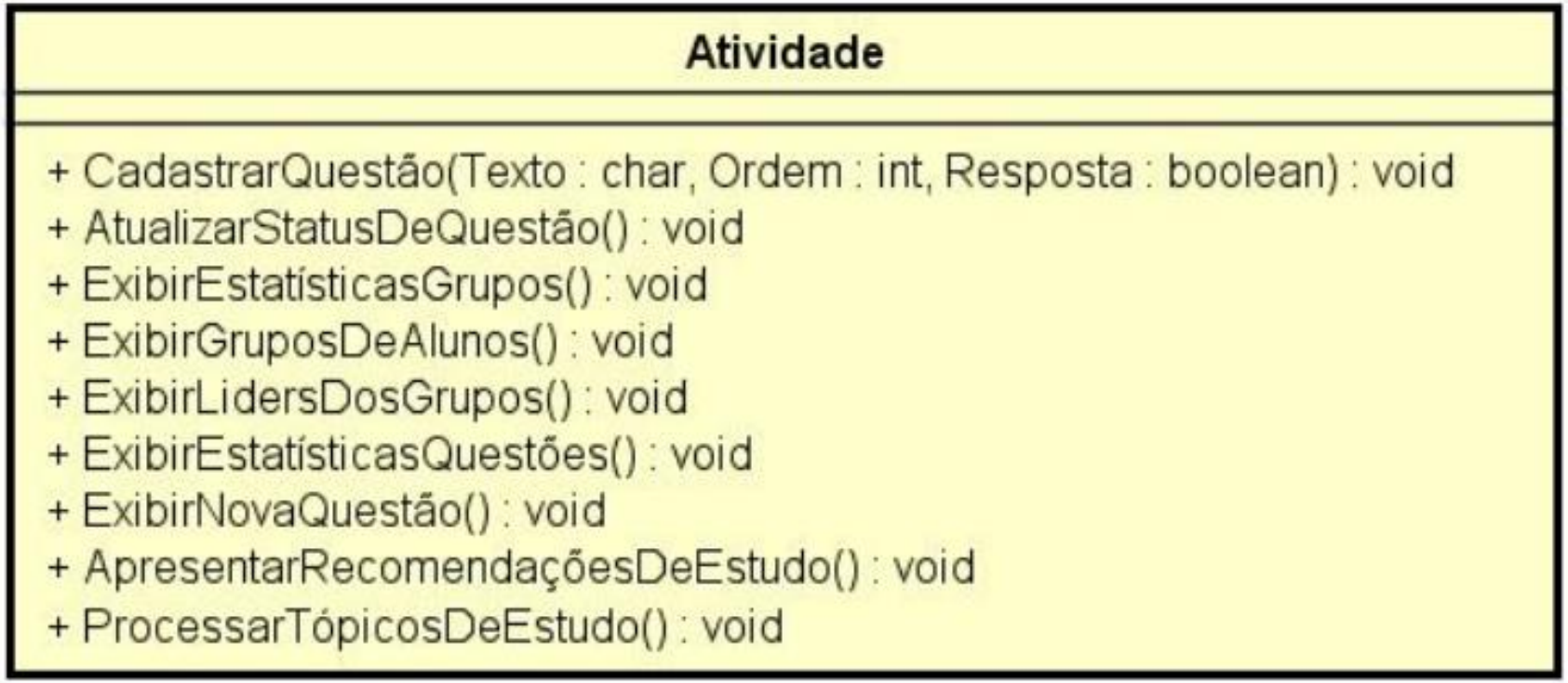
Coesão Muito Baixa

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



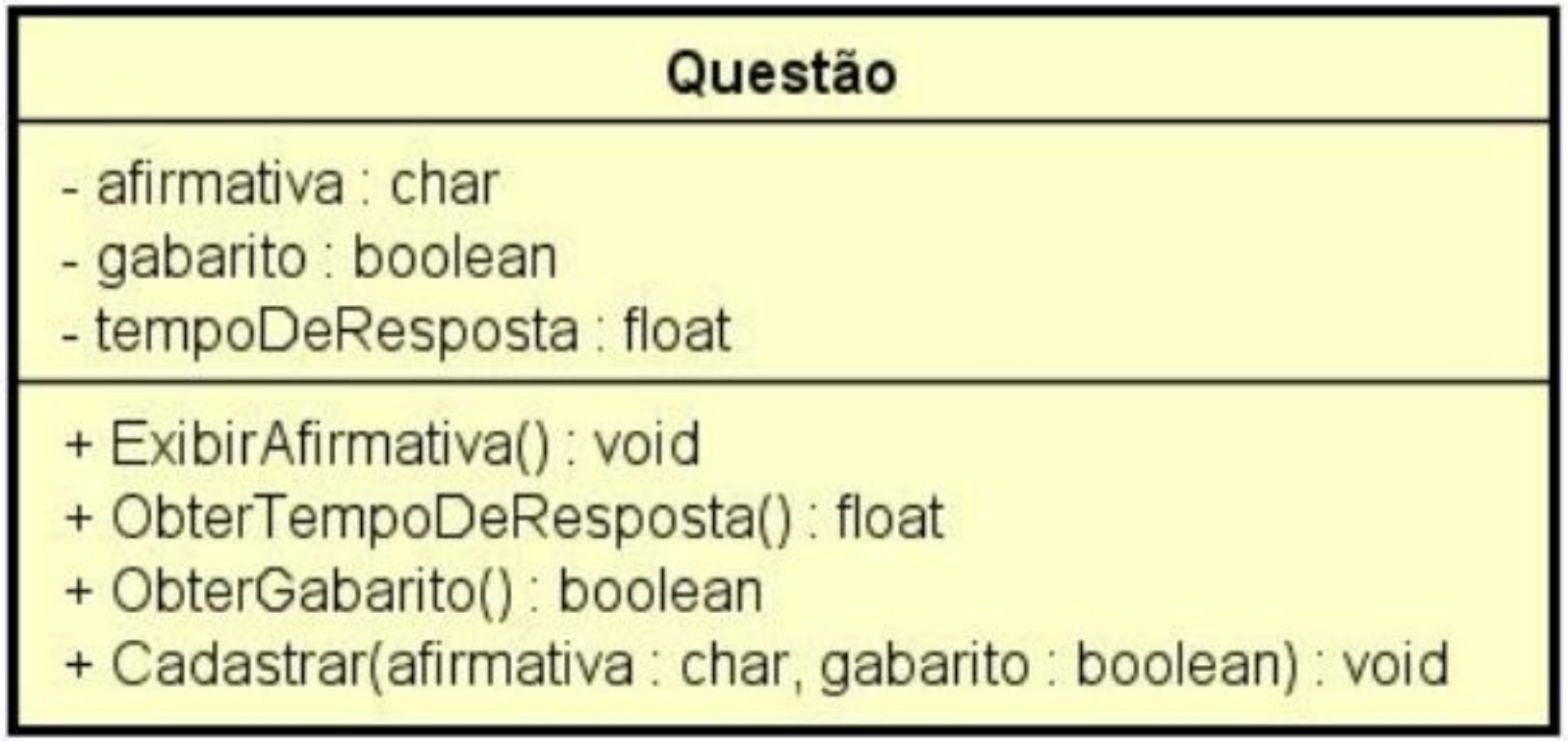
Coesão Baixa

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



Coesão Alta

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



Coesão Moderada

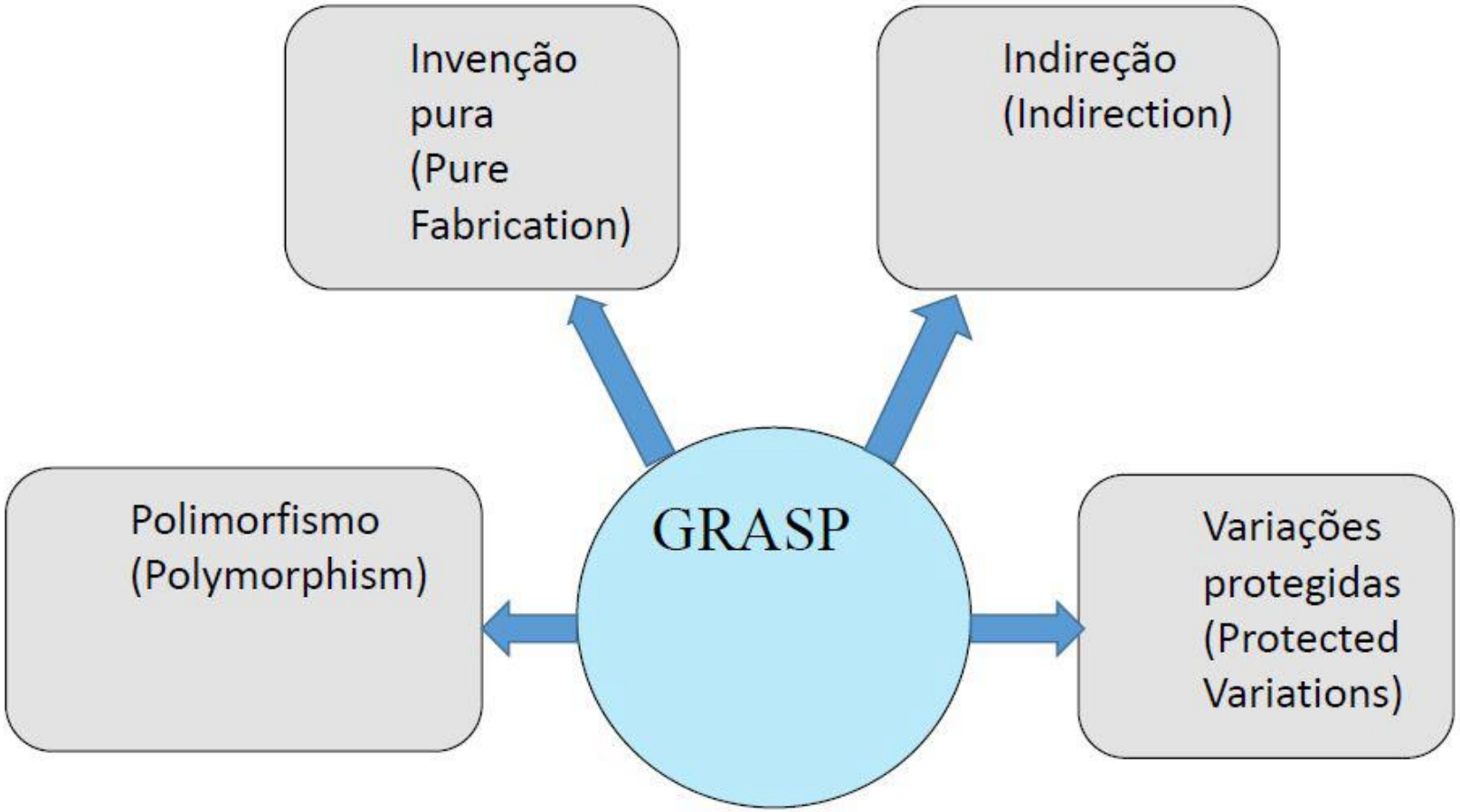
Uma classe tem peso leve e responsabilidade exclusiva em algumas áreas logicamente relacionadas ao conceito da classe, mas não uma com as outras

Padrão Coesão Alta

Benefícios:

- Aumento da clareza e compreensão do projeto
- Simplificação de manutenção
- Baixo acoplamento
- Reuso facilitado

Padrões GRASP Avançados



Padrão Polimorfismo

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

Padrão Polimorfismo

O Problema Sem Polimorfismo (Design Rígido)

Necessidade: Você tem um conjunto de objetos relacionados (por exemplo, Cachorro, Gato, Pássaro) que todos herdam de uma classe base (Animal). Você precisa executar uma ação (emitirSom()), mas o comportamento dessa ação depende do tipo exato do objeto.

Solução Rígida (Antipadrão): Você cria uma função que usa lógica condicional para verificar o tipo de objeto e executa o código apropriado:

```
se (tipoDeObjeto == 'Cachorro') {
    executaSomDoCachorro();
} senao se (tipoDeObjeto == 'Gato') {
    executaSomDoGato();
}
```

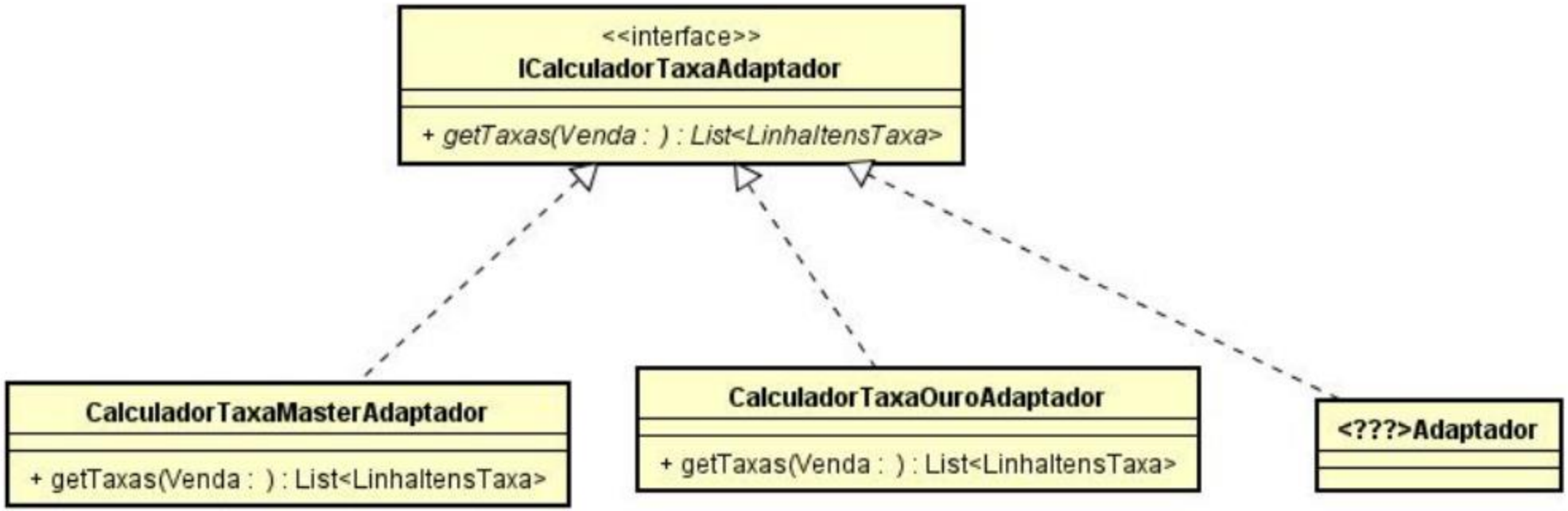
Padrão Polimorfismo

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



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

Padrão Polimorfismo

Quando alternativas ou comportamentos relacionados variam segundo o tipo (classe), deve-se atribuir a responsabilidade pelo comportamento (usando operações polimórficas) aos tipos para os quais o comportamento varia.

Padrão Polimorfismo

Vantagens:

- As extensões exigidas para as novas variações são fáceis de adicionar
- Novas implementações podem ser introduzidas sem afetar os clientes

Contra-indicações

- Não usar polimorfismo para adicionar uma flexibilidade para uma possível futura variação
  - O esforço pode não compensar

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?

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?

Às vezes, durante o projeto é preciso atribuir responsabilidades que não são encaixam naturalmente em nenhuma das classes conceituais

Padrão Invenção Pura

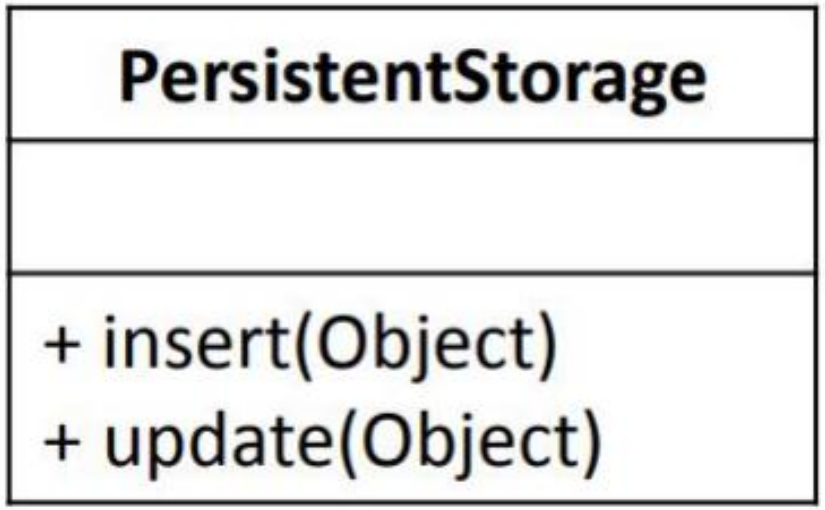
Problema: Qual objeto deve ter a responsabilidade quando não se quer violar a 'Coesão Alta' e o 'Acoplamento Baixo' ou outros objetivos, mas as soluções oferecidas pelo Especialista (por exemplo) não são apropriadas?

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

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



Padrão Invenção Pura

Vantagens:

- coesão alta é favorecida
- potencial de reutilização pode aumentar devido à presença de classes de Invenção Pura refinadas

Contra-indicações:

- uso extremo – funções simples se tornam objetos

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?

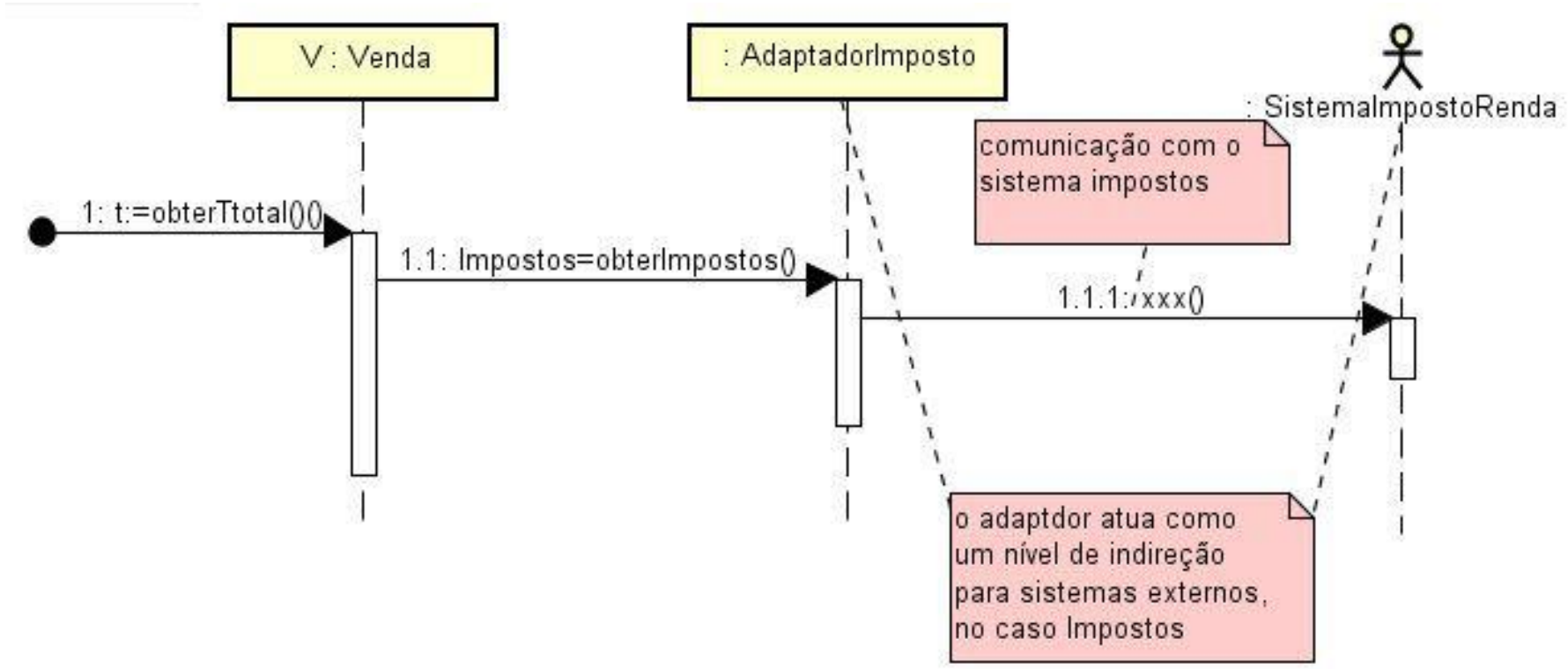
Solução: Atribuir a responsabilidade de ser o mediador entre outros componentes ou serviços a um objeto intermediário, para que eles não sejam diretamente acoplados

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?

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