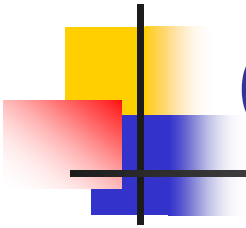
Diagramas de Comunicação e Padrões GRASP

Análise e Projeto Orientados a Objetos

Profa. Rosana T. Vaccare Braga

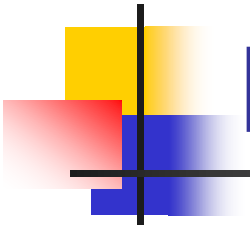
(alguns slides foram obtidos e/ou adaptados a partir do material fornecido pelo Prof. Raul Wazlawick)



O que vimos até agora

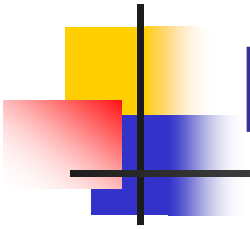
- Diagramas de Caso de Uso
- Casos de uso no formato completo (abstrato ou concreto)
- Modelo Conceitual
- Diagramas de Seqüência do Sistema

  (Usados em Cenários)
- Contratos de Operações
- Notação dos Diagramas de Colaboração/Comunicação



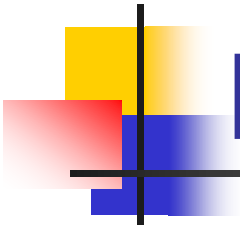
Padrões

- Desenvolvedores de software experientes criaram um repertório de princípios gerais e boas soluções para guiar a construção de software
- Essas soluções foram descritas em um formato padronizado (nome, problema, solução) e podem ser usadas em outros contextos
- Surgiram com base no trabalho do arquiteto Christopher Alexander, 1977
- Ganharam impulso após a publicação dos Design Patterns – Gamma e outros – GoF - 1994



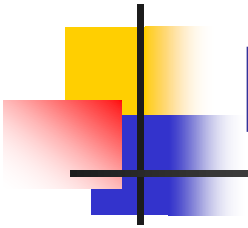
Padrões

- Padrões usualmente não contêm novas idéias
  - organizam conhecimentos e princípios existentes, testados e consagrados
- **Padrão** é uma descrição nomeada de um problema e uma solução, que pode ser aplicado em novos contextos



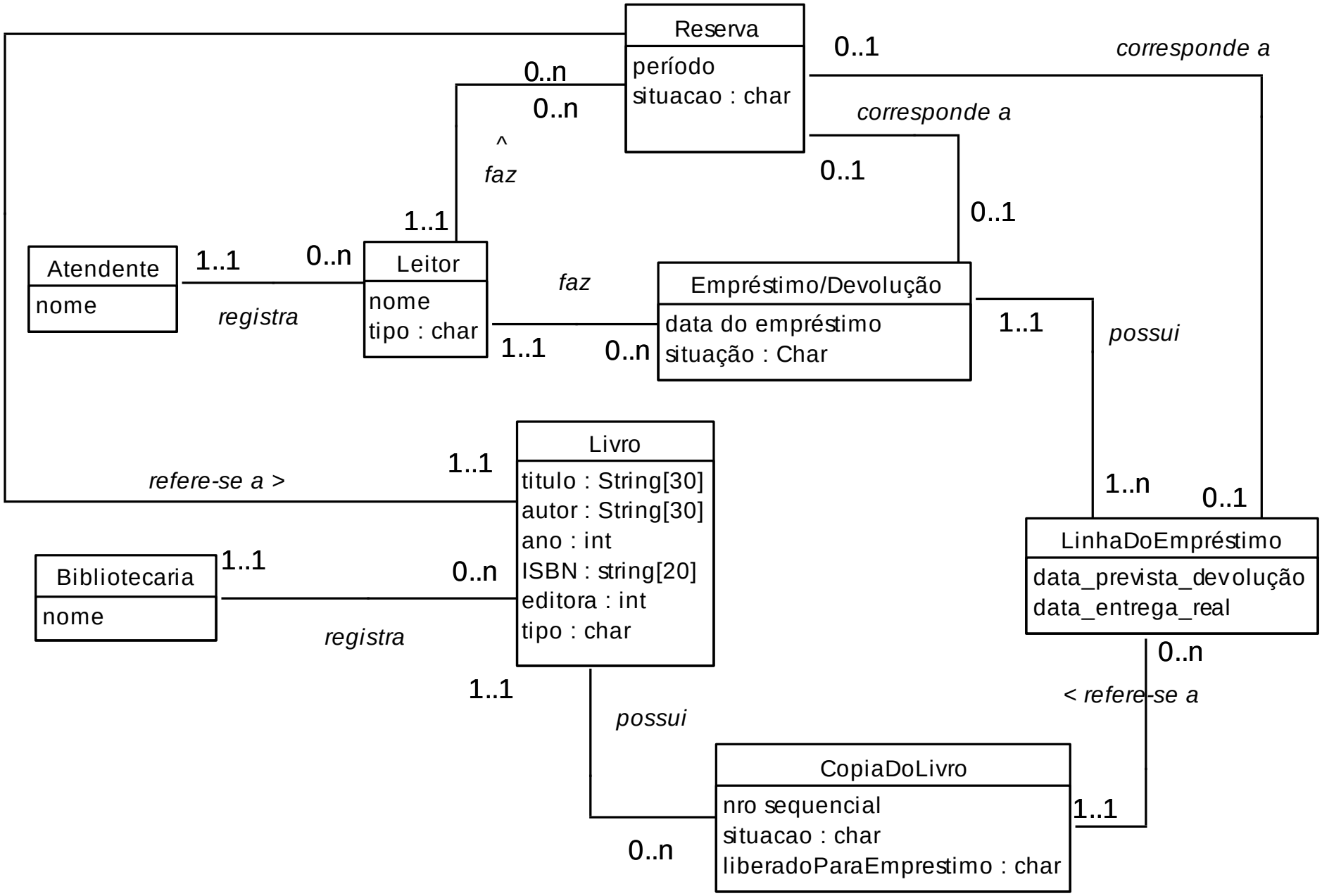
Padrões GRASP

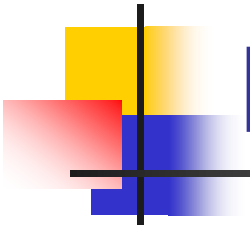
- GRASP = ***G****eneral* ***R****esponsibility* ***A****ssignment* ***S****oftware* ***P****atterns*
- Descrevem princípios fundamentais de atribuição de responsabilidade a objetos
- Alguns padrões GRASP principais:
  - Especialista (*Expert*)
  - Criador (*Creator*)
  - Coesão alta (*High Cohesion*)
  - Acoplamento fraco (*Low Coupling*)
  - Controlador (*Controller*)



Especialista

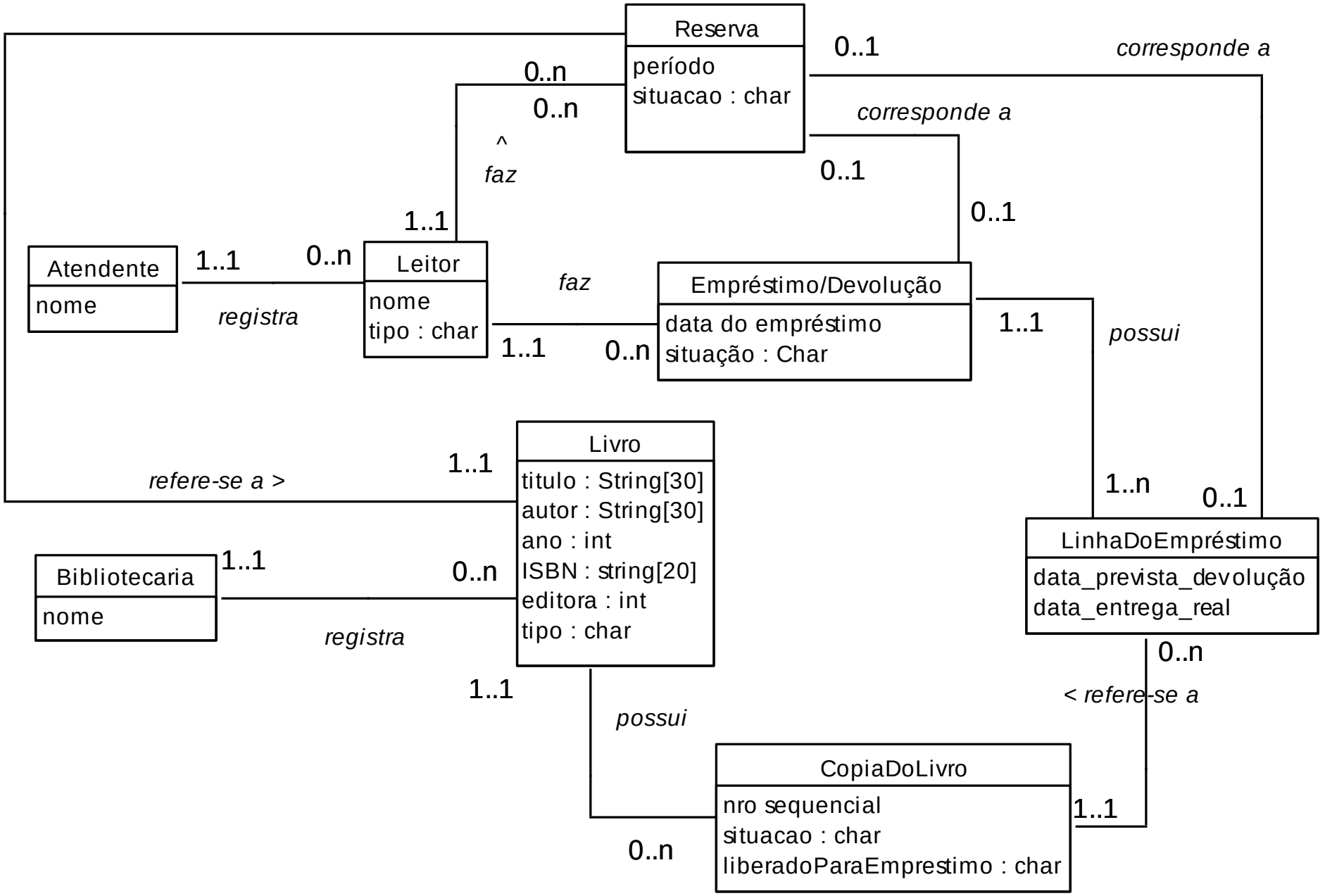
- Problema: qual é o princípio mais básico de atribuição de responsabilidades a objetos ?
- Solução: Atribuir responsabilidade ao especialista da informação.
- Exemplo: no sistema de biblioteca, quem seria o responsável por calcular a data de devolução de um livro?

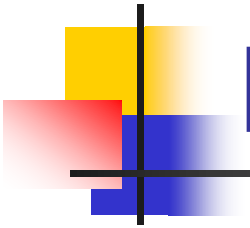

Reserva
período
situacao : char
0..1
corresponde a
0..n
0..n
corresponde a
^
faz
0..1
1..1
0..1
Atendente
nome
1..1
registra
0..n
Leitor
nome
tipo : char
faz
Empréstimo/Devolução
data do empréstimo
situação : Char
1..1
0..n
1..1
possui
Livro
titulo : String[30]
autor : String[30]
ano : int
ISBN : string[20]
editora : int
tipo : char
1..1
refere-se a >
1..n
0..1
LinhaDoEmpréstimo
data_prevista_devolução
data_entrega_real
Bibliotecaria
nome
1..1
registra
0..n
0..n
< refere-se a
1..1
possui
CopiaDoLivro
nro sequencial
situacao : char
liberadoParaEmprestimo : char
1..1
0..n



Especialista

- A data de devolução ficará armazenada no atributo data_prevista_devolução do objeto LinhaDoEmprestimo
- Mas quem possui conhecimentos necessários para calculá-la?

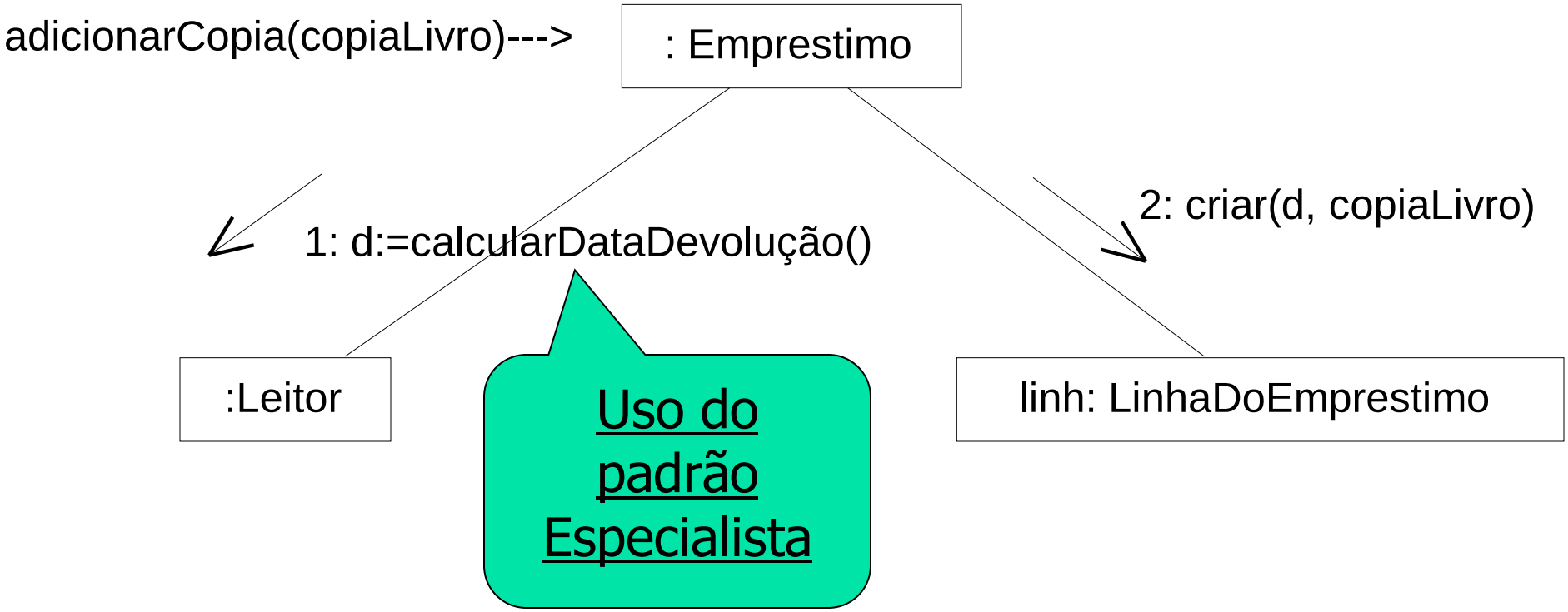

Reserva
período
situacao : char
0..1
corresponde a
0..n
0..n
^
faz
corresponde a
0..1
0..1
1..1
Atendente
nome
1..1
0..n
Leitor
nome
tipo : char
faz
Empréstimo/Devolução
data do empréstimo
situação : Char
1..1
possui
registra
1..1
0..n
Livro
titulo : String[30]
autor : String[30]
ano : int
ISBN : string[20]
editora : int
tipo : char
1..1
refere-se a >
1..n
0..1
LinhaDoEmpréstimo
data_prevista_devolução
data_entrega_real
Bibliotecaria
nome
1..1
0..n
registra
0..n
< refere-se a
1..1
possui
CopiaDoLivro
nro sequencial
situacao : char
liberadoParaEmprestimo : char
1..1
0..n



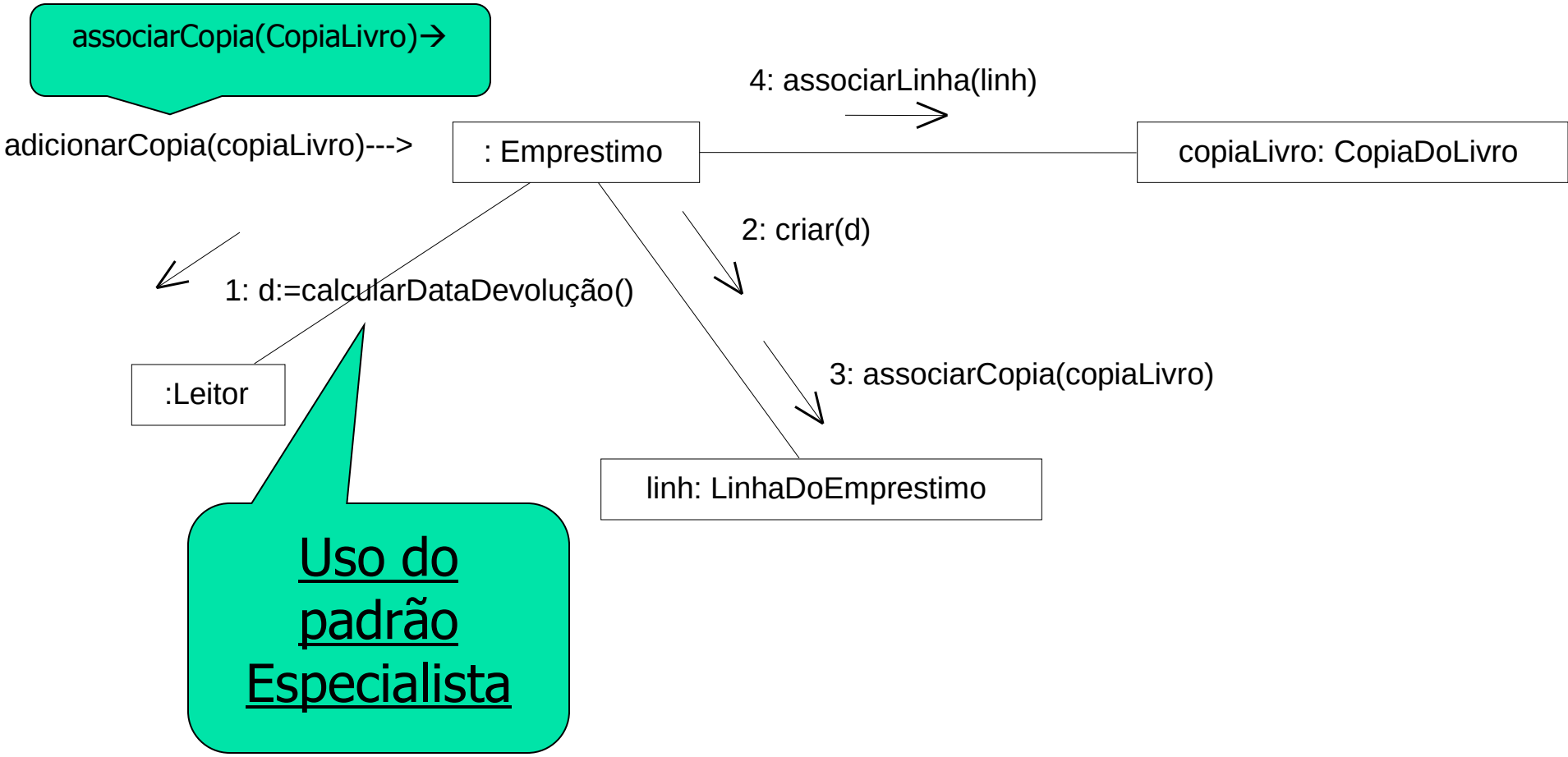
Especialista

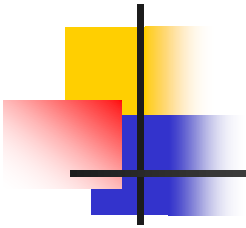
- Pelo padrão especialista, Leitor deve receber esta atribuição, pois conhece o tipo de Leitor (por exemplo, aluno de graduação, aluno de pós-graduação, professor, etc), que é utilizado para calcular a data em que o livro deve ser devolvido

Especialista



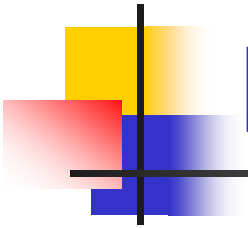
Especialista – Alternativa com mais detalhes





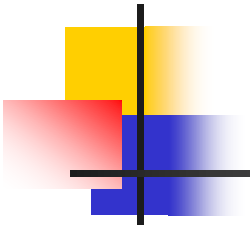
Especialista

- Onde procurar pela classe especialista?
  - Começar pelas classes já estabelecidas durante o projeto
  - Se não encontrar, utilizar o Modelo Conceitual
- Lembrar que existem especialistas parciais que colaboram numa tarefa
  - informação espalhada $\rightarrow$ comunicação via mensagens
- Existe uma analogia com o mundo real



Especialista

- Benefícios:
  - Mantém encapsulamento → favorece o acoplamento fraco
  - O comportamento fica distribuído entre as classes que têm a informação necessária (classes "leves") → favorece alta coesão
  - Favorece o reuso
- Contra-indicações
  - contra indicado quando aumenta acoplamento e reduz coesão
  - Ex: quem é responsável por salvar um Empréstimo no banco de dados?

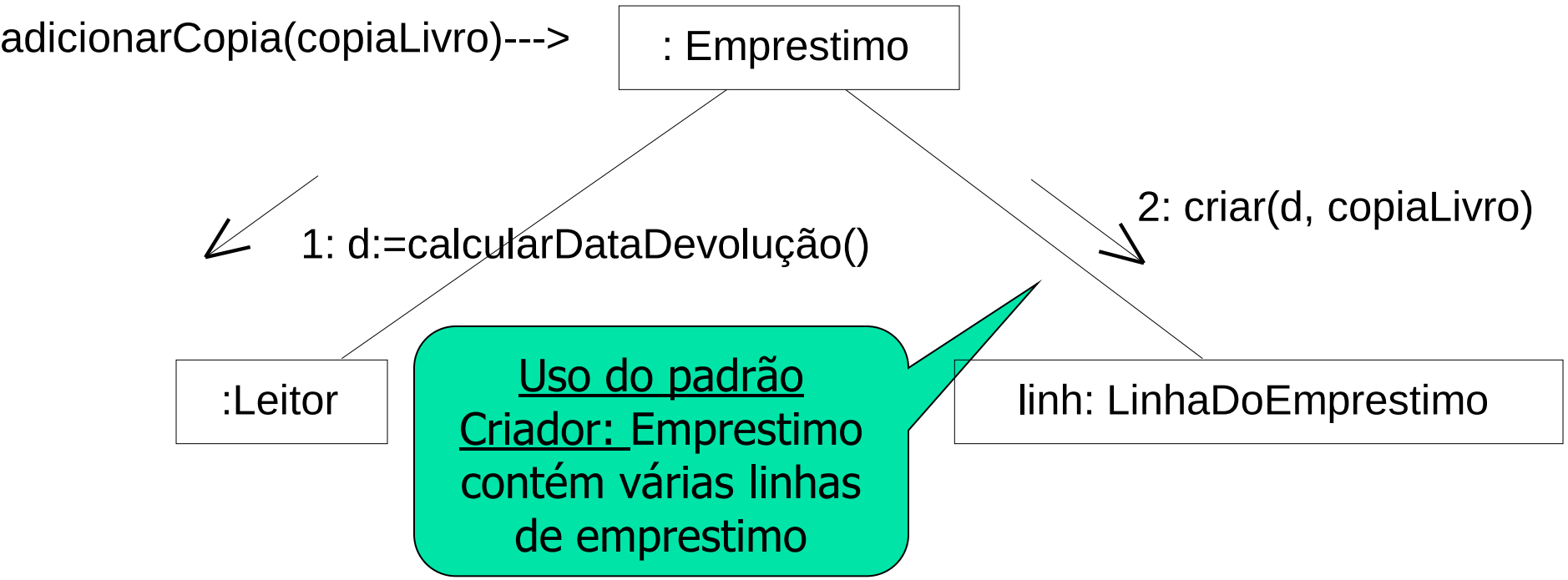


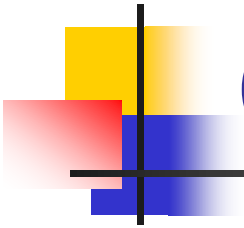
Criador

- Problema: Quem deveria ser responsável pela criação de uma nova instância de alguma classe ?
- Solução: atribua à classe B a responsabilidade de criar uma nova instância da classe A se uma das seguintes condições for verdadeira:
  - B agrega objetos de A
  - B contém objetos de A
  - B registra objetos de A
  - B usa objetos de A
  - B tem os valores iniciais que serão passados para objetos de A, quando de sua criação

Criador

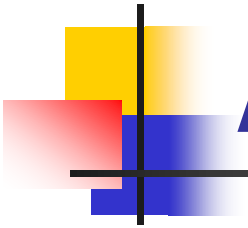
- Exemplo: No sistema da Biblioteca, quem é responsável pela criação de um *ItemDeEmprestimo* ?





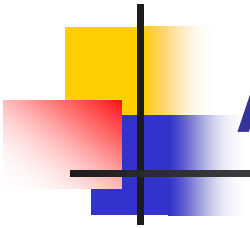
Criador

- Discussão
  - objetivo do padrão: definir como criador o objeto que precise ser conectado ao objeto criado em algum evento
    - escolha adequada favorece acoplamento fraco
  - objetos agregados, contêineres e registradores são bons candidatos à responsabilidade de criar outros objetos
  - algumas vezes o candidato a criador é o objeto que conhece os dados iniciais do objeto a ser criado



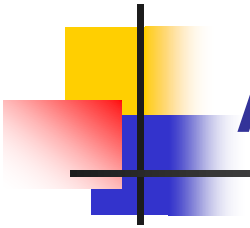
Acoplamento

- Acoplamento: dependência entre elementos (classes, subsistemas,...), normalmente resultante de colaboração para atender a uma responsabilidade
- O acoplamento mede o quanto um objeto está conectado a, tem conhecimento de, ou depende de outros objetos
  - acoplamento fraco (ou baixo) – um objeto não depende de muitos outros
  - acoplamento forte (ou alto) – um objeto depende de muitos outros



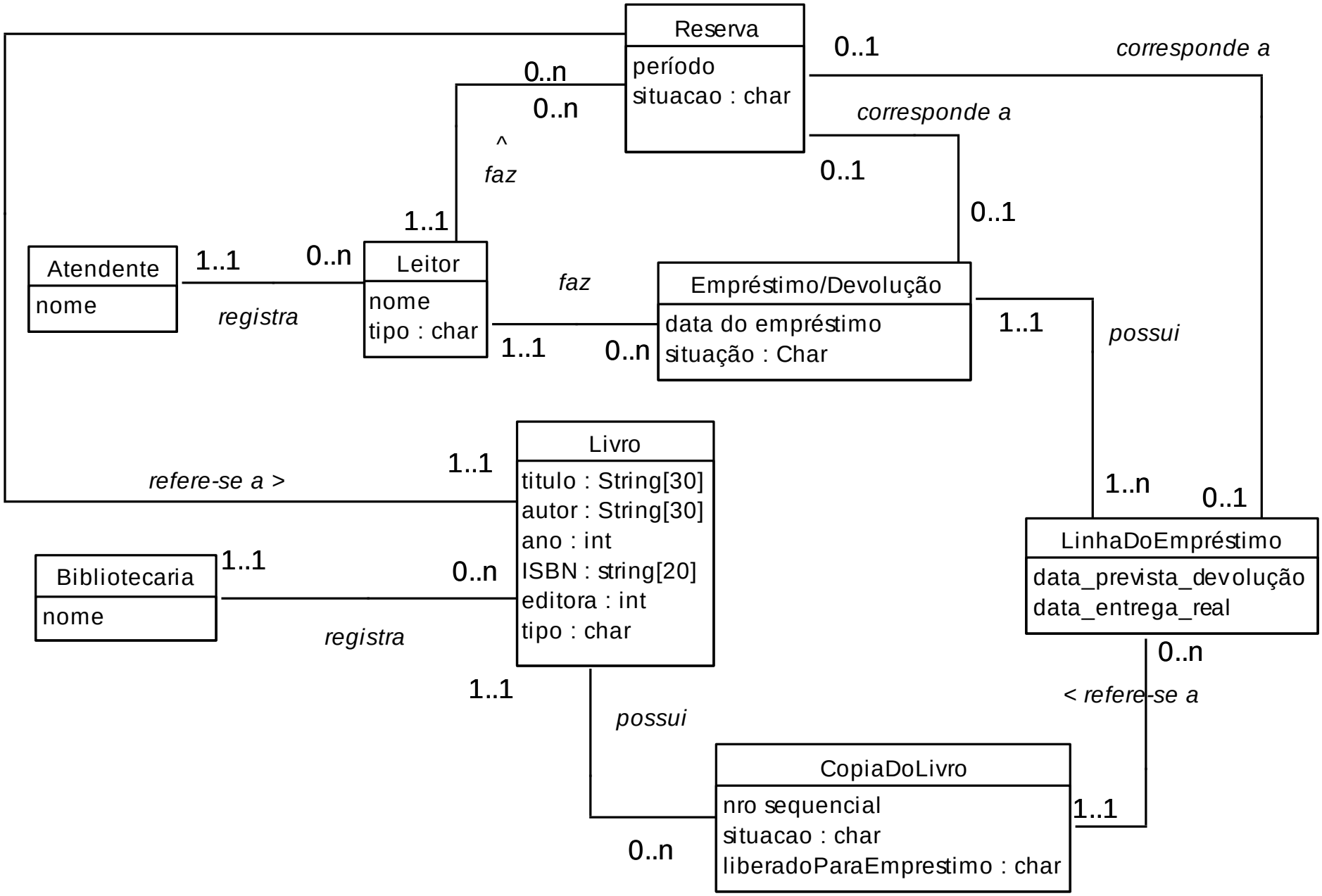
Acoplamento

- Problemas do acoplamento alto:
  - mudanças em classes interdependentes forçam mudanças locais
  - dificulta a compreensão do objetivo de cada classe
  - dificulta a reutilização

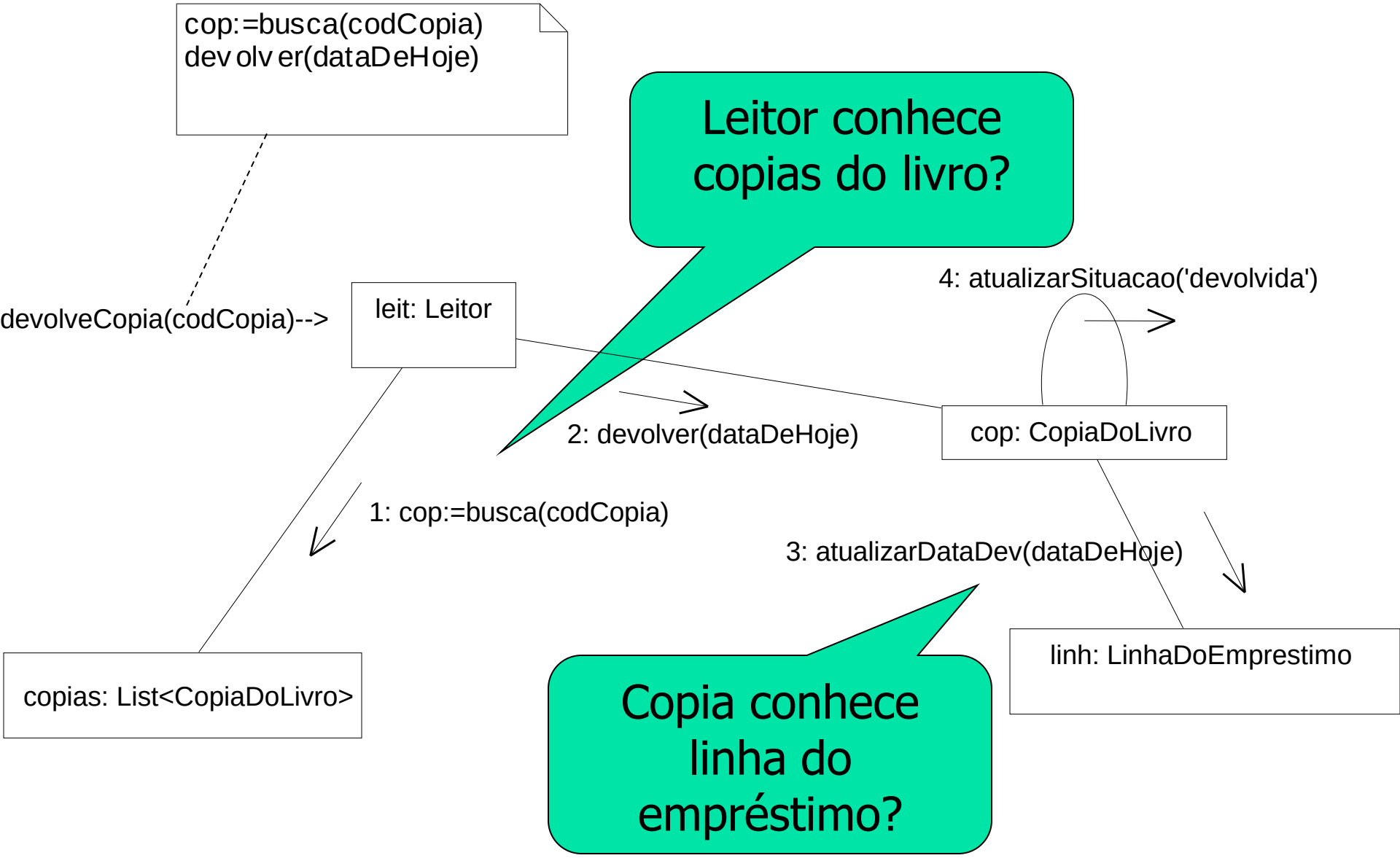


Acoplamento Fraco

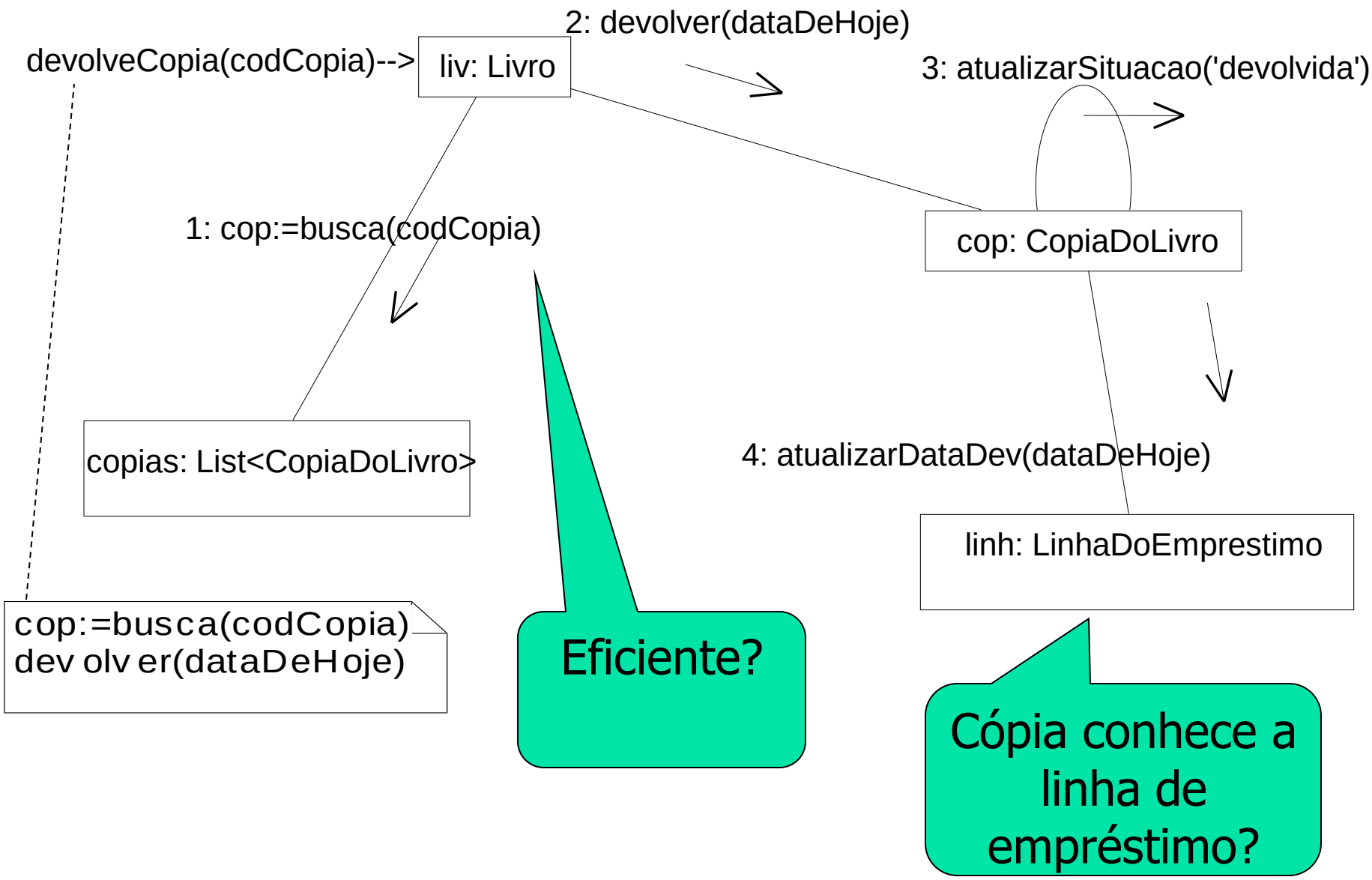
- Problema: como favorecer a baixa dependência e aumentar a reutilização ?
- Solução: Atribuir responsabilidade de maneira que o acoplamento permaneça baixo.
- Exemplo: No sistema de biblioteca, suponha que queremos realizar a devolução da cópia do livro. Qual classe deve ser responsável por essa tarefa?


Reserva
período
situacao : char
0..1
corresponde a
0..n
0..n
corresponde a
^
faz
0..1
1..1
0..1
Atendente
nome
1..1
registra
0..n
Leitor
nome
tipo : char
faz
Empréstimo/Devolução
data do empréstimo
situação : Char
1..1
0..n
1..1
possui
Livro
titulo : String[30]
autor : String[30]
ano : int
ISBN : string[20]
editora : int
tipo : char
1..1
refere-se a >
1..n
0..1
LinhaDoEmpréstimo
data_prevista_devolução
data_entrega_real
Bibliotecaria
nome
1..1
registra
0..n
0..n
< refere-se a
1..1
possui
CopiaDoLivro
nro sequencial
situacao : char
liberadoParaEmprestimo : char
1..1
0..n

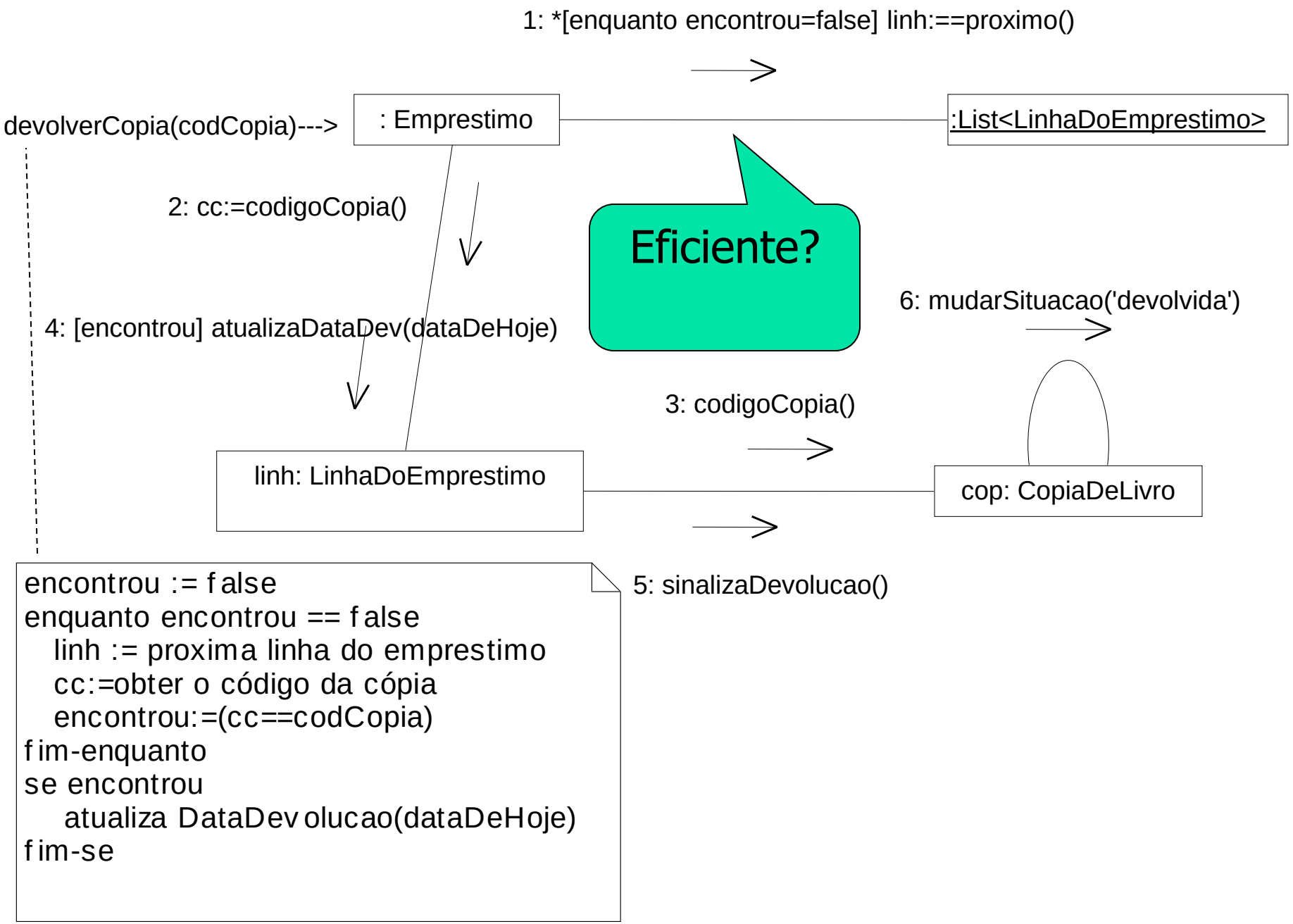
Projeto 1: responsabilidade atribuída ao *Leitor*



Projeto 2: responsabilidade atribuída ao *Livro*

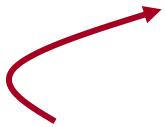


Projeto 3: responsabilidade atribuída ao *Empréstimo*

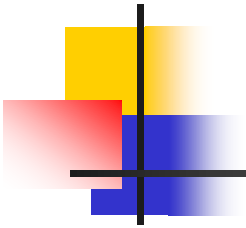


Qual projeto é melhor?

- Qual dos projetos anteriores favorece o acoplamento fraco?
  - projeto 1 e 2 – acoplamento aumenta (entre cópia do livro e linha do empréstimo, acoplamento entre leitor e cópia do livro)
  - projeto 3 – não aumenta acoplamento

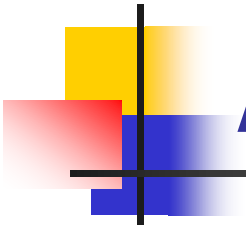


PREFERÍVEL



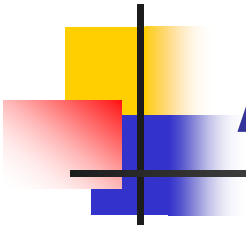
Formas de Acoplamento

- Um objeto tem um atributo que referencia um objeto de outra classe
- Um objeto tem um método que referencia um objeto de outra classe
  - parâmetro, variável local ou retorno
- Um objeto invoca os serviços de um objeto de outra classe
- Uma classe é subclasse de outra, direta ou indiretamente



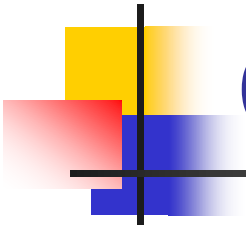
Acoplamento Fraco

- Discussão:
  - Acoplamento fraco → classes mais independentes
    - reduz impacto de mudanças
    - favorece reuso de classes
  - Considerado em conjunto com outros padrões
  - Extremo de acoplamento fraco não é desejável
    - fere princípios da tecnologia de objetos – comunicação por mensagens
    - projeto pobre: objetos inchados e complexos, responsáveis por muito trabalho → baixa coesão



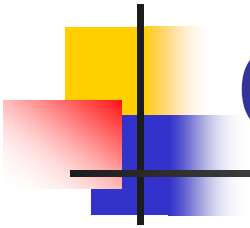
Acoplamento Fraco

- Discussão:
  - dica: concentre-se em reduzir o acoplamento em pontos de evolução ou de alta instabilidade do sistema
- Benefícios:
  - classes são pouco afetadas por mudanças em outras partes
  - classes são simples de entender isoladamente
  - conveniente para reutilização



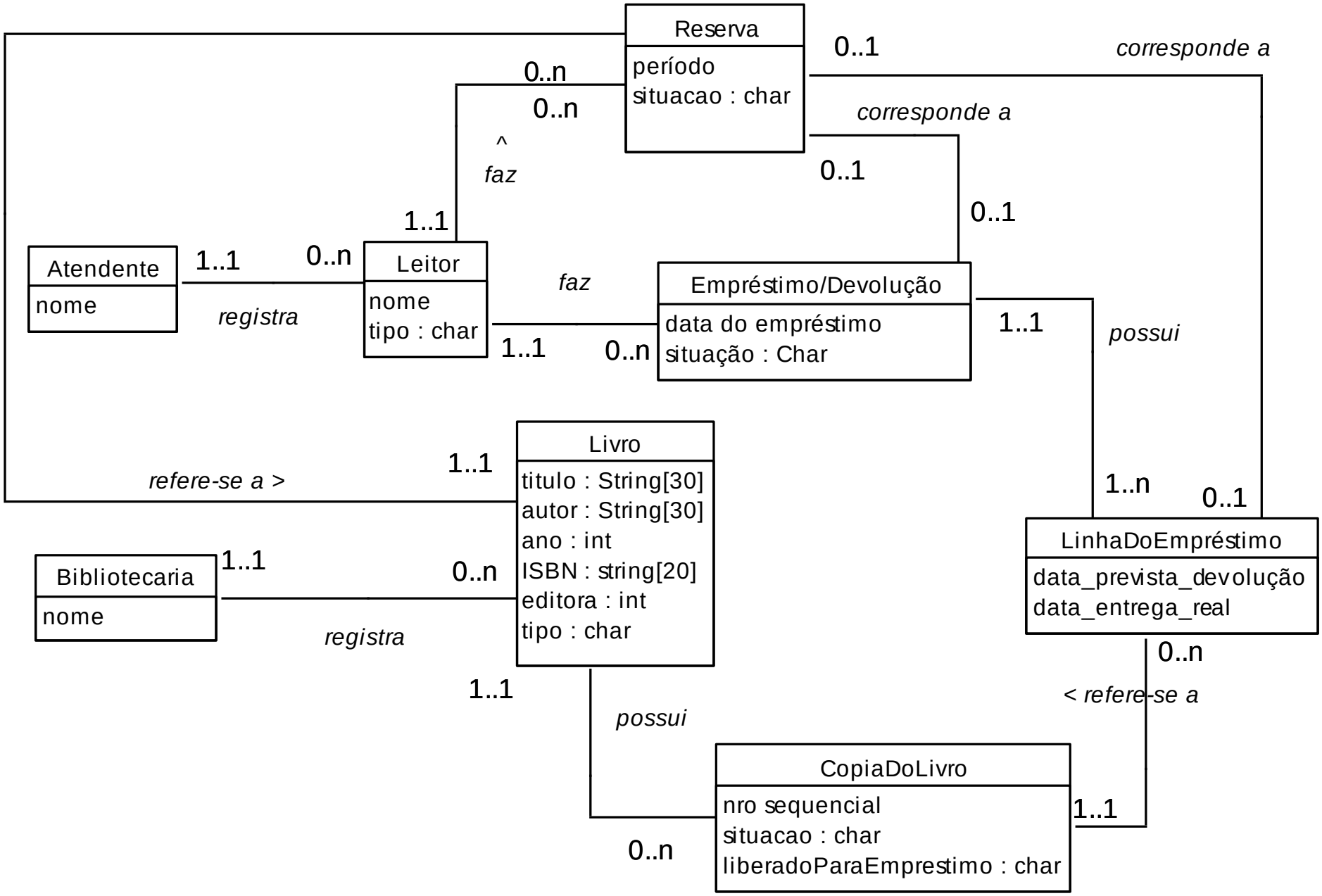
Coesão

- **Coesão** mede o quanto as responsabilidades de um elemento (classe, objeto, subsistema,...) são fortemente focalizadas e relacionadas
- Objeto com Coesão Alta → objeto cujas responsabilidades são altamente relacionadas e que não executa um volume muito grande de trabalho
- Objeto com Coesão Baixa → objeto que faz muitas tarefas não relacionadas ou executa muitas tarefas
  - difícil de compreender, reutilizar e manter
  - constantemente afetado por mudanças

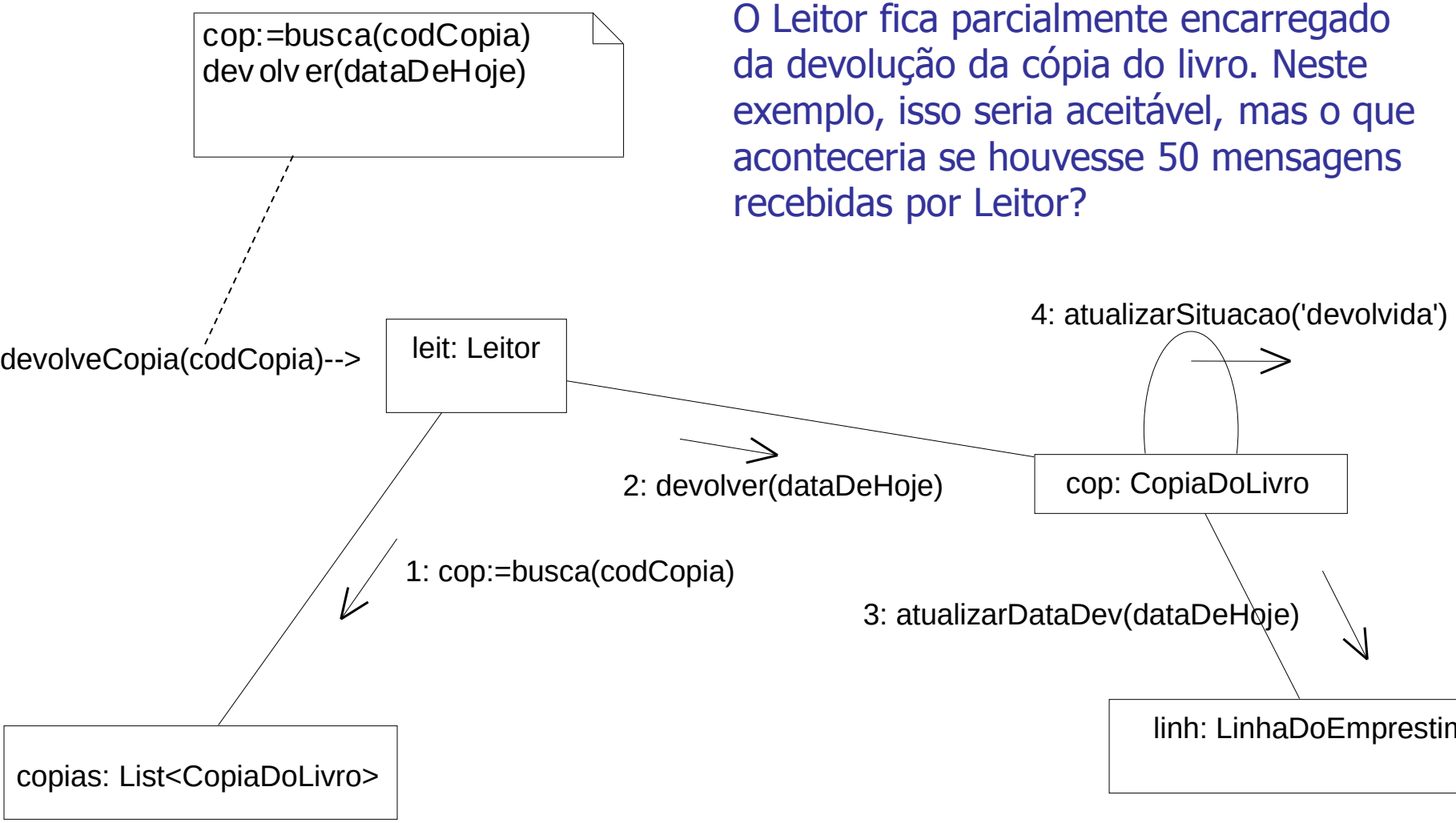


Coesão Alta

- Problema: Como manter a complexidade sob controle?
- Solução: Atribuir responsabilidade de tal forma que a coesão permaneça alta.
- Exemplo (o mesmo para o acoplamento fraco): No sistema de biblioteca, suponha que queremos realizar a devolução da cópia do livro. Qual classe deve ser responsável por essa tarefa?

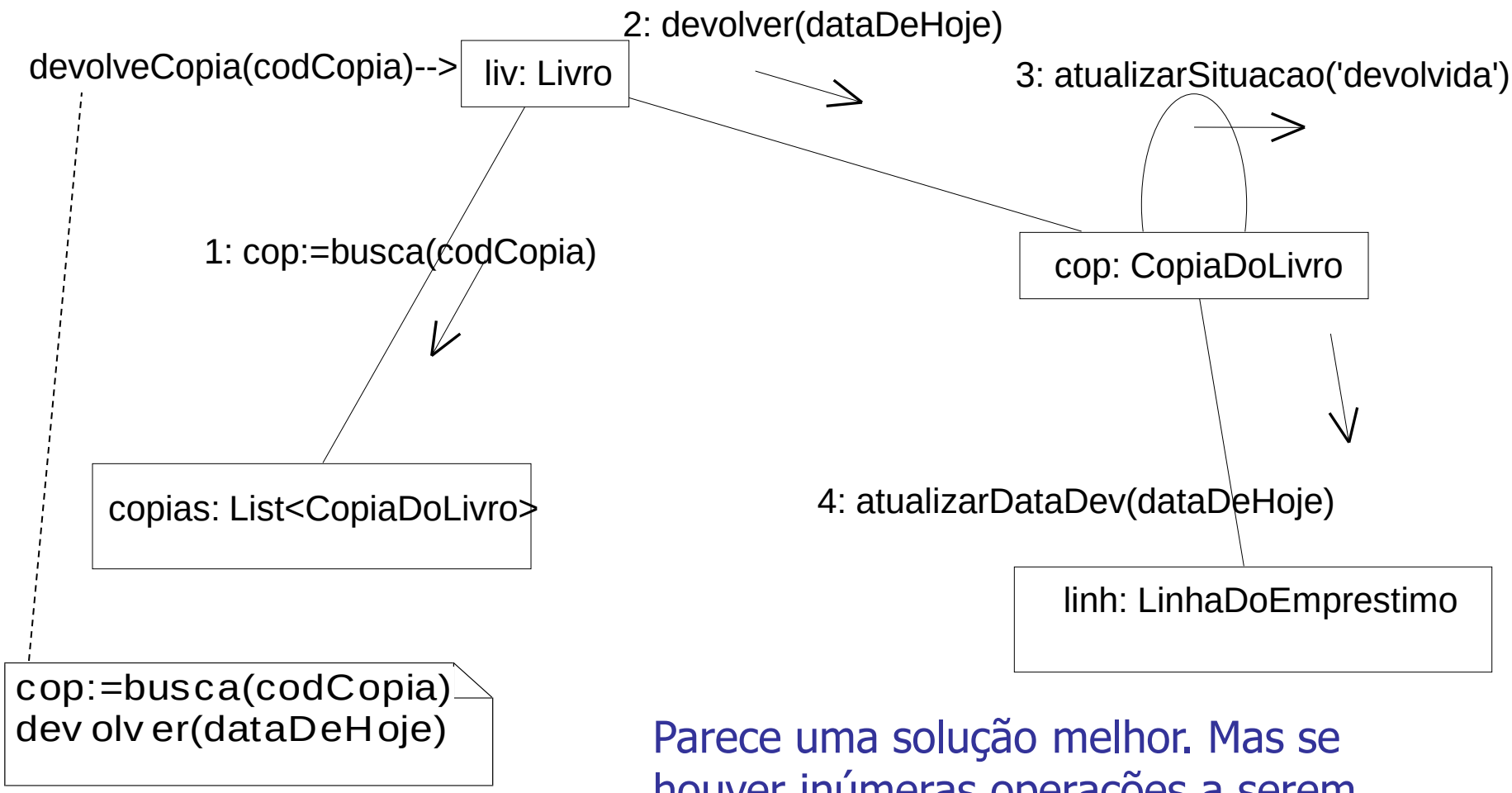

Reserva
período
situacao : char
0..1
corresponde a
0..n
0..n
corresponde a
^
faz
0..1
1..1
0..1
Atendente
nome
1..1
registra
0..n
Leitor
nome
tipo : char
faz
Empréstimo/Devolução
data do empréstimo
situação : Char
1..1
0..n
1..1
possui
Livro
titulo : String[30]
autor : String[30]
ano : int
ISBN : string[20]
editora : int
tipo : char
1..1
refere-se a >
1..n
0..1
LinhaDoEmpréstimo
data_prevista_devolução
data_entrega_real
Bibliotecaria
nome
1..1
registra
0..n
0..n
< refere-se a
1..1
possui
CopiaDoLivro
nro sequencial
situacao : char
liberadoParaEmprestimo : char
1..1
0..n

Projeto 1: responsabilidade atribuída ao *Leitor*



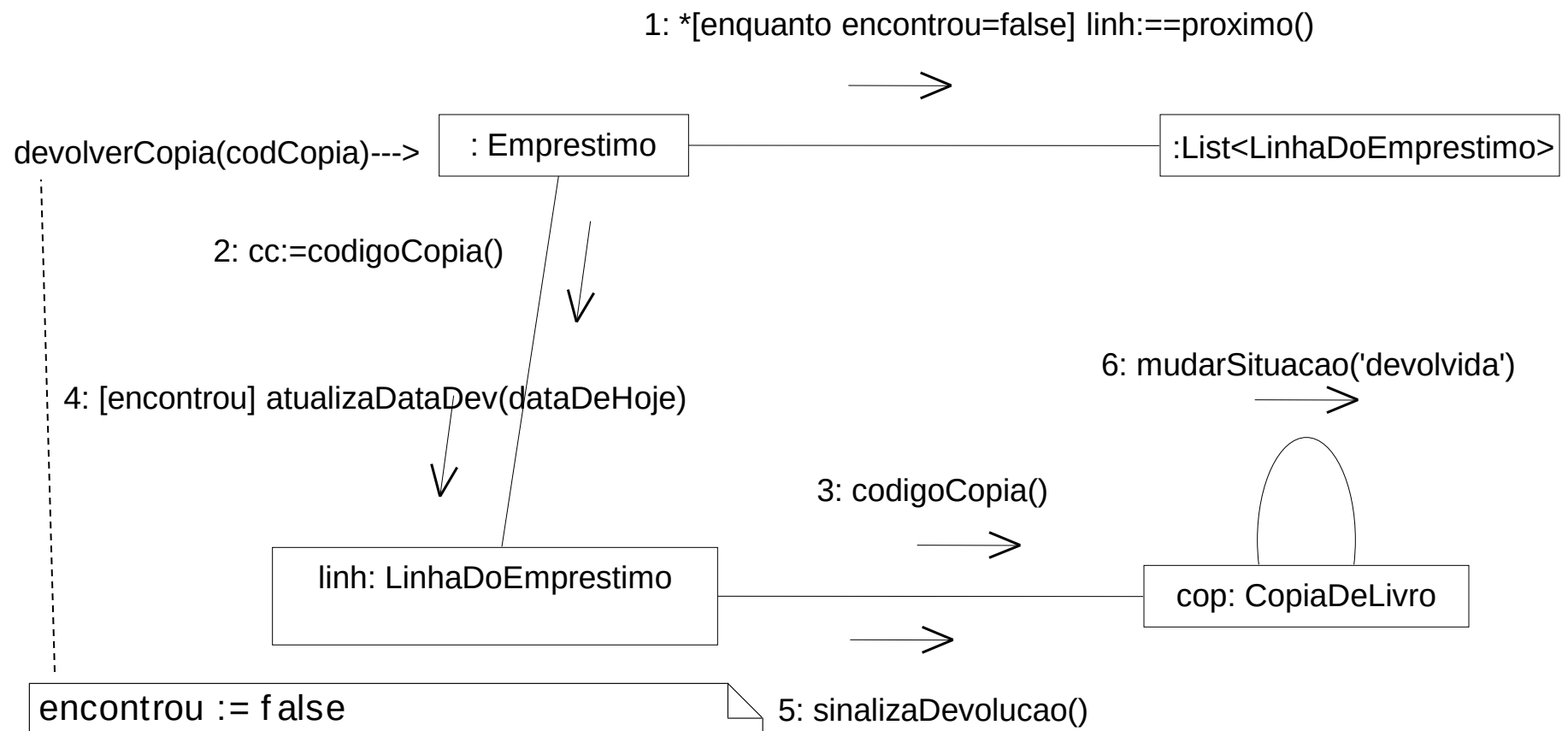
O Leitor fica parcialmente encarregado da devolução da cópia do livro. Neste exemplo, isso seria aceitável, mas o que aconteceria se houvesse 50 mensagens recebidas por Leitor?

Projeto 2: responsabilidade atribuída ao *Livro*



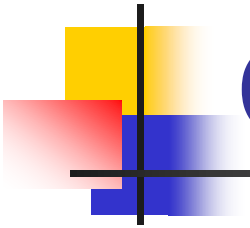
Parece uma solução melhor. Mas se houver inúmeras operações a serem feitas com o livro, ocorre o mesmo problema de Leitor.

Projeto 3: responsabilidade atribuída ao *Empréstimo*



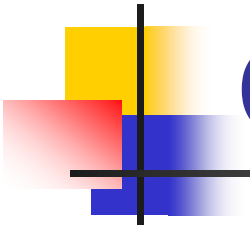

```
encontrou := false
enquanto encontrou == false
   linh := proxima linha do emprestimo
   cc:=obter o código da cópia
   encontrou:=(cc==codCopia)
fim-enquanto
se encontrou
    atualiza DataDevolucao(dataDeHoje)
fim-se
```

Esta é a melhor solução. O objeto empréstimo representa eventos bem definidos no sistema de biblioteca (empréstimo e devolução), por isso é mais intuitivo que ele assuma esta responsabilidade.



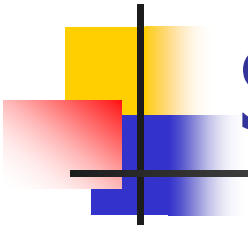
Coesão Alta

- Discussão:
  - Coesão alta, assim como Acoplamento Fraco, são princípios que devem ser considerados no projeto de objetos
    - má coesão traz acoplamento ruim e vice-versa
  - regra prática: classe com coesão alta tem um número relativamente pequeno de métodos, com funcionalidades relacionadas, e não executa muito trabalho
  - analogia com mundo real
    - ex: pessoas que assumem muitas responsabilidades não associadas podem tornar-se (e normalmente tornam-se) ineficientes



Coesão Alta

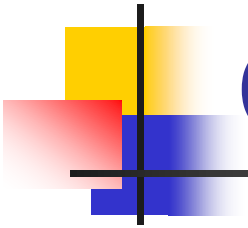
- Benefícios
  - mais clareza e facilidade de compreensão no projeto
  - simplificação de manutenção e acréscimo de funcionalidade/melhorias
  - favorecimento do acoplamento fraco
  - aumento no potencial de reutilização
    - classe altamente coesa pode ser usada para uma finalidade bastante específica



Será que esta solução é ideal?

- Ainda temos um problema: quando ocorre o evento de devolução da cópia, ainda não é conhecido o objeto *empréstimo* ao qual a cópia emprestada se refere.
- Portanto, é preciso eleger alguma classe, que conheça os empréstimos, para receber a mensagem *devolverCopia*.
- Essa classe terá que identificar o objeto empréstimo cujo código de cópia seja igual ao parâmetro fornecido

A pergunta anterior não está respondida nos slides a seguir. Voltaremos a ela no fim deste assunto.

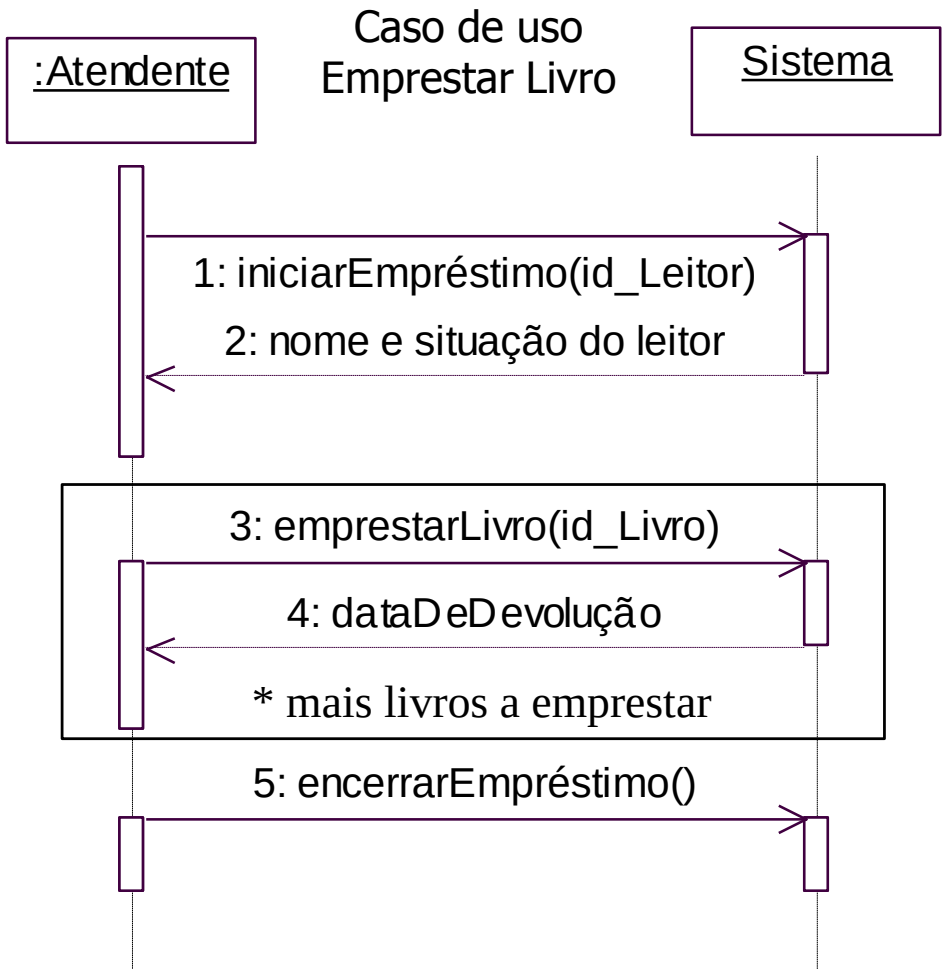


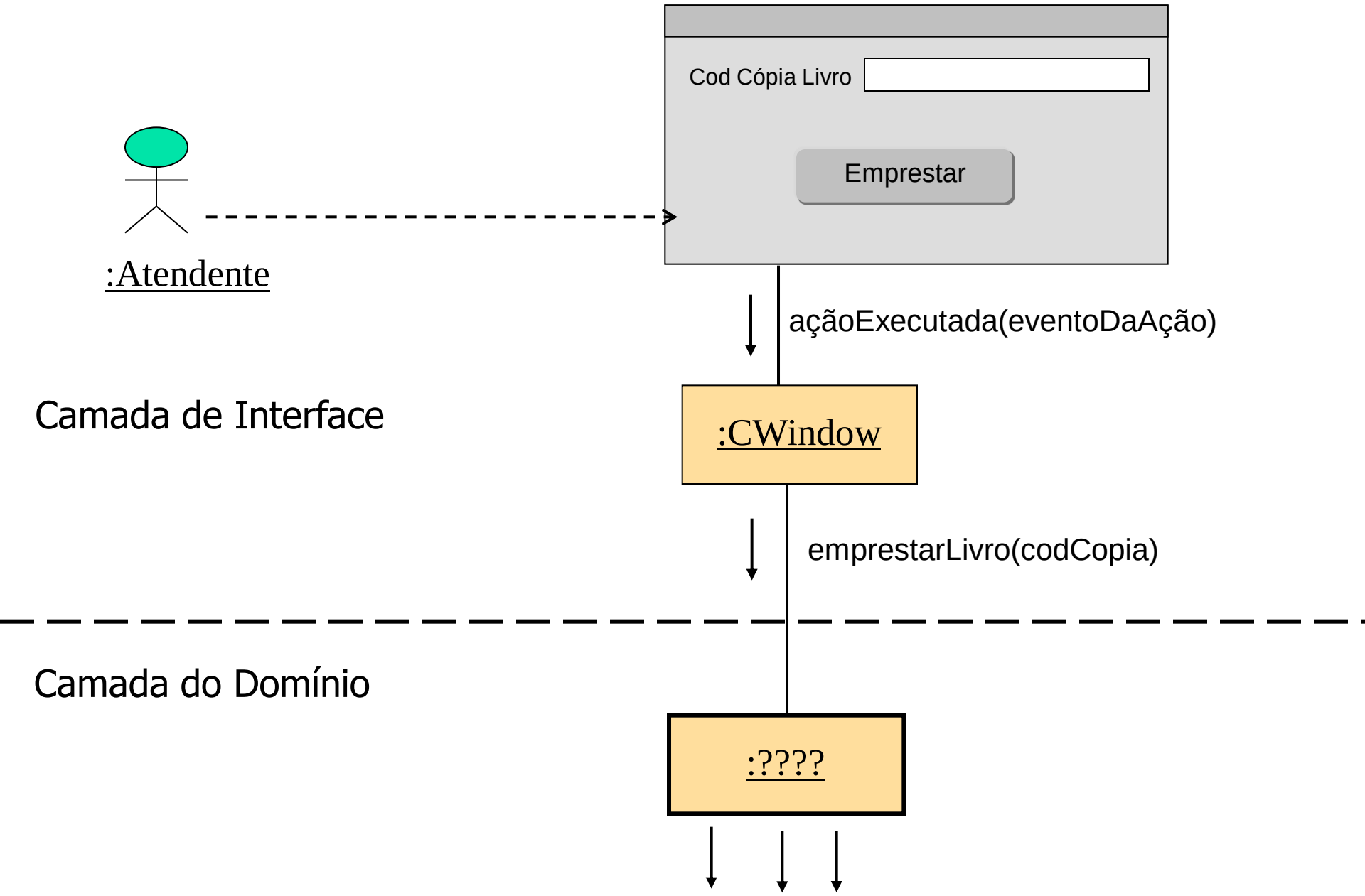
Controlador

- Problema: Quem deve ser responsável por tratar um evento do sistema ?
- Solução: A responsabilidade de receber ou tratar as mensagens de eventos (operações) do sistema pode ser atribuída a uma classe que:
  - represente todo o sistema, um dispositivo ou um subsistema – chamado de controlador fachada - OU
  - represente um cenário de um caso de uso dentro do qual ocorra o evento
    - TratadorDe<NomeDoCasoDeUso>, ControladorDe<NomeDoCasoDeUso>

Controlador

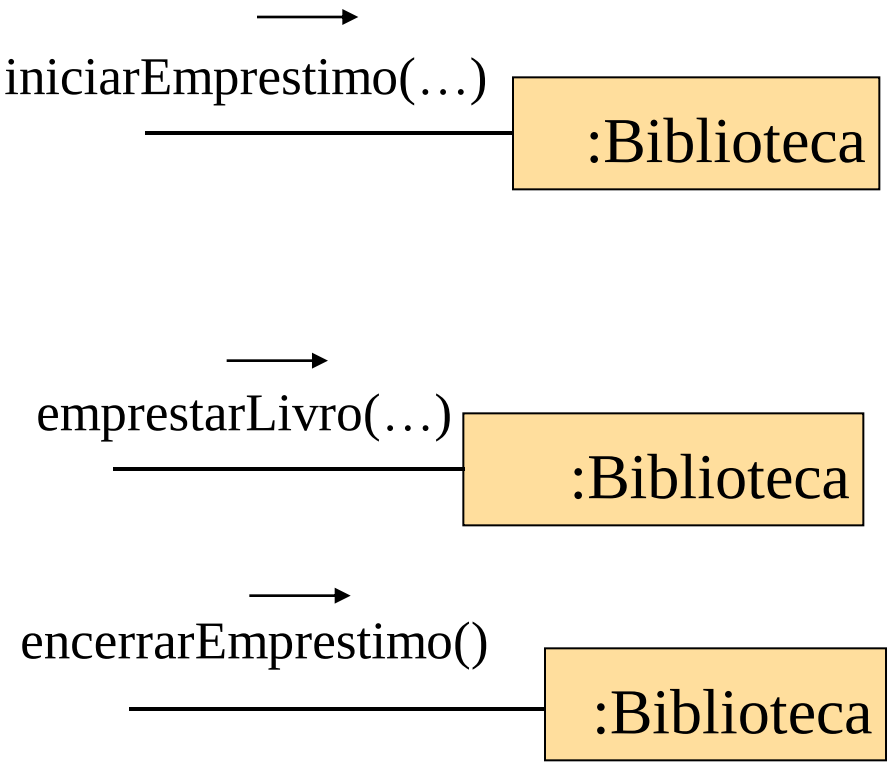
- Exemplo: quem vai tratar os eventos do sistema de biblioteca?



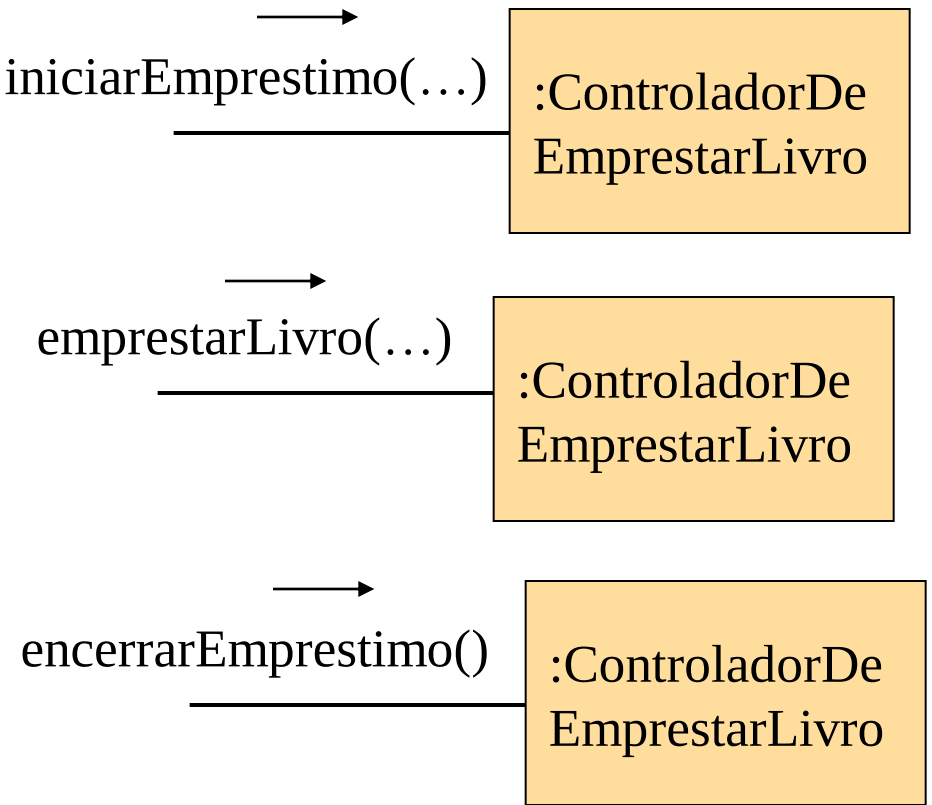

Cod Cópia Livro
Emprestar
:Atendente
açãoExecutada(eventoDaAção)
Camada de Interface
:CWindow
emprestarLivro(codCopia)
Camada do Domínio
:????

Exemplo: Opções de Controlador

- todo o sistema (controlador fachada): Biblioteca



- um tratador artificial do caso de uso: *ControladorDeEmprestarLivro*

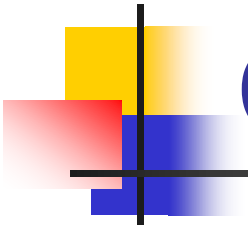


Discussão: Controladores Fachada

- Um controlador fachada deve ser um objeto (do domínio) que seja o ponto principal para as chamadas provenientes da interface com o usuário ou de outros sistemas
  - pode ser uma abstração de uma entidade física – ex: *TerminalDeAtendimento*
  - pode ser um conceito que represente o sistema – ex: *Biblioteca*
- São adequados quando não há uma quantidade muito grande de eventos de sistema
- Não é possível redirecionar mensagens do sistema para controladores alternativos (ex: outros subsistemas)

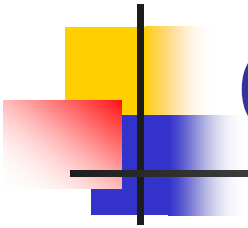
Discussão :Controladores de Casos de Uso

- Deve existir um controlador diferente para cada caso de uso
  - Por exemplo, o ControladorDeEmprestarLivro será responsável pelas operações iniciarEmpréstimo, emprestarLivro e encerrarEmpréstimo
- Não é um objeto do domínio, e sim uma construção artificial para dar suporte ao sistema. Ex: *ControladorDeEmprestarLivro*, *ControladorDeDevolverLivro*
- Pode ser uma alternativa se a escolha de controladores fachada deixar a classe controladora com alto acoplamento e/ou baixa coesão (controlador inchado por excesso de responsabilidades)
- É uma boa alternativa quando existem muitos eventos envolvendo diferentes processos.



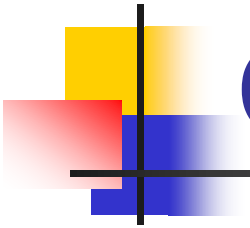
Controladores inchados

- Classe controladora mal projetada - inchada
  - coesão baixa – falta de foco e tratamento de muitas responsabilidades
- Sinais de inchaço:
  - uma única classe controladora tratando todos os eventos, que são muitos. Comum com controladores fachada
  - o próprio controlador executa as tarefas necessárias para atender o evento, sem delegar para outras classes (coesão alta, não especialista)
  - controlador tem muitos atributos e mantém informação significativa sobre o domínio, ou duplica informações existentes em outros lugares



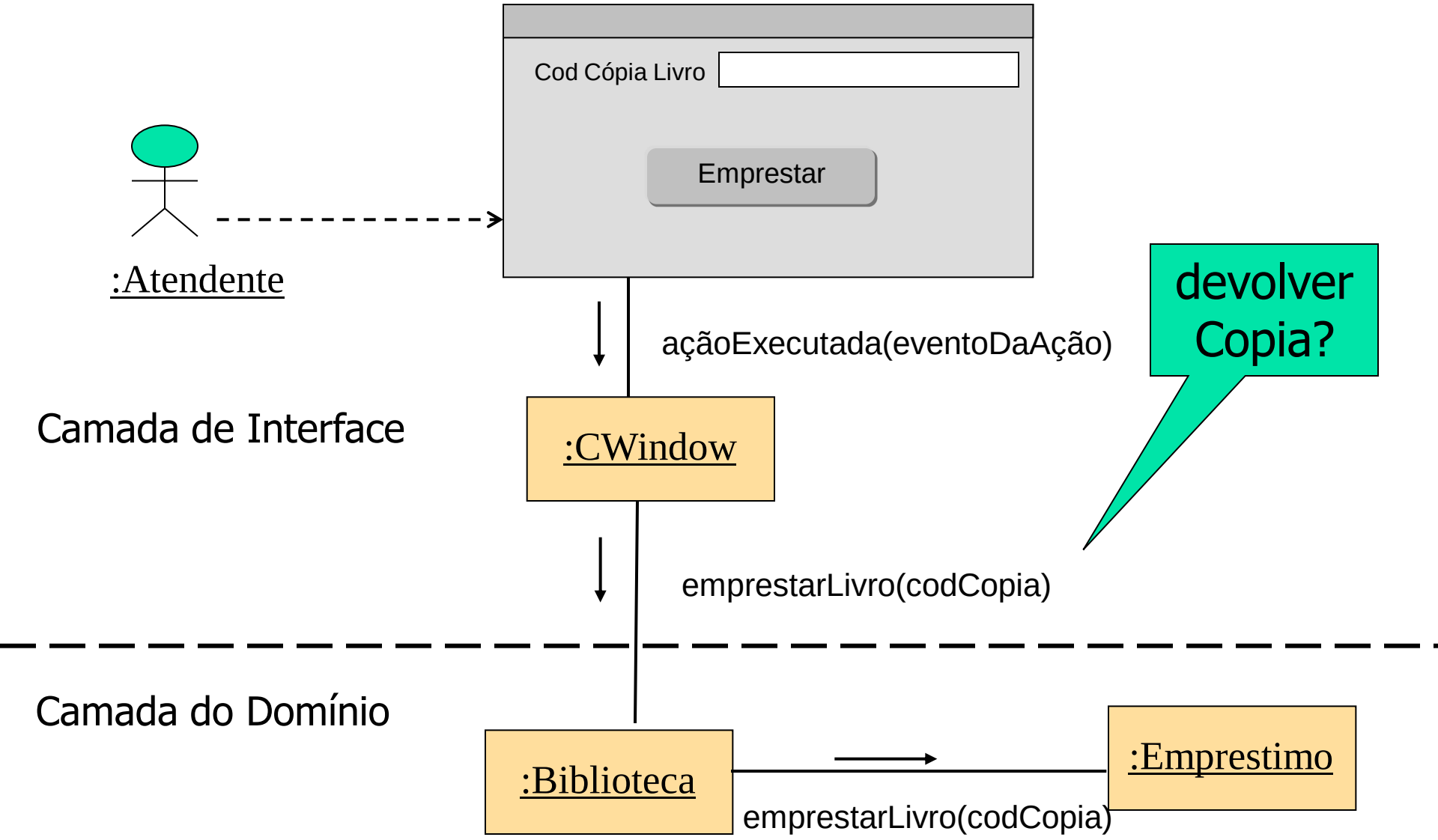
Controlador

- Curas para controladores inchados
  - acrescentar mais controladores – misturar controladores fachada e de casos de uso
  - delegar responsabilidades
- Corolário: objetos de interface (como objetos "janela") e da camada de apresentação não devem ter a responsabilidade de tratar eventos do sistema



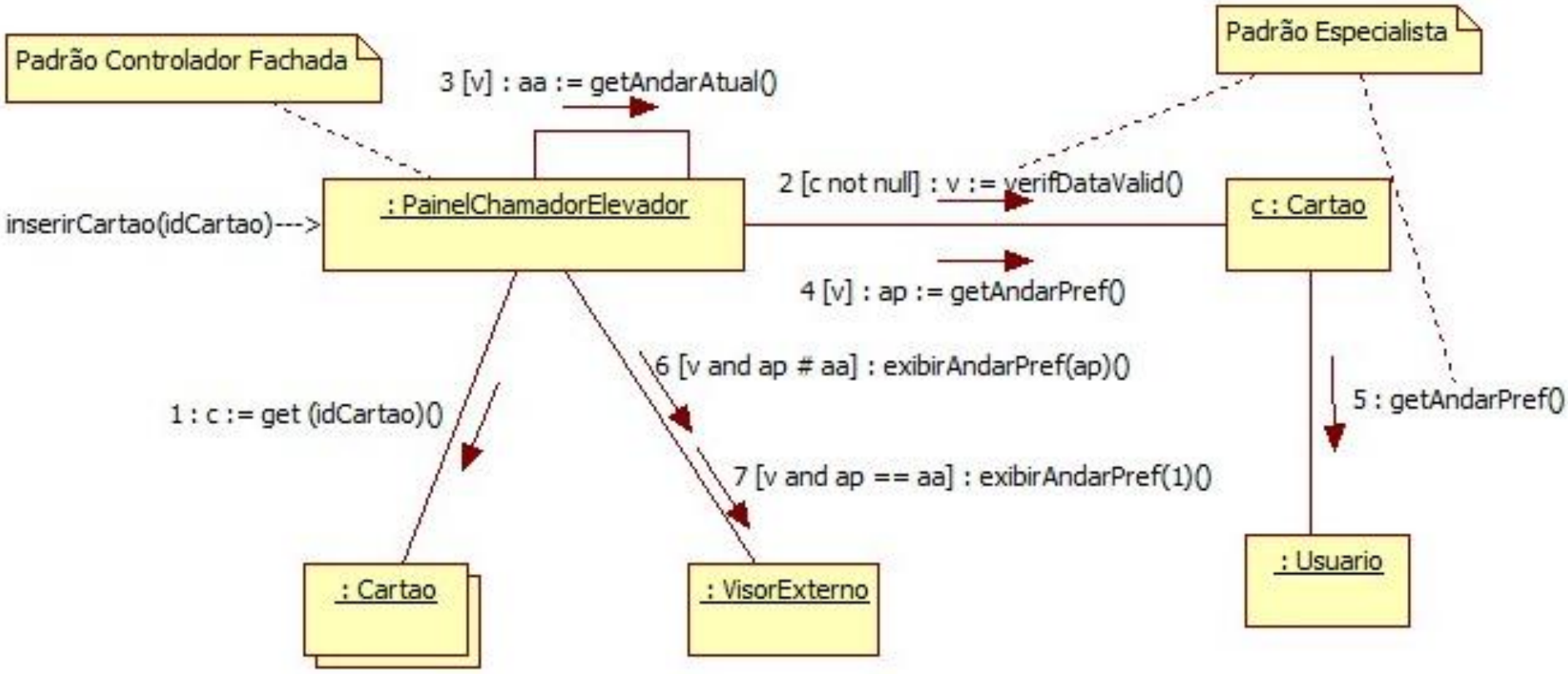
Controlador

- Benefícios:
  - aumento das possibilidades de reutilização de classes e do uso de interfaces “plugáveis”.
  - conhecimento do estado do caso de uso – controlador pode armazenar estado do caso de uso, garantindo a seqüência correta de execução de operações



Exercício: refazer o DC de devolverCopia, usando Biblioteca como classe controladora da operação

Projeto – Diagramas de comunicação do Elevador – com GRASP



Projeto – Diagramas de comunicação do Elevador – com GRASP