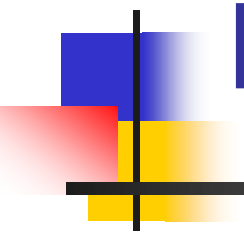
Frameworks
O Problema da Persistência

Análise e Projeto Orientados a Objetos

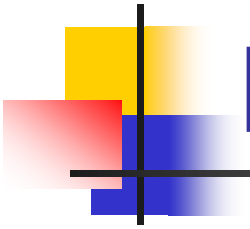
Profa Dra Rosana T. V. Braga



Frameworks

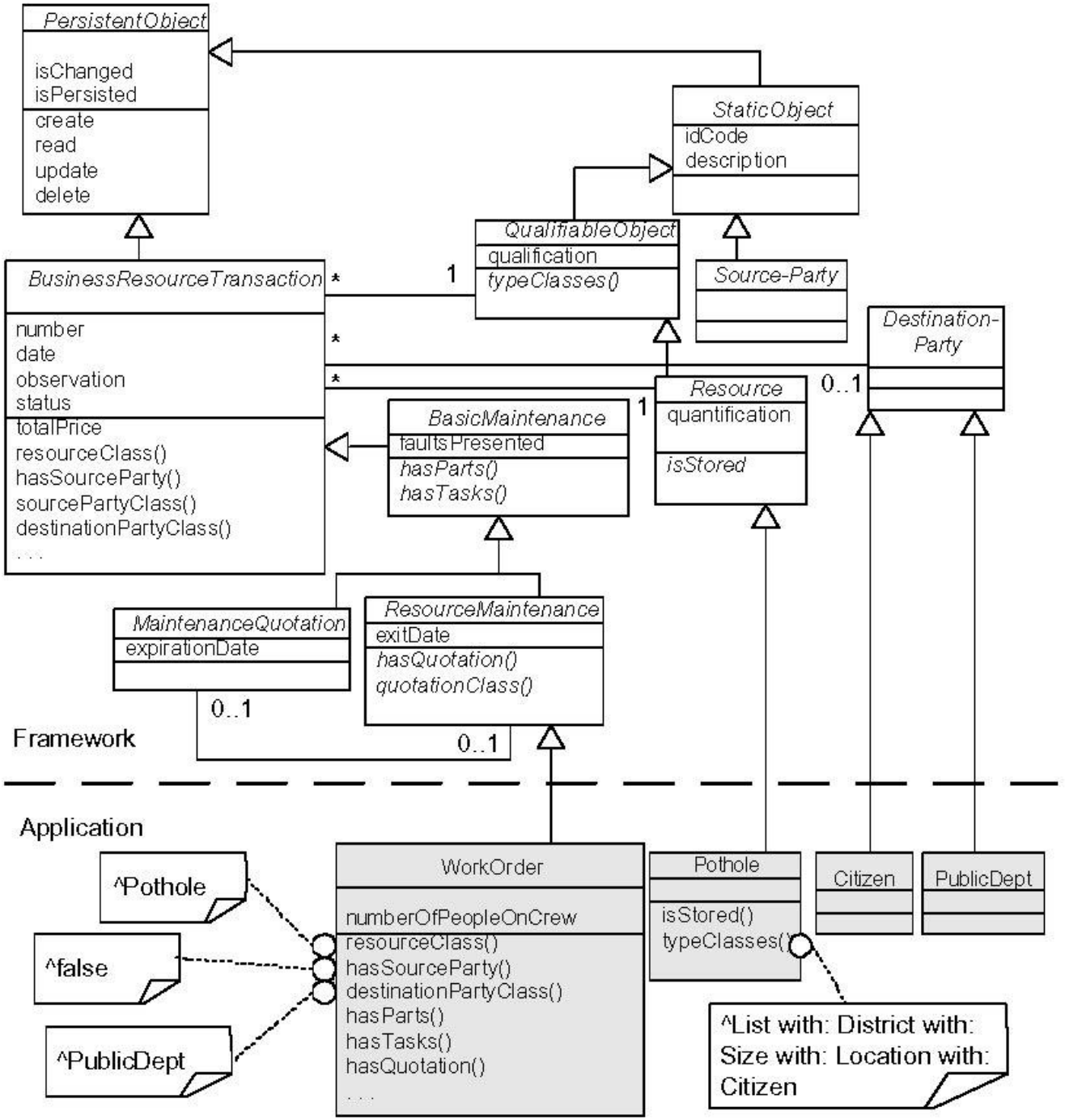
Frameworks

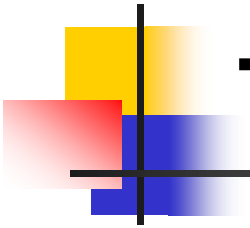
- Definições:
  - Aplicação semi-completa reutilizável que, quando especializada, produz aplicações personalizadas (Johnson & Foote, 1988) ou
  - Coleção de classes abstratas e concretas e a interface entre elas, representando o projeto de um sub-sistema (Pree, 1995)



Frameworks

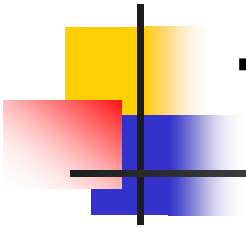
- Hot-Spots
  - **Representam as partes do framework de aplicação que são específicas de sistemas individuais**
  - **São projetados para serem genéricos - podem ser adaptados às necessidades da aplicação**
- Frozen-Spots
  - **Definem a arquitetura geral de um sistema de software - seus componentes básicos e os relacionamentos entre eles**
  - **Permanecem fixos em todas as instanciações do framework de aplicação**


PersistentObject
isChanged
isPersisted
create
read
update
delete
StaticObject
idCode
description
QualifiableObject
qualification
typeClasses()
Source-Party
BusinessResourceTransaction
number
date
observation
status
totalPrice
resourceClass()
hasSourceParty()
sourcePartyClass()
destinationPartyClass()
. . .
*
1
*
*
1
Destination-Party
0..1
Resource
quantification
isStored
BasicMaintenance
faultsPresented
hasParts()
hasTasks()
MaintenanceQuotation
expirationDate
ResourceMaintenance
exitDate
hasQuotation()
quotationClass()
0..1
0..1
Framework
Application
^Pothole
^false
^PublicDept
WorkOrder
numberOfPeopleOnCrew
resourceClass()
hasSourceParty()
destinationPartyClass()
hasParts()
hasTasks()
hasQuotation()
. . .
Pothole
isStored()
typeClasses()
Citizen
PublicDept
^List with: District with: Size with: Location with: Citizen



Tipos de Frameworks

- Framework caixa branca:
  - reuso por herança e associação dinâmica
  - deve-se entender detalhes de como o framework funciona
- Framework caixa preta:
  - reuso por composição ou definição de interfaces para os componentes.
  - deve-se entender apenas a interface do cliente

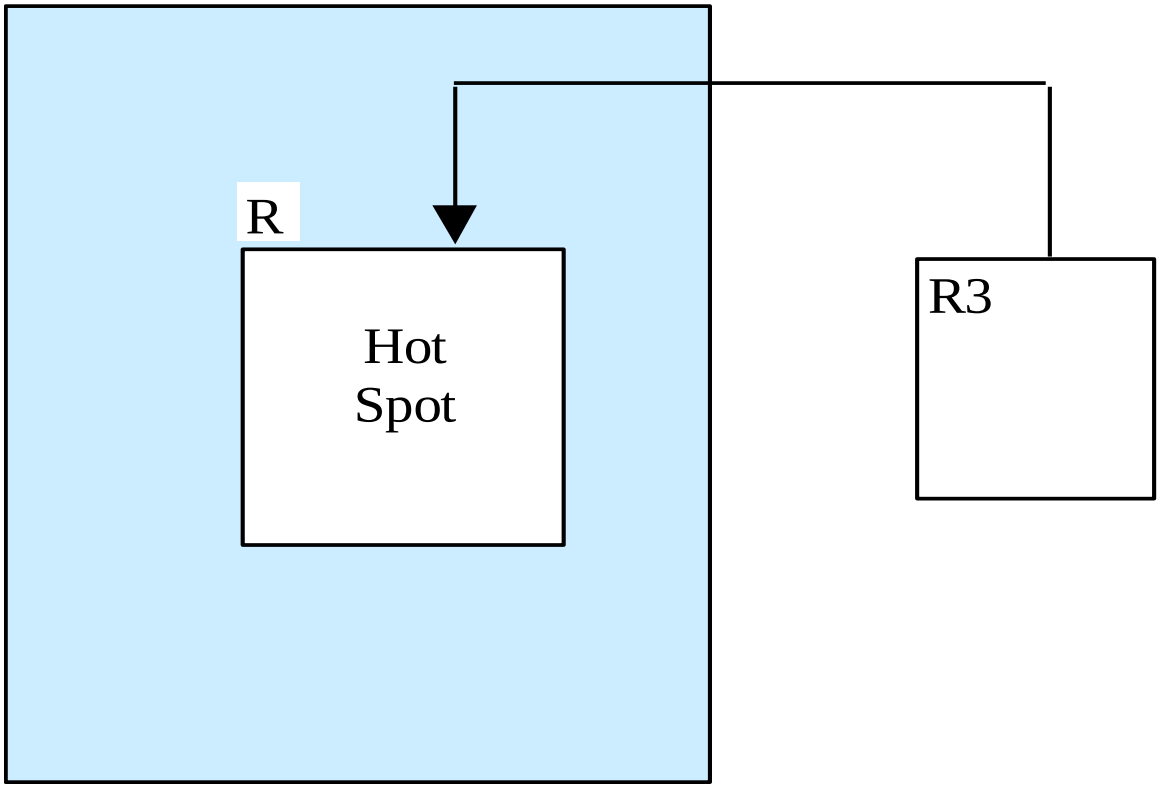


Tipos de Frameworks

- Framework caixa cinza:
  - combinação do caixa branca e do caixa-preta
  - reuso por herança, associação dinâmica e definição de interfaces
  - Levantamento realizado por Yassin e Fayad em 1999: 55% dos frameworks caixa-cinza, 30% dos frameworks caixa-branca e 15% dos frameworks caixa-preta

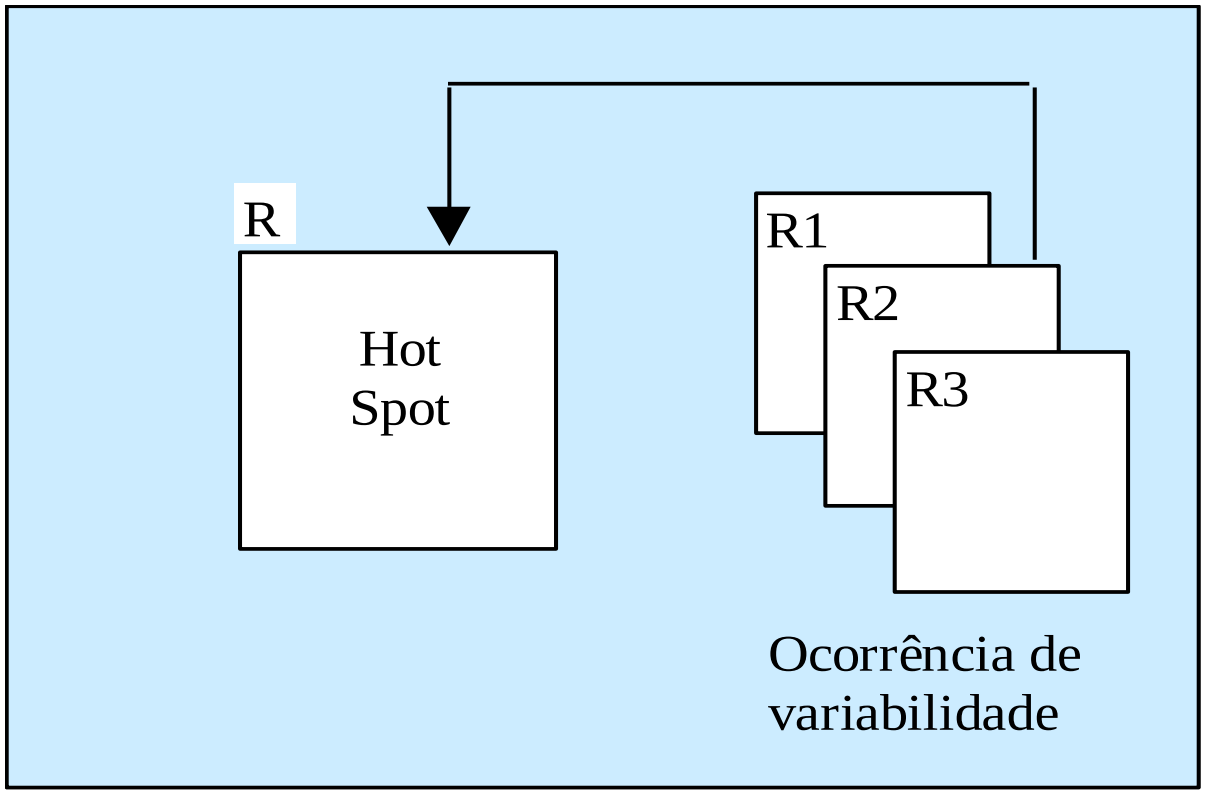
Tipos de Frameworks

Framework Caixa Branca



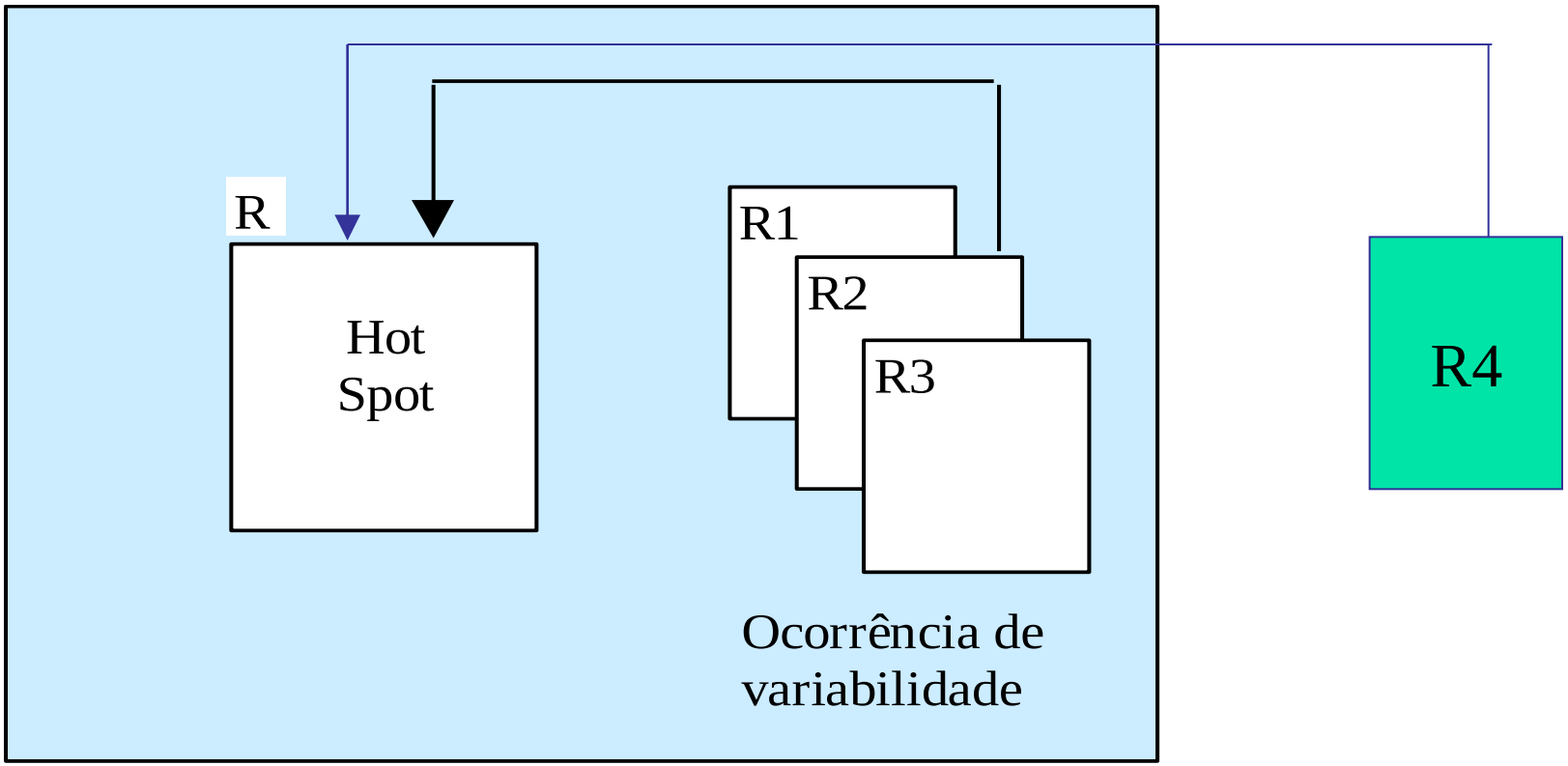
Tipos de Frameworks

Framework Caixa Preta



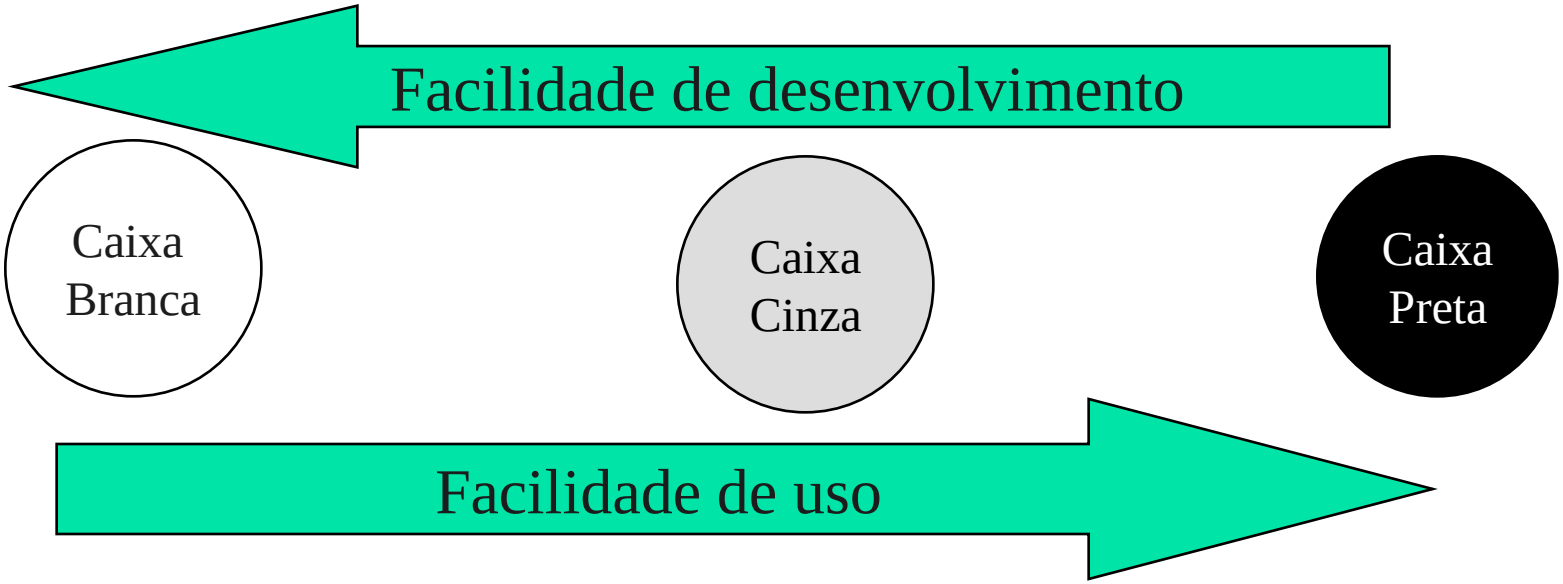
Conceitos Básicos

Framework Caixa Cinza

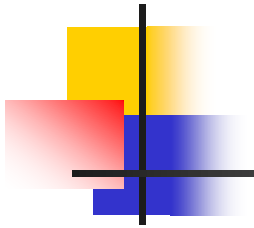


Tipos de framework

- framework caixa branca é mais fácil de projetar
- framework caixa preta é mais fácil de usar
- frameworks caixa-branca evoluem para se tornar mais caixa preta



Classificação de Frameworks de aplicação



- Frameworks de Infra-estrutura do Sistema
  - simplificam o desenvolvimento da infra-estrutura de sistemas portáveis e eficientes,
  - exemplos: sistemas operacionais, comunicação, interfaces com o usuário e ferramentas de processamento de linguagem
  - em geral são usados internamente em uma organização de software e não são vendidos a clientes diretamente

Classificação de Frameworks de aplicação

- Frameworks de Integração de Middleware
  - Middleware: camada de software entre a rede e as aplicações, que provê serviços como identificação, autenticação, autorização, diretórios e segurança
  - Frameworks de middleware: usados em geral para integrar aplicações e componentes distribuídos.
  - projetados para melhorar a habilidade de desenvolvedores em modularizar, reutilizar e estender sua infra-estrutura de software para funcionar "sem costuras" em um ambiente distribuído
  - exemplos: Object Request Broker (ORB), middleware orientado a mensagens e bases de dados transacionais

Classificação de Frameworks de aplicação

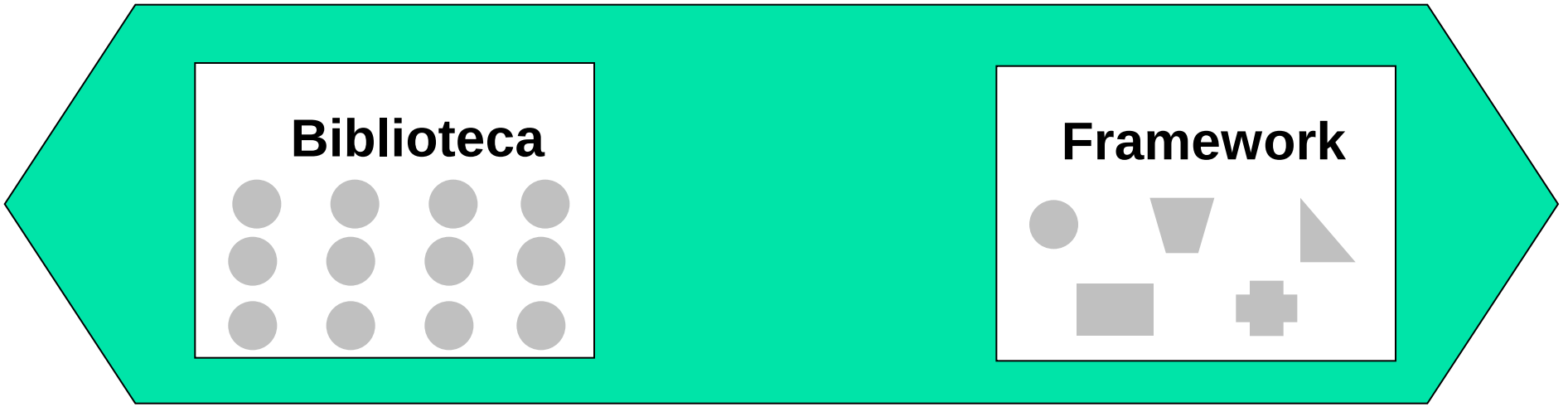
- Frameworks de Aplicação Empresarial
  - voltados a domínios de aplicação mais amplos e são a pedra fundamental para atividades de negócios das empresas.
  - exemplos: telecomunicações, aviação, manufatura e engenharia financeira.
  - são mais caros para desenvolver ou comprar, mas podem dar um retorno substancial do investimento, já que permitem o desenvolvimento de aplicações e produtos diretamente

Frameworks de aplicação empresarial

Inversão de Controle

- Biblioteca de subrotinas
  - Programa do usuário chama código reusado
  - Usuário projeta estrutura do programa
- Framework
  - Código reusado chama programa do usuário
  - Estrutura do programa determinada principalmente pelo código reusado

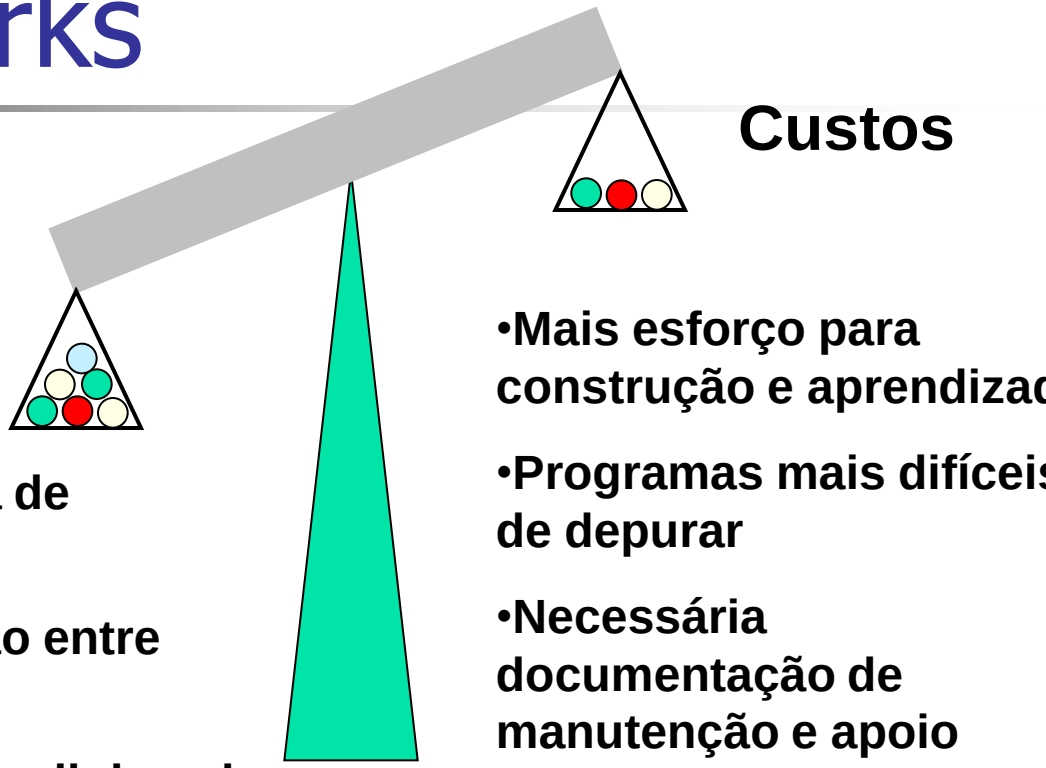
Frameworks



- Conjunto de classes instanciadas pelo cliente
- Cliente chama funções
- Fluxo de controle não pré-definido
- Interação não pré-definida
- Não tem comportamento “default”

- Cuida da personalização através de subclasses
- Chama funções do cliente
- Controla o fluxo de execução
- Define a interação dos objetos
- Tem comportamento “default”

Frameworks

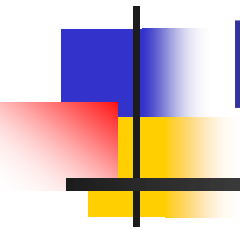


Benefícios

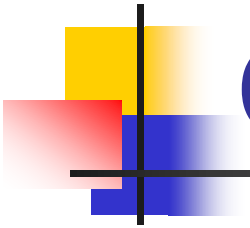
- **Economia a longo prazo**
- **Aproveitamento da experiência de especialistas do domínio**
- **Maior consistência e integração entre aplicações**
- **Redução da manutenção: menos linhas de código nas aplicações, reparos no framework se propagam pelas aplicações**
- **Melhora da produtividade: os programadores podem se concentrar nas características específicas de suas aplicações**

Custos

- **Mais esforço para construção e aprendizado**
- **Programas mais difíceis de depurar**
- **Necessária documentação de manutenção e apoio**



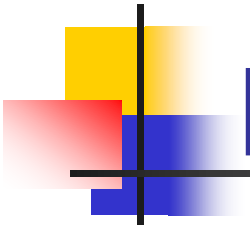
Frameworks, padrões e persistência



Objetivo

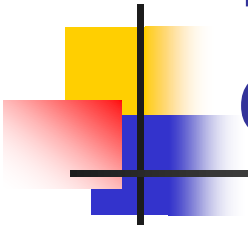
- Mostrar o uso de padrões na construção de um framework para persistência
- Usar esse exemplo para explicar problemas gerais do projeto de frameworks e alguns aspectos críticos dos serviços que apóiam a persistência de objetos.

O problema: objetos persistentes



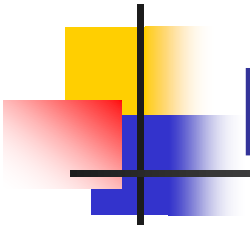
- Suponha que todas as instâncias de *Livro* residam em algum mecanismo de armazenamento persistente e precisem ser trazidas para a memória durante o uso do sistema de Biblioteca
- **Objetos Persistentes** são aqueles que necessitam armazenamento, por exemplo em um banco de dados relacional ou orientado a objetos

Mecanismos de armazenamento e objetos persistentes



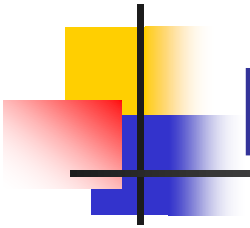
- Banco de dados de objetos: não necessitam de serviços de persistência adicionais
- Banco de dados relacionais: representações diferentes devido ao desencontro entre as representações de dados orientadas a registros e as orientadas a objetos. São necessários serviços para resolver esse problema
- Outros: meios de armazenamento como arquivos comuns, banco de dados hierárquicos, etc. Mesmo problema dos relacionais

Solução: um framework para persistência

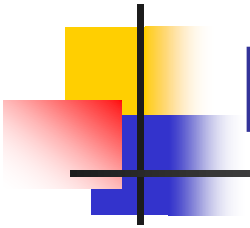


- Framework para persistência: conjunto de classes reutilizáveis que fornece serviços para objetos persistentes.
- Tarefas típicas:
  - traduzir objetos para registros
  - Salvar objetos no banco de dados
  - Traduzir registros para objetos

Requisitos para o framework para persistência (FP)



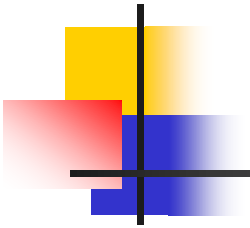
- Funções:
  - Armazenar e recuperar os objetos em um mecanismo de armazenamento persistente
  - Efetuar confirmação (*commit*) e retrocesso (*rollback*) de transações do banco de dados
- Qualidades:
  - Extensível, mínimo de mudanças no código,
  - Fácil de usar, transparente



Possível Solução

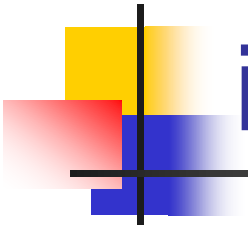
- Criar uma classe ObjetoPersistente, com atributos e métodos para apoiar a persistência.
- Todos os objetos persistentes devem herdar dessa classe, direta ou indiretamente.
- **Problema: *forte acoplamento***

Idéias-chave para implementar o FP



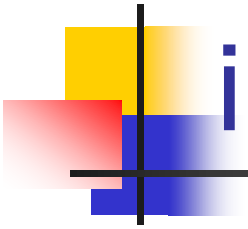
- Mapeamento – deve ser possível mapear uma classe e seu armazenamento persistente (p.ex. atributos do objeto = campos do registro)
- Identidade do objeto – identificador único para evitar duplicatas indesejadas
- Materialização / Desmaterialização – transformação de uma representação de dados não orientada a objetos em um objeto (e vice-versa)

Idéias-chave para implementar o FP



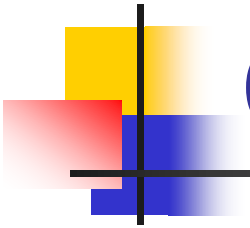
- Materialização sob demanda (*lazy*) – nem todos os objetos são materializados de uma vez, mas somente quando necessário ou requisitado. Isso pode ser implementado usando Referências Inteligentes (*smart references*) – padrão Proxy Virtual
- Objetos complexos – como representá-los e materializá-los?

Idéias-chave para implementar o FP



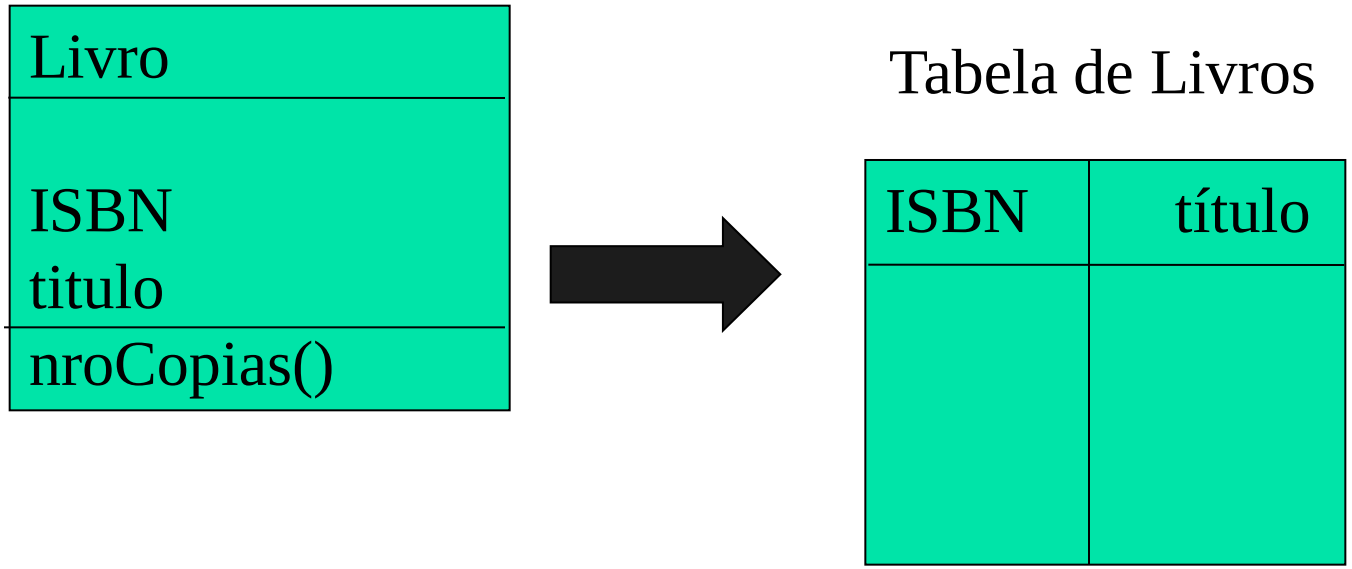
- Estado de transação de um objeto– objetos modificados ficam "sujos" (dirty) para determinar se precisam ser salvos de volta no seu armazenamento persistente.
- Operações em transações – efetivar a transação (*commit*) e desfazer a transação (*rollback*)
- Busca – encontrar objetos com base em alguns critérios

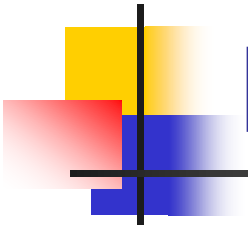
O Padrão Representar Objetos como Tabelas



- Problemas: Como mapear um objeto para um arquivo ou um esquema de banco de dados relacional?
- Solução: Defina uma tabela para cada classe de objetos persistentes. Os atributos de objetos que contém tipos de dados primitivos são mapeados para colunas.

O Padrão Representar Objetos como Tabelas

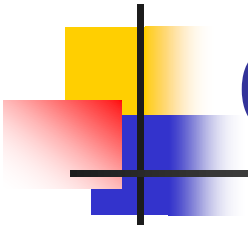




Problemas

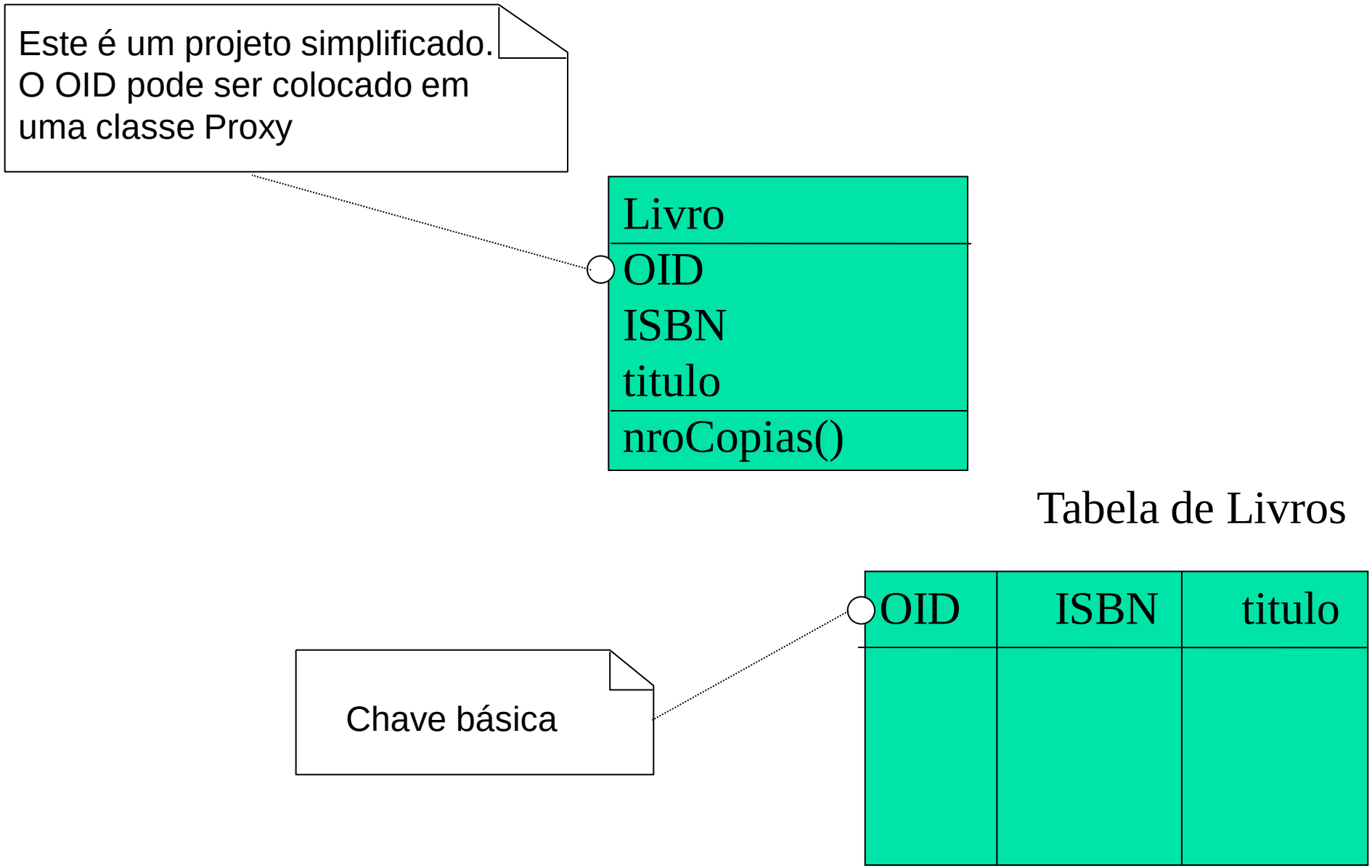
- Para objetos apenas com atributos simples, o mapeamento é direto. Porém, objetos complexos possuem atributos que referenciam outros objetos (às vezes complexos por sua vez).

Padrão "Identificador do Objeto"

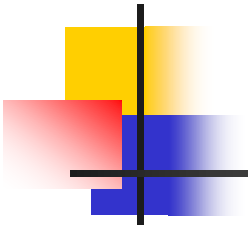


- Problema: como relacionar objetos com registros e garantir que a materialização de um objeto não resulte em objetos duplicados?
- Solução: Atribuir um identificador de objeto (OID) para cada registro de objeto.
- Exemplo: usar o Identificador Universal Único de 16 bytes, o qual garante que cada valor seja único para qualquer data e hora.
  - Existe uma API do Windows que fornece uma função para geração automática desse identificador.

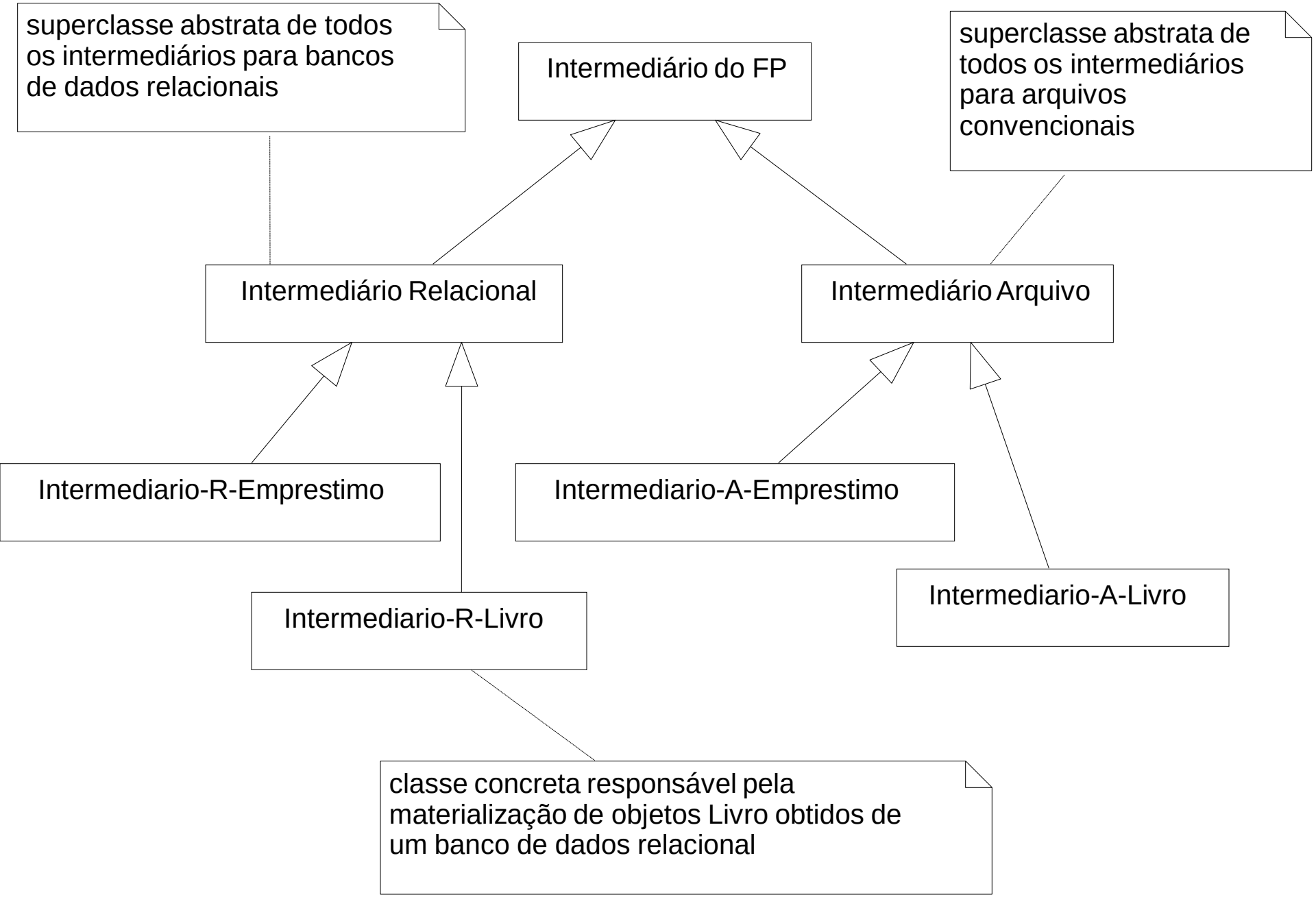
Padrão "Identificador do Objeto"



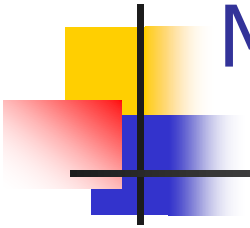
O Padrão Intermediário (“Database Broker”)



- Problema: como evitar o alto acoplamento de uma classe de objetos persistentes ao conhecimento do mecanismo de persistência? Como evitar a baixa coesão ao acrescentar a uma classe responsabilidades não relacionadas à sua função principal?
- Solução: Criar uma classe Intermediária “Broker”, responsável pela materialização, desmaterialização e memorização prévia (cache) dos objetos

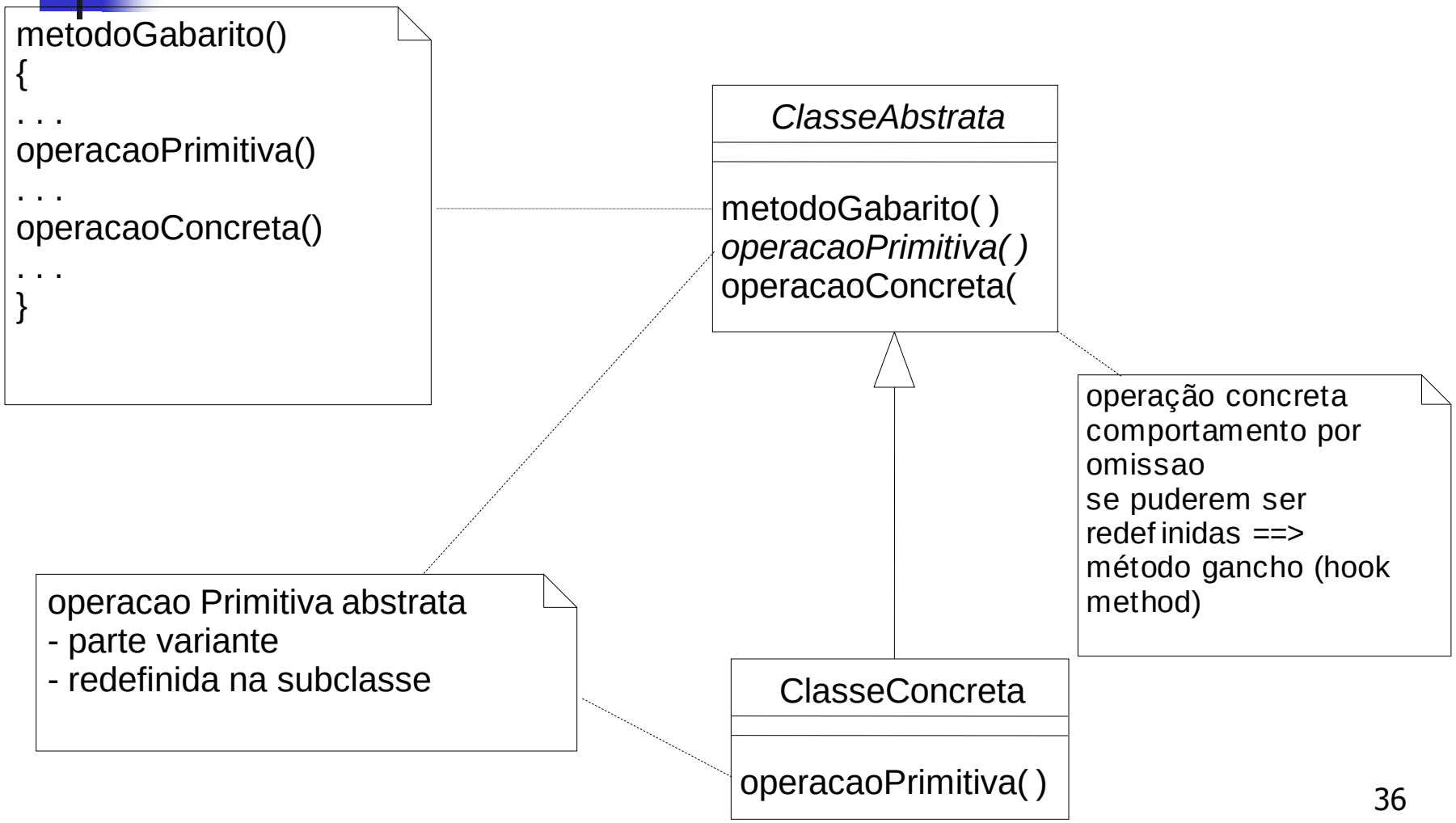

superclasse abstrata de todos os intermediários para bancos de dados relacionais
Intermediário do FP
superclasse abstrata de todos os intermediários para arquivos convencionais
Intermediário Relacional
Intermediário Arquivo
Intermediario-R-Emprestimo
Intermediario-A-Emprestimo
Intermediario-R-Livro
Intermediario-A-Livro
classe concreta responsável pela materialização de objetos Livro obtidos de um banco de dados relacional

O Padrão Método Gabarito ("Template Method")

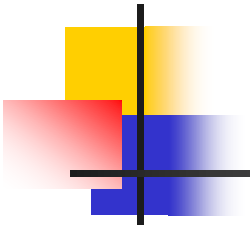


- Em um framework, muitos métodos só serão definidos pelas classes instanciadas a partir dele.
- A idéia do Template Method é definir um método gabarito em uma superclasse, que contenha o esqueleto do algoritmo, com suas partes variáveis e invariáveis. Esse método invoca outros métodos, alguns dos quais são operações que podem ser definidas em uma subclasse.
- Assim, as subclasses podem redefinir os métodos que variam, acrescentando comportamento específico dos pontos variáveis.

Padrão Método Gabarito

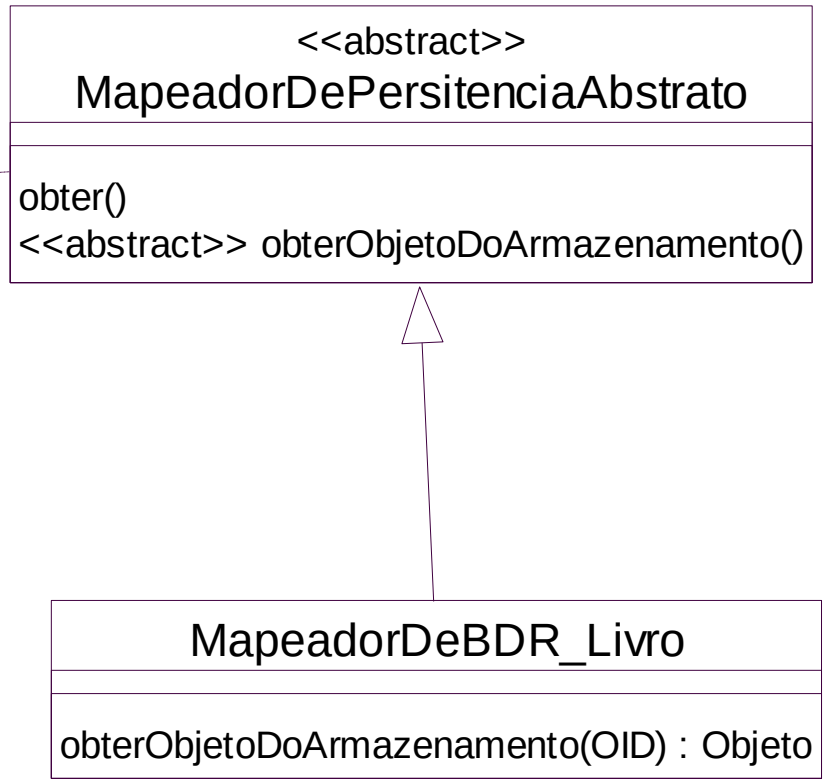


Uso do Padrão Método Gabarito no FP



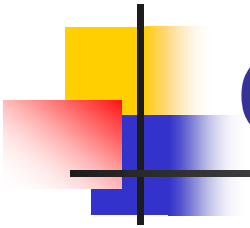
- No FP, um objeto pode ser recuperado pelo método *objeto(OID)*
- Esse método invoca o método *obterObjetoDoArmazenamento(OID)*, que deve criar uma instância vazia do objeto e mover os dados do registro para esse objeto. Se o objeto já tiver sido materializado, estará disponível em cache.

```
//metodo gabarito
public final Objeto obter(OID oid)
{ obj:= objetoEmCache.obter(oid);
  if (obj == null)
    {obj := obterObjetoDoArmazenamento(oid)
     objetoEmCache.atrib(oid,obj);
     }
return obj;
}
```



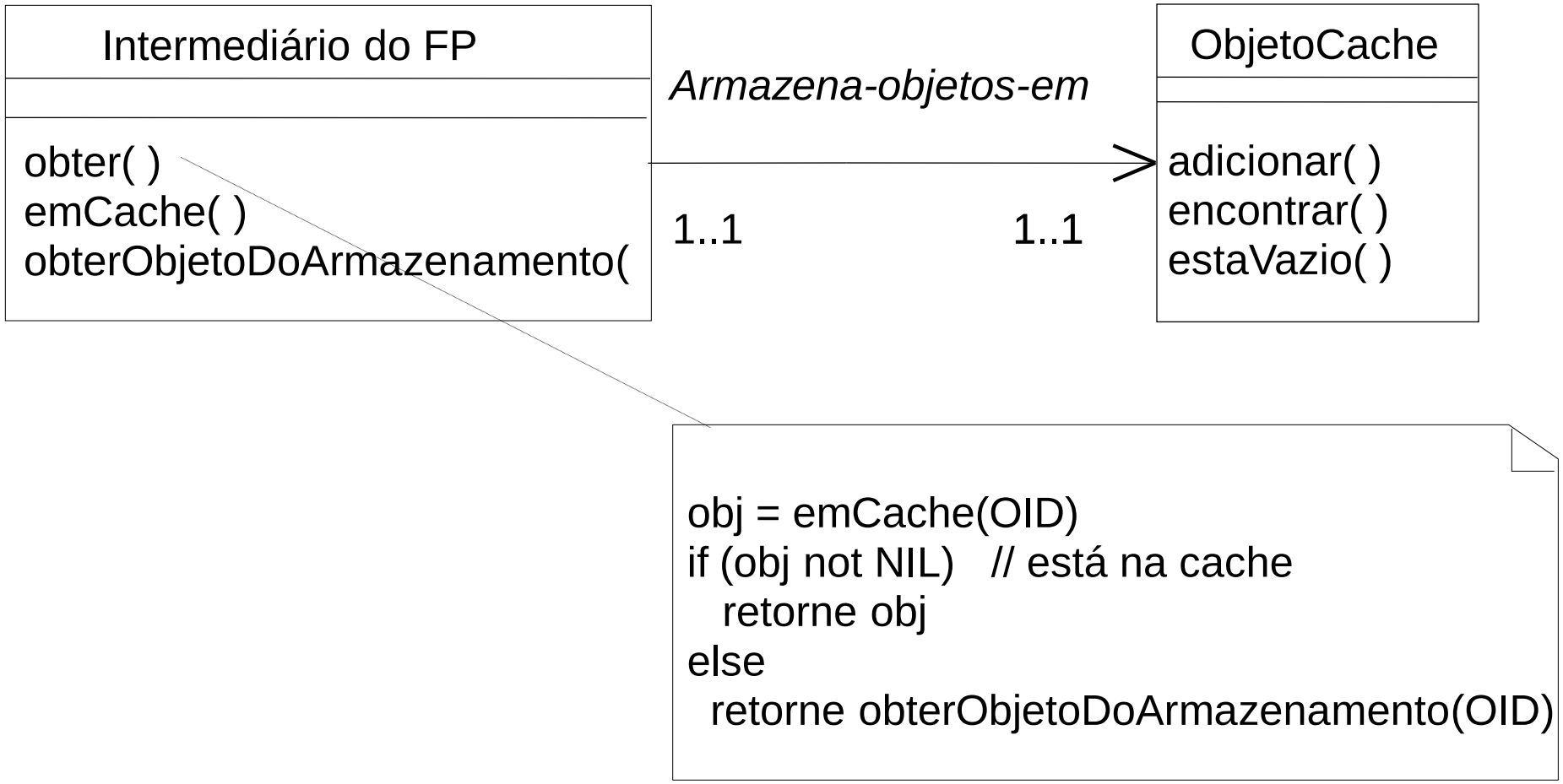

```
// sobrepoe o método abstrato
protected Objeto obterObjetoDoArmazenamento(OID oid)
{
String chave=oid.paraCadeia;
regBD=resultado da execução SQL de:
   "Select * from LIVRO where chave="+chave
Livro livro = new Livro();
livro.atribOID(oid);
livro.atribISBN(regBD.obterColuna("ISBN"));
livro.atribTitulo(regBD.obterColuna("titulo"));
return livro;
}
```

O Padrão Gerenciamento de Cache

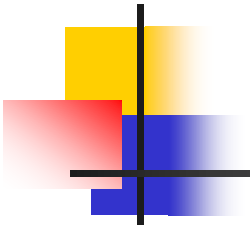


- É desejável manter os objetos materializados em uma memória temporária local (cache), para melhorar o desempenho e o suporte a operações de gerenciamento de transações.
- Esse padrão propõe que a classe Intermediária do banco de dados relacional fique responsável pela manutenção de sua cache.
- Quando os objetos são materializados, são colocados na cache, tendo como chave o seu OID. Nas solicitações subsequentes o Intermediário busca primeiramente na cache e, se não encontrar o objeto, faz a materialização

Intermediários mantém um Cache de objetos materializados



Caches para gerenciamento de transações



- New Clean Cache – objetos novos não modificados
- Old Clean Cache –objetos antigos (já persistidos) materializados, ainda não modificados
- New Dirty Cache – objetos novos já modificados
- Old Dirty Cache – objetos antigos materializados, já modificados
- Old Delete Cache – objetos antigos materializados a serem removidos

Múltiplos Caches para o Intermediário

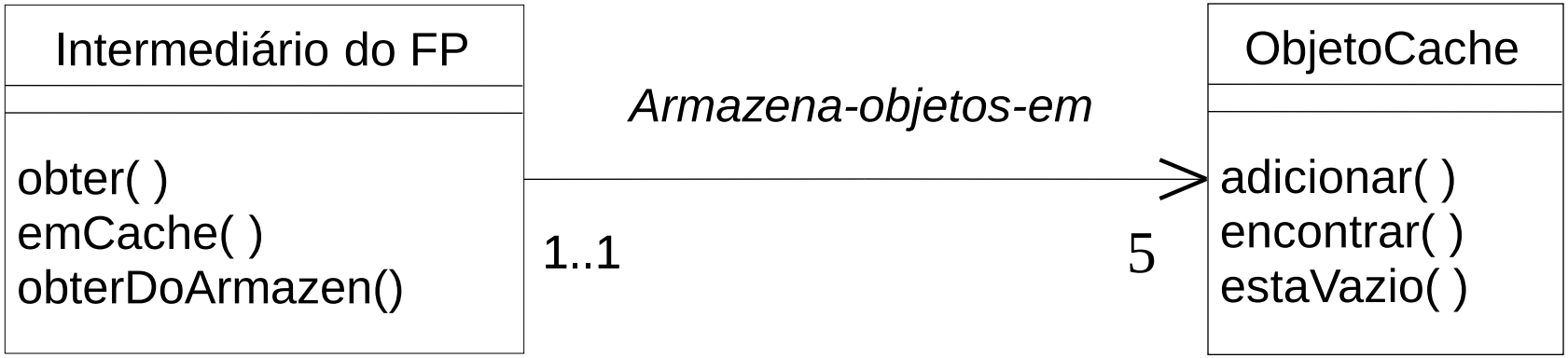
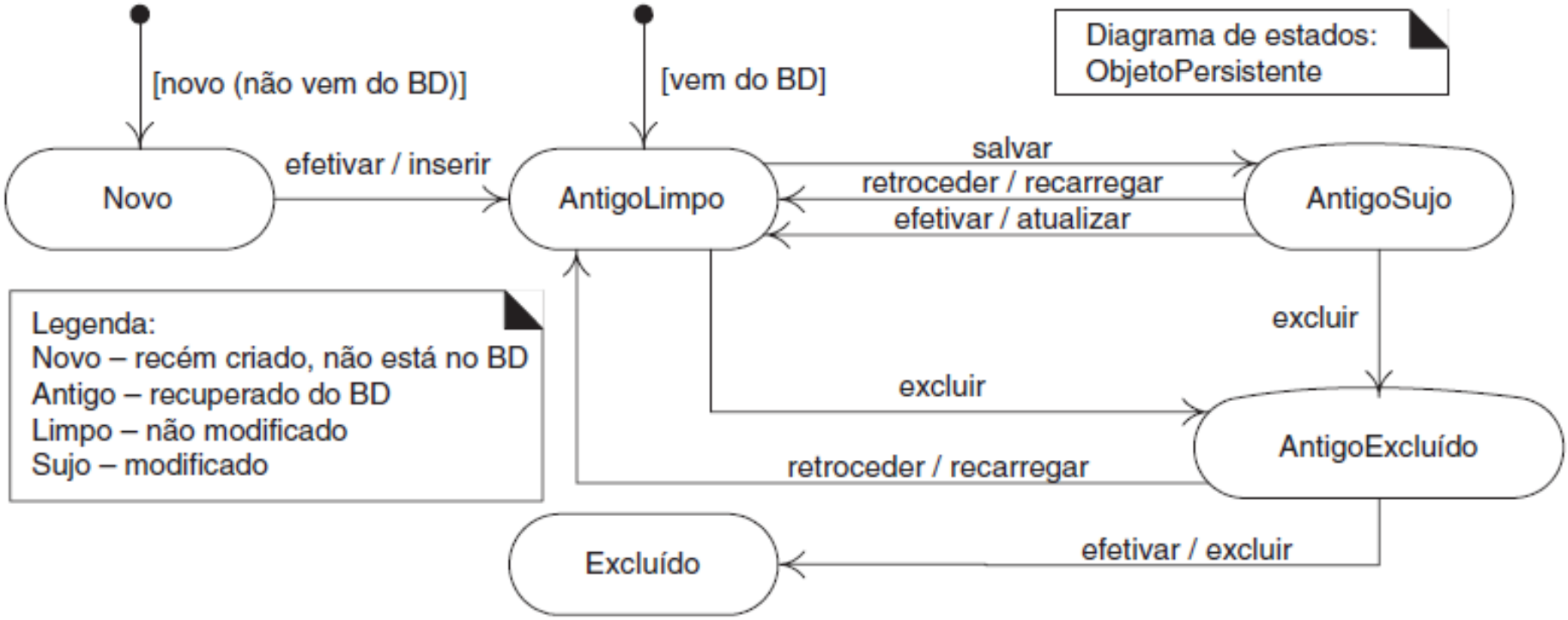
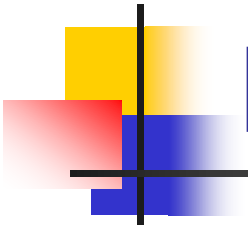


Diagrama de estados para objeto persistente

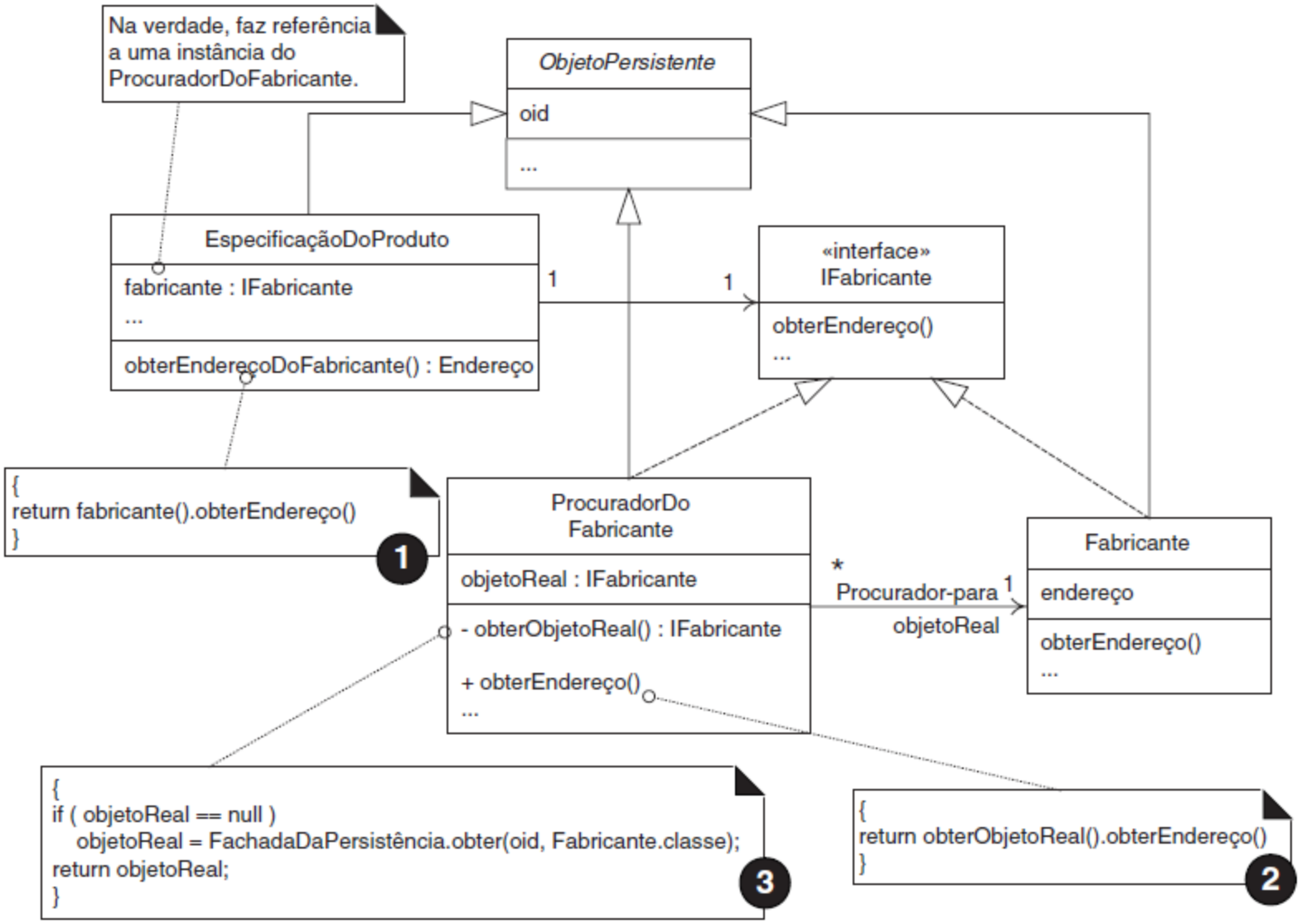




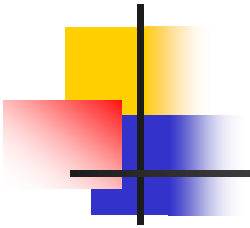
Referências Inteligentes

- Às vezes é desejável adiar a materialização de um objeto até que ela seja absolutamente necessária → materialização sob demanda
- Isso pode ser feito com o padrão GoF Proxy Virtual
- Um Proxy virtual é uma referência inteligente para o objeto real. Ele materializa o objeto real quando referenciado pela primeira vez, portanto implementa a materialização sob demanda
- É considerado uma referência inteligente porque é tratado pelo cliente como se fosse o próprio objeto.

Proxy Virtual



Como representar relacionamentos em tabelas



- Representação de objetos complexos, que possuem conexões com outros objetos, e não simplesmente atributos primitivos simples.
- Consideraremos bancos de dados relacionais
  - Associação um-para-um
    - Coloque uma chave estrangeira que seja um OID em uma ou em ambas as tabelas ou
    - Crie uma tabela associativa que registra os OIDs de cada objeto no relacionamento

Como representar relacionamentos em tabelas

- Associação um-para-muitos
  - Crie uma tabela associativa que registra os OIDs de cada objeto no relacionamento
- Associação muitos-para-muitos
  - Crie uma tabela associativa que registra os OIDs de cada objeto no relacionamento

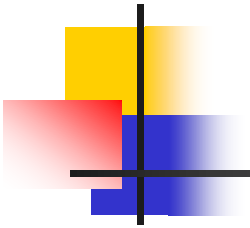
Para simplificar, usaremos sempre uma tabela associativa

Materialização de uma hierarquia de composição

- Considere Emprestimo – LinhaDoEmprestimo – CopiaDoLivro
- O que significa materializar Emprestimo?
  - Emprestimo é materializado? Ou
  - Emprestimo, suas linhas de empréstimo e suas respectivas cópias do livro são materializadas?

Materialização de uma hierarquia de composição

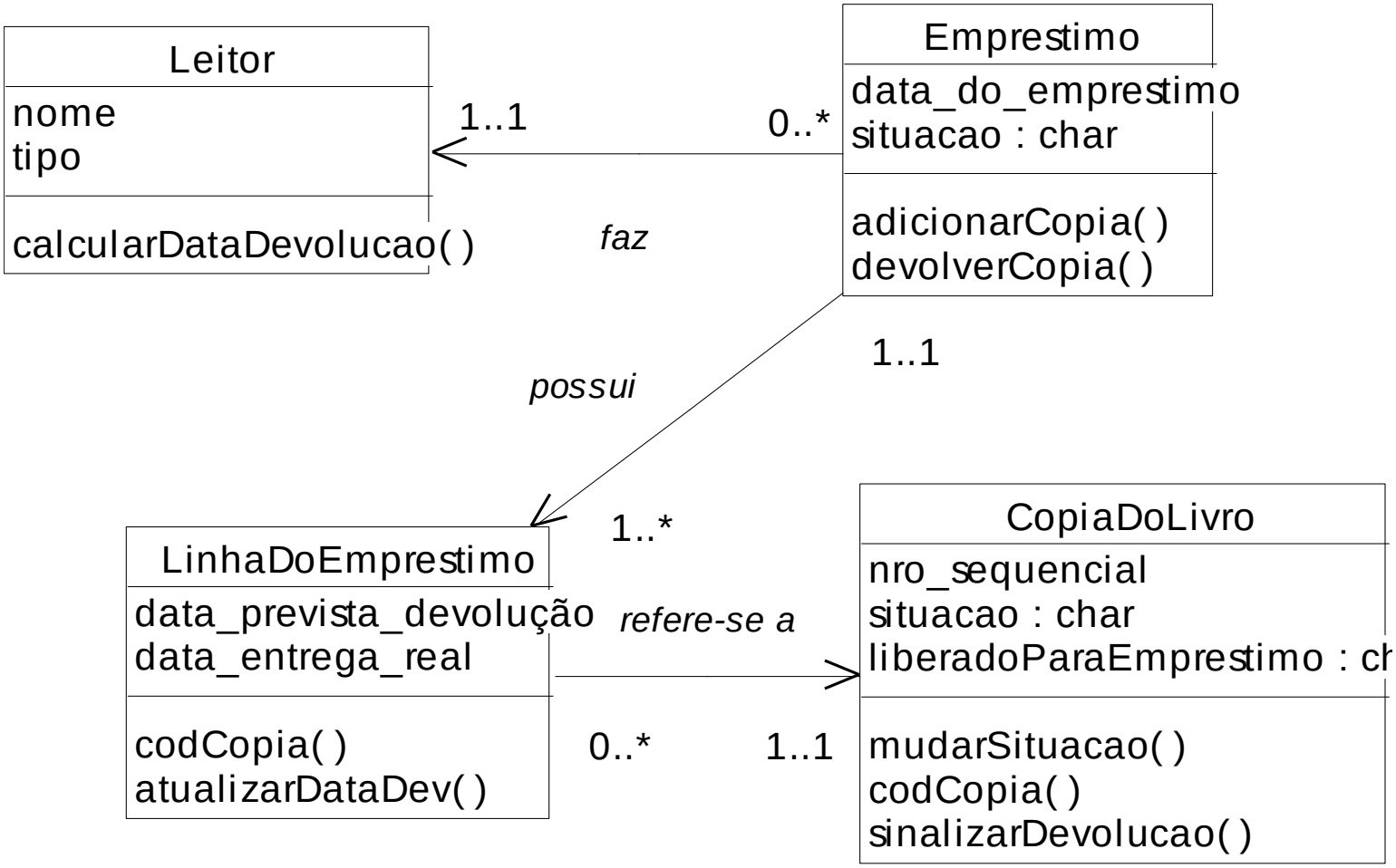
- No caso de hierarquias com muitos níveis, é possível que dezenas ou centenas de objetos relacionados precisem também ser materializados.
  - Lentidão
  - Ineficiência (espaço gasto)



Solução: Materialização sob demanda

- O padrão Instanciação de Objeto Complexo propõe adiar a materialização de objetos de acordo com as formas de acesso e requisitos de desempenho.
- Caso extremo: 100% de materialização sob demanda.
- Meio termo: materialização até 1 ou 2 níveis da hierarquia. Se for usado um Intermediário diferente para cada objeto persistente, é possível decidir esse nível individualmente.

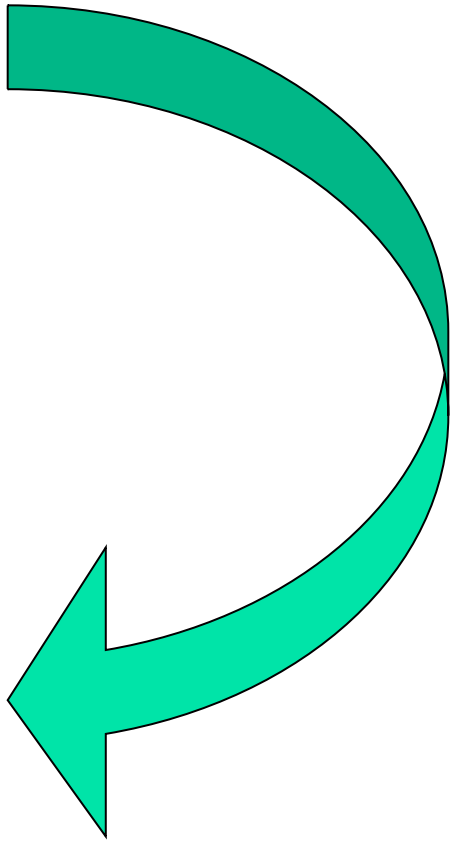
Exemplo: Emprestimo



Exemplo: Materialização de Emprestimo

Emprestimo

OID	Dt_emprestimo	situacao
E1	10/05/2005	P
E2	15/05/2005	P



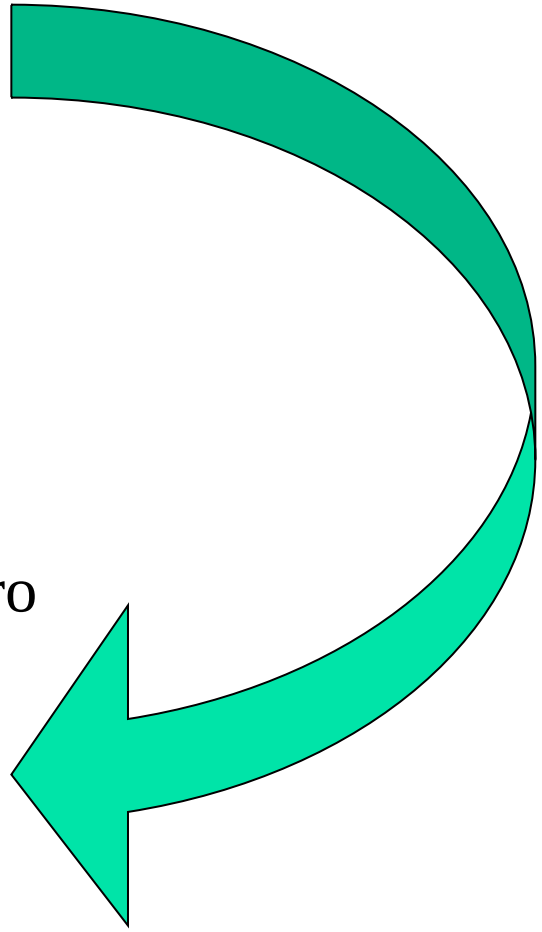
LinhaDeEmprestimo

OID	OID-e	Dt_prev_dev	Dt_entrega_real
Lie1	E1	10/05/2005	
Lie2	E1	15/05/2005	
Lie3	E2	11/05/2005	

Exemplo: Materialização de Emprestimo

LinhaDeEmprestimo

OID	OID-e	Dt_prev_dev	Dt_entrega_real
Lie1	E1	10/05/2005	
Lie2	E1	15/05/2005	
Lie3	E2	11/05/2005	



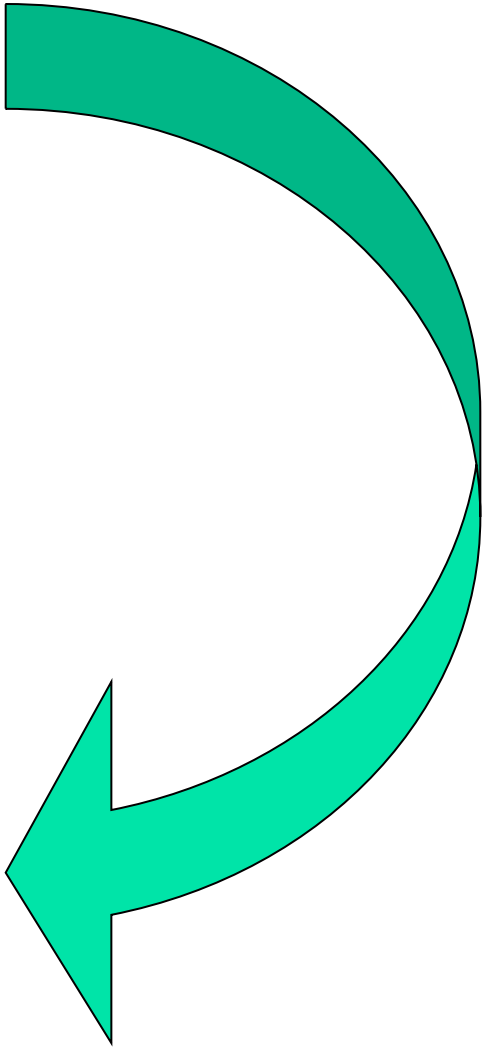
LinhaDeEmprestimo para CopiaDeLivro

Lie-OID	CopiaLivro-OID
Lie1	CL1
Lie2	CL2

Exemplo: Materialização de Emprestimo

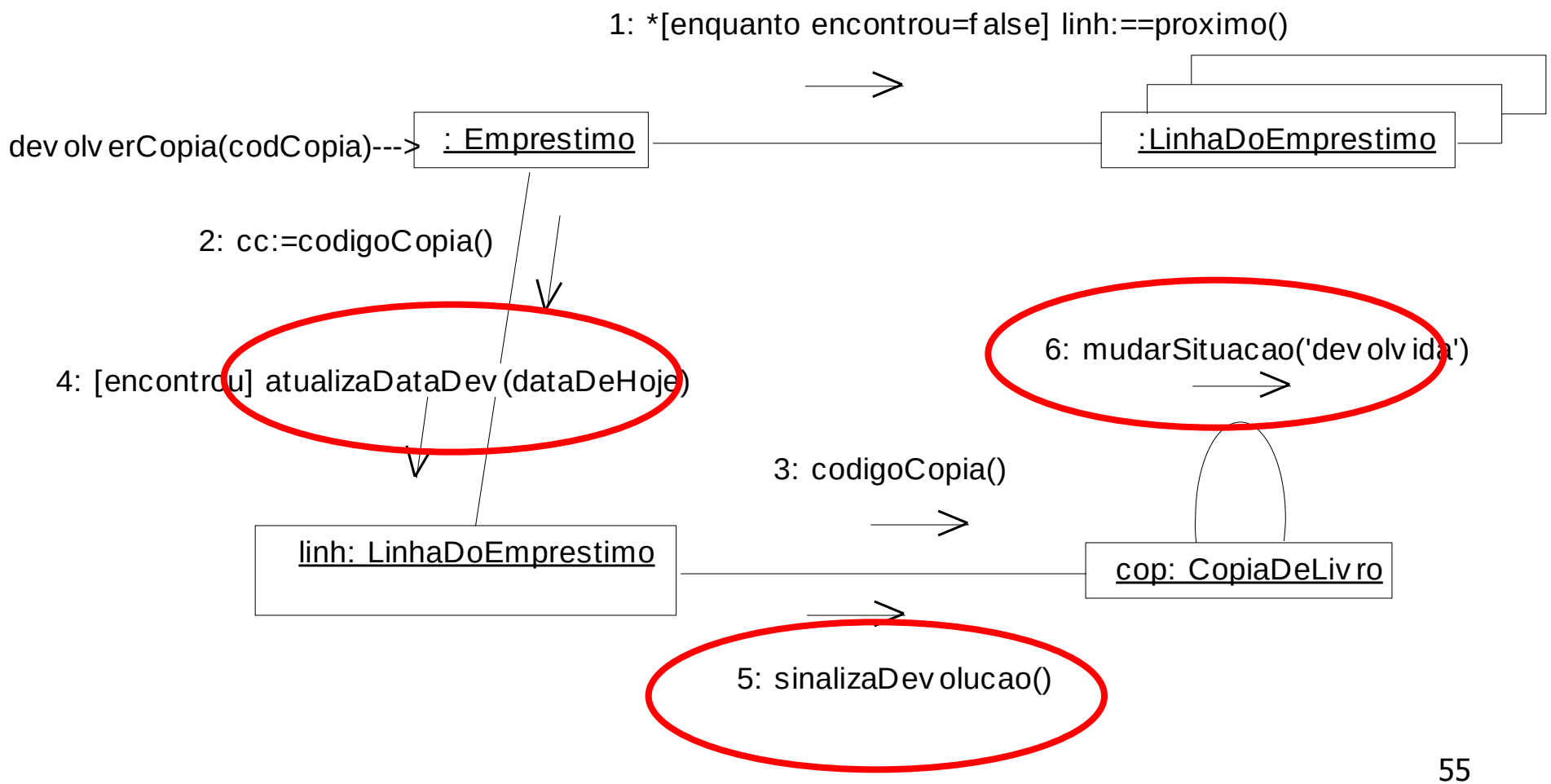
LinhaDeEmprestimo para CopiaDeLivro

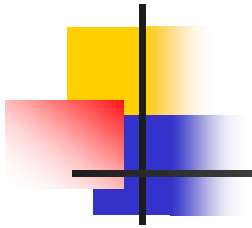
Lie-OID	**CopiaLivro-OID**
Lie1	CL1
Lie2	CL2



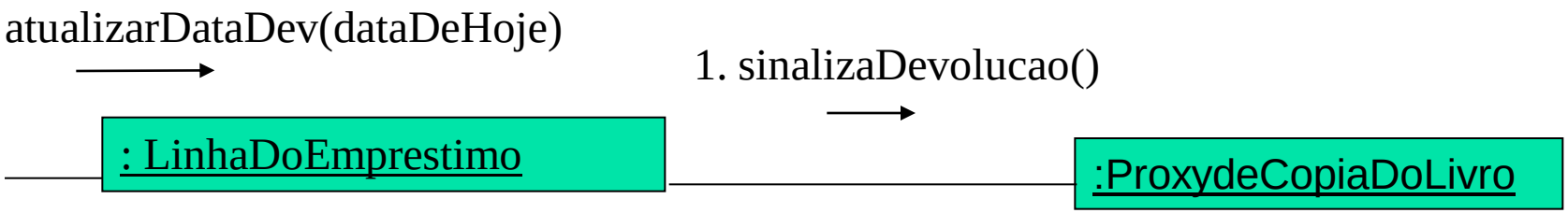
CopiaDeLivro

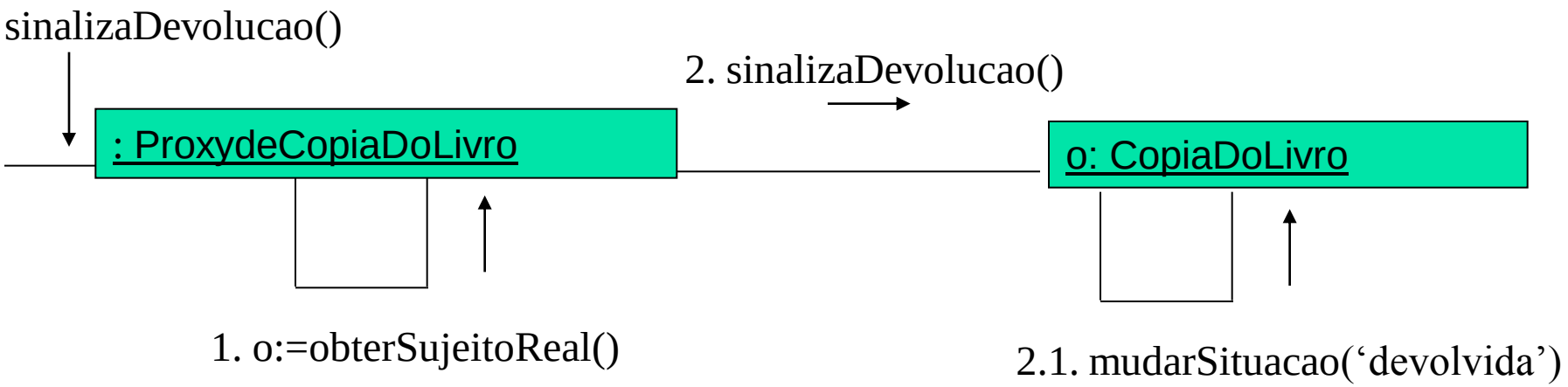
OID	**nroSequencial**	**situacao**	**libParaEmprest**
CL1	1	P	S
CL2	2	C	S

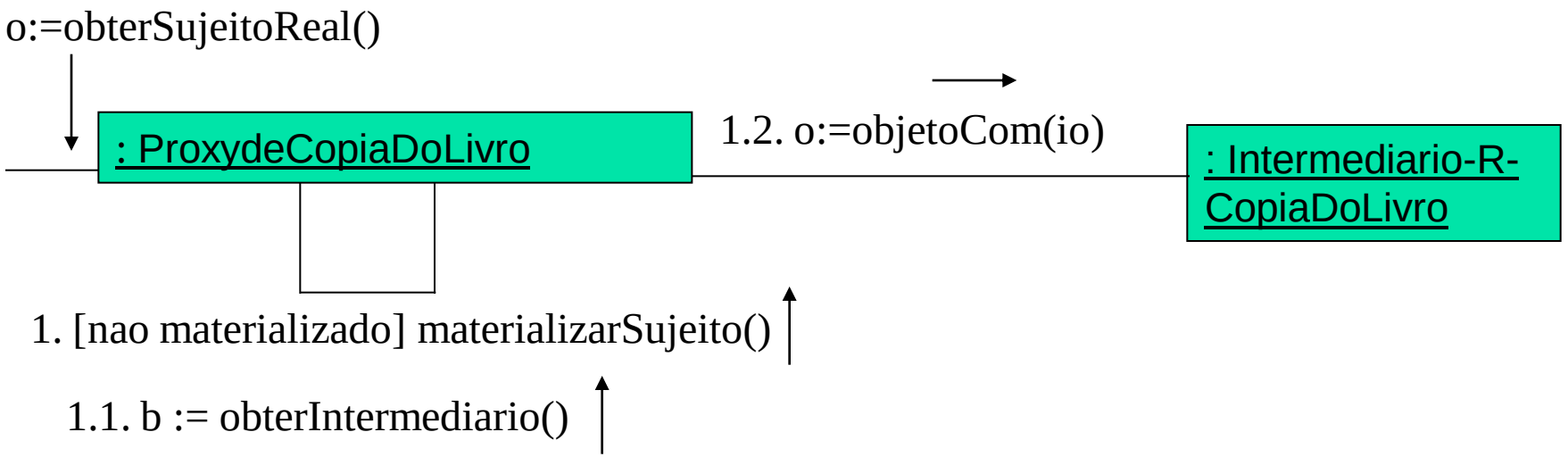

1: *[enquanto encontrou=false] linh:==proximo()
devolverCopia(codCopia)--->
: Emprestimo
:LinhaDoEmprestimo
2: cc:=codigoCopia()
4: [encontrou] atualizaDataDev(dataDeHoje)
6: mudarSituacao('devolvida')
3: codigoCopia()
linh: LinhaDoEmprestimo
cop: CopiaDeLivro
5: sinalizaDevolucao()

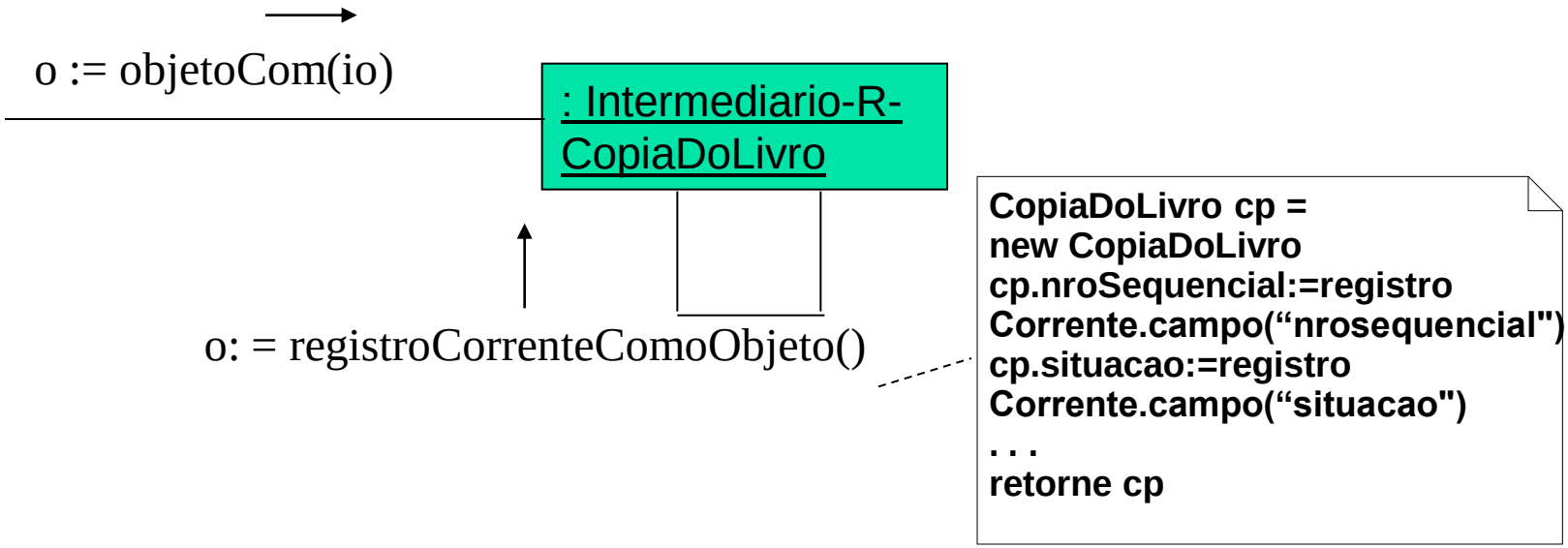


- Usando um Proxy para CopiaDoLivro, este referenciará o proxy e não o objeto real. Assim, o objeto CopiaDoLivro só será materializado, por exemplo, quando o método sinalizaDevolucao() for invocado, pois ele precisará chamar o método mudarSituacao(), que precisará do SujeitoReal para poder modificar a situação do livro


atualizarDataDev(dataDeHoje)
1. sinalizaDevolucao()
: LinhaDoEmprestimo
:ProxydeCopiaDoLivro


sinalizaDevolucao()
2. sinalizaDevolucao()
: ProxydeCopiaDoLivro
o: CopiaDoLivro
1. o:=obterSujeitoReal()
2.1. mudarSituacao('devolvida')


o:=obterSujeitoReal()
: ProxydeCopiaDoLivro
1.2. o:=objetoCom(io)
: Intermediario-R-CopiaDoLivro
1. [nao materializado] materializarSujeito()
1.1. b := obterIntermediario()


o := objetoCom(io)
: Intermediario-R-CopiaDoLivro
o: = registroCorrenteComoObjeto()
CopiaDoLivro cp =
new CopiaDoLivro
cp.nroSequencial:=registro
Corrente.campo("nrosequencial")
cp.situacao:=registro
Corrente.campo("situacao")
. . .
retorne cp