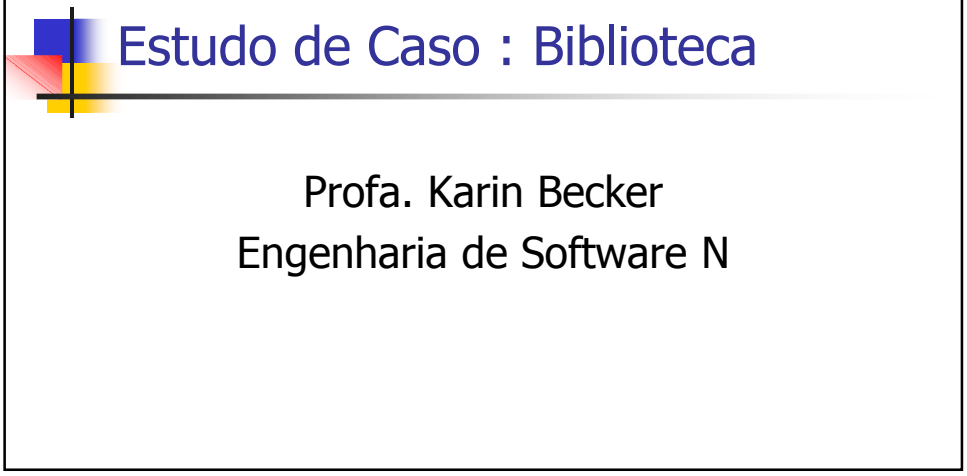
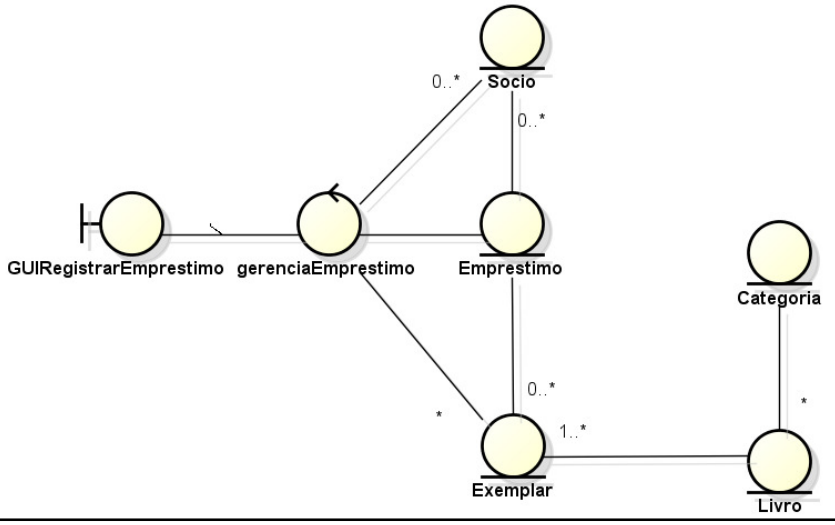

Estudo de Caso : Biblioteca
Profa. Karin Becker
Engenharia de Software N


Modelo de Análise

Registrar Emprestimo



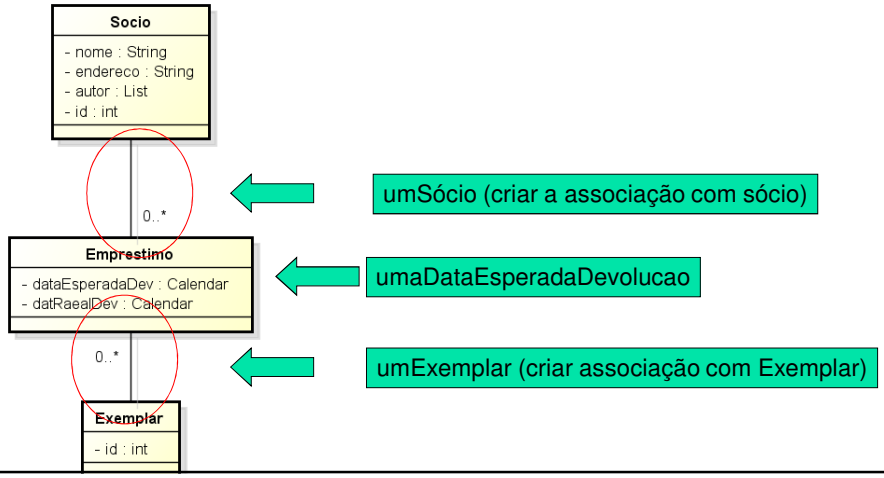
Modelo de Projeto

Sumário

- Thinking in the small
  - Problemas pontuais do projeto OO
- Thinking in the large
  - Decisões arquiteturais

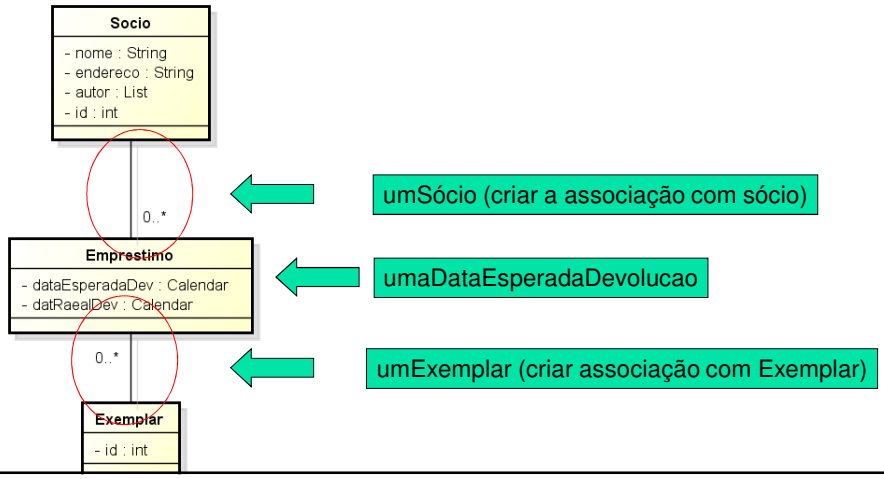
Thinking in the small ...

- O que eu preciso para criar uma instância de Empréstimo?



Thinking in the small ...

- Mas e de onde eu consigo esta informação?



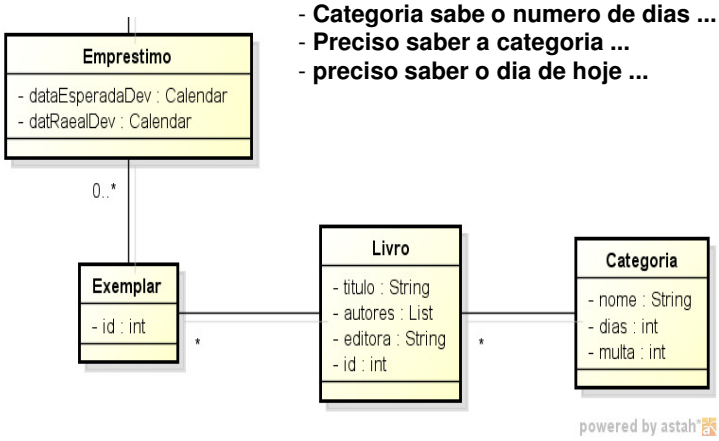
Thinking in the small ...

- umSocio
  - via interface, usuário me fornece o nome do sócio ... Tenho que encontrar o objeto umSocio que tenha aquele nome
- umExemplar
  - via interface, usuário me fornece o identificador do exemplar ... Tenho que encontrar o objeto umExemplar que tenha aquele identificador
- dataEsperadaDevolucao
  - Não é fornecida via interface
  - Espera-se que o sistema calcule : hoje + numero de dias definido na categoria à qual o exemplar do livro pertence

Thinking in the small

dataEsperadaDev

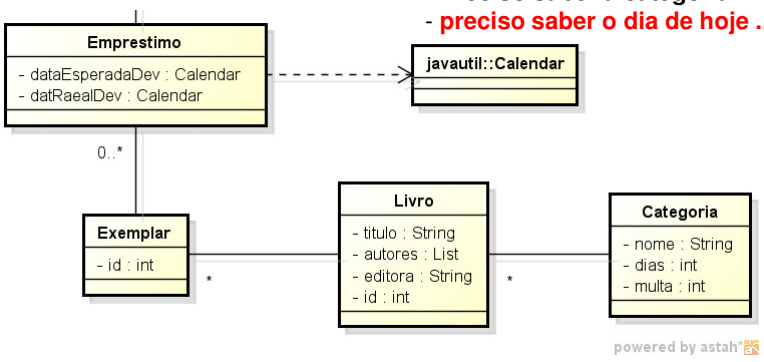
-Especialistas
- **Categoria sabe o numero de dias ...**
- **Preciso saber a categoria ...**
- **preciso saber o dia de hoje ...**



Thinking in the small

dataEsperadaDev

-Especialistas
- **Categoria sabe o numero de dias ...**
- **Preciso saber a categoria ...**
- **preciso saber o dia de hoje ...**



Calendar é uma das muitas escolhas feitas com base na tecnologia

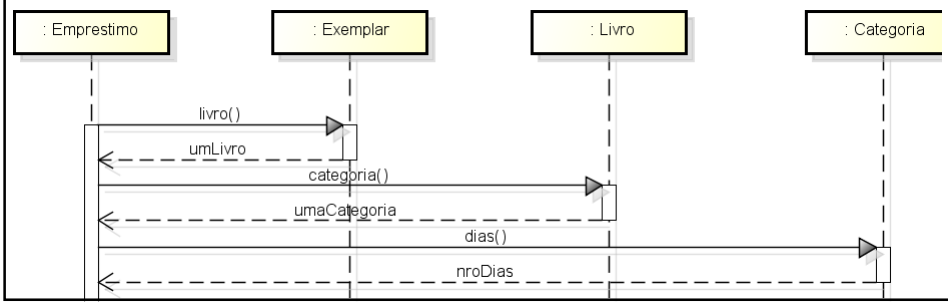
hoje : Calendar.getInstance()

dataDev : hoje.add(.....)

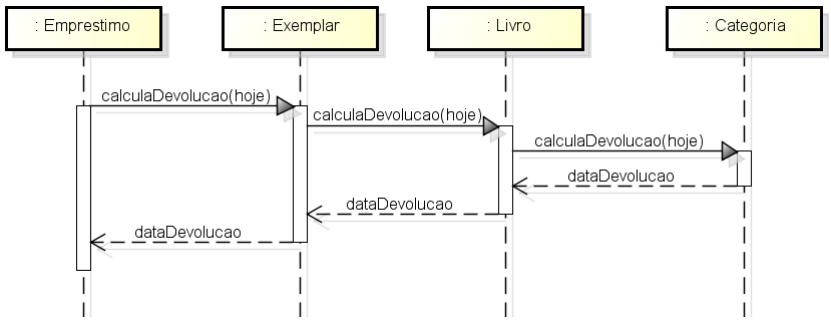
hoje + nroDias definido por categoria

- Exemplar
  - livro()
- Livro
  - categoria()
- Categoria
  - dias()
- umExemplar.livro().categoria().dias()

Emprestimo é Classe DEUS
As demais classes são estruturas de dados de luxo

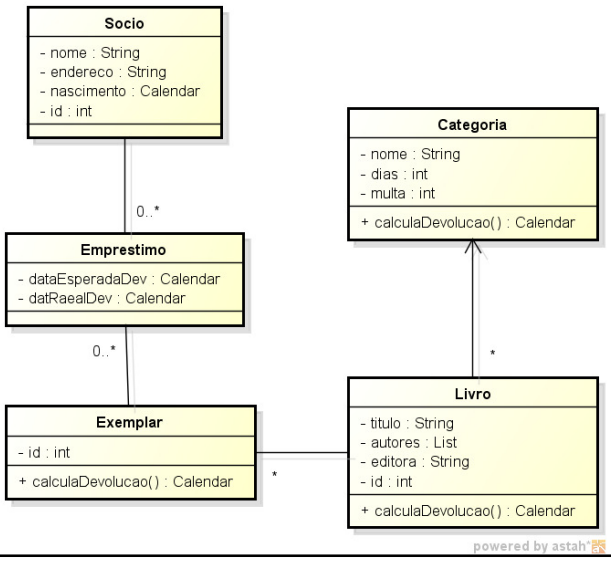


hoje + nroDias definido por categoria



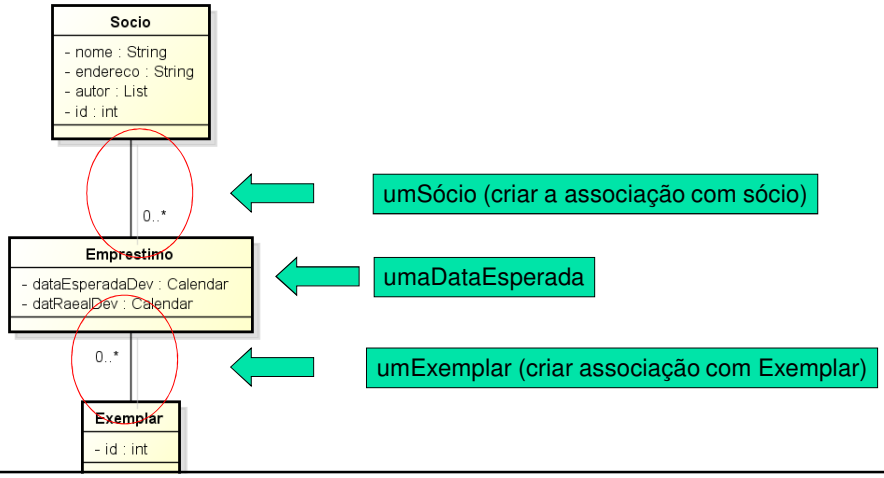
• Categoria podia ter criado a data hoje (ao invés de recebê-la como argumento)
• distribuição de responsabilidades mais equilibrada
• nro de mensagens trocadas é igual nas duas opções
• se trocar a forma de calcular a data (por exemplo, com base no exemplar, Emprestimo não é afetado)
• mensagens trocadas com Calendar não são mostradas pelo nível de detalhe de implementação

Assim sendo ...



Thinking in the small ...

- e para umSocio e umExemplar?



Thinking in the small

- umSocio
  - via interface, usuário me fornece o nome do sócio ... Tenho que encontrar o objeto umSocio que tenha aquele nome
- Problemas
  - Como converter o nome que vem da interface no objeto Sócio que tem aquele nome?
    - Valor vs. Objeto
    - Objeto vs. Coleção de objetos

Thinking in the small ...

- Objeto vs. valor
  - "joao" é um valor que a propriedade nome pode assumir
  - Qual seu nome?

"joao"

"joao"

"maria"

Thinking in the small ...

- Objeto vs. coleção
  - Uma coleção de pessoas
  - Qual destes objetos tem nome "maria"?

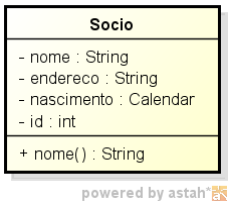
"joao"

"joao"

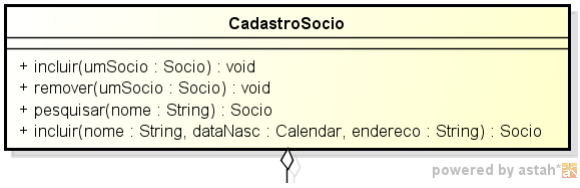
"maria"

Thinking in the small

- Socio : um sócio



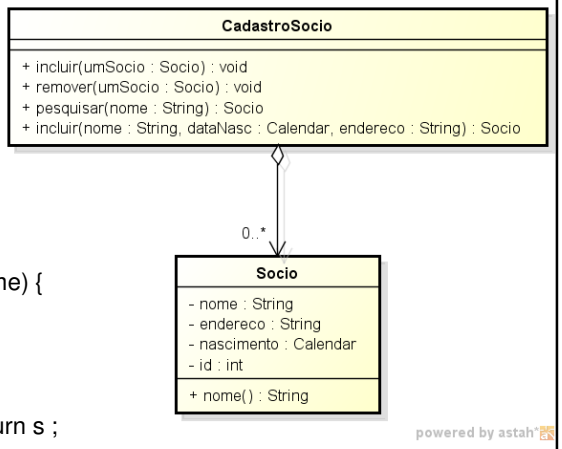
- Cadastro de Socios: uma coleção de sócios



Obs: para simplificar, nomes são únicos

Thinking in the small ...

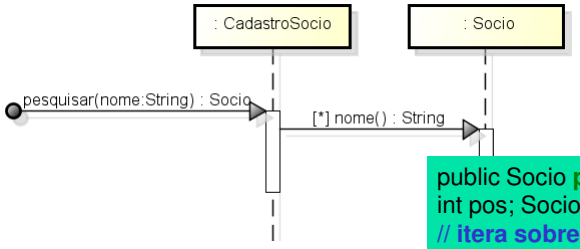
```
public class CadastroSocios {
// usa lista
private ArrayList<Socio> lista ;

public CadastroSocios ( ) {
lista = new ArrayList<Socio > ( ) ;
}
. . .

public Socio pesquisar(String nome) {
int pos; Socio s;
// itera sobre a lista
while(int pos < lista . size ( ) ) {
s = lista . get(pos ) ;
if(s .nome ( ) . equals(nome ) ) return s ;
pos++;}
return null ;
}
.....}
```



- **outras opções de estrutura de coleção (HashMap)**
- **pressupõe tudo na memória (sem persistência)**

Thinking in the small

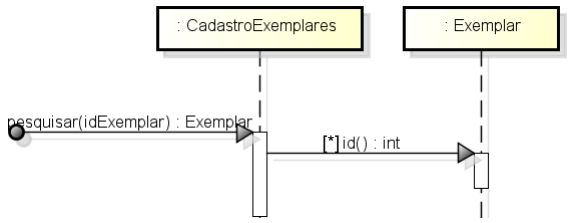
- umSocio
  - Suponha que eu tenho um nome que venha da interface ... Como acho o objeto Socio correspondente?




```
public Socio pesquisar(String nome) {
int pos; Socio s;
// itera sobre a lista
while(int pos < lista . size ( ) ) {
s = lista . get(pos ) ;
if(s .nome ( ) . equals(nome ) ) return s ;
pos++;}
return null ;
}
```

Thinking in the small

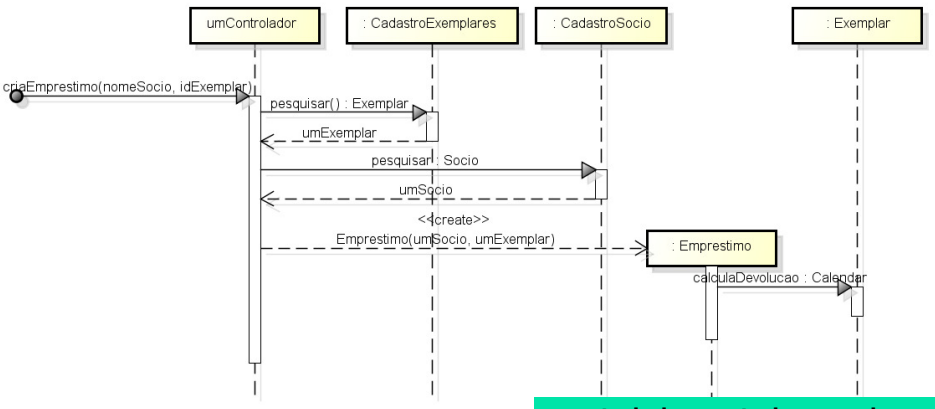
- Seguindo o mesmo raciocício : umExemplar
  - Suponha que eu tenho um id de exemplar que venha da interface ... Como acho o objeto Exemplar correspondente?



Thinking in the small ...

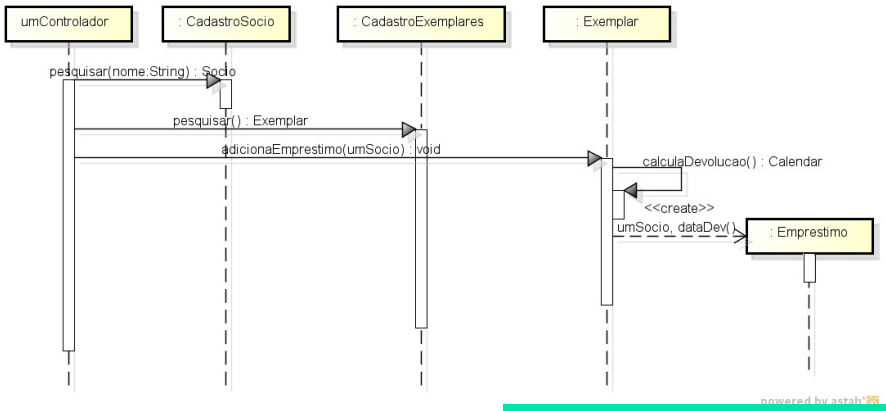
- Pronto
  - Já tenho a data
  - Já tenho o exemplar
  - Já tenho o sócio
  - Agora é só criar o objeto Emprestimo

Criando um emprestimo – Opção 1



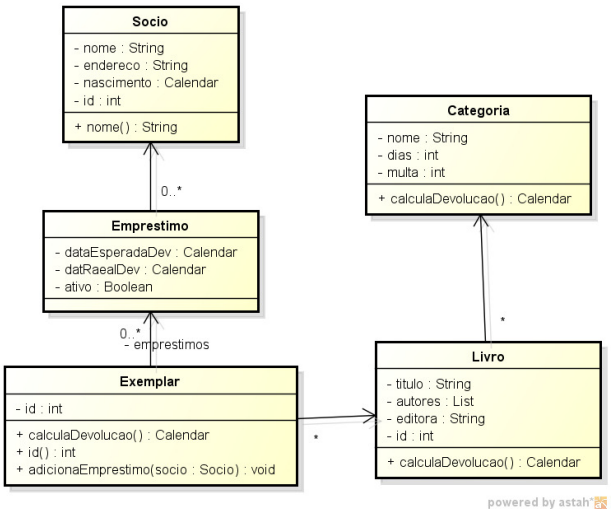
- **controlador controla pesquisas**
- **controlador responsavel pela criação do emprestimo**

Criando um emprestimo : opção 2

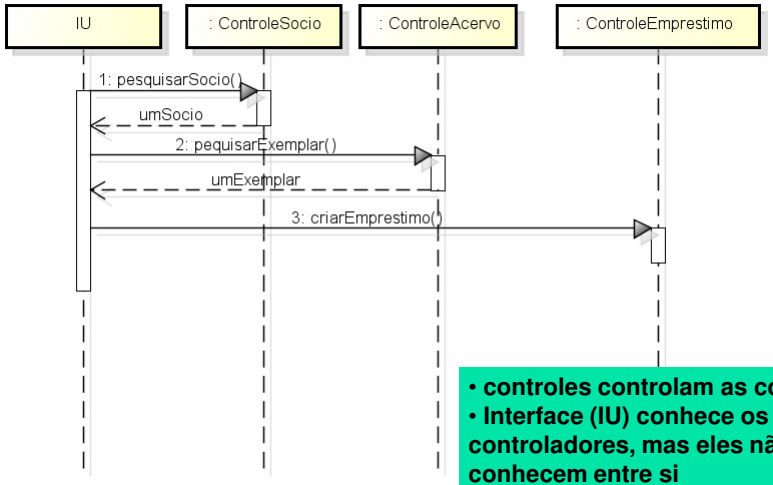


- **controlador controla pesquisas**
- **controlador delega a Exemplar a criação do emprestimo**

Evoluindo as classes – opção 2

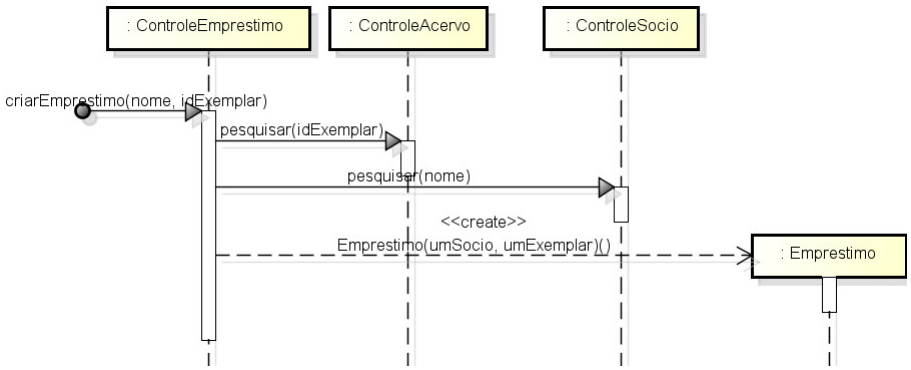


Alternativas para os controladores?



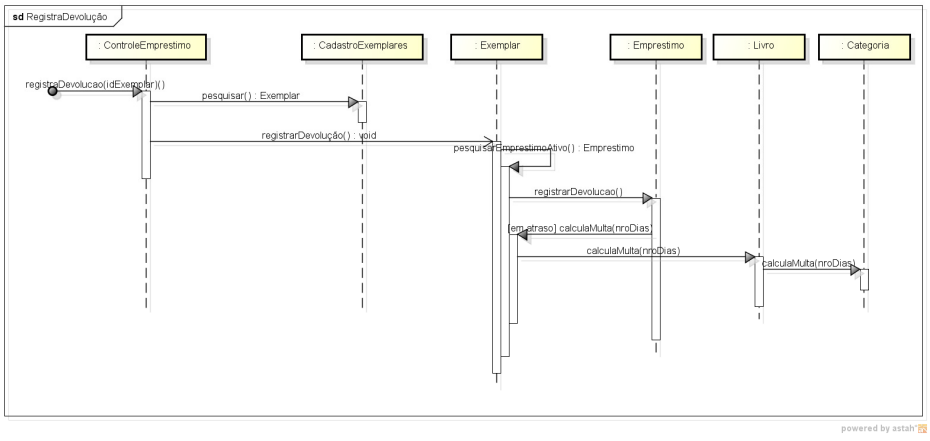
- **controles controlam as coleções**
- **Interface (IU) conhece os 3 controladores, mas eles não se conhecem entre si**
- **tomar cuidado para que a interface não acabe controlando a lógica da transação**

Alternativas para os controladores?



- **controles controlam as coleções**
- **ControleEmprestimo deve conhecer os outros dois controladores**
- **tomar cuidado pois os controles não podem evoluir independentemente**

Registrar Devolução



Registrar Devolução

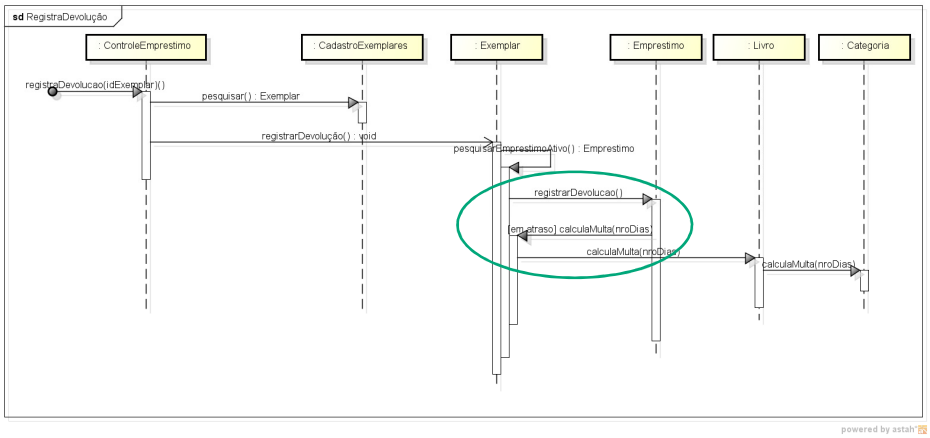
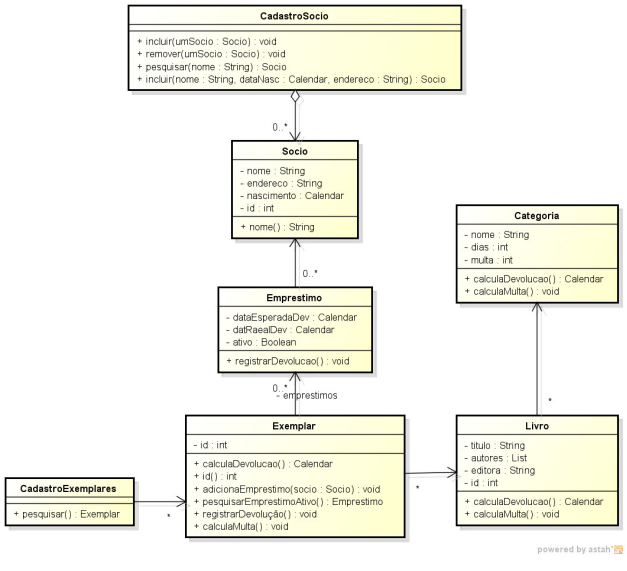
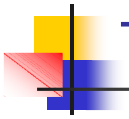


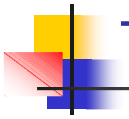
Diagrama de Classes





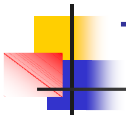
Thinking in the Large ...

- Requisições que vem da interface ...
  - Quem trata o evento da interface?
  - Como manter a interface desacoplada da semântica (evolução, portabilidade, etc) ?
- Objeto que corresponde a um dado valor
  - Valor vs. objeto
  - como tratar as coleções?
  - Na prática, raramente todos os objetos permanecem em memória



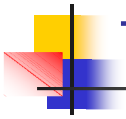
Thinking in the large ...

- Tomar a decisão sobre as classes que lidam com estes problemas requer decisões sobre a arquitetura !!
  - Divisão em subsistemas
    - Responsabilidade
    - Interfaces
    - Dependências
  - As classes recheiam os subsistemas
  - Processo iterativo
    - Arquitetura
    - Projeto detalhado



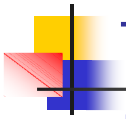
Thinking in the large ...

- No meu modelo de análise, identifiquei três pacotes de serviço
  - Sócios
  - Acervo
  - Empréstimo
- Vou adaptar esta divisão em meu projeto
  - Sócios + Empréstimos
  - Acervo
- Vou usar o padrão arquitetural camadas também



Thinking in the large

- Padrões arquiteturais : camadas
  - ajuda a pensar sobre responsabilidades genéricas (o papel de cada camada)
  - "separation of concerns"
  - Mas ...
    - Vai criar mais classes, algumas só para redirecionar chamadas
      - Cada camada é uma fachada para o resto
    - Vai criar um overhead de comunicação
      - muitas indireções
  - Tenho que estabelecer compromissos !!



Thinking in the large

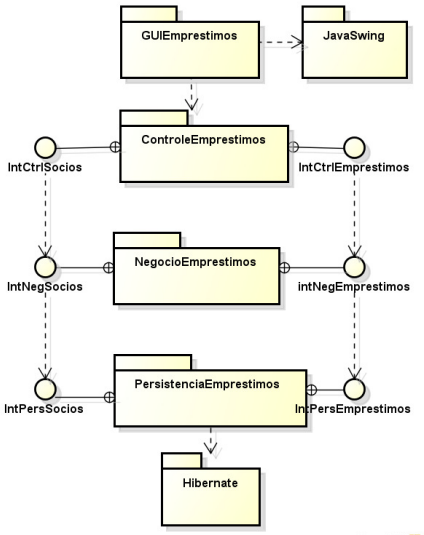
- Camada Interface
  - Telas, botoes, combos, menus, etc ...
  - Vou usar Java Swing
- Camada Controle
  - Vai isolar a apresentação do resto
  - Vai funcionar como uma fachada para os objetos de domínio
  - Vai controlar transações
- Camada Domínio
  - Vai ter os objetos do domínio necessários às transações em memória
- Camada Persistência
  - Vai fazer tratar das questões de persistência dos objetos
  - Vou usar o padrão DAO (Data Access Object)
  - Vou usar um framework de persistência (Hibernate, Toplink, OpenJPA, etc)

Padrão Arquitetural Camadas
Padrões de Projeto (Fachada, DAO, Singleton)

O padrão DAO

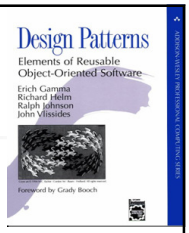
- **DAO** (Objeto de Acesso a Dados)
  - separar das regras da aplicação a tecnologia de persistência de dados
  - promover isolamento e flexibilidade, principalmente, quando se deseja, por exemplo, mudar o SGBD
  - Resolver a impedância entre LPOO e bancos de dados relacionais, entre elas
    - Chaves primárias
    - Associações por chaves estrangeiras
- Ao considerar a persistência, cada uma de nossas coleções funciona como um DAO
  - DAOSocio (incluir, remover, pesquisar ....)
  - DAOExemplar (incluir, remover, pesquisar ....)
- Pode ter mais ou menos inteligência
  - Persistência pura
  - Montagem de objetos na memória (ex: exemplar – livro - categoria)

Thinking in the large ...



E agora?
- **Continuo tomando decisões e revisando a architetura**
- **Padrões de projeto/arquiteturais podem me ajudar ... reuso de experiência**

Padrões GOF



Scope	Purpose: Creational	Purpose: Structural	Purpose: Behavioral
Class	Factory Method	Adapter (class)	Interpreter Template Method
Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Scope: domain over which a pattern applies

Purpose: reflects what a pattern does