

# *O Que é Arquitetura de Software?*



- se refere aos principais elementos que compõem um sistema de software e os inter-relacionamentos entre eles

# *O Que é um Modelo?*



- Uma representação simplificada de alguma coisa que elimina características consideradas menos importantes
- Abstrair significa simplificar, removendo coisas menos importantes para se concentrar naquilo que é mais importante



# *Uma Visão Geral de UML*

Prof. Fabio Kon

IME/USP

Apresentação baseada em slides de Kendall V. Scott

# *Linguagem de Modelagem Unificada*

UML é uma linguagem padrão da OMG para

- visualização,
  - especificação,
  - construção e
  - documentação
- de software orientado a objetos.

# *Visualização*



- A existência de um modelo visual facilita a comunicação e faz com que os membros de um grupo tenham a mesma idéia do sistema.
- Cada símbolo gráfico tem uma semântica bem definida.

# *Especificação*



É uma ferramenta poderosa para a especificação de diferentes aspectos arquiteturais e de uso de um sistema.

# *Construção*



- Geração automática de código a partir do modelo visual
- Geração do modelo visual a partir do código
- Ambientes de desenvolvimento de software atuais permitem:
  - movimentações em ambos sentidos e
  - manutenção da consistência entre as duas visões.

# Documentação



Pode incluir artefatos como:

- *Deliverables* (documentos como especificação de requisitos, especificações funcionais, planos de teste, etc.).
- Materiais que são importantes para controlar, medir, e refletir sobre um sistema durante o seu desenvolvimento e implantação.



# *Descrição Arquitetônica*

UML oferece uma forma padrão de se desenhar as “plantas” (como em arquitetura) de um sistema de forma a incluir

- aspectos abstratos (processos de negócio, funcionalidades do sistema)
- aspectos concretos (classes C++/Java, esquemas de bancos de dados, componentes de software reutilizáveis)

# *Razões para Modelar*

- Comunicar a estrutura e o comportamento desejado de um sistema.
- Visualizar e controlar a arquitetura de um sistema.
- Para melhorar o nosso entendimento de um sistema e, assim, expor oportunidades para melhorias e reutilização.
- Para administrar os riscos e *trade-offs*.

# *Diagramas Estruturais*

Usados para visualizar, especificar, construir e documentar aspectos estáticos de um sistema

- diagrama de classes
- diagrama de pacotes
- diagrama de objetos
- diagrama de componentes
- diagrama de implantação

# *Usos Comuns para Diagramas de Classes*

- Modelar o vocabulário do sistema, em termos de quais abstrações fazem parte do sistema e quais caem fora de seus domínios.
- Modelar as colaborações/interações (sociedades de elementos que trabalham em conjunto oferecendo algum comportamento cooperativo).
- Modelagem lógica dos dados manipulados pelo sistema (servindo de base para a definição formal do modelo da base de dados).

# *Notação para Classes*



<b>Nome</b>
Atributos
Operações

# *Notações Alternativas*

Nome
Atributos
<i>Operações</i>

<b>Nome</b>
-------------

<b>Nome</b>
Atributos
Operações
Responsabilidades

itálico → abstrata

# *Especificação do Acesso*

Nome
+ atrib1 - atrib2
+ op1 - op2 # op3

+ public

- private

# protected

# *Relacionamentos*



São conexões entre classes:

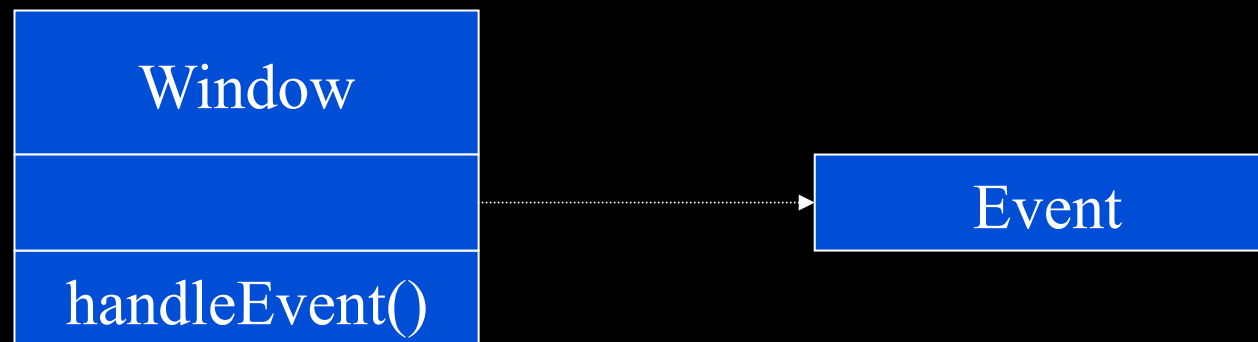
1. dependência
2. generalização
3. associação



# *Dependência*

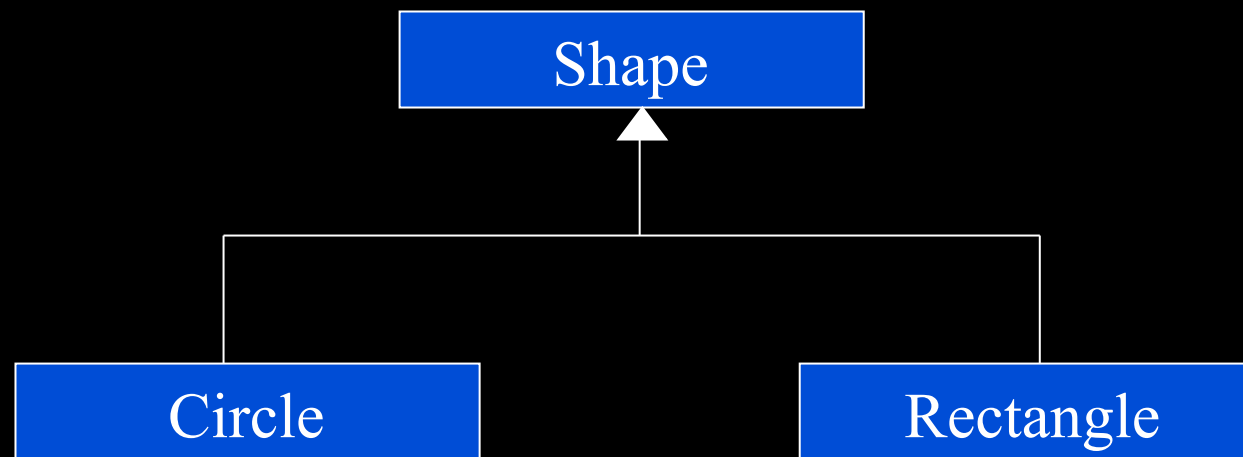
É uma relação do tipo “usa” na qual mudanças na implementação de uma classe podem causar efeitos em outra classe que a usa.

Exemplo: uma classe usa a outra.



# Generalização

É uma relação do tipo “é um” entre uma coisa geral (superclasse) e uma coisa mais específica (subclasse).



# Associação

É uma relação estrutural na qual classes ou objetos estão interconectados.

Uma associação entre objetos é chamada de uma ligação (*link*).



# *Ornamentos para Associações*



- nome
- papel
- multiplicidade
- agregação
- composição

# *Nome da Associação*

descreve a natureza da relação:



pode indicar a direção:



# *Papéis*



- Classes e objetos podem assumir papéis diferentes em diferentes momentos.



# *Multiplicidade*

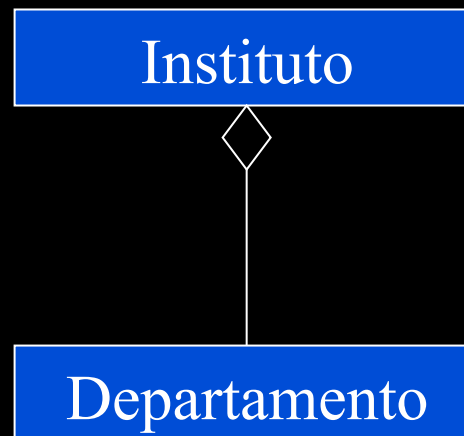
- Valores possíveis: valor exato, intervalo, ou \* para “muitos”.

- Exemplo:



# Agregação

É uma relação do tipo “todo/parte” ou “possui um” na qual uma classe representa uma coisa grande que é composta de coisas menores.

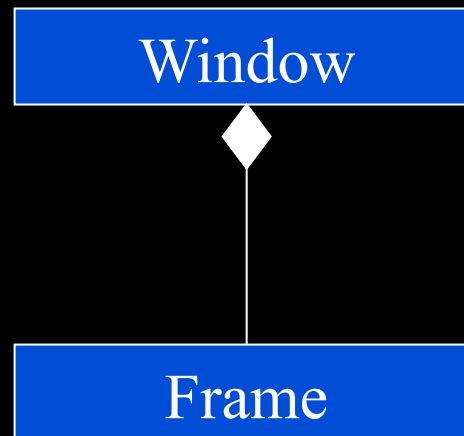


( diamante vazio )



# Composição

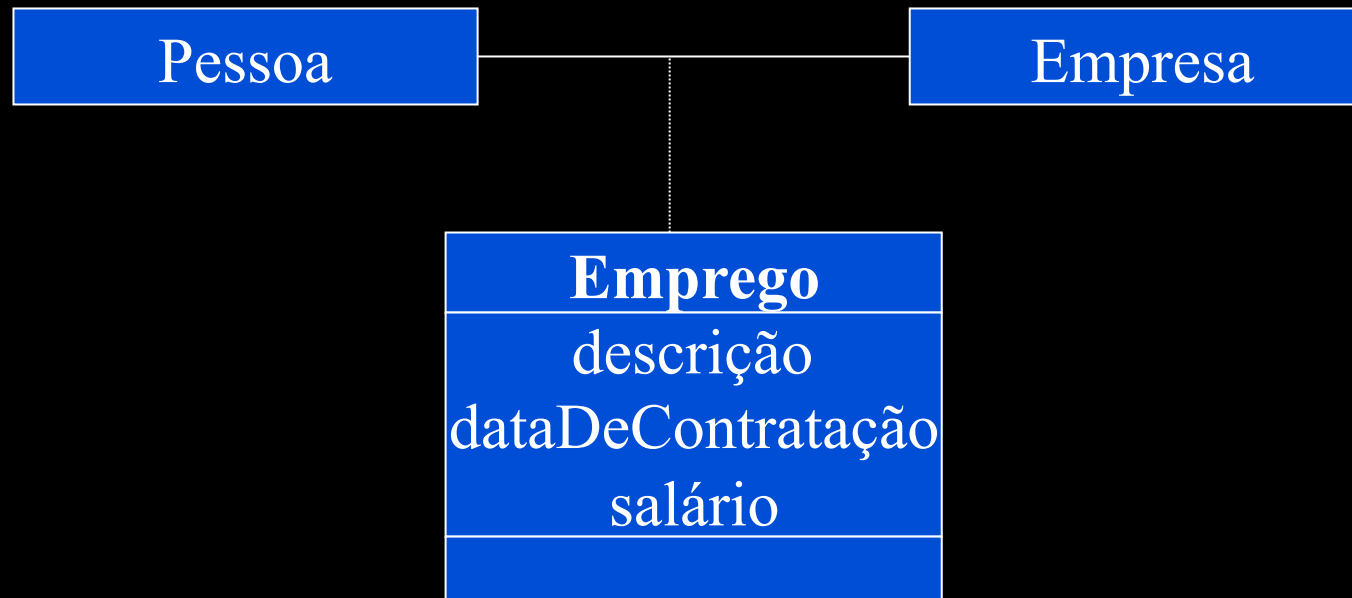
É um tipo especial de agregação na qual as partes são inseparáveis do todo.



( diamante cheio )

# *Classes de Associação*

Uma classe de associação possui as propriedades de classes e de associações:



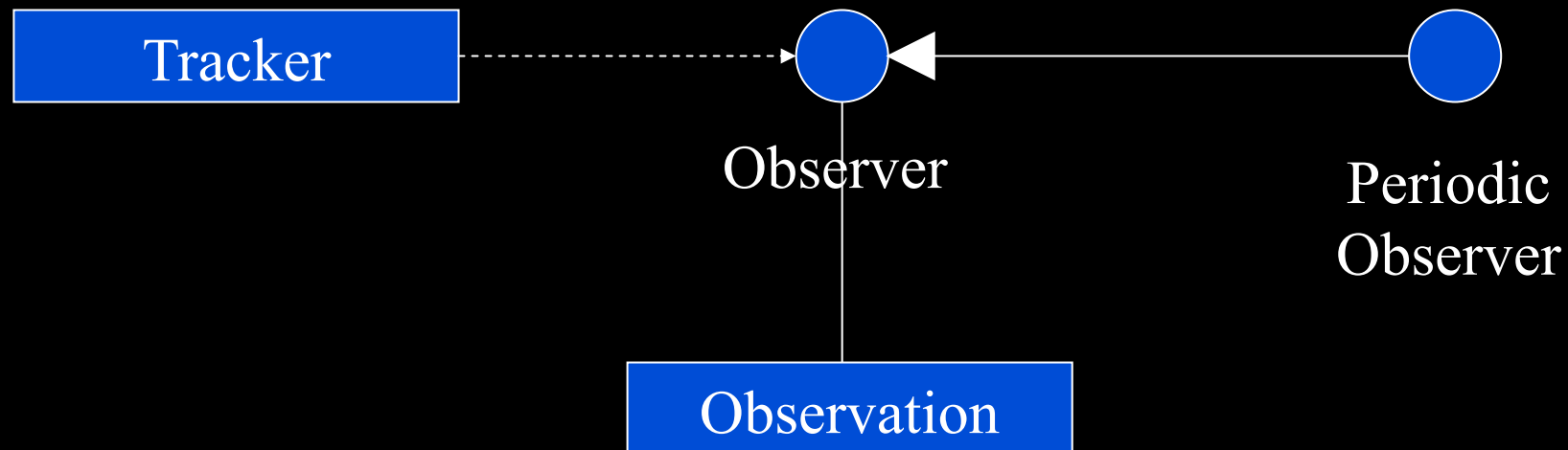
# Interfaces

É uma coleção de operações que possui um nome. É usada para especificar um tipo de serviço sem ditar a sua implementação.



# *Interfaces e Relacionamentos*

Uma interface pode participar de generalizações, associações e dependências.

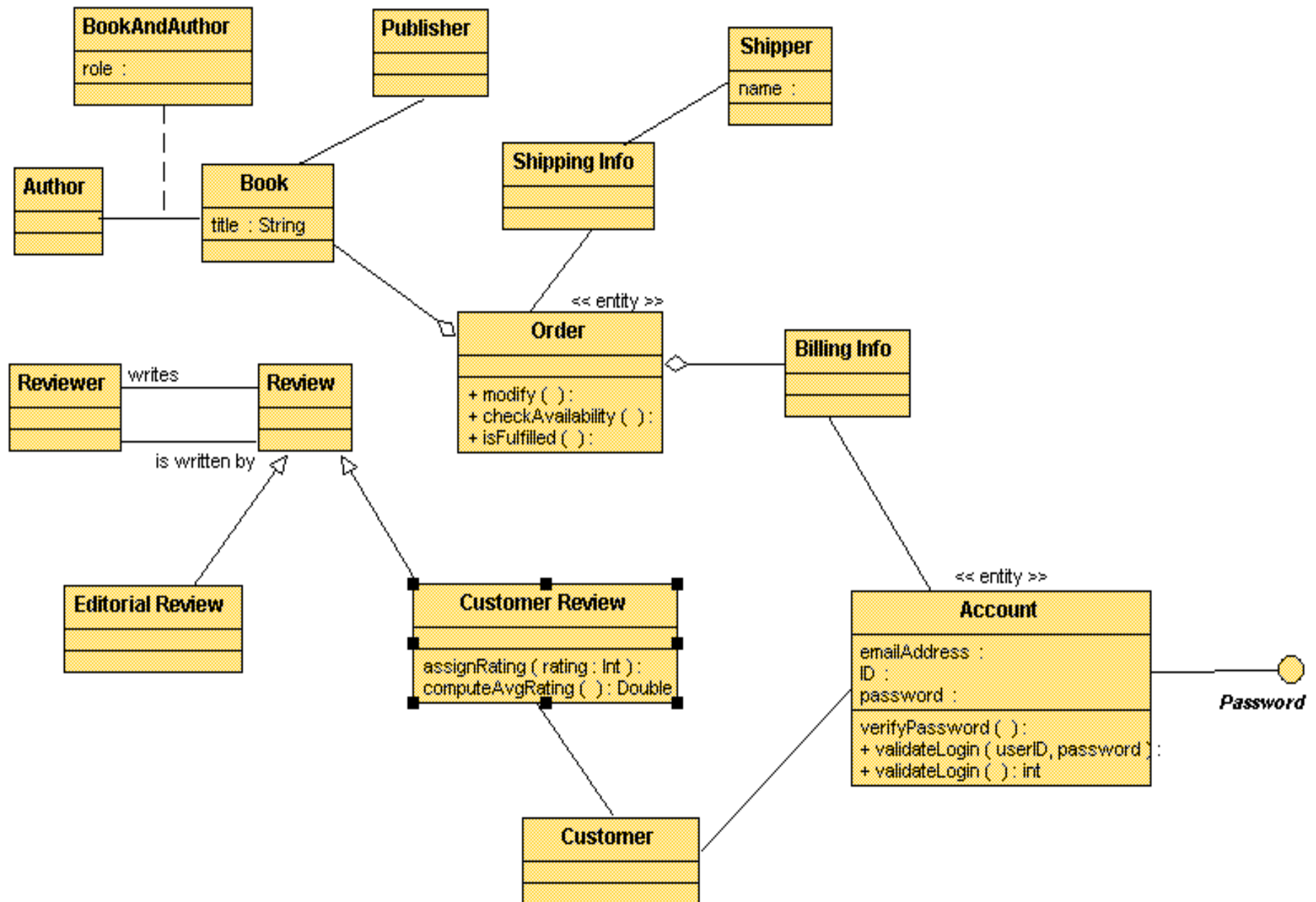


# Realização

É uma relação entre uma interface e a classe que a implementa, i.e., que provê o serviço definido pela interface.



Uma classe pode implementar (*realize*) várias interfaces.



# Ornamentos e Extensibilidade

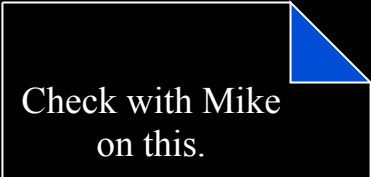
Um ornamento é algo como uma nota que adiciona texto ou algum elemento gráfico ao modelo.

UML oferece vários mecanismos que podem ser utilizados para estender a linguagem “oficial”.

- estereótipos
- valores rotulados (*tagged values*)
- restrições

# Notas

É um símbolo gráfico contendo texto ou figuras oferecendo algum comentário ou detalhes sobre um elemento de um modelo.



Check with Mike  
on this.



See [http://www.  
softdocwiz.com](http://www.softdocwiz.com)



See encrypt.ps



# *Estereótipos*

É uma extensão do vocabulário de UML que permite a criação de um tipo básico novo que é específico ao problema que está sendo resolvido.



# *Estereótipos Padrão em UML*

cerca de 50, incluindo:

- *become* (indica uma dependência na qual um objeto se torna outro)
- *enumeration* (especifica um tipo enumerado incluindo seus possíveis valores)
- *utility* (uma classe na qual todos os valores e atributos pertencem à classe (e não às suas instâncias))

# *Valores Rotulados*

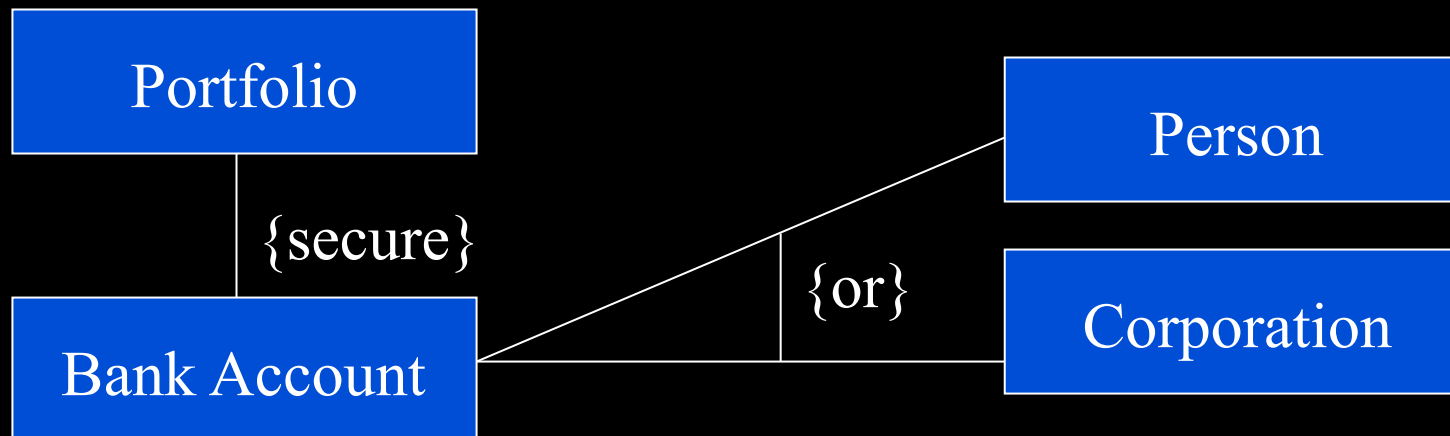
Permite a especificação de propriedades de elementos de um modelo:

GL Account  
{persistent}

TargetTracker  
{release = 2.0}

# Restrições

Especifica uma condição que deve ser satisfeita pelo sistema.



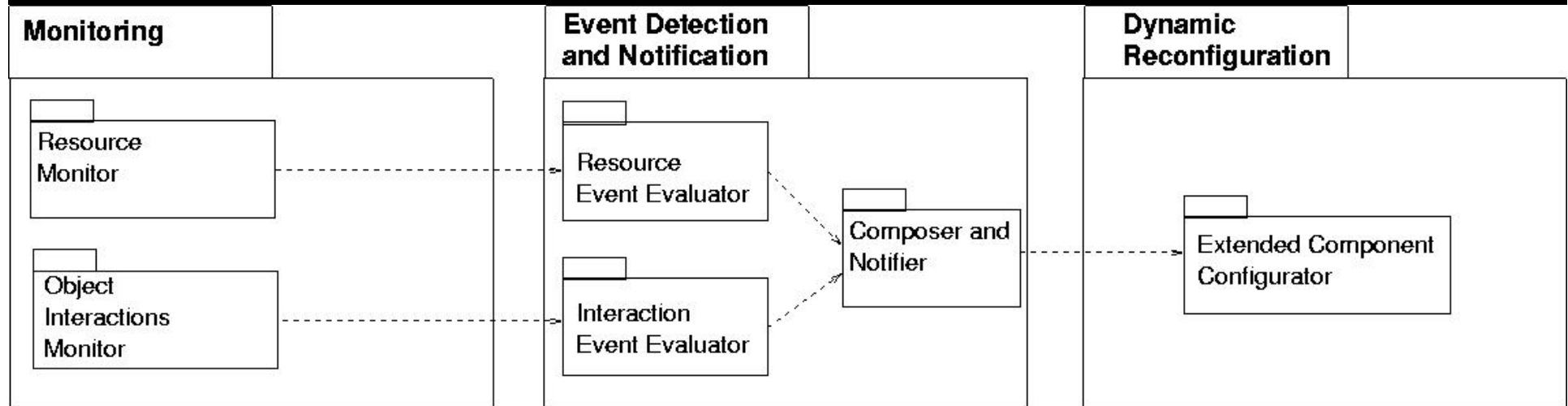
# *Pacotes*



- Um mecanismo para organizar elementos de um modelo (classes, diagramas, etc. ) em grupos.
- Cada elemento de um modelo pertence a um único pacote. O seu nome dentro do pacote deve ser único.

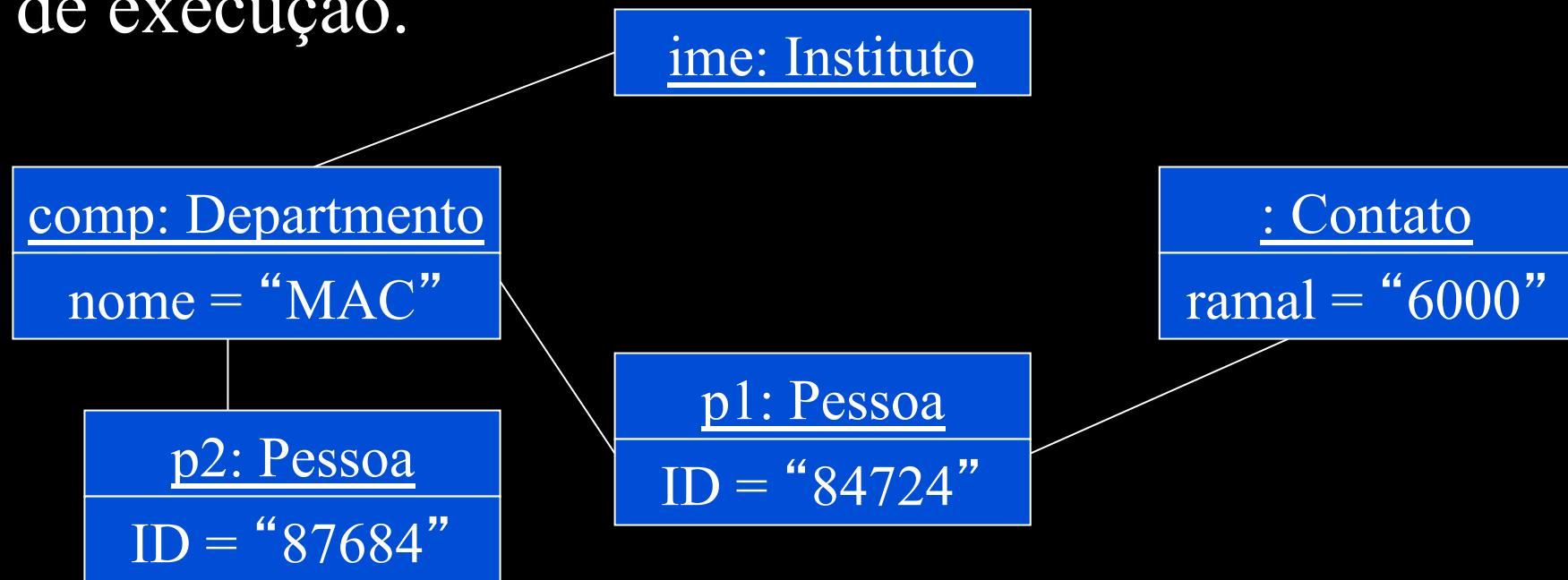
# *Um Diagrama de Pacotes*

- Arcabouço para construção de sistemas distribuídos adaptativos (de Francisco Silva<sup>2</sup>).



# Diagrama de Objetos

Mostra um conjunto de objetos e seus relacionamentos em um certo instante em tempo de execução.



# *Componente*

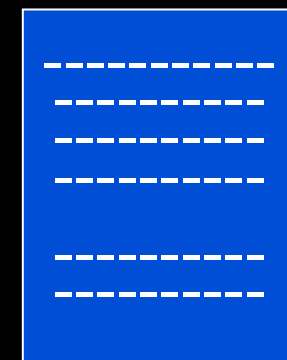
É uma parte de um sistema que pode ser substituída e que oferece uma implementação de um conjunto de interfaces.

Exemplos práticos:

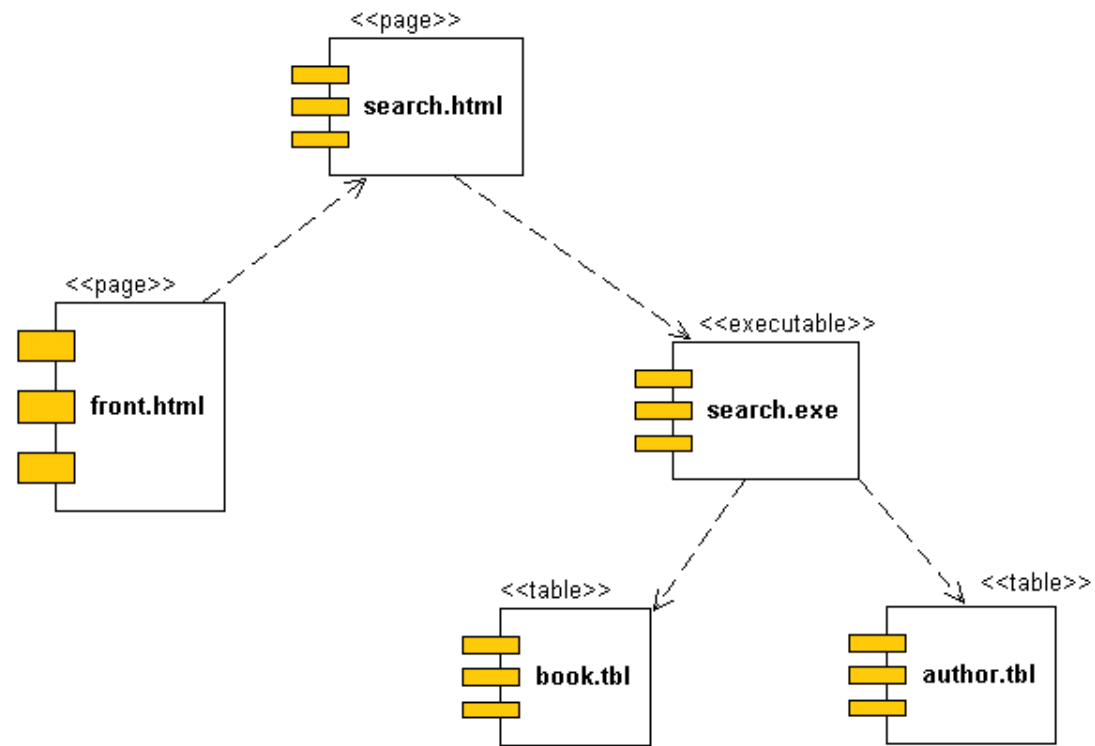
- Biblioteca de carga dinâmica (DLL)
- Componente CORBA
- Enterprise Java Bean (EJB)



# *Notação para Componentes*



signal.cpp



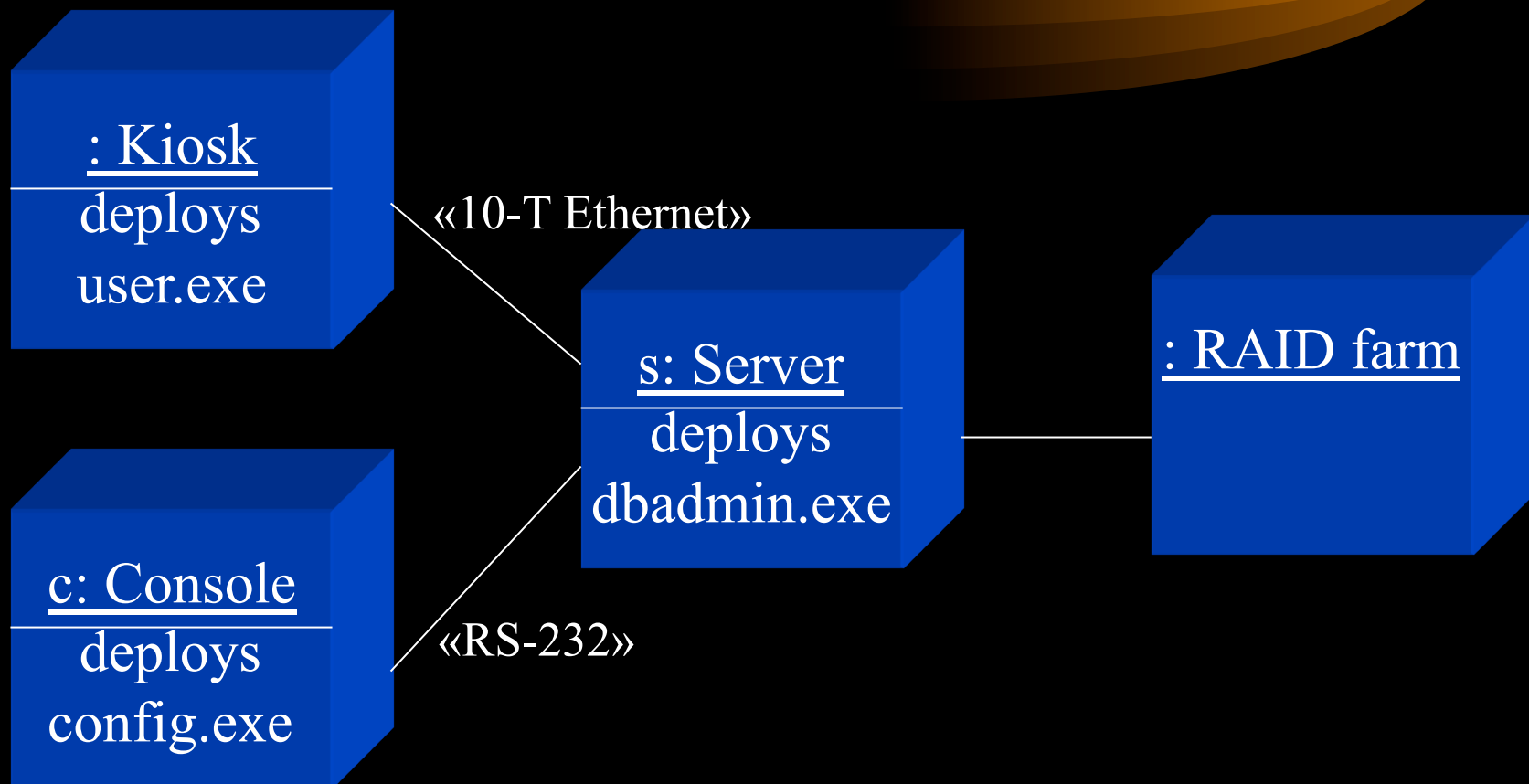
# *Nó*



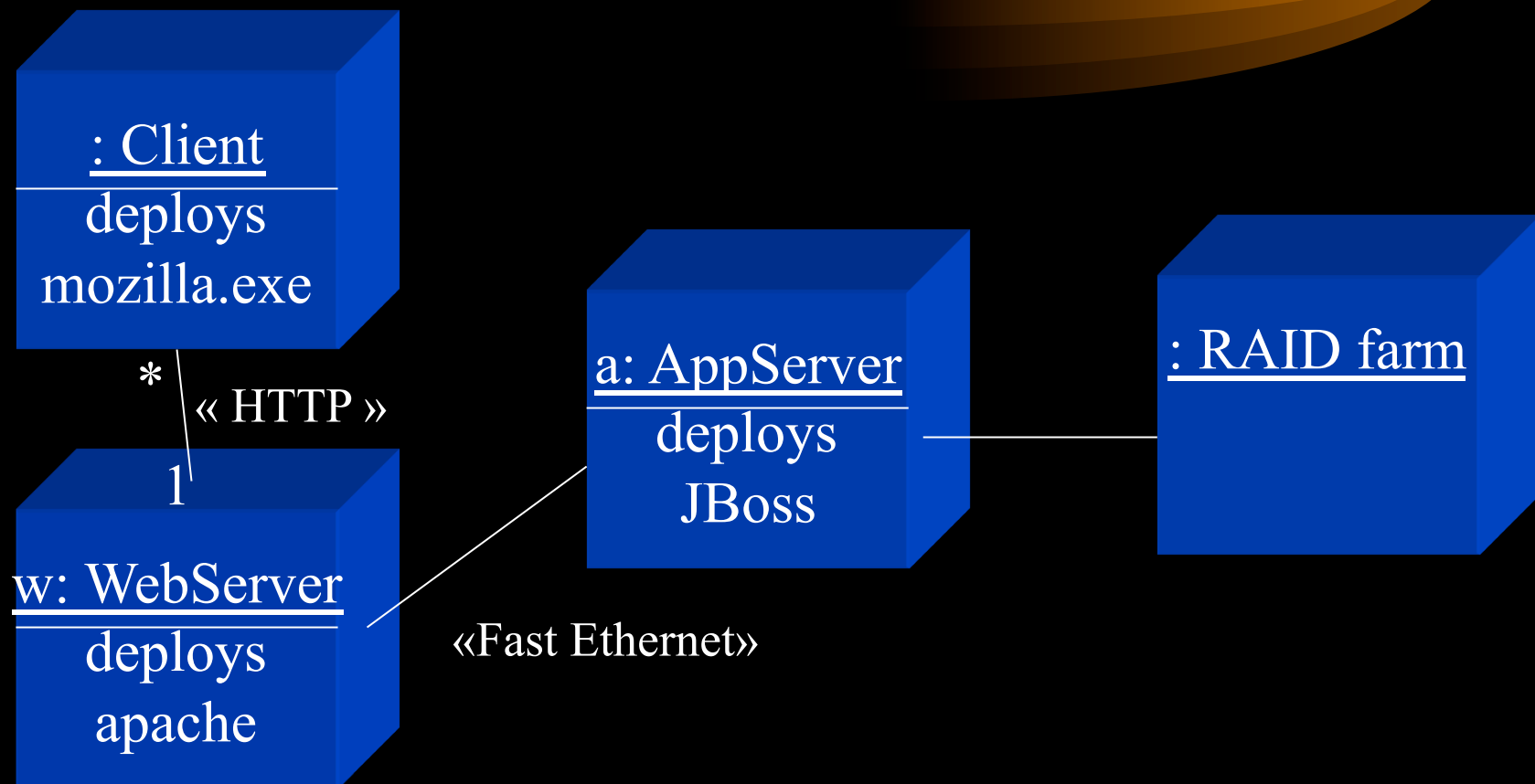
Representa um elemento físico capaz de oferecer recursos computacionais.

Em geral, possui pelo menos memória e processador.

# Diagrama de Implantação



# Diagrama de Implantação



## *Exercício em pares*

Escreva um diagrama de classes para modelar o seguinte sistema:

- Uma pessoa pode ser homem ou mulher. Além disso, ela tem um pai, uma mãe e pode ter 0 ou mais filhos. Pais, mães e filhos são também pessoas.
- Homens têm nome, CPF e time de futebol.
- Mulheres têm nome, CPF e cantor preferido.