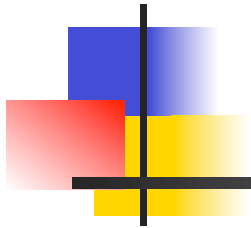
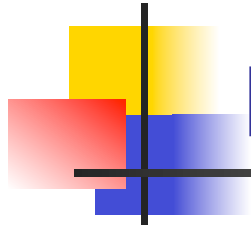


Métodos de Construção de Software: Orientação a Objetos



Graduação em Informática
5181-Análise de Sistemas de Software
2012

Profª Itana Gimenes (imsgimenes@uem.br)
Profª Thelma Elita Colanzi (thelma@din.uem.br)



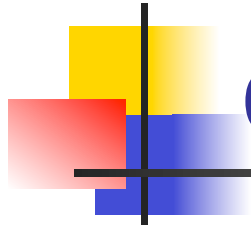
Problemas com Modelagem Funcional

- Aproximação de conceitos com o mundo real
 - vivemos num mundo de objetos
- Dados e processos separados
- Information Hiding (Ocultação de Informações)
- Tipos abstratos de dados
- Reutilização de software



Introdução à Orientação a Objetos (O-O)

- Iniciou com as linguagens de programação, 1980s.
Ex. Simula, Smalltalk, Eifel, C++, Java.
- Aplicada à análise e projeto, 1990s
- Grandes apelos
 - Possibilidade de construir software a partir de componentes existentes ao invés de sempre começar do zero.
 - Possibilidade de evoluir entre os estágios de desenvolvimento mantendo consistência entre os modelos e conceitos.

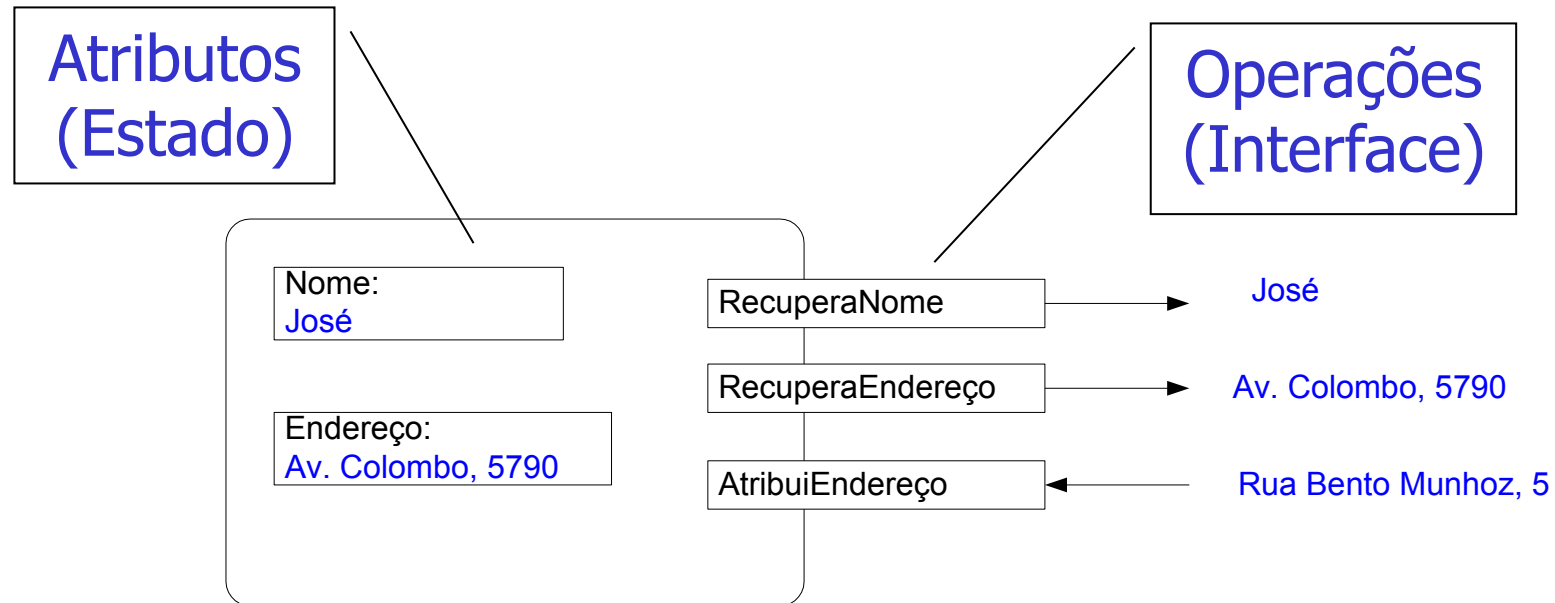


Conceitos de O-O

- **Objeto** é algo similar à uma instância de uma entidade como visto na análise estruturada, pois é uma unidade da qual queremos representar informações no sistema.
- Objetos possuem **atributos**
- Um conjunto de atributos forma o **estado** do objeto
- Objetos possuem **operações (serviços ou métodos)** que manipulam o estado do objeto.
- As operações associadas com um objeto são chamadas de **interface** pois constituem o único meio de manipular o estado do objeto.



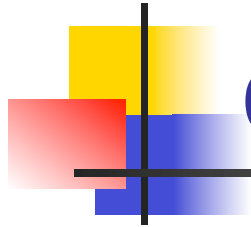
Ilustração de Objeto





Observações Importantes

- **Encapsulamento (ocultação de informação)**
 - Os dados associados a um objeto não estão disponíveis diretamente para os usuários do objeto.
 - A única maneira de utilizar os dados é através das operações visíveis na interface.
 - A implementação das operações não são visíveis ao usuário.
- **Independência de dados** – implementação das operações podem ser alteradas sem afetar os usuários dos objetos. A interface continua a mesma, apenas as ações internas são modificadas.



Comunicação entre objetos

- **Passagem de mensagens**
 - Chamada de uma operação sobre um objeto.
- Um sistema orientado a objetos consiste de vários objetos que se comunicam entre si.

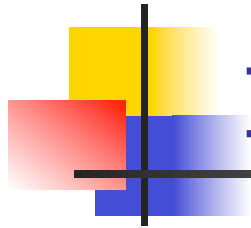
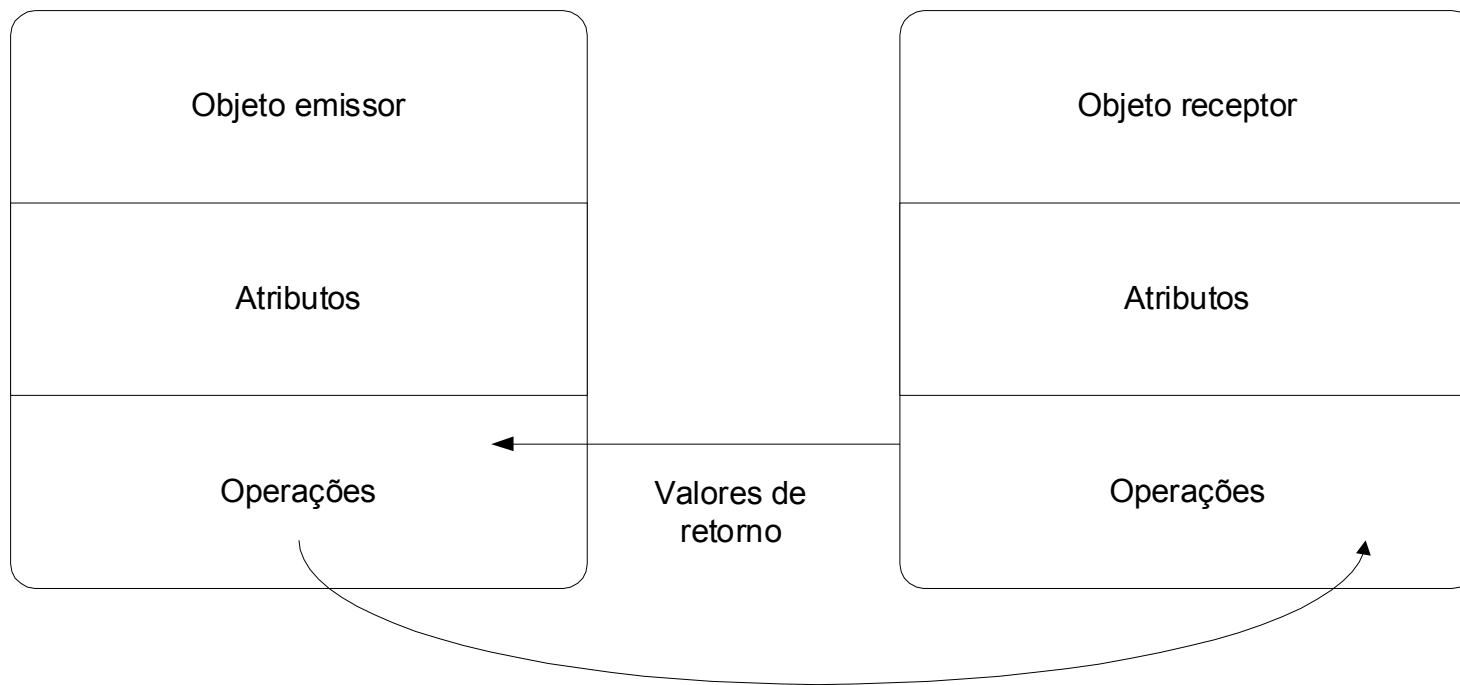


Ilustração de Comunicação

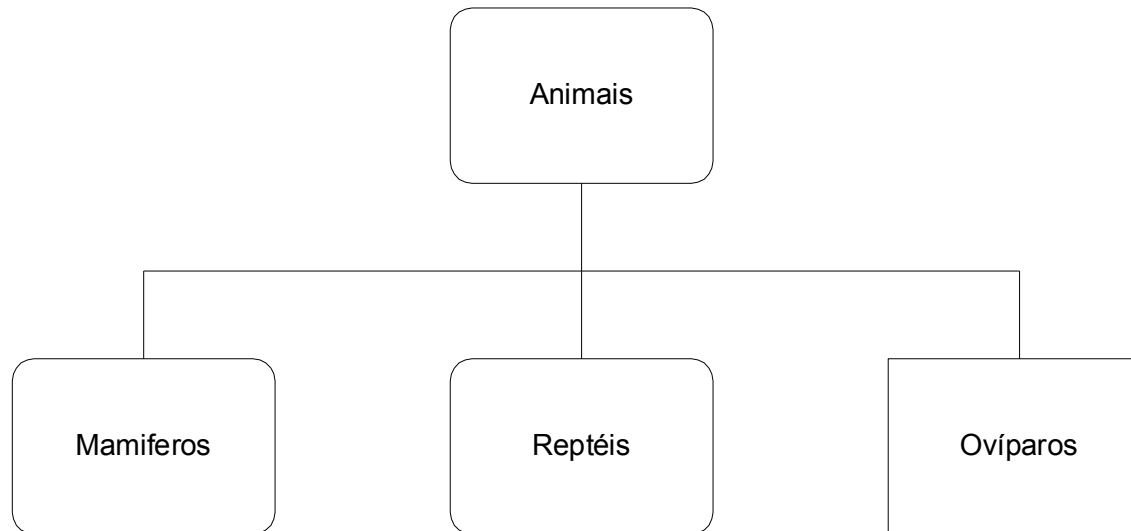


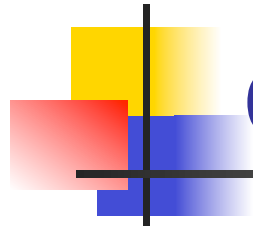
Mensagem [destino, operação, parametros]



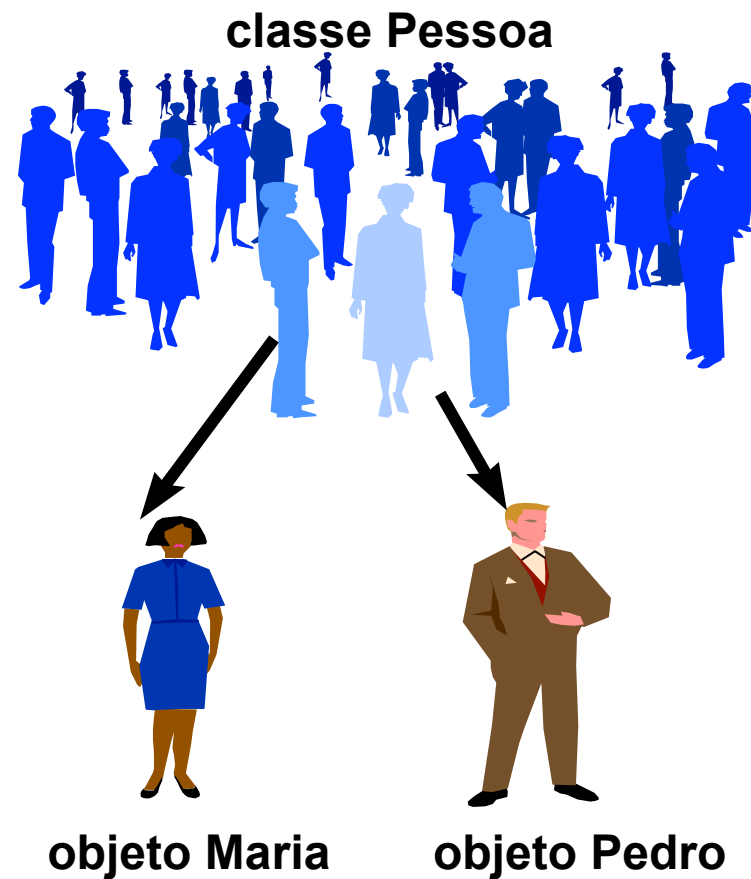
Classes

- Objetos que tem características comuns (atributos e operações) são agrupados em categorias chamadas de **Classes**.



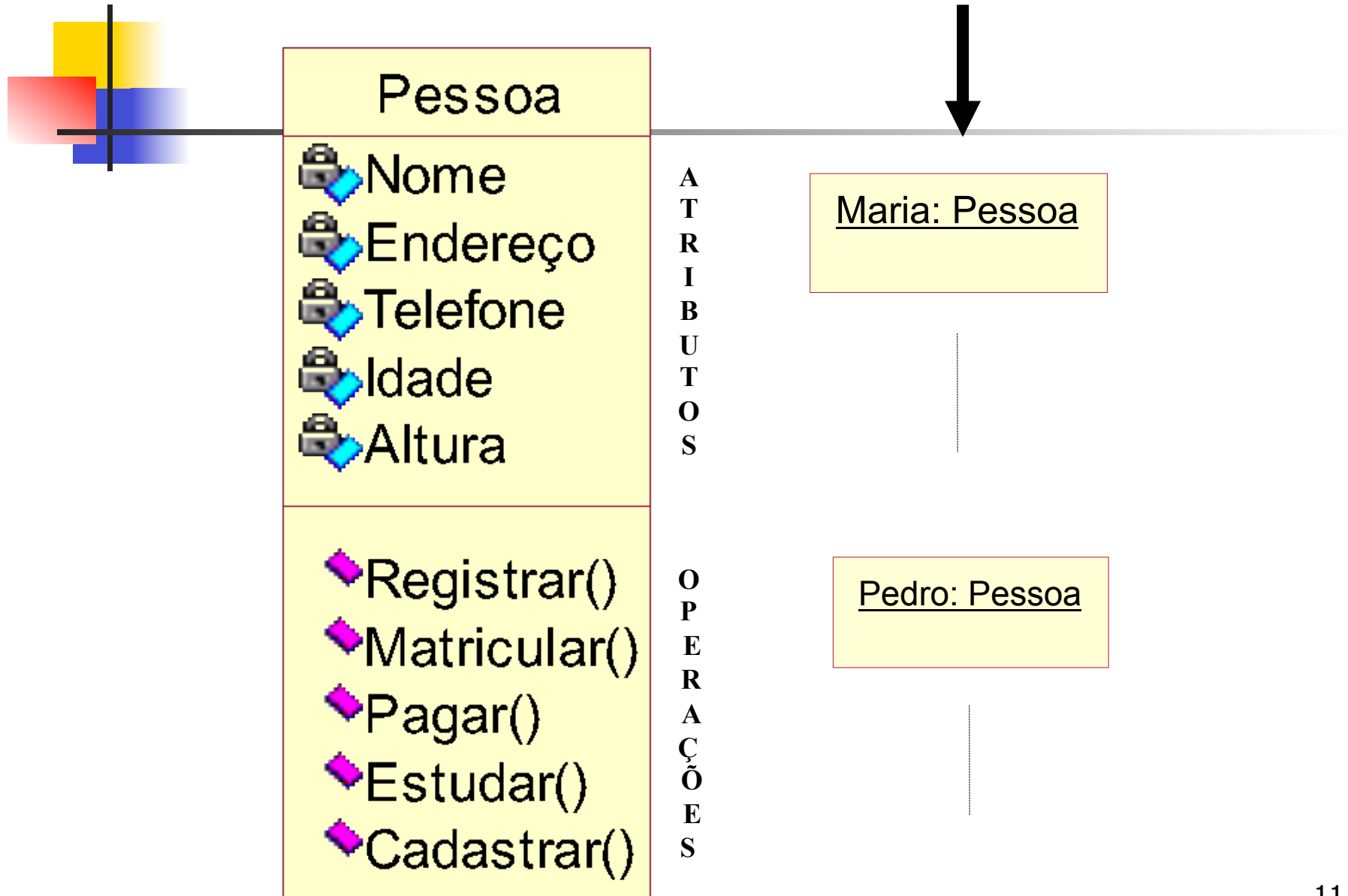


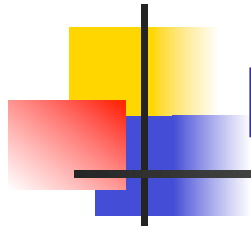
Classe e Objeto



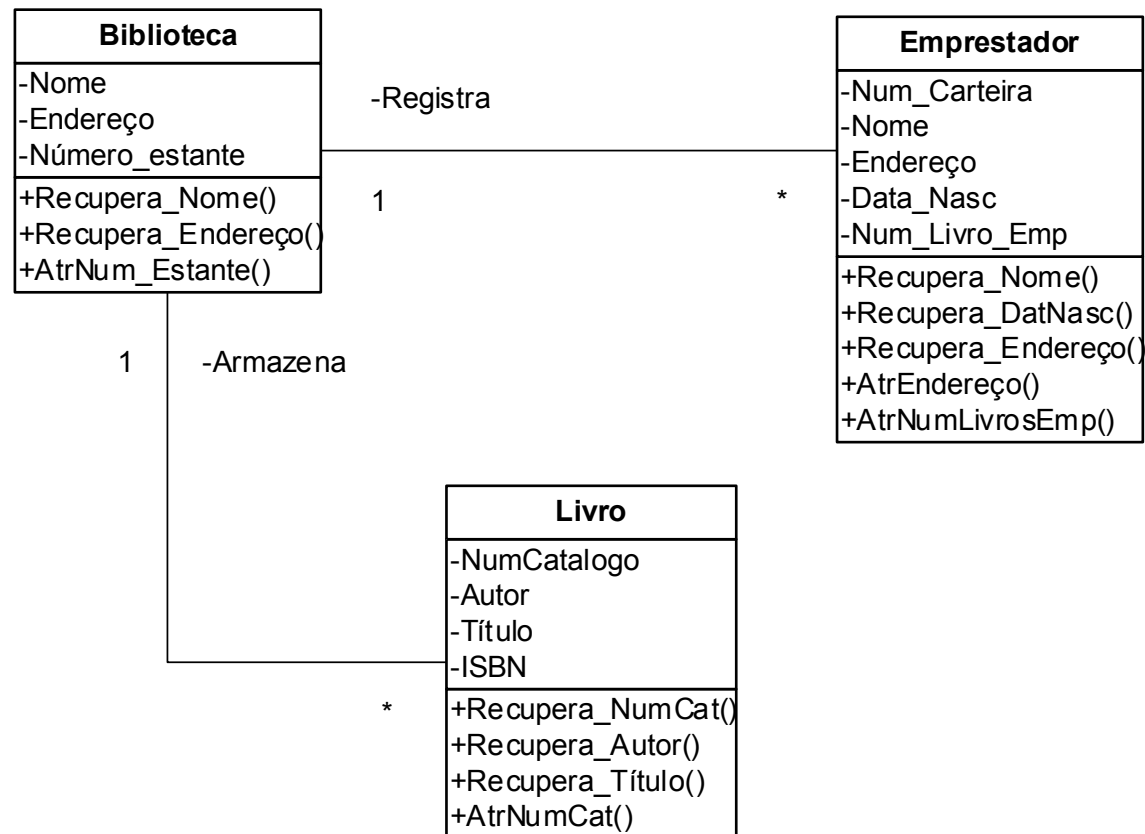
Classe

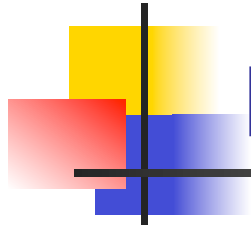
Objetos





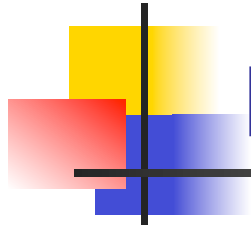
Relacionamento entre classes





Reutilização

- A abordagem O-O facilita reutilização pois objetos encapsulam dados e operações.
- Exemplo: a classe livro pode ser reutilizada em um sistema de venda de livros.

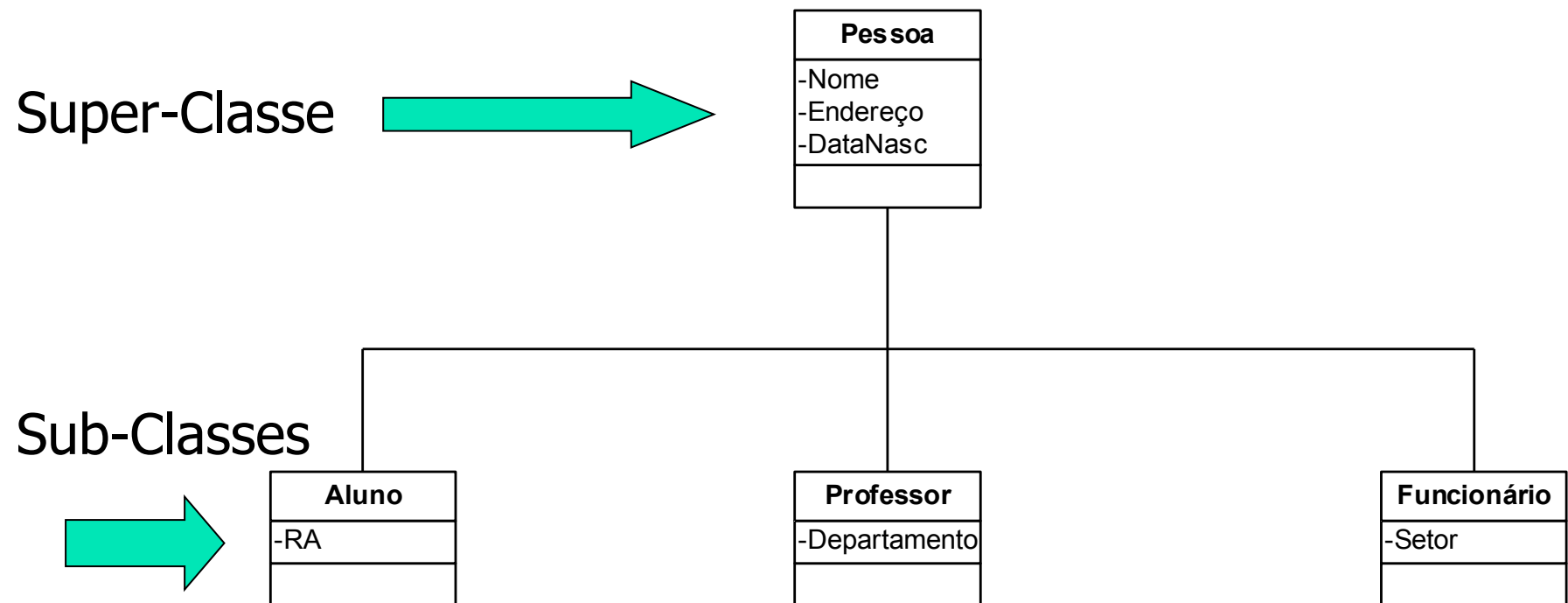


Exercício

- Imagine um sistema para controlar vendas em um supermercado.
 - Quais seriam as classes?
 - Quais os potenciais atributos?
 - Quais os relacionamentos entre as classes?

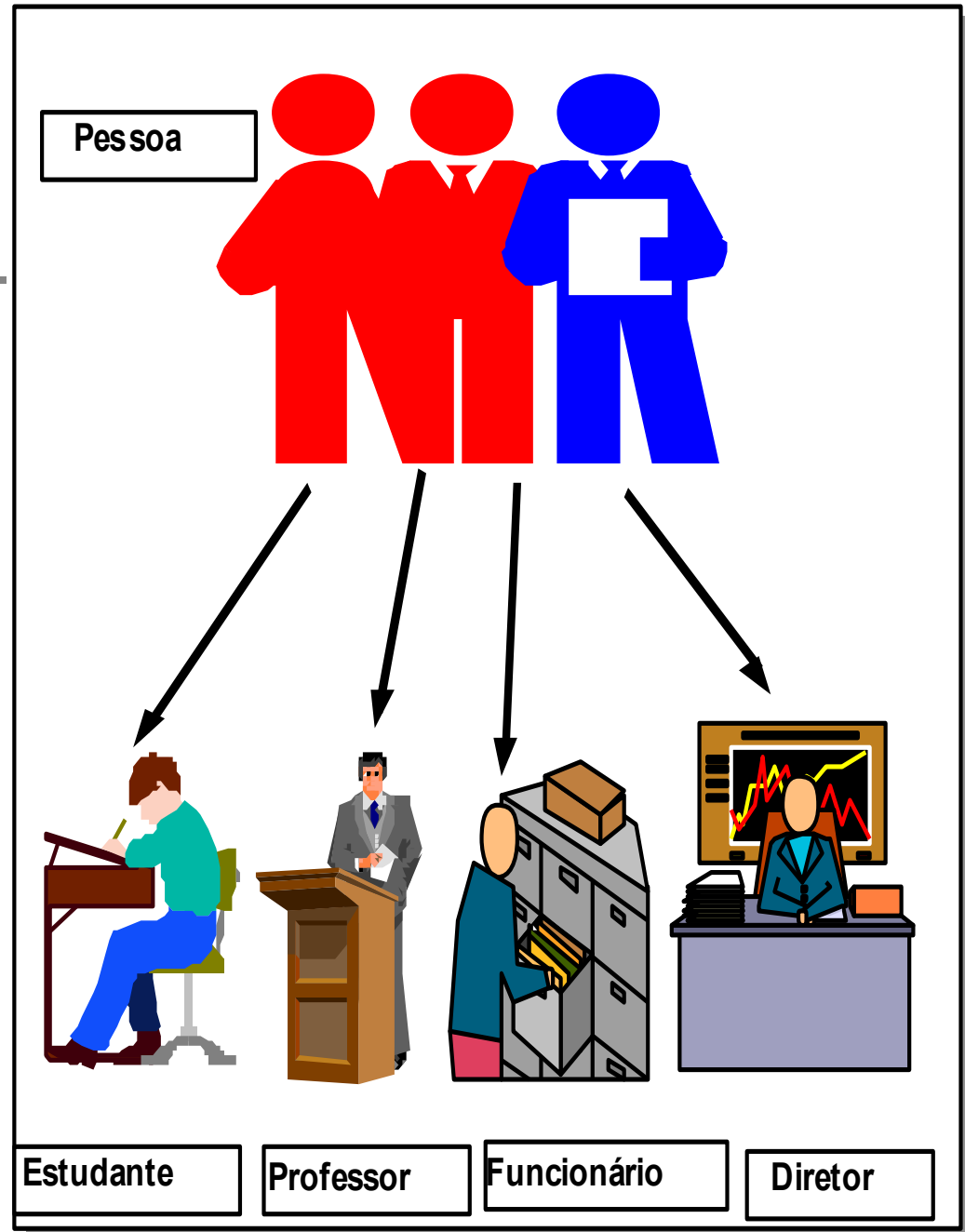
Herança

A herança pode reduzir substancialmente as repetições nos projetos e programas e é uma das principais vantagens dos sistemas OO.



HERANÇA

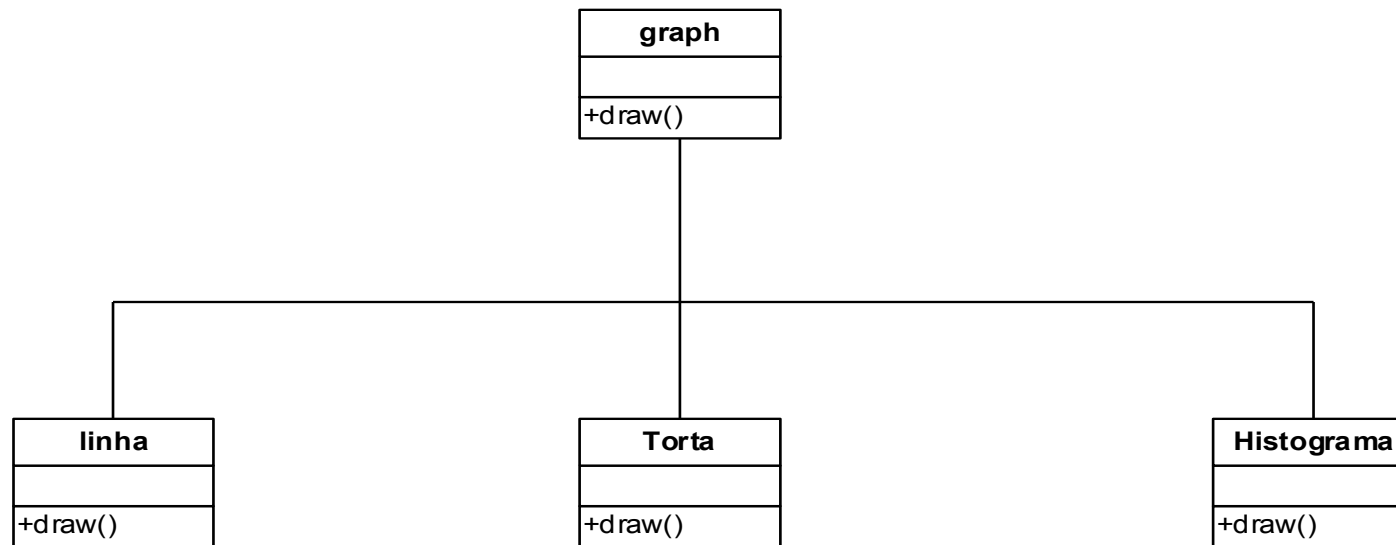
Classes com
atributos e
operações iguais
podem ser
agrupadas em
hierarquias





Polimorfismo

- Aplicação da mesma operação a diferentes tipos de objetos.
- Ex. writeln do Pascal que imprime qualquer tipo de parâmetro (string, inteiro, real).



A operação `draw()` é overloaded .



Exemplos de Código

Programa para mostrar formatos na tela

```
class shape {  
    point center;  
    color co1;  
    // ...  
public:  
    void move(point to) { center=to; draw();}  
    point where() {return center;}  
    virtual void draw();  
    virtual void rotate(int);  
    // ...  
};
```



Exemplos de Código

Programa para mostrar formatos na tela

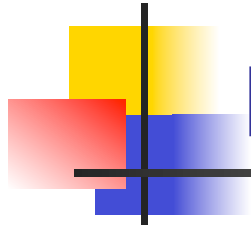
```
class circle: public shape {  
    int radius;  
Public:  
    void draw();  
    void rotate(int i);  
    // ...  
};
```



Exemplos de Código

Programa para mostrar formatos na tela

- Criando ou declarando um círculo
`circle* umcírculo = new circle(); circle umcírculo;`
- Referência a umcírculo
`x = umcírculo->where()
umcírculo.where();`
- Vetor para girar todos os formatos de 45°
`for (int i = 0; i<no_of_shapes; i++)
shape_vec[i].rotate(45);`



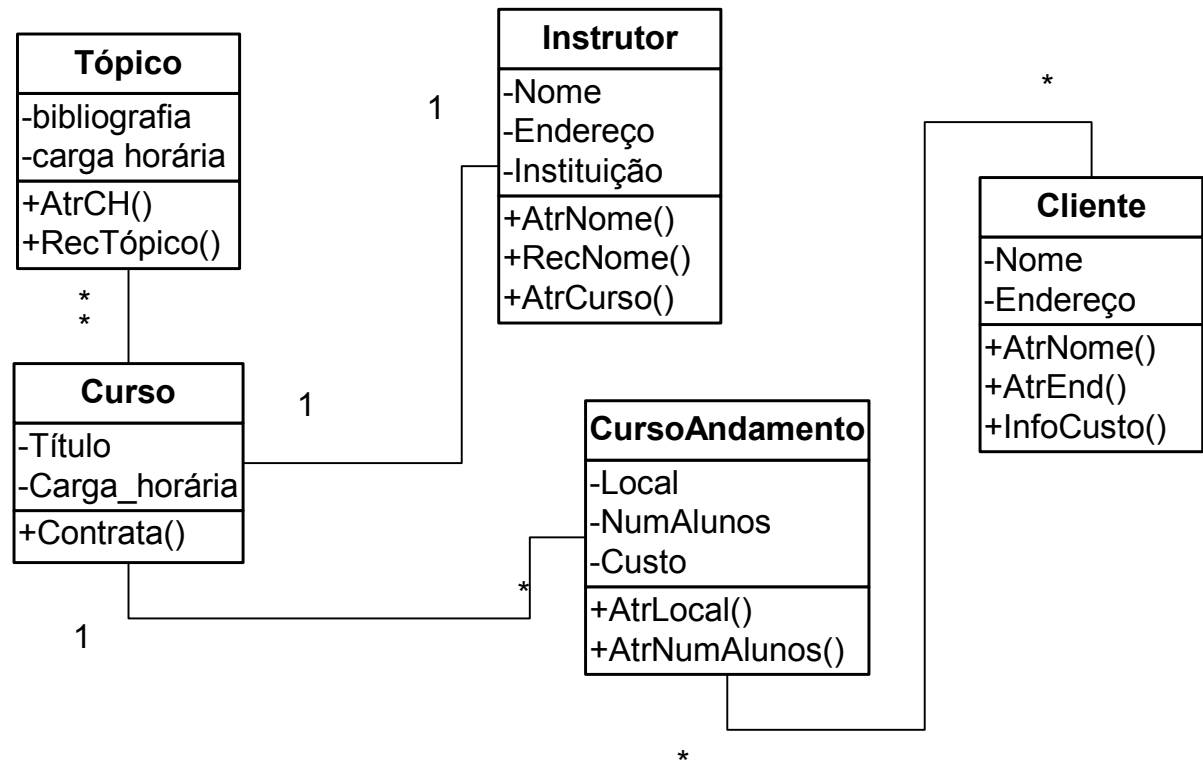
Exercício

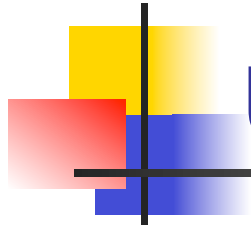
- Imagine um sistema para controlar cursos de treinamento em uma empresa
 - Quais seriam as classes?
 - Quais os potenciais atributos e operações entre as classes?
 - Quais os relacionamentos entre as classes?
 - Quais as potenciais mensagens entre as classes?



Possível Solução

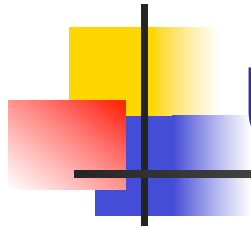
- Agendar cursos
- Preparar material
- Contratar instrutores
- Marcar exames
- Gerenciar cursos em andamento





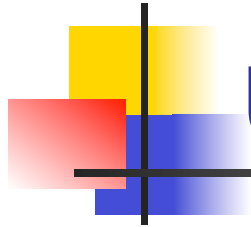
Unified Modeling Language (UML)

- Não é uma linguagem de programação.
- Linguagem de modelagem visual utilizada para especificar, visualizar, construir e documentar artefatos de software.
- UML é utilizada para entender, projetar, navegar, configurar, manter e controlar informações sobre um sistema.
- UML não supõe um processo de desenvolvimento padrão, esta pode ser utilizada em qualquer processo.
- Surgiu como padrão para consolidar a notação utilizada em vários métodos existentes inicialmente. Ex. OMT, Booch, Jacobson, Fusion.



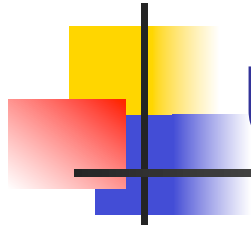
UML: Visões

- Uma visão é um subconjunto das construções de UML que representam um aspecto do sistema.
- Um ou mais diagramas são utilizados para fornecer uma notação visual para os conceitos associados a cada visão.
- As visões são agrupadas em áreas principais



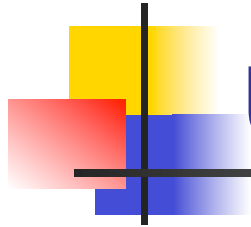
UML: Áreas Principais

- Estrutural –descreve as “coisas” de um sistema e seus relacionamentos.
- Dinâmica – descreve o comportamento do sistema no tempo.
- Física – descreve os recursos computacionais do sistema e a alocação de artefatos para estes recursos.
- Gerenciamento de modelo – descreve a organização dos modelos em unidades hierárquicas.



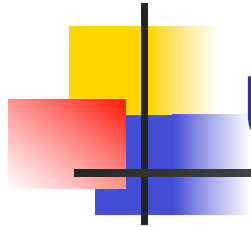
UML: Área Estrutural

- Visão estática – diagrama de classes
- Visão de projeto – diagrama de estrutura interna, diagrama de componentes, diagrama de colaboração
- Visão de casos de uso – diagrama de casos de uso



UML: Área Dinâmica

- Visão da máquina de estados – diagrama de estados.
- Visão de atividades – diagrama de atividades.
- Visão de interação – diagrama de sequência e diagrama de comunicação.



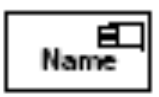
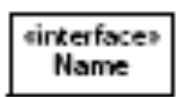




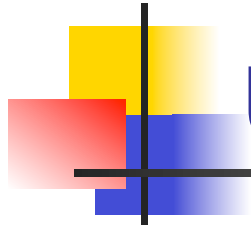
UML: Áreas Física e de Gerenc. Modelo

- Área Física – diagrama de instalação (distribuição)
- Área de Gerenciamento de Modelo
 - Pacotes especiais constituem unidades organizacionais que incluem subsistemas e modelos
 - Diagrama de pacotes
 - Esta visão cruza todas as outras.

Classificadores

Modelam conceitos -tipos de elementos utilizados –para cada classificador existe uma representação diagramática.

<i>Classifier</i>	<i>Function</i>	<i>Notation</i>
actor	An outside user of a system	
collaboration	A contextual relationship among objects playing roles	
component	A modular part of a system with well-defined interfaces	
interface	A named set of operations that characterize behavior	
node	A computational resource	
use case	A specification of the behavior of an entity in its interaction with outside agents	



UML: Visão estática

- Representação de uma classe
- Relacionamentos
 - Associação
 - Agregação e composição
 - Generalização
 - Herança e Herança Múltipla
- Exemplo de uma bilheteria: diagrama de classes
- Diagrama de objetos

Classes e Relacionamentos

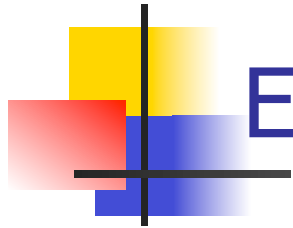
Subscription
series: String priceCategory: Category number: Integer
cost (): Money reserve (series: String, level: SeatLevel) cancel ()

class name

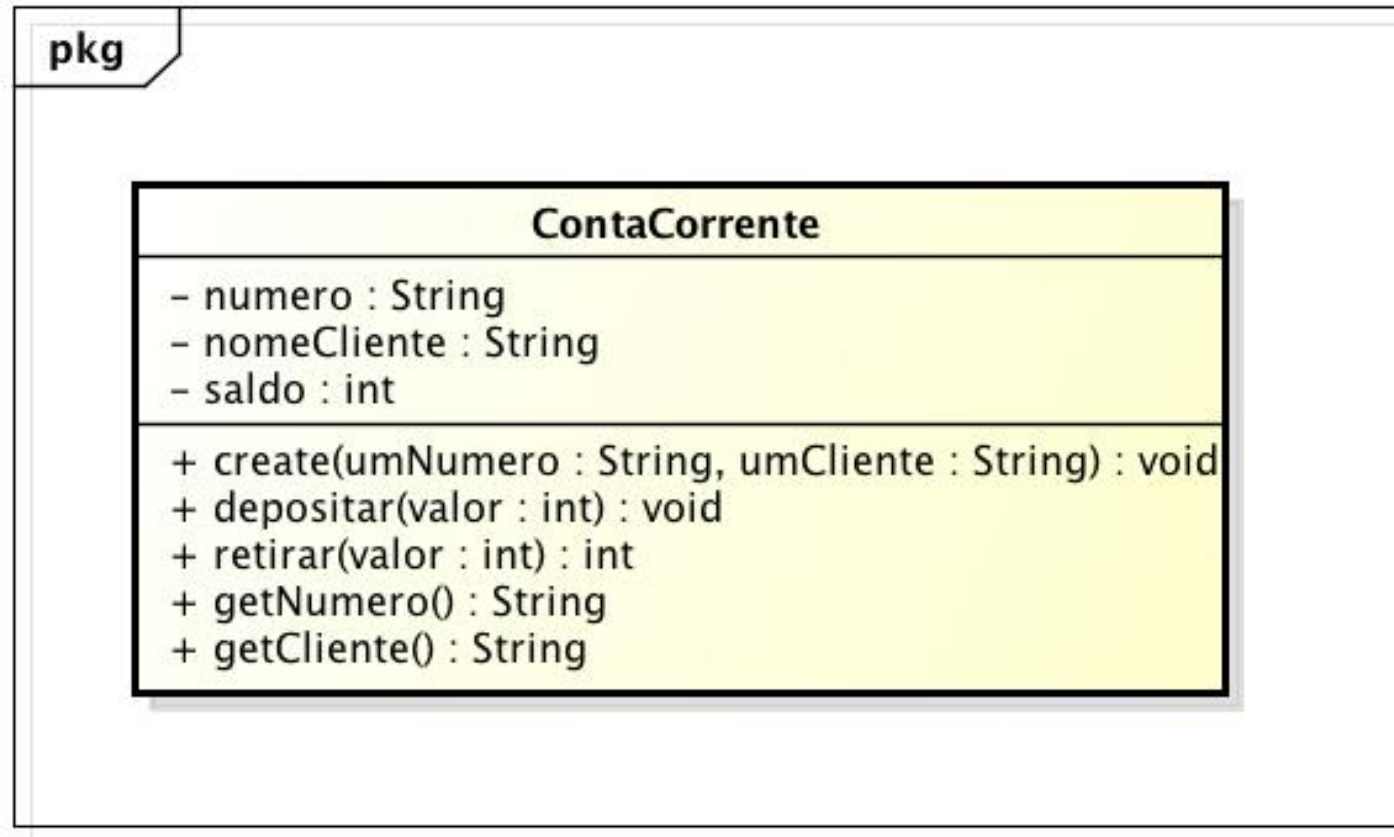
attributes

operations

<i>Relationship</i>	<i>Function</i>	<i>Notation</i>
association	A description of a connection among instances of classes	_____
dependency	A relationship between two model elements	- - - - ->
generalization	A relationship between a more specific and a more general description, used for inheritance and polymorphic type declarations	—>
realization	Relationship between a specification and its implementation	- - - - ->



Exemplo



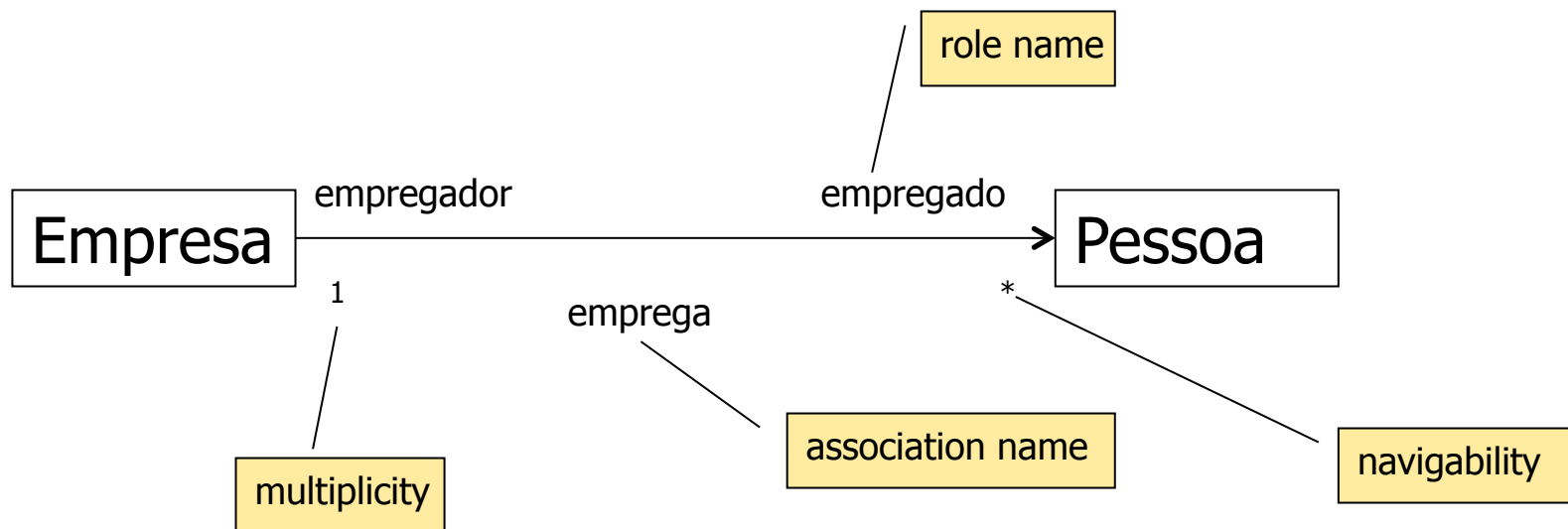


Visibilidade

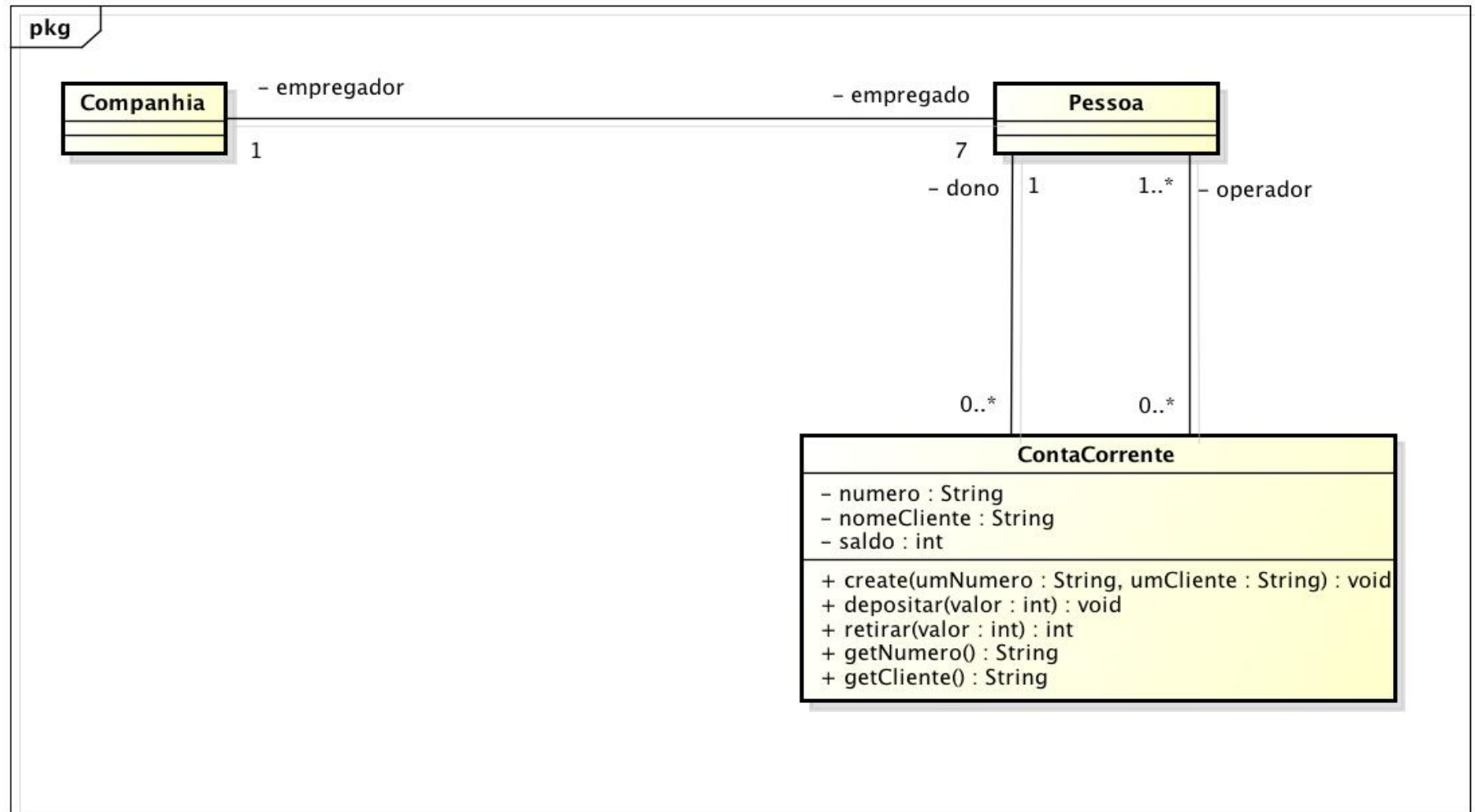
Decoração	Nome da visibilidade	Semântica
+	Pública	Qualquer elemento que pode acessar a classe pode acessar a operação
-	Privada	Apenas operações dentro de uma classe pode acessar a operação
#	Protegida	Apenas operações dentro da classe, ou dos filhos desta podem acessar a operação
~	Pacote	Qualquer elemento dentro de um mesmo pacote ou de um pacote aninhado pode acessar a operação

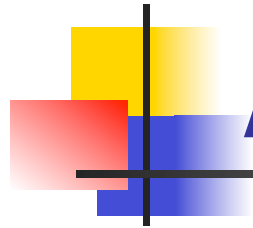


Associação

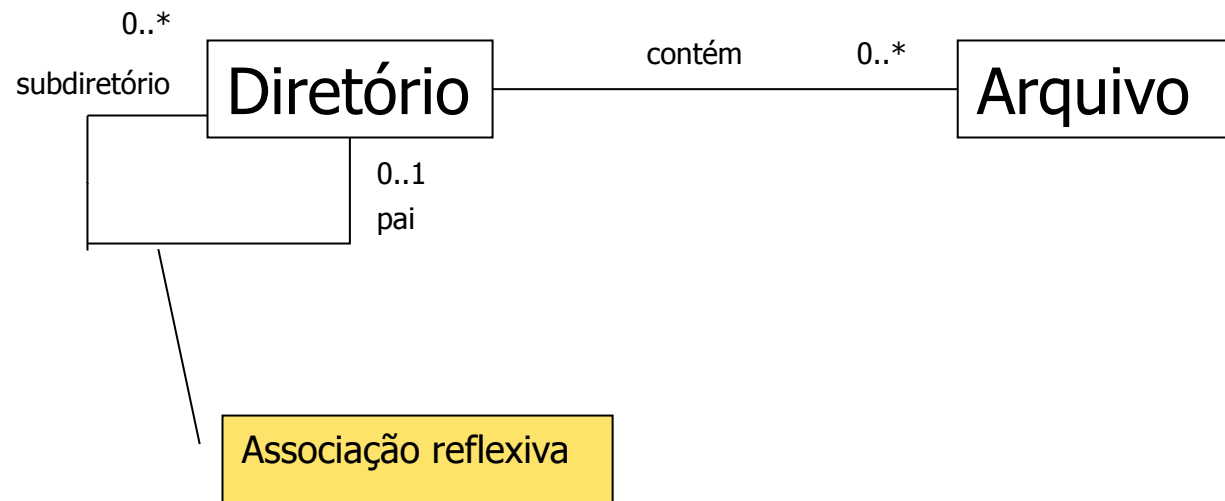


Exemplo





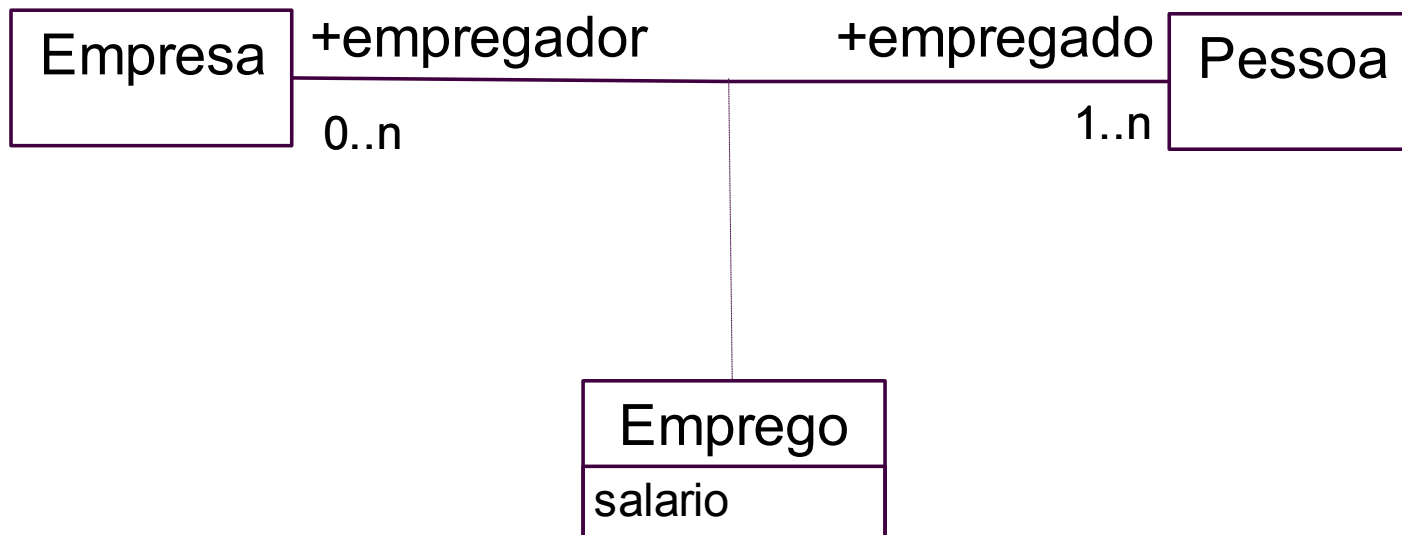
Associações reflexivas



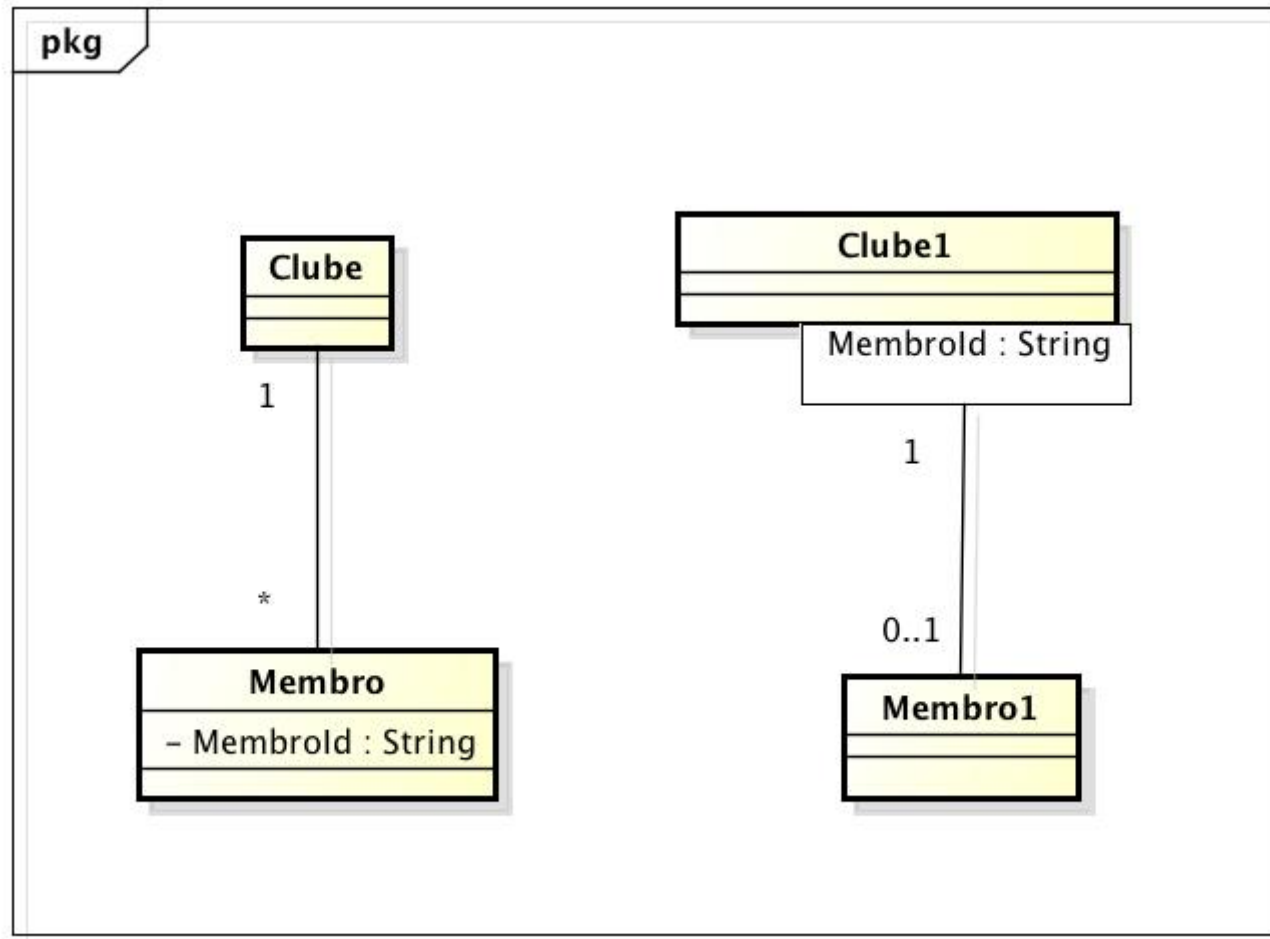


Associação: Classe Associativa

- É uma associação que também possui propriedades de classe (ou uma classe que tem propriedades de uma associação). É mostrada como uma classe, ligada por uma linha tracejada a uma associação.

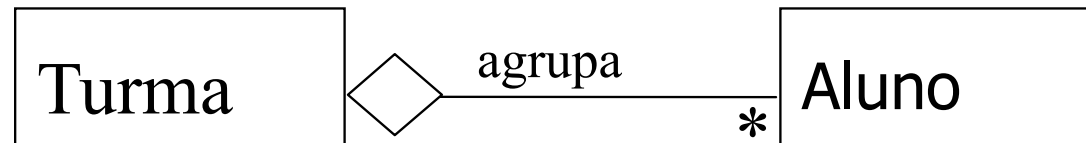


Qualifiers



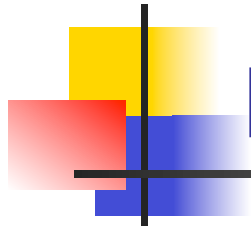
Agregação e Composição

- Agregação

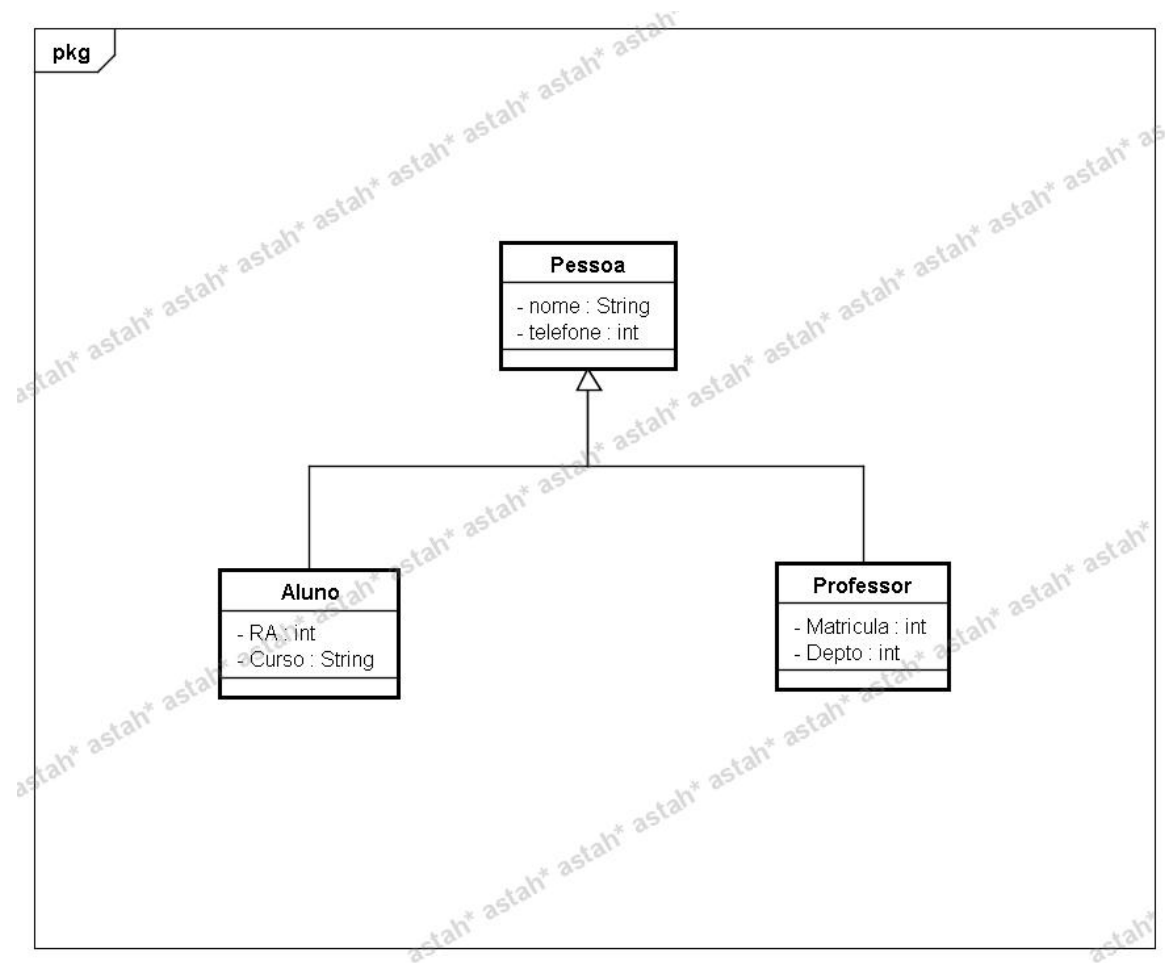


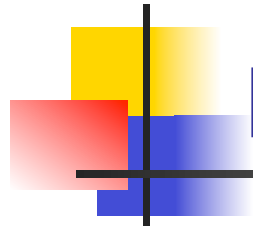
- Composição



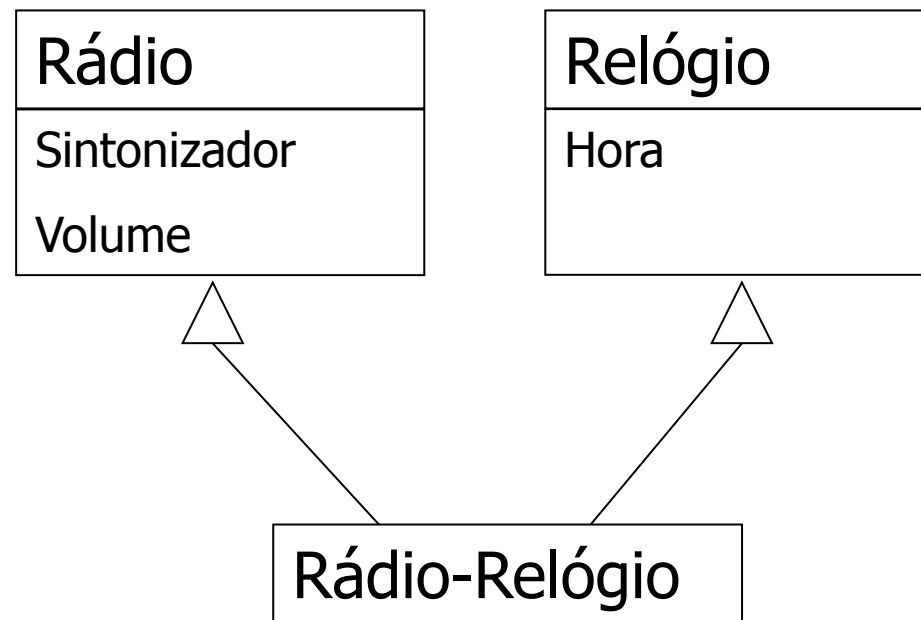


Herança





Herança Múltipla



Exemplo de uma bilheteria: diagrama de classes

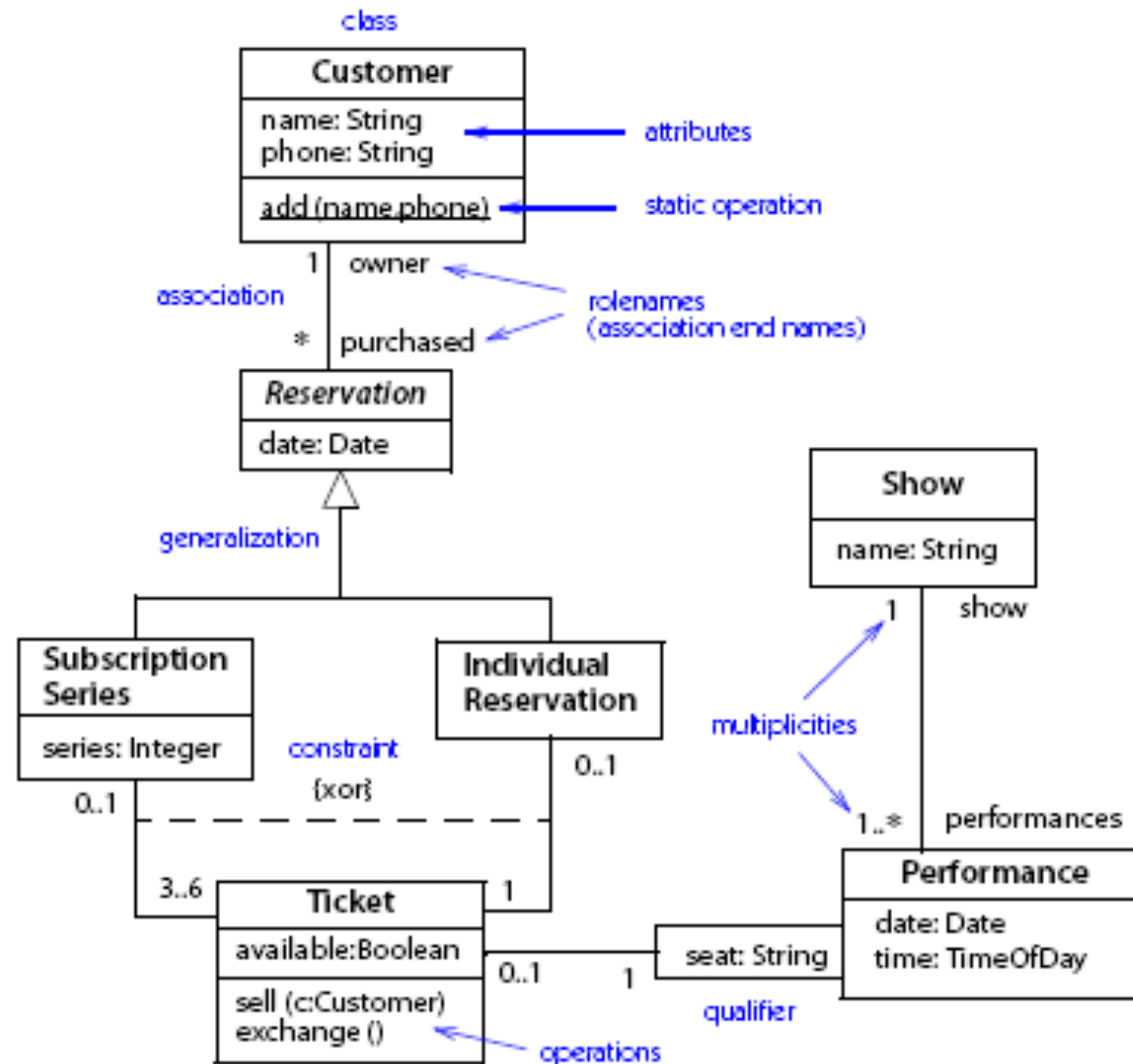
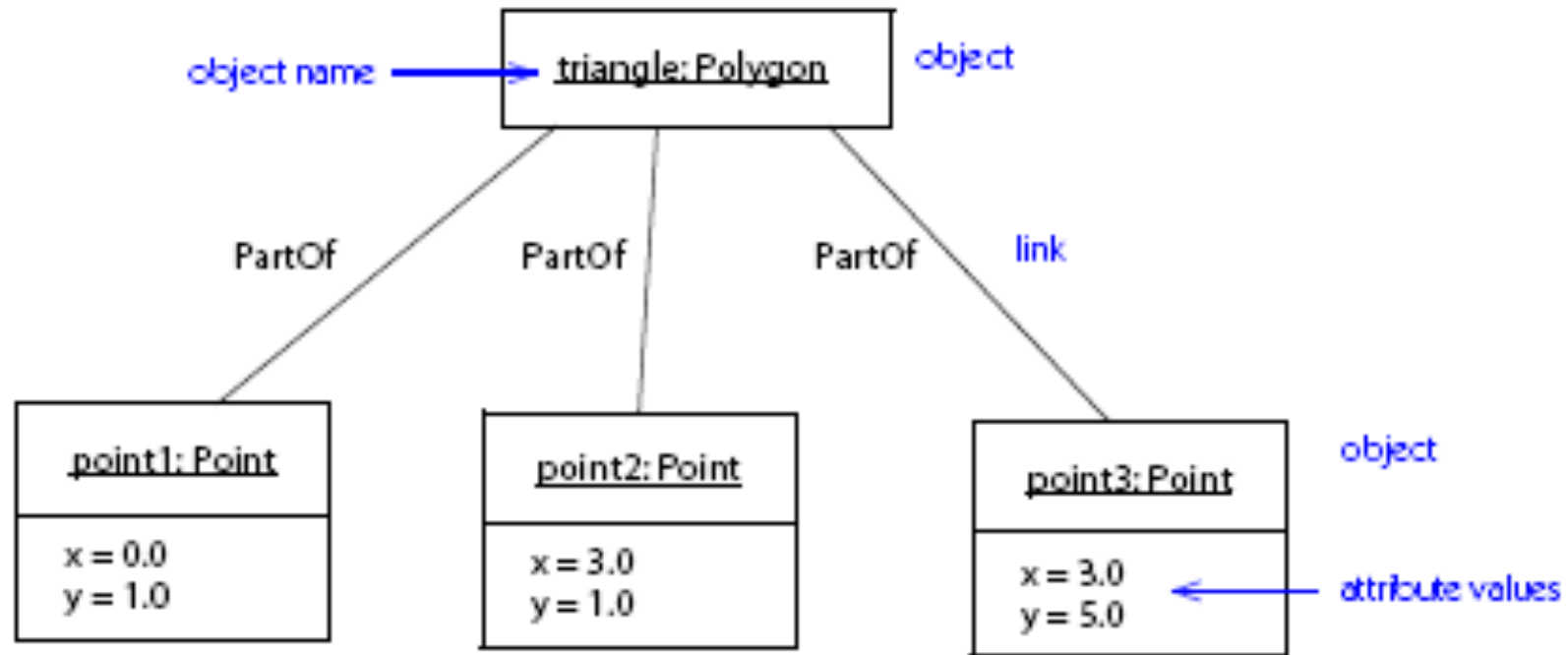
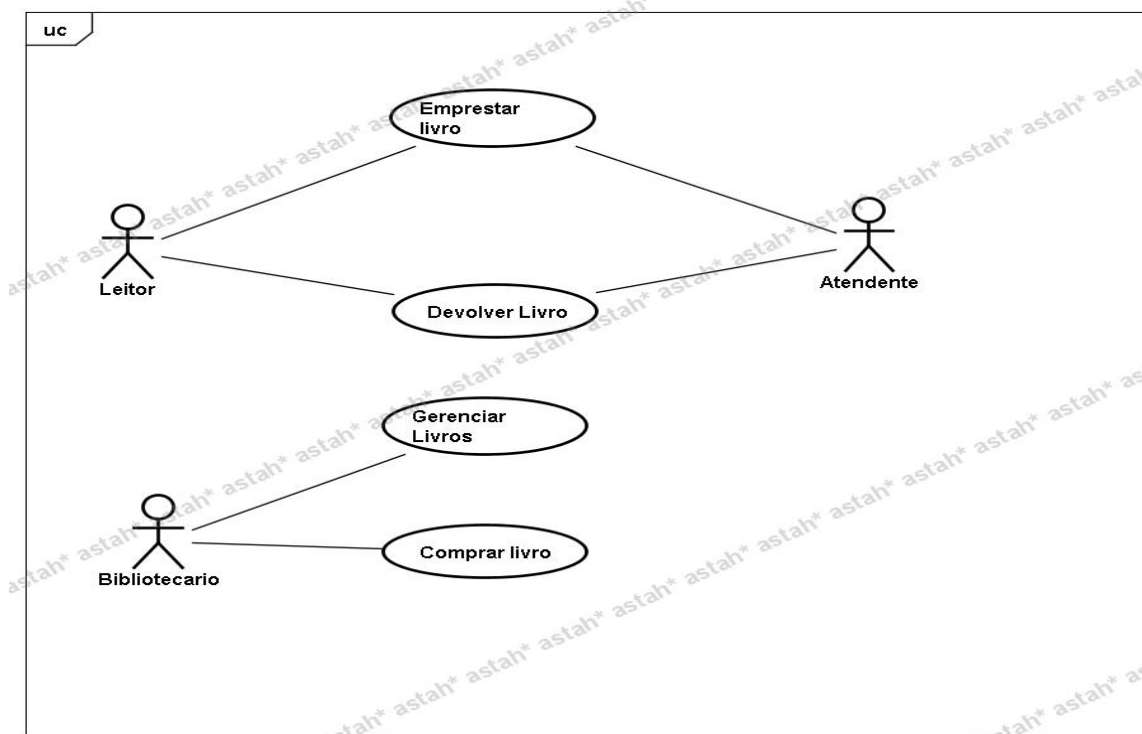


Diagrama de Objetos

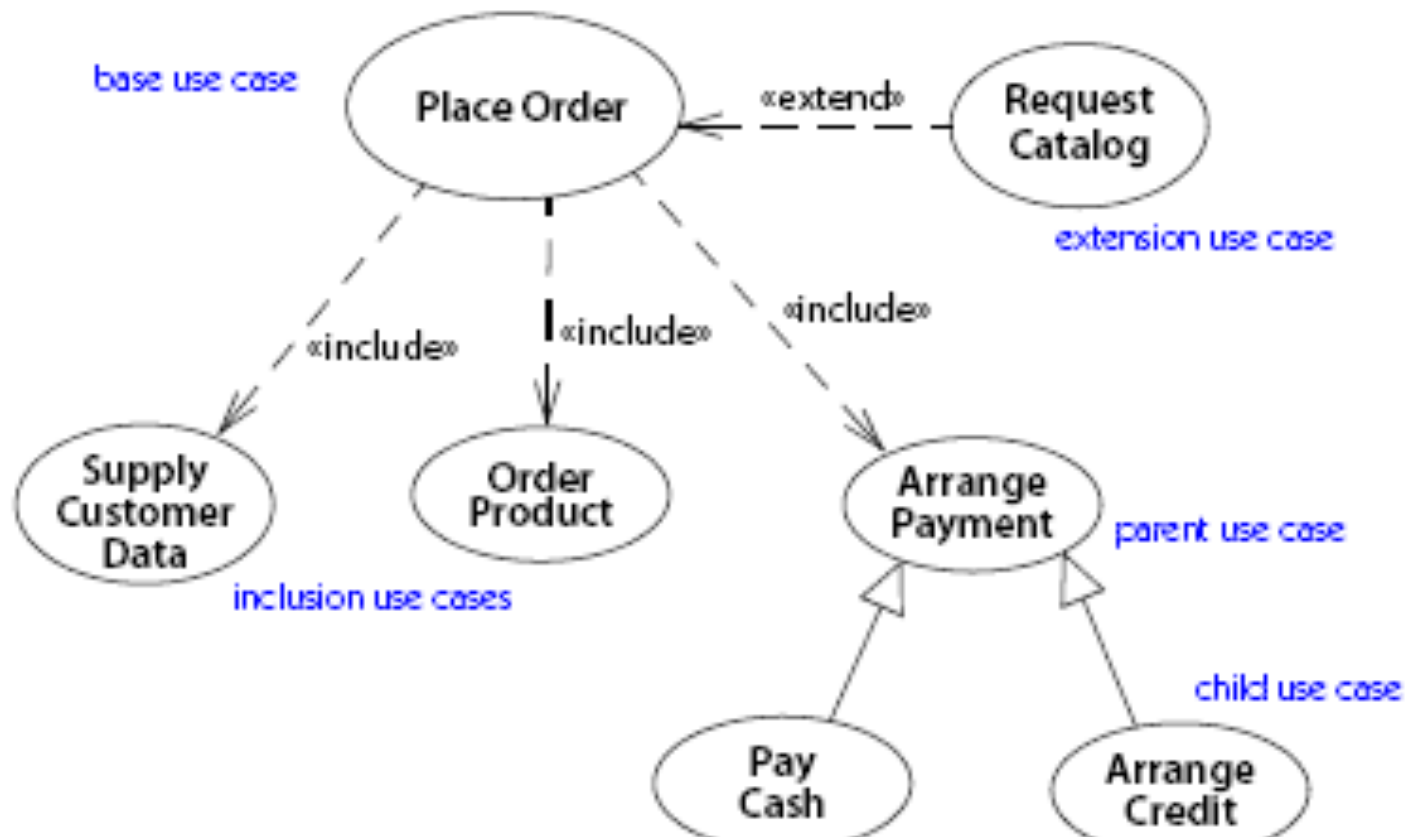


Visão de casos de uso

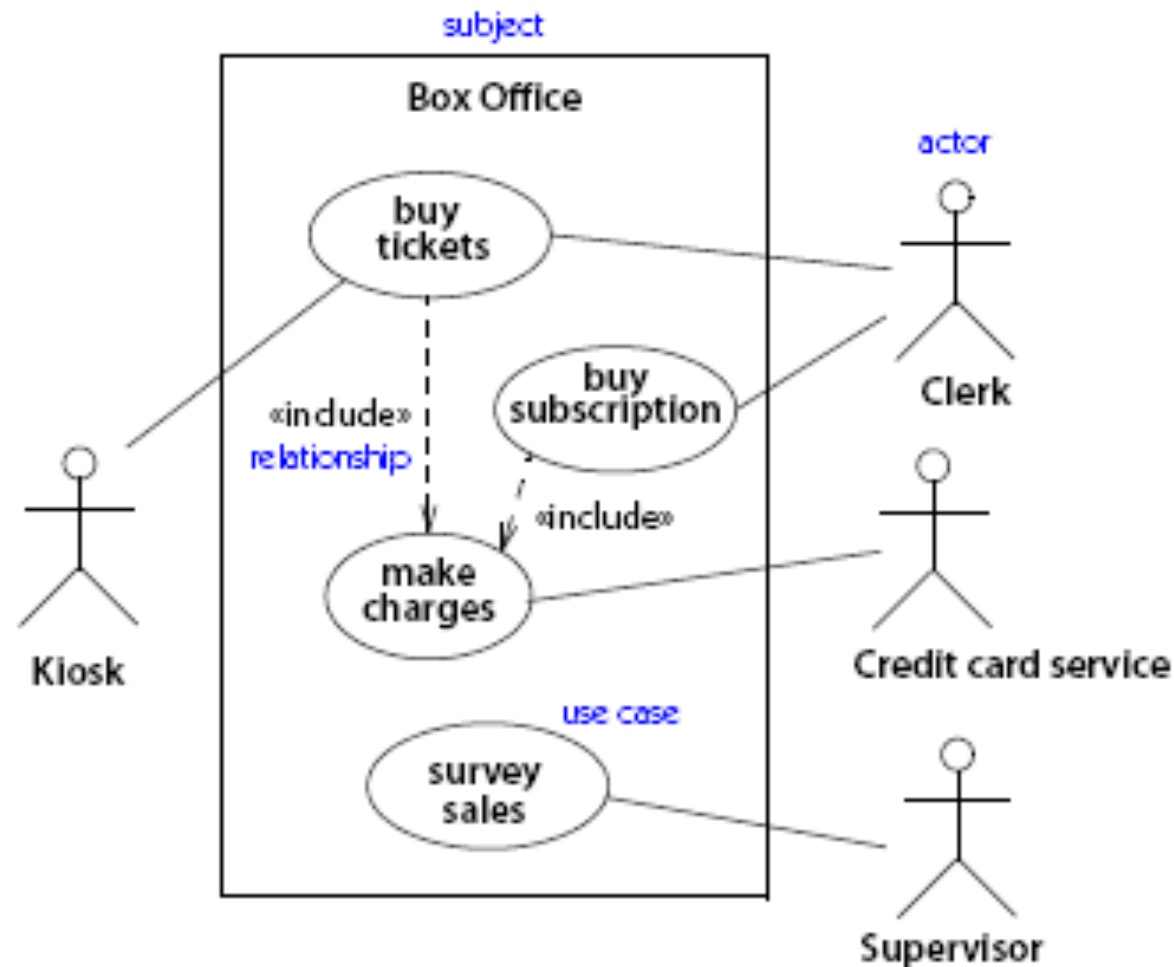
- Diagrama de casos de uso
- Tipos de relacionamentos entre casos de uso

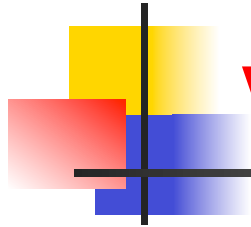


Relacionamientos entre casos de uso



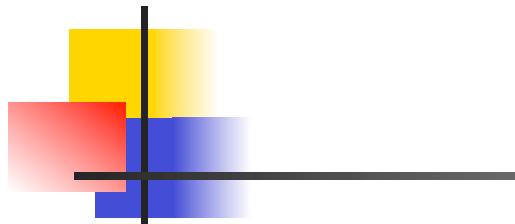
Exemplo de uma bilheteria: diagrama de casos de uso





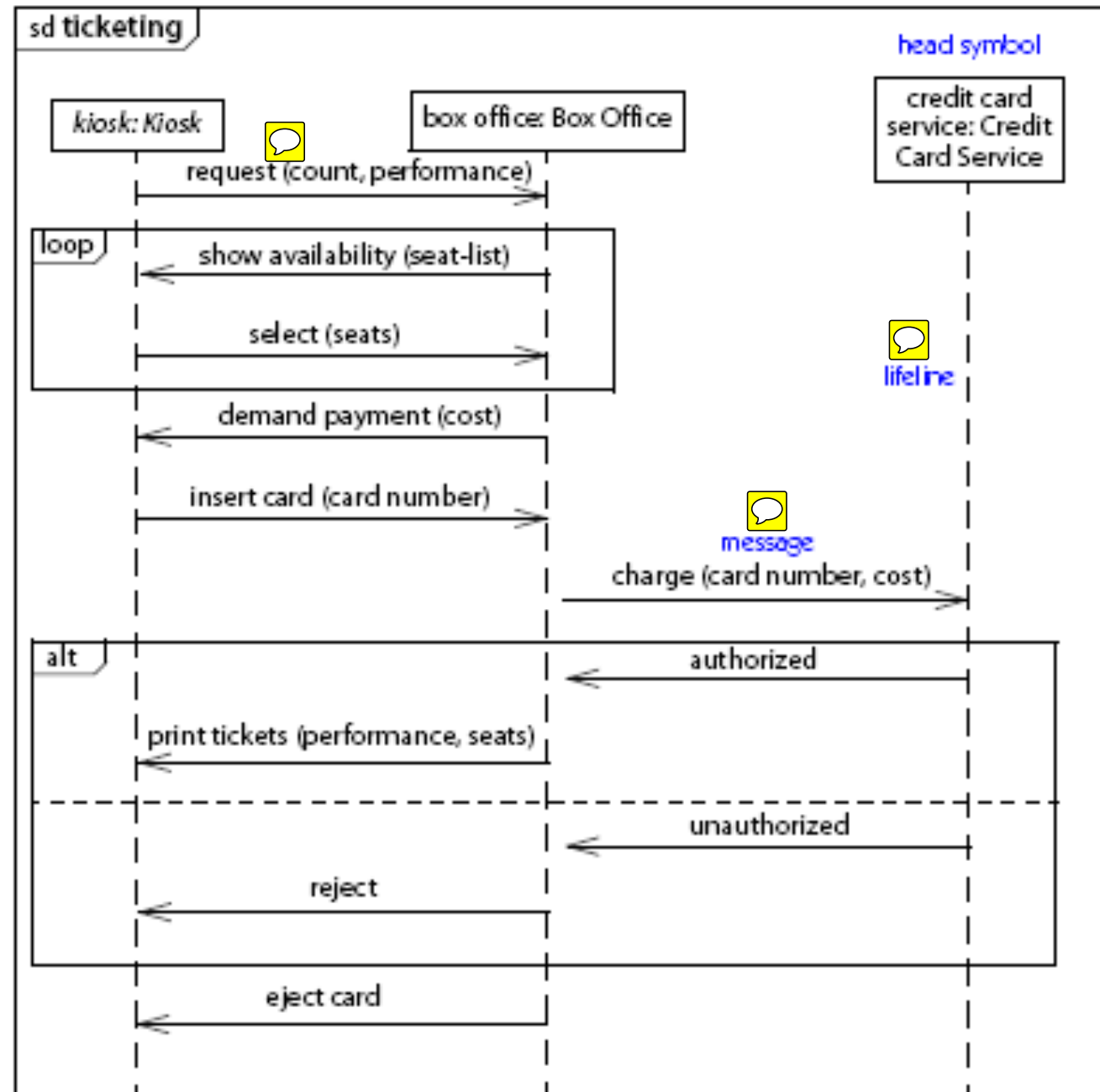
Visão de interação

- Diagrama de comunicação
- Diagrama de sequência



Exemplo de
uma
bilheteria
(compra de
tickets):

diagrama
de
seqüência



Exemplo de uma bilheteria: diagrama de comunicação

Exemplo referente ao recebimento de um pedido

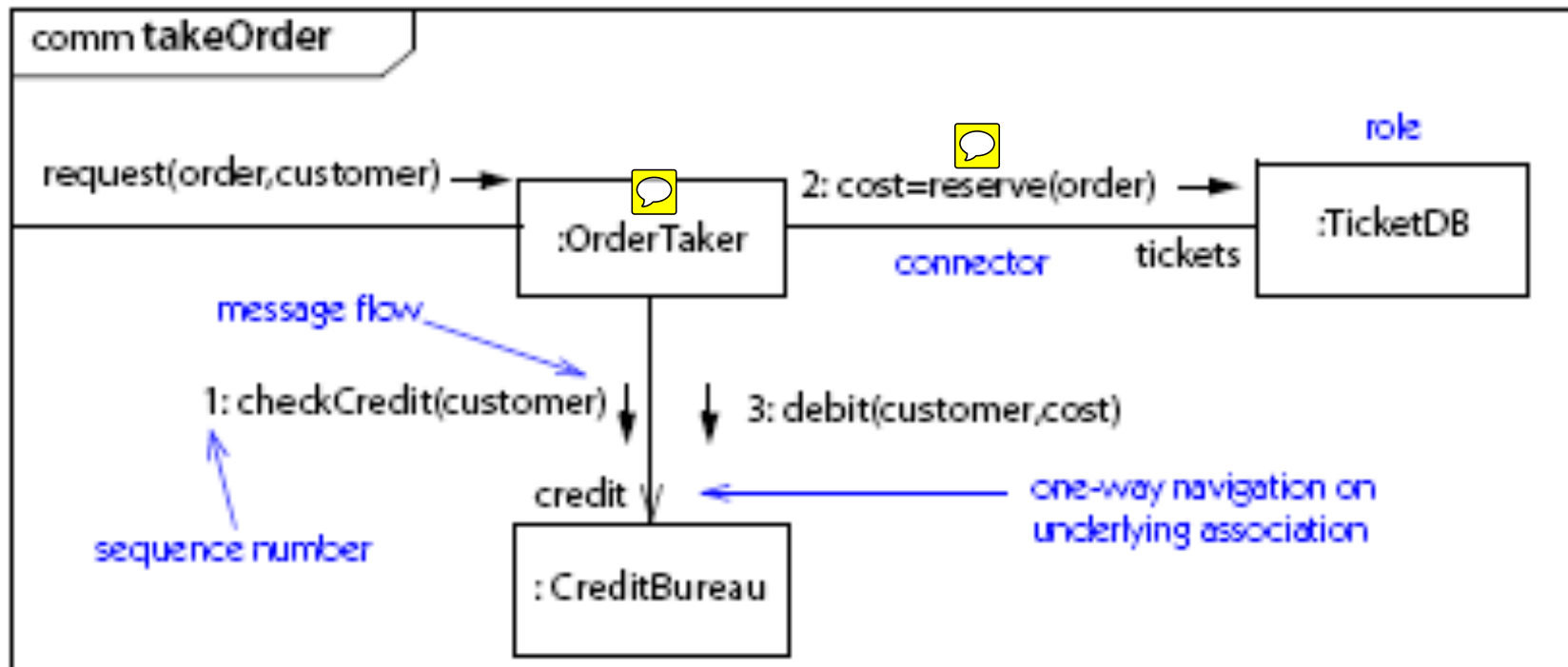
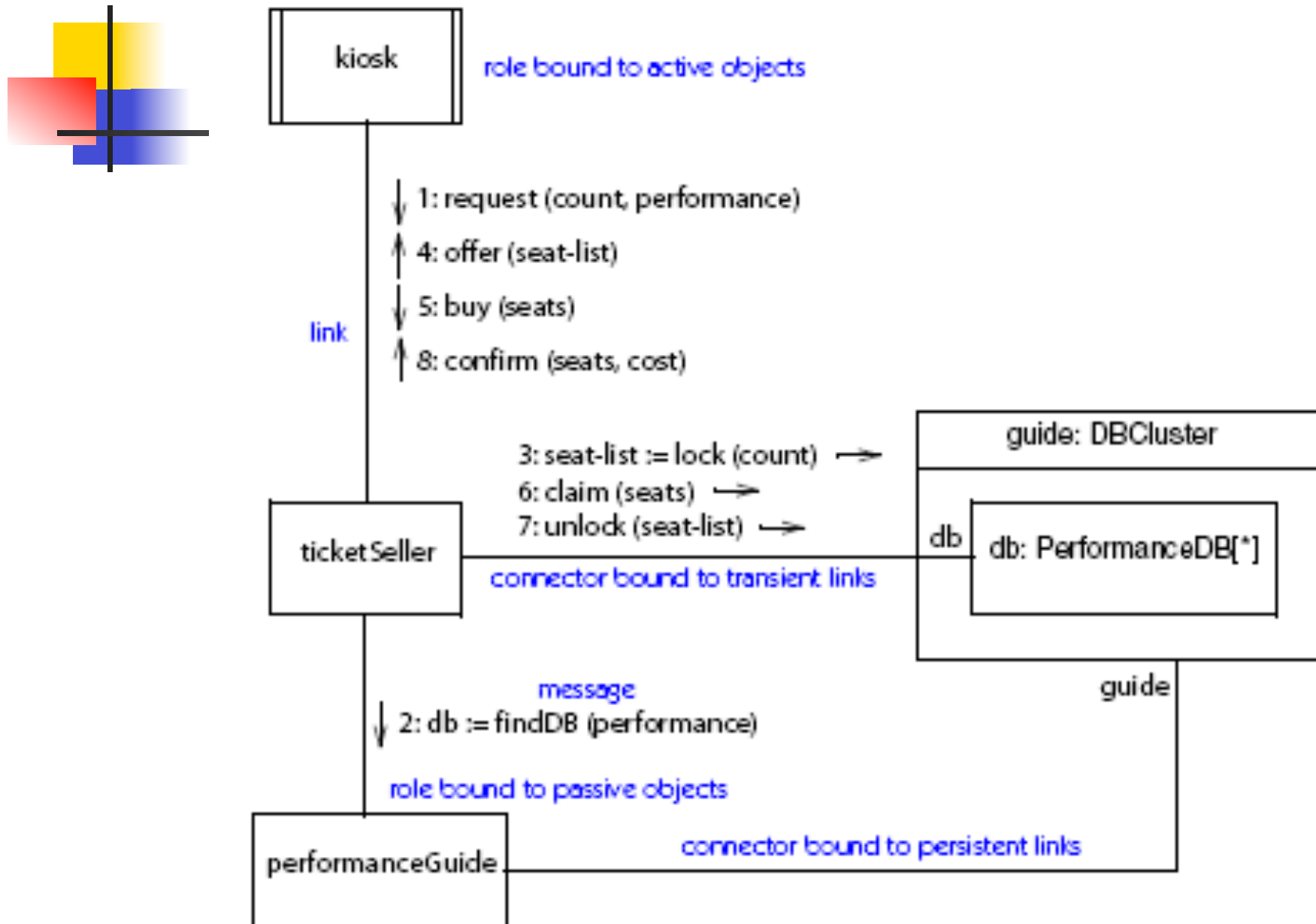
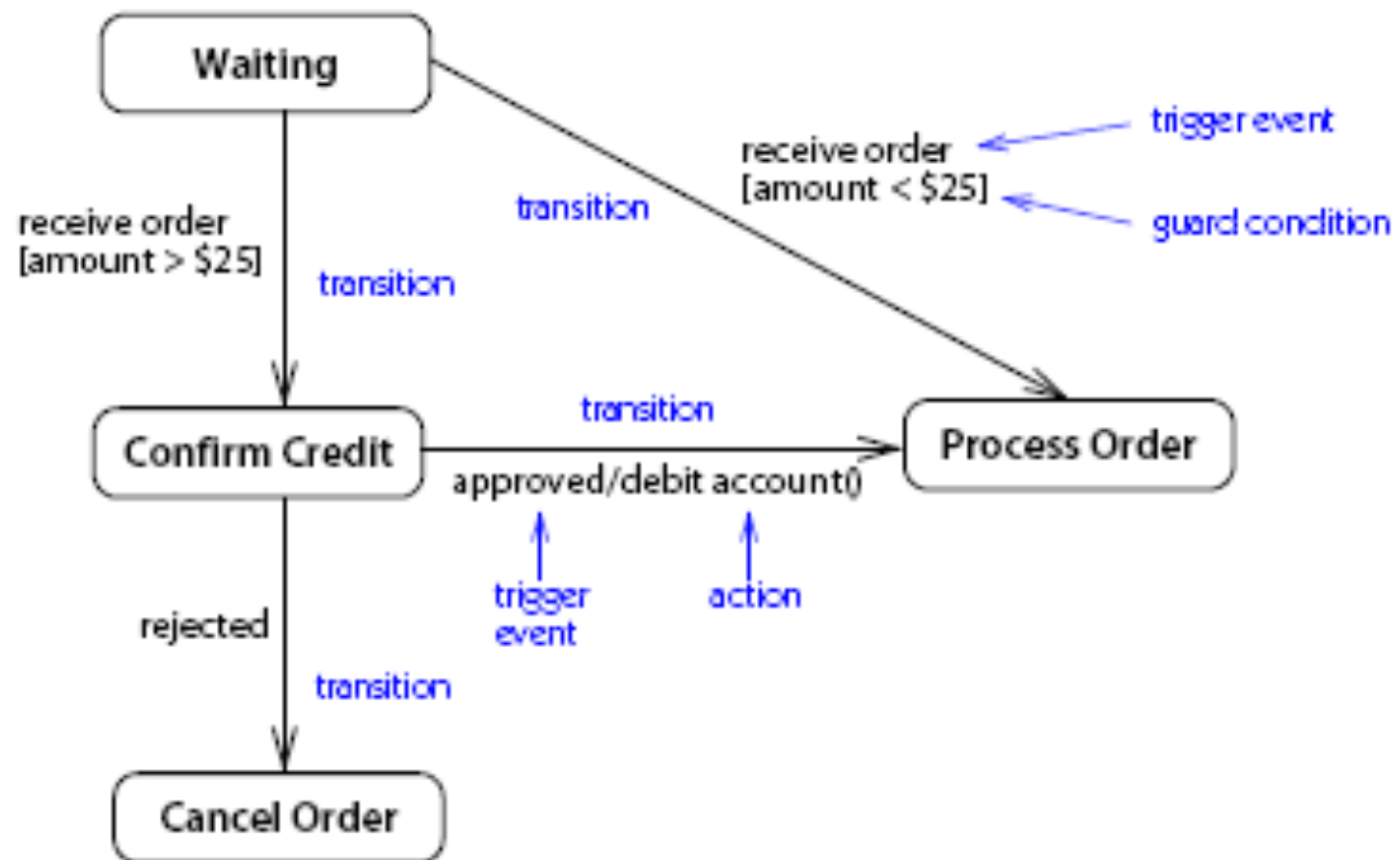


Diagrama de comunicação: Exemplo referente à compra de tickets



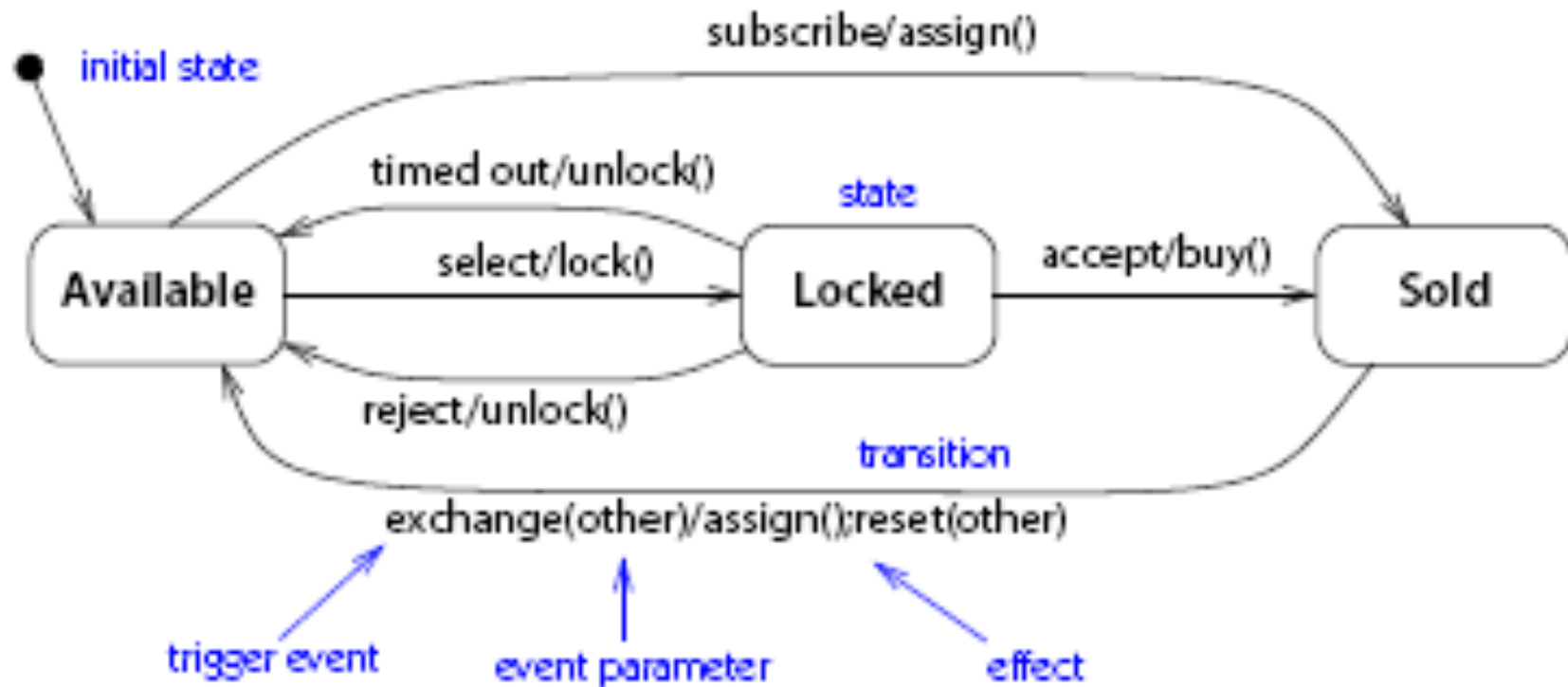
Visão de máquina de estados

Diagrama de estados referente à confirmação de um pedido



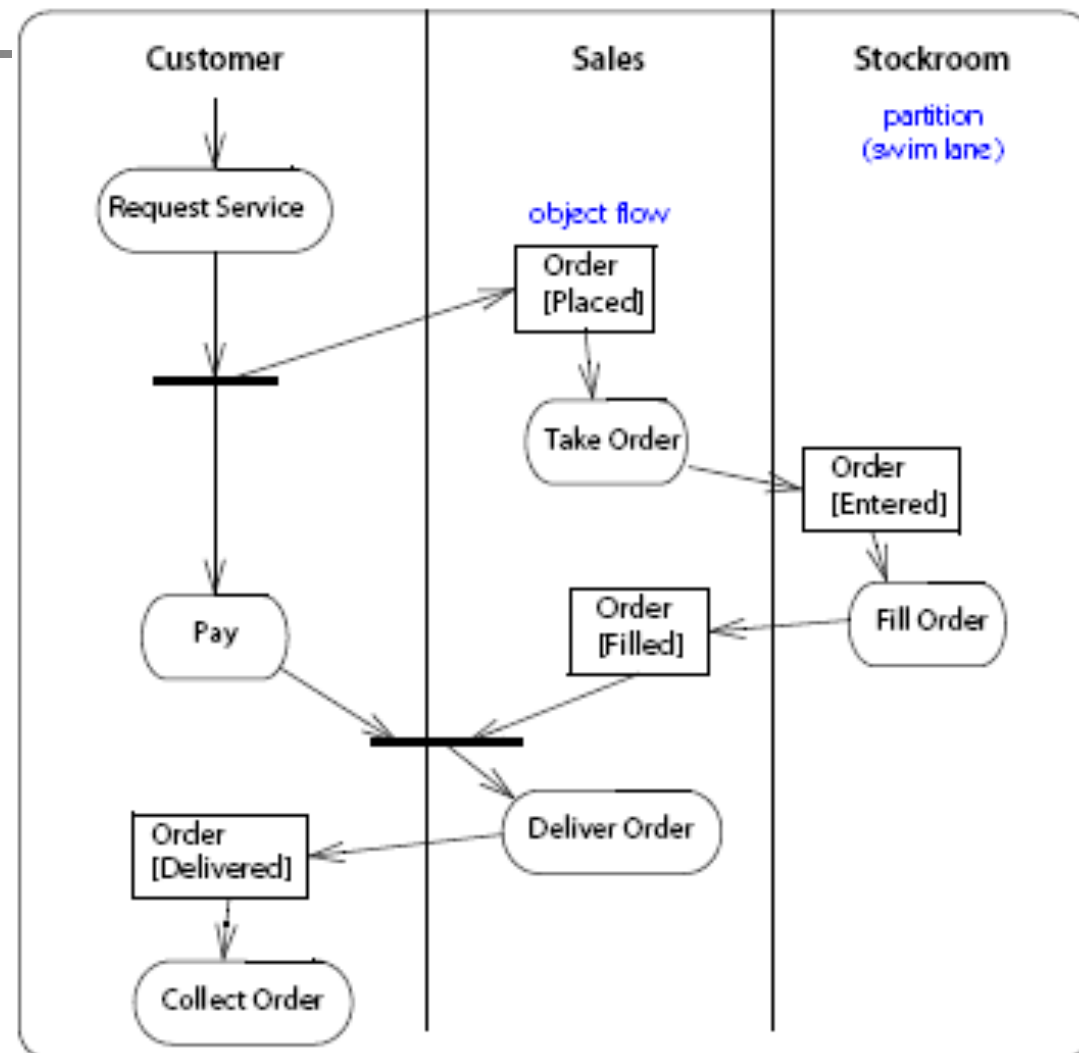
Visão de máquina de estados

- Diagrama de estados referente a um ticket



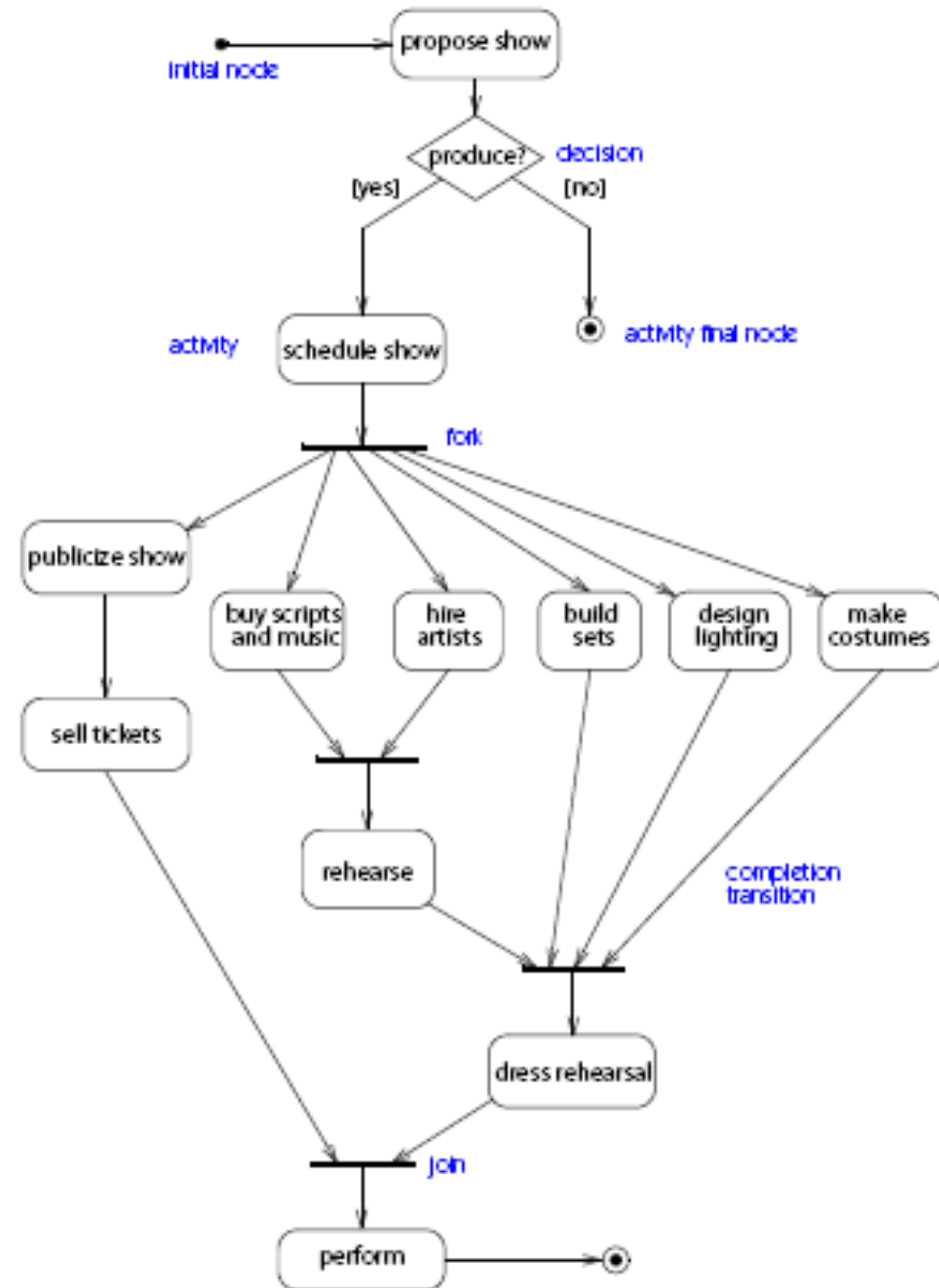
Visão de atividades

- Exemplo de processamento de um pedido



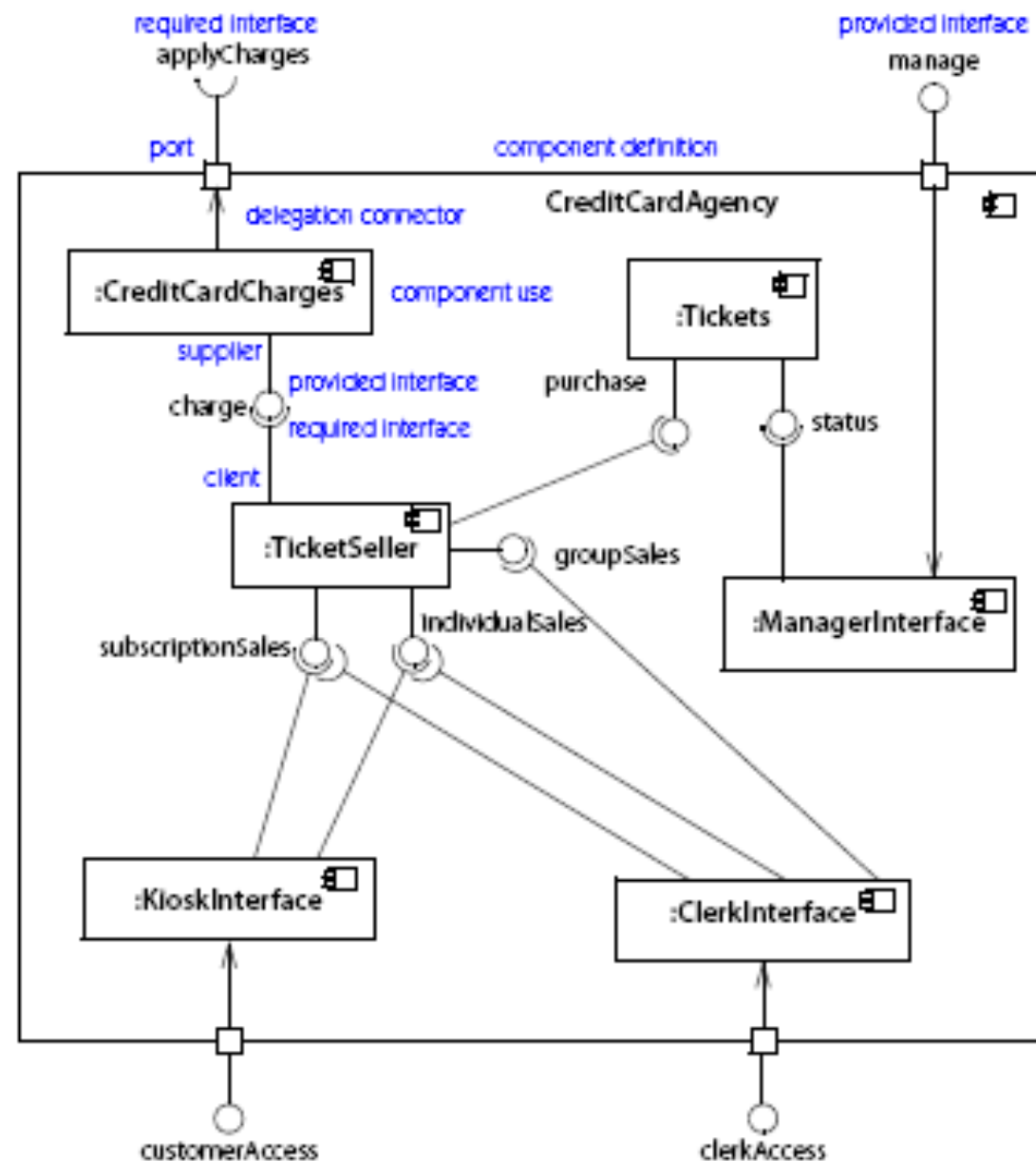


Exemplo de
uma bilheteria
(escalonamento
de show):
diagrama de
atividades



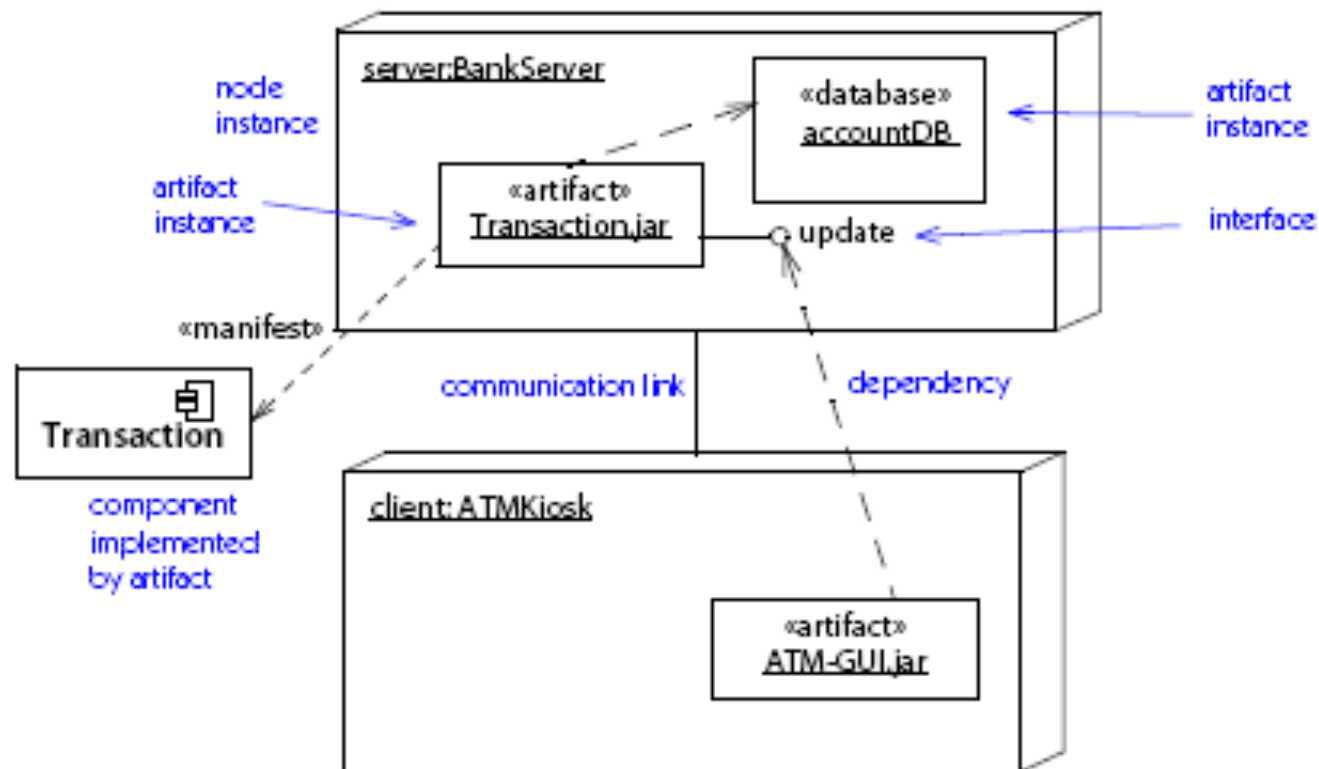
Visão de Projeto

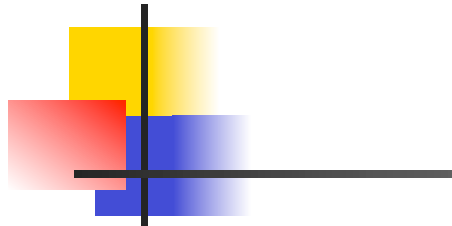
- Diagrama de componentes:
Exemplo de uma bilheteria



Visão de Instalação

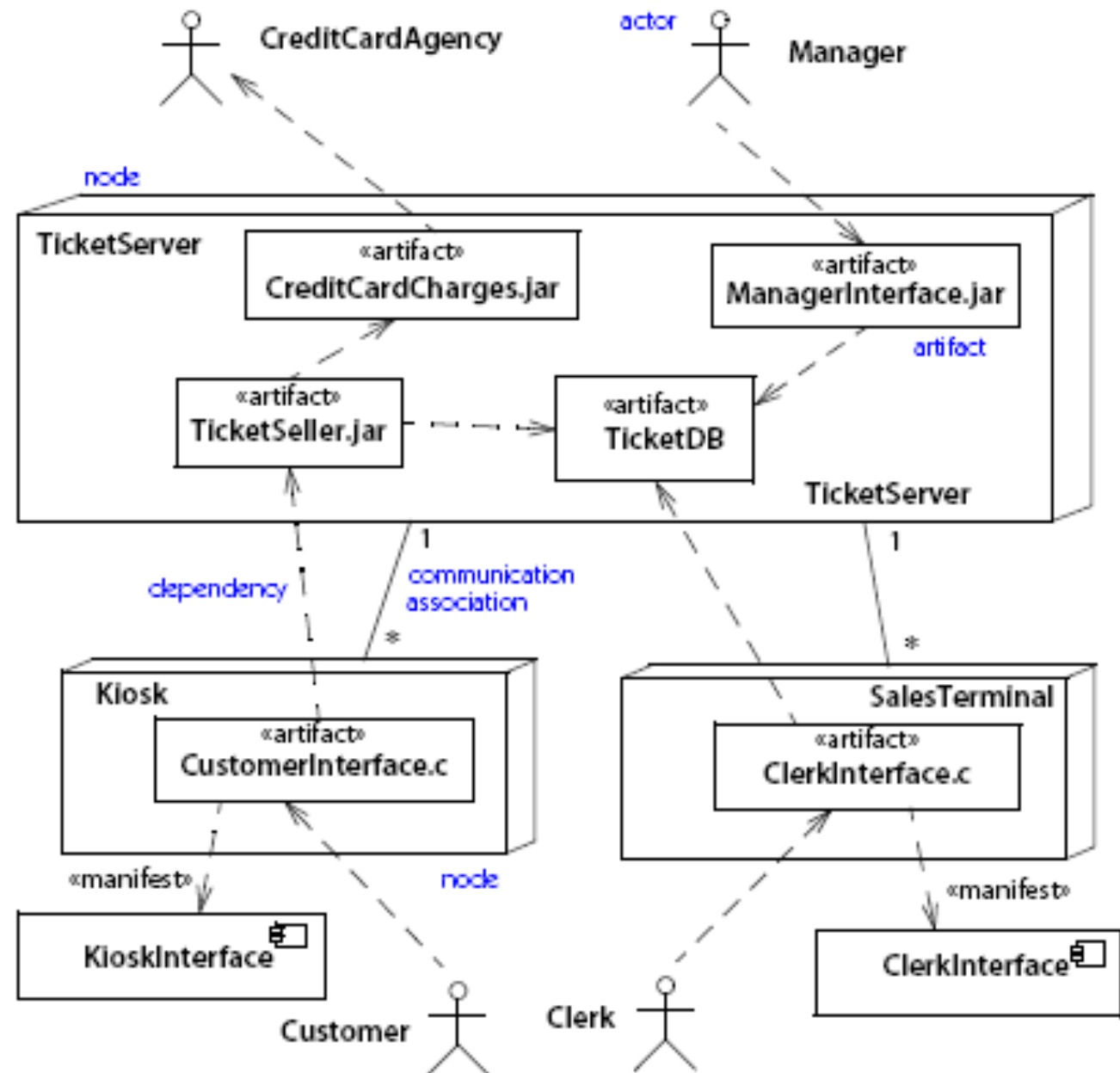
- Exemplo de um Kiosk de venda de tickets





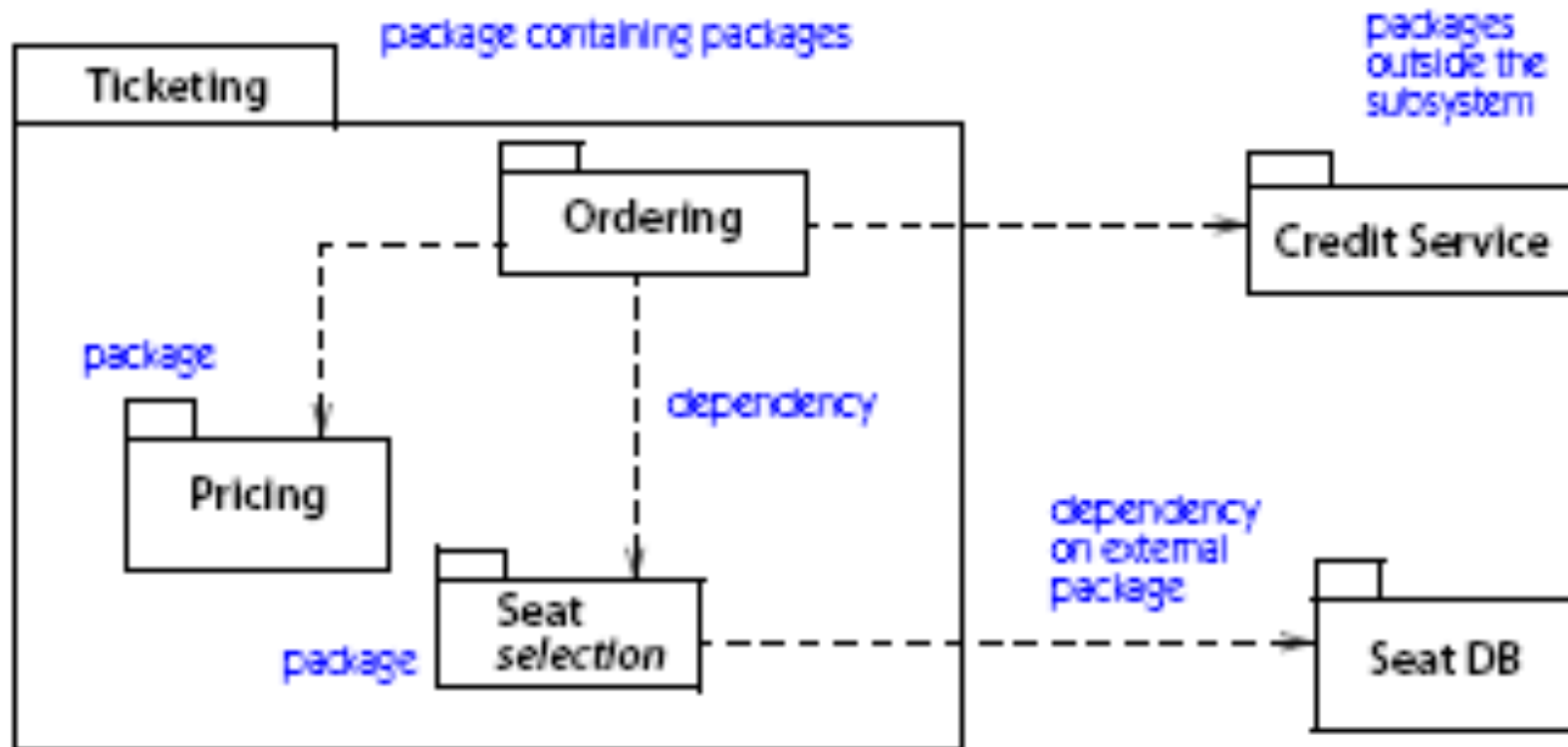
Exemplo de
uma
bilheteria:

diagrama de
instalação



Visão de gerenciamento de modelos

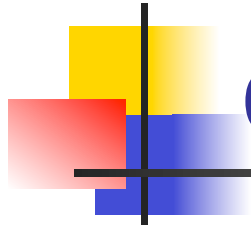
- Exemplo de um Kiosk de venda de tickets





Exercício

- Um banco tem vários caixas eletrônicos que estão geograficamente distribuídos e conectados via rede a um servidor central.
- Cada caixa eletrônico tem uma leitora de cartão, uma caixa de dinheiro, um teclado e uma impressora.
- Utilizando um caixa eletrônico, um cliente pode retirar dinheiro da conta corrente ou poupança, consultar saldo de conta e transferir dinheiro entre contas.
- Uma transação é iniciada quando o cliente insere o cartão na leitora de cartões. Está codificado no cartão: número, data de início e data de expiração. Supondo que o cartão é reconhecido, o sistema verifica se data de expiração é válida, se a senha está correta e o cartão foi roubado ou perdido. O cliente pode fazer três tentativas de inserir a senha. O cartão é confiscado se a terceira falhar. Os cartões roubados ou perdidos também são confiscados.
 - Fazer os diagramas de classes, casos de uso, comunicação para um caso de uso e estados para uma classe.



Conclusões

- O-O traz vantagens superando deficiências de métodos anteriores.
- UML tem sido amplamente utilizada como padrão para especificação de sistemas seguindo a abordagem O-O.