



Introdução à Linguagem UML

Paulo Sousa

Instituto Superior de Engenharia do Porto
Instituto Politécnico do Porto



Parte I

Introdução



UML ...

- UML é uma linguagem para
 - Visualização
 - Especificação
 - Construção
 - Documentação
- UML fornece método padrão para descrever um sistema tendo em conta aspectos conceptuais e/ou concretos desse sistema



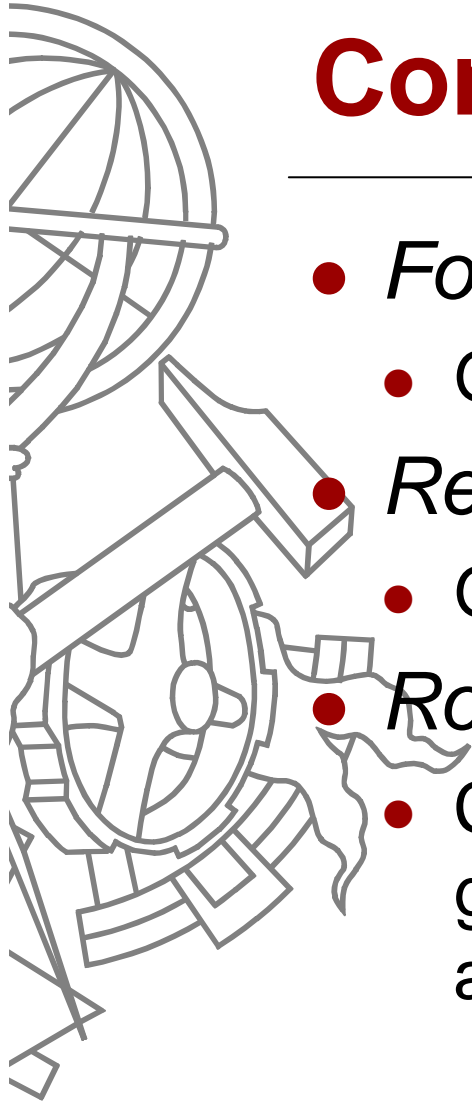
Visualização

- Modelos explícitos facilitam a comunicação
- Algumas estruturas transcendem o que pode ser representado numa linguagem de programação
- Cada símbolo tem uma semântica bem definida



Especificação

- A notação UML permite a especificação das decisões importantes ao nível da análise, desenho e implementação



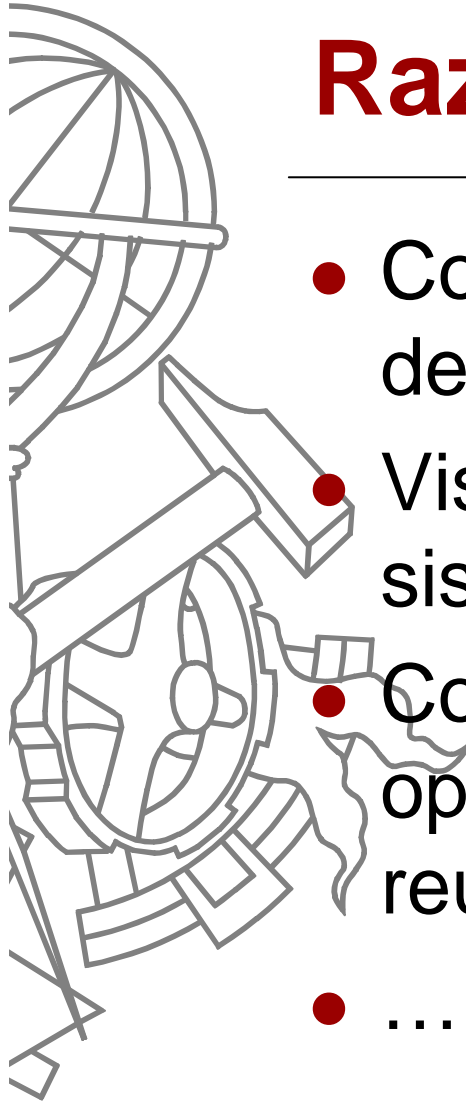
Construção

- *Forward engineering*
 - Geração de código com base no modelo
- *Reverse engineering*
 - Geração do modelo com base no código
- *Round-trip engineering*
 - Ciclo iterativo de desenvolvimento com geração de código a partir de um modelo e actualização do modelo com base no código



Documentação

- A notação UML permite a criação de documentação dos artefactos existentes o sistema:
 - Conceitos do problema
 - Cenários de instalação
- É possível acrescentar links para documentação externa
 - Documentos de requisitos, planos de testes, ...



Razões para modelar

- Comunicar a estrutura e comportamento desejado(desejável) para o sistema
- Visualizar e controlar a arquitectura do sistema
- Compreender o sistema e expor oportunidades de simplificação e reutilização
- ...



Tipos de diagramas

- Estruturais
 - Consideram aspectos estáticos do sistema
- Comportamentais
 - Consideram aspectos dinâmicos do sistema



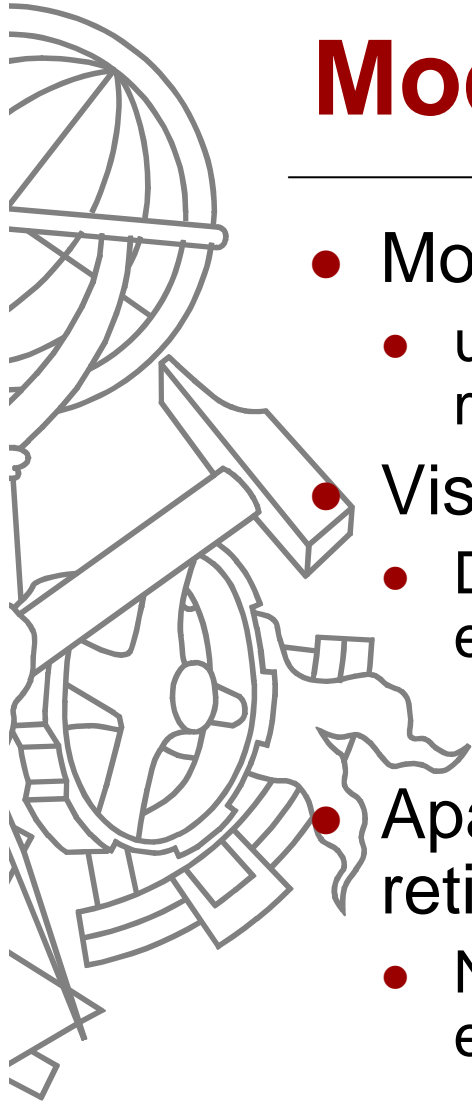
Tipos de diagramas (II)

- Estruturais
 - Classes
 - Objectos
 - Componentes
 - Instalação



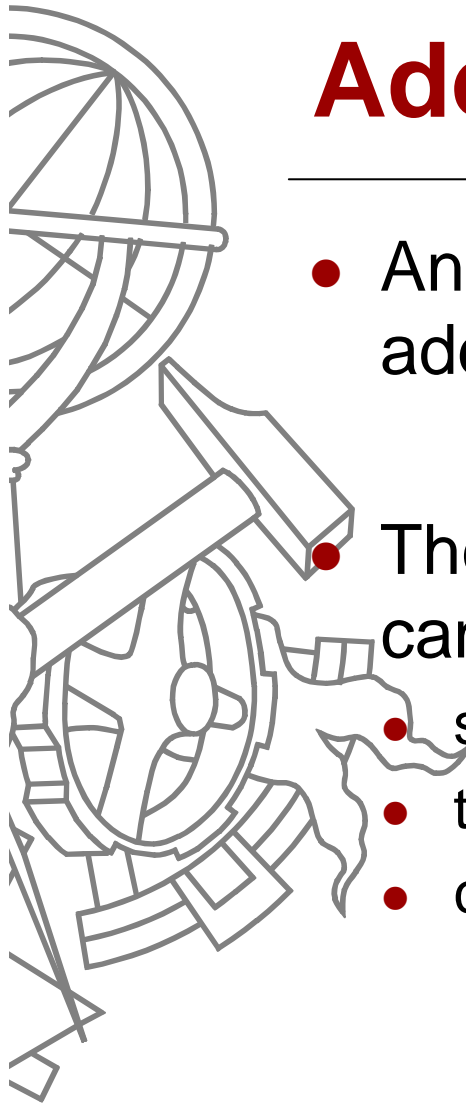
Tipos de diagramas (III)

- Comportamentais
 - Casos de utilização
 - Sequência
 - Colaboração
 - Estados
 - Actividades



Modelos e vistas

- Modelo
 - uma “base de dados” dos elementos criados para modelar o problema
- Vistas
 - Diagramas sobre o modelo onde são colocados os elementos existentes
- Apagar um elemento de um diagrama **não** retira esse elemento do modelo!
 - Nas ferramentas Rational “del” remove do diagrama e “CTRL+del” apaga do modelo

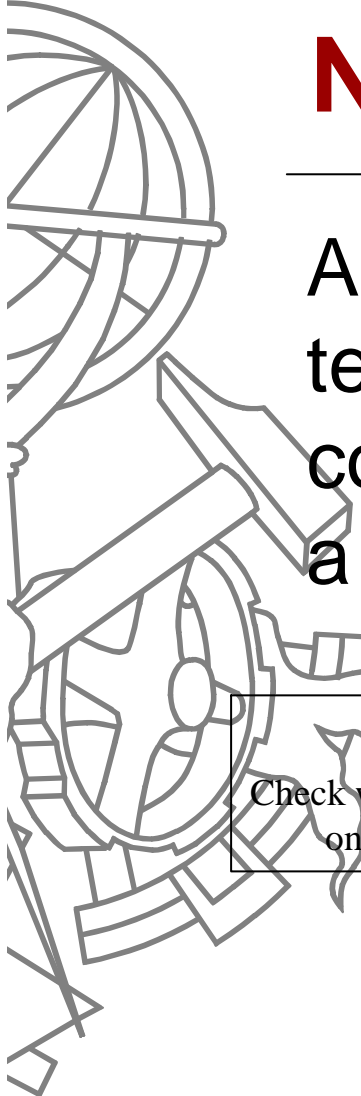


Adornments and Extensibility

- An adornment is an item, such as a note, that adds text or graphical detail to a model.
- The UML offers various mechanisms that you can use to extend the “official” language.
 - stereotypes
 - tagged values
 - constraints

Notes

A **note** is a graphical symbol containing text and/or graphics that offer(s) some comment or detail about an element within a model.



Check with Mike
on this.

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



See encrypt.doc

Stereotypes

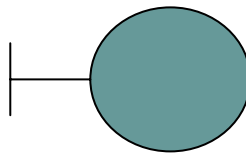
A **stereotype** is an extension of the vocabulary of the UML that allows you to create a new kind of “building block” that’s specific to the problem you’re trying to solve.



«interface»
Observer

The diagram shows a UML class diagram for an interface. It is a rectangular box divided into three horizontal compartments. The top compartment contains the text «interface» and Observer. The middle compartment is empty. The bottom compartment contains the text update().

update()

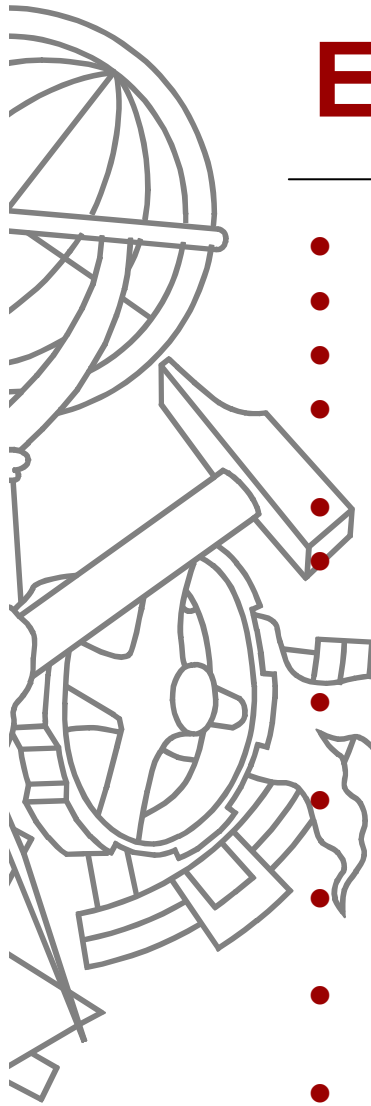


Humidity Sensor



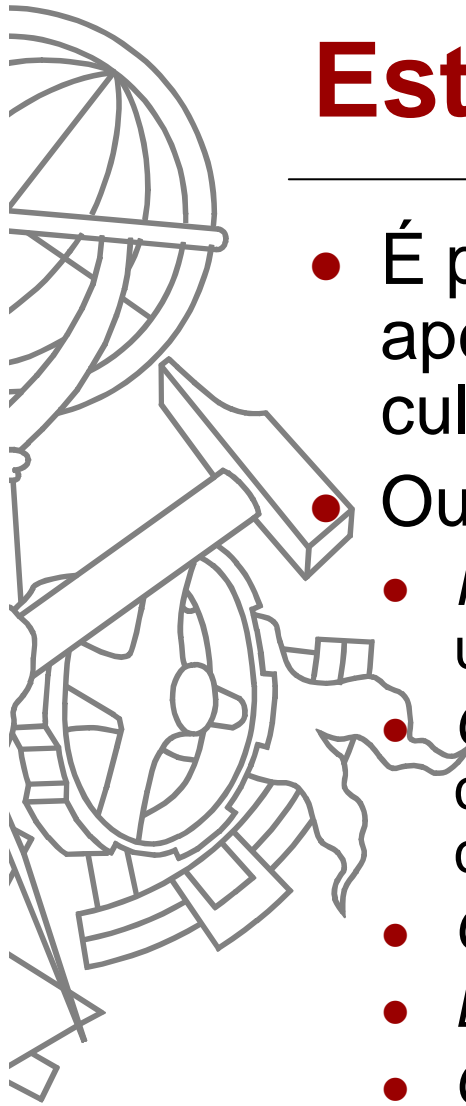
«control»
TargetTracker

The diagram shows a UML class diagram for a TargetTracker. It is a rectangular box divided into two horizontal compartments. The top compartment contains the text «control» and TargetTracker. The bottom compartment is empty.



Estereótipos padrão

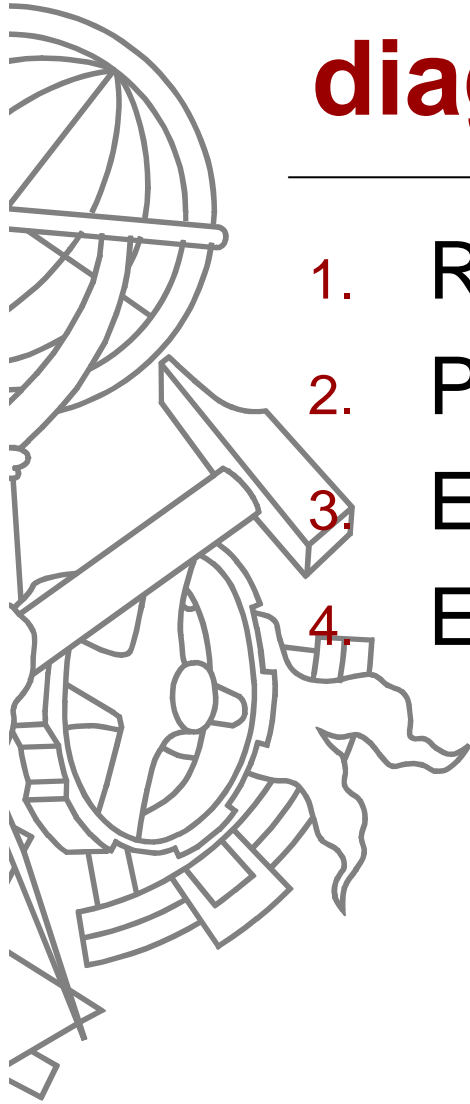
- *copy* – indica uma cópia exacta de um objecto;
- *document* – indica um documento;
- *executable* – indica um componente executável num nó;
- *extend* – usado para indicar uma extensão do comportamento padrão de um caso de utilização (ex., situações de erro);
- *file* – indica um ficheiro com código fonte ou dados;
- *include* – indica que um determinado caso de utilização inclui explicitamente um outro case de utilização (ex., comportamentos comuns);
- *invariant* – representa uma restrição sobre um elemento que deve ser sempre satisfeita (i.e., deve ser sempre verdadeira);
- *postcondition* – representa uma restrição sobre uma operação que deve ser sempre verdadeira após a execução da operação;
- *precondition* – representa uma restrição sobre uma operação que deve ser sempre verdadeira antes da execução da operação;
- *system* – estereotipo associado a um “package” que representa o sistema que se está a modelar;
- *table* – representa uma tabela de base de dados;



Estereotipos não padrão

- É possível criar novos estereótipos associados apenas com o problema em questão ou com a cultura da empresa
- Outros estereótipos não padrão
 - *Form* – indica um formulário de interação com o utilizador)
 - *COM* – indica que a classe é implementada como um componente COM; ou o componente físico é um componente COM
 - *CORBA* – igual ao anterior para CORBA
 - *DLL* – indica que o componente é uma DLL
 - *Constructor* ou *create* – indica um método construtor

Utilização dos diferentes diagramas



1. Requisitos/funcionalidades
2. Processos
3. Estrutura lógica
4. Estrutura física



Requisitos/funcionalidades

dialogando com o cliente identificar as funcionalidades de alto nível (as “grandes” funções do sistema) pretendidas no sistema para cada perfil de utilizador, recorrendo a diagramas de casos de utilização



Processos

continuando o diálogo com o cliente, analisar e efectuar uma descrição de alto nível dos processos existentes no sistema e das interacções entre os diferentes intervenientes nesses processos (“workflow”), recorrendo a diagramas de interacção (sequência e colaboração).



Estrutura lógica

partindo do diagramas de casos de utilização e dos diagramas de interacção de alto nível, identificar e descrever detalhadamente as diferentes entidades existentes no sistema (recorrendo a diagramas de classes), bem como detalhar os diagramas de interacção anteriores por forma a incluir as classes de implementação identificadas e respectivas operações (diagramas de sequência e de colaboração).



Estrutura física

identificar os diferentes elementos físicos do sistema (ex., bibliotecas de funções, executáveis) recorrendo-se de diagramas de componentes, bem como identificar os recursos de “hardware” necessários à instalação do sistema usando para tal diagramas de instalação.



Parte II

Diagramas



Diagramas Estruturais

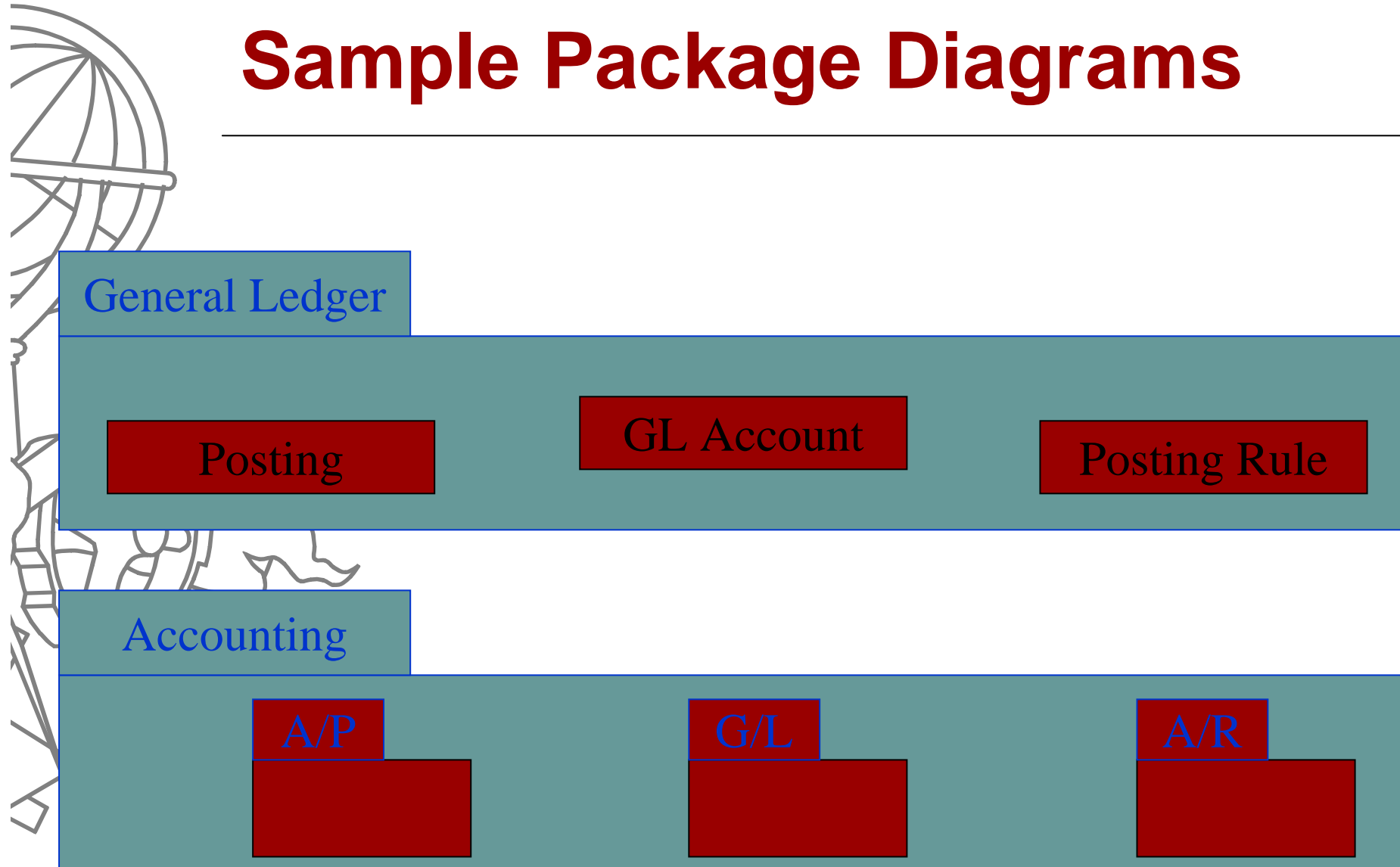
Diagrama de “Pacotes”



Package

- A **package** is a general-purpose mechanism for organizing elements of a model, such as classes or diagrams, into groups.
- Every element within a model is uniquely owned by one package. Also, that element's name must be unique within that package.

Sample Package Diagrams



Diagramas Comportamentais



Diagrama de Casos de Utilização



Use Case and Actor

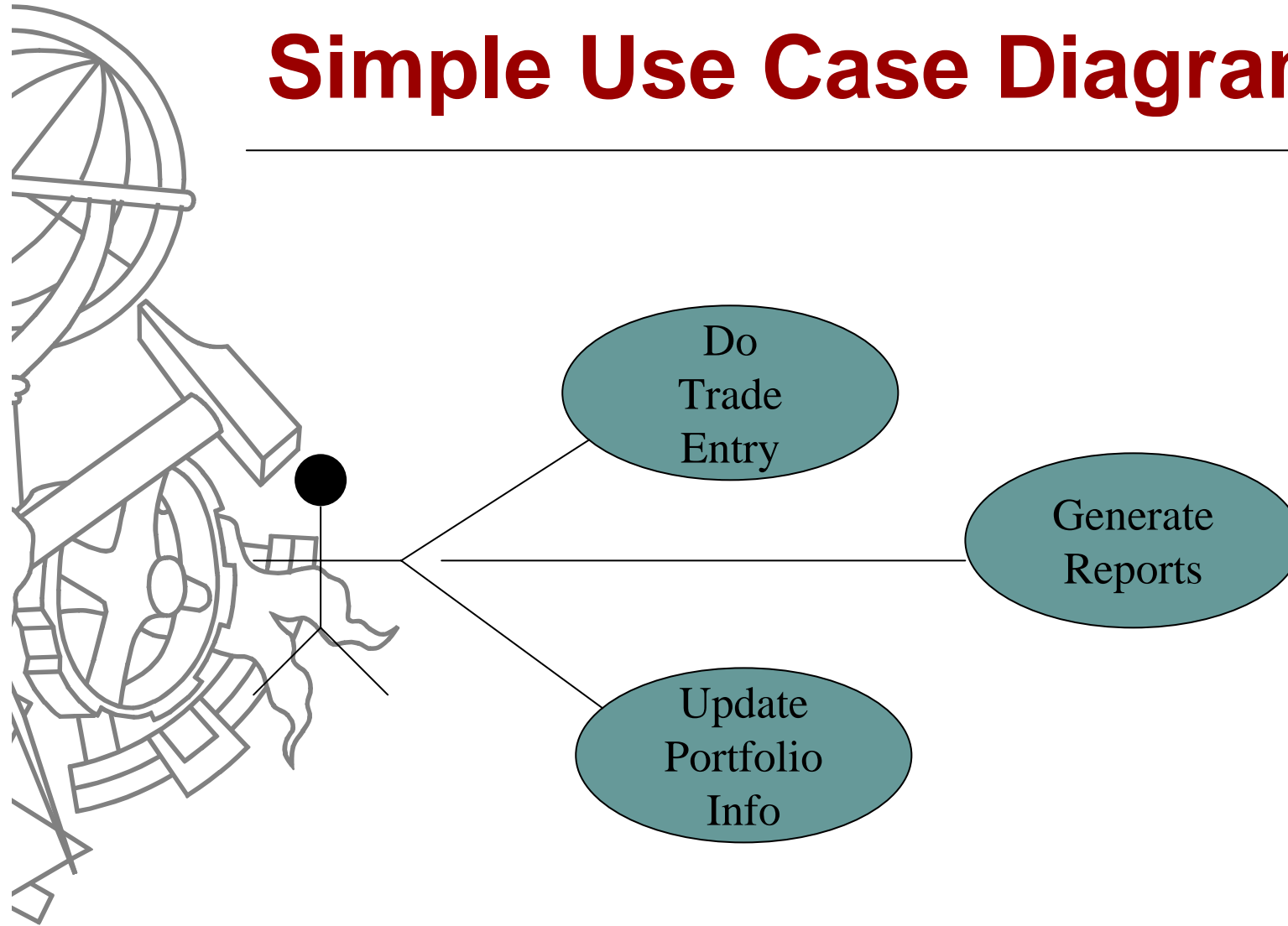
- A **use case** is a sequence of actions, including variants, that a system performs to yield an observable result of value to an actor.
- An **actor** is a coherent set of roles that human and/or non-human users of use cases play when interacting with those use cases.



Flows of Events

- The main flow of events (basic course of action) describes the “sunny-day” scenario.
- Each exceptional flow of events (alternate course of action) describes a variant, such as an error condition or an infrequently occurring path.

Simple Use Case Diagram





Organizing Use Cases

- packages
- generalization
- include
- extend

Use Case Packages

Packages of use cases can be very useful in assigning work to sub-teams.



Portfolios

Create
New
Portfolio

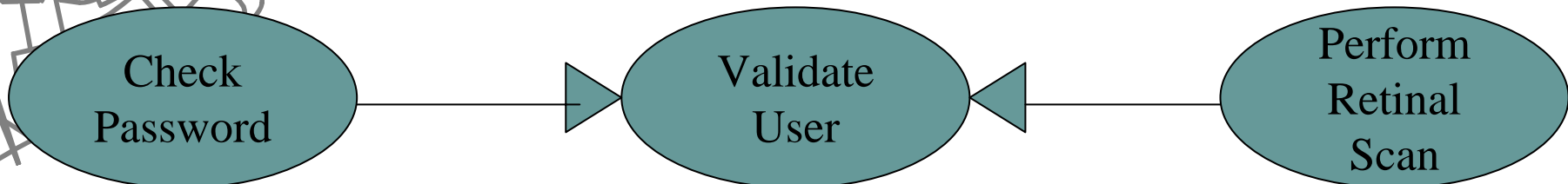
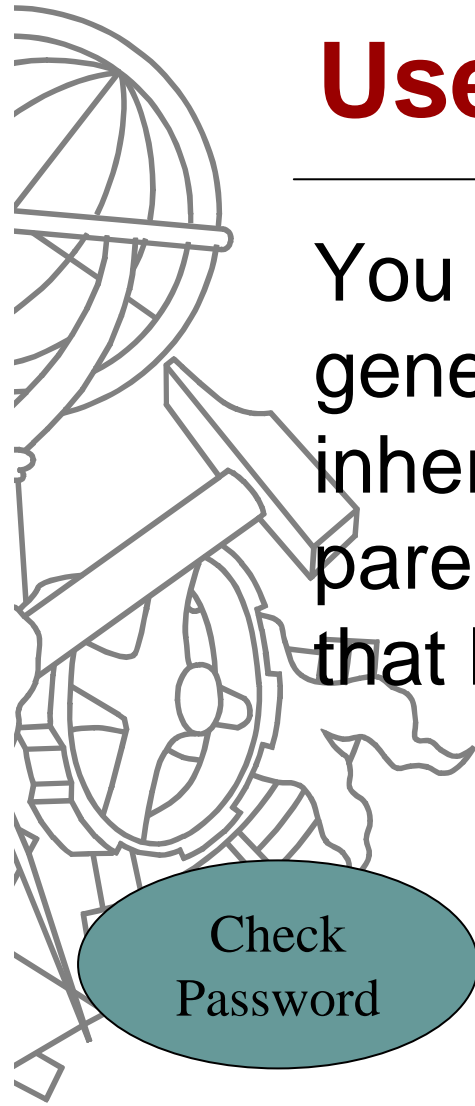
Aggregate
Portfolios

View
Portfolio

Generate
Portfolio
Report

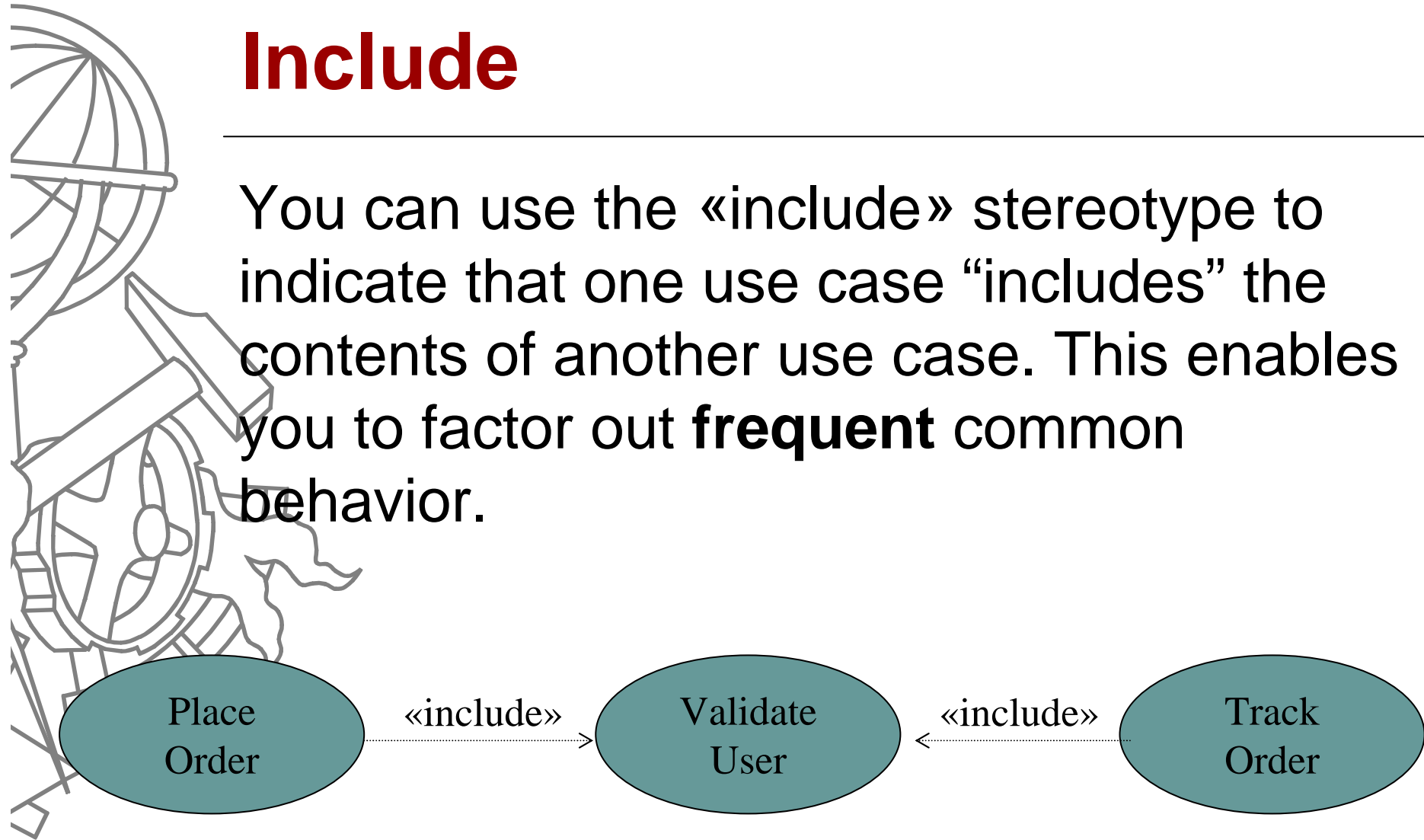
Use Case Generalization

You can generalize use cases just like you generalize classes: the child use case inherits the behavior and meaning of the parent use case, and can add to or override that behavior.



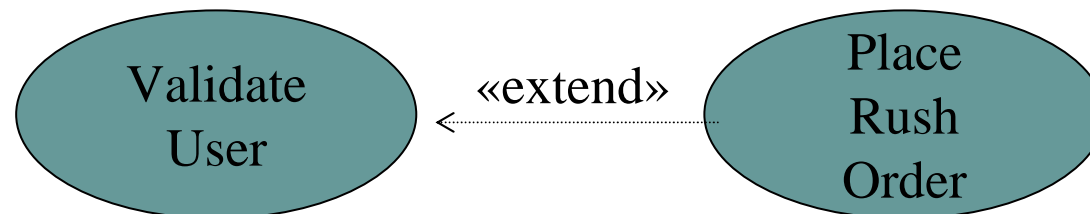
Include

You can use the «include» stereotype to indicate that one use case “includes” the contents of another use case. This enables you to factor out **frequent** common behavior.

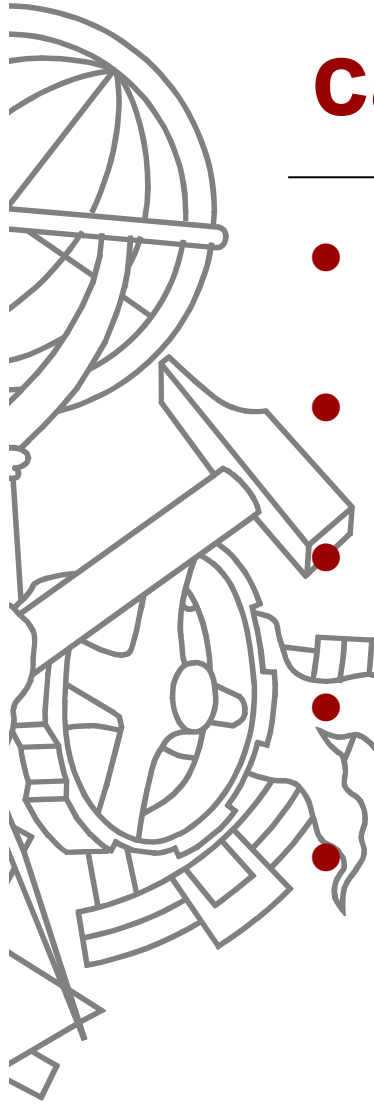


Extend

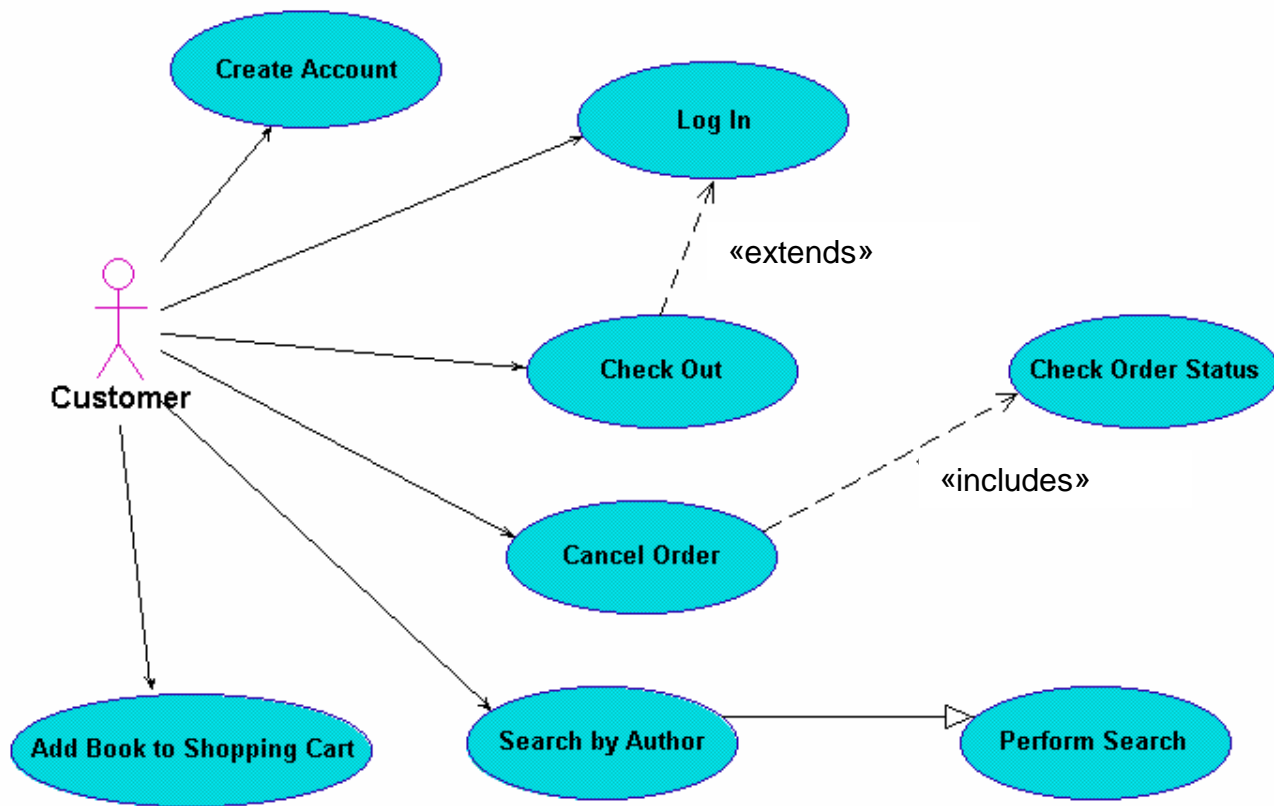
You can use the «extend» stereotype to indicate that one use case is “extended” by another use case. This enables you to factor out **infrequent** common behavior.



Algumas notas sobre use cases



- Relações entre use cases **não** significam fluxo de execução
- Sistemas externas são modelados como actores
- Só se modelam os use cases do próprio sistema
- O importante é o texto do use case e não o diagrama
- O texto do use case deve descrever o fluxo normal de eventos
 - As situações excepcionais são descritas numa secção à parte



Exercício

- Supor um cenário de portal intranet de uma empresa para publicação dos contactos e informação sobre funções dos colaboradores
- Elaborar um diagrama de use cases para este cenário

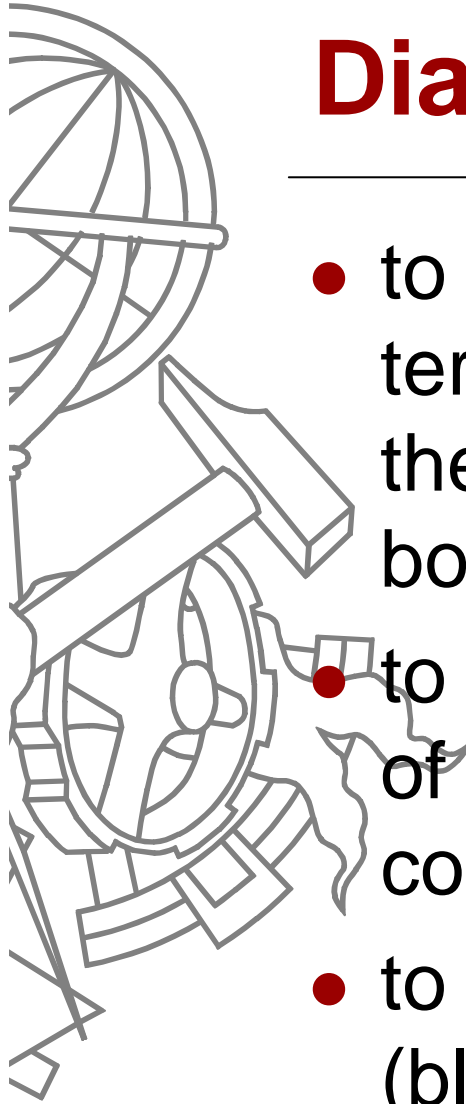




Diagramas Estruturais

Diagrama de Classes

Common Uses of Class Diagrams



- to model vocabulary of the system, in terms of which abstractions are part of the system and which fall outside its boundaries
- to model simple collaborations (societies of elements that work together to provide cooperative behavior)
- to model logical database schema (blueprint for conceptual design of database)



Class

- A **class** is a description of a set of objects that share the same attributes, operations, relationships, and semantics.
- An **attribute** is a named property of a class that describes a range of values that instances of the property may hold.
- An **operation** is a service that can be requested from an object to affect behavior.

Class Notation



| |
|-------------|
| Name |
| Attributes |
| Operations |



Atributos e operações

- Atributos

- Nome-do-atributo : tipo-de-dados [= valor-inicial]
- `idade : int = 0`
- `dtNascimento : Data`

- Operações

- Nome-da-operação (argumentos) : tipo-de-retorno
- Argumentos
 - [{ in | out | inout }] Nome-do-argumento : tipo-de-dados
- `setIdade(in idade : int) : void`
- `getIdade() : int`



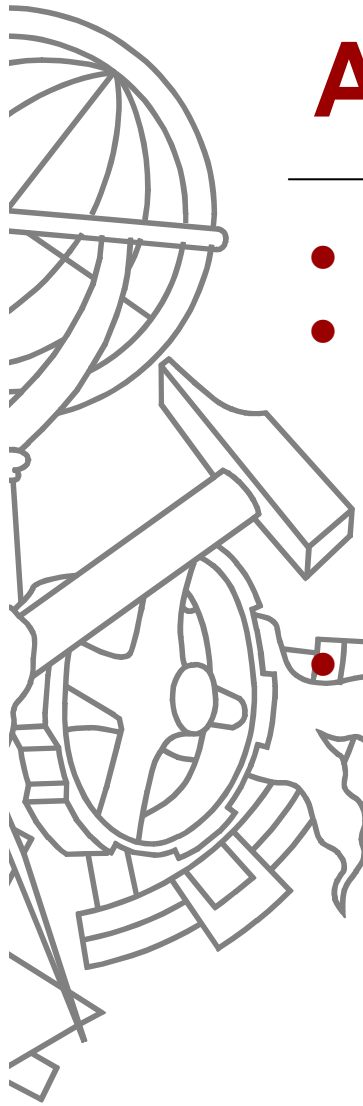
Visibilidade

- Indica o tipo de “visibilidade” de um atributo ou de uma operação de uma classe
 - Privado (-)
 - Apenas é acessível do interior; Só as operações da própria classe tem acesso
 - `-idade : int`
 - Público (+)
 - É acessível do exterior; Qualquer objecto tem acesso
 - `+getIdade() : int`
 - Protegido (#)
 - É acessível pelo interior e por classes derivadas⁴⁴



Visibilidade (II)

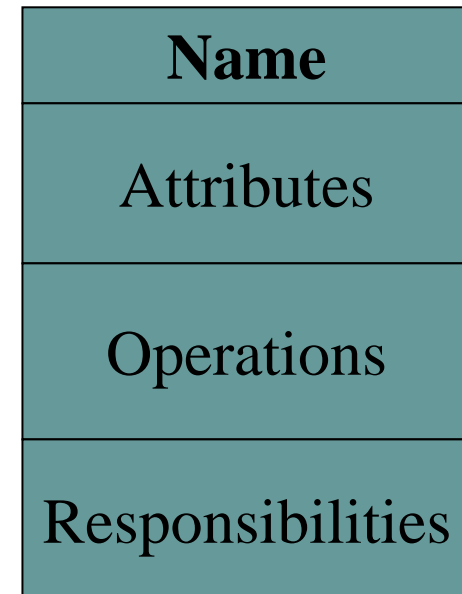
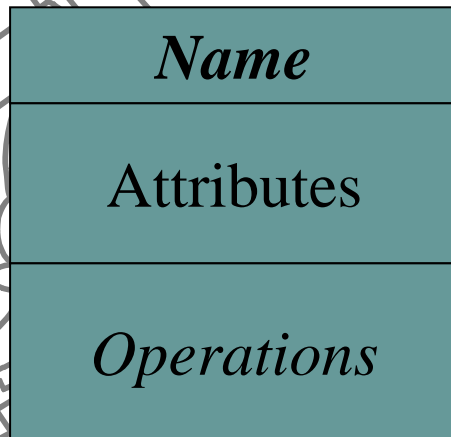
- Tipicamente **todos** os atributos de uma classe devem ser privados
 - Evita a manipulação directa do estado do objecto
 - Garante que o objecto estará sempre num estado válido (pelo menos é única e exclusivamente da responsabilidade da classe)
- Fornece-se operações públicas para aceder aos atributos da classe bem como modificar o seu valor



Accessors/Mutators

- São tipos de métodos/operações especiais de uma classe
- *Accessor* ou *getter*
 - Um método que permite consultar o valor de um atributo
 - Normalmente o nome do método é construído com a palavra *get* e o nome do atributo:
 - `getDia() : int`
- *Mutator* ou *setter*
 - Um método que permite modificar o valor de um atributo
 - Este método consegue validar o novo valor evitando dessa forma colocar o objecto num estado inválido
 - Normalmente o nome do método é construído com a palavra *set* e o nome do atributo:
 - `setDia(d : int) : void`

Alternative Class Notations



italics → abstract



Métodos e classes abstractas

- Diz-se de um método que é definido em termos de “assinatura” mas não é implementado
 - Apenas se define o nome do método, os argumentos e o tipo de retorno sem fornecer a sua implementação
- Uma classe abstracta é aquela que tem pelo menos um método abstracto
 - Não permite a criação de objectos desta classe pois nem todos os comportamentos estão definidos
 - Obriga à implementação dos métodos abstractos em classes derivadas
- Notação UML: itálico
 - *mover() : void*



Relationships

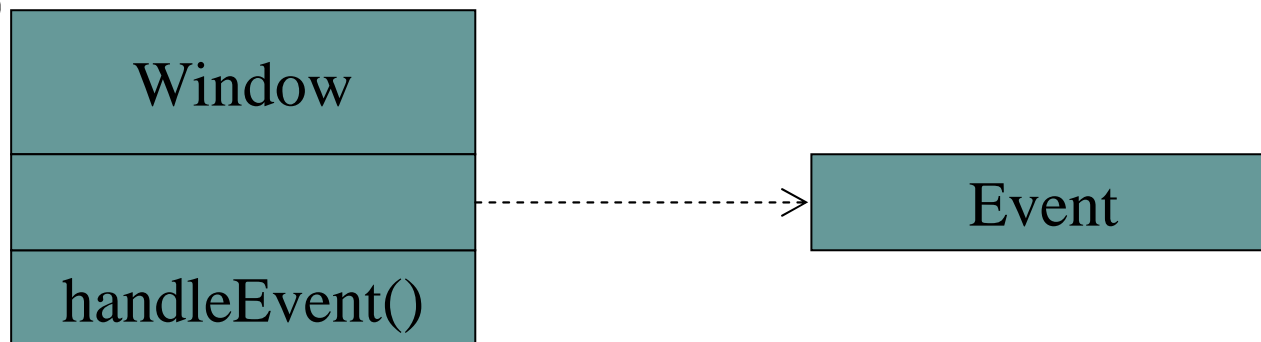
connections between classes

- dependency
- generalization
- association

Dependency

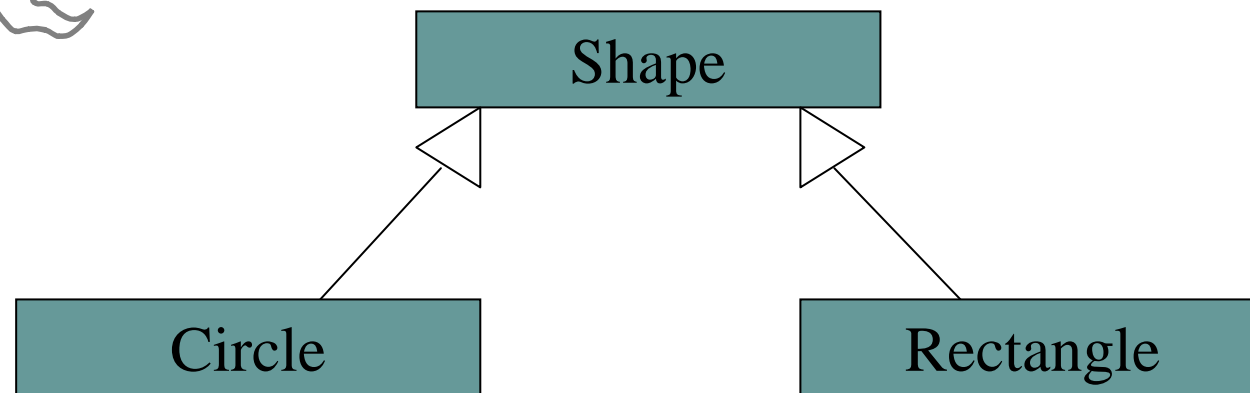
A **dependency** is a “using” relationship within which the change in the specification of one class may affect another class that uses it.

Example: one class uses another in operation



Generalization

A **generalization** is a “kind of” or “is a” relationship between a general thing (superclass or parent) and a more specific thing (subclass or child).



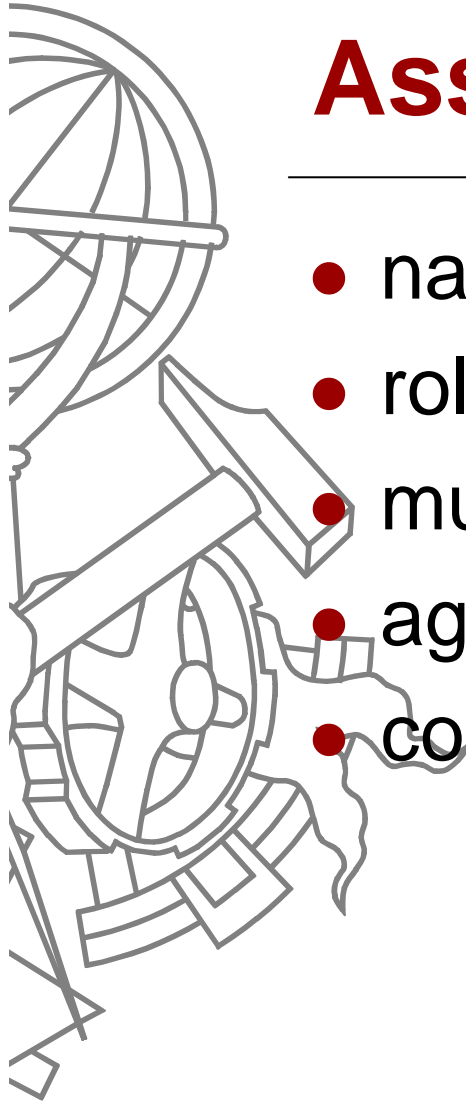
Association

An **association** is a structural relationship within which classes or objects are connected to each other. (An association between objects is called a link.)



Person

Company



Association Adornments

- name
- role
- multiplicity
- aggregation
- composition

Association Name

describes nature of relationship:

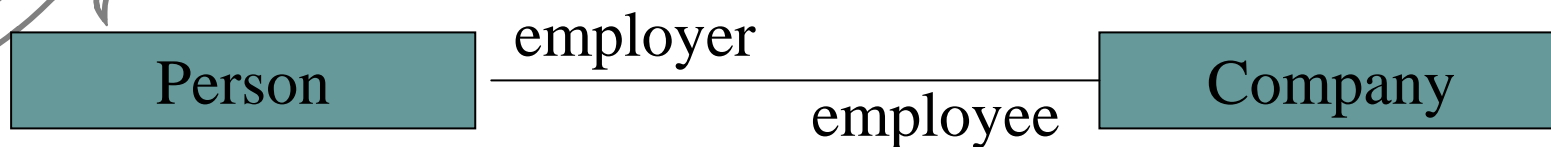


can also show direction to read name:



Association Roles

- describe “faces” that classes present to each other within association
- class can play same or different roles within different associations



Association Multiplicity

- possible values same as for classes: explicit value, range, or * for “many”
- Example: a Person is employed by one Company; a Company employs one or more Persons



Associação direccional

- Corresponde a um atributo na classe como referência para um objecto da classe associada



EntidadeMundo

The diagram shows two light blue rectangular boxes with black borders. The top box contains the text 'EntidadeMundo' and the bottom box contains 'Posicao'. A black arrow points from the bottom of the top box to the top of the bottom box, indicating a directional association from EntidadeMundo to Posicao.

Posicao

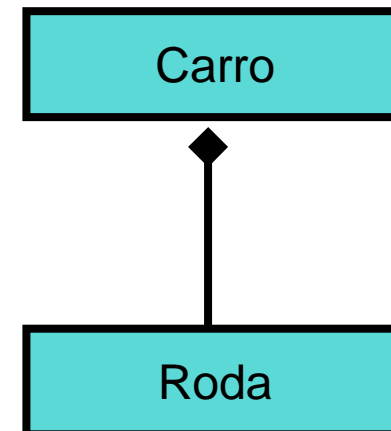
```
class EntidadeMundo {  
    ...  
    Posicao pos;  
    ...  
}
```

Multiplicidade nas associações



- Quando uma associação tem uma multiplicidade diferente de 1 é necessário uma **coleção** de referências
 - Por exemplo, um vector

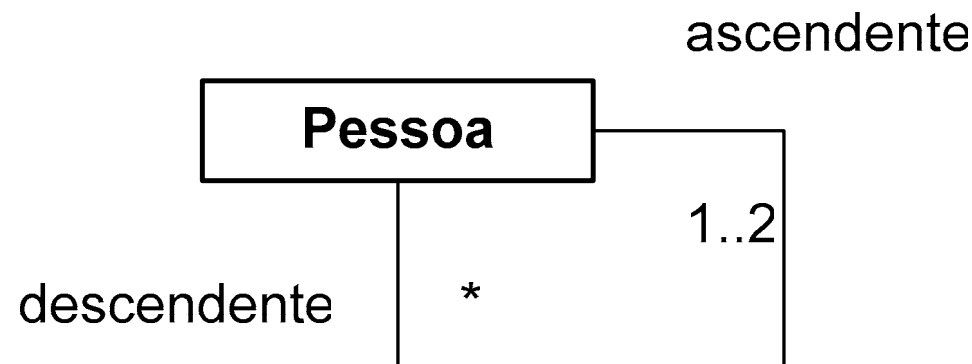
```
class Carro {  
    ...  
    Roda[] rodas;  
    ...  
}
```

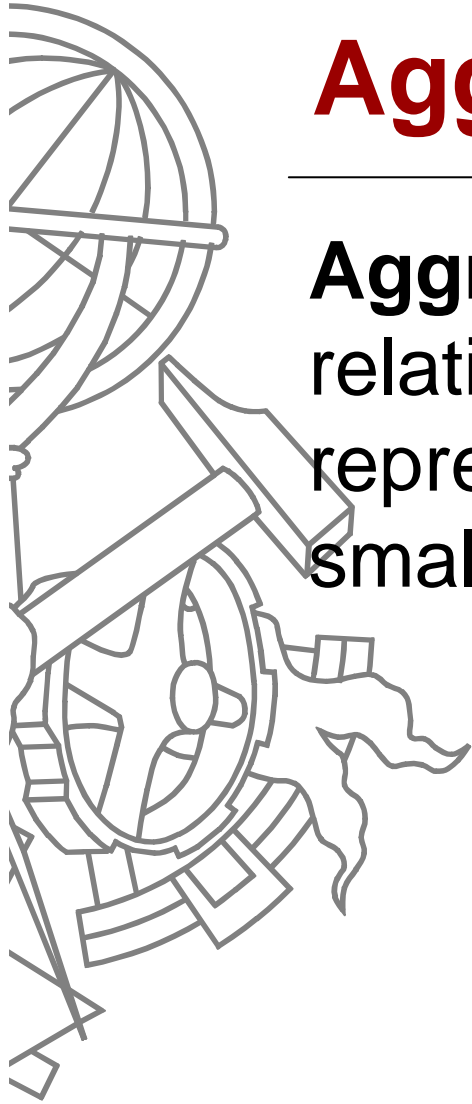




Associações

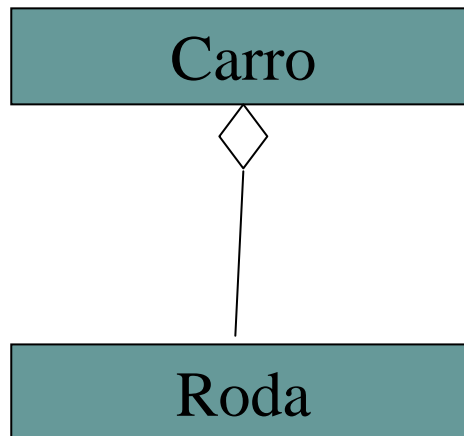
- Um objecto pode ter associações com outros objectos da mesma classe
- Em termos de diagrama de classes UML, essas associações representam-se como “arcos”





Aggregation

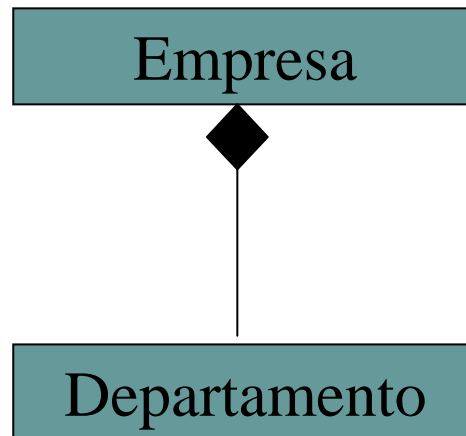
Aggregation is a “whole/part” or “has a” relationship within which one class represents a larger thing that consists of smaller things.





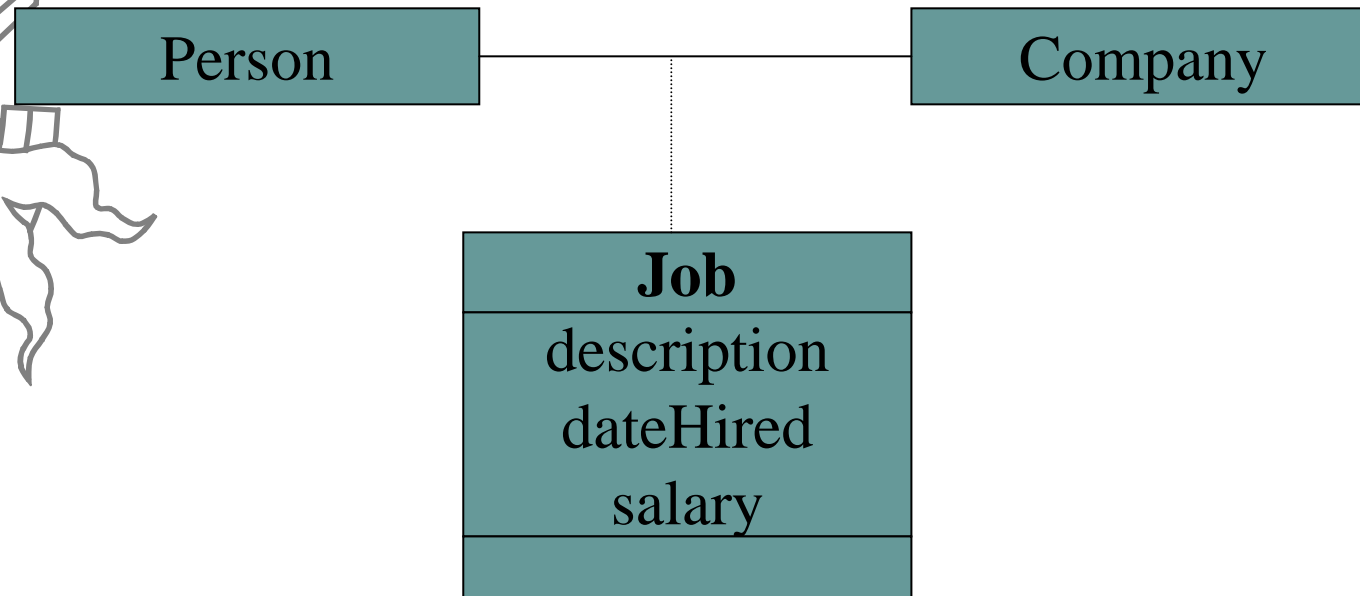
Composition

Composition is a special form of aggregation within which the parts are *inseparable* from the whole.



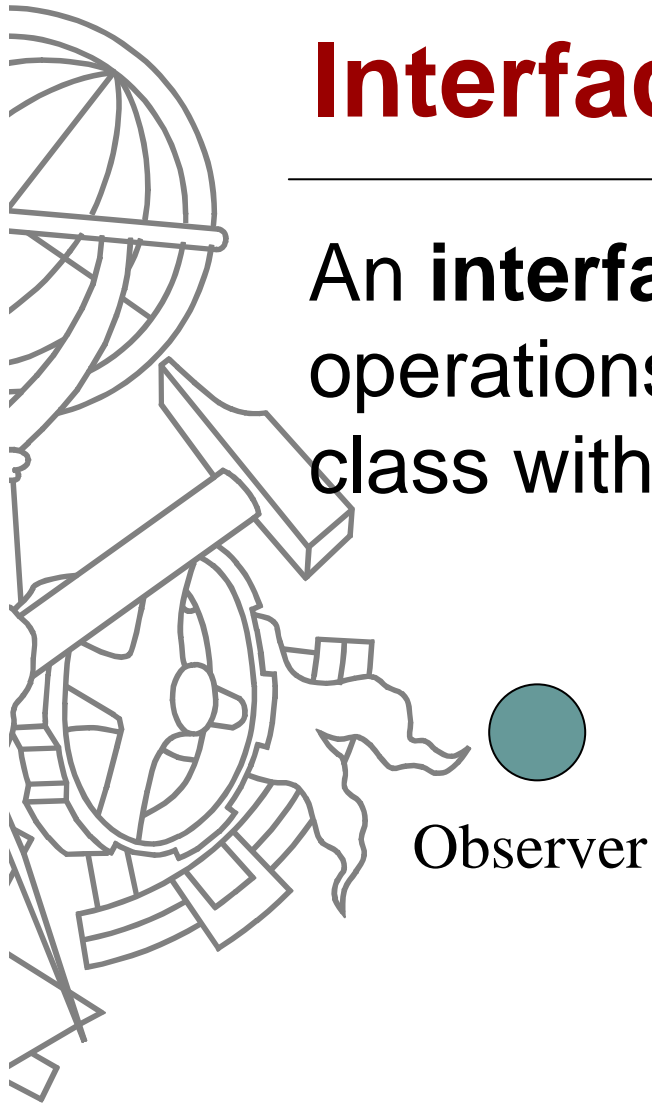
Association Classes

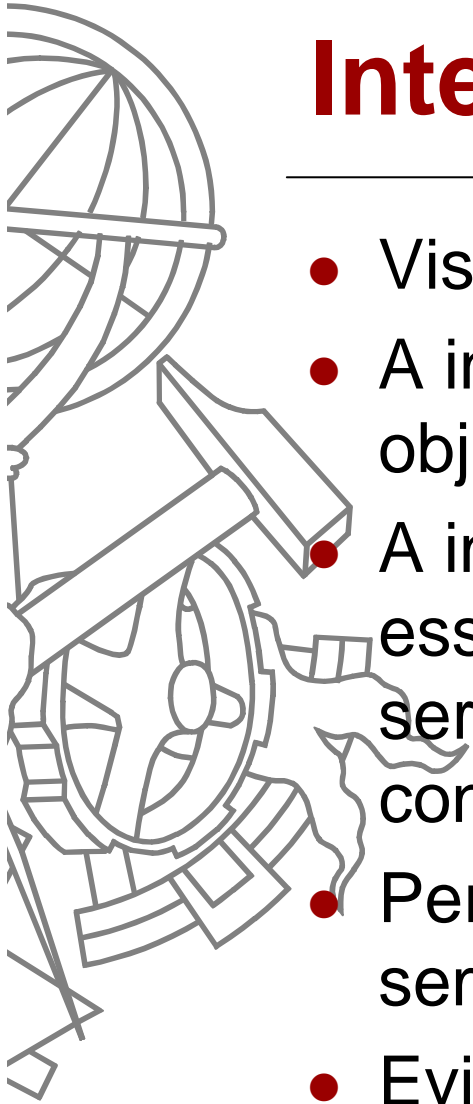
An **association class** has properties of both an association and a class.



Interfaces

An **interface** is a named collection of operations used to specify a service of a class without dictating its implementation.





Interface (II)

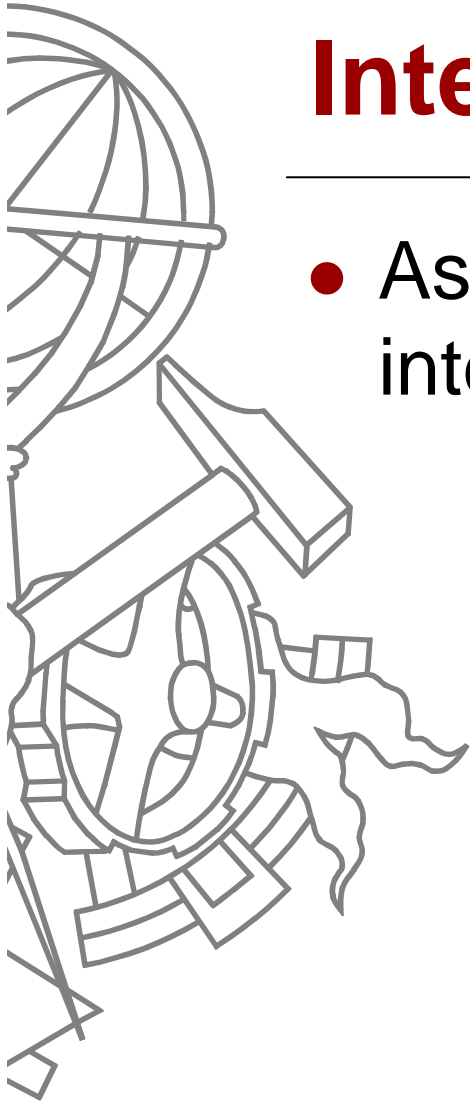
- Visão como serviços
- A interface descreve o serviço fornecido pelo objecto.
- A interface funciona como um contracto para esse serviço: se o serviço é fornecido, então será fornecido de acordo com o descrito no contracto
- Permite a captura de semelhança entre classes sem impor relações artificiais entre elas
- Evidencia o serviço prestado sem forçar qualquer tipo de implementação

Slide 64

GECAD1

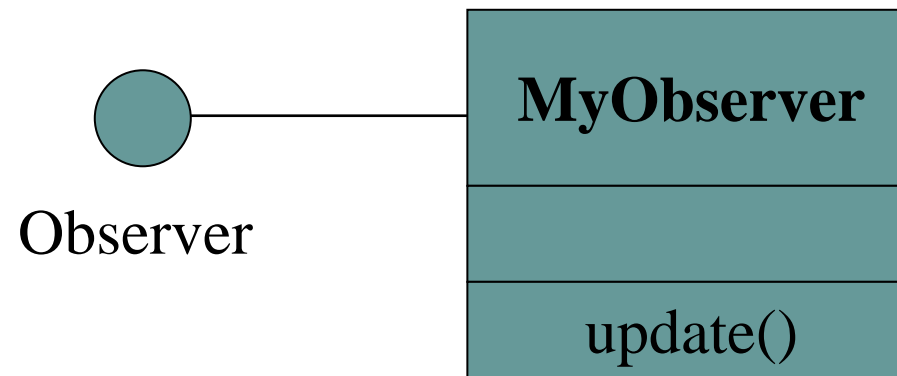
figura

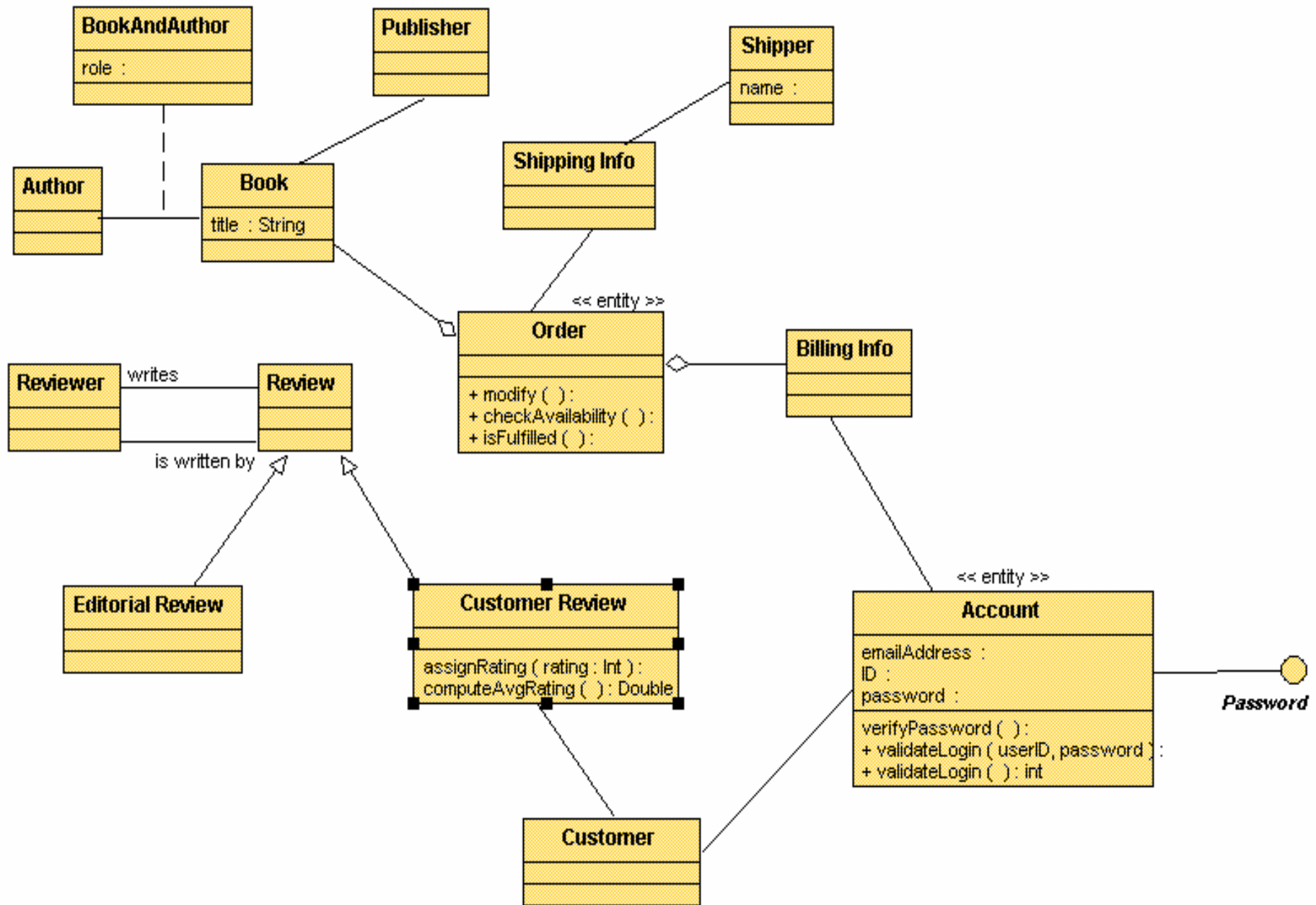
GECAD; 06-12-2002



Interface (III)

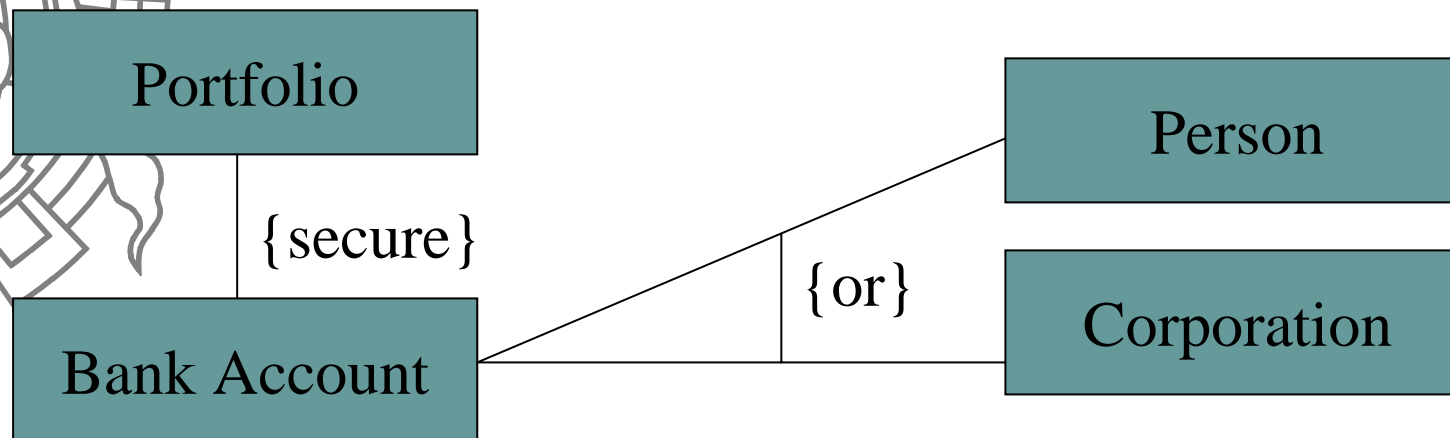
- As classes podem “realizar” uma ou mais interfaces





Constraints

A **constraint** is an extension of the semantics of one or more model elements which specifies a condition that must be true.





Exercício

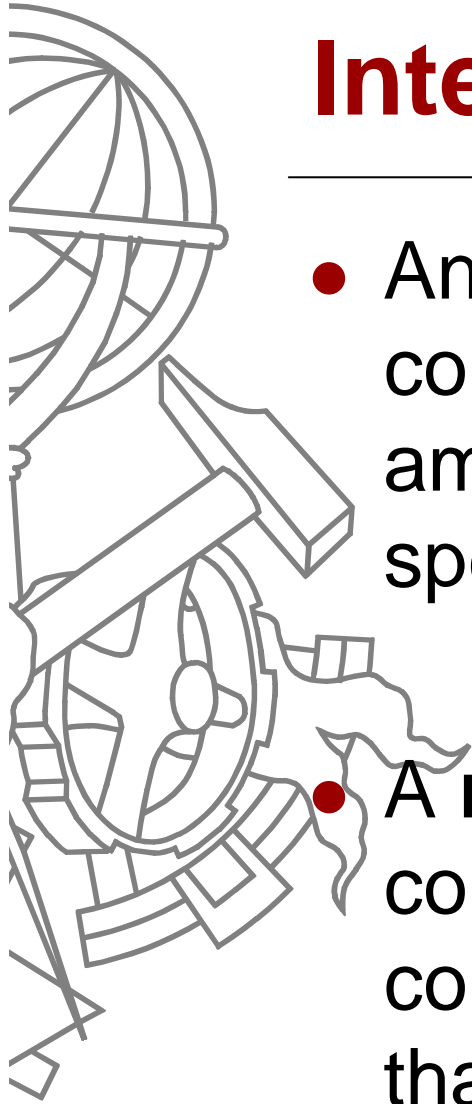
- Continuação do portal intranet de uma empresa
- Elaborar um diagrama de classes para o cenário que descreveram nos diagramas de use case



Diagramas Comportamentais



Diagrama de Sequência



Interaction and Message

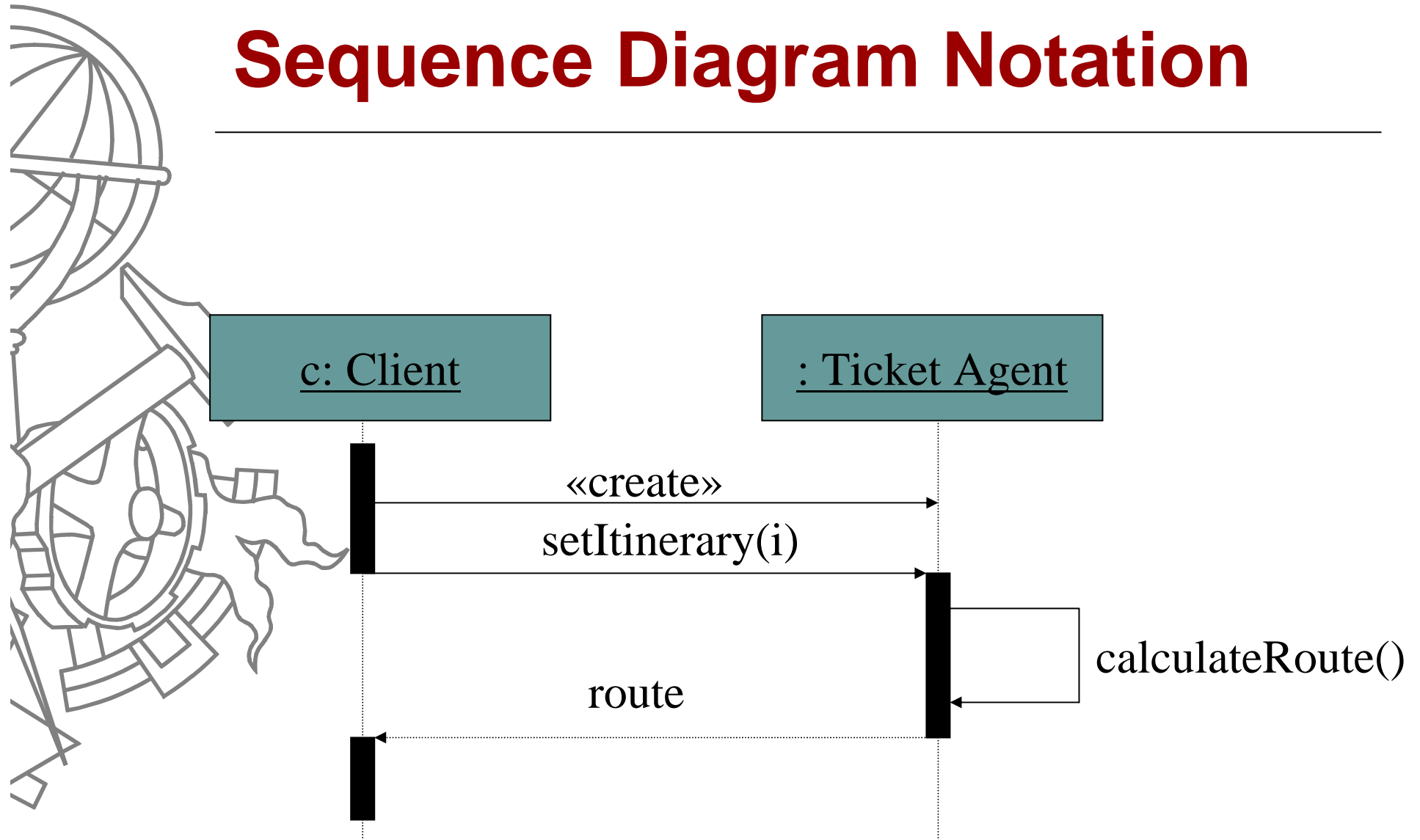
- An **interaction** is a behavior that comprises a set of messages, exchanged among a set of objects, to accomplish a specific purpose.
- A **message** is the specification of a communication between objects that conveys information, with the expectation that some kind of activity will ensue.

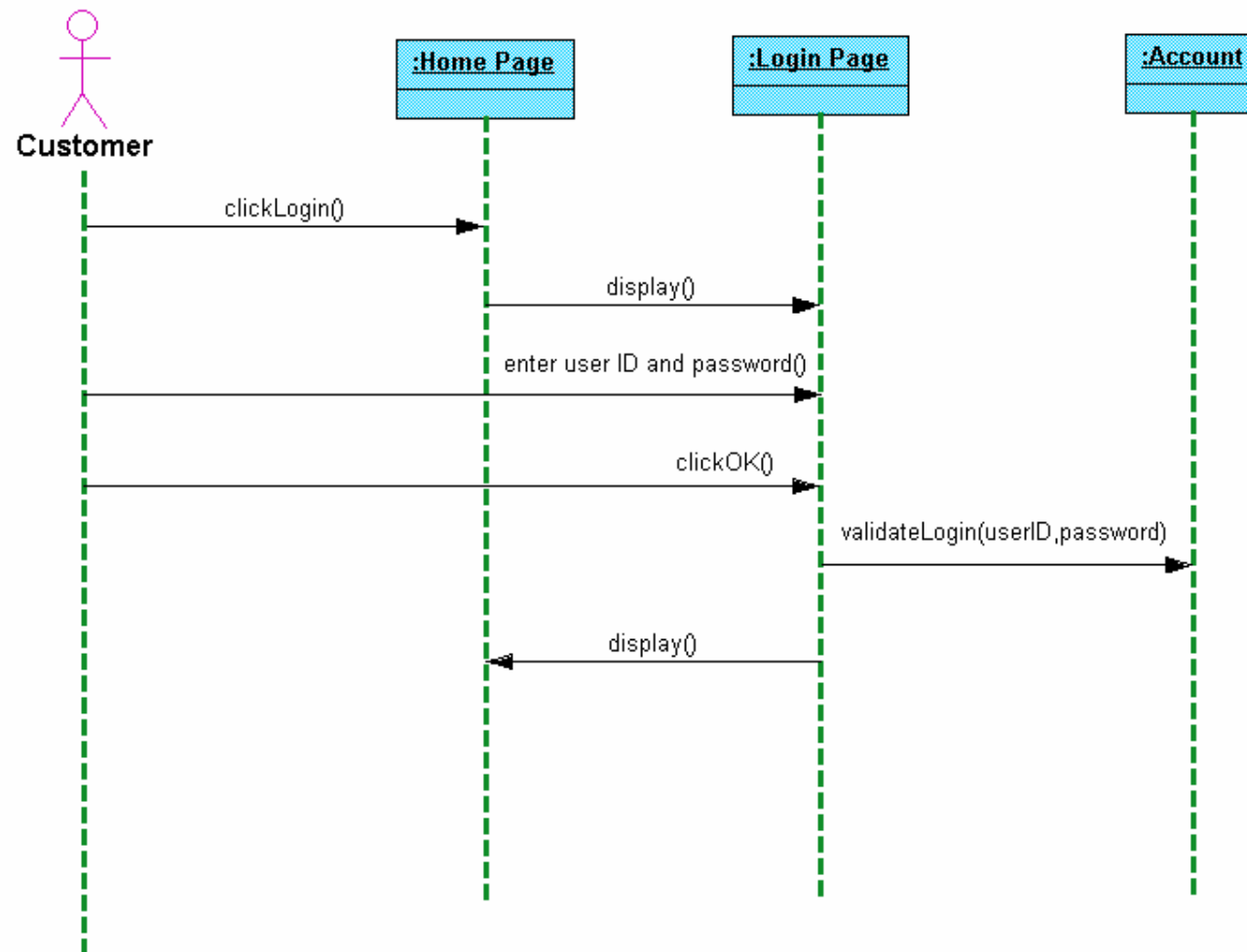


Sequence Diagram

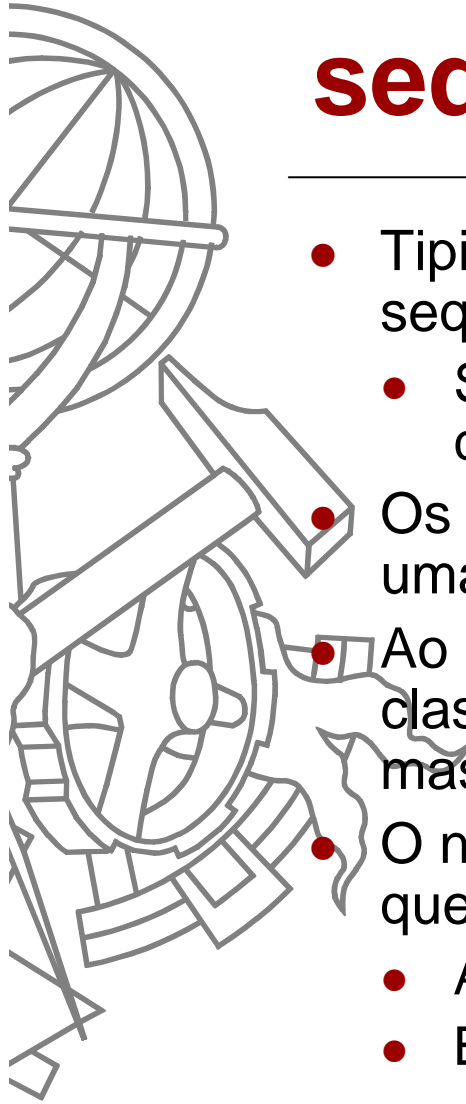
- A **sequence diagram** is an interaction diagram that emphasizes the time ordering of messages.
- A **lifeline** is a vertical dashed line that represents the lifetime of an object.
- A **focus of control** is a tall, thin rectangle that shows the period of time during which an object is performing an action.

Sequence Diagram Notation



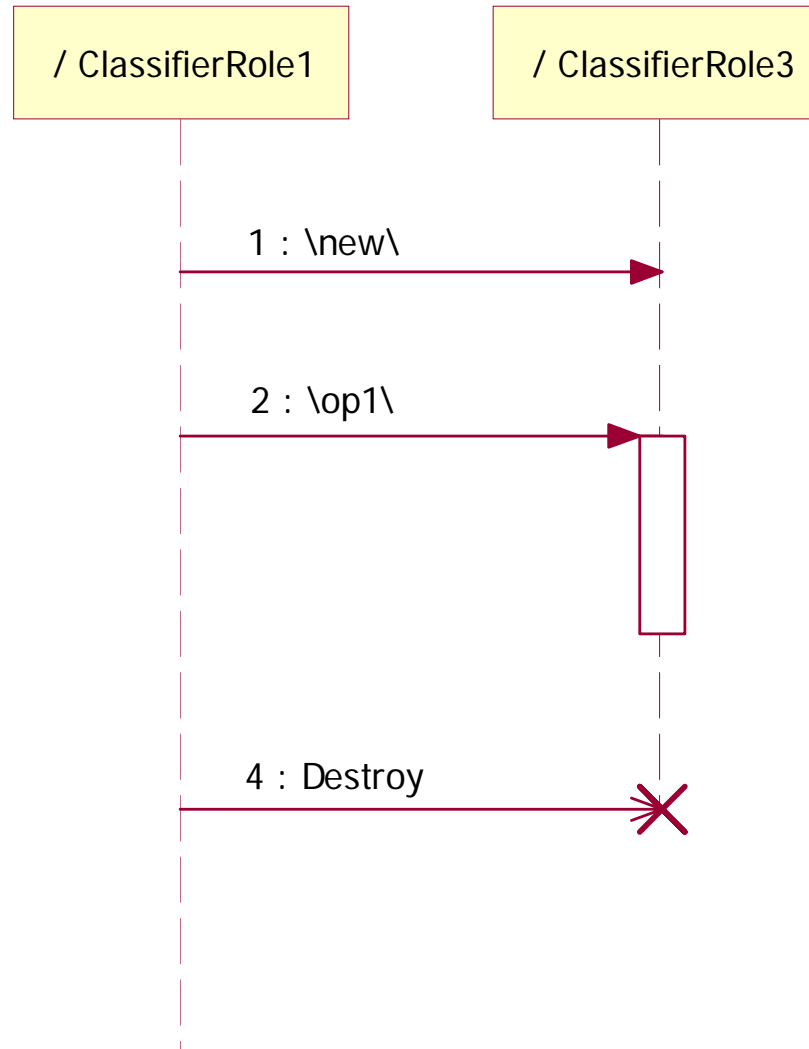


Utilização de diagramas de sequência

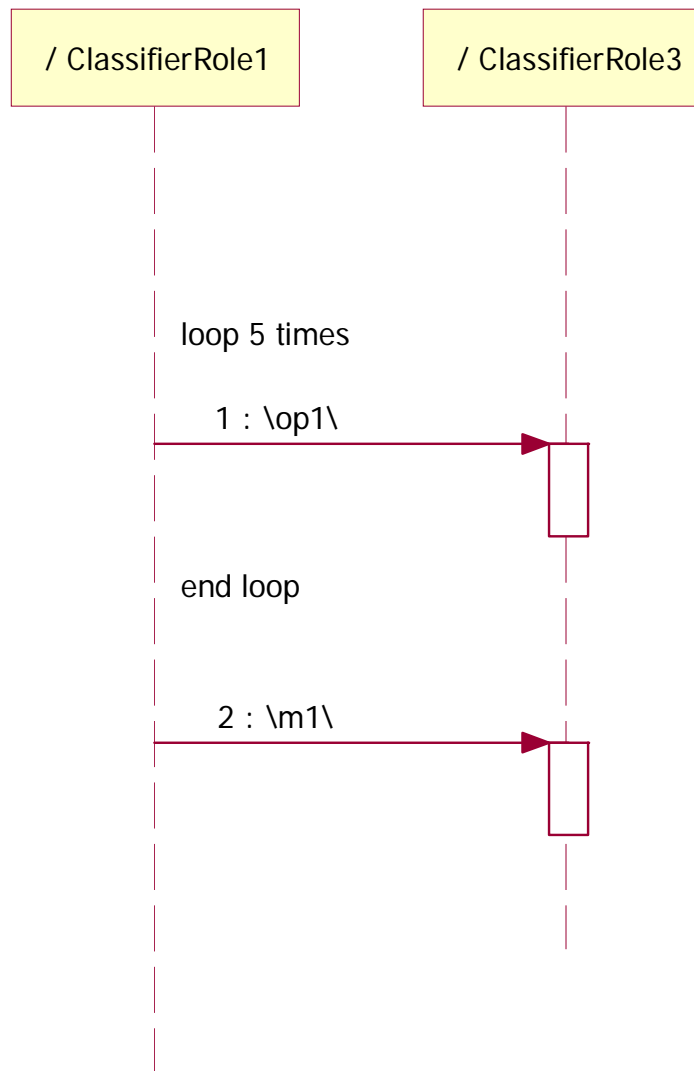


- Tipicamente cada use case é descrito por um diagrama de sequência de alto nível
 - São muitas vezes sub-diagramas associados a um use case
- Os focos de controlo são provocados pela recepção de uma mensagem
- Ao elaborar o diagrama de sequência utilizam-se as classes identificadas previamente no diagrama de classes mas também é normal criar novas classes
- O nível de detalhe do diagrama de sequência depende do que se quer transmitir.
 - Alto nível para compreensão geral do processo
 - Baixo nível para implementação

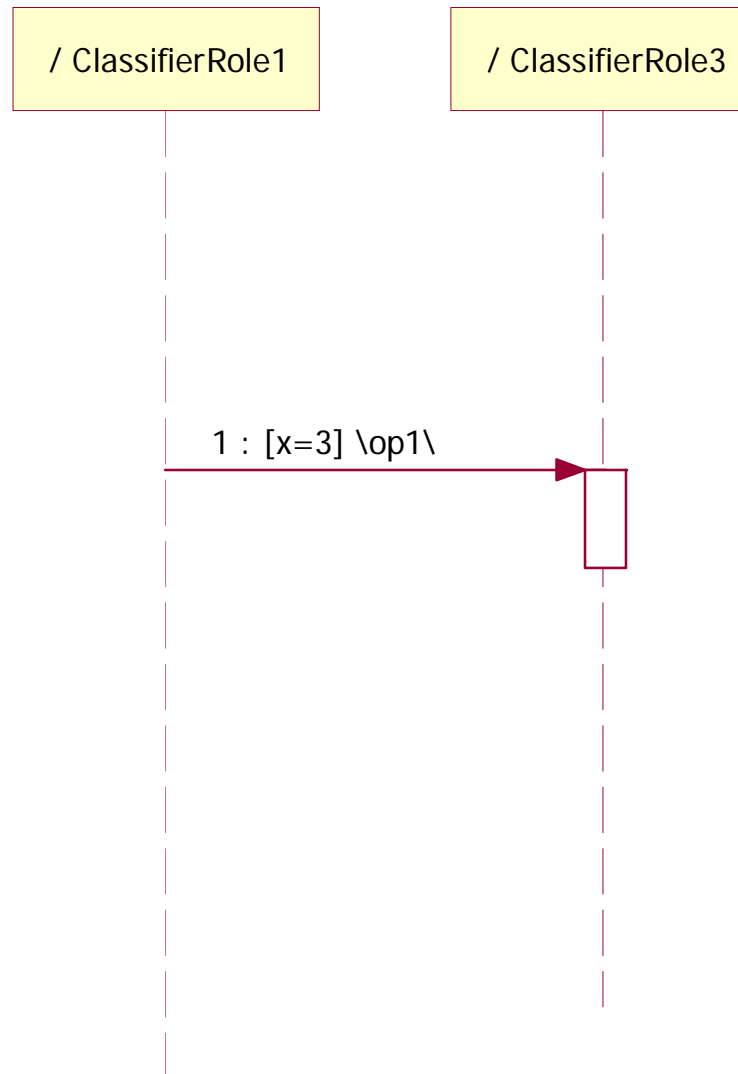
Criação e destruição



Ciclos e decisões (UML 1.x)



Ciclos e decisões (UML 1.x)



Exercício

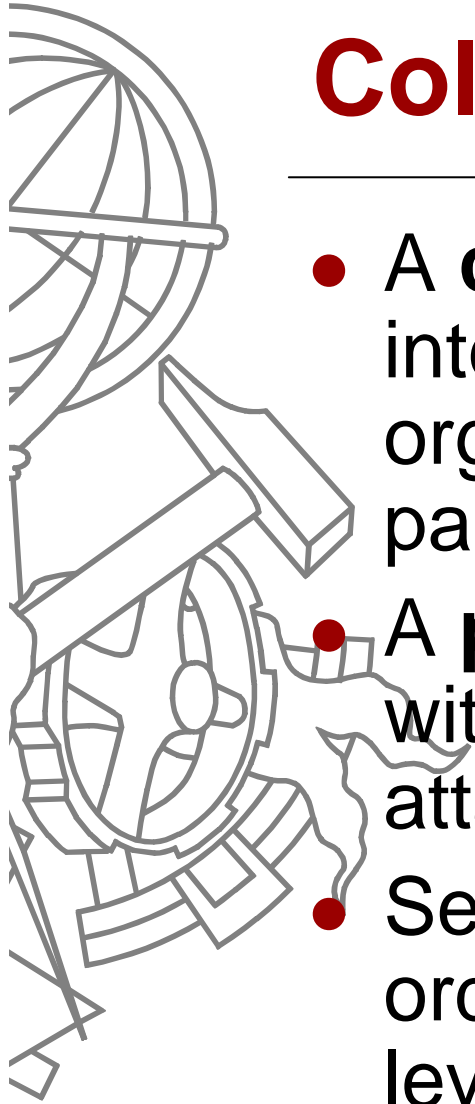
- Continuação do portal intranet de uma empresa
- Elaborar os diagramas de sequência para cada use case identificado



Diagramas Comportamentais



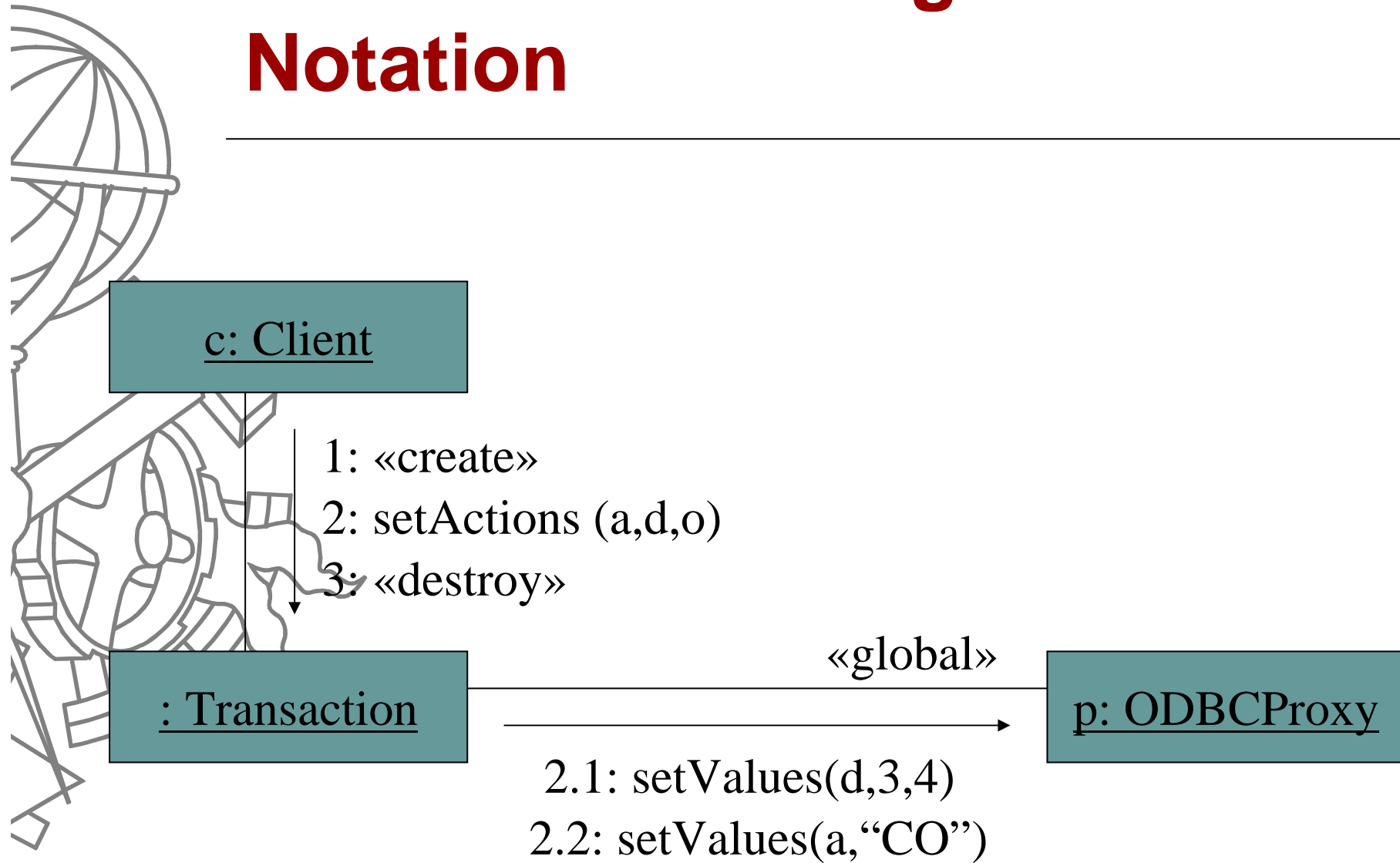
Diagrama de Colaboração

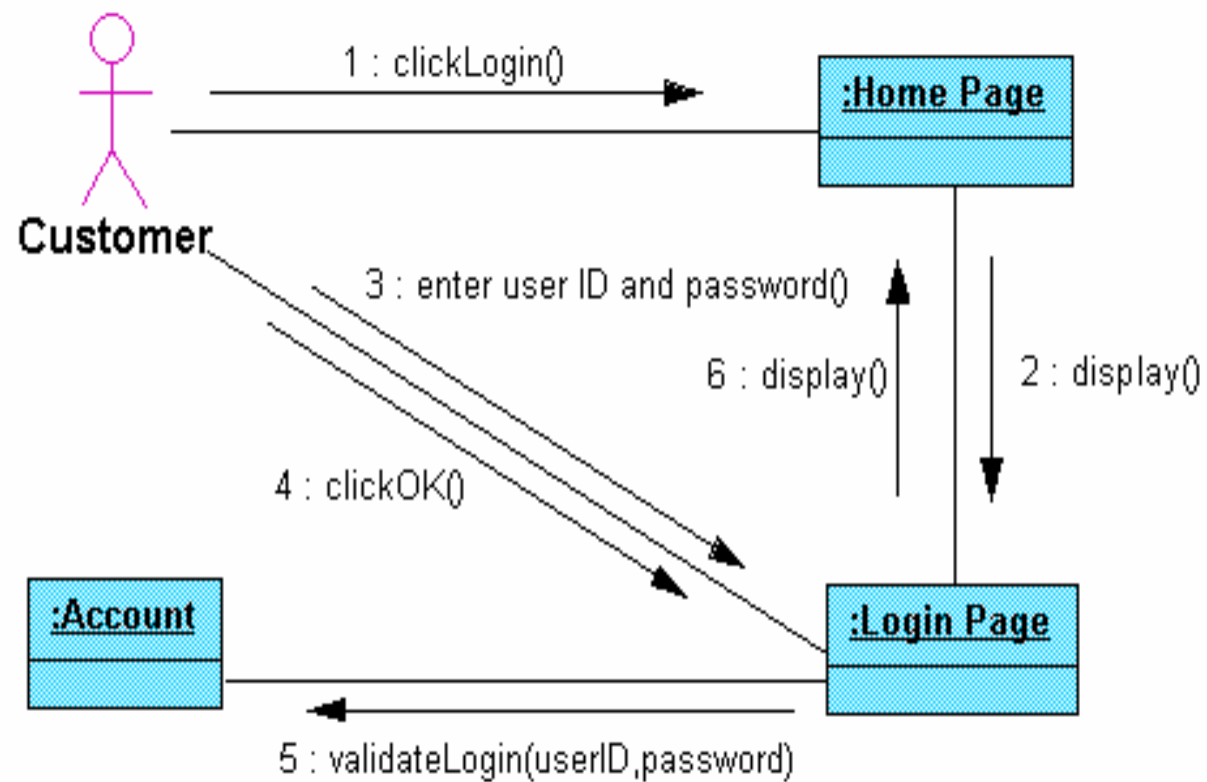


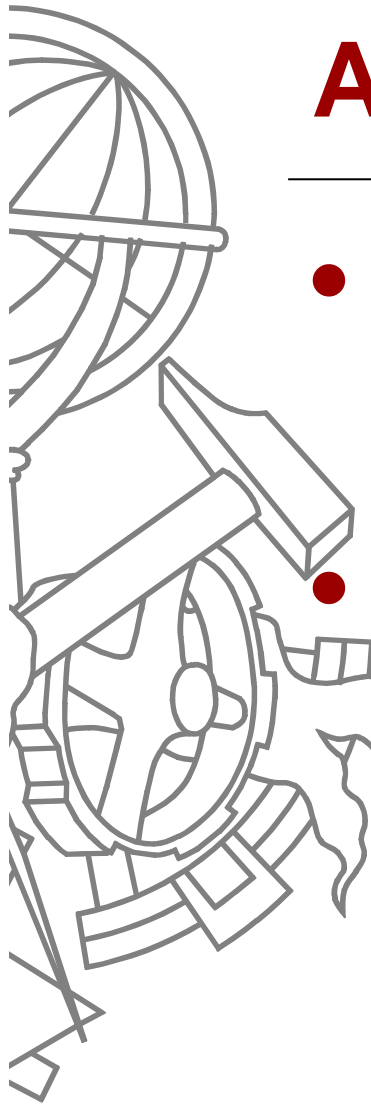
Collaboration Diagram

- A **collaboration diagram** is an interaction diagram that emphasizes the organization of the objects that participate in the interaction.
- A **path** is a link between objects, perhaps with a stereotype such as «local» attached.
- Sequence numbers indicate the time ordering of messages, to one or more levels.

Collaboration Diagram Notation







Algumas notas

- É possível gerar um diagrama de sequência e a partir de um diagrama de colaboração e vice-versa
- Ambos apresentam a mesma informação
 - Sequência: dá ênfase ao aspecto temporal das mensagens
 - Colaboração: dá ênfase à ligação existente (topologia) entre as classes

Exercício

- Continuação do portal intranet de uma empresa
- Elaborar os diagramas colaboração correspondentes aos diagramas de sequência elaborados no passo anterior





Diagramas Comportamentais

Diagrama de Estados



State, Event, and Signal

- A **state** is a condition in which an object can reside during its lifetime while it satisfies some condition, performs an activity, or waits for an event.
- An **event** is a significant occurrence that has a location in time and space.
- A **signal** is an asynchronous communication from one object to another.

State Notation



Idle

Cooling

Heating

Activating

Active



State Machine and Transition

- A **state machine** is a behavior that specifies the sequences of states that an object goes through in its lifetime, in response to events, and also its responses to those events.
- A **transition** is a relationship between two states; it indicates that an object in the first state will perform certain actions, then enter the second state when a given event occurs.



Entry and Exit Actions

- An **entry action** is the first thing that occurs each time an object enters a particular state.
- An **exit action** is the last thing that occurs each time an object leaves a particular state.

Tracking

entry/setMode(onTrack)
exit/setMode(offTrack)



Activities

- An **activity** is an interruptible sequence of actions that an object can perform while it resides in a given state. (Actions are not interruptible.)

Tracking

do/followTarget

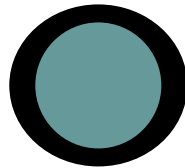


Initial and Final States

The **initial state** is the default starting place for a state machine.



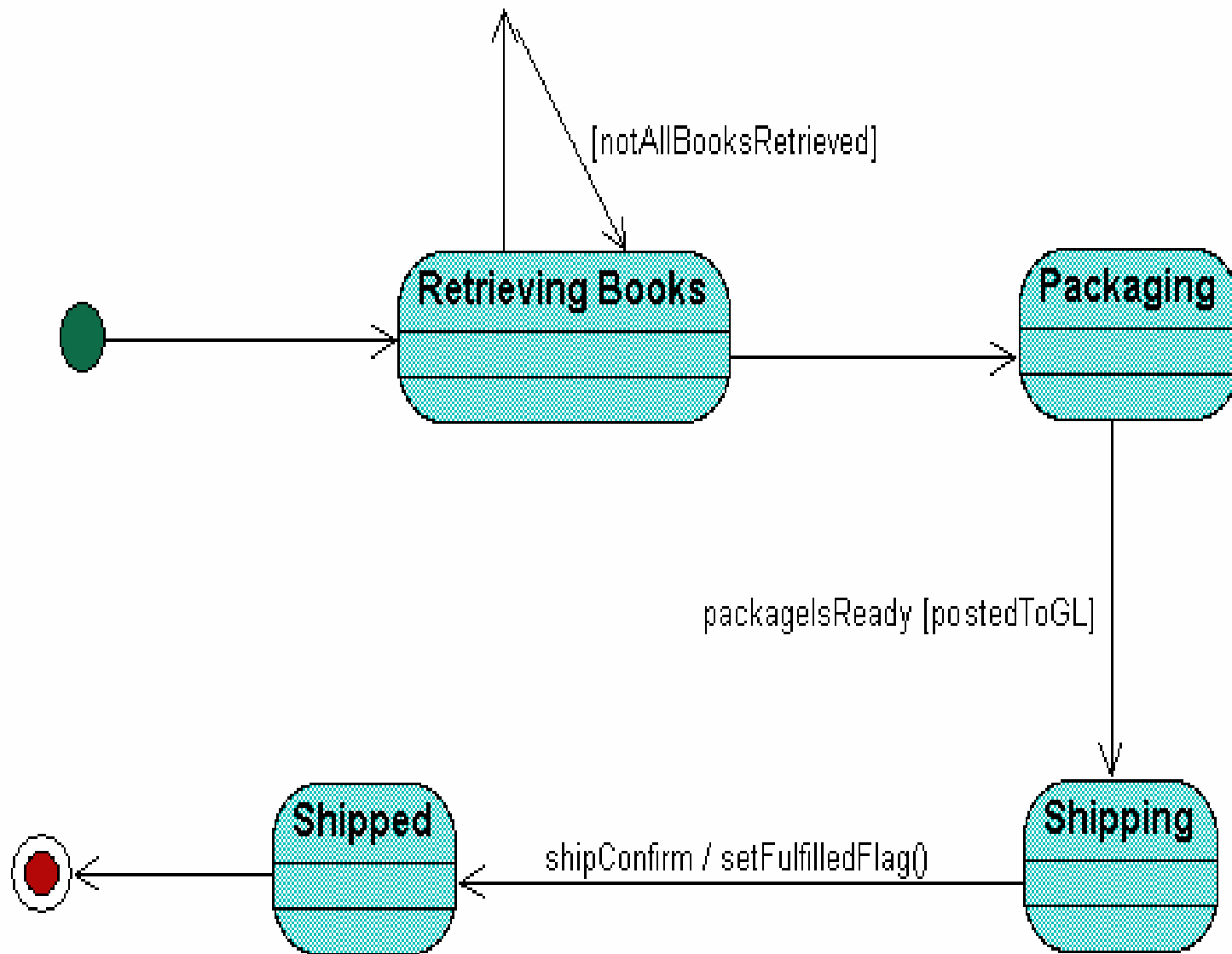
The **final state** indicates the completion of the state machine's execution.





Transições

- Condições de guarda
 - [$x < 5$]
- Eventos
 - Cancel key pressed
- Acções
 - / handleCancelKey()
- Combinação dos três anteriores
 - [$x < 5$] / $x = 5$
 - Cancel key presses / handleCancelKey()
 - [$x < 5$] Cancel key pressed
 - [allowCancel = true] Cancel key pressed / handleCancelKey()





Algumas notas

- Apenas se modela o estado dos objectos/classes pertinentes
- São normalmente sub-diagramas associados à classe em questão

Exercício

- Continuação do portal intranet de uma empresa
- Elaborar o(s) diagrama(s) de estado das principais entidades identificadas no diagrama de classes





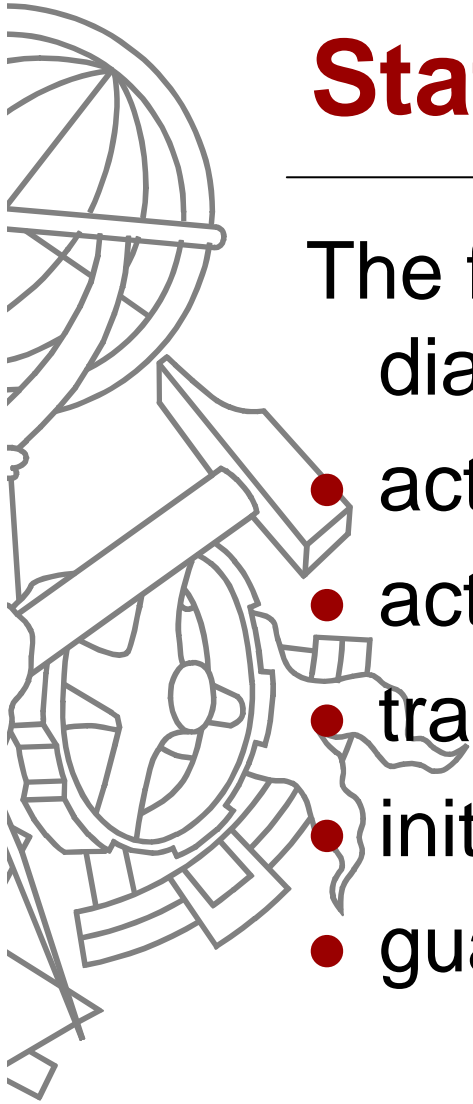
Diagramas Comportamentais

Diagrama de Actividades



Why Activity Diagrams?

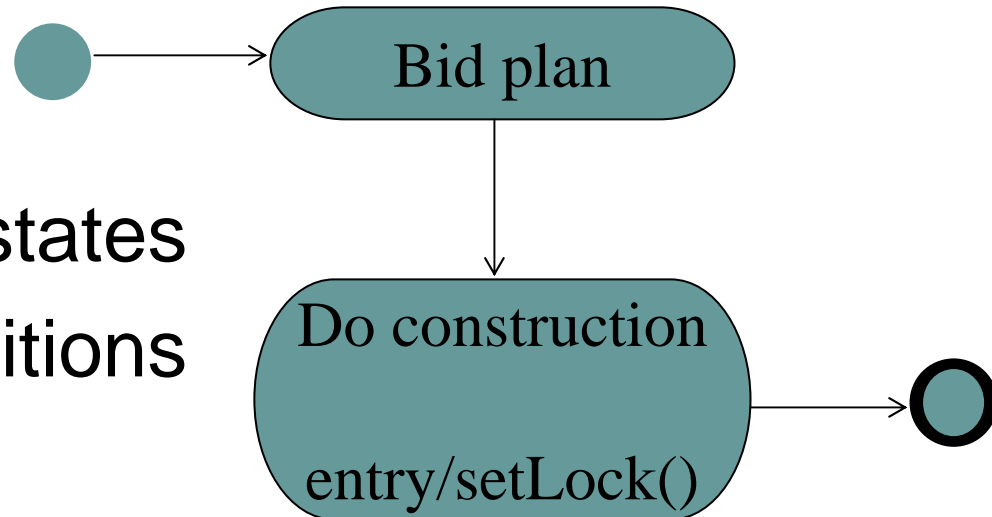
- An **activity diagram**, which resembles a flowchart, is useful for modeling workflows and the details of operations.
- While an interaction diagram looks at the objects that pass messages, an activity diagram looks at the operations that are called by objects.



State Diagram Carryovers

The following items are common to state diagrams and activity diagrams:

- activities
- actions
- transitions
- initial/final states
- guard conditions





Breaking Up Flows

- alternate paths:

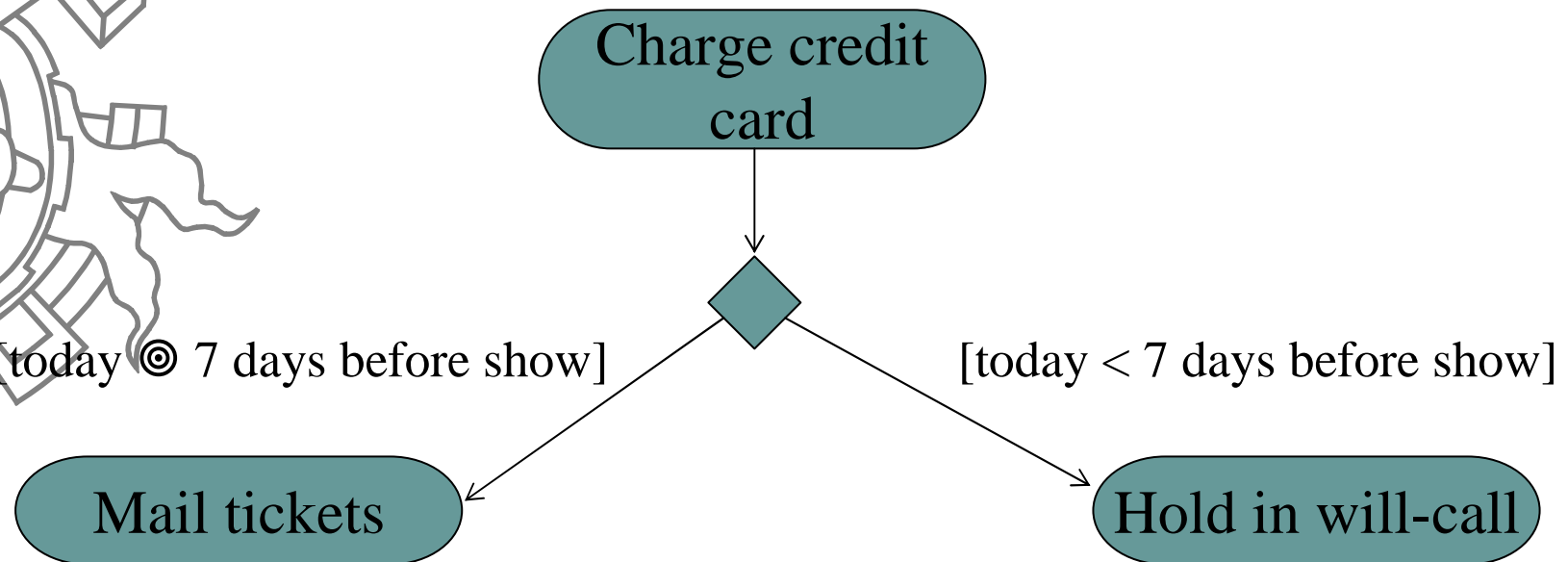
- branch
- merge

- parallel flows:

- fork
- join

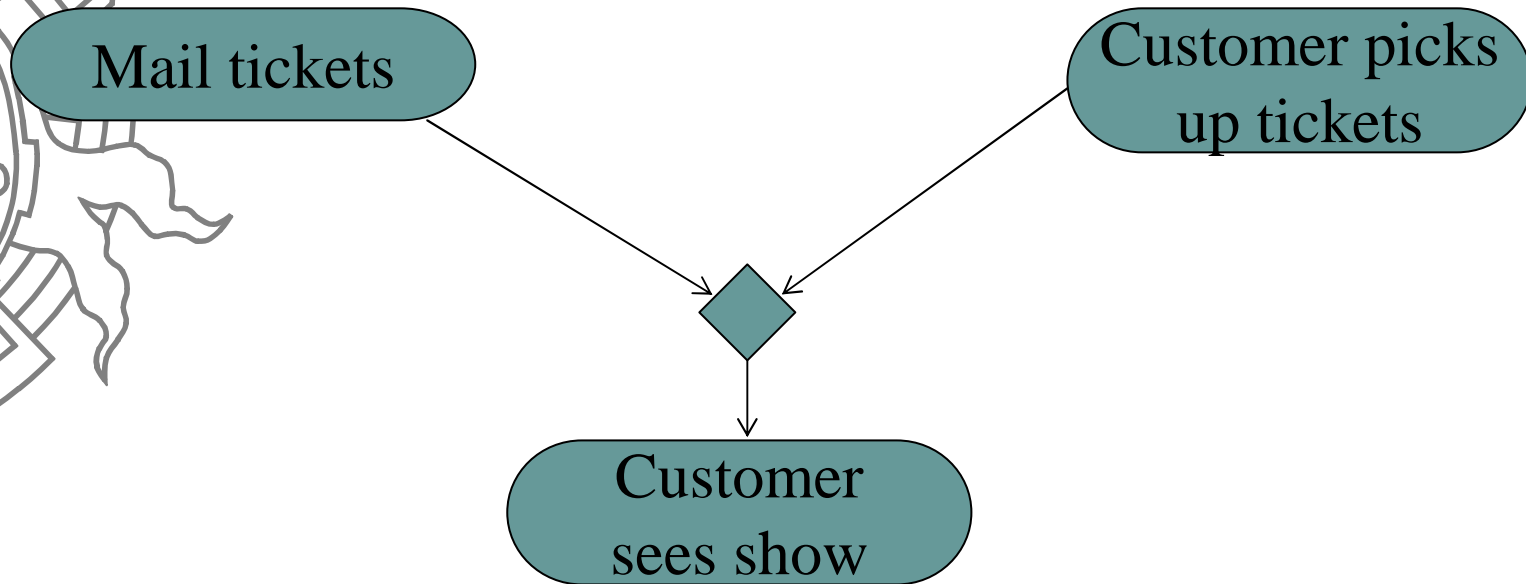
Branching

A **branch** has one incoming transition and two or more outgoing transitions:



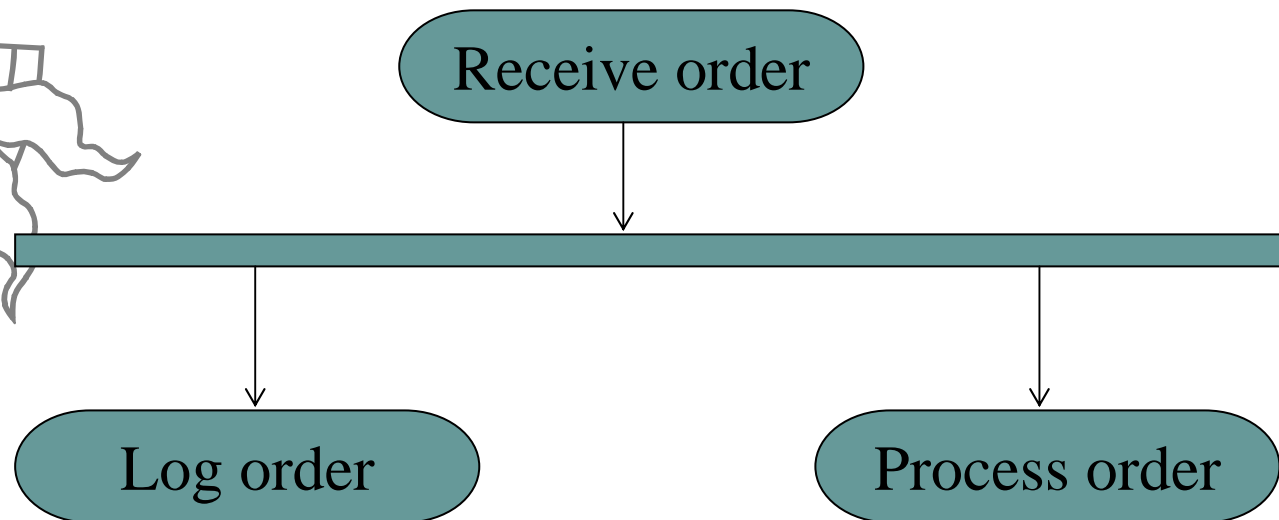
Merging

A **merge** has two or more incoming transitions and one outgoing transition:



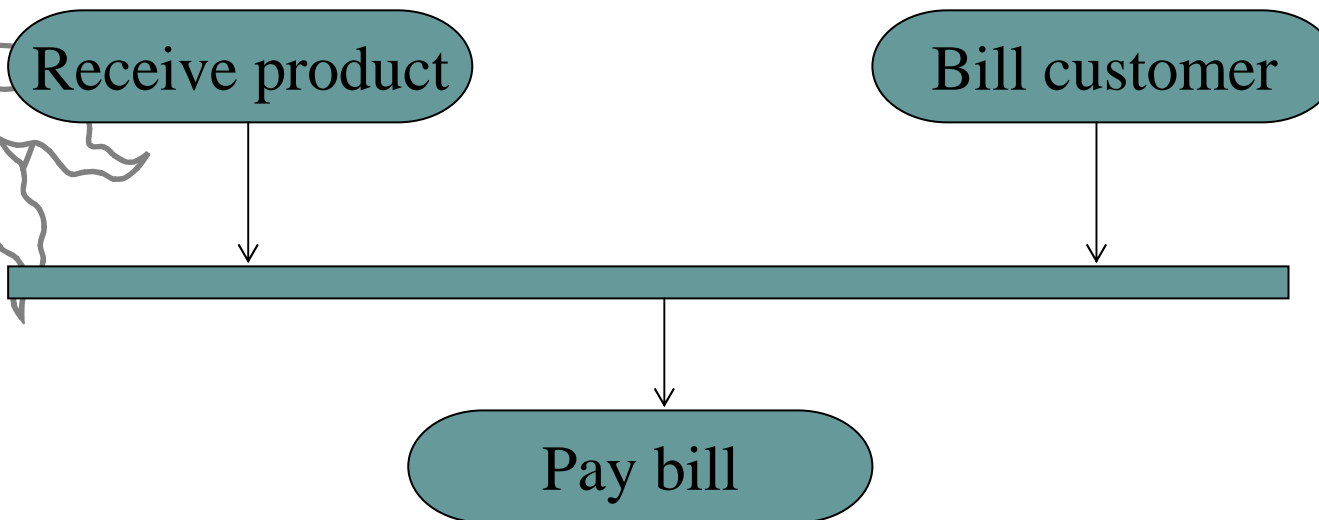
Forking

A **fork** represents the splitting of a single flow of control into two or more concurrent flows of control:



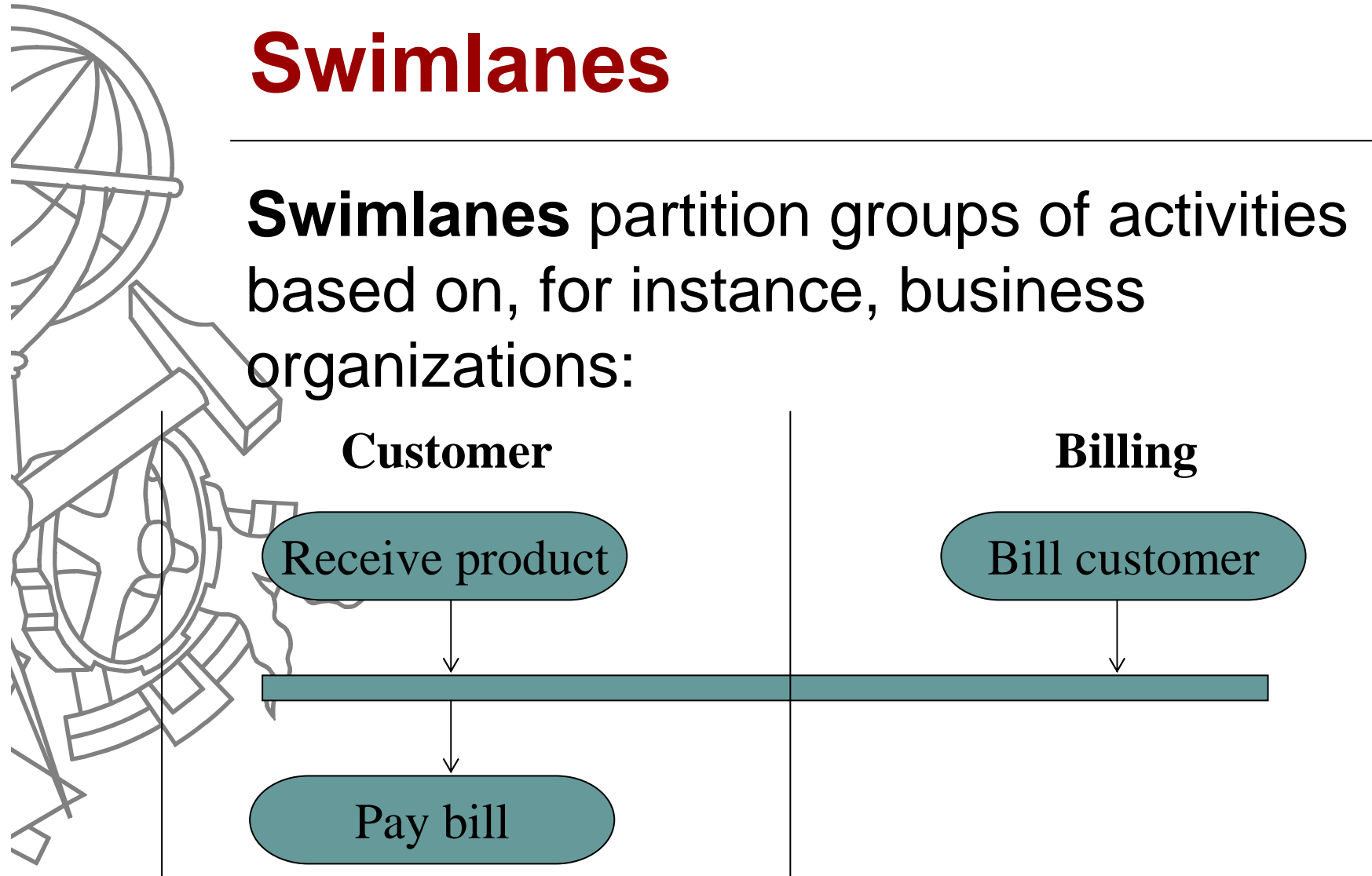
Joining

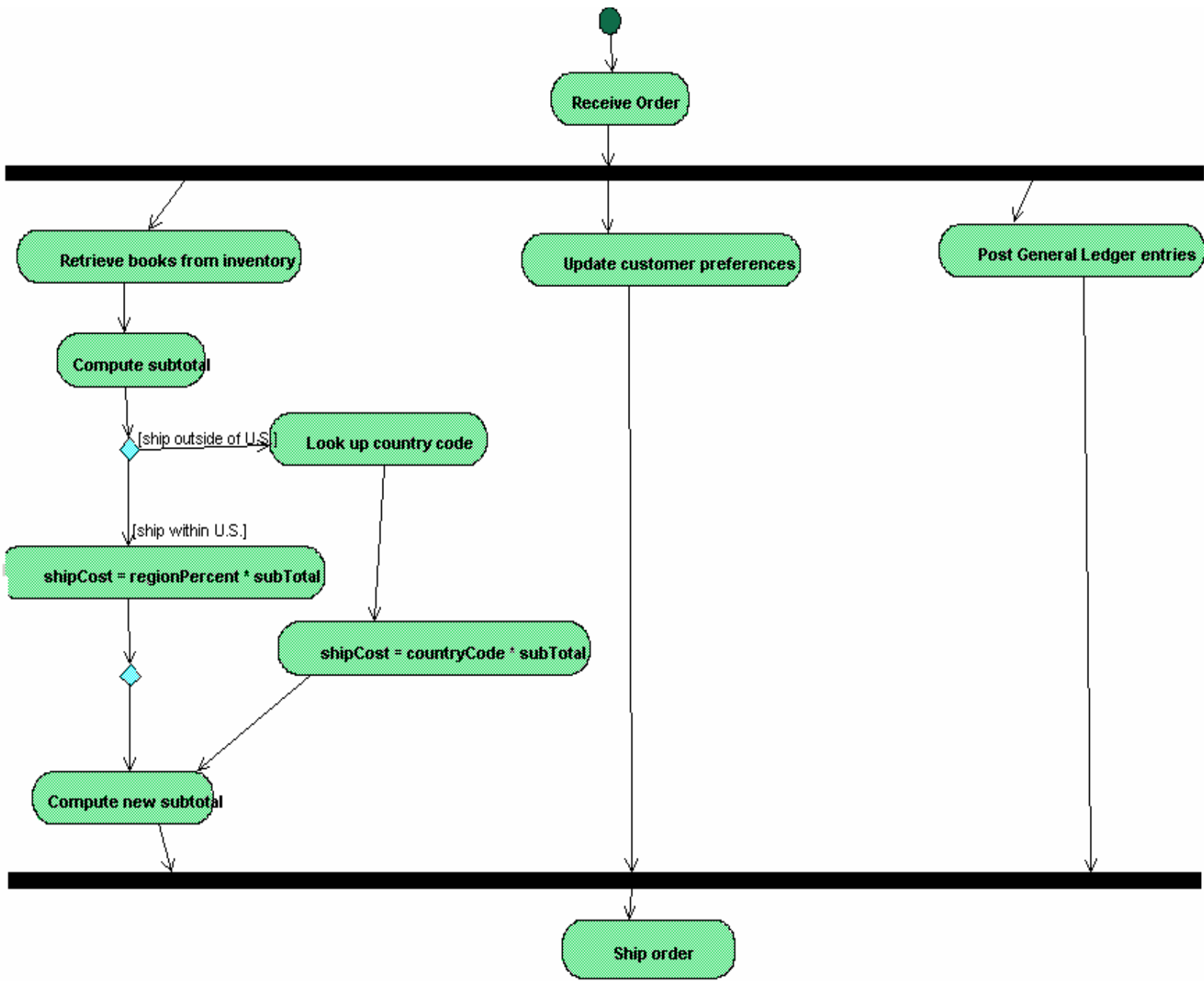
A **join** represents the synchronization of two or more flows of control into one sequential flow of control:



Swimlanes

Swimlanes partition groups of activities based on, for instance, business organizations:







Algumas notas

- São utilizados para modelar:
 - a lógica de uma operação de uma classe
 - um processo de negócio
- São normalmente sub-diagramas de classes ou de use cases

Exercício

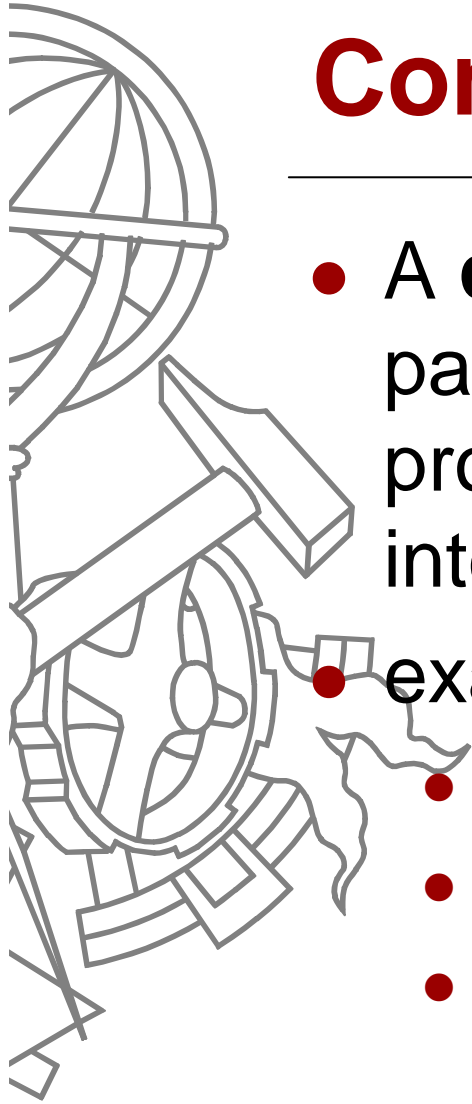
- Continuação do portal intranet de uma empresa
- Elaborar o(s) diagrama(s) de actividades para as principais actividades/processos do sistema





Diagramas Estruturais

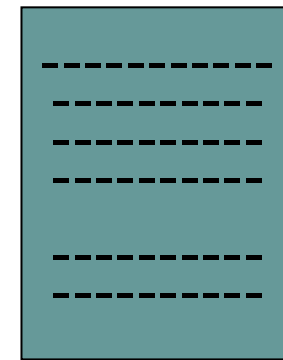
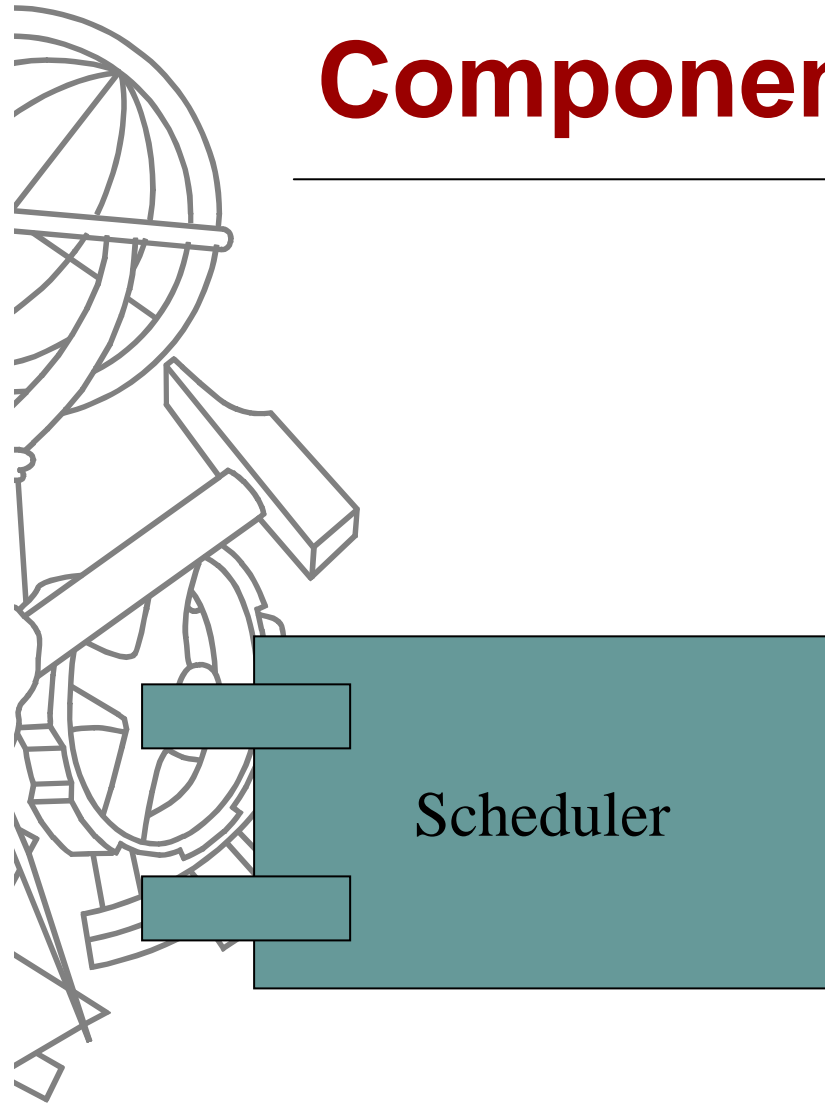
Diagrama de Componentes



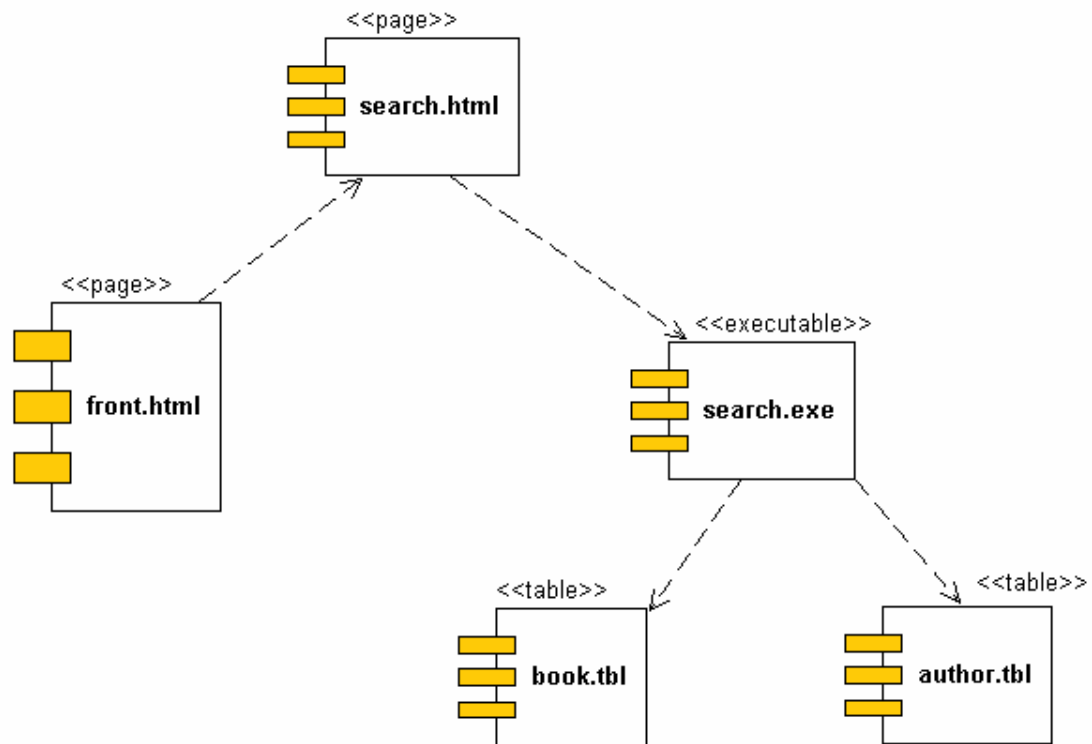
Component

- A **component** is a physical, replaceable part of a system that conforms to, and provides the realization of, a set of interfaces.
- examples:
 - dynamic link library (DLL)
 - COM+ component
 - Enterprise Java Bean (EJB)

Component Notation



signal.cpp



Exercício

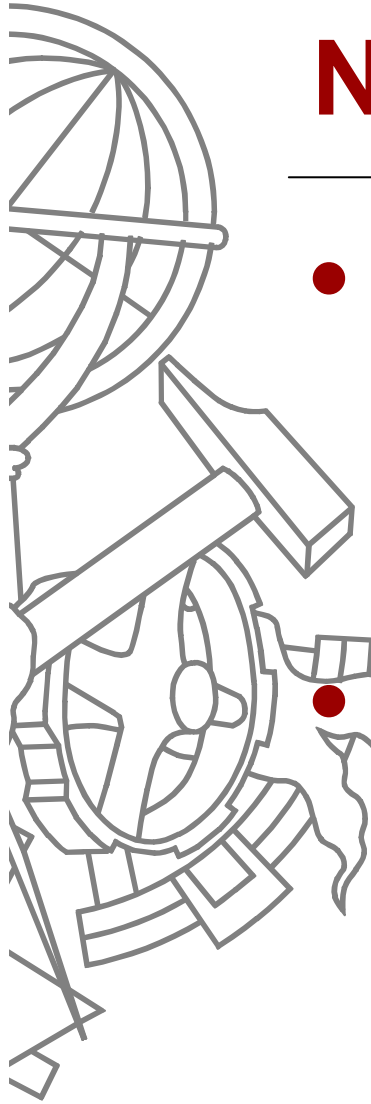
- Continuação do portal intranet de uma empresa
- Elaborar o diagrama de componentes do sistema



Diagramas Estruturais



Diagrama de Instalação (deployment)



Node

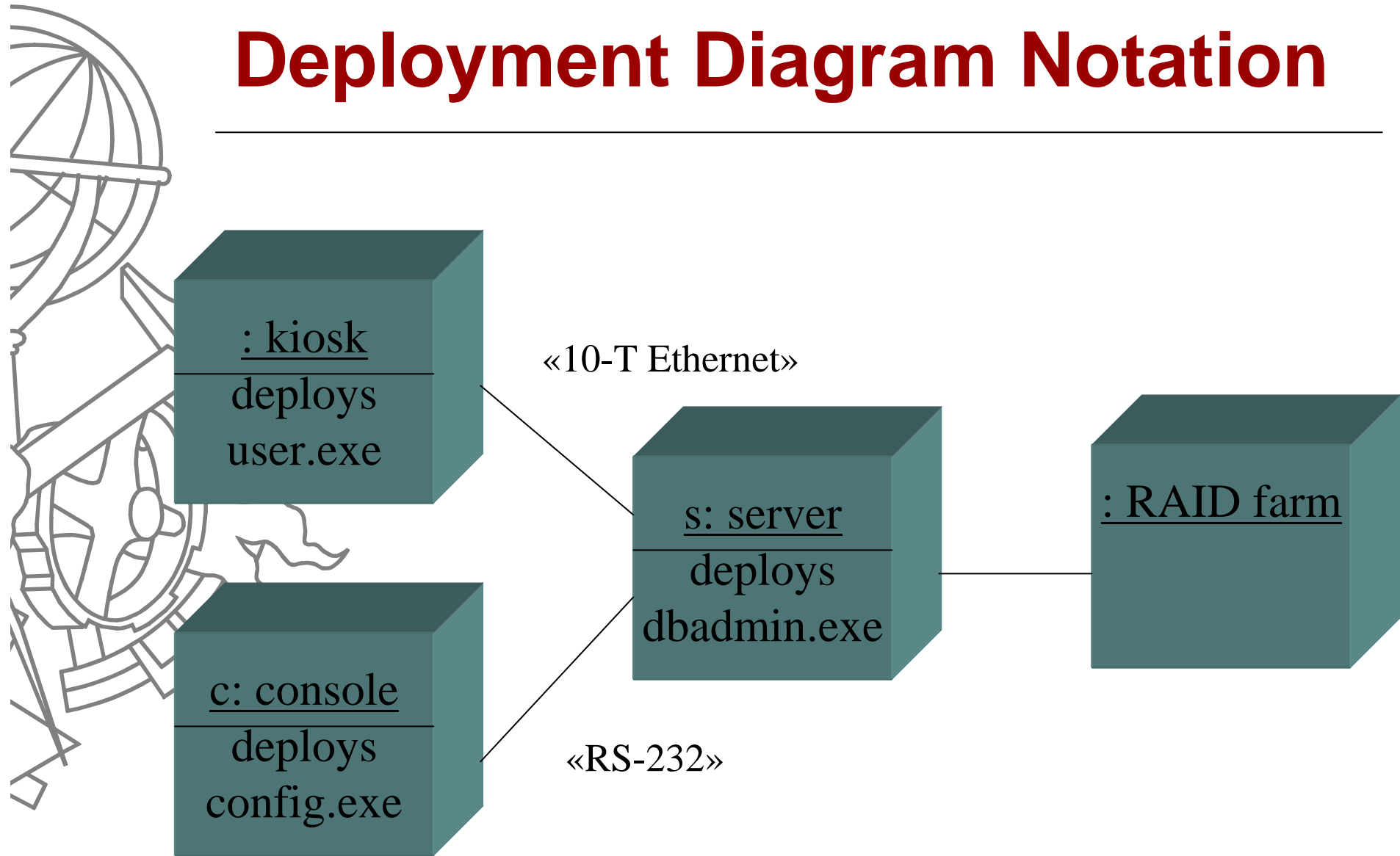
- A **node** is a physical element, which exists at run time, that represents some computation resource.
- This resource generally has at least some memory; it often has processing capability.

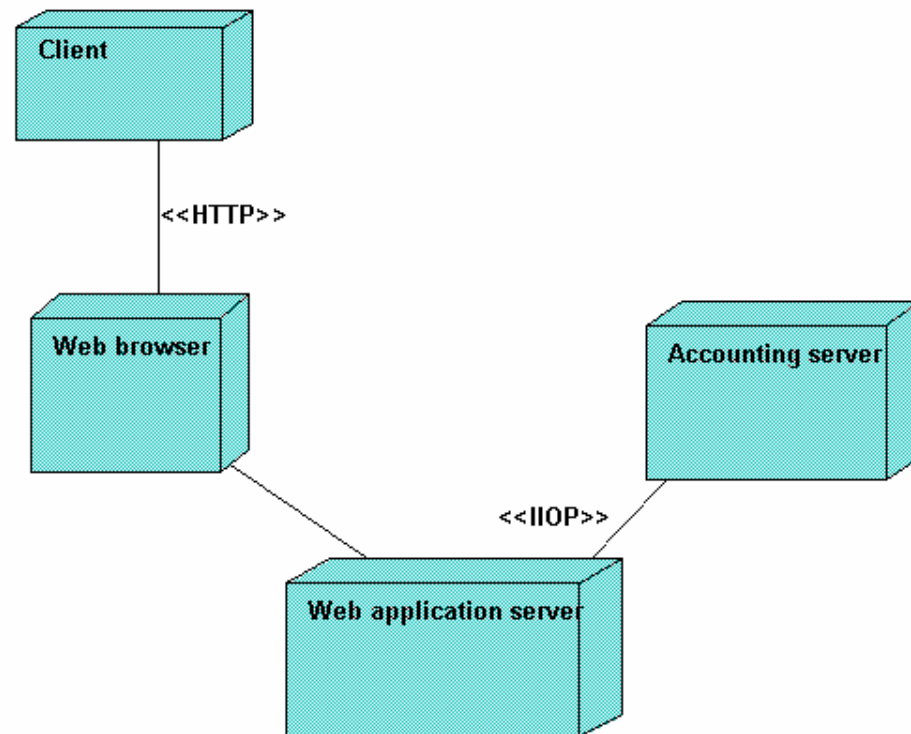


Nodes and Components

- **Components** are things that participate in the execution of a system; nodes are things that execute components.
- A distribution unit is a set of components that have been allocated to a node as a group.

Deployment Diagram Notation







Exercício

- Continuação do portal intranet de uma empresa
- Elaborar um ou mais diagramas de instalação correspondentes a diferentes cenários de instalação da aplicação



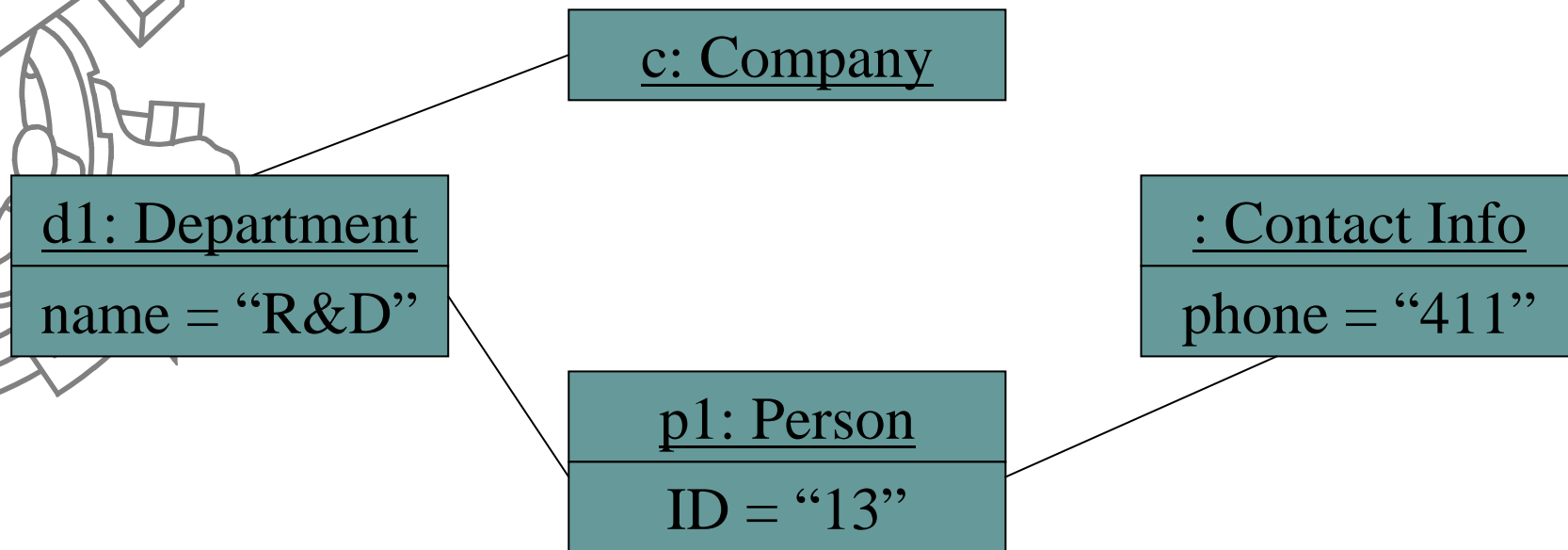


Diagramas Estruturais

Diagrama de Objectos

Object Diagram

An object diagram shows a set of objects, and their relationships, at a specific point in time.





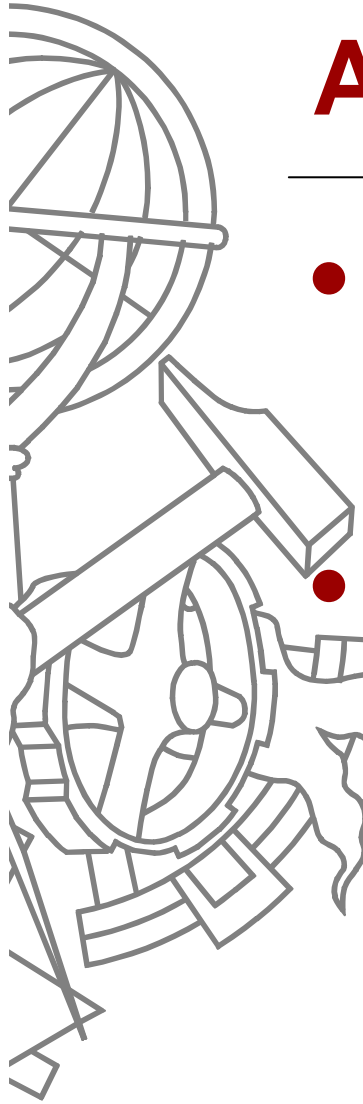
Quando usar

- Permite evidenciar um “fotograma” da execução da aplicação para visualizar o estado interno de cada objecto nesse instante



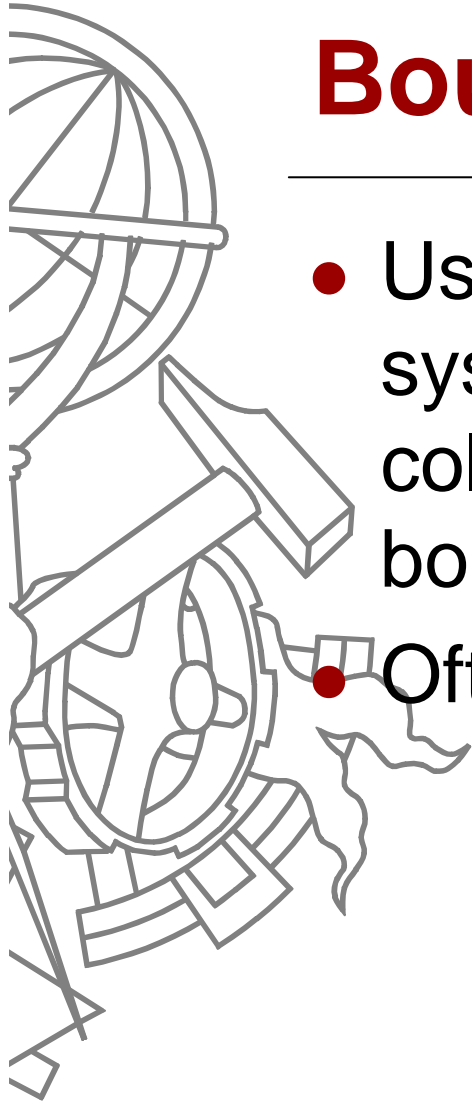
Parte III

Classes de Análise



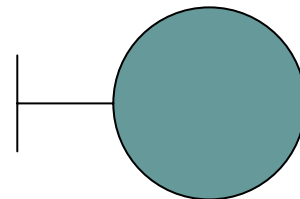
Analysis Classes

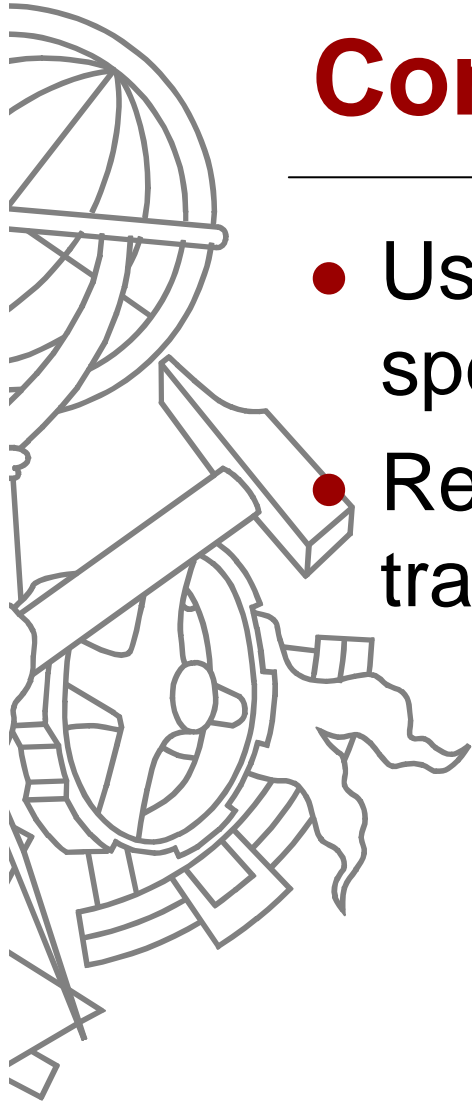
- Introduced by Jacobson in his “white” book (Object-Oriented Software Engineering; Addison-Wesley, 1992)
- Used in Analysis workflow of Rational’s Unified Process
 - Boundary class
 - Control class
 - Entity class



Boundary Class

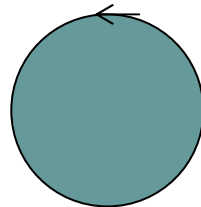
- Used to model interactions between system and its actors and clarify and collect requirements on system's boundaries
- Often represent windows, screens, APIs

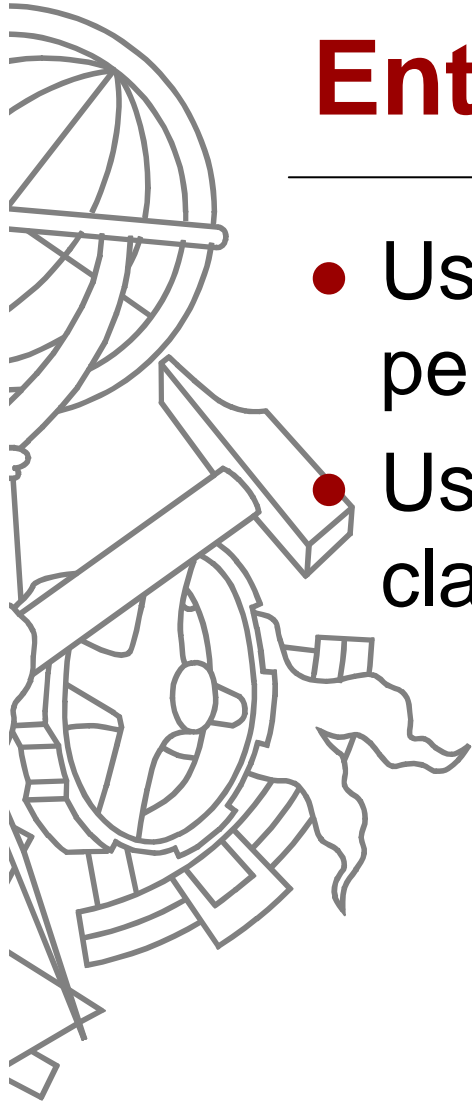




Control Class

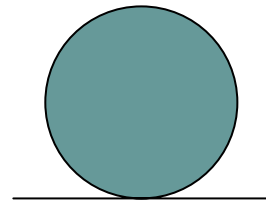
- Used to encapsulate control related to specific use case
- Represent coordination, sequencing, transactions, and control of other objects

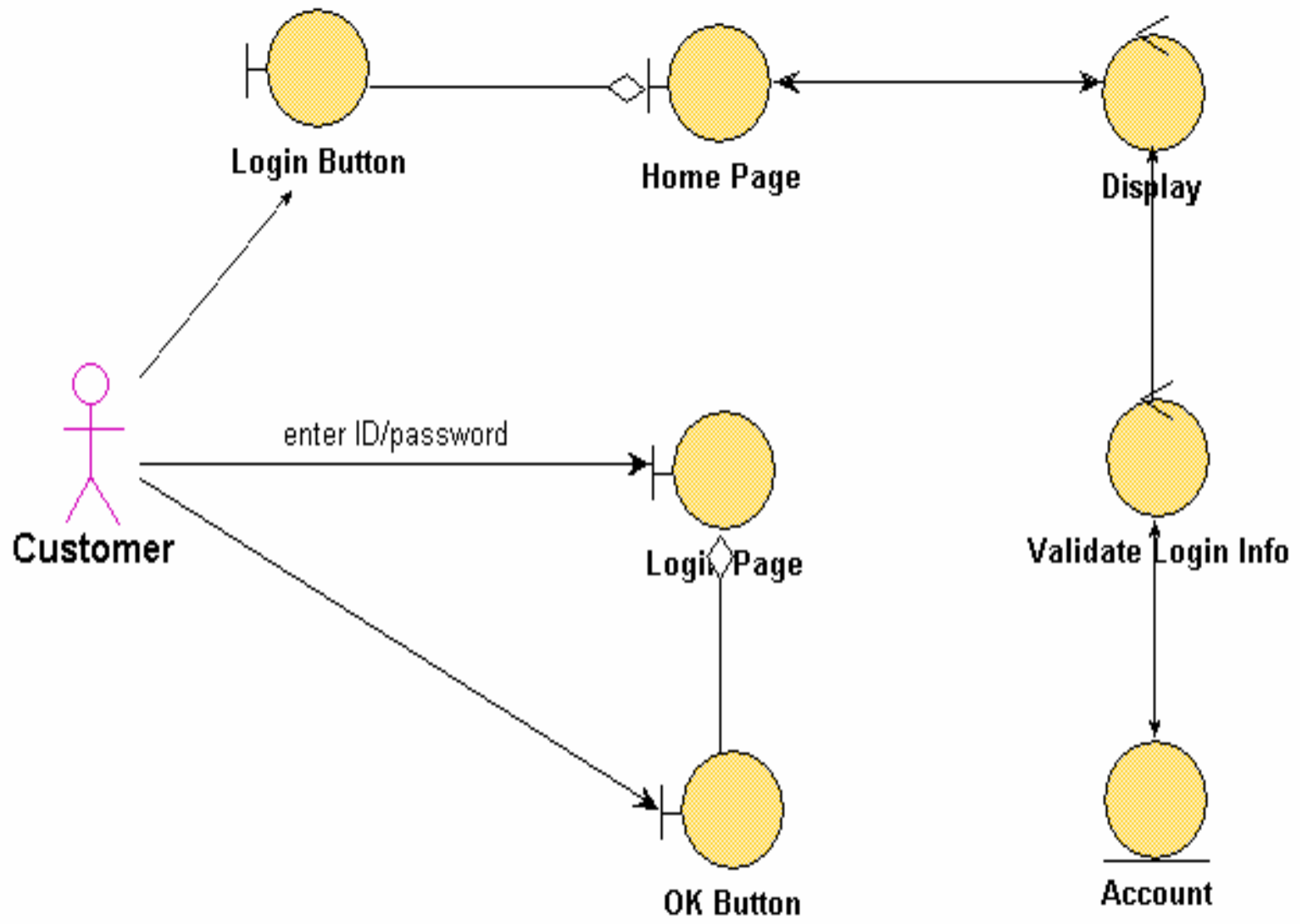


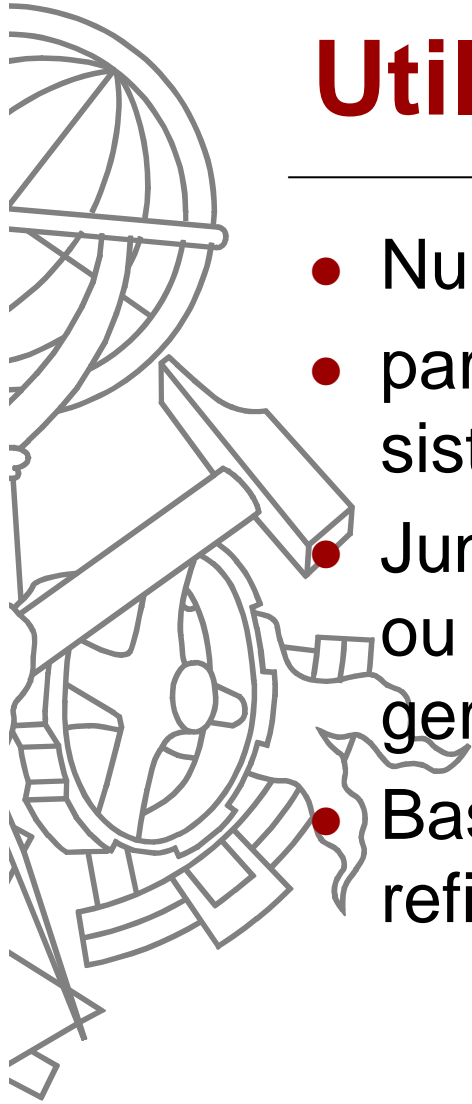


Entity Class

- Used to model long-lived (possibly persistent) information
- Usually correlates directly to class on class diagram







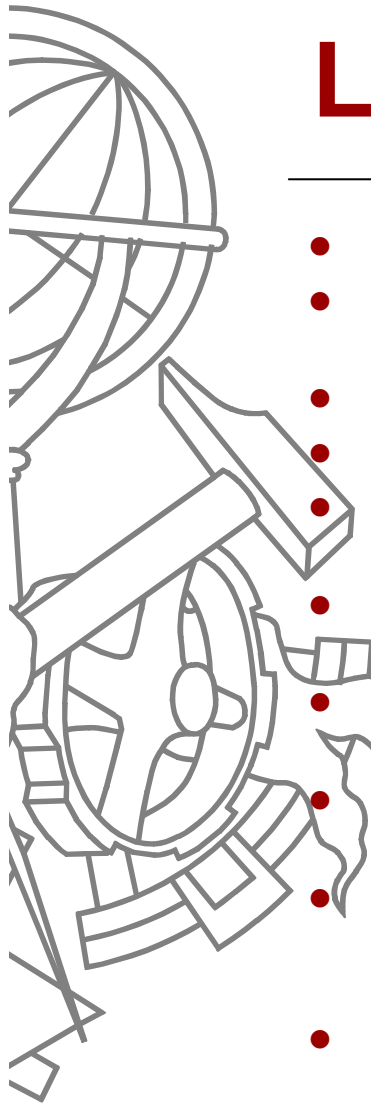
Utilização

- Numa fase inicial da análise
- para capturar a muito **alto nível** a estrutura do sistema a desenvolver
- Juntamente com os diagramas de sequência ou actividades para descrever em passos gerais os **principais** processos a implementar
- Base, através de um processo iterativo, para refinação até ao modelo de implementação



Parte IV

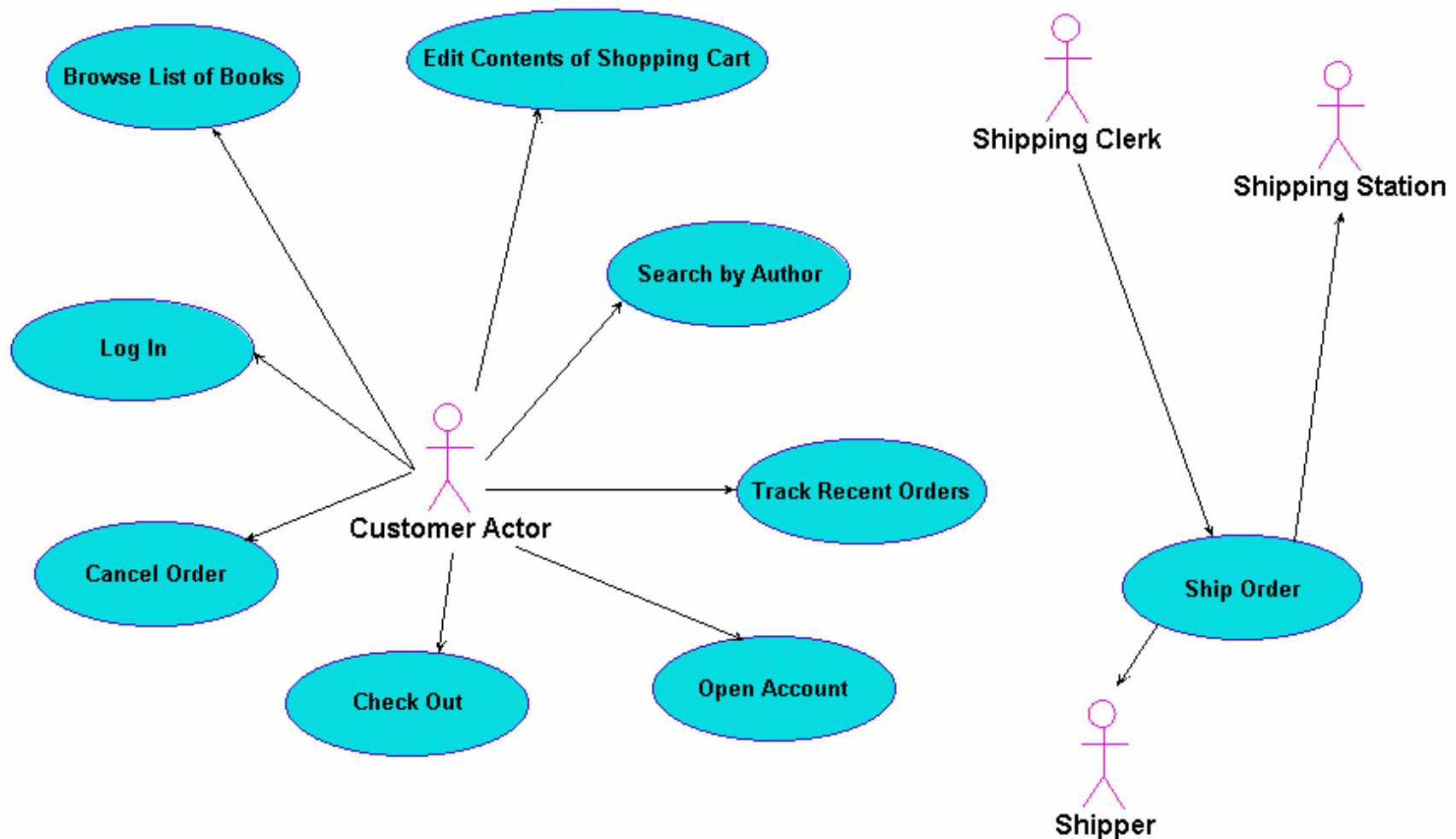
Um Exemplo



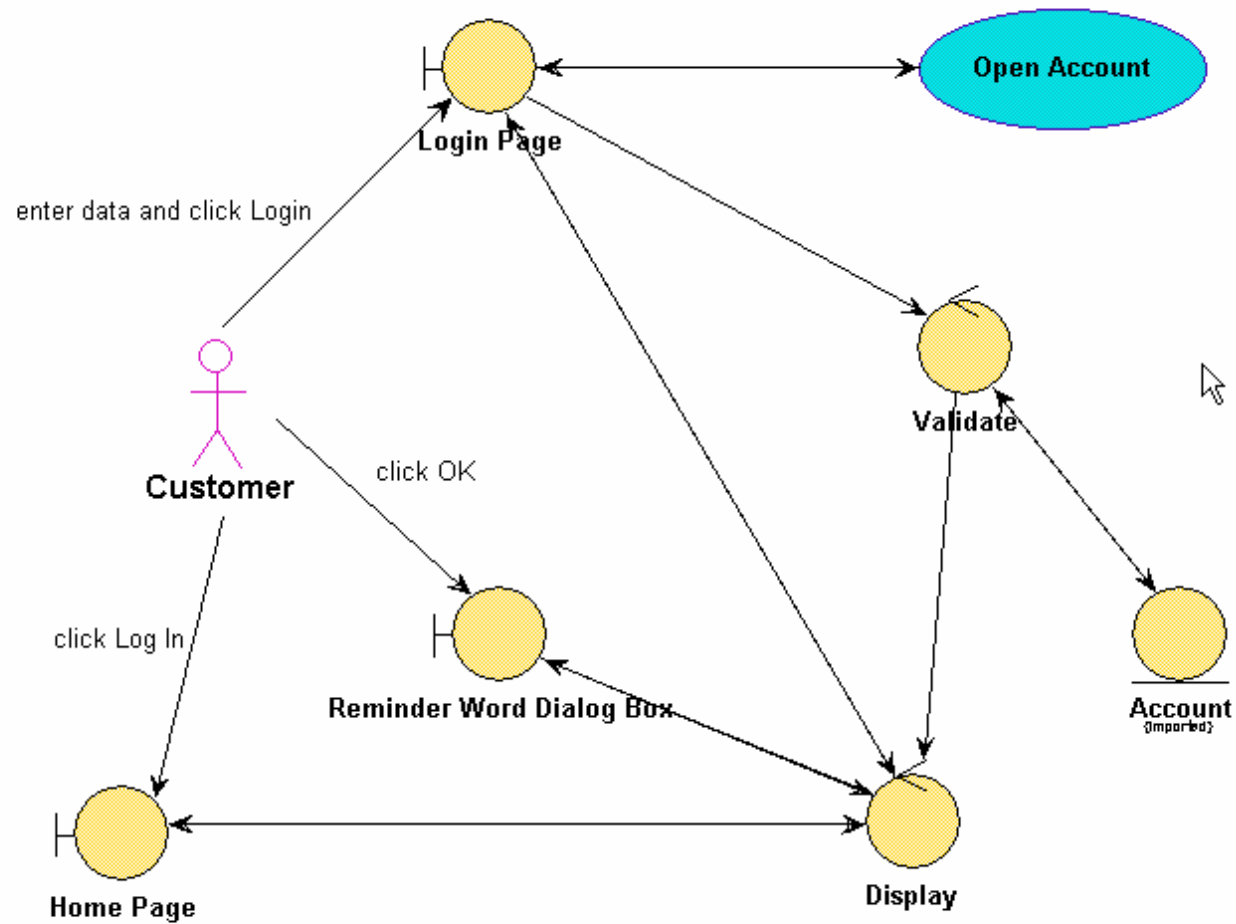
Livraria na internet

- A livraria tem que aceitar encomendas via internet.
- A livraria deve manter uma lista de contas de clientes até o limite de um milhão de clientes.
- As contas dos clientes devem estar protegidas por password.
- Deve ser possível efectuar pesquisas no catalogo principal da livraria.
- Essas pesquisas devem poder ser efectuadas por diversos métodos (ex., titulo, autor, ISBN).
- A livraria deve fornecer um método seguro de pagamento via cartão de crédito.
- Devem existir ligações electrónicas entre a aplicação web a base de dados e o sistema de expedição (ex., DHL).
- Devem existir ligações electrónicas entre a aplicação web a base de dados e o sistema de gestão de inventário.
- A livraria deve manter um esquema de comentários aos livros existentes no catalogo, sendo possível a qualquer utilizador fazer um novo comentário.
- Os livros devem ser classificados segundo uma pontuação baseada nos *inputs* dos utilizadores.

Diagrama de Caso de Utilização



Caso de Utilização “Login”



Caso de Utilização “Edit Shopping Cart”

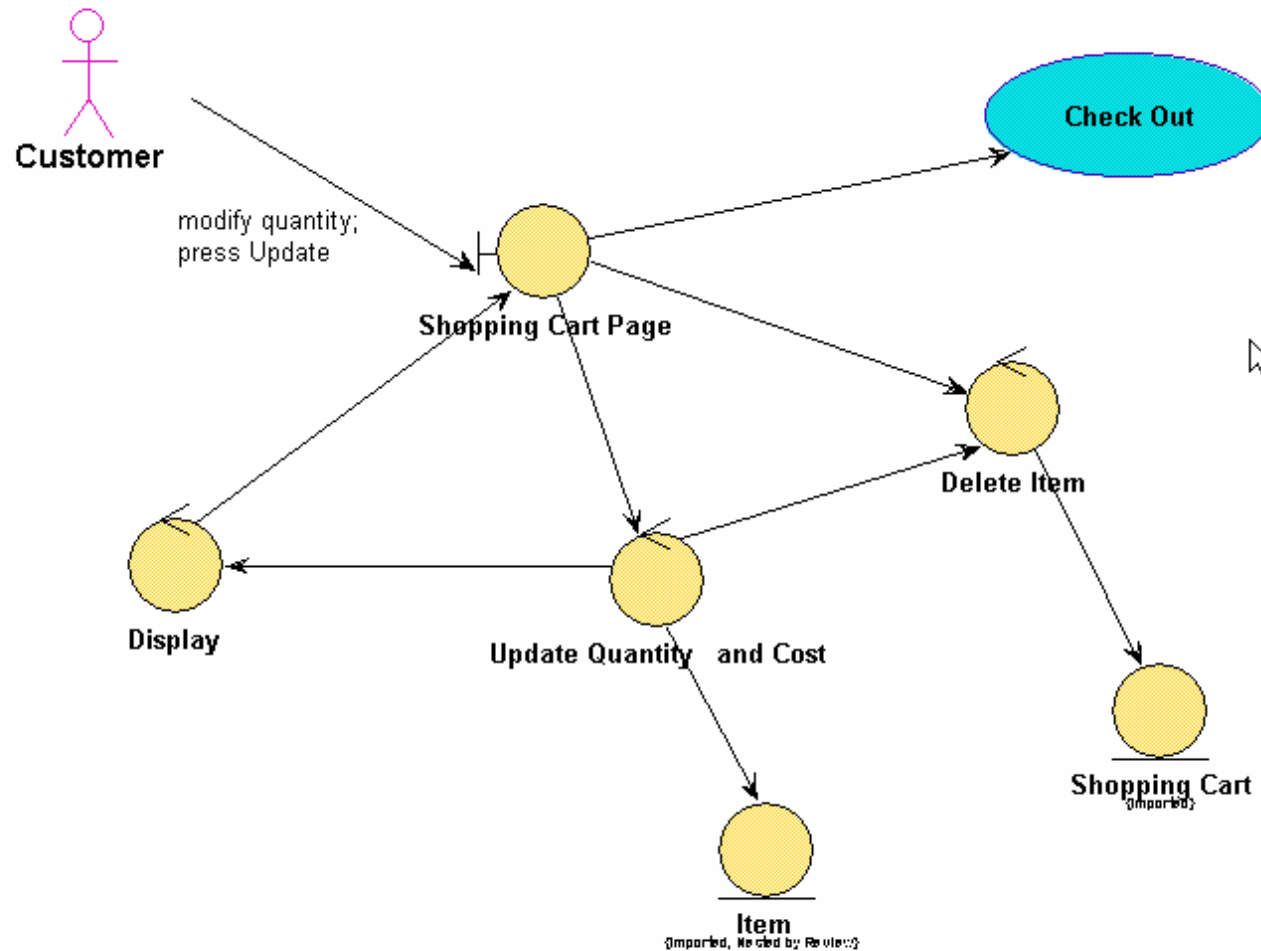
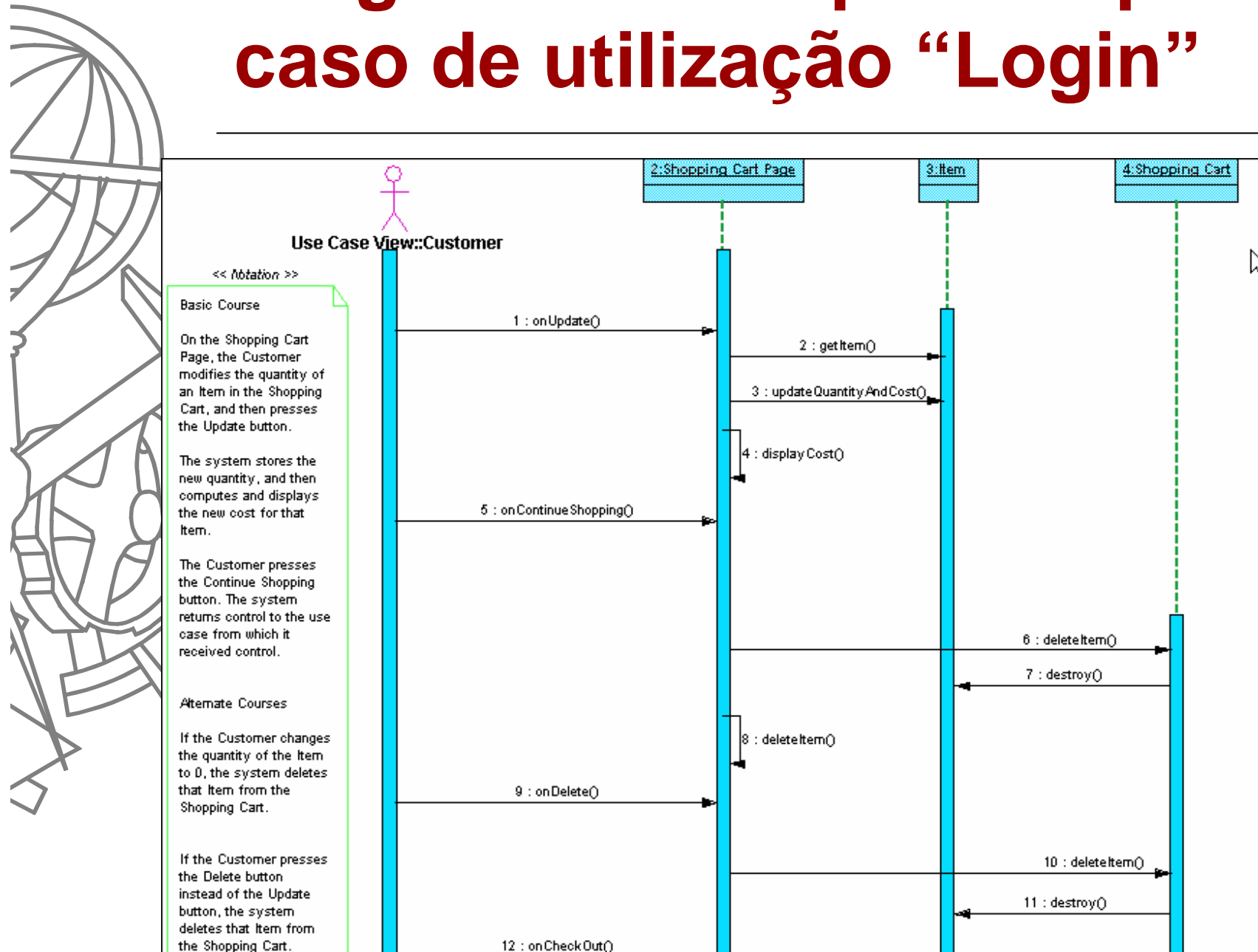


Diagrama de sequência para caso de utilização “Login”



Modelo do domínio

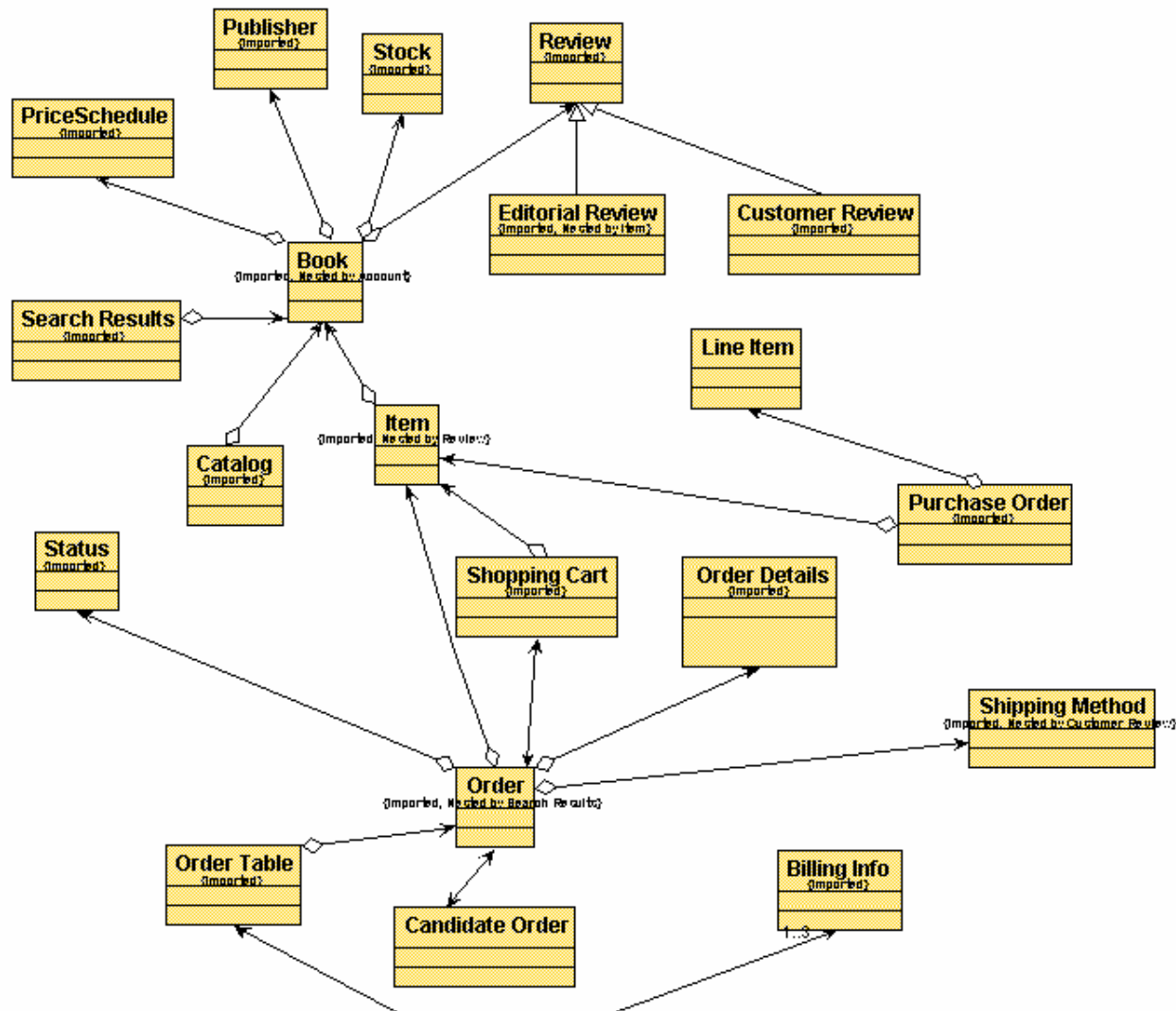


Diagrama de classes (I)

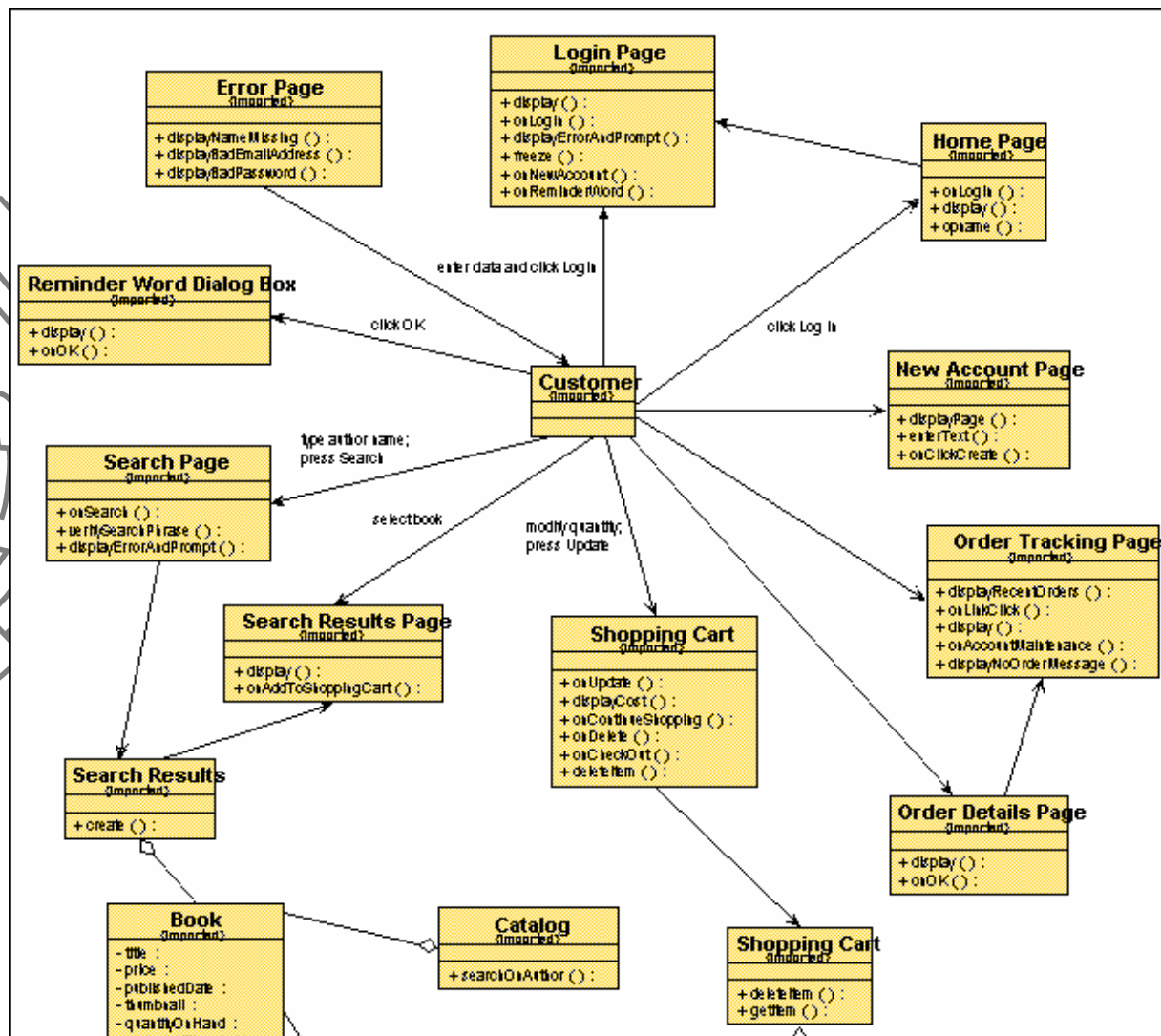


Diagrama de classes (II)

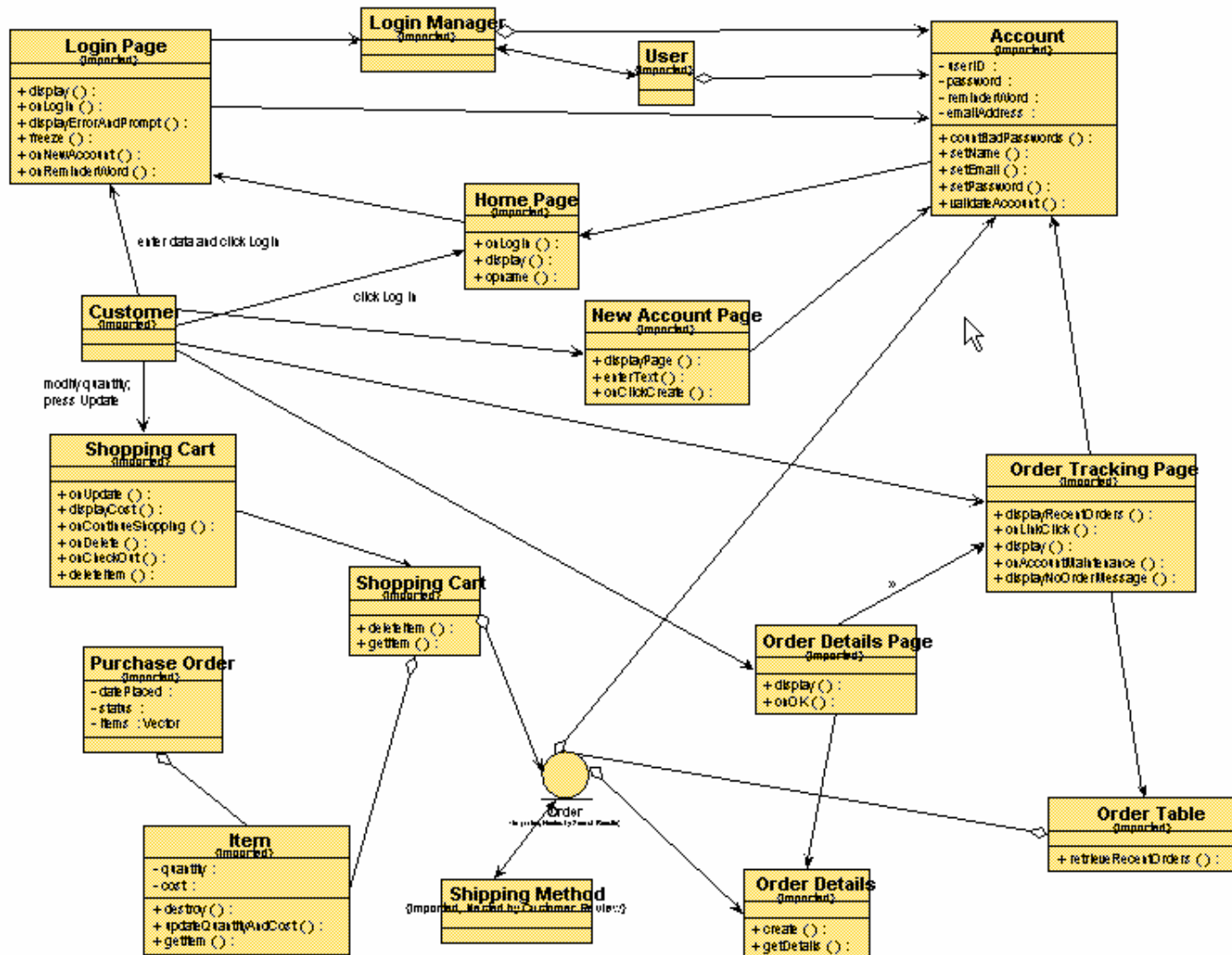
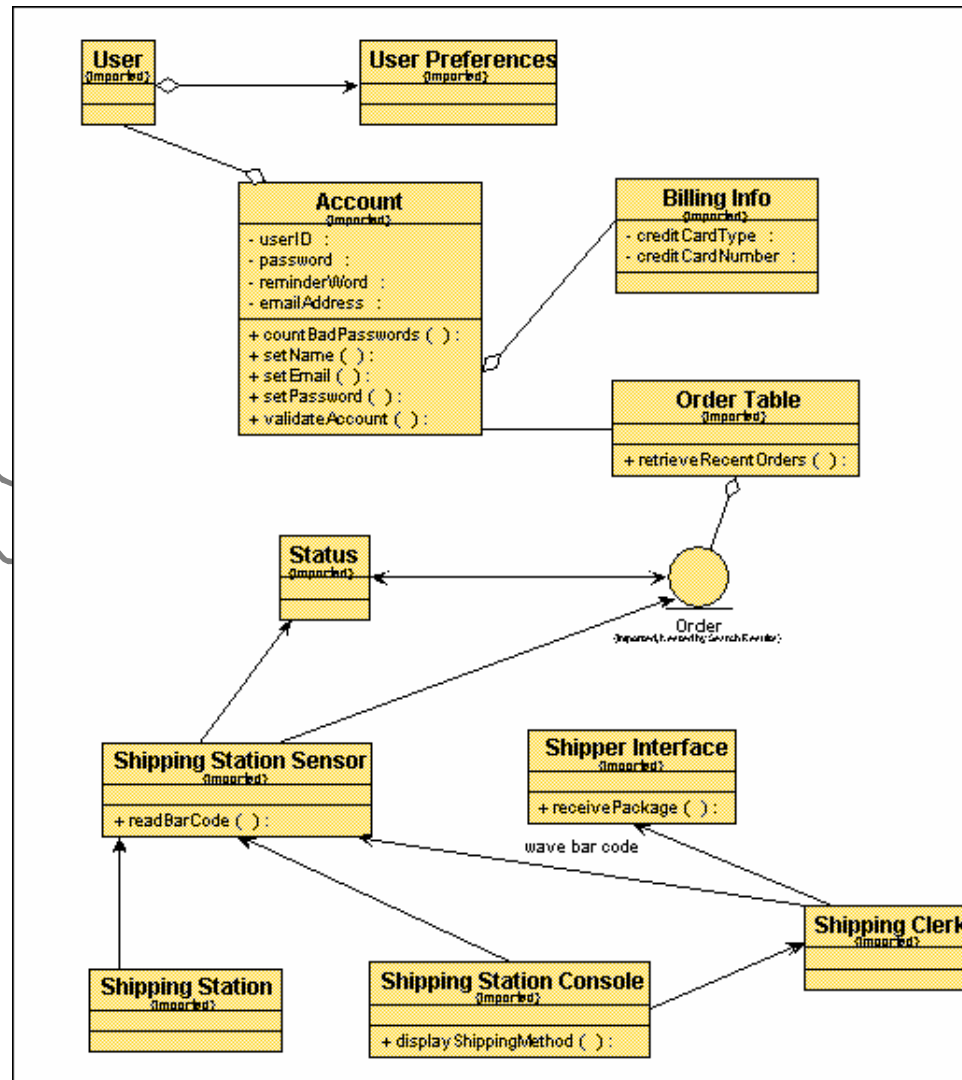


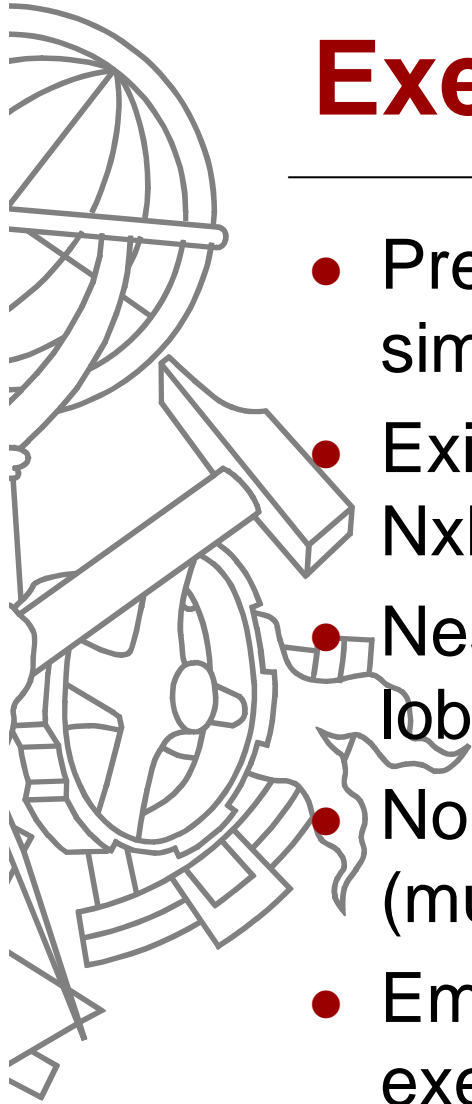
Diagrama de classes (III)





Parte V

Um Exercício



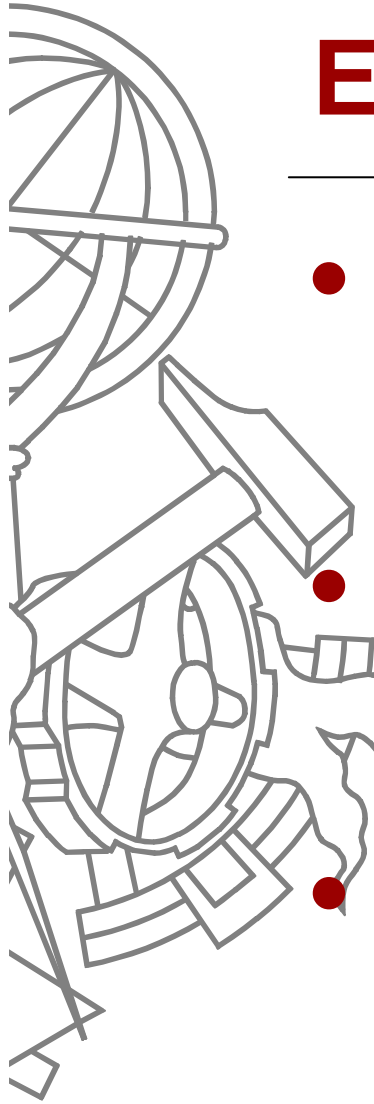
Exercício (1/2)

- Pretende-se um sistema que permita fazer simulações de eco-sistemas
- Existe um mundo (tipicamente uma grelha de $N \times M$ posições)
- Nesse mundo existirão seres/objectos (ex., lobos, veados, ervas, pedras).
- Normalmente em cada quadrado da grelha (mundo) apenas pode existir um ser/objecto
- Em cada passo da simulação cada objecto/ser executa uma acção (por ex., dar um passo)



Exercício (2/2)

- A simulação dura um número de passos prédefinidos pelo utilizador
- O utilizador pode configurar o mundo indicando:
 - Dimensão (NxM)
 - Número e tipo de entidades a colocar
- A ideia é ter diferentes tipos de entidades (ex., 1 veado e 4 lobos)
- Cada tipo de entidade tem os seus próprios objectivos
 - Lobo → capturar o veado
 - Veados → fugir dos lobos e comer erva



Exercício (3/3)

- Cada entidade pode “ver” o mundo à sua volta e movimentar-se no mundo à sua volta
- Normalmente apenas se movimenta uma casa (N, S, E, O) ou (N, NE, E, SE, S, SO, O)
- Normalmente apenas “vê” uma casa mas pode-se implementar animais com visão mais apurada



Fim do Módulo



Bibliografia

- Parte desta apresentação é baseada em:
 - “An Introduction to the Unified Modelling Language”, Kendal Scott, Embarcadero Webinars, 2000.
 - Rosenberg, Doug (2001). *UML for e-Commerce*. Seminário Iconix.



Onde obter mais informação

- UML.org
 - <http://www.uml.org/>
- UML na OMG
 - <http://www.omg.org/technology/uml/index.htm>
- Dicionário UML
 - <http://www.usecasedriven.com/UML.htm>
- Cetus Links
 - http://www.cetus-links.org/oo_uml.html