ENGENHARIA DE SOFTWARE
INTRODUÇÃO

Luís Morgado

2024

INTELIGÊNCIA ARTIFICIAL E ENGENHARIA DE SOFTWARE

O desenvolvimento de sistemas com base em inteligência artificial é caracterizado pela **elevada complexidade** desses sistemas

Requer métodos adequados de **engenharia de software**, nomeadamente, no que se refere à modelação de sistemas, bem como a linguagens de modelação adequadas, como é o caso da *linguagem UML*

Dois problemas principais têm relevância crescente no desenvolvimento, operação e manutenção de software, *complexidade* e *mudança*

- **Complexidade**
  - Expressa na crescente dificuldade de desenvolvimento, operação e manutenção de software, na crescente sofisticação do software produzido, bem como na quantidade crescente de recursos envolvidos, nomeadamente, em termos de capacidade de processamento e de memória utilizada
- **Mudança**
  - Expressa no ritmo crescente a que o software necessita de ser produzido, ou modificado, para satisfazer as necessidades dos respetivos contextos de utilização, quer a nível de funcionalidades disponibilizadas, quer a nível das tecnologias utilizadas

CRESCIMENTO EXPONENCIAL DE RECURSOS COMPUTACIONAIS

IBM Real-Time Computer Complex - NASA Manned Spacecraft Center
Década de 1960

Como exemplo de comparação do crescimento da capacidade computacional disponível nos dispositivos actuais, muitos dos telefones móveis de hoje têm mais capacidade computacional que os computadores envolvidos na missão lunar da NASA na década de 1960

SOFTWARE

Hoje

ENGENHARIA DE SOFTWARE

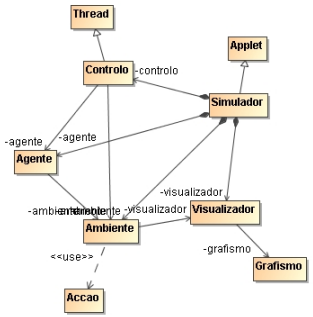
A *engenharia de software* é uma área de engenharia orientada para a especificação, desenvolvimento e manutenção de software, que tem por objectivo o desenvolvimento, operação e manutenção de software de modo *sistemático* e *quantificável*

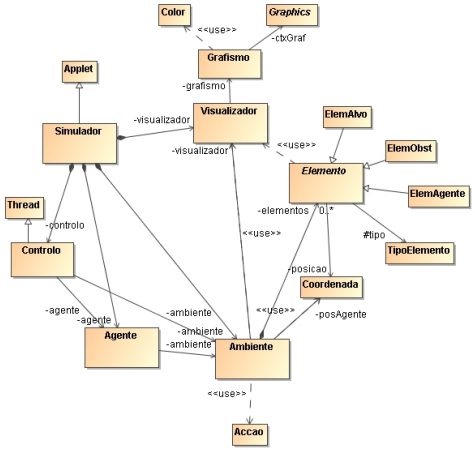
- **Sistemático**
  - Significa a capacidade de **realizar o desenvolvimento de software de forma organizada e previsível**, de modo a garantir a satisfação dos requisitos definidos, incluindo tempo e recursos necessários
- **Quantificável**
  - Significa a **capacidade de avaliar os meios envolvidos e os resultados produzidos** no desenvolvimento de software, utilizando métodos e processos adequados para **garantir a qualidade e o desempenho do software produzido**, bem como a criação de documentação e a monitorização do processo de desenvolvimento
  - O desenvolvimento de software de forma quantificável é importante para garantir que os requisitos do software sejam cumpridos, bem como para prever e gerir os recursos necessários para o desenvolvimento e operação do software produzido

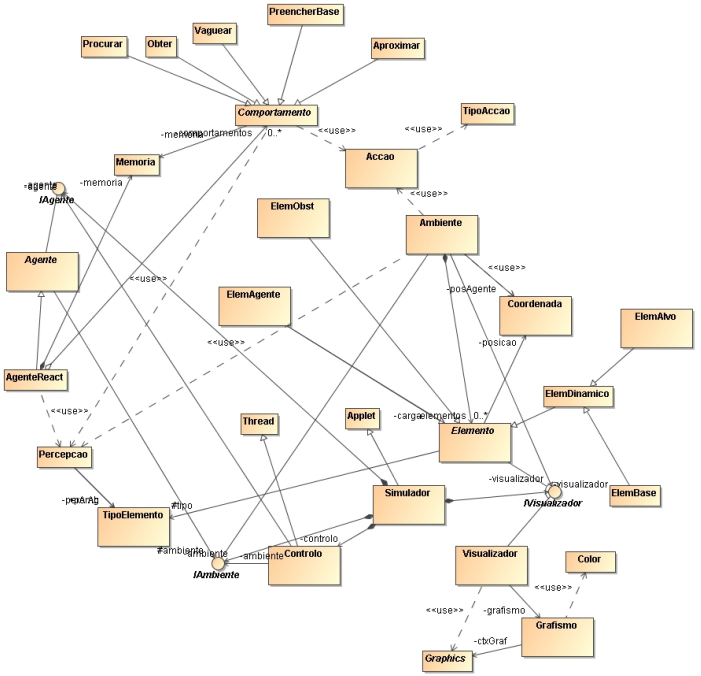
SOFTWARE E COMPLEXIDADE

Complexidade no desenvolvimento de software

- Expressa-se numa **dificuldade crescente em compreender e gerir as partes e as relações entre partes que constituem um sistema de software**, a qual se pode observar numa representação gráfica (num modelo) desse software
- Na prática, essa complexidade traduz-se na **dificuldade e esforço crescente para a concepção e implementação do software**, à medida que vão sendo incluídos mais aspectos do seu funcionamento

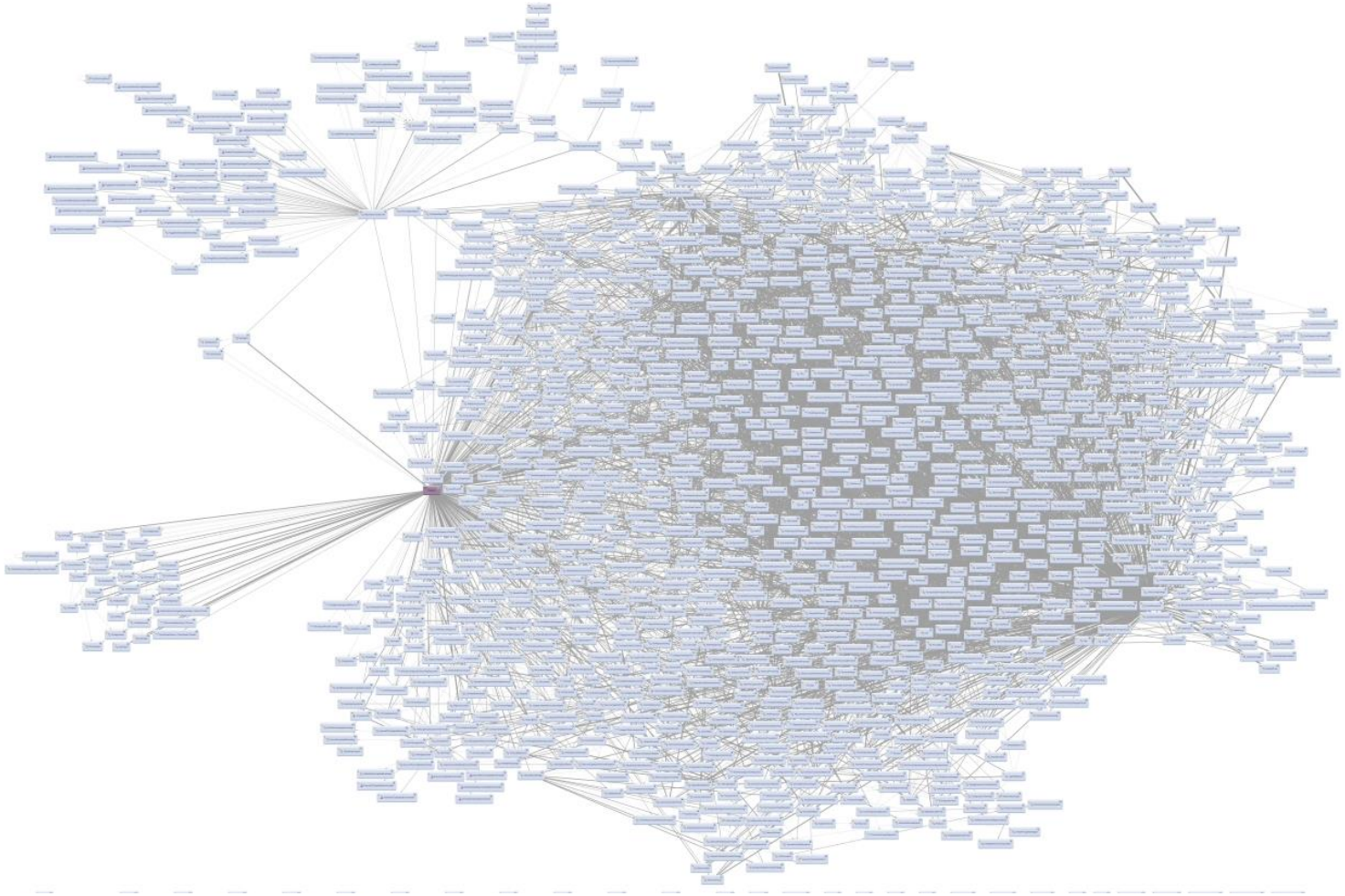






SOFTWARE E COMPLEXIDADE

O crescimento da complexidade de um sistema de forma não adequadamente controlada, pode tornar o **esforço de desenvolvimento muito elevado**, a ponto de comprometer a viabilidade do respetivo projecto



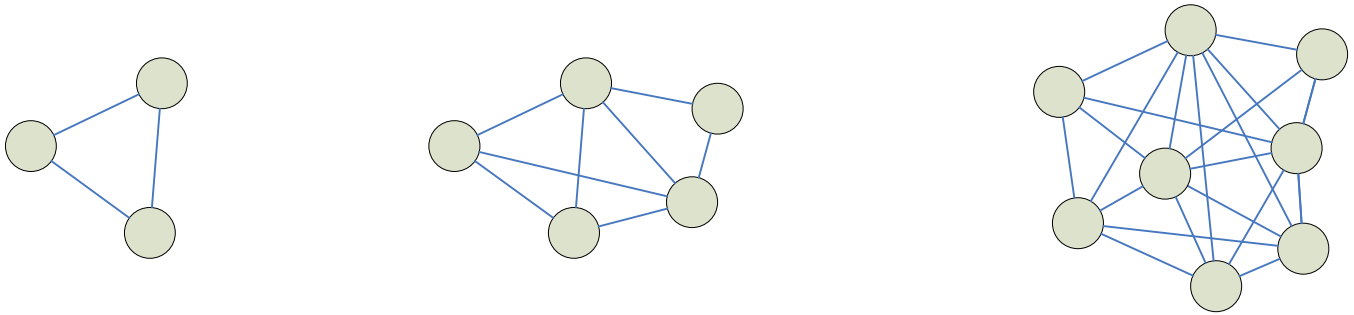
COMPLEXIDADE

Grau de **dificuldade de previsão** das propriedades de um sistema dadas as propriedades das partes individuais [Weaver, 1948]

- Relacionada com a **informação** que é necessária para a caracterização de um sistema
- Um sistema é tanto mais **complexo** quanto mais **informação** for necessária para a sua **descrição**
- Reflecte-se no **esforço** necessário para geração da **organização** (**ordem**) do sistema

O PROBLEMA DA COMPLEXIDADE

COMPLEXIDADE ESTRUTURAL



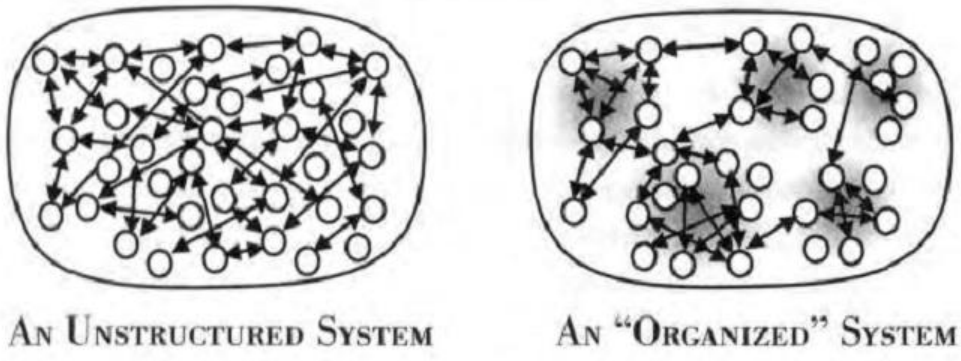
- **UM PROBLEMA DE INTERACÇÃO**
  - De partes do sistema
  - De elementos de informação
  - De elementos das equipas de desenvolvimento
- **EXPLOSÃO COMBINATÓRIA**
  - Um sistema com duas vezes mais partes é muito mais do que duas vezes mais complexo

CRESCIMENTO EXPONENCIAL
DA COMPLEXIDADE

COMPLEXIDADE E ORGANIZAÇÃO

Tipos de complexidade associados à organização de um sistema

- Complexidade **organizada**
  - Resulta de padrões de inter-relacionamento entre as partes *correlacionáveis* no espaço e no tempo
  - **Ordem, organização, função, propósito**
- Complexidade **desorganizada**
  - Resulta do número e *heterogeneidade* das partes de um sistema
    - As partes podem interactuar entre si, mas a **interacção é irregular**
  - **Desordem, desorganização, perda de função e de propósito**



AN UNSTRUCTURED SYSTEM

AN "ORGANIZED" SYSTEM

[Flake, 1998]

ARQUITECTURA DE SOFTWARE

A arquitectura de software tem por objectivo abordar o problema da complexidade com base numa organização adequada do software a produzir

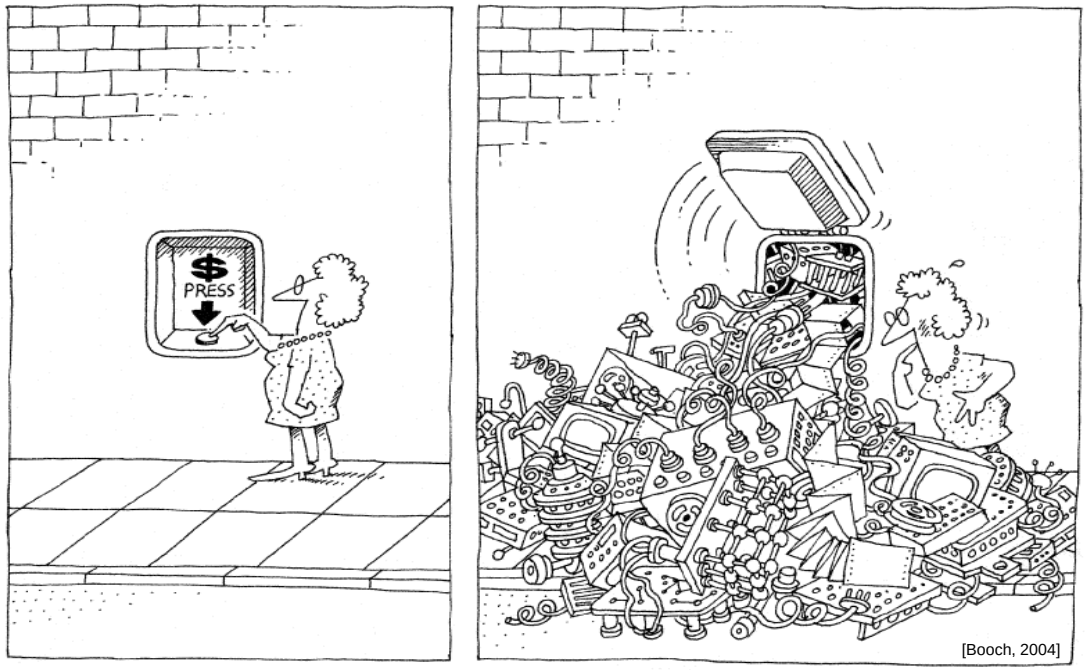
Tem por base dois princípios principais: *abstração* e *modularização*

- **ABSTRACÇÃO**
- **MODULARIZAÇÃO**

Objectivo: reduzir a complexidade desorganizada e controlar a complexidade organizada necessária ao propósito do sistema

COMPLEXIDADE

- Redução
  - Eliminar complexidade desorganizada
- Controlo
  - Gerir complexidade organizada (necessária para realizar o propósito do sistema)



Quando a complexidade não é adequadamente resolvida, mas apenas acumulada, expressar-se-á em algum momento…

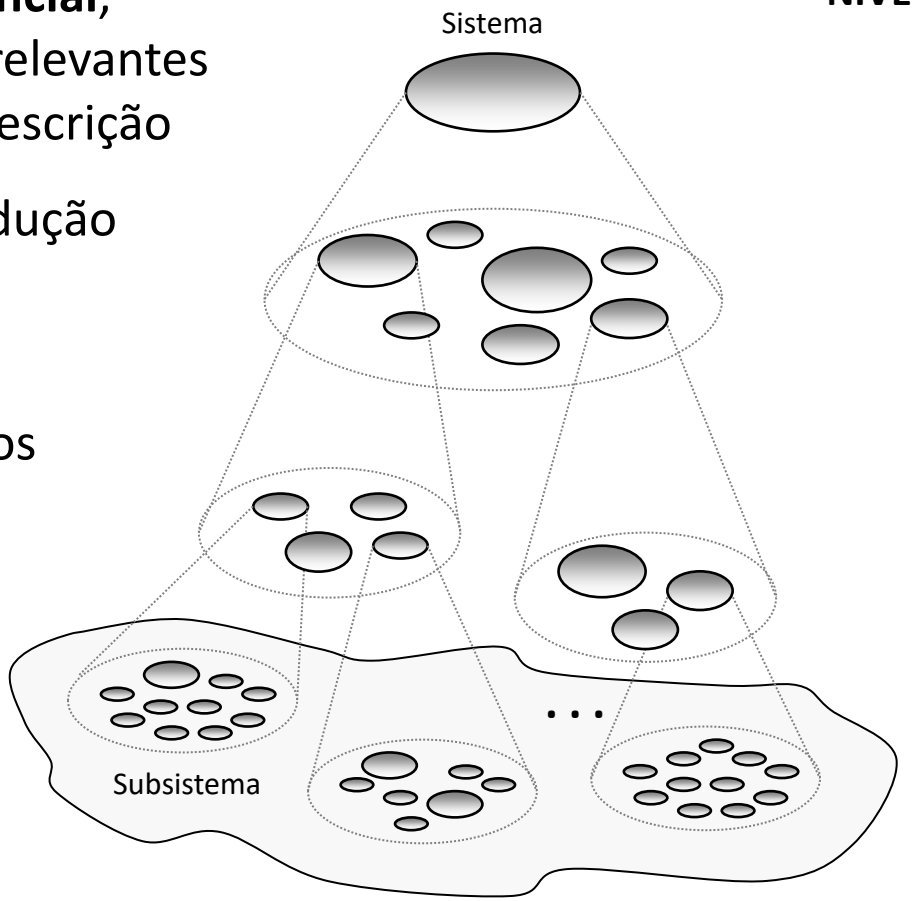
ABSTRACÇÃO

- Processo de descrição de conhecimento a *diferentes níveis de detalhe* (**quantidade de informação**) e *tipos de representação* (**estrutura da informação**) [Korf, 1980]
- **Abstracção é uma ferramenta base para lidar com a complexidade**
  - Identificação de características comuns a diferentes partes
  - Realçar o que é essencial, omitir detalhes não relevantes
  - Modelos
- Desenvolvimento de um sistema complexo
  - Criação de ordem de forma progressiva
  - Processo iterativo guiado por conhecimento

ABSTRACÇÃO

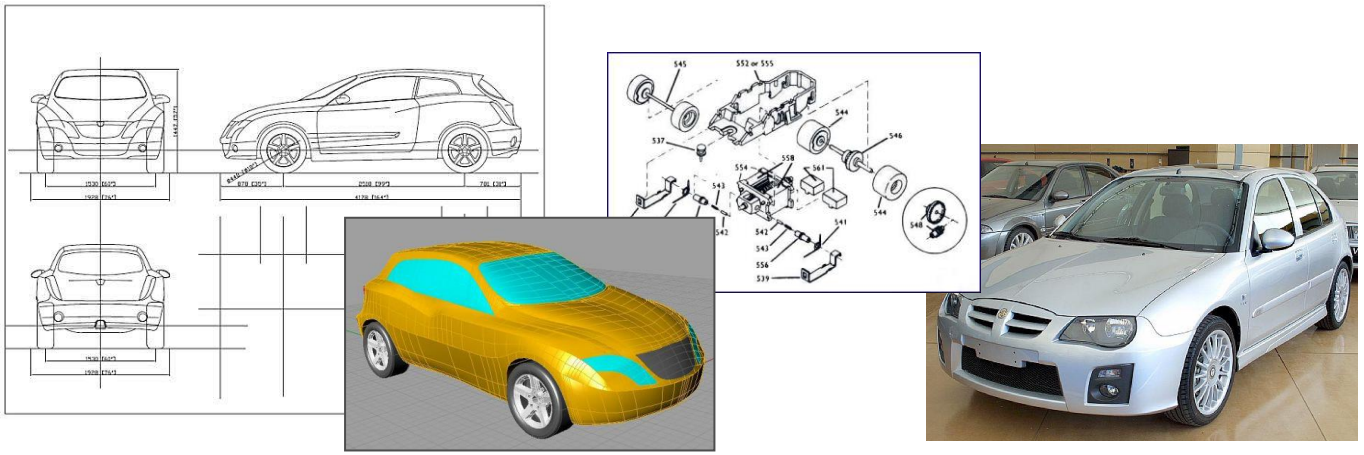
FERRAMENTA BASE PARA LIDAR COM A COMPLEXIDADE

- **Realçar** o que é **essencial**, omitir detalhes não relevantes para o contexto de descrição
- **Simplificação** por redução da complexidade da descrição
- **Focagem** nos aspectos mais relevantes para o contexto de descrição
- **Modelos**
  - Descrições abstractas de um sistema



MODELO

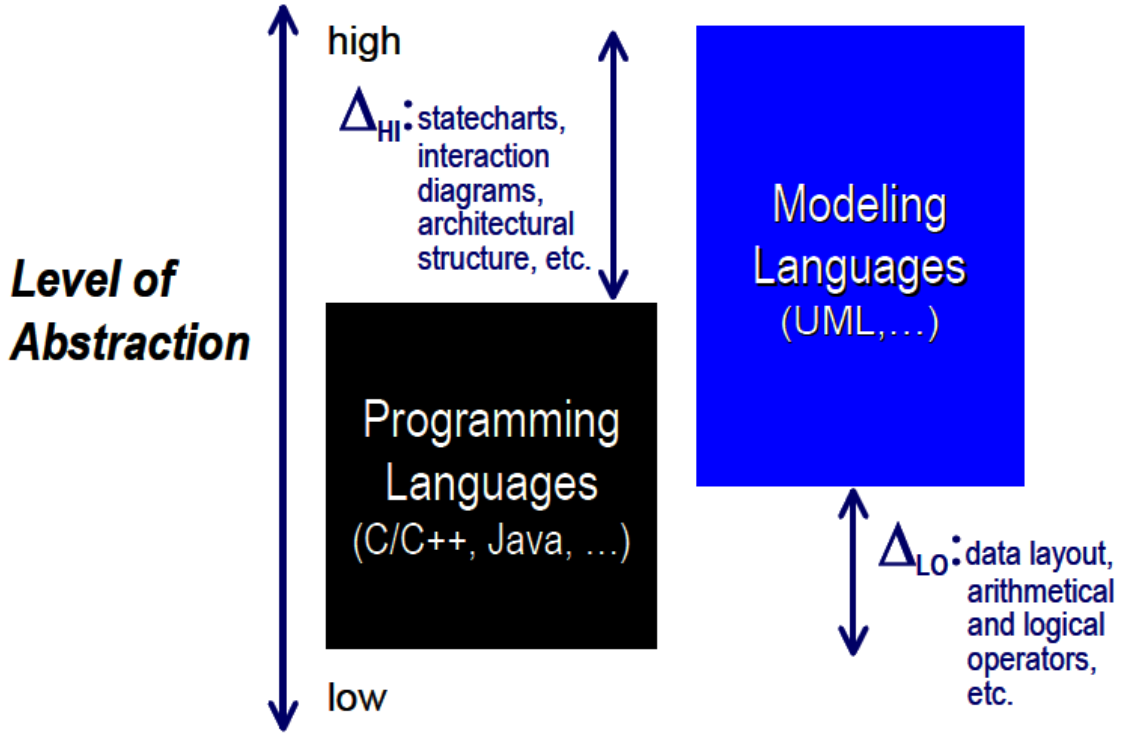
- **Representação abstracta de um sistema**
  - Especificação com base em conceitos abstractos das características fundamentais de um sistema
  - Representação de conhecimento acerca de um sistema
- **Meio para lidar com a complexidade**
  - Obtenção e sistematização progressiva de conhecimento
  - Compreensão e comunicação acerca do sistema
  - Especificação de referência para a realização do sistema
  - Documentação de um sistema



LINGUAGENS DE MODELAÇÃO

DESCRIÇÃO DO SISTEMA A DIFERENTES NÍVEIS DE ABSTRACÇÃO

As diferentes linguagens de especificação de software estão orientadas para a descrição de software a diferentes níveis de abstracção, desde níveis mais detalhados, como as linguagens *assembly*, até níveis mais abstractos como a ***linguagem UML***



[Selic, 2003]

Elementos de descrição diferentes em cada nível de abstracção

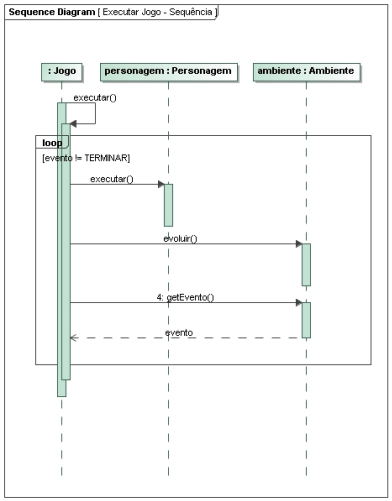
Δ_{HI} : Elementos disponíveis em UML não disponíveis em linguagens mais específicas

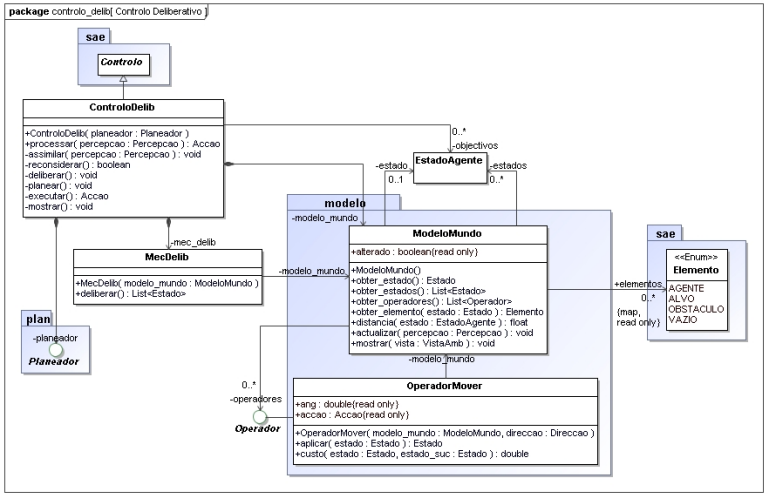
Δ_{LO} : Elementos disponíveis em linguagens mais específicas não disponíveis em UML

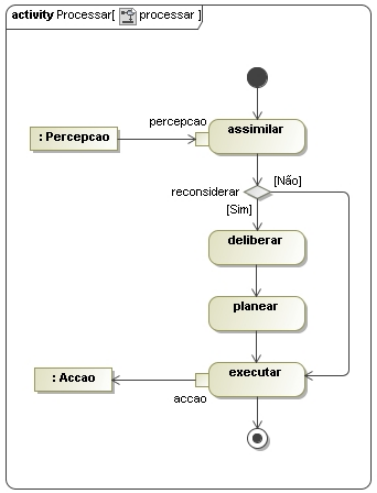
LINGUAGEM UML (*Unified Modeling Language*)

A linguagem UML (*Unified Modeling Language*) é uma linguagem de modelação geral de sistemas, orientada para a **concepção e modelação de software de forma organizada e sistemática**, facilitando a compreensão e descrição dos modelos desenvolvidos

- É uma **linguagem normalizada** que **facilita a compreensão e comunicação** dos modelos, quer para quem concebe os modelos, quer para quem os utiliza
- **Facilita a produção de código**, possibilitando uma tradução organizada e sistemática dos modelos para código, incluindo a geração automática de código
- Adopta uma abordagem ***orientada a objectos***, o que **contribui para lidar com a complexidade**, bem como para facilitar a reutilização de código

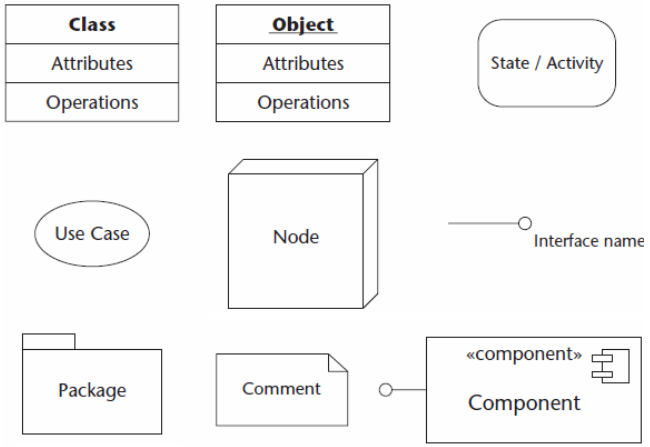






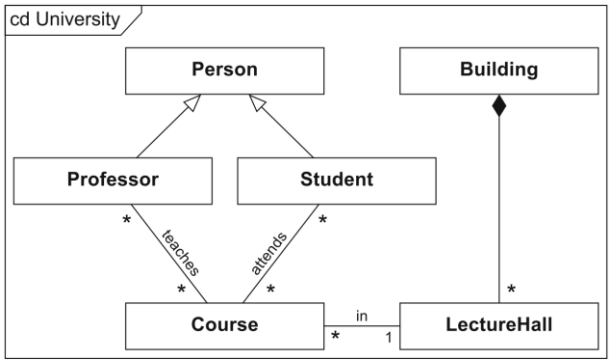
LINGUAGEM UML

- **Classificadores** (tipos de elementos base)
  - *Class*
  - *Association*
  - *Interface*
  - *Component*
  - *Package*
  - *State*
  - *Activity*
  - (…)



[Eriksson et al., 2004]

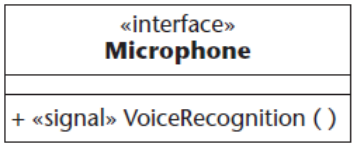
- **Diagramas**
  - Na linguagem UML os modelos são organizados em diagramas que permitem representar diferentes perspectivas de modelação de um sistema
  - Representação gráfica facilita a percepção das relações entre elementos



LINGUAGEM UML

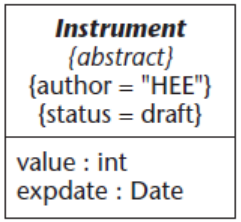
- **Mecanismos de Extensão**
  - **Estereótipos**
    - Mecanismo de extensão da linguagem que possibilita a atribuição de significado específico a elementos base da linguagem
    - Permitem estender o vocabulário UML de modo a definir elementos de modelação, com características específicas, a partir dos já existentes
    - Podem ter representação textual ou gráfica
  - **Propriedades**
    - São utilizadas para definir propriedades dos elementos da linguagem, com eventual associação de informação específica, por exemplo, autor ou versão
  - **Restrições**
    - Especificam a semântica ou condições que se devem verificar em relação a elementos de um modelo

Representação textual de estereótipo
no formato <<*estereótipo*>>

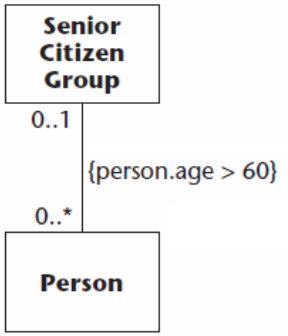


Notação:
«*estereótipo*»

Representação gráfica
(estereótipo *interface*)



Notação:
{*propriedade*}

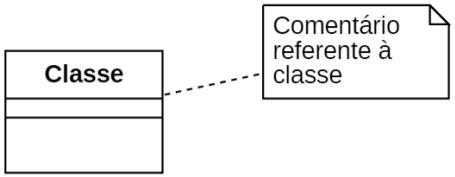


Notação:
{*restrição*}

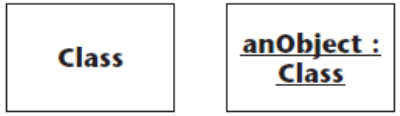
[Eriksson et al., 2004]

LINGUAGEM UML

- **Complementos de Informação**
  - **Adornos**
    - Informação textual ou gráfica indicativa de aspectos específicos de um elemento, por exemplo, atributo ou método *estático*
  - **Comentários**
    - Elemento gráfico contendo informação textual referente a elementos de um modelo

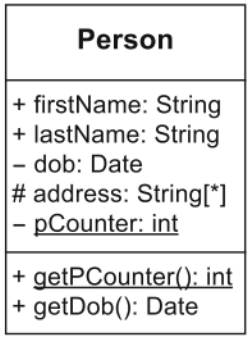


Nome de classe: **Negrito**
Nome de objecto: **Sublinhado**



[Eriksson et al., 2004]

Atributo ou método *estático*: Sublinhado



[Seidl, 2012]

LINGUAGEM UML

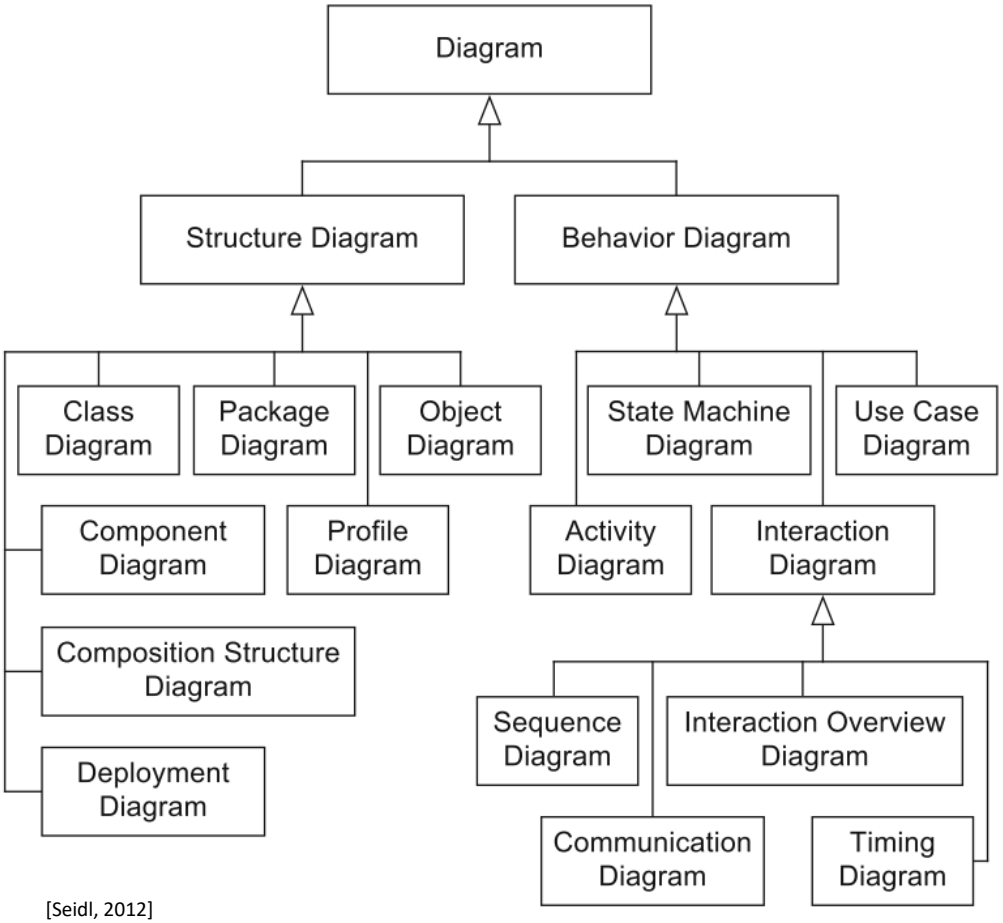
PERSPECTIVAS DE MODELAÇÃO

Diagramas

Na linguagem UML os modelos são organizados em diagramas que permitem representar diferentes perspectivas de modelação de um sistema, organizados em duas perspectivas principais:

- ***Perspectiva estrutural***
  - Referente à organização das partes e relações entre partes que constituem a estrutura de um sistema
- ***Perspectiva comportamental***
  - Referente à organização funcional e dinâmica de um sistema que determina o seu comportamento

Tipos de diagramas

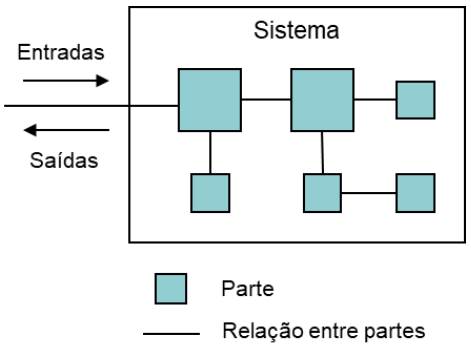


[Seidl, 2012]

MODELOS DE ESTRUTURA

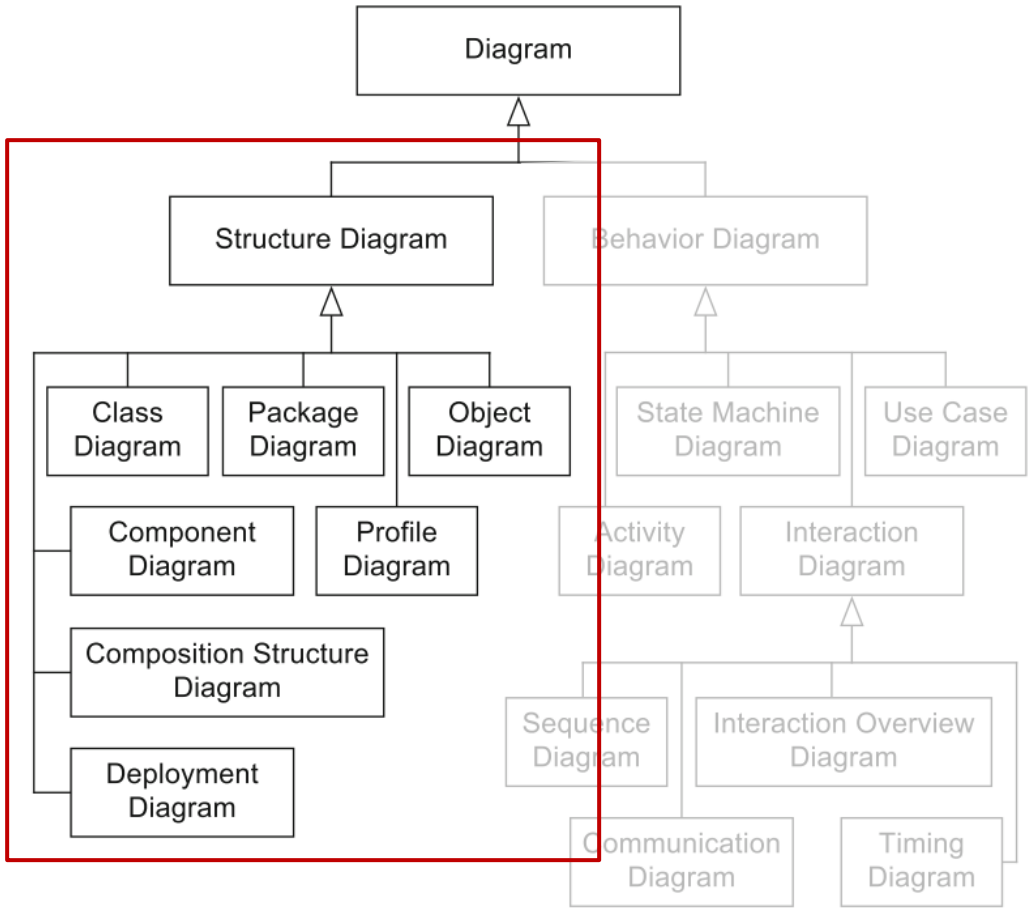
A **estrutura** de um sistema define o **conjunto de partes e relações entre partes** que o constituem

A **estrutura** define a **organização de um sistema num espaço**, sendo um sistema lógico (*software*), o espaço é lógico, concretizado na memória do sistema, as partes residem nessa memória sob a forma de valores na memória (estruturas de dados), relacionadas através de relações de dados, por exemplo, dados que referenciam outras estruturas de dados



Linguagem UML

Perspectiva Estrutural

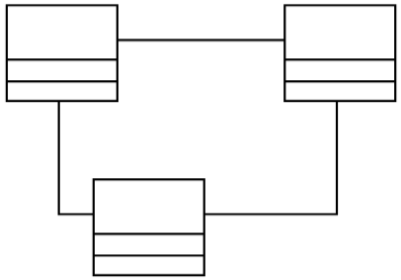


[Seidl, 2012]

DIAGRAMAS DE CLASSES

MODELOS DE ESTRUTURA

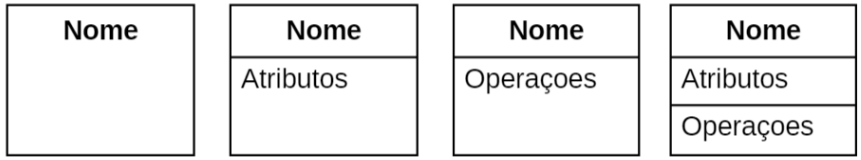
- Descrevem uma abstracção das partes e das relações entre partes de um sistema
  - Organização *estática* do sistema
  - Foco na estrutura

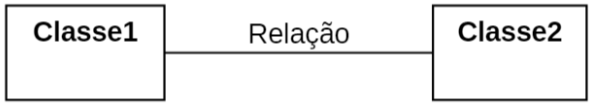


- **Classes**
  - **Atributos**
    - Definição de estrutura
  - **Operações** (*métodos*)
    - Definição de comportamento

- **Relações**
  - Representam interdependência entre partes

Diferentes modos de representar graficamente uma *classe*, com diferentes níveis de detalhe

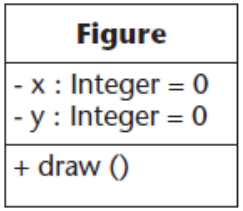


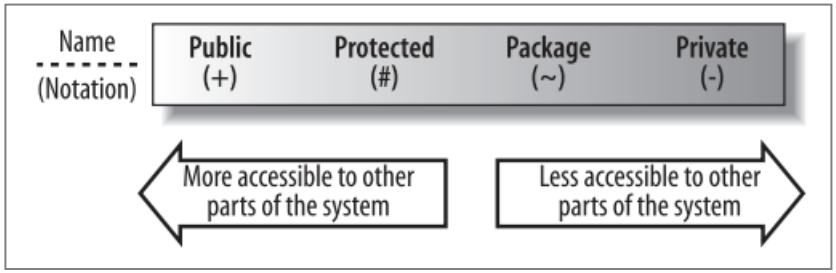


DIAGRAMAS DE CLASSES

Atributos

- Representação de estrutura
  - Elementos de informação
- Caracterizados por:
  - Designação
  - Tipo
  - Visibilidade
    - Público (+)
    - Privado (-)
    - Protegido (#)
    - Pacote (~)
- Sintaxe:
  - visibility name:type = init_value {property_string}

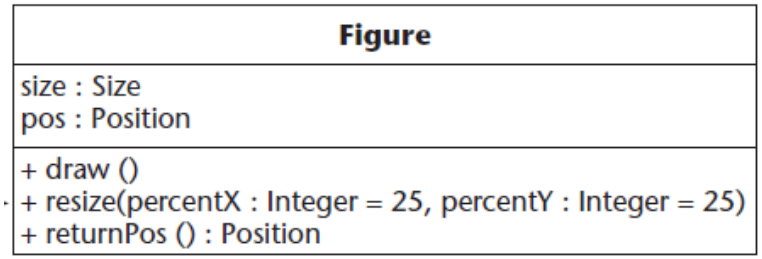




[Miles & Hamilton, 2006]

Operações

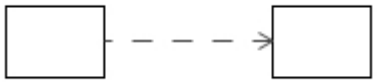
- Representação de comportamento
  - Acções ou funções suportadas



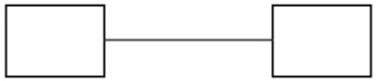
DIAGRAMAS DE CLASSES

Relações entre classes

- **Dependência**
  - Relação na qual uma parte utiliza ou depende de outra parte, por exemplo, uma classe tem um método com um parâmetro que é uma instância de outra classe, ou cria localmente uma instância de outra classe



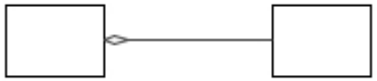
- **Associação**
  - Relação na qual uma parte está estruturalmente associada a outra parte através de um dos seus atributos, ou seja, um dos seus atributos tem informação acerca de outra parte, por exemplo, uma classe tem um atributo cujo tipo é uma instância de outra classe



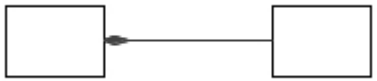
DIAGRAMAS DE CLASSES

Relações entre classes

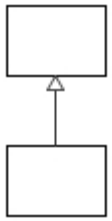
- **Agregação**
  - Caso particular de associação que indica que uma parte agrega outras partes, as quais podem existir independentemente da parte agregadora, por exemplo, a relação entre turma e aluno



- **Composição**
  - Caso particular de associação que indica que uma parte é composta por outras partes que só existem no contexto da parte composta, por exemplo, a relação entre apartamento e divisão (o apartamento é composto por várias divisões que não existem separadas do todo)



- **Generalização**
  - Relação estrutural que indica que uma parte é uma especialização de outra parte mais geral

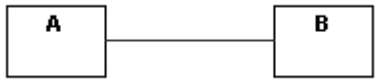


DIAGRAMAS DE CLASSES

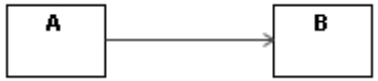
Propriedades das relações

- **Direcção**
  - Indica a navegabilidade de uma associação, ou seja, que partes contêm informação acerca de outras partes
  - Pode ser *bidirecional* ou *unidirecional*
- **Multiplicidade**
  - Indica quantas instâncias de uma parte estão associadas a instâncias de outras partes
- **Papel**
  - Indica o papel de uma parte numa associação, ou seja, o significado concreto ou a responsabilidade que lhe corresponde
- **Qualificação**
  - O qualificador de uma associação designa uma chave utilizada para obter um item da colecção respectiva

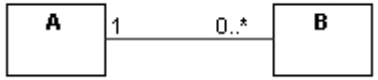
Associação *bidirecional*, as instâncias de ambas as partes conhecem-se reciprocamente



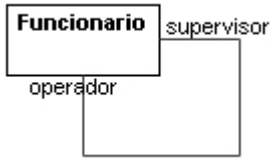
Associação *unidirecional*, apenas as instâncias da parte A conhecem instâncias da parte B



Por cada instância de A existem zero ou mais de B, por cada instância de B existe uma instância de A



As instâncias de funcionário desempenham os papéis de *operador* ou *supervisor* (cada operador tem associado um supervisor, cada supervisor tem associado um operador

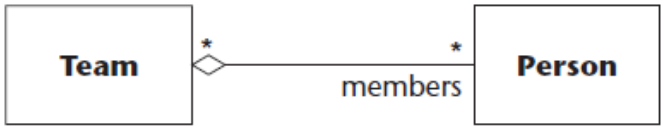


O acesso aos produtos de uma encomenda é realizado através do atributo *codigoProduto*

DIAGRAMAS DE CLASSES

Agregação

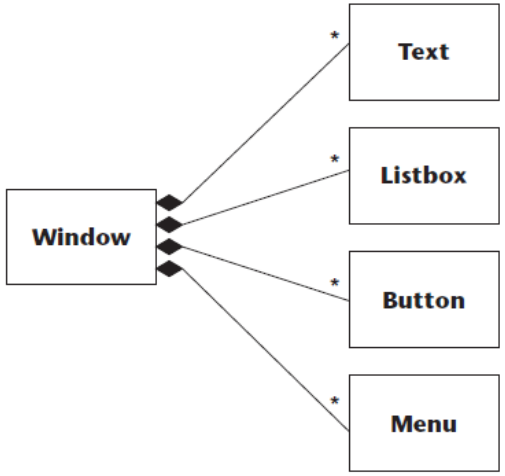
- Relação Parte - Todo
- Denota que uma parte contém outra em termos lógicos ou físicos
- Forma fraca de composição de partes
  - Não define restrições semânticas acerca de:
    - Pertença das partes
    - Criação e destruição das partes
    - Períodos de existência das partes



Relação *parte-todo*

Composição

- Forma forte de composição de partes
  - Define restrições semânticas específicas
    - O todo cria e destrói as partes
    - Sobreposição de períodos de existência das partes
    - Implica exclusividade na composição das partes
  - Organização hierárquica em árvore
    - Níveis de abstracção

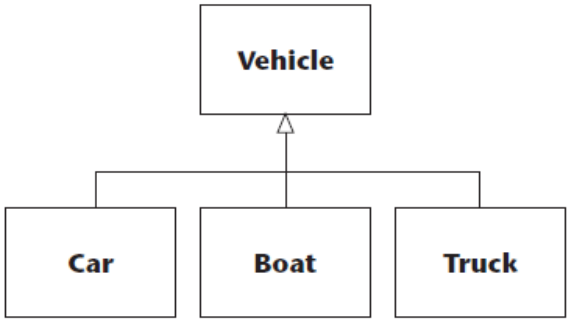


Regra da *não-partilha*

[Eriksson et al., 2004]

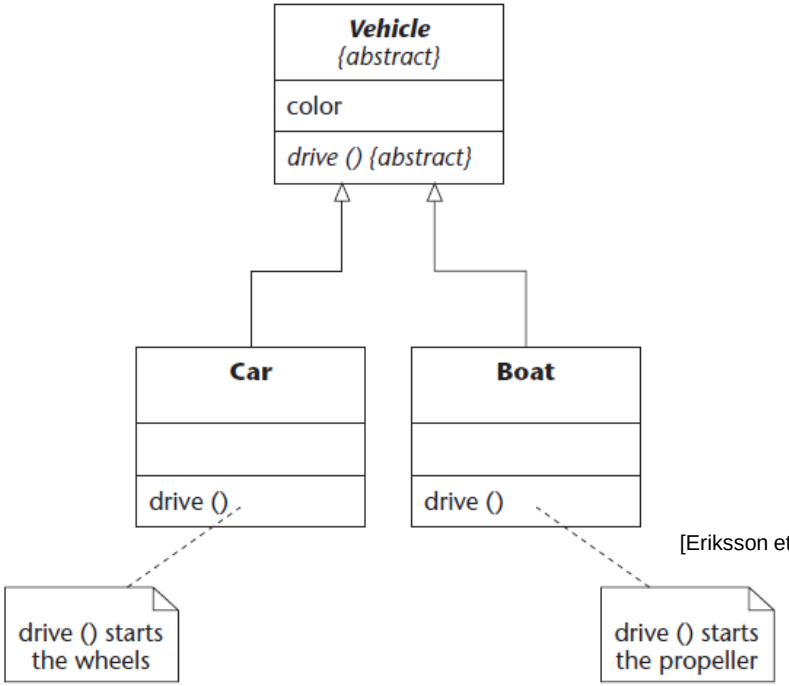
DIAGRAMAS DE CLASSES

Generalização



A *generalização* é uma relação estrutural que indica que uma parte é uma especialização de outra parte mais geral

Polimorfismo



[Eriksson et al., 2004]

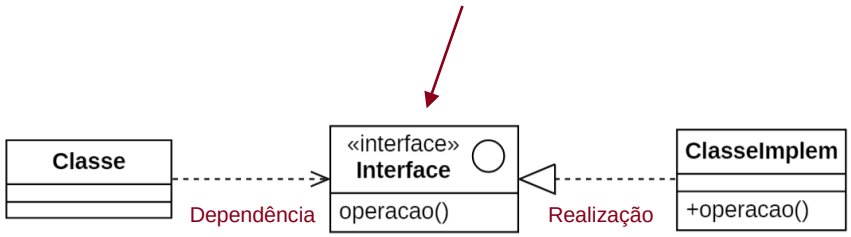
O *polimorfismo* é uma característica da modelação e programação orientada a objectos que possibilita que objectos de uma mesma classe assumam formas distintas, nomeadamente expressões comportamentais distintas, por exemplo, operações com diferentes implementações que podem ser utilizadas de forma homogénea independentemente do tipo dos objectos, isso é conseguido com uma definição comum das operações através de uma *classe abstracta* ou de uma *interface*

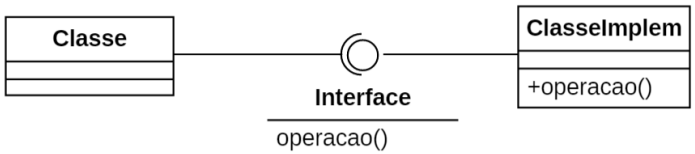
DIAGRAMAS DE CLASSES

- **Interface**
  - Classificador que define as características visíveis de uma classe
  - Conjunto **coeso** de características
  - Define um contrato
    - Não implementa essas características
  - Promove a modularidade através do **encapsulamento** da implementação
    - Define um contrato de serviços independente da forma de implementação

- **Realização**
  - Representa implementação
  - Relação entre uma interface e uma classe ou um componente
  - Uma classe ***realiza*** as características de uma interface implementando-as
  - Uma classe pode implementar mais que uma interface

A interface define um contracto de prestação de serviços independente da implementação

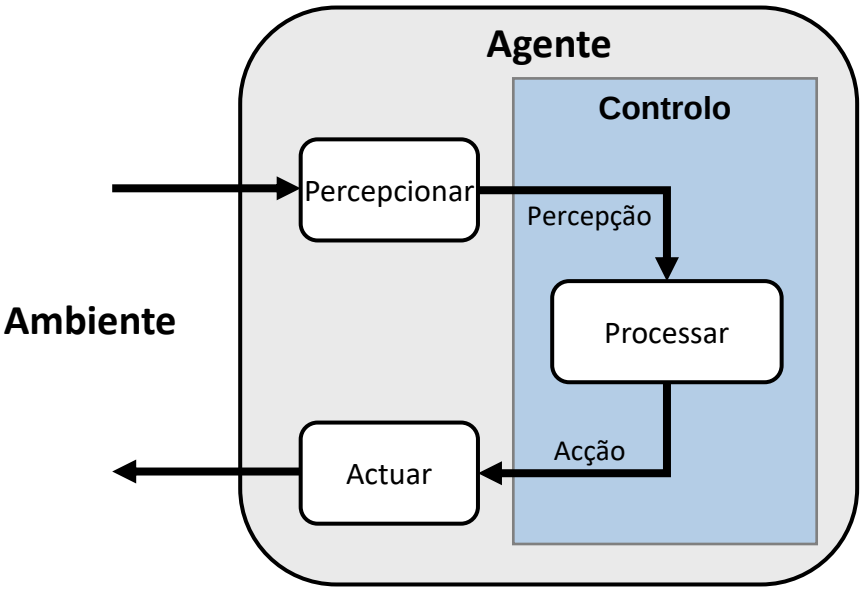




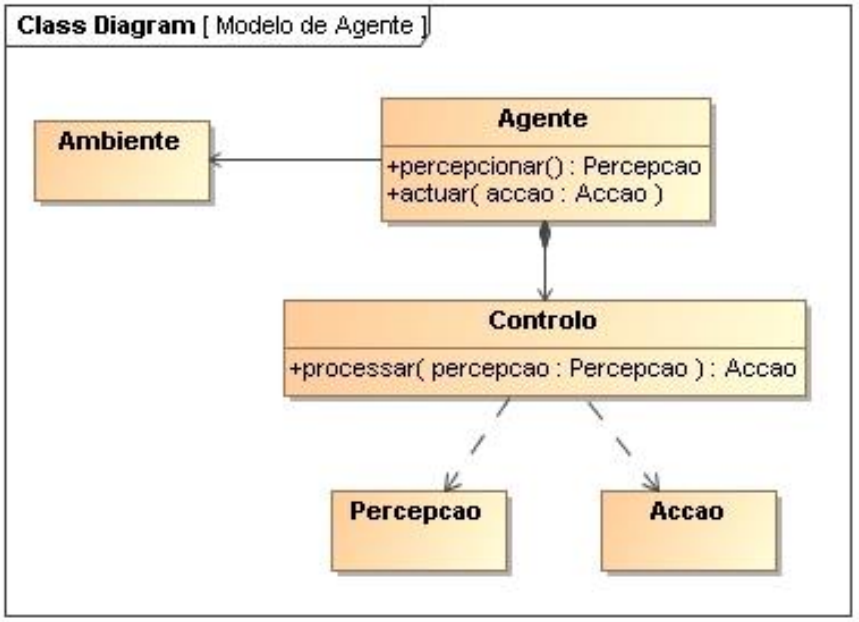
Diferentes notações gráficas associadas ao conceito de interface

EXEMPLO: MODELO DE AGENTE

Representação conceptual de agente



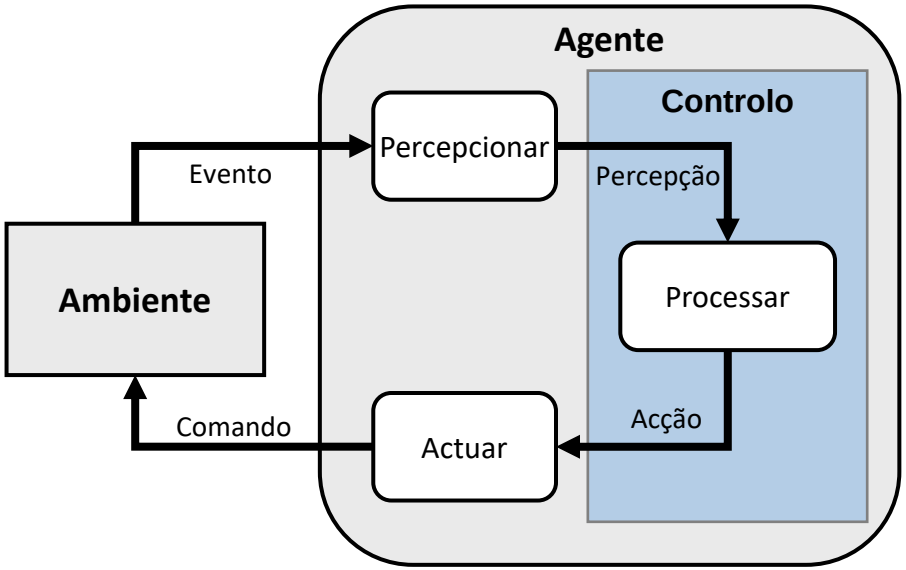
Modelo estrutural de agente na linguagem UML



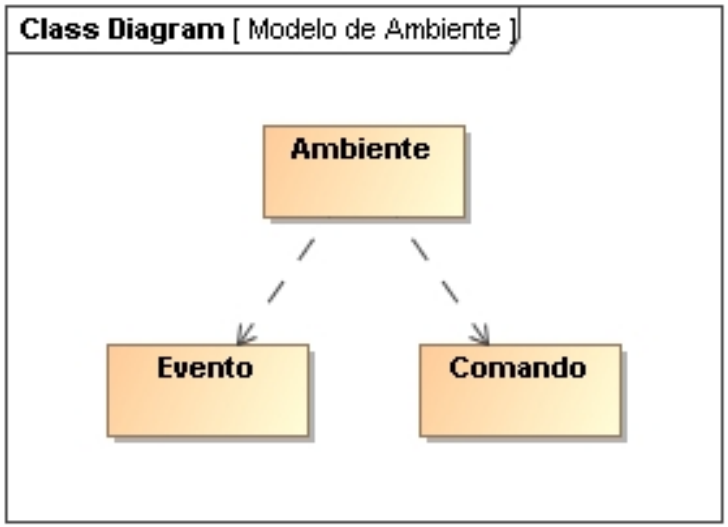
EXEMPLO: MODELO DE AMBIENTE

Representação conceptual da relação entre agente e ambiente

Para um ambiente onde é possível executar comandos e observar eventos

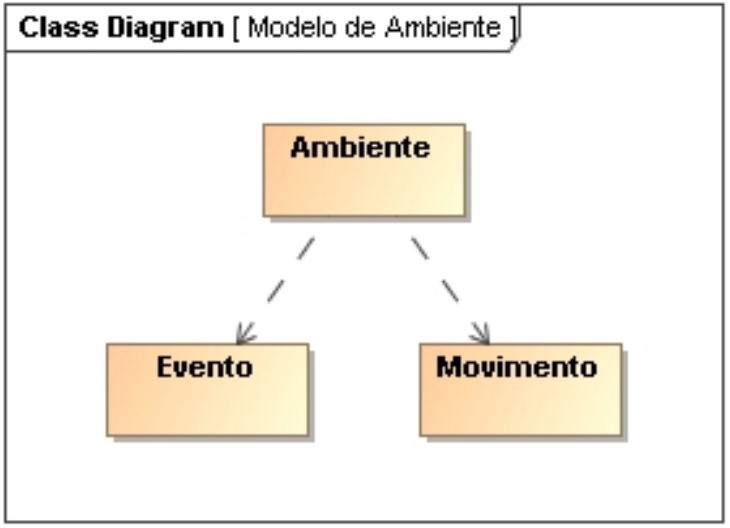


Modelo estrutural de ambiente na linguagem UML



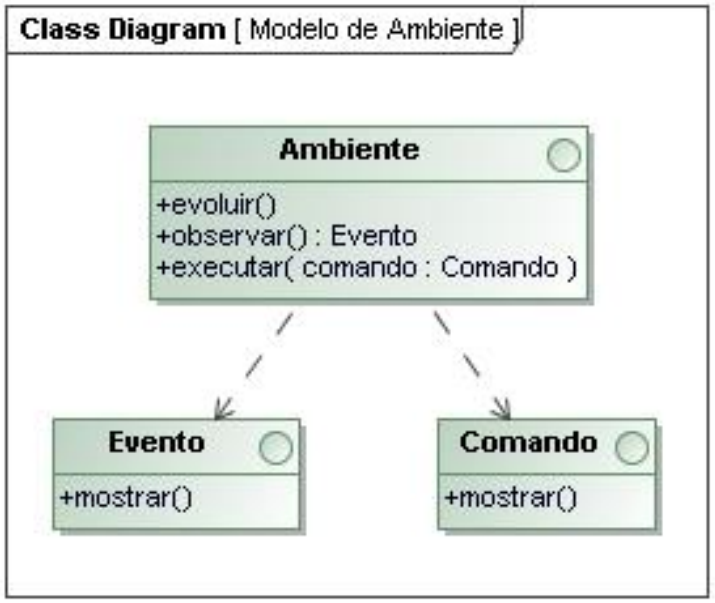
EXEMPLO: MODELO DE AMBIENTE

Modelo estrutural de ambiente
Modelo de domínio (representação geral dos conceitos do domínio do problema)



Modelo estrutural de ambiente
Detalhado com contratos funcionais (*interfaces*) de modo independente de possíveis implementações

Considerando um ambiente que evolui no tempo, onde é possível executar comandos e observar eventos, e onde é possível mostrar comandos e eventos

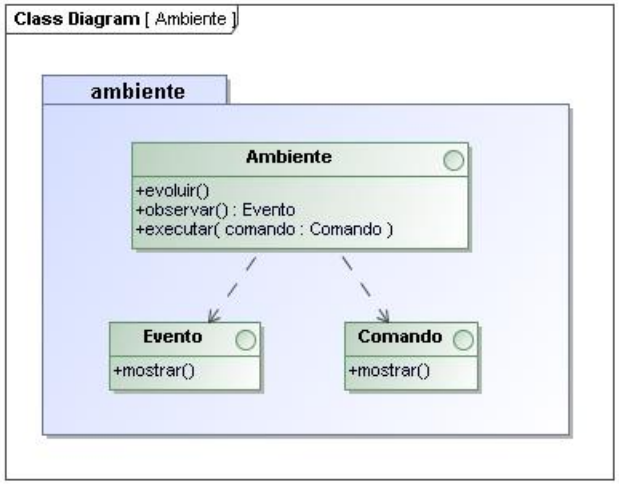


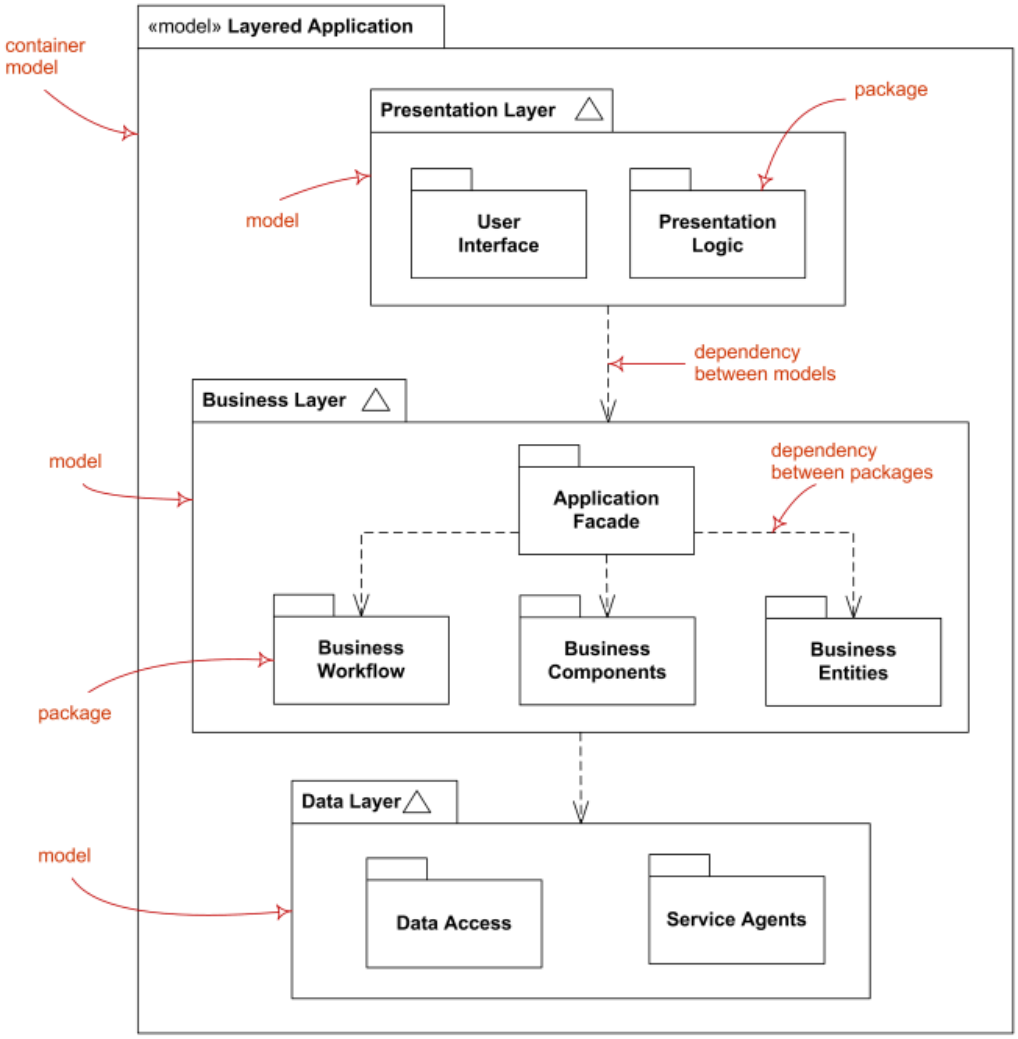
PASTAS (*PACKAGES*)

Organização estruturada dos elementos do modelo

- Gestão e organização de sistemas complexos
- Foco nas dependências
- Divisão de um sistema em subsistemas

Exemplo:



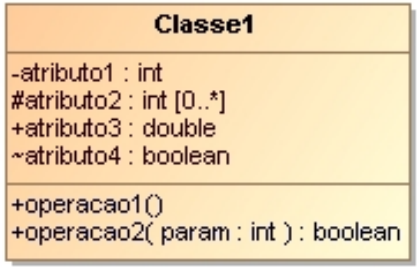


[uml-diagrams.org]

CONVERSÃO MODELO - IMPLEMENTAÇÃO

Classificadores base

Classe:

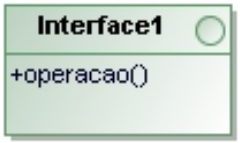



```
public class Classe1
{
    private int atributo1;
    protected int[] atributo2;
    public double atributo3;
    boolean atributo4;

    public void operacao1()
    {
    }

    public boolean operacao2(int param)
    {
        return false;
    }
}
```

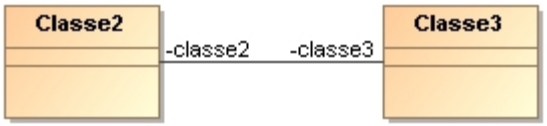
Interface:




```
public interface Interface1
{
    void operacao( );
}
```

CONVERSÃO MODELO - IMPLEMENTAÇÃO

Relações entre classes

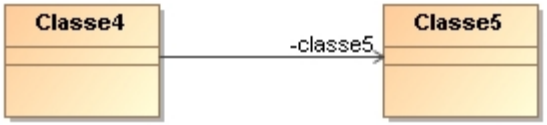



```
public class Classe2
{
    private Classe3 classe3;
}
```

```
public class Classe3
{
    private Classe2 classe2;
}
```

Numa associação, uma parte está estruturalmente associada a outra parte através de um dos seus atributos, ou seja, um dos seus atributos tem informação acerca de outra parte, por exemplo, uma classe tem um atributo cujo tipo é uma instância de outra classe

A parte que referencia deve ter um atributo correspondente à parte referenciada, o nome do atributo deve corresponder ao papel da parte referenciada na associação

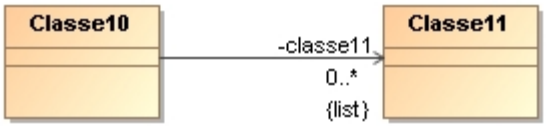



```
public class Classe4
{
    private Classe5 classe5;
}
```

```
public class Classe5
{

}
```

Se a multiplicidade for diferente de 1, deve ser definido um contentor de elementos correspondente à multiplicidade e eventuais restrições de dados definidas

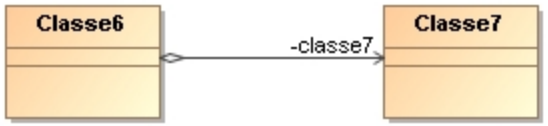



```
public class Classe10
{
    private java.util.List<Classe11> classe11;
}
```

CONVERSÃO MODELO - IMPLEMENTAÇÃO

Relações entre classes

Agregação:



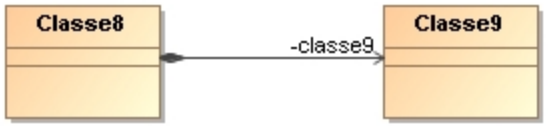
A agregação não requer a criação das partes agregadas quando a parte agregadora é criada

```
public class Classe6
{
    private Classe7 classe7;
}
```

```
public class Classe7
{

}
```

Composição:



A composição requer a criação das partes agregadas quando a parte composta é criada

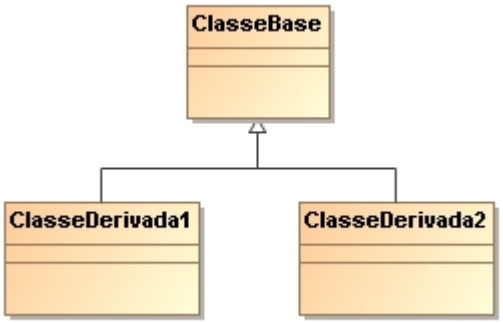
```
public class Classe8
{
    private Classe9 classe9 = new Classe9();
}
```

```
public class Classe9
{

}
```

CONVERSÃO MODELO - IMPLEMENTAÇÃO

Herança (generalização/especialização)




```
public class ClasseBase
{

}
```

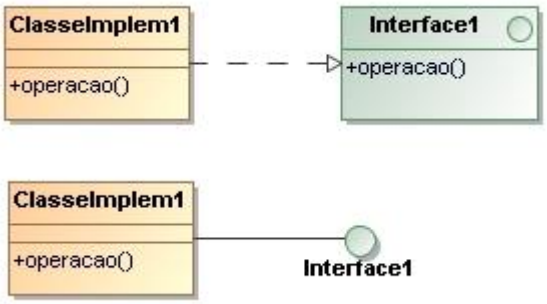
```
public class ClasseDerivada1 extends ClasseBase
{

}
```

```
public class ClasseDerivada2 extends ClasseBase
{

}
```

Realização de interfaces




```
public interface Interface1
{
    void operacao();
}
```

```
public class ClasseImplem1 implements Interface1
{
    void operacao()
    {

    }
}
```

BIBLIOGRAFIA

[Pressman, 2003]
R. Pressman, *Software Engineering: a Practitioner's Approach,* McGraw-Hill, 2003.

[Booch et al., 1998]
G. Booch, J. Rumbaugh, I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1998.

[Miles & Hamilton, 2006]
R. Miles, K. Hamilton, *Learning UML 2.0*, O'Reilly, 2006.

[Eriksson et al., 2004]
H. Eriksson, M. Penker, B. Lyons, D. Fado, *UML 2 Toolkit*, Wiley, 2004.

[Douglass, 2009]
B. Douglass, *Real-Time Agility: The Harmony/ESW Method for Real-Time and Embedded Systems Development*, Addison-Wesley, 2009.

[SRC, 2015]
Semiconductor Research Corporation, *Rebooting the IT Revolution*, 2015.

[Korf, 1980]
R. Korf, Toward a model of representation changes, Artificial Intelligence, Volume 14, Issue 1, 1980.