ENGENHARIA DE SOFTWARE
INTRODUÇÃO

Luís Morgado

2024

ARQUITECTURA DE SOFTWARE

Vertentes principais para abordar o problema da complexidade: *métricas*, *princípios* e *padrões*

- **MÉTRICAS**
- **PRINCÍPIOS**
- **PADRÕES**

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)

MÉTRICAS DE ARQUITECTURA

Definem *medidas de quantificação* da arquitectura de um software *indicadoras da qualidade* dessa arquitectura

- **ACOPLAMENTO**
  - Grau de interdependência entre subsistemas
- **COESÃO**
  - Nível coerência funcional de um subsistema/módulo (até que ponto esse módulo realiza uma única função)
- **SIMPLICIDADE**
  - Nível de facilidade de compreensão/comunicação da arquitectura
- **ADAPTABILIDADE**
  - Nível de facilidade de alteração da arquitectura para incorporação de novos requisitos ou de alterações nos requisitos previamente definidos

ACOPLAMENTO

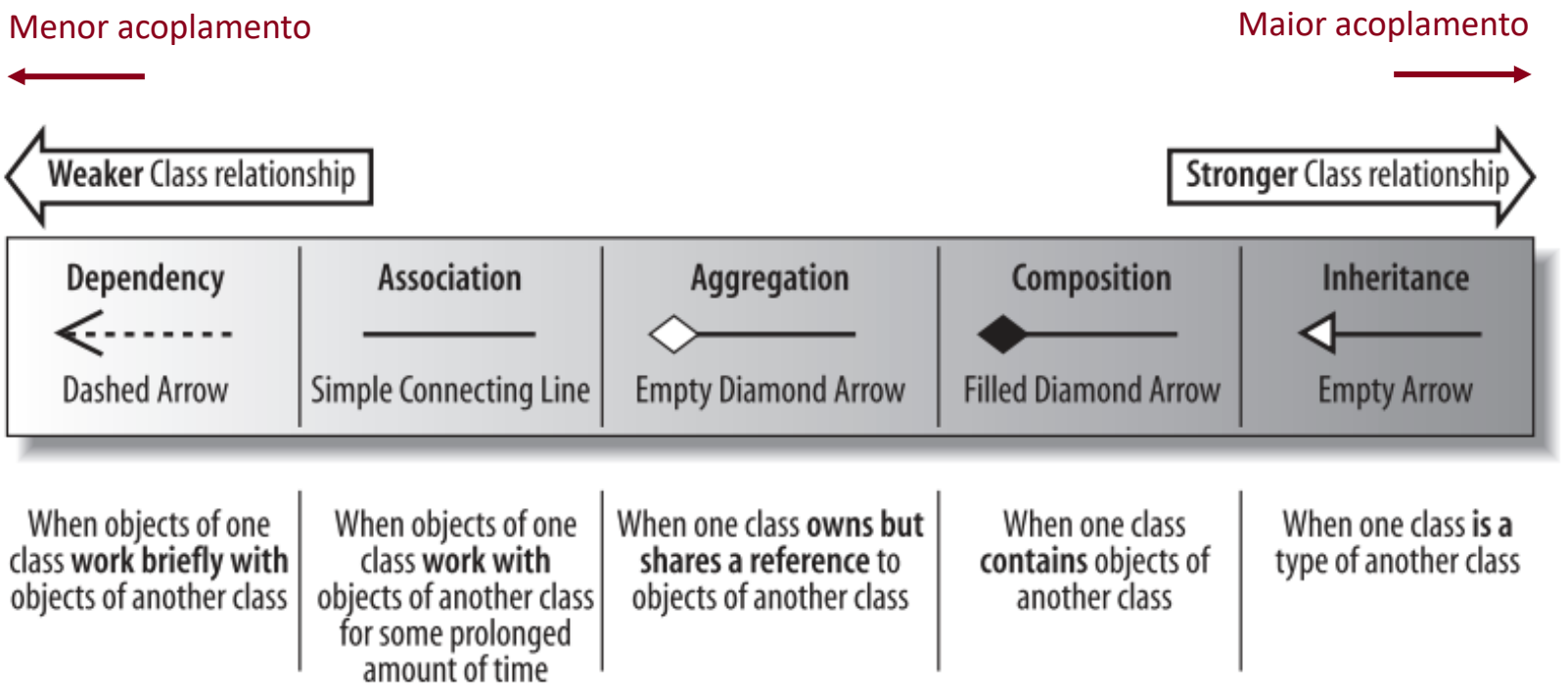
- O conceito de *acoplamento* refere-se ao **grau de dependência** entre diferentes partes de um sistema, sendo tanto maior quanto maior a dependência entre as partes de um sistema
- O acoplamento pode ser medido utilizando diferentes métricas, como o número de **associações entre elementos** ou o grau de complexidade de uma interface
- O objetivo é **criar software com o menor acoplamento possível**, de modo a reduzir a complexidade e a facilitar a sua manutenção e evolução
- A **modularidade** e o **encapsulamento** são princípios de arquitectura de software que contribuem para a redução do acoplamento

- **ACOPLAMENTO**
  - Grau de interdependência entre partes de um sistema
  - Característica **inter-modular**
    - Expressa relações entre partes (módulos)
  - Deve ser minimizado
    - Redução de complexidade
    - Facilidade de manutenção e evolução

ACOPLAMENTO EM MODELOS DE ESTRUTURA

Linguagem UML

Relações entre classes e nível de acoplamento



[Miles & Hamilton, 2006]

COESÃO

- O conceito de *coesão* refere-se à forma como os elementos de um sistema estão **agrupados de forma coerente** entre si, será tanto maior quando mais relacionados entre si forem os elementos agrupados em cada módulo
- A coesão pode ser medida utilizando diferentes critérios, por exemplo, por **tipo de função das partes agrupadas**, sendo designada neste caso por *coesão funcional*
- O objetivo é criar **software que seja claro e fácil de entender, manter e evoluir**, facilitando a reutilização e aumentando assim a eficiência do desenvolvimento
- A **modularidade** e a **factorização** são princípios de arquitectura de software que contribuem para o aumento da coesão

- **COESÃO**
  - Nível de coerência das partes agrupadas num módulo ou subsistema
    Característica **intra-modular**
    - Expressa relações interiores aos módulos (agrupamentos)
  - Deve ser maximizada
    - Redução de complexidade
    - Facilidade de evolução, manutenção e reutilização

PRINCÍPIOS DE ARQUITECTURA

Definem *meios orientadores* da concepção de arquitectura de software, no sentido de garantir a qualidade da arquitectura produzida, nomeadamente, no que se refere à *minimização do acoplamento*, à *maximização da coesão* e à *gestão da complexidade*

- **ABSTRACÇÃO**
- **MODULARIZAÇÃO**
  - **DECOMPOSIÇÃO**
  - **ENCAPSULAMENTO**
- **FACTORIZAÇÃO**

→ **- ACOPLAMENTO**

+ COESÃO

↓

GESTÃO DA COMPLEXIDADE

MODULARIZAÇÃO

O conceito de *modularização* refere-se à capacidade de **organização de um sistema em partes coesas, ou módulos**, que podem ser interligados entre si para produzir a função do sistema, está relacionado com dois aspectos principais, *decomposição* e *encapsulamento*

- **DECOMPOSIÇÃO**
  - De um sistema em partes coesas
    - Para sistematizar interacções
    - Para lidar com a explosão combinatória
  - **FACTORIZAÇÃO**
    - Eliminação de redundância
    - Garantia de consistência
- **ENCAPSULAMENTO**
  - **Isolamento dos detalhes internos** das partes de um sistema em relação ao exterior
    - Para reduzir dependências (interacções)
    - Relacionar estrutura e função no contexto de uma parte
    - Acesso exclusivo através das *interfaces* disponibilizadas
  - **INTERFACES**
    - **Contractos funcionais** para interação com o exterior

FACTORIZAÇÃO

O conceito de *factorização* refere-se à decomposição das partes de um sistema de modo a **eliminar redundância** (partes repetidas), relaciona-se com o conceito matemático correspondente de decomposição de uma expressão em *factores* (partes de um produto), por exemplo, $ax + bx = (a + b)x$

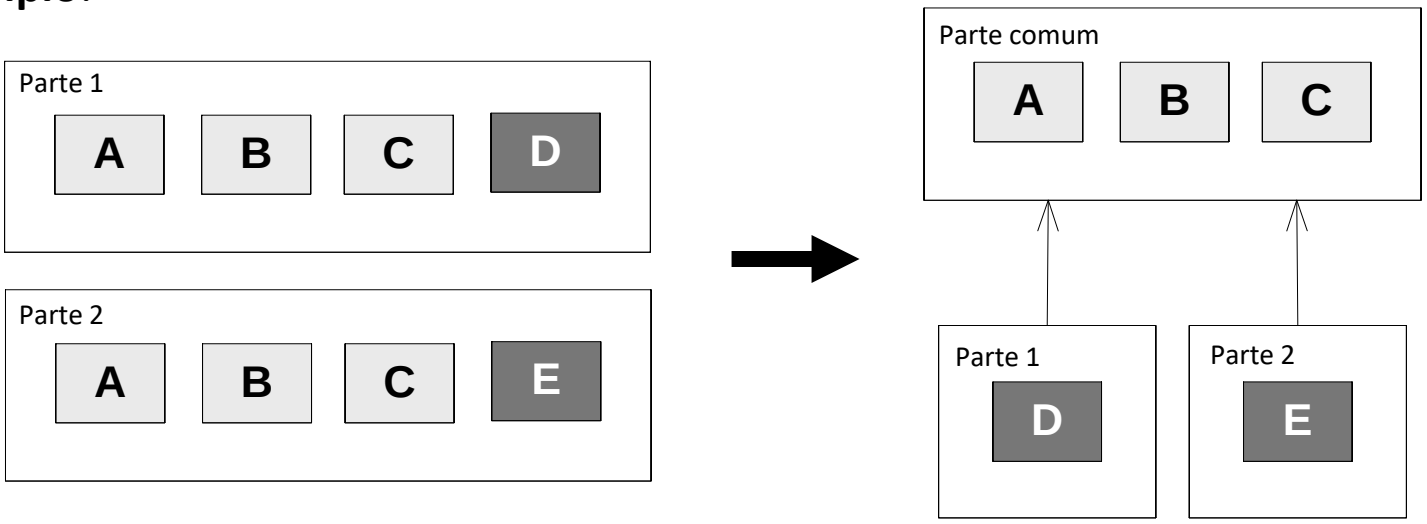
REDUNDÂNCIA

Existência de partes repetidas num sistema, é uma das principais causas de anomalias e de complexidade desorganizada no desenvolvimento de software

Redução de redundância por factorização

As partes repetidas são eliminadas, as partes mantidas são partilhadas

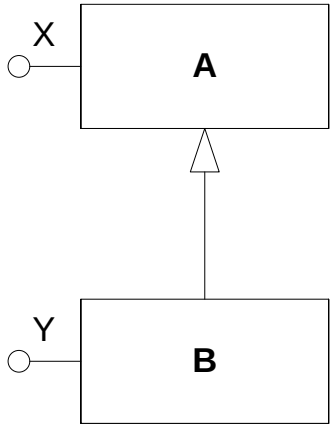
Exemplo:



MECANISMOS DE FACTORIZAÇÃO

HERANÇA

- Factorização estrutural
- B **é** A
- Nível de **acoplamento alto**

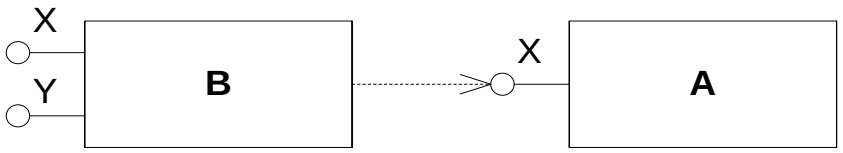


B **é** A

B disponibiliza interfaces X e Y herdando X de A

DELEGAÇÃO

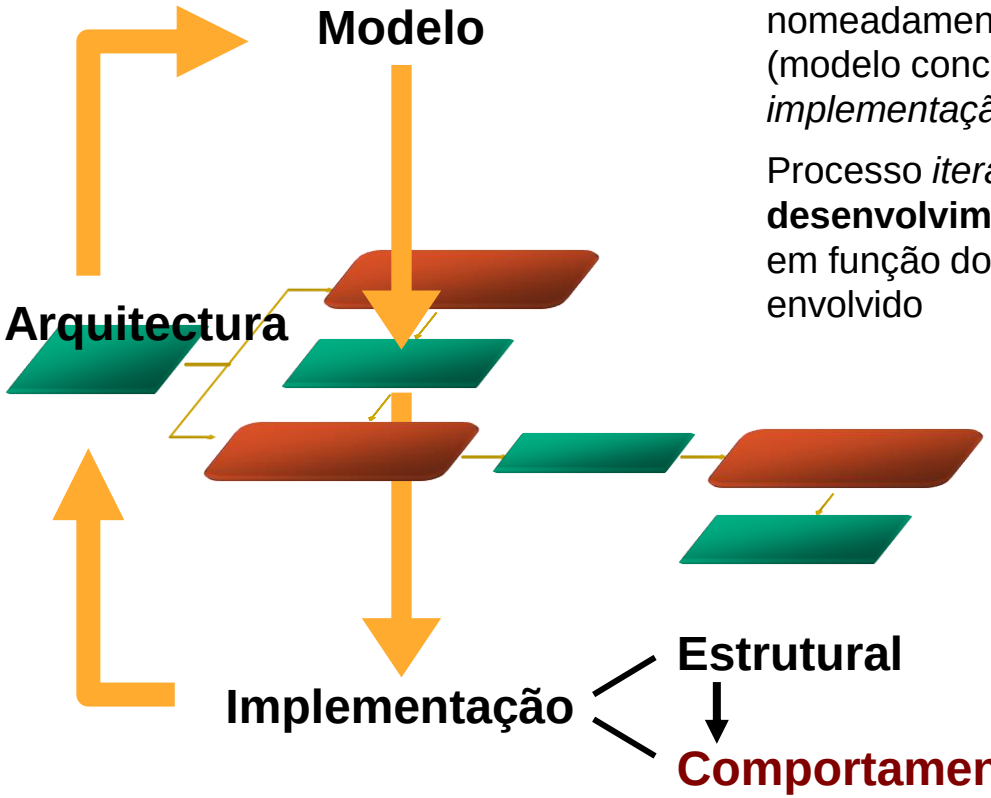
- Factorização funcional
- B **utiliza** A
- Nível de **acoplamento baixo**
- Acoplamento pode variar **dinamicamente**



B **utiliza** A

B disponibiliza interfaces X e Y utilizando A para delegar a funcionalidade de X

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE



O *processo de desenvolvimento* de software consiste na **criação da organização de um sistema de forma progressiva**, através de diferentes níveis de abstracção, nomeadamente *modelo* (conceptual), *arquitectura* (modelo concreto, orientado para a implementação) e *implementação*

Processo *iterativo* em que as diferentes **actividades de desenvolvimento são alternadas ao longo do tempo** em função do conhecimento e do nível de detalhe envolvido

A implementação deve ser realizada em duas etapas principais:

- **Implementação estrutural**
  - Realização de código relativo à estrutura do sistema (*código estrutural*)
- **Implementação comportamental**
  - Realização de código relativo ao comportamento do sistema (*código comportamental*)

Criação da organização de um sistema de forma incremental de modo a facilitar a gestão da complexidade

PROCESSO ITERATIVO

MODELOS DE COMPORTAMENTO

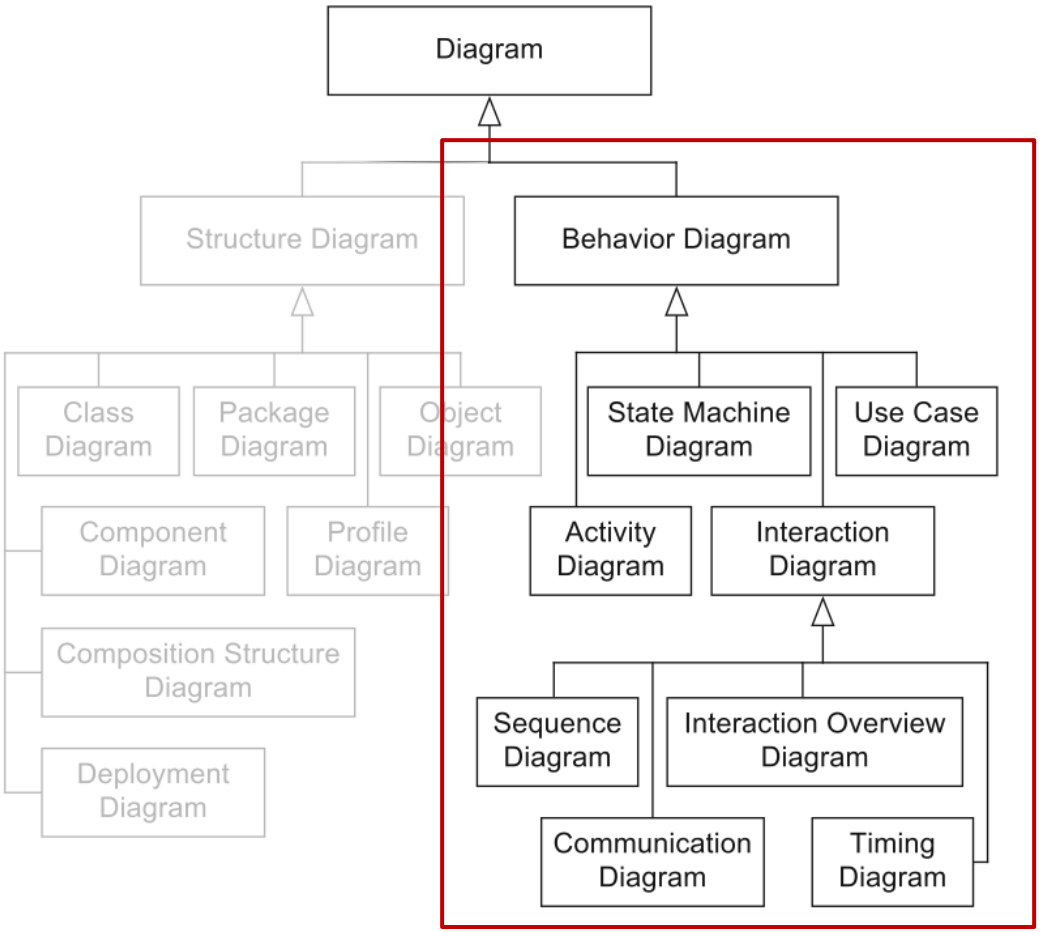
O **comportamento** de um sistema corresponde à forma como o sistema age perante as suas entradas e o seu estado interno

Duas perspectivas principais de modelação:

- **Interacção**
  - Descreve a forma como as partes de um sistema interagem entre si e com o exterior para produzir o comportamento do sistema
- **Dinâmica**
  - Evolução no tempo
  - Descreve os estados que um sistema pode assumir e a forma como eles evoluem ao longo do tempo, determinando o comportamento do sistema

Linguagem UML

Perspectiva Comportamental

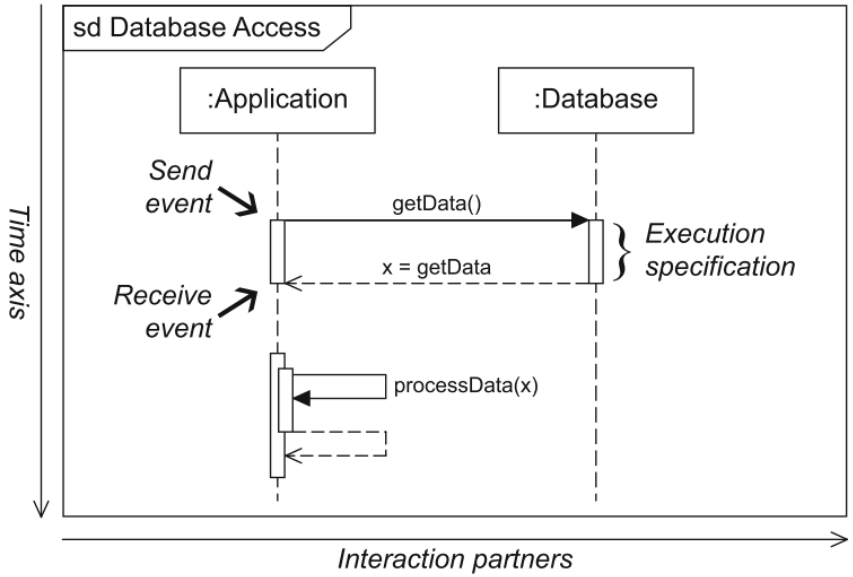


[Seidl, 2012]

MODELOS DE INTERACÇÃO

REPRESENTAÇÃO DE COMPORTAMENTO

- **Diagramas de sequência**
  - Descrevem a comunicação entre partes do sistema e/ou com o exterior
  - Ênfase na sequência temporal de interacção
- **Organização bidimensional**
  - **Tempo** – vertical
  - **Estrutura** (partes) – horizontal
- **Elementos de modelação**
  - **Linha de vida** (*lifeline*)
    - Representa evolução temporal
  - **Foco de activação** (*activation bar*)
    - Representam execução de operações
  - **Mensagem**
    - Partes trocam mensagens
  - **Operador**
    - Fragmento de interacção com semântica específica

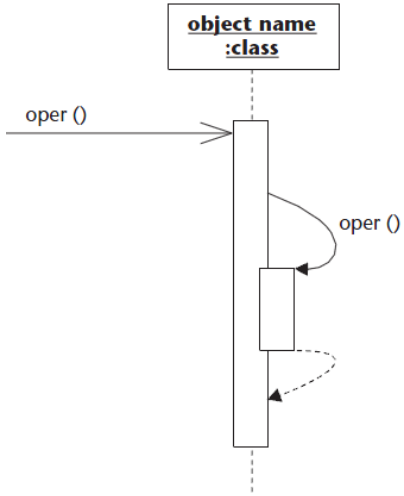


[Seidl, 2012]

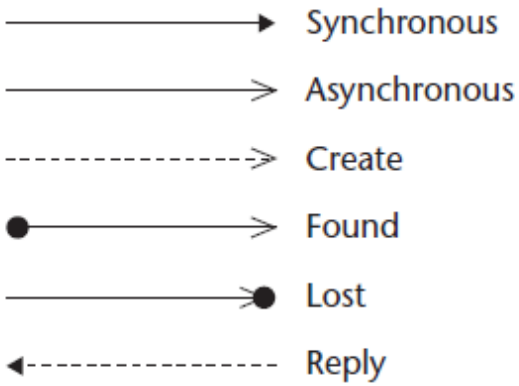
DIAGRAMAS DE SEQUÊNCIA

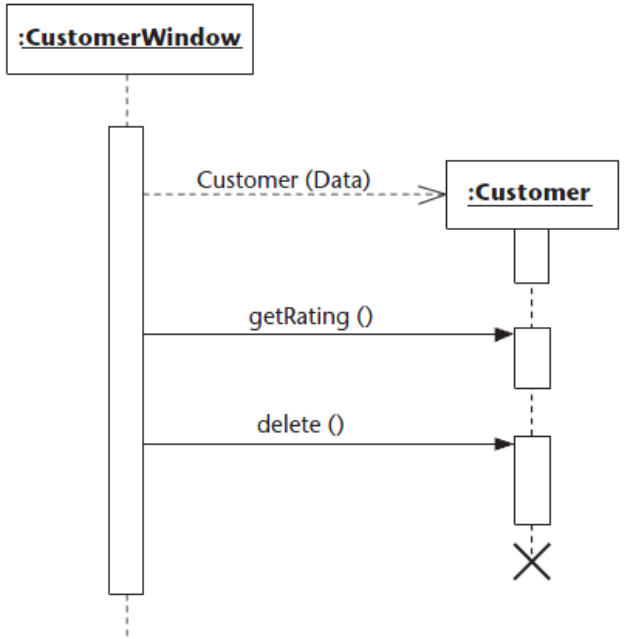
- **Mensagens**
  - **Sintaxe**
    - [attribute=] message-name [(argument)] [:return-value]

[Condição] Mensagem



- **Tipos de mensagens**

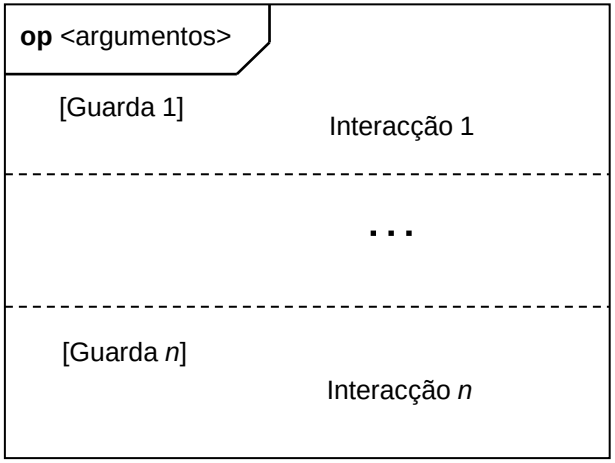


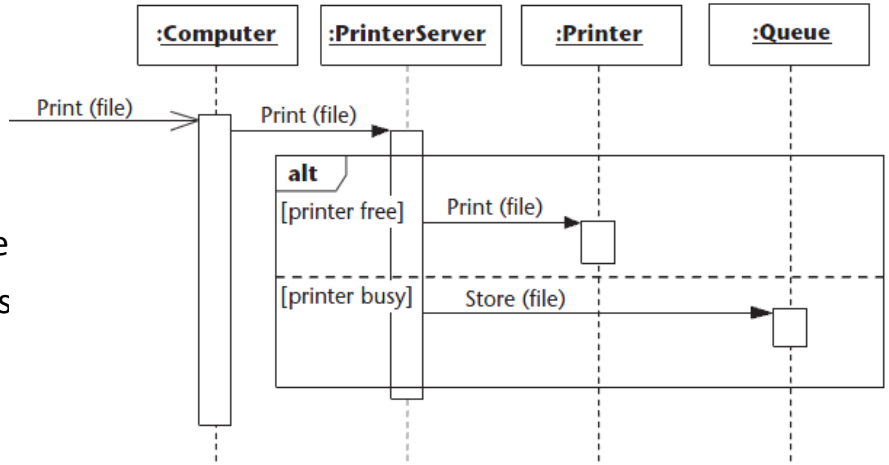


DIAGRAMAS DE SEQUÊNCIA

- **Operador**
  - Fragmento de interacção com semântica específica

- **Tipos de operadores**
  - **ref**: referência a fragmento de interacção
  - **loop**: repetição de fragmento de interacção
  - **break**: fim de repetição de fragmento de interacção
  - **alt**: selecção de fragmento de interacção
  - **par**: regiões concorrentes (paralelas)
  - **assert**: fragmento de interacção requerido
  - **opt**: fragmento de interacção opcional
  - **neg**: especificação negativa (não pode acontece
  - **region**: região crítica (não são permitidas outras mensagens)





EXEMPLO: JOGO

Pretende-se implementar um jogo com uma personagem virtual que interage com um jogador humano.

O jogo consiste num ambiente onde a personagem tem por objectivo registar a presença de animais através de fotografias.

Quando o jogo se inicia a personagem fica numa situação de procura de animais. Quando detecta algum ruído aproxima-se e fica em inspecção da zona, procurando a fonte do ruído. Quando volta a haver silêncio a personagem volta a uma situação de procura de animais. Quando detecta um animal a personagem aproxima-se e fica em observação. Caso o animal continue presente, a personagem observa o animal e fica preparada para o registo, se ocorrer a fuga do animal a personagem fica em inspecção da zona, à procura de uma fonte de ruído. Na situação de registo, se o animal continuar presente fotografa-o, caso ocorra a fuga do animal ou a personagem tenha conseguido uma fotografia do animal, a personagem fica novamente numa situação de procura.

A interacção com o jogador é realizada em modo de texto.

EXEMPLO: ARQUITECTURA DO JOGO

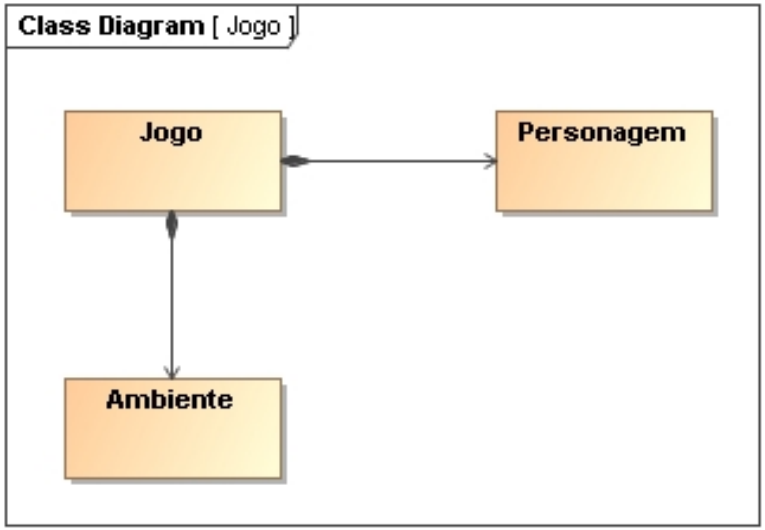
O **jogo** consiste num **ambiente** onde a **personagem** tem por objectivo registar a presença de animais através de fotografias

Conceitos do domínio do problema

Jogo

- Ambiente
- Personagem

Modelo de estrutura



Comportamento?

EXEMPLO: ARQUITECTURA DO JOGO

Modelo de interacção: Iniciar jogo

O **jogo** consiste num **ambiente** onde a **personagem** tem por objectivo registar a presença de animais através de fotografias.

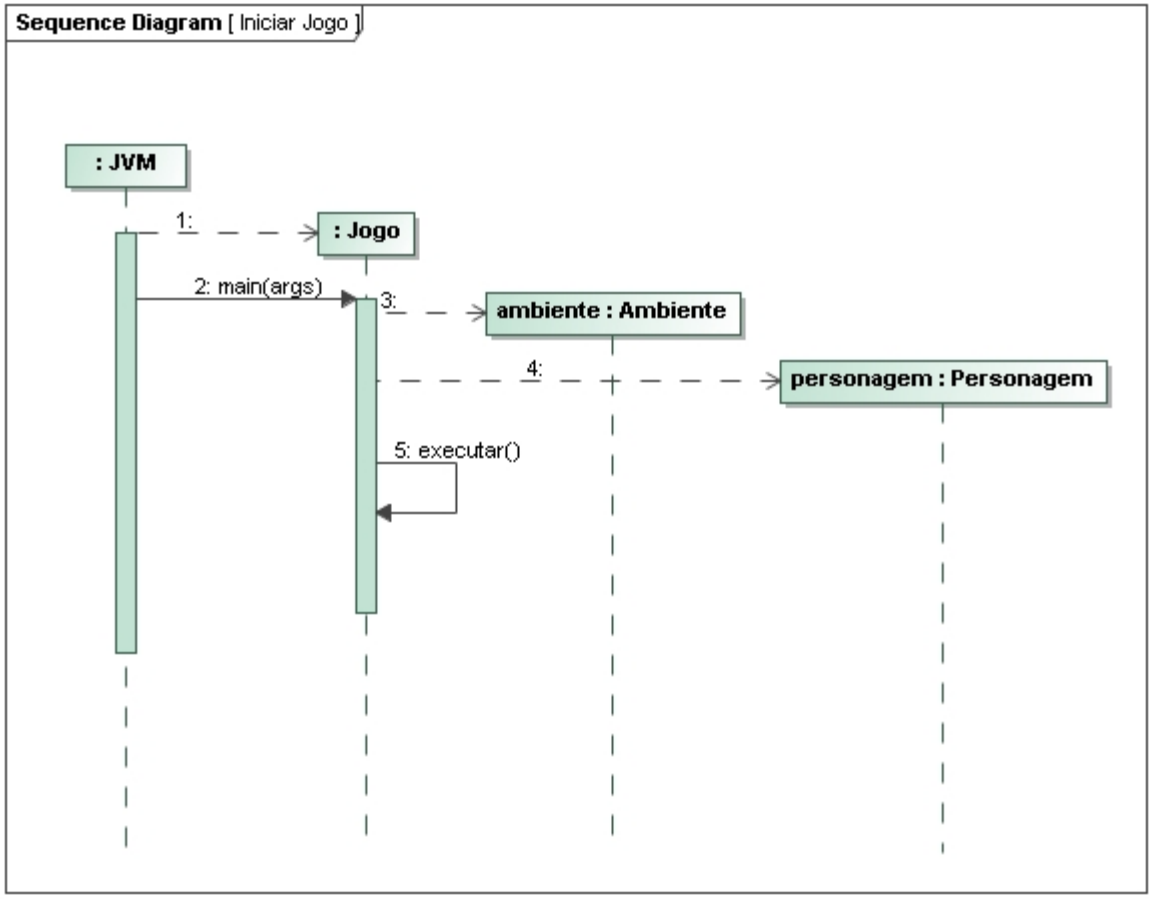
Conceitos do domínio do problema

Jogo

- Ambiente
- Personagem

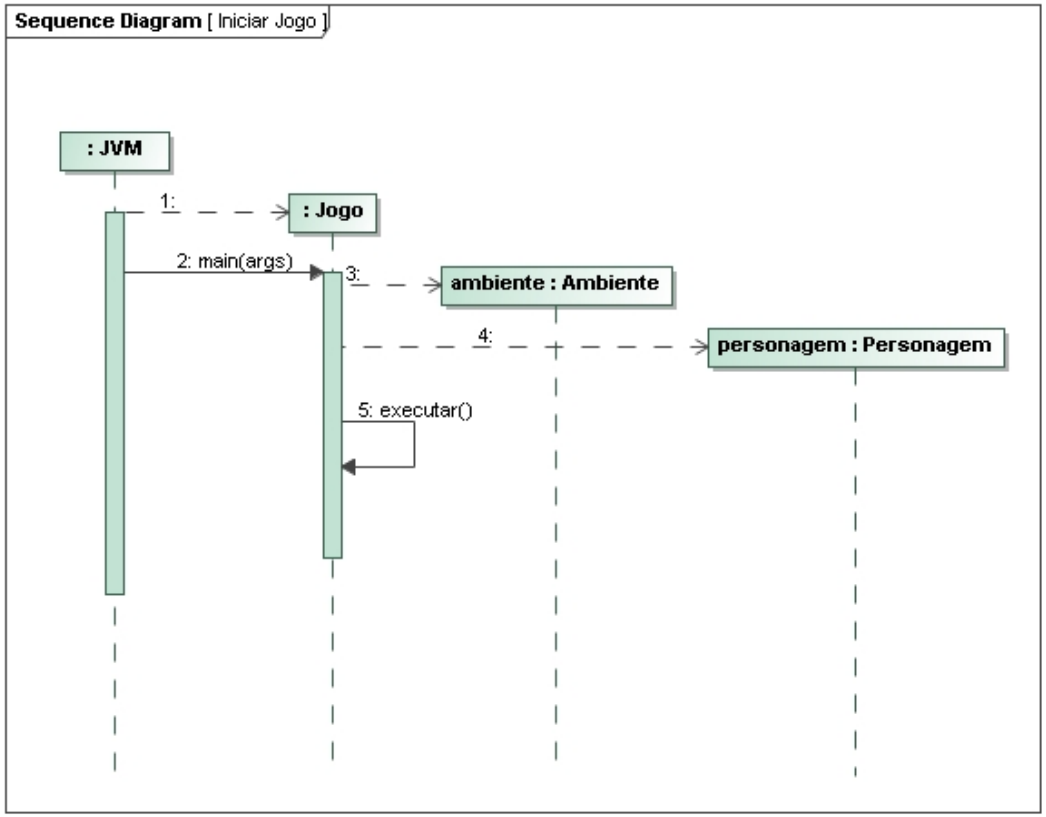
Ambiente de execução

JVM: Java Virtual Machine

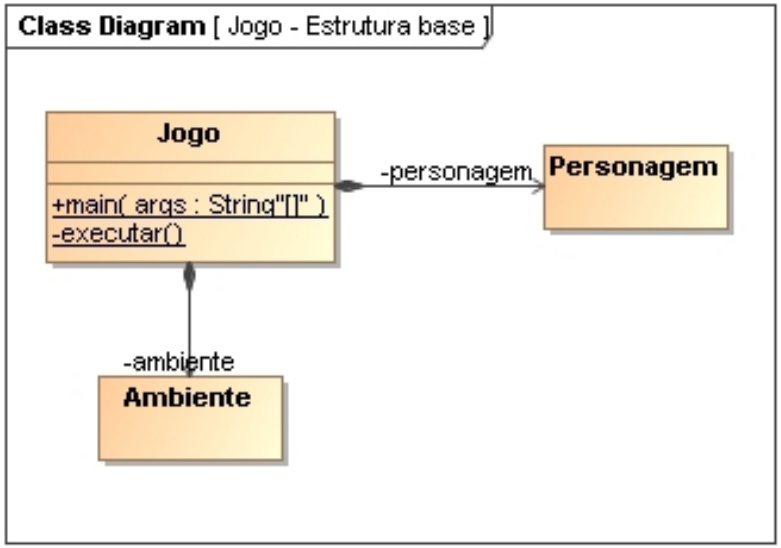


EXEMPLO: ARQUITECTURA DO JOGO

Modelo de interacção
(Diagrama de sequência)



Modelo de estrutura
(Diagrama de classes)



BIBLIOGRAFIA

[Watson, 2008]

Andrew Watson, *Visual Modeling: past, present and future*, OMG, 2008.

[Meyer, 1997]

B. Meyer, *UML: The Positive Spin*, American Programmer - Special UML issue, 1997.

[Yelland et al., 2002]

Yelland, M. J., B. I. Moat, R. W. Pascal and D. I. Berry, *CFD model estimates of the airflow over research ships and the impact on momentum flux measurements*, Journal of Atmospheric and Oceanic Technology, 19(10), 2002.

[Selic, 2003]

B. Selic, *Brass bubbles: An overview of UML 2.0*, Object Technology Slovakia, 2003.

[Graessle, 2005]

P. Graessle, H. Baumann, P. Baumann, *UML 2.0 in Action*, Packt Publishing, 2005.

[Eriksson et al., 2004]

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

[Seidl, 2012]

UML Classroom: An Introduction to Object-Oriented Modeling, M. Seidl *et al.,* Springer, 2012

[Douglass, 2006]

B. Douglass, *Real-Time UML*, Telelogic, 2006.