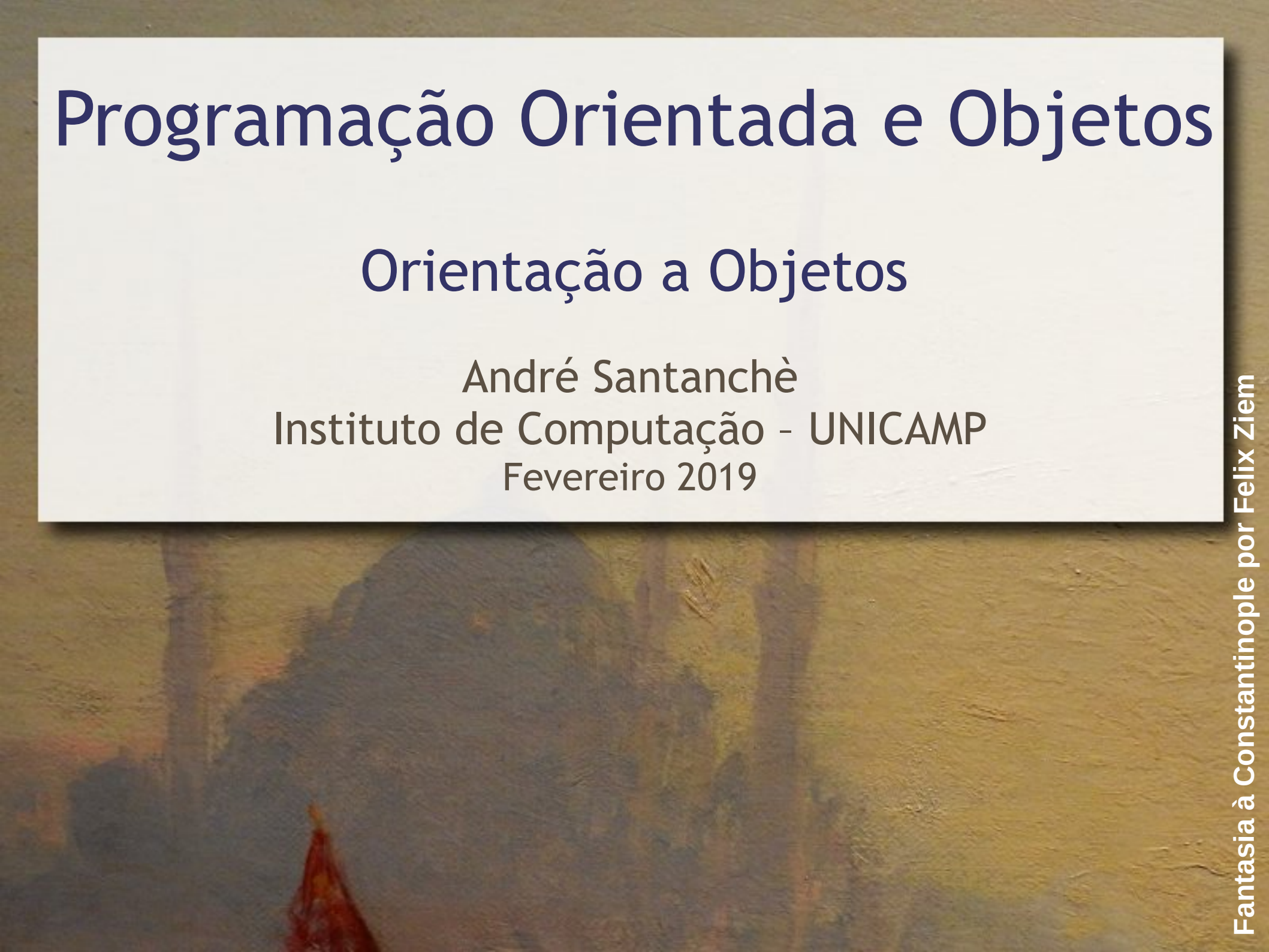
Programação Orientada e Objetos

Orientação a Objetos

André Santanchè
Instituto de Computação – UNICAMP
Fevereiro 2019


Fantasia à Constantinople por Felix Ziem

Exercício 1
Financiamento

Financiamento – Parte 1
Problema

- Uma associação de bairro realiza empréstimos populares a juros baixos.
- Todo o controle de um empréstimo é feito em papel.
- No último ano a procura foi tão alta que a associação não está dando conta de controlar tantos empréstimos em papel.

Financiamento – Parte 1
Problema

- Você foi contratado pela associação para resolver este problema do controle de empréstimos.
- Enumere em alto nível as ações que você realizaria do início até o final do processo uma vez contratado.

Ciclo de Desenvolvimento

- Levantamento de requisitos
- Projeto
- Implementação
- Avaliação

Levantamento de Requisitos

By University of Nottingham

Giving direction by Batoul Zaiter (C.O.D. Library) on Flickr

Universo de Discurso ou Mini-mundo

- “Um banco de dados representa algum aspecto do mundo real, às vezes chamado de **mini-mundo** ou de **universo de discurso** (UoD – Universe of Discourse).”

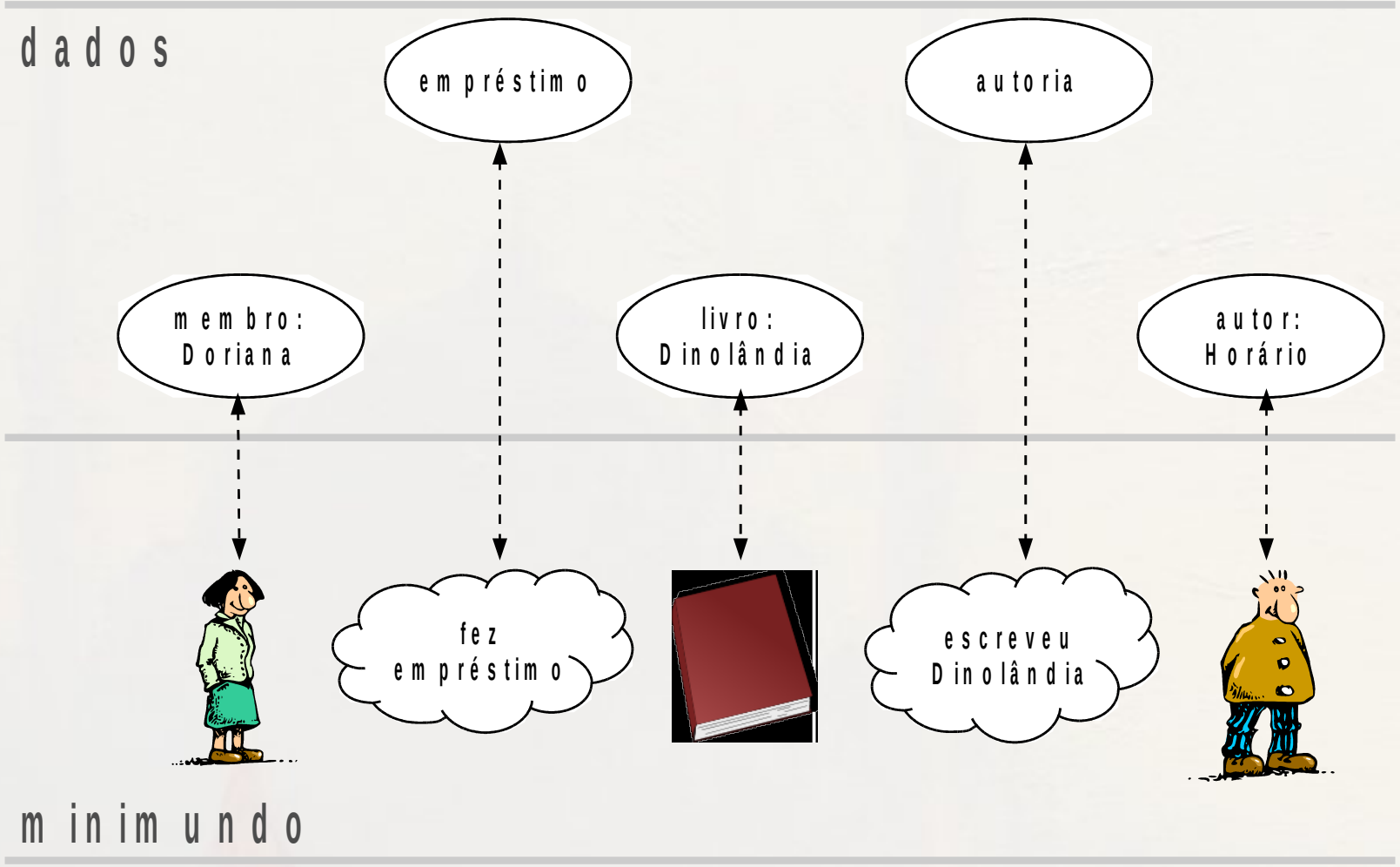
(Elmasri & Navathe, 2011)

Universo de Discurso ou Mini-mundo

- Recorte do mundo real a ser representado

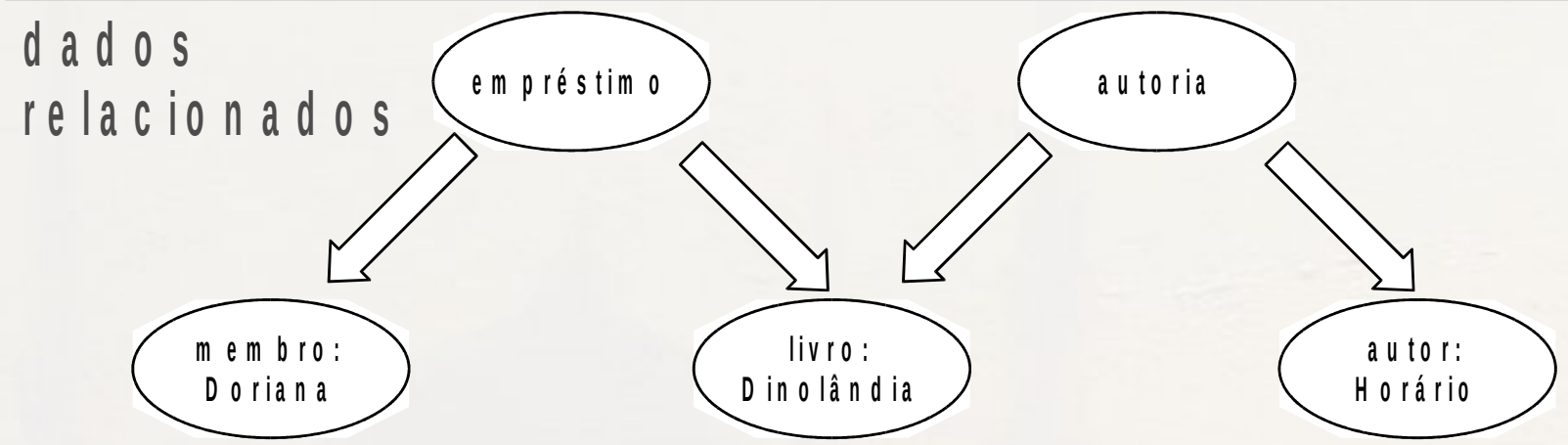
Dados

- Fatos registrados – significado implícito



Banco de Dados

- Coleção de dados relacionados



Abstração

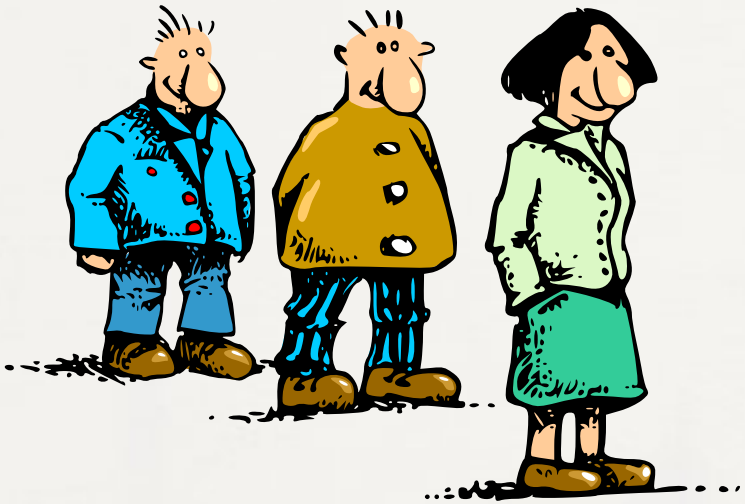
Problema x Abstração

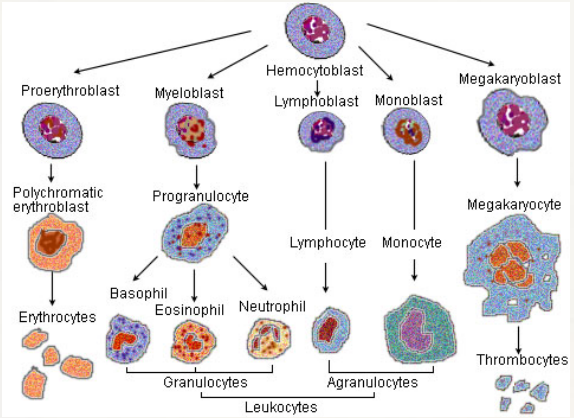
- “Para resolver um problema é necessário escolher uma abstração da realidade” (Almeida, 2010)

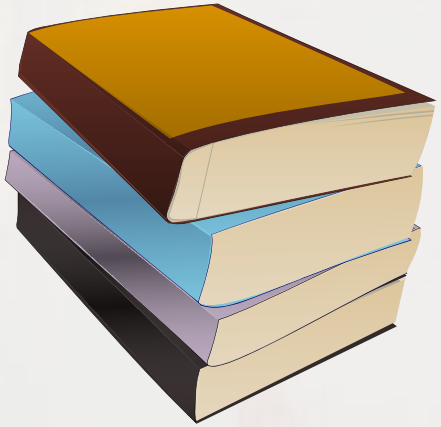
Abstração

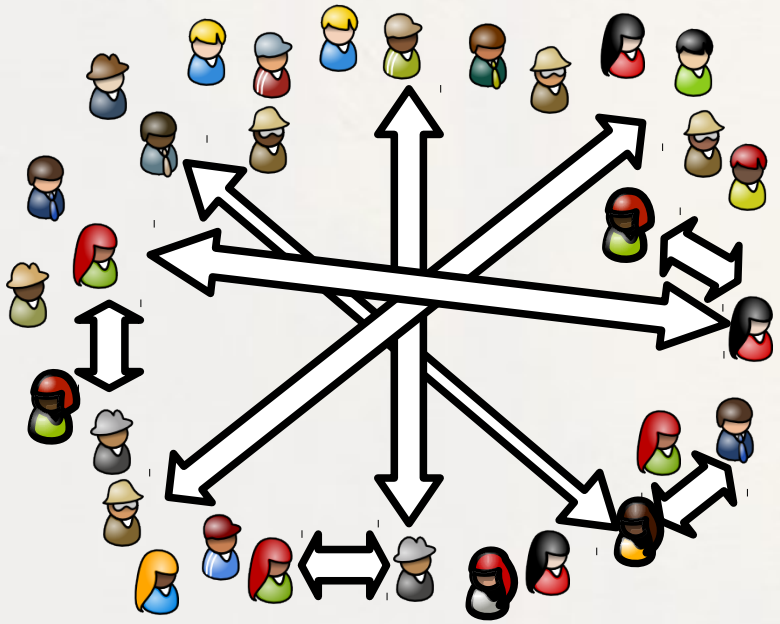
- “**processo mental** que consiste em **escolher ou isolar um aspecto** determinado de um estado de coisas relativamente complexo, a fim de simplificar a sua avaliação, classificação ou para permitir a comunicação do mesmo” (Houaiss, 2006)
- Abstrações ajudam a gerenciar a complexidade do software (Shaw, 1984)

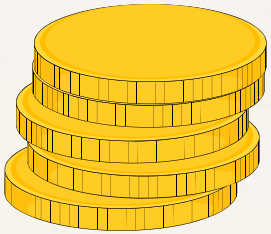
Abstrações do Dia a Dia











Objetos em Engenharia de Software

Como Modelamos o Mundo?

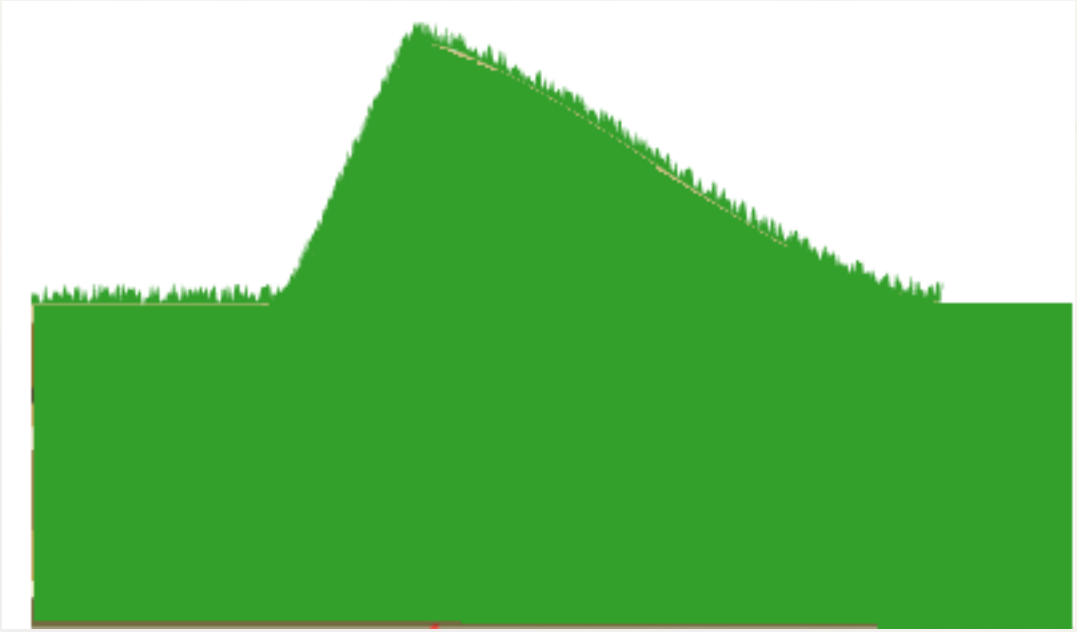
Ilusão dos Objetos

Intuitivo

Fantasia à Constantinople por Felix Ziem

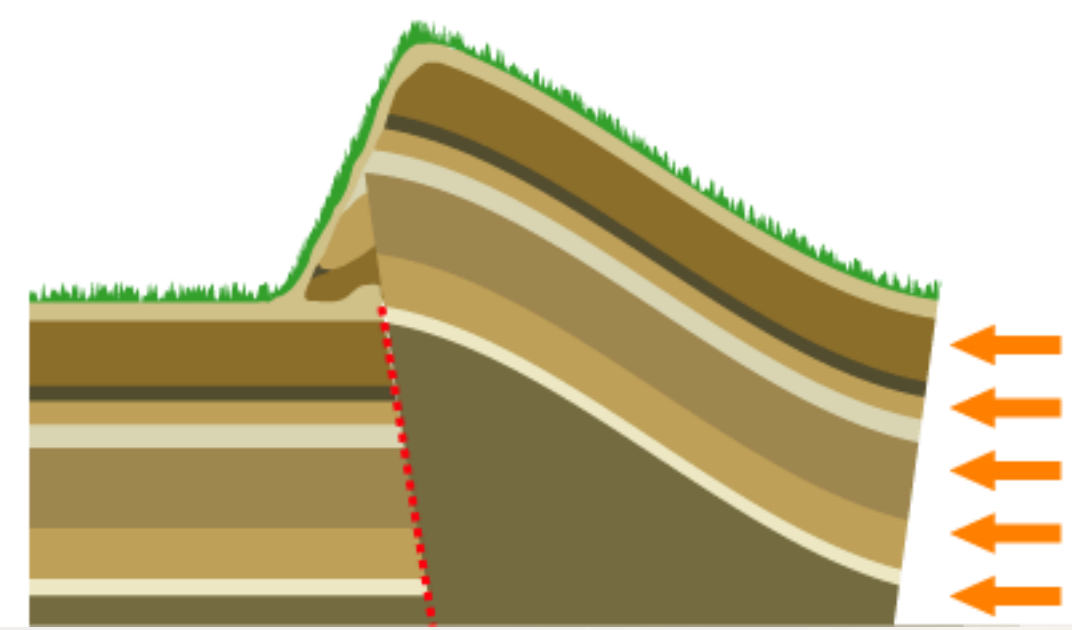
Objetos

- Montanha



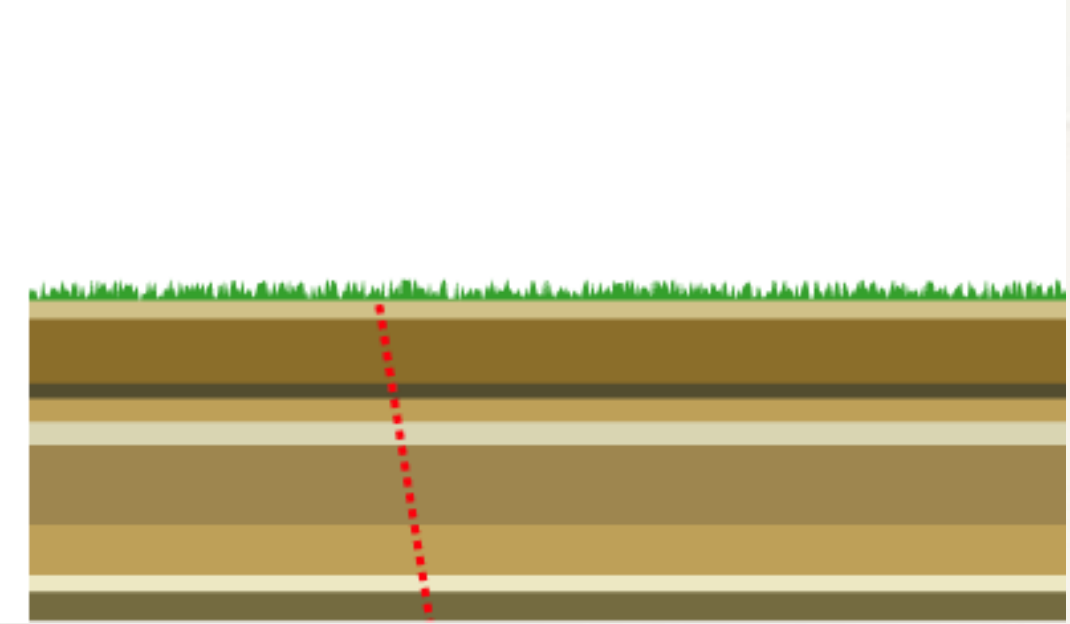
Objetos

- Montanha



Objetos

- Montanha



Mountain by reverse fault by Takami Torao

Objetos

- Estação rodoviária

Noção de Objeto

- Psicologia do desenvolvimento:
  - Quando crianças representam objetos como entidades permanentes?
  - Que persistem:
    - Através do tempo e espaço
    - À oclusão

(Santos & Hood, 2009)

Noção de Objetos

- Objetos permanecem?
  - “Of course, the concept of object permanence itself is really a misnomer, as all objects comprise energy in continuous states of change.” (Santos & Hood, 2009)

Noção de Objetos

- Objetos necessários
  - “One of the most functionally relevant aspects of physical objects is the fact that they persist—standardly speaking, objects do not go in and out of existence and, thus, it is important that an organism be able to represent their continued presence even when they cannot be directly perceived or apprehended.” (Santos & Hood, 2009)

Noção de Objetos

- Existência independente do observador
  - “[...] nervous systems were developed via natural selection to represent objects so that organisms may interact with the external world in an adaptive way, and thus, brains are built to capture what is functionally relevant about objects.” (Santos & Hood, 2009)

Formalizando Objetos

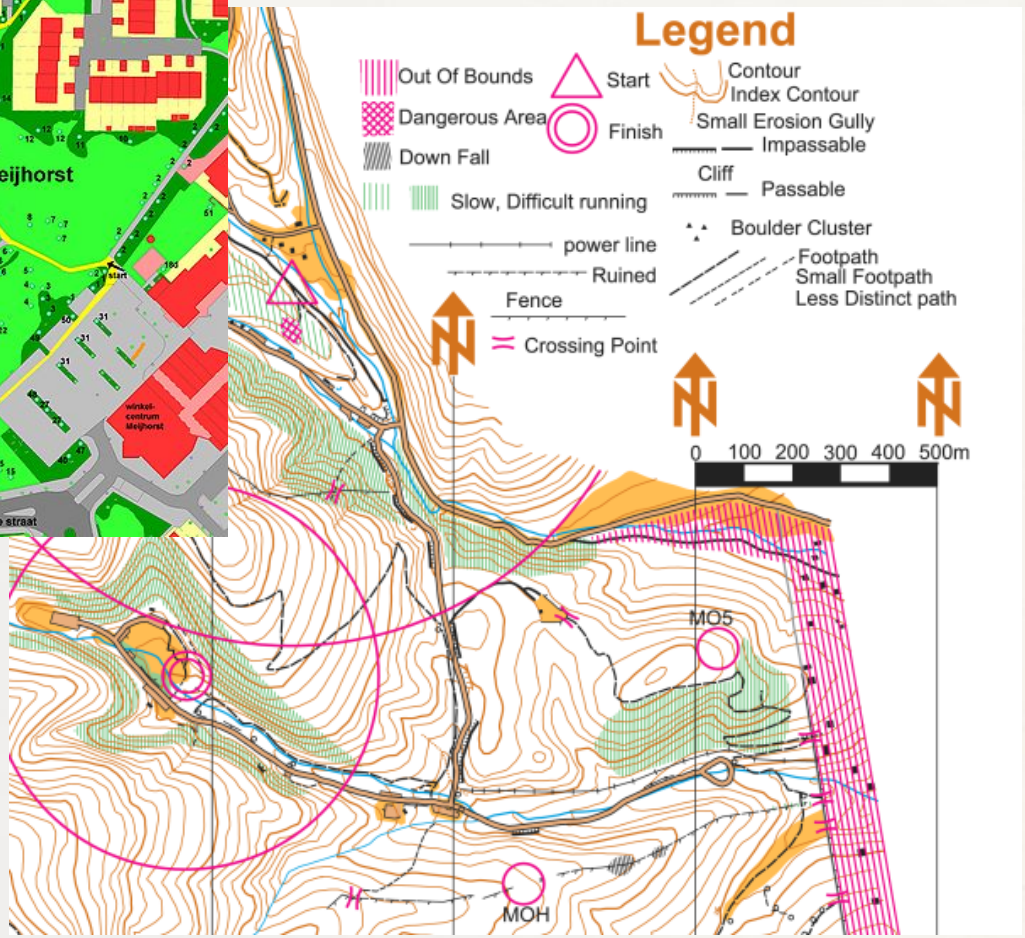
Por que formalizar?

Modelos e Comunicação

By University of Nottingham

Giving direction by Batoul Zaiter (C.O.D. Library) on Flickr

Modelos e Comunicação

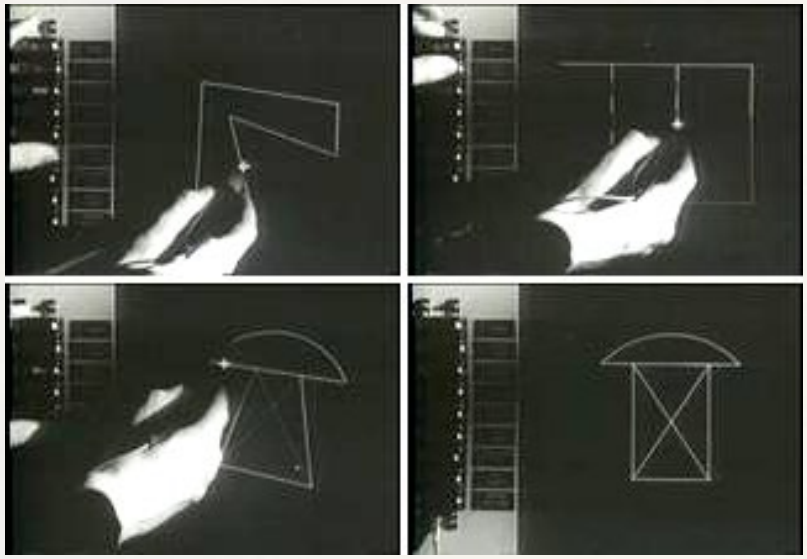


Modelo Orientado a Objetos (OO)

Programação Orientada a Objetos

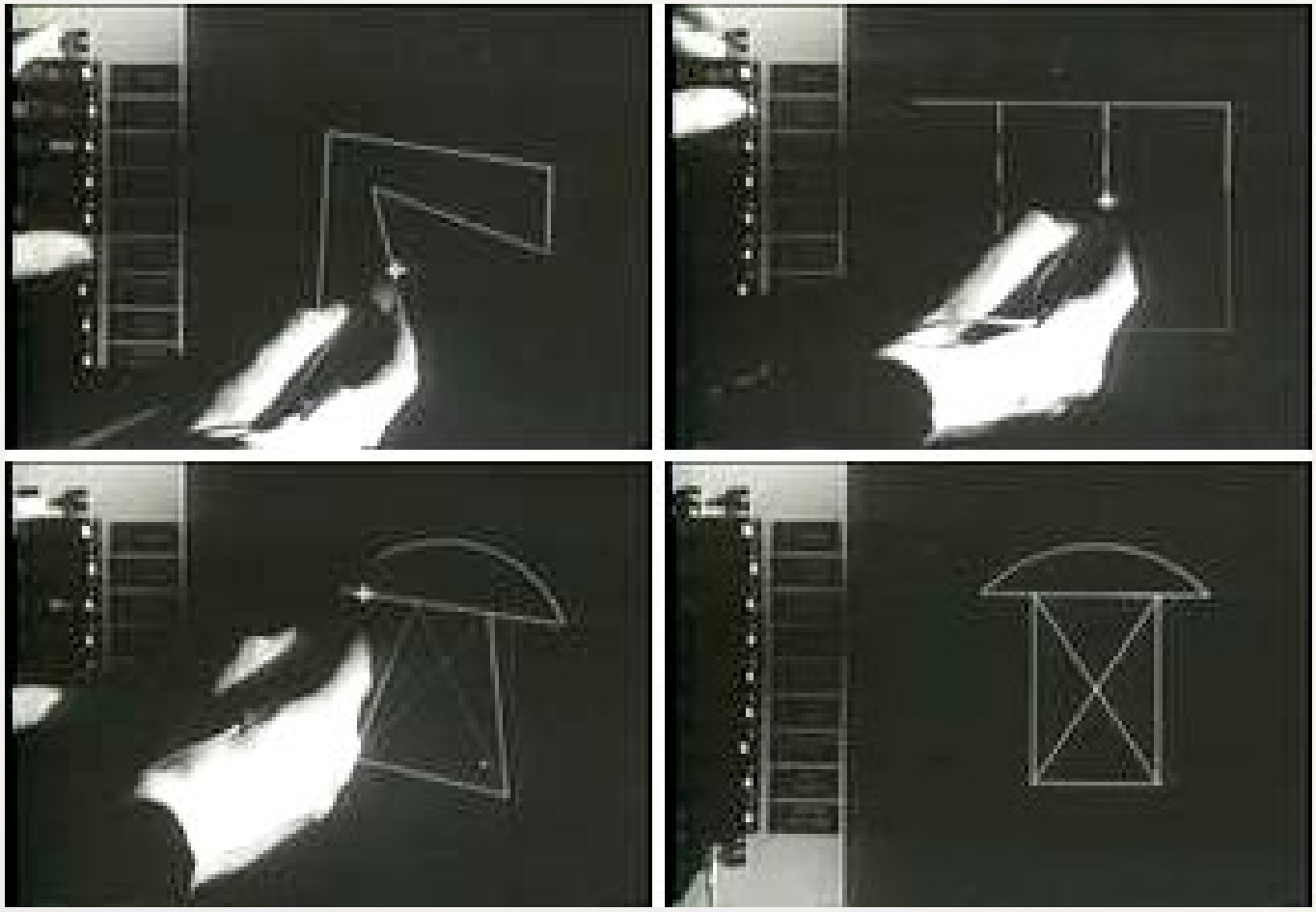
Sketchpad

- Programa desenvolvido em 1962
- Autor: Ivan Sutherland
- Influenciou a construção dos conceitos de interface gráfica e programação orientada a objetos



Fair use, https://en.wikipedia.org/w/index.php?curid=8641142

Sketchpad



Fair use, https://en.wikipedia.org/w/index.php?curid=8641142

LISP

- Especificada em 1958
- Segunda linguagem de alto nível mais antiga (depois do FORTRAN) em uso hoje
- Tornou-se uma família de linguagens
- Foi pioneira em diversos conceitos

LISP

6.2 Association Lists

Within the LISP I system there is a list of atomic symbols already available in the system. Each of these atomic symbols has an association list* associated with it, and in fact the atomic symbol on the list of atomic symbols points to the location of the association list. Integers are included in the list of atomic symbols, but floating-point numbers are listed on a separate list which will be discussed below.

---

* In the local M.I.T. patois, association lists are also referred to as "property lists", and atomic symbols are sometimes called "objects".

SIMULA 67

- Inicialmente uma linguagem para simulações que se tornou de propósito geral
- Primeira Linguagem Orientada a Objetos
- Desenvolvida em 1960 no Norwegian Computing Center em Oslo
- Autores: Ole-Johan Dahl and Kristen Nygaard
- Introduziu objetos, classes, herança, rotinas virtuais e coleta de lixo

Modelo Orientado a Objetos

- SIMULA 67
  - Primeira Linguagem Orientada a Objetos
- Smalltalk
  - Projeto Dynabook
  - “Este ‘Dynabook’ foi baseado na visão de computadores pessoais baratos do tamanho de um caderno, tanto para adultos quanto crianças, com a capacidade de lidar com todas as suas respectivas necessidades de informação”. [KRE98]

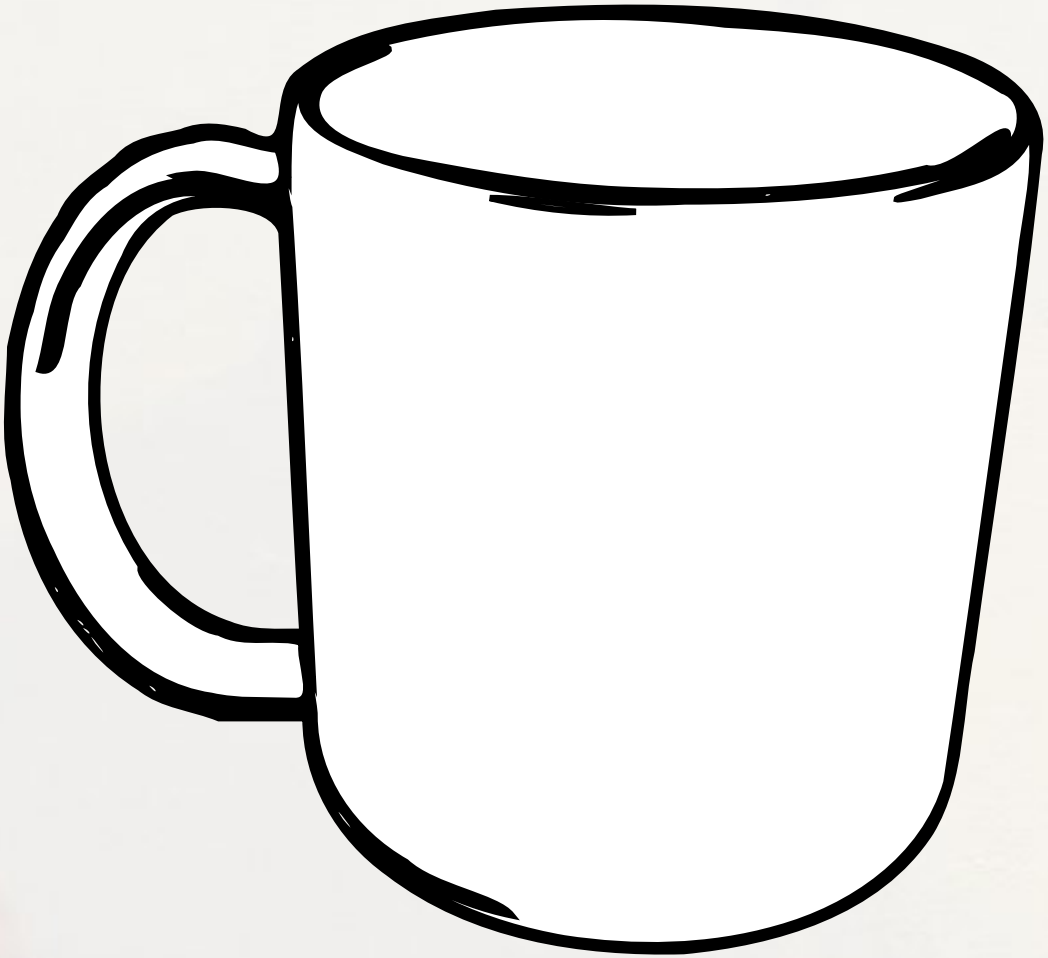
As Duas Faces da OO

- Abordagem de abstração
- Estrutura de dados

Mania de Classificar

Todas as Canecas são a mesma Caneca

Generalizando ou Estereotipando



Generalizando ou Estereotipando

Generalizando ou Estereotipando

Classes ou Estereótipos

- Cérebro – captura funcionalidade relevante dos objetos

(Santos & Hood, 2009)

- Generalização – essencial para sobrevivência
  - memória – guarda estereótipos
  - previsão das funcionalidades

(Bloom, 2007)

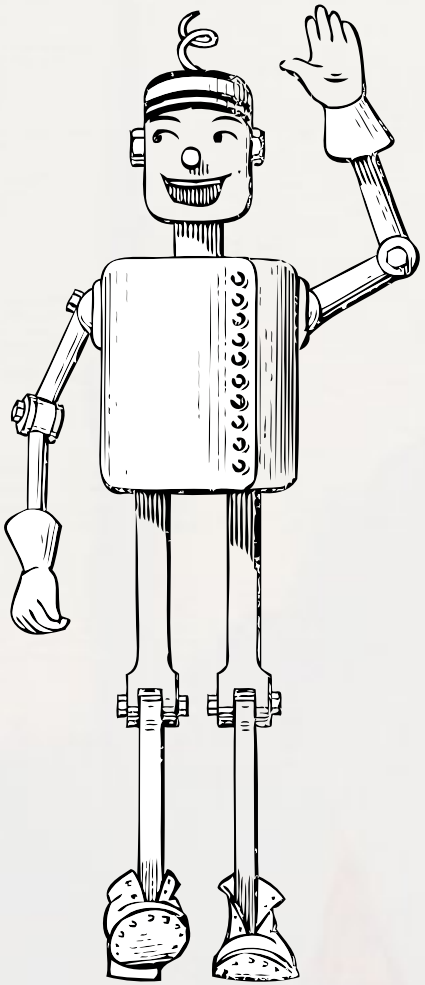
Estereótipos

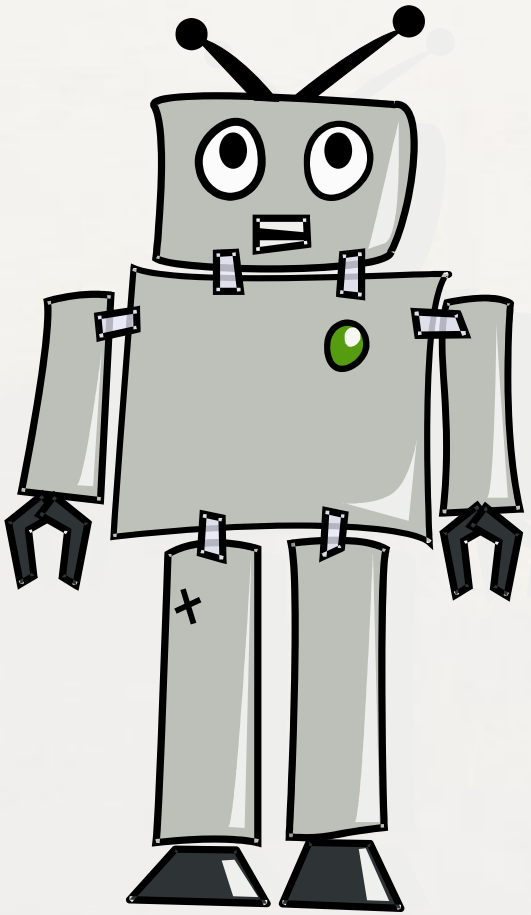
Invariantes e Propriedades

Classificando
Invariantes e Propriedades

Desafios da Representação Compartilhada

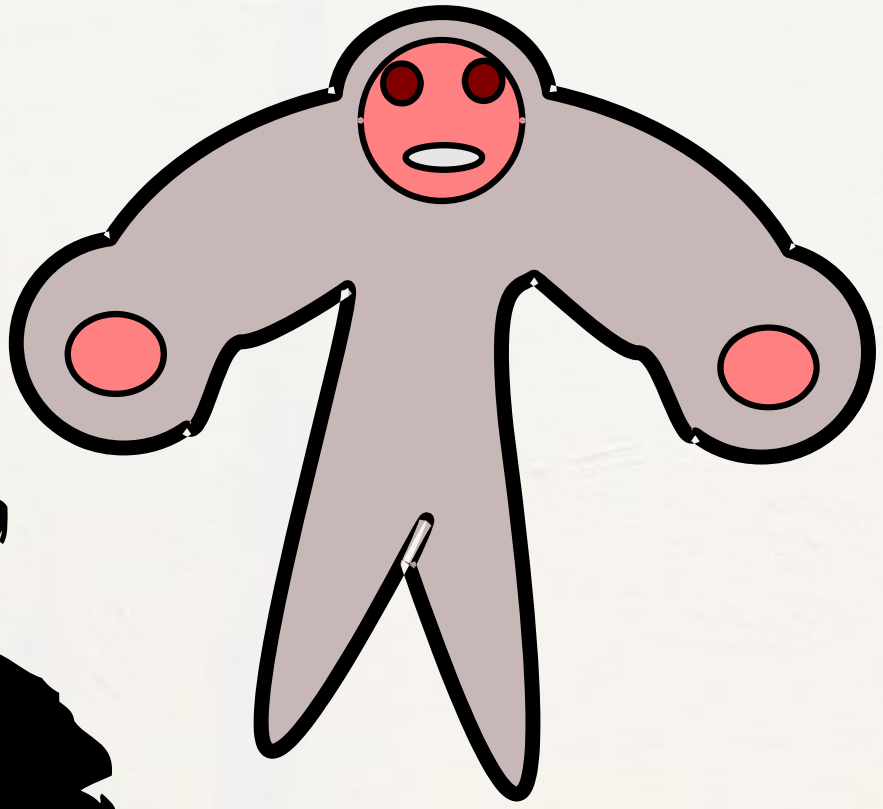
Estereótipos





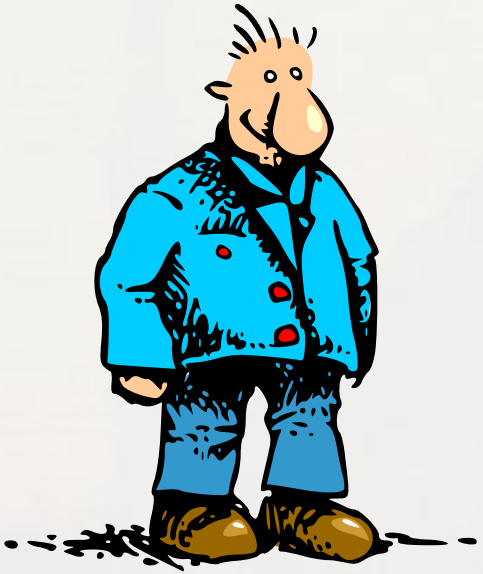
Desafios da Representação Compartilhada

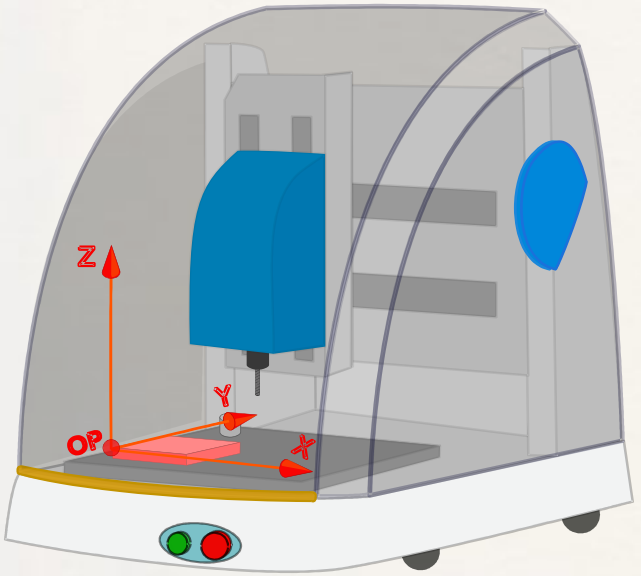
Estereótipos



Estereótipos
Abstrações Humanas

- São o mundo real ou descrevem o mundo real?





Objetos e Memória

Memória de Curta Duração (Trabalho)

- Armazena:
  - produtos intermediários do pensamento
  - representações produzidas pelo Sistema Perceptual
- Operações mentais:
  - obtém operandos
  - deixam resultados intermediários

(Rocha, 2003)

Chunks

- “Conceitualmente a MCD é constituída de chunks: elementos ativados da MLD, que podem ser organizados em unidades maiores.”

(Rocha, 2003, p. 55)

Estereótipos

- Capturar funcionalidade relevante
  - “[...] nervous systems were developed via natural selection to represent objects so that organisms may interact with the external world in an adaptive way, and thus, brains are built to capture what is functionally relevant about objects.” (Santos & Hood, 2009)

Estereótipos

- Estereótipo
  - “We tend to use the term to refer to information we have about categories and intuitions we have about the typicality, our frequency of certain features of categories.” (Bloom, 2007)

Estereótipos

- Essencial para sobrevivência
  - And it turns out that collecting information about categories is essential to our survival. We see novel things all the time and if we were not capable of learning and making guesses, educated guesses, about these novel things we would not be able to survive. So, when you see this object over here you categorize it as a chair and you recognize that you could probably sit on it.” (Bloom, 2007)

Estereótipos

- Generalização
  - “And if you were suddenly stripped of your ability to make generalizations, you'd be at a loss. You wouldn't know what to eat, how to interact. So, some sort of ability to record information and make generalizations is absolutely essential to making it through life.” (Bloom, 2007)

Modelos

- São fundamentais para:
  - Percepção (propósito e funcionalidades)
  - Memória
  - Comunicação
- Percebemos o mundo em objetos
- Os generalizamos e classificamos

As Duas Faces da OO

- Abordagem de abstração
- Estrutura de dados

Segunda Face

Estrutura de Dados

Abstrações em Computação

Tipo Abstrato de Dados

Tipo Abstrato de Dados (TAD)
Abstract Data Type (ADT)

- “O termo 'tipo abstrato de dados' se refere ao conceito matemático básico que define um tipo de dados” (Tenenbaum, 1990)
- Conceito matemático
  - Não considera aspectos de implementação
    - Ex.: eficiência de tempo e espaço

(Tenenbaum, 1990)

Tipo Abstrato de Dados (TAD)
Abstract Data Type (ADT)

- “Um tipo abstrato de dados define uma classe de objetos abstratos que é completamente caracterizada pelas operações disponíveis nestes objetos. Isto significa que um tipo abstrato de dados pode ser definido pela definição e caracterização das operações daquele tipo.” (Liskov, 1974)

Exercício 1 - Empréstimo

- Escreva um módulo para calcular a próxima parcela de um financiamento
- Dados disponíveis
  - **S** - valor da primeira parcela
  - **N** - número de parcelas
  - **J** – percentual de juros mensal
- Cada nova parcela é sempre calculada em relação à anterior:

$$P_{atual} = P_{anterior} \text{ mais Juros}$$

Classe – Arquitetura Modular

Estudo de Caso 1

Modularização Sucessiva

Modularização Sucessiva
Primeira Versão

C
Programa sem modularização

Modularização Sucessiva
Primeira Versão

```
#include <stdio.h>

int main() {
    float s = 200;
    int n = 5;
    float j = 1;

    float p = s;
    for (int i = 1; i <= n; i++) {
        printf("O valor da parcela %d eh %3.2f\n", i, p);
        p = p + (p * (j/100));
    }
}
```

Modularização Sucessiva
Segunda Versão

C
Programa com modularização básica – apenas funções

Modularização Sucessiva
Segunda Versão

```
#include <stdio.h>

float es;
int   en;
float ej;
int   corrente;
float p;

void novoEmprestimo(float s, int n, float j) {
    es = s;
    en = n;
    ej = j;
    corrente = 1;
    p = s;
}
```

Modularização Sucessiva
Segunda Versão

```
float proximaParcela() {
    float retorno = p;
    corrente++;
    if (corrente <= en)
        p = p + (p * (ej/100));
    else
        p = 0;
    return retorno;
}

int main() {
    novoEmprestimo(200, 5, 1);
    int i = 1;
    float p = proximaParcela();
    while (p > 0) {
        printf("O valor da parcela %d eh %3.2f\n", i, p);
        p = proximaParcela();
        i++;
    }
}
```

Modularização Sucessiva
Segunda Versão

- Vantagens:
  - Módulos encapsulam a lógica do cálculo de combinações, de forma que ela possa ser reusada.

Modularização Sucessiva
Segunda Versão

- Problemas:
  - Os módulos depende das variáveis globais do módulo principal.
  - O módulo principal se torna responsável por detalhes de implementação dos módulos, o que prejudica o reuso:
    - cada vez que um programa reusar os módulos precisará declarar as variáveis;
    - as novas variáveis declaradas podem entrar em conflito com outras já existentes, o que exige modificação do código.

Modularização Sucessiva
Terceira Versão

C

Transferência das variáveis globais para os módulos, a fim de remover a dependência

Uma variável global é declarada e passada como parâmetro para os módulos

Modularização Sucessiva
Terceira Versão

```
#include <stdio.h>

void novoEmprestimo(float s, int n, float j,
                    int *corrente, float *p) {
    *corrente = 1;
    *p = s;
}

void proximaParcela(float s, int n, float j,
                    int *corrente, float *p) {
    (*corrente)++;
    if (*corrente <= n)
        *p = *p + (*p * (j/100));
    else
        *p = 0;
}
```

Modularização Sucessiva
Terceira Versão

```
int main() {
    int corrente;
    float p;

    novoEmprestimo(200, 5, 1, &corrente, &p);

    int i = 1;
    while (p > 0) {
        printf("O valor da parcela %d eh %3.2f\n", i, p);
        proximaParcela(200, 5, 1, &corrente, &p);
        i++;
    }
}
```

Modularização Sucessiva
Terceira Versão

- Vantagens:
  - A variável do módulo principal se torna independente da variável dos módulos (o nome pode ser diferente).
- Problemas:
  - O módulo principal continua precisando declarar e manter as variáveis globais, o que ainda causa dependência das funções.
  - Neste ponto esgotam-se as possibilidades da modularização baseada em funções.

Modularização Sucessiva
Quarta Versão

C++
Classe

Modularização Sucessiva

Sétima Versão – Bits07Instancia.h

```
class Bits07Instancia
{
    int combinacoes;
    public:
        Bits07Instancia();
        int proximoNumeroCombinacoes();
};
```

Modularização Sucessiva

Sétima Versão – Bits07Instancia.cpp

```
#include "Bits07Instancia.h"


Bits07Instancia::Bits07Instancia()
{
    combinacoes = 1;
}


int Bits07Instancia::proximoNumeroCombinacoes()
{
    combinacoes *= 2;
    return combinacoes;
}
```

Modularização Sucessiva
Sétima Versão – Bits07.c

```
#include <stdio.h>


#include "Bits07Instancia.h"


int main () {
  Bits07Instancia objeto;


  int bits;
  for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes\n", bits,
  objeto.proximoNumeroCombinacoes());
}
```

Modularização Sucessiva
Sexta Versão

Java
O módulo é transformada em uma classe

Modularização Sucessiva
Sexta Versão – Classe

```
package pt.c02oo.s01estudocaso.s06classe;

public class Bits06Classe
{
    static int combinacoes;

    static void inicializa()
    {
        combinacoes = 1;
    }

    static int proximoNumeroCombinacoes()
    {
        combinacoes *= 2;
        return combinacoes;
    }
}
```

Modularização Sucessiva

Sexta Versão – Programa Principal

```
package pt.c02oo.s01estudocaso.s06classe;

public class Bits06
{
    public static void main(String args[])
    {
        Bits06Classe.inicializa();
        for (int bits = 1; bits <= 8; bits++)
            System.out.println(bits + " = " +
                               Bits06Classe.proximoNumeroCombinacoes());
    }
}
```

Modularização Sucessiva

Sétima Versão

Java

Classe com instância

Modularização Sucessiva
Sétima Versão – Classe

```
package pt.c02oo.s01estudocaso.s07objeto;

public class Bits07Instancia
{
    int combinacoes;

    Bits07Instancia()
    {
        combinacoes = 1;
    }

    int proximoNumeroCombinacoes()
    {
        combinacoes *= 2;
        return combinacoes;
    }
}
```

Modularização Sucessiva

Sétima Versão – Programa Principal

```
package pt.c02oo.s01estudocaso.s07objeto;

public class Bits07
{
    public static void main(String args[])
    {
        Bits07Instancia objeto;
        objeto = new Bits07Instancia();

        for (int bits = 1; bits <= 8; bits++)
            System.out.println(bits + " = " +
                               objeto.proximoNumeroCombinacoes());
    }
}
```

Estudo de Caso 1

Modularização Sucessiva

- Programa que calcula e apresenta o número de combinações possíveis que podem ser realizadas com bits, que variam de 1 a 8.
- Mostra etapas sucessivas do processo de modularização, até se chegar à classe.

Modularização Sucessiva
Primeira Versão

C
Programa sem modularização

Modularização Sucessiva
Primeira Versão

```c
#include <stdio.h>

int main()
{
    int combinacoes = 1,
        bits;

    for (bits = 1; bits <= 8; bits++)
    {
        combinacoes *= 2;
        printf("%d bits = %d combinacoes",
               bits, combinacoes);
    }

    return 0;
}
```

Modularização Sucessiva
Segunda Versão

C
Programa com modularização básica – apenas funções

Modularização Sucessiva
Segunda Versão

```
int combinacoes;

void inicializa() {
    combinacoes = 1;
}

int proximoNumeroCombinacoes() {
    combinacoes *= 2;
    return combinacoes;
}

int main() {
    int bits;
    inicializa();
    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes",
                bits, proximoNumeroCombinacoes());
    return 0;
}
```

Modularização Sucessiva
Segunda Versão

- Vantagens:
  - Módulos encapsulam a lógica do cálculo de combinações, de forma que ela possa ser reusada.

Modularização Sucessiva
Segunda Versão

- Problemas:
  - Os módulos dependem do programa principal que mantém a variável "combinacoes".
  - O programa principal se torna responsável por detalhes de implementação dos módulos, o que prejudica o reuso:
    - cada vez que um programa reusar os módulos precisará declarar a variável "combinacoes";
    - a nova variável "combinacoes" declarada pode entrar em conflito com uma já existente, o que exige modificação do código.

Modularização Sucessiva
Terceira Versão

C

Tentativa de transferir a variável "combinacoes" para os módulos, a fim de remover a dependência

Modularização Sucessiva
Terceira Versão

```
void inicializa() {
    int combinacoes;
    combinacoes = 1;
}

int proximoNumeroCombinacoes() {
    int combinacoes;
    combinacoes *= 2;
    return combinacoes;
}

int main() {
    int bits;
    inicializa();
    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes",
               bits, proximoNumeroCombinacoes());
}
```

Modularização Sucessiva
Terceira Versão

- Erro de execução:
  - A variável local e criada e destruída a cada entrada/saída de cada um dos módulos, impossibilitando a continuidade desejada.

Modularização Sucessiva
Quarta Versão

C

Uma variável global é declarada e passada como parâmetro para os módulos

Modularização Sucessiva
Quarta Versão

```c
void inicializa(int *combinacoes) {
    *combinacoes = 1;
}

int proximoNumeroCombinacoes(int *combinacoes) {
    *combinacoes *= 2;
    return *combinacoes;
}

int main() {
    int combinacoes;
    int bits;

    inicializa(&combinacoes);

    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes\n", bits,
                    proximoNumeroCombinacoes(&combinacoes));
}
```

Modularização Sucessiva
Quarta Versão

- Vantagens:
  - A variável do programa principal se torna independente da variável dos módulos (o nome pode ser diferente).
- Problemas:
  - O programa principal continua precisando declarar e manter a variável "combinacoes", o que ainda causa dependência dos módulos
  - Neste ponto esgotam-se as possibilidades da modularização baseada em procedures e functions.

Modularização Sucessiva
Quinta Versão

C

Os módulos menores (funções) são colocados dentro de um módulo maior

Modularização Sucessiva

Quinta Versão – bits05module.h

```
void inicializa();

int proximoNumeroCombinacoes();
```

Modularização Sucessiva

Quinta Versão – bits05module.c

```c
#include "bits05module.h"

static int combinacoes;

void inicializa()
{
    combinacoes = 1;
}

int proximoNumeroCombinacoes()
{
    combinacoes *= 2;
    return combinacoes;
}
```

Modularização Sucessiva

Quinta Versão - bits05.c

```c
#include "bits05module.h"

int main()
{
    int bits;

    inicializa();

    for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes",
                bits, proximoNumeroCombinacoes());

    return 0;
}
```

Modularização Sucessiva
Quinta Versão

- Vantagens:
  - O módulo expõe apenas as interfaces das funções, escondendo detalhes da implementação
  - O módulo controla e mantém o estado da variável "combinacoes", que não e visível para o programa principal.

Modularização Sucessiva
Quinta Versão

- Problemas:
  - O módulo funciona para apenas uma instância. Se precisássemos de dois cálculos de combinações em paralelo teríamos problemas.
  - O uso de múltiplas instâncias é possível mas complicado.

Modularização Sucessiva
Sexta/Sétima Versão

C++
Classe

Modularização Sucessiva

Sétima Versão – Bits07Instancia.h

```
class Bits07Instancia
{
    int combinacoes;
    public:
        Bits07Instancia();
        int proximoNumeroCombinacoes();
};
```

Modularização Sucessiva

Sétima Versão – Bits07Instancia.cpp

```
#include "Bits07Instancia.h"


Bits07Instancia::Bits07Instancia()
{
    combinacoes = 1;
}


int Bits07Instancia::proximoNumeroCombinacoes()
{
    combinacoes *= 2;
    return combinacoes;
}
```

Modularização Sucessiva
Sétima Versão – Bits07.c

```
#include <stdio.h>


#include "Bits07Instancia.h"


int main () {
  Bits07Instancia objeto;


  int bits;
  for (bits = 1; bits <= 8; bits++)
        printf("%d bits = %d combinacoes\n", bits,
  objeto.proximoNumeroCombinacoes());
}
```

Modularização Sucessiva
Sexta Versão

Java
O módulo é transformada em uma classe

Modularização Sucessiva
Sexta Versão – Classe

```
package pt.c02oo.s01estudocaso.s06classe;

public class Bits06Classe
{
    static int combinacoes;

    static void inicializa()
    {
        combinacoes = 1;
    }

    static int proximoNumeroCombinacoes()
    {
        combinacoes *= 2;
        return combinacoes;
    }
}
```

Modularização Sucessiva

Sexta Versão – Programa Principal

```
package pt.c02oo.s01estudocaso.s06classe;

public class Bits06
{
    public static void main(String args[])
    {
        Bits06Classe.inicializa();
        for (int bits = 1; bits <= 8; bits++)
            System.out.println(bits + " = " +
                               Bits06Classe.proximoNumeroCombinacoes());
    }
}
```

Modularização Sucessiva

Sétima Versão

Java

Classe com instância

Modularização Sucessiva
Sétima Versão – Classe

```
package pt.c02oo.s01estudocaso.s07objeto;

public class Bits07Instancia
{
    int combinacoes;

    Bits07Instancia()
    {
        combinacoes = 1;
    }

    int proximoNumeroCombinacoes()
    {
        combinacoes *= 2;
        return combinacoes;
    }
}
```

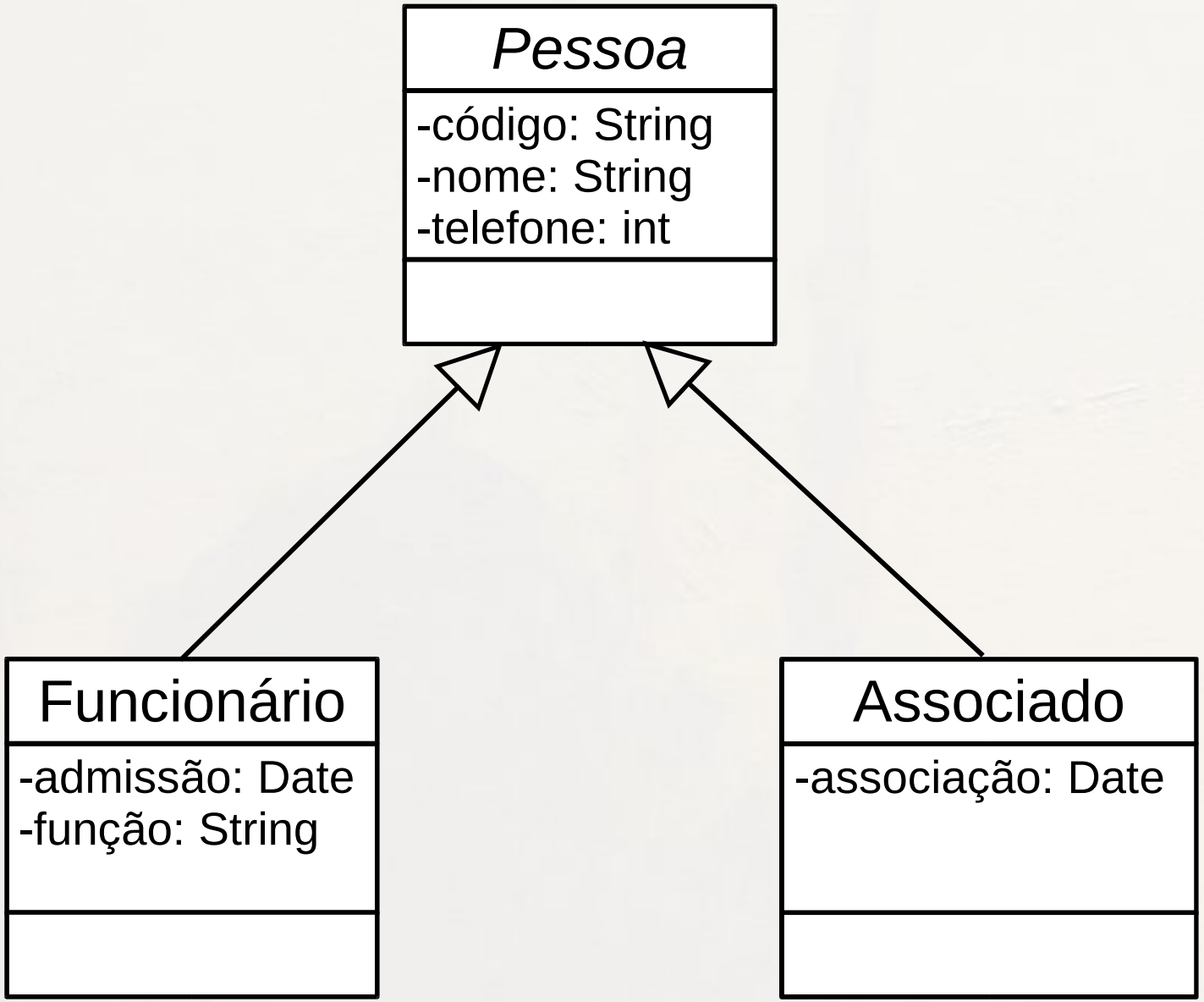
Modularização Sucessiva

Sétima Versão – Programa Principal

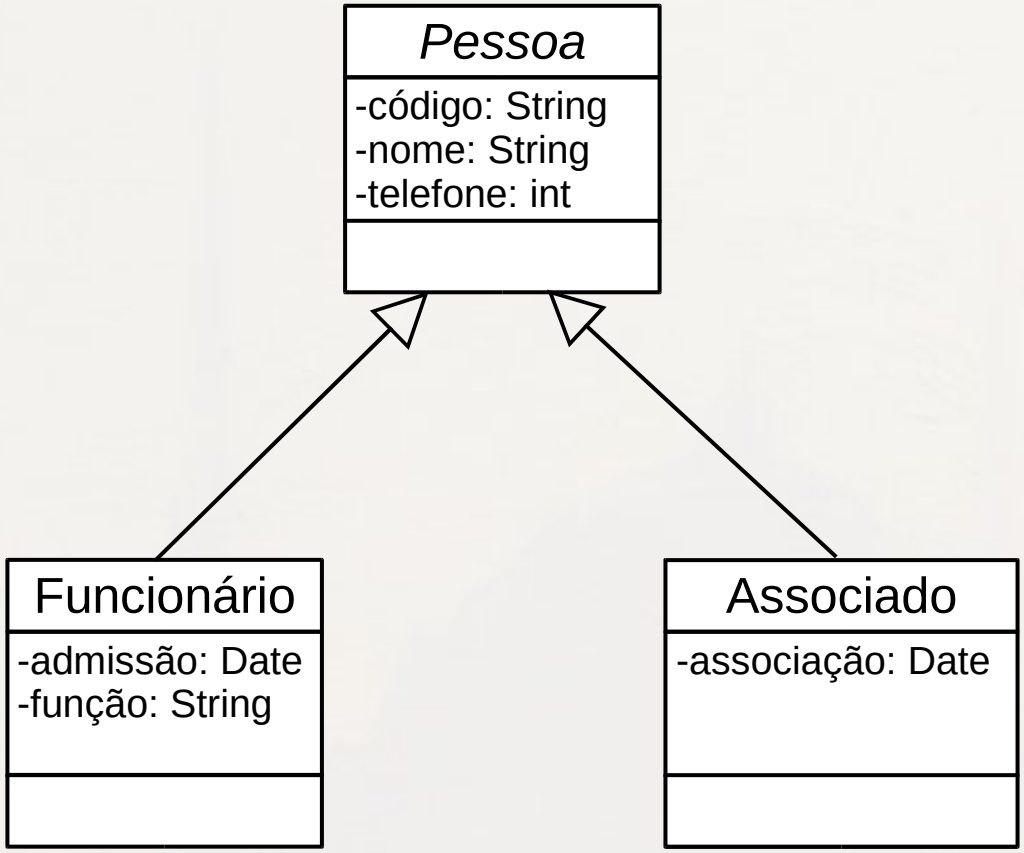
```
package pt.c02oo.s01estudocaso.s07objeto;

public class Bits07
{
    public static void main(String args[])
    {
        Bits07Instancia objeto;
        objeto = new Bits07Instancia();

        for (int bits = 1; bits <= 8; bits++)
            System.out.println(bits + " = " +
                               objeto.proximoNumeroCombinacoes());
    }
}
```

OO: Herança

UML: Herança

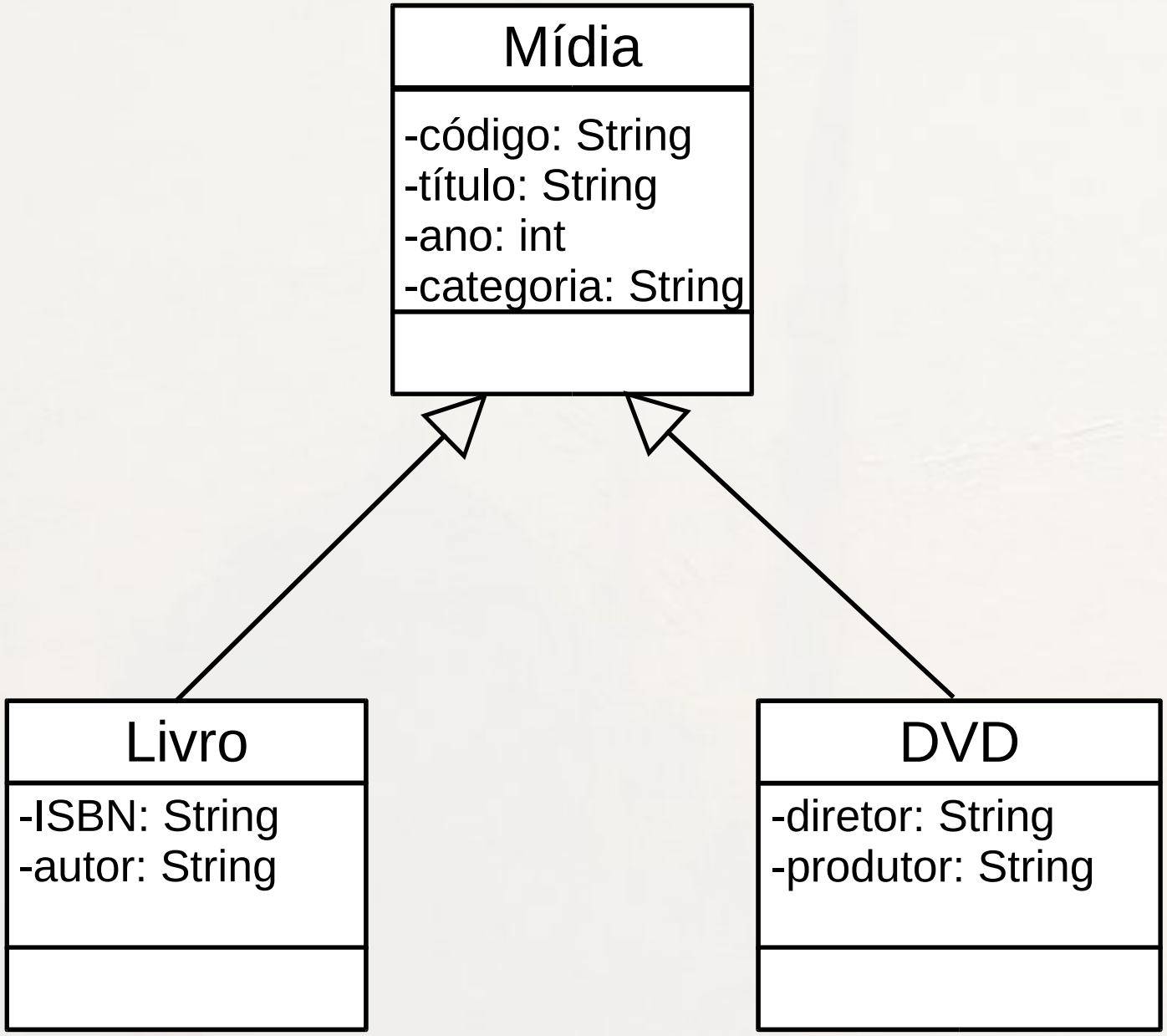

Pessoa
-código: String
-nome: String
-telefone: int
Funcionário
-admissão: Date
-função: String
Associado
-associação: Date

Especialização total x Classe abstrata

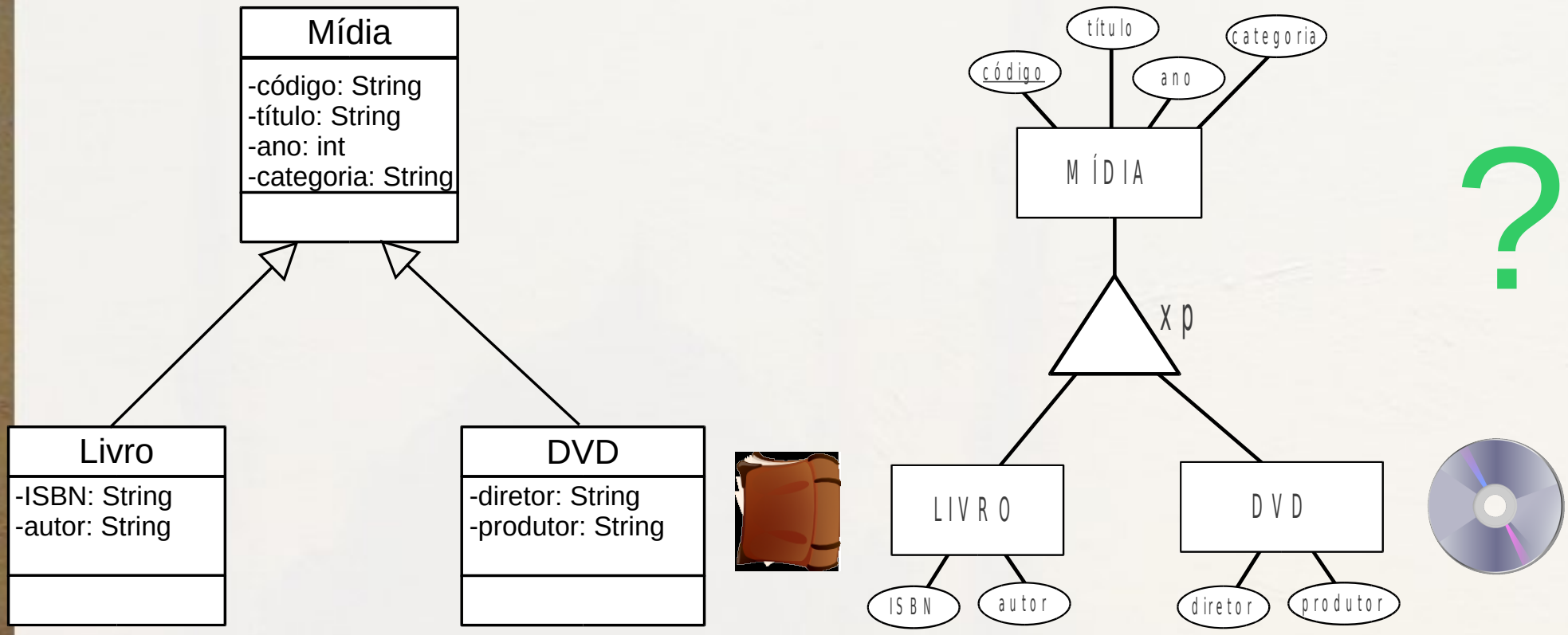


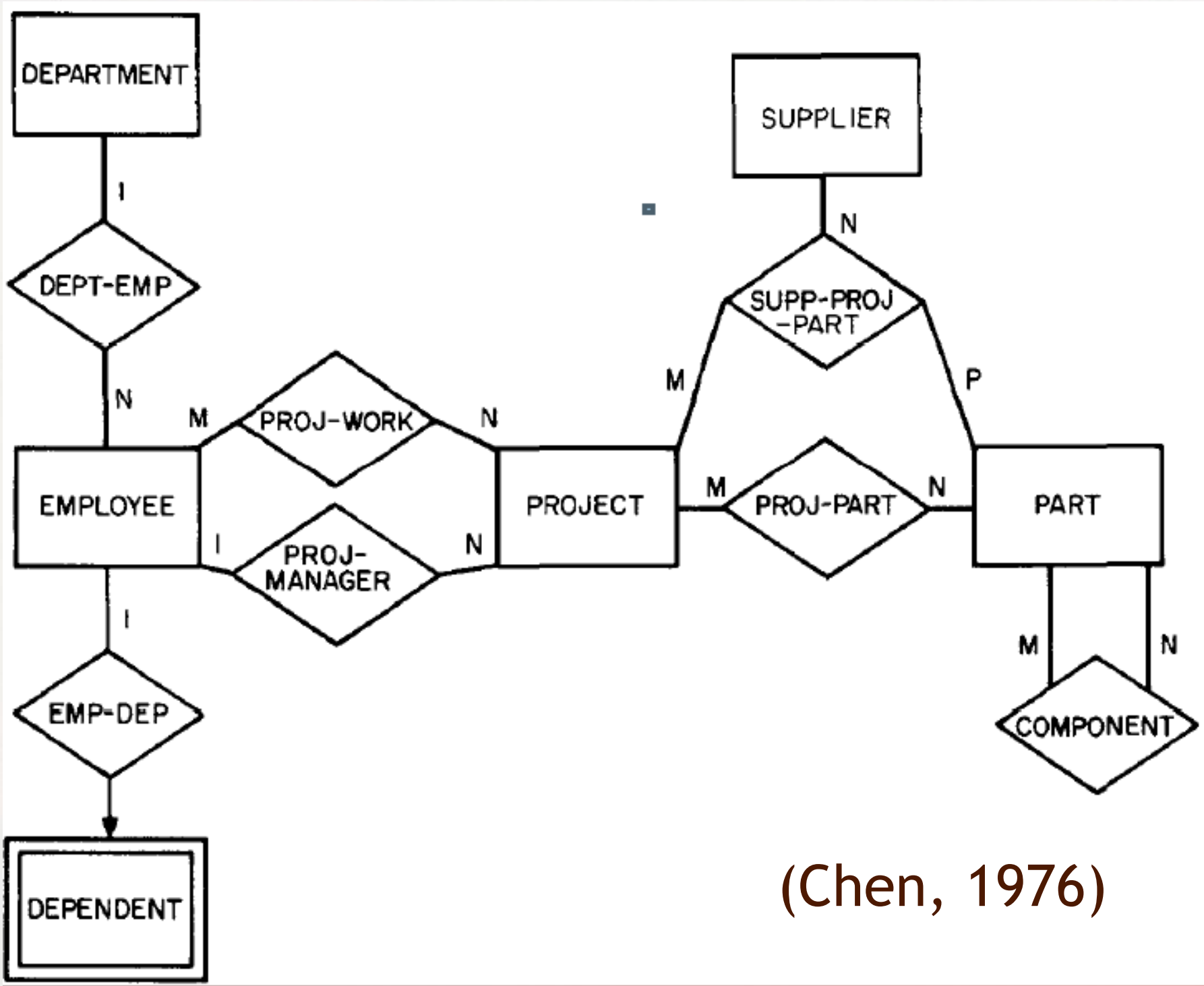
especialização total x classe abstrata

UML: Herança


Mídia
-código: String
-título: String
-ano: int
-categoria: String
Livro
-ISBN: String
-autor: String
DVD
-diretor: String
-produtor: String

Especialização parcial x Classe




DEPARTMENT
I
DEPT-EMP
N
EMPLOYEE
M
PROJ-WORK
N
PROJECT
I
PROJ-
MANAGER
N
I
EMP-DEP
DEPENDENT
SUPPLIER
N
SUPP-PROJ
-PART
M
P
M
PROJ-PART
N
PART
M
N
COMPONENT

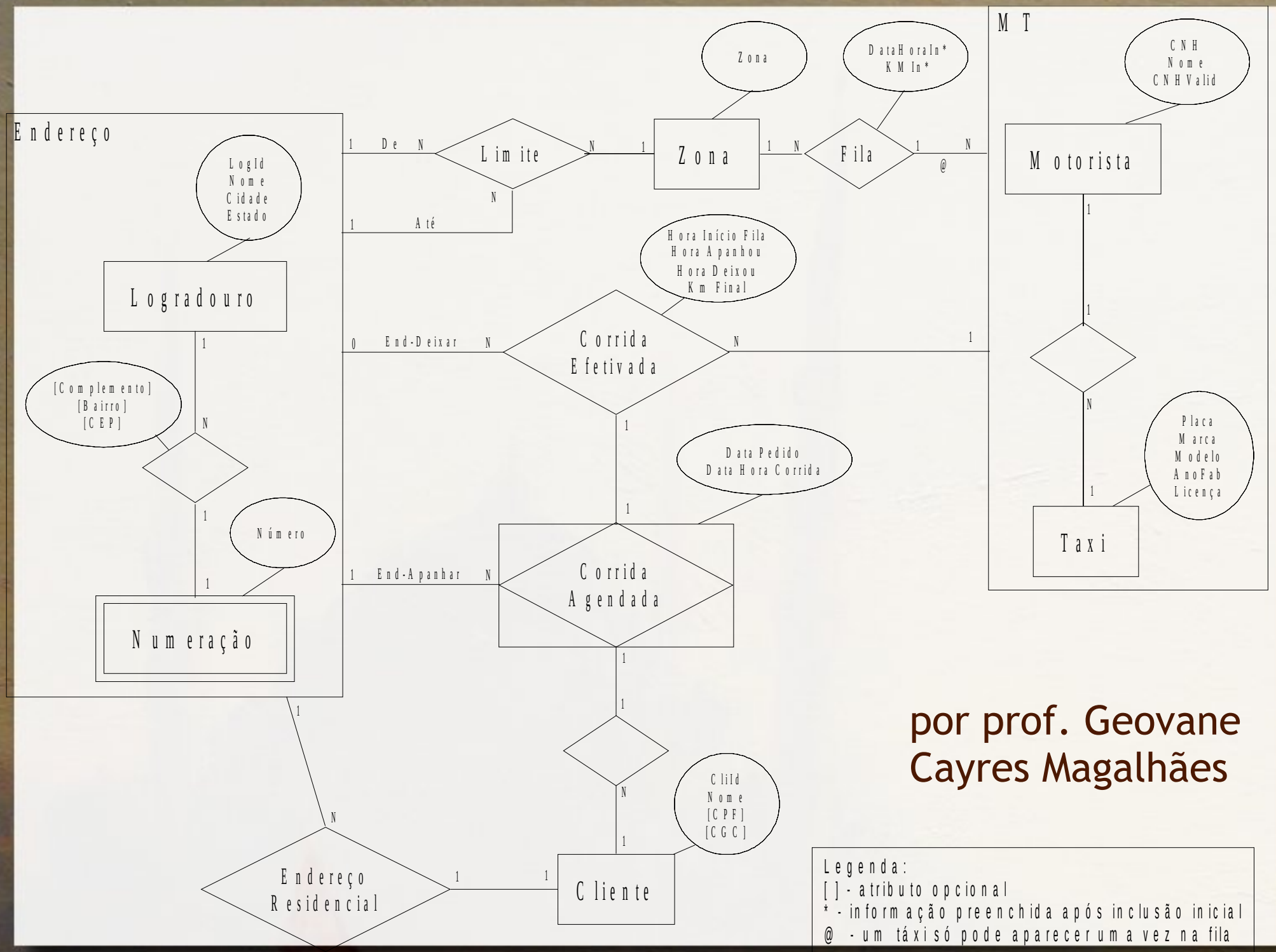
(Chen, 1976)

Exercício
parte 3

- Vírus podem ser classificados em diversas categorias. A categoria retrovírus é tratada com coquetéis de medicamentos. Um coquetel é composto por vários medicamentos, cada um em uma concentração específica.
- Os tratamentos de retrovírus baseados em coquetéis também devem especificar dosagens específicas por tipo de paciente.
- Considere dois cenários de restrição:
  - somente retrovírus são tratados com coquetéis
  - retrovírus só são tratados com coquetéis

Caso dos Taxis

- Exemplo criado por prof. Geovane Cayres Magalhães
  - http://www.ic.unicamp.br/~geovane/mo410-091/caso.html



por prof. Geovane Cayres Magalhães

Legenda:
[ ] - atributo opcional
* - informação preenchida após inclusão inicial
@ - um táxi só pode aparecer uma vez na fila

Referências

- Chen, Peter Pin-Shan (1976) **The entity-relationship model – toward a unified view of data**. ACM Trans. Database Systems, ACM, 1, 9-36.
- Elmasri, Ramez; Navathe, Shamkant B. (2005) **Sistemas de Bancos de Dados**. Addison-Wesley, 4ª edição em português.
- Guimarães, Célio (2003) **Fundamentos de Bancos de Dados: Modelagem, Projeto e Linguagem SQL**. Editora UNICAMP, 1ª edição.
- Heuser, Carlos Alberto (2004) **Projeto de Banco de Dados**. Editora Sagra Luzzato, 5ª edição.

Referências

- Ramakrishnan, Raghu; Gehrke, Johannes (2003) **Database Management Systems.** McGraw-Hill, 3rd edition.

Referências Bibliográficas

- Almeida, Charles Ornelas , Guerra, Israel; Ziviani, Nivio (2010) **Projeto de Algoritmos** (transparências aula).
- Bloom, Paul (2007) **Introduction to Psychology** – transcrição das aulas (aula 17). Yale University.
- Ferreira, Aurélio B. H. (1989) **Minidicionário da Língua Portuguesa**. Rio de Janeiro, Editora Nova Fronteira.
- Houaiss, Instituto Antônio. **Dicionário Houaiss da língua portuguesa** (2006) Editora Objetiva, Março.
- IBM - International Business Machines Corporation. **IBM Smalltalk Tutorial** [Online] http://www.wi2.uni-erlangen.de/sw/smalltalk/
- Liskov, Barbara; Zilles, Stephen. **Programming with abstract data types** (1974) ACM SIGPLAN Notices, 9 (4) p. 50.

Referências Bibliográficas

- McCarthy, J.; Brayton, R.; Edwards, D.; Fox, P.; Hodes, L.; Luckham, D.; Maling, K.; Park, D.; Russell, S. (March 1960). **LISP I Programmers Manual**. Boston, Massachusetts: Artificial Intelligence Group, M.I.T. Computation Center and Research Laboratory: 88f.
- Meyer, Bertrand (1997) **Object-Oriented Software Construction – Second Edition**. USA, Prentice-Hall, Inc.
- Miller, Robert (2004) **6.831 User Interface Design and Implementation (lecture notes)**. MIT OpenCourseware.

Referências Bibliográficas

- Rocha, Heloisa Vieira da, Baranauskas, Maria Cecilia Calani (2003) **Design e Avaliação de Interfaces Humano-Computador.** NIED/UNICAMP.
- Santos, L. R., & Hood, B. M. (2009). **Object representation as a central issue in cognitive science**. The Origins of Object Knowledge: The Yale Symposium on the Origins of Object & Number Representation. Oxford: Oxford University Press.
- Shaw, M. **Abstraction Techniques in Modern Programming Languages** (1984) IEEE Software, 1, 4, 10-26.
- Tenenbaum, Aaron M.; Langsam, Yedidyah; Augenstein, Moshe J. **Data Structures Using C** (1990) Prentice Hall, Upper Saddle River, NJ.

Referências

- Bloom, Paul (2007) **Introduction to Psychology** – transcrição das aulas (aula 17). Yale University.
- Chen, Peter Pin-Shan (1976) **The entity-relationship model – toward a unified view of data**. ACM Trans. Database Systems, ACM, 1, 9-36.
- Dijkstra, E. W. (1986) **On a cultural gap**. The Mathematical Intelligencer. vol. 8, no. 1, pp. 48-52.
- Elmasri, Ramez; Navathe, Shamkant B. (2005) **Sistemas de Bancos de Dados**. Addison-Wesley, 4a. edição em português.
- Elmasri, Ramez; Navathe, Shamkant B. (2011) **Sistemas de Bancos de Dados**. Pearson, 6a. edição em português.
- Guimarães, Célio (2003) **Fundamentos de Bancos de Dados: Modelagem, Projeto e Linguagem SQL**. Editora UNICAMP, 1a. edição.

Agradecimentos

- Luiz Celso Gomes Jr (professor desta disciplina em 2014) pela contribuição na disciplina e nos slides.

André Santanchè
http://www.ic.unicamp.br/~santanche

Licença

- Estes slides são concedidos sob uma Licença Creative Commons. Sob as seguintes condições: Atribuição, Uso Não-Comercial e Compartilhamento pela mesma Licença.
- Mais detalhes sobre a referida licença Creative Commons veja no link: http://creativecommons.org/licenses/by-nc-sa/3.0/

- Fotografia da capa feita por André Santanchè no Petit Palais (Paris) em 17/02/2013 do quadro: Fantasia à Constantinople de Felix Ziem