Universidade Federal de Uberlândia
Faculdade de Computação

Introdução a linguagens orientadas a objetos

Prof. Renato Pimentel

2024/2

Sumário

1 Introdução a linguagens orientadas a objetos

Linguagem de programação I

Como *softwares* são construídos?

Linguagens de programação são utilizadas para a construção de programas em computadores

Linguagem de programação II

Uma **linguagem de programação** pode ser definida como:

- Conjunto limitado de símbolos e comandos, utilizados para criar programas;
- Método padronizado para expressar instruções para um computador

Por meio dela se estabelece uma comunicação com o computador, fazendo com que ele compreenda e execute o que o programador determinar.

Uma linguagem (natural ou de programação) é formada por:

- **Sintaxe**: a forma ou estrutura das expressões;
- **Semântica**: o significado das expressões.

Sintaxe I

Sintaxe: determina regras de como se escreve de forma correta em uma linguagem (*regras de escrita*).

Frase sintaticamente correta

Os seguintes países fazem parte do Mercosul: Brasil, Argentina, Paraguai, Uruguai e Venezuela

Frase sintaticamente incorreta

Os **serguintes** países **faz** parte do Mercosul: Brasil, Argentina, Paraguai, Uruguai e Venezuela

Sintaxe em programação I

Considere o comando para a criação e declaração de uma variável, em linguagem Java:

```
int idade;
```

Considere agora o comando para atribuição de valor a uma variável, em linguagem Java:

```
idade = 10;
```

Estes comandos estão sintaticamente corretos, na linguagem de programação Java.

Sintaxe em programação II

Considere agora os seguintes comandos, também em Java:

```
Int idade;      // ERRO: Int
int idade       // ERRO: Falta ;
```

No caso de atribuição de valores:

```
idade := 10;    // ERRO: :=
idade = 10;     // ERRO: Falta ;
```

Estes comandos estão sintaticamente incorretos, tratando-se especificamente de Java.

Sintaxe em programação III

Durante o início do aprendizado de uma linguagem de programação, é natural demorar muito tempo procurando erros de sintaxe.

Conforme o programador ganha experiência, entretanto, cometerá menos erros, e os identificará mais rapidamente.

Semântica I

A **Semântica** está relacionada ao significado das palavras ou frases:

Semântica II

Frase semanticamente correta

O Sol é uma estrela

Frase semanticamente incorreta

Os modelos mais sofisticados trazem acentos com abas laterais, volante com ajuste de altura e profundidade e fácil acesso aos pedais.

Semântica em programação I

Considere os comandos, em Java:

```
int idade;      // comandos sintatica e
idade = 10;     // semanticamente corretos
```

Considere agora os seguintes comandos:

```
int idade;
idade = 10.7; // comando de atribuição
semanticamente incorreto
```


Semântica em programação II

Há **erros de semântica** relacionados ao raciocínio/ lógica de programação

- Para este tipo de erro, o programa executará com sucesso (o computador não irá gerar quaisquer mensagens de erro)
- Mas ele não fará o esperado, apesar de fazer exatamente o que o programador mandar.

- Ocorre em diferentes níveis:
  - Linguagem de máquina;
  - Linguagem de baixo nível;
  - Linguagem de alto nível.
  - Linguagem de muito alto nível.

Linguagem de máquina

- Conjunto básico de instruções, em **código binário** (0s e 1s), características de cada computador, correspondentes às suas operações básicas:
  - Instruções lógicas e aritméticas;
  - Instruções para transferência de Informação;
  - Instruções para testes;
  - Instruções de entrada/saída;
- Programação inviável para seres humanos.

Linguagens de baixo nível I

- Linguagem simbólica: bem próxima da **linguagem de máquina**;
- Programação baseada na manipulação de registradores e utilização de interrupções para prover entrada/saída;
- Como há uma correspondência **biunívoca** entre instruções simbólicas e instruções da máquina, as linguagens simbólicas:
  - Dependem do processador utilizado;
  - Permitem escrever programas muito eficientes;
  - Porém: são de utilização muito difícil e sujeitas a erros;

  Exemplo: **Assembly**

Linguagens de baixo nível II

Exemplo de programa em Assembly: escrever a mensagem "Olá, mundo" na tela (fonte: https://pt.wikipedia.org/wiki/Programa_Ol%C3%A1_Mundo#Assembly):

```
1   variable:
2   .message    db    "Ola, Mundo!$"
3   code:
4   mov   ah, 9
5   mov   dx, offset .message
6   int   0x21
7   ret
8
```

Linguagens de baixo nível III

A linguagem Assembly (linguagem de montagem) usa nomes (mnemônicos) e símbolos em lugar dos números:

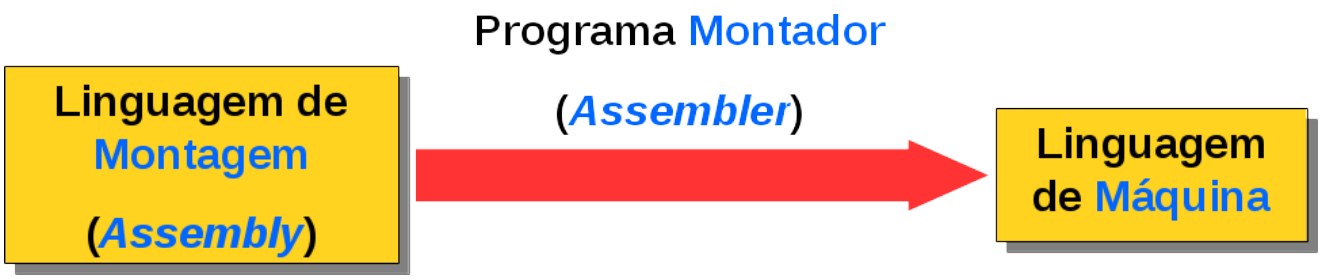
- Utiliza palavras abreviadas (mnemônicos) indicando operações

```
mov R1, R2
```

- Mnemônico MOV (abreviação de MOVE)
- 2 registradores como parâmetros: R1 e R2
- Processador comanda o movimento do conteúdo de R2 para R1 equivalente à instrução Java `R1 = R2;`

Linguagens de baixo nível IV

A tradução/conversão da linguagem Assembly para a linguagem de máquina se chama **montagem**:



- Programas são escritos usando uma linguagem parecida com a nossa (vocabulário corrente);
- Independe da arquitetura do computador;
- Programação mais rápida e eficiente: mais fácil; programas menos sujeitos a erros;

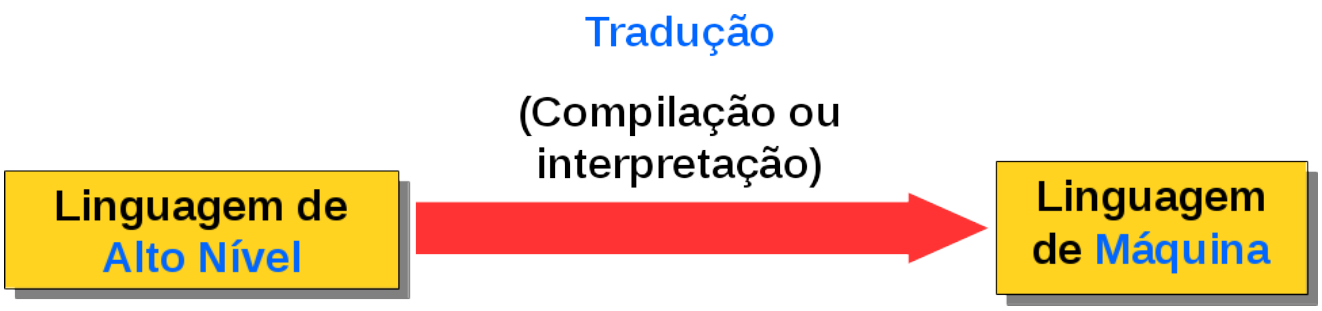
Algumas linguagens de programação e ano em que foram desenvolvidas:

1957	FORTRAN	1972	C	1984	Standard ML
1958	ALGOL	1975	Pascal	1986	C++
1960	LISP	1975	Scheme	1986	CLP(R)
1960	COBOL	1977	OPS5	1986	Eiffel
1962	APL	1978	CSP	1988	CLOS
1962	SIMULA	1978	FP	1988	Mathematica
1964	BASIC	1980	dBase II	1988	Oberon
1964	PL/1	1983	Smalltalk	1990	Haskell
1966	ISWIM	1983	Ada	1995	Delphi
1970	Prolog	1983	Parlog	1995	**Java**

```java
class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println ("Bem Vindos ao curso de POO");
    }
}
```


Os programas escritos em linguagens de alto nível são denominados **código fonte**.

- Os códigos fonte devem ser convertidos para a linguagem de máquina (**tradução**)



Linguagens de muito alto nível I

As linguagens de muito alto nível têm uma estrutura ainda mais próxima da linguagem humana:

- Definem "o que" deve ser feito, e não "como" deve ser feito
- **Exemplo:** linguagens de consulta a banco de dados, como SQL

Linguagens de muito alto nível II

Exemplo: SQL, linguagem de consulta para manipular bases de dados.

nome	email	telefone	salario	cargo	*id
João da Silva	jsilva@swhere.com	7363-2334	2300	Gerente	1034
Carlos Ribas	cribas@cblanca.org	8334-2334	1800	Auxiliar	2938
Madalena Reis	mreis@portal.com	6451-5672	2000	Contador	7567
Patricia Horws	phorws@mail.com	4513-6564	2900	Gerente	2314
Carlito Fox	cfox@uol.com.br	5642-7873	1500	Auxiliar	5622
Ricardo Alves	ralves@portal.com	9302-4320	2000	Programador	6762

Apresentar os dados dos campos nome e telefone da tabela Funcionario:

```
SELECT nome, telefone FROM funcionario;
```

Tradução

Vimos que a conversão de **código fonte** (linguagem de alto nível) para **código executável** (linguagem de máquina) – tradução – é feita de 2 formas:

1. compilação
2. interpretação

Vejamos como funciona cada uma destas 2 formas.

Compilação I

- **Análise sintática e semântica**: Programa fonte escrito em uma linguagem qualquer (linguagem fonte) ⇒ **programa objeto** equivalente escrito em outra linguagem (linguagem objeto);
- **Compilador**: software tradutor em que a linguagem fonte é uma linguagem de alto nível e a linguagem objeto é uma linguagem de baixo nível (como **assembly**) ou de máquina.

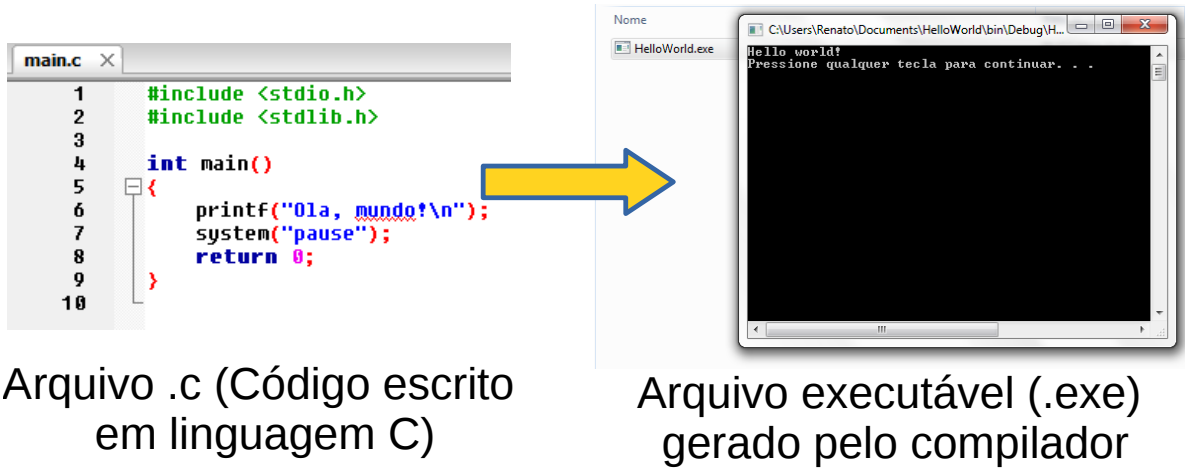


Figura 1: O compilador

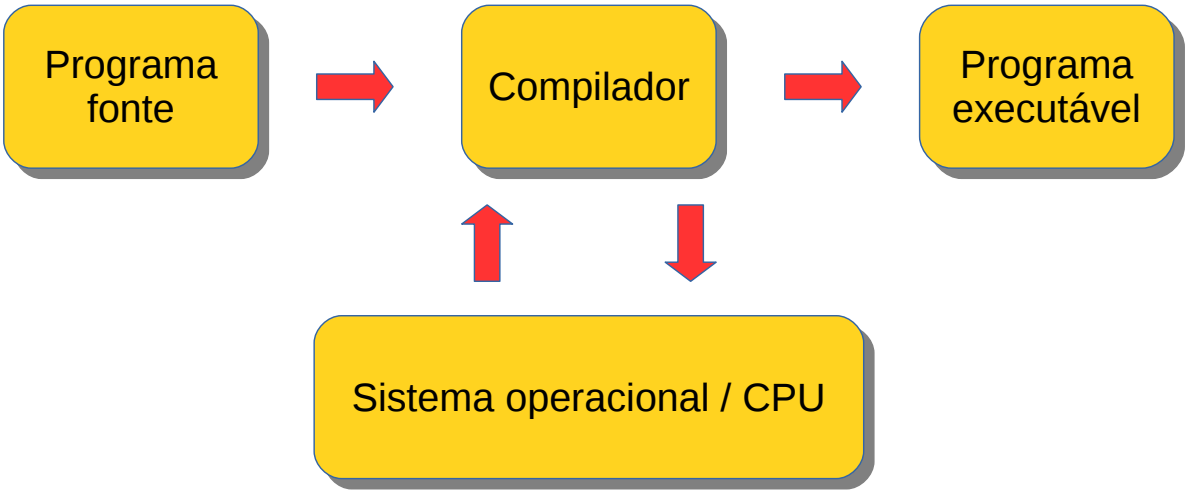


Figura 2: Geração do programa executável

Compilação IV

O código executável produzido não é portável

- Diferentes compiladores são construídos para as diferentes arquiteturas de processadores

Exemplos de linguagens **compiladas**:

- C
- Pascal
- FORTRAN
- C++

Interpretação I

- É um programa que interpreta (análise sintática e semântica) as instruções do programa fonte **uma a uma**, gerando o resultado;
- Toda vez que o programa é executado, o tradutor transforma cada instrução do código-fonte em uma instrução de código-objeto, que é imediatamente executada:
  - Não é criado todo um programa-objeto, mas apenas a conversão instrução a instrução.

Interpretação II

Um interpretador é um programa que executa repetidamente a seguinte sequência:

1. Obter o próximo comando do programa
2. Determinar que ações devem ser executadas
3. Executar estas ações

Caso haja alguma linha de código mal codificada (não respeitando o que foi definido na linguagem), o programa termina sua execução abruptamente em **erro**.

Interpretação III

Exemplos de linguagens **interpretadas**:

- HTML
- Haskell
- Lisp

Compilação × interpretação

- Compilação:
  - O programa fonte não é executado diretamente; O programa fonte é convertido em programa objeto e depois é executado; Vantagem: Execução muito mais rápida.
- Interpretação:
  - Executa-se diretamente o programa fonte;
  - Não existe a geração de um módulo ou programa objeto;
  - Vantagem: Programas menores e mais flexíveis.

Paradigmas de programação I

- Questão: como resolver um determinado problema?
- **Paradigmas de programação**
  - Relacionados à forma de pensar do programador;
  - Como ele busca a solução para o problema;
  - Mostra como o programador analisou e abstraiu o problema a ser resolvido.

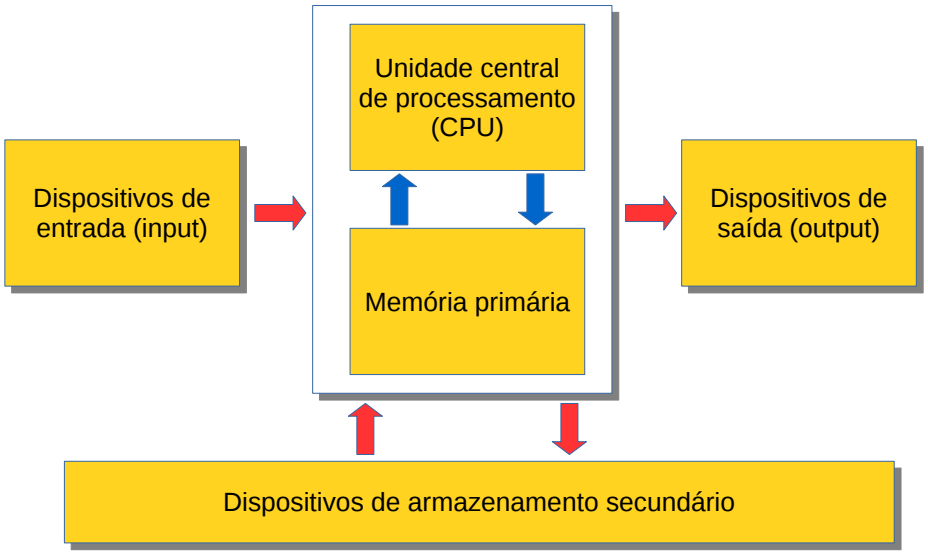
Paradigmas de programação são estilos utilizados pelos programadores para conceber um programa.
Um paradigma é um modelo ou padrão conceitual suportado por linguagens que agrupam certas características em comum
Respondem de forma diferente a perguntas como:

- O que é um programa?
- Como são modelados e escritos?

- Paradigmas
  - Estruturado;
  - Funcional;
  - Lógico;
  - Orientado a objetos;

Paradigma estruturado I

- Primeiro a surgir, ainda dominante
- Implementado com base na **máquina de von Neumann**



Paradigma estruturado II

- Também chamado **imperativo**;
- Utilização de 3 estruturas:
  - **Sequencial**;
  - **Condicional**;
  - **Repetição** ou **iterativa**.
- Busca quebrar um problema complexo em partes menores;
- **Programador**:
  - Analisa o problema;
  - Tenta relacionar ações que deverão ser executadas.

Paradigma estruturado III

O programador descreve a resolução de um problema através de uma série de tarefas elementares (comandos), que o computador pode compreender e executar

```
leia (num1)
leia (num2)
se (num1 > num2) então
   imprima (num1 é maior que num2)
senão
   imprima (num2 é igual ou maior a num1)
```

Ao final, a sequência de comandos define a resolução do problema;
A programação é dada por uma sequência de comandos que manipula um volume de dados.

Paradigma estruturado IV

Exemplos de linguagens: **Fortran**, C, Basic, Pascal

```
PROGRAM MAIN
INTEGER I, I_START, I_END, I_INC
REAL A(100)

I_START = 1
I_END = 100
I_INC = 1

DO I = I_START, I_END, I_INC

   A(I) = 0.0E0

END DO

END
```

Paradigma funcional I

- Qualquer computação é formulada em termos de funções;
- Funções são aplicadas aos parâmetros, e retornam um valor;
- As variáveis são os parâmetros formais das funções;
- Execução do programa = avaliar funções/expressões.

Paradigma funcional II

Exemplo da média de uma sequência de números:

```
Divide( Soma(Numeros), Conta(Numeros) )
```

- Estrutura de dados principal: listas
- O paradigma funcional é geralmente utilizado na área de **Inteligência Artificial (IA)**

Exemplos de linguagens: **Lisp**, Haskell , Miranda

```
(defun factorial (N)
   "Compute the factorial of N."
   (if (= N 1)
      1
      (* N (factorial (- N 1)))))
```


Perspectiva de um paradigma baseado no raciocínio: **o que** se quer, em vez de **como** atingi-lo
No paradigma lógico, programar é fornecer dados da seguinte natureza:

- **axiomas**, que indicam alguns fatos sobre o mundo
- regras para derivação de outros fatos (**inferência**)

Paradigma lógico ou declarativo II

No modo de programação:

```
homem(socrates).      %Socrates é um homem
mortal(X):-homem(X). % Todos os homens são
                     % mortais
```

No modo de pergunta (execução):

```
?- mortal(socrates).
Yes
```

O programa deve ter um grande volume de informações, denominados de fatos, para que o sistema chegue à solução.

Paradigma lógico ou declarativo III

Exemplos de linguagens: **Prolog**, GPSS

```
man(john).
man(adam).

woman(sue).
woman(eve).

married(adam, eve).

married(X) :-
   married(X, _).
married(X) :-
   married(_, X).

human(X) :-
   man(X).
human(X) :-
   woman(X).

% Who is not married?
?- human(X), not married(X).
   X = john ; X = sue
```

Paradigma orientado a objetos I

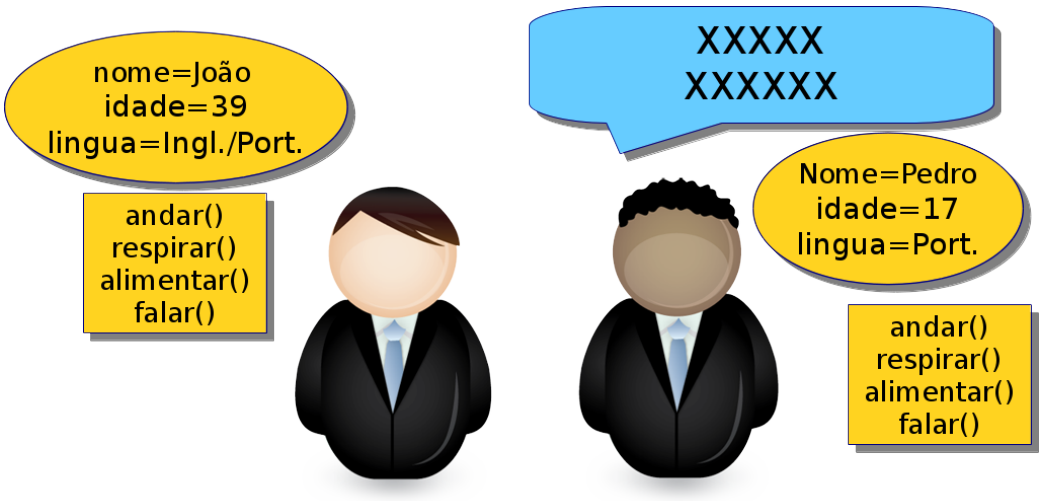
O programa é organizado em função de **objetos**.

Objeto

- Entidade independente com uma identidade e certas características próprias
- Um objeto contém não só as estruturas de dados, mas também as funções que sobre eles agem

Paradigma orientado a objetos II

A comunicação entre diferentes objetos ocorre por meio de **trocas de mensagens**:



Comparativo: Estruturado vs. OO

Programação estruturada:

- Preocupação maior é com as **estruturas de controle** (como construir programas usando as estruturas de sequência, seleção e repetição)
- Preocupa-se com os **algoritmos** e cada módulo que compõe um programa é um algoritmo específico

Programação orientada a objetos:

- Preocupação maior é com os **dados** que o programa irá tratar.
- Preocupa-se com as estruturas de dados e com as operações que podem ser executadas os dados.
- Cada estrutura de dados e suas operações é denominada **classe**.
- Cada módulo que compõe um programa é uma classe.

Exemplo de programação estruturada em C

```c
#include <stdio.h>
#include <stdlib.h>

// Define um novo tipo de dados
typedef struct {
   int num, den;
} fracao;

int mdc(int a, int b) { // Podem existir operacoes sobre outros tipos de dados
   int r;
   while (b>0) {
      r = a%b;
      a = b;
      b = r;
   }
   return a;
}

// cada possivel operacao sobre o tipo de dados e uma funcao independente
fracao simplificar(fracao a) {
   int m;
   m = mdc(a.num, a.den);
   a.num = a.num/m;
   a.den = a.den/m;
   return a;
}

fracao somar(fracao a, fracao b) {
   fracao c;
   c.num = a.num*b.den + a.den*b.num;
   c.den = a.den*b.den;
   c = simplificar(c);
   return c;
}

// A manipulacao de dados do novo tipo pode ser feita em qualquer funcao do programa
int main(int argc, char *argv[])
{
   fracao a,b,c;
   a.num = 5;
   a.den = 3;
   b.num = 2;
   b.den = 7;
   c= somar(a,b);
   printf("c = %d/%d\n", c.num, c.den);
   return 0;
}
```


Exemplo de programação OO em Java I

```java
// arquivo: Fracao.java
// A classe Fracao define a estrutura de dados e as operacoes possiveis sobre essa estrutura
public class Fracao
{
   private int den;
   private int num;

   public Fracao() {
      num = 0;
      den = 1;
   }

   public Fracao(int a, int b) {
      num = a;
      den = b;
   }

   private void simplificar() {
      int r,x,y;
      x = num;
      y = den;
      while (y>0) {
         r = x%y;
         x = y;
         y = r;
      }
      num = num/x;
      den = den/x;
   }

   public void somar(Fracao a, Fracao b) {
      den = a.den*b.den;
      num = a.num*b.den + b.num*a.den;
      simplificar();
   }

   public void mostrar() {
      System.out.println(num + "/" + den);
   }
}
```

Exemplo de programação OO em Java II

```java
// arquivo: Exemplo.java
// A classe Exemplo constroi OBJETOS da CLASSE Fracao e USA tais
     objetos como desejar.
public class Exemplo
{
   public static void main(String[] args)
   {
      Fracao a = new Fracao(5,3);
      Fracao b = new Fracao(2,7);
      Fracao c = new Fracao();
      c.somar(a,b);
      c.mostrar();
   }
}
```


Comparativo: Estruturado vs. OO

Na programação orientada a objetos, o principal conceito é o de **objeto**.

Programação estruturada

```
int x,y;
fracao a,b;

x = 5;
a.num = 3;
a.den = 2;
```

x e y são **variáveis** do tipo primitivo `int`, ou seja, exemplares do tipo `int`.
a e b são variáveis de um **novo tipo** `Fracao`, definido pelo usuário, ou seja, exemplares do tipo `Fracao`.

Programação orientada a objetos

```
int x,y;
Fracao a,b;

x = 5;
a = new Fracao(3,2);
```

a e b são **objetos** de uma **nova classe** `Fracao`, definida pela usuário, ou seja, **instâncias** da classe `Fracao`.
Uma classe define **todas** as **operações** (**métodos**) possíveis sobre seus objetos.

Especificando problemas I

Como fazer um programa a partir da especificação de um problema?

Exemplo: Desenvolva um programa que, dado um aluno, calcule a média das provas da disciplina de Programação Orientada a Objetos (POO), conforme critérios apresentados em aula.

Como você faria para resolver o problema?

Especificando problemas II

Algo como:

- Identificar os **dados** de entrada;
- Identificar que **resultados** o programa deve fornecer;
- Escrever uma **sequência de instruções** (**comandos**) para *manipular* os dados de entrada e gerar os resultados esperados.

Especificando problemas III

Uma especificação mais completa:

> Desenvolva um sistema para fazer a avaliação da disciplina de POO. O sistema deverá receber as notas de todas as avaliações realizadas pelos alunos (provas e projeto), bem como suas frequências e deverá gerar um relatório com as seguintes informações: nota final e situação de cada aluno, menor e maior nota da turma, nota média da turma, total de alunos aprovados, reprovados e de exame.

Modelagem em programação estruturada I

- **Para problemas maiores:** divide-se o problema em **funções** (procedimentos, sub-rotinas) com objetivos claros e uma interface bem definida com outras funções
- **Para problemas ainda maiores:** agrupam-se funções relacionadas em **módulos** (possivelmente em arquivos diferentes)

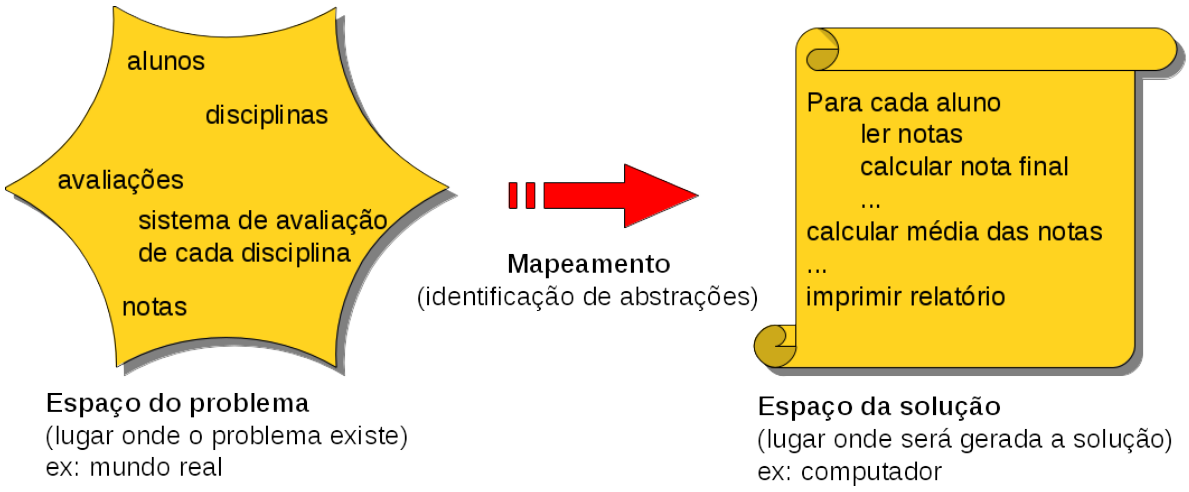
Programação estruturada

Desenvolvimento *top-down* (a. parte-se de uma solução inicial geral e vai decompondo o problema; b. resolvem-se os problemas menores).

Mas ainda assim as funções e módulos são **listas de instruções** para manipular **dados**.

Em programação estruturada, os programas são uma sequência de comandos (ordens) para o computador executar.

Modelagem I



A construção de um software envolve um processo de **mapeamento** de objetos pertencentes ao **espaço de problemas** para o **espaço de soluções**. De maneira que operações sobre essas representações abstratas correspondam a operações do mundo real.

Modelagem II

Quanto mais próximo (conceitualmente) o espaço de soluções estiver do espaço de problemas, mais fácil será:

- O desenvolvimento da aplicação;
- Assegurar a compreensão, confiabilidade e manutenção da aplicação.

Modelagem III

No **mundo real** lidamos com **objetos**, como *pessoas*, *carros*, *celulares*, *computadores*, ...

Tais objetos não são como **funções**:

Mundo real I

Um **objeto real** tem ambos:

- **Características** (atributos):
  - Pessoa: cor dos olhos, idade, nome, CPF, salário, ...
  - Carro: potência, número de portas, ano, modelo, ...
- **Comportamento** – ex.: resposta do objeto a algum estímulo:
  - Ao tocar o telefone, a pessoa atende;
  - Quando o freio é pressionado, o carro para.

Mundo real II

Em programação estruturada, as funções trabalham sobre os dados, mas não têm uma ligação íntima com os mesmos.

Mundo real III

A **programação orientada a objetos** é uma forma de pensar um problema utilizando conceitos do **mundo real**, e não conceitos computacionais:

- Objetos do mundo real são mapeados em objetos no computador;
- Funções e dados estão juntos, formando um objeto.

Orientação a objetos I

Etapas da modelagem:

- **Abstrair**: formular conceitos generalizados extraindo características comuns de exemplos específicos, descartando detalhes que não são importantes
- **Modelar**: criar um modelo que represente o objeto do mundo real

Orientação a objetos II

Exemplo da avaliação em POO:

- Que **objetos** podemos identificar?
- Existem grupos de objetos semelhantes (**classes** de objetos)?
- Quais são as características (**atributos**) de cada um?
- Quais são os possíveis **comportamentos** dos objetos?
- Como eles **interagem**?

Orientação a objetos III

Vantagens:

1. A modelagem do problema é mais simples;
2. Pode-se trabalhar em um nível mais elevado de abstração;
3. Maior facilidade para reutilização de código;
4. Maior facilidade de comunicação com os usuários e outros profissionais;
5. Redução das linhas de código programadas;
6. Separação das responsabilidades (classes e objetos podem ser desenvolvidos independentemente);
7. Maior flexibilidade do sistema e escalabilidade;
8. Facilidade de manutenção.

Linguagens orientadas a objetos I

Linguagens para programação OO são linguagens que têm facilidades para o desenvolvimento de programas OO.

- Mas o conceito depende mais da mentalidade do programador do que da linguagem de programação utilizada.
- É possível ter:
  - programas razoavelmente OO em linguagens imperativas;
  - programas estruturados em linguagens OO.

POO trata da organização geral do programa e não de detalhes de implementação.

Linguagens orientadas a objetos II

Algumas linguagens OO:

1. **SIMULA**
   - Final da década de 60;
   - Codificar simulações;
   - Primeira linguagem que implementava alguns conceitos de OO, como classes e herança.
2. **SMALLTALK**
   - Linguagem criada pela Xerox entre as décadas de 70 e 80;
   - Primeira linguagem orientada a objetos;
   - Ideias de SIMULA + princípio de objetos ativos, prontos a reagir a mensagens que ativam comportamentos específicos do objeto;
   - Pode ser considerada a linguagem OO mais pura.

3. **C++**
   - Década de 80;
   - Adaptar os conceitos da OO à linguagem C;
   - É uma linguagem híbrida (multi-paradigma: imperativo, OO, programação genérica).
4. **Java**
   - Desenvolvida nos anos 90
   - Popularizou a Orientação a Objetos
   - Baseada no C++
   - Linguagem altamente portável
   - Grande interação com a Web

Considere a especificação de notas de POO apresentada em slides anteriores. Faça um programa em uma linguagem estruturada (ex. C) para solucioná-lo.

Identifique os passos necessários à modelagem desse problema segundo o paradigma estruturado.

Referências

Os slides dessa apresentação foram cedidos por:

- Prof. M. Zanchetta do Nascimento, FACOM/UFU
- Profa. Katti Faceli, UFSCar/Sorocaba
- Prof. José Fernando R. Junior, ICMC-USP/São Carlos
- Prof. Senne, Unesp/Guaratinguetá
- Prof. Augusto Chaves, UNIFESP/SJCampos

Outras referências usadas: Apostilas de POO em:
http://www.riopomba.ifsudestemg.edu.br/dcc/dcc/materiais/1662272077_POO.pdf
http://www.jack.eti.br/www/arquivos/apostilas/java/poo.pdf
SEBESTA, ROBERT W., Conceitos de Linguagens de Programação, 5a ed., Bookman, 2003