FACULDADE SENAC BLUMENAU

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

- Disciplina: Programação Orientada a Objetos
- Semestre: 2º
- Docente: Prof. Pedro Edmundo Floriani

Apresentação

Prof: Pedro Edmundo Floriani

E-mail: pedro_edmundo@hotmail.com

Estudou em ensino público todo ensino fundamental

Graduação:

Ciências da Computação – FURB

Complementação Pedagógica - UNIVALI

Mestrado:

Computação Aplicada – UNIVALI

Outras formações:

Gestor, Analista e desenvolvedor de Sistemas – VKF Informática LTDA

Atuação Profissional

- Diversas empresas ligadas a desenvolvimento de software.
- **Funções:**
- Operador de computador
- Programador de computador
- Analista de sistemas
- Gerente de projetos
- Gerente de empresa e setor de desenvolvimento de software

- Professor em diversas Faculdades:
- Faculdade SENAC - Blumenau
- Faculdade UNIASSELVI – Blumenau
- Faculdade ASSEVIM – Brusque

Sobre o curso

PERFIL PROFISSIONAL DE CONCLUSÃO

O Tecnólogo em Análise e Desenvolvimento de Sistemas ao final do curso apresenta as seguintes competências:

- Analisa e especifica sistemas computacionais de informação.
- Desenvolve sistemas para ambiente Web, Desktop e dispositivos móveis.
- Diagnostica e propõe soluções computacionais de informação adequadas às regras ou ambientes do negócio.
- ...

PERFIL PROFISSIONAL DE CONCLUSÃO

- ...
- Conhece teorias e técnicas pautadas em ergonomia, segurança, qualidade e ética.
- Atua de forma ética, autonoma e pró ativa, com responsabilidade socioambiental e respeitando a diversidade sociocultural.
- Trabalha em equipe, interagindo em situações diversas, para atingir os objetivos organizacionais.

Sobre a Disciplina

Objetivo da disciplina

- Capacitar o aluno para criar programas com o paradigma da Orientação a Objetos.

Objetivo do curso

O Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas tem como objetivo formar profissionais capazes de analisar, especificar e desenvolver sistemas computacionais de informação, com conhecimento das teorias e técnicas inerentes, pautadas em ergonomia, segurança, qualidade e ética.

Ementa

- Estruturas básicas de programação Orientada a Objetos; Associações, multiplicidade e dependência; Associação de agregação; Associação de composição; Generalização, especialização e polimorfismo; Interface de Objetos; Pacotes.

Conteúdos Programáticos

Saber - Desdobramento da Ementa	Saber fazer - Habilidades	Critérios	Indicadores de Aprendizagem
1. Introdução a programação 1.1 Estruturas fundamentais do paradigma 1.2 Tratamento de exceções 1.3 Tratamento de String 1.4 Operações matemáticas fundamentais 1.5 Diferenças e semelhanças: Programação estruturada X POO	1. Identificar as particularidades entre programação estruturada e a POO	1. Comparação	1. Apresenta as particularidades entre programação estruturada e a POO

Conteúdos Programáticos

Saber - Desdobramento da Ementa	Saber fazer - Habilidades	Critérios	Indicadores de Aprendizagem
2. Coleções 2.1 Definição 2.2 Arrays unidimensionais 2.3 Arrays bidimensionais 2.4 Arrays multidimensionais 2.5 Listas	2. Utilizar coleções para manipulação de dados em uma linguagem de programação	2. Relacionar	2. Utiliza coleções para manipulação de dados em uma linguagem de programação

Conteúdos Programáticos

Saber - Desdobramento da Ementa	Saber fazer - Habilidades	Critérios	Indicadores de Aprendizagem
3 Orientação a Objetos 3.1 Estruturas básicas de programação Orientada a Objetos 3.2 Classes, atributos, pacotes e objetos 3.3 Manipulação de data e hora 3.4 Manipulação de arquivos 3.5 Eventos	3. Abstrair e descrever as estruturas de POO	3. Análise	3. Descreve as estruturas de POO

Conteúdos Programáticos

Saber - Desdobramento da Ementa	Saber fazer - Habilidades	Critérios	Indicadores de Aprendizagem
4 Criação de métodos 4.1 Métodos com e sem retorno 4.2 Recursividade 4.3 Sobrecarga e passagem de parâmetros 4.4 Encapsulamento	4. Criar métodos considerando os atributos e necessidades do objeto	4. Síntese	4. Cria métodos considerando os atributos e necessidades do objeto

Conteúdos Programáticos

Saber - Desdobramento da Ementa	Saber fazer - Habilidades	Critérios	Indicadores de Aprendizagem
5 Associações 5.1 Multiplicidade e dependência 5.2 Agregação 5.3 Composição 5.4 Generalização, especialização, polimorfismo e abstração 5.4.1 Sobrescrita 5.5 Introdução à Interfaces gráficas 5.6 Introdução a banco de dados	5. Desenvolver programas orientados a objeto utilizando paradigma da POO	5. Aplicabilidade e visão sistêmica	5. Desenvolve programas orientados a objeto utilizando paradigma da POO

Indicador Essencial

Desenvolve programas com o paradigma da Orientação a Objetos.

Recursos Didáticos:

- Textos;
- Bibliografias;
- Artigos;
- Projetor multimídia;
- Quadro;
- Laboratório de Informática;
- Biblioteca.

Estratégias metodológicas:

- Explanação dialogada (docente/discente);
- Debates;
- Trabalho em equipe;
- Análise e discussão de textos;
- Implementação de exercícios em laboratório de informática;
- Estudos de caso.

Instrumentos de avaliação:

- Mini seminários;
- Trabalhos individuais e em grupo;
- Avaliações escritas;
- Resenhas.

Atividade de Estudo Orientada (AEO):

- Pesquisas;
- Resenhas;
- Trabalhos individuais e em grupo;
- Participação em eventos;
- Atividades, leituras e exercícios.

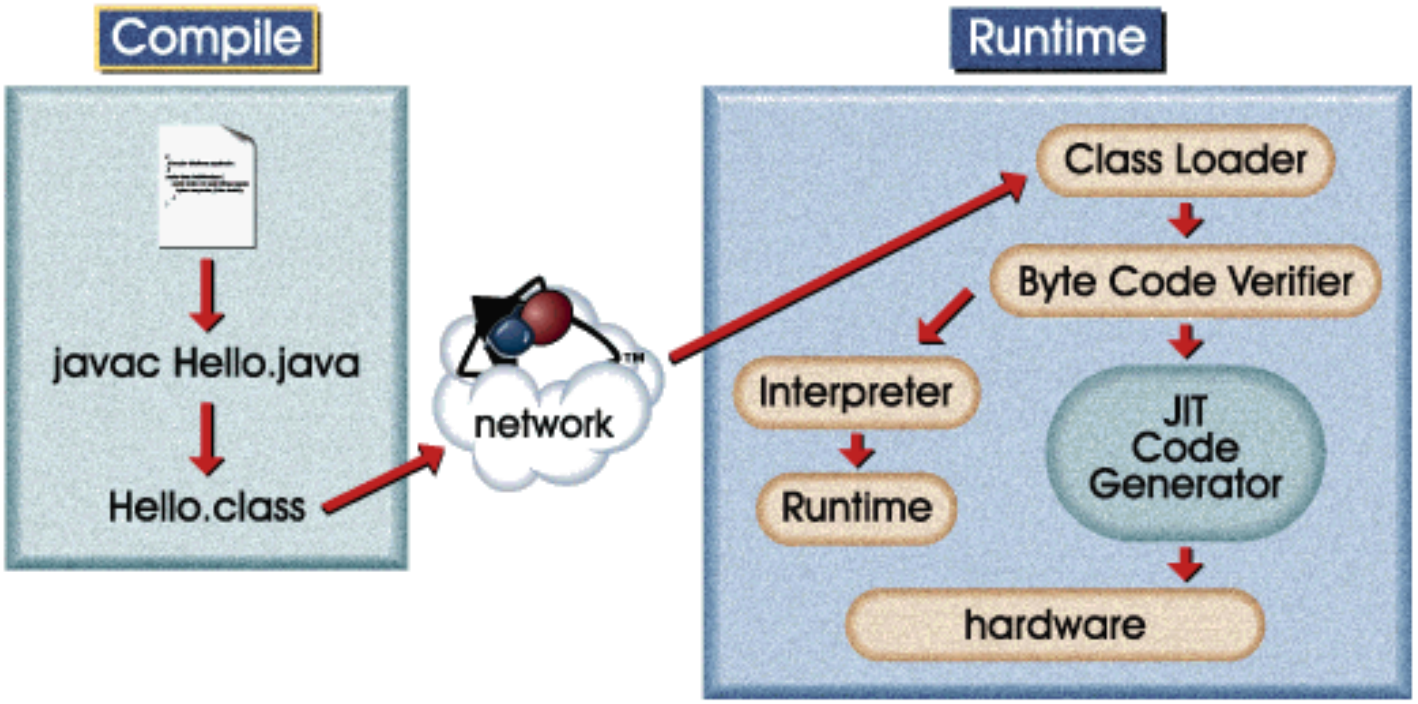
Linguagem Java

- Origem
  - linguagem originalmente desenvolvida para eletrodomésticos;
  - foi projetada para ser uma linguagem com características modernas de programação
  - nasceu considerando a Internet como ambiente operacional.
- Principais características
  - Propósito geral
  - Orientada a objetos e fortemente tipada
  - Robusta
    - sem ponteiros e alocação direta de memória
    - tratamento de exceções
  - Concorrente

Funcionamento

- Compilação do Fonte (.java) para *bytecode* da Java Virtual Machine (JVM)
- Interpretação e execução do *bytecode* (.class)
- “Escreva uma vez, execute em qualquer lugar”

Funcionamento



Java e a Internet

- A linguagem Java possibilita a construção de mini-programas - *applets* - que podem executar no *browser* do cliente.
    - Este Applets são automaticamente carregados da rede, dispensando a instalação de softwares no cliente.
- Os Applets permitem a construção de home pages com
    - animações, sons,
    - entrada/consistência de dados,
    - acesso a bancos de dados,
    - segurança, etc.
- Servlet
- JSP
- JSF

Outras vantagens da linguagem Java

- Facilidades para desenvolvimento de aplicações em redes com protocolo TCP/IP (sockets, datagrama)
- Gerência automática de memória (*garbage collection*)
- Vários fornecedores de ambientes de desenvolvimento
- Portabilidade
  - independência de plataforma de hardware e software

Programação em Java

Comentários

Tipos de dados

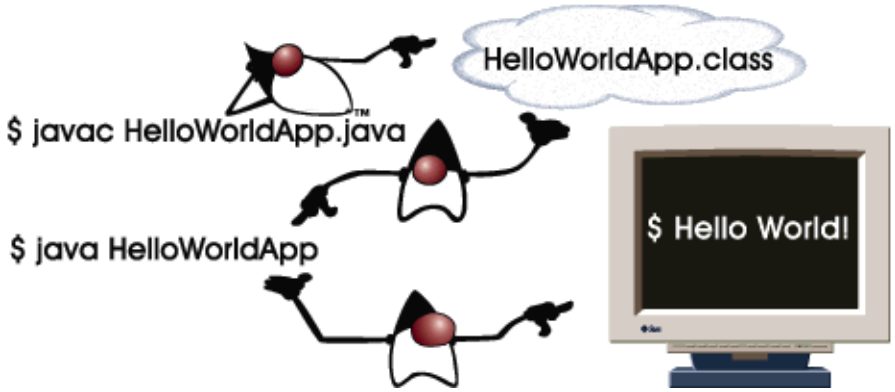
Literais

Operadores

Expressões

Variáveis

Fluxo de execução



Hello World

```
class HelloWorldApp {

   public static void main(String[] args) {

    System.out.println("Hello World"); //Exibe na tela

   }
}
```

Entendendo o Hello World

Tudo em Java funciona no interior de classes.

A estrutura de um programa Java é uma definição de classe.

O ponto de entrada de uma aplicação é o método **main**.

```
class HelloWorldApp {
   public static void main(String[] args) {
    System.out.println("Hello World"); //Exibe na
   tela
   }
}
```

Comentários

De única linha //

// Exibe na tela

De uma ou mais linhas /* */

/* comentário que vai se estendendo
até fechar com */

De documentação /** */

/ Indicam que o comentário deve ser inserido em qualquer documentação gerada automaticamente, por exemplo**
pelo javadoc . */

Ponto-e-vírgulas, blocos, espaços em branco e *case-sensitive*

Em Java as instruções terminam com ponto-e-vírgula (;).

Um *bloco* está sempre entre chaves - { e } e constitui uma instrução composta. Dois blocos aninhados são mostrados no exemplo:

Instruções compostas para a declaração de classe **{ }**

Instruções que abrangem a declaração main method **{ }**

Os *espaços em branco* são permitidos entre os elementos do código-fonte, sem qualquer restrição. Os espaços em branco incluem espaços, tabulações e novas linhas. Usados para melhorar a aparência visual e a legibilidade do código-fonte

Java é ***case-sensitive***, ou seja, maiúsculas são diferentes de minúsculas. Exemplo: TRUE != true

Tipos primitivos e seus valores

Inteiros

byte: 8 bits, -128 a 127
short: 16 bits, -32768 a 32767
int: 32 bits, -2147483648 a 2147483647
long: 64 bits, ... (200L - literal 200 long)

Reais

float: 32 bits (1f - literal 1 float; 1e+9f)
double: 64 bits (1d - literal 1 double; 47e-341d)

Caracter

char (Unicode caracter): 16 bits (‘a’ - literal)

Lógicos

boolean (1 bit). Valores literais: { true, false }

Literais

Literais inteiros (*int*)

- Decimais: 1 , 2, 45 ,...
- Octais: 07, 010
- Hexadecimais: 0xff (255)

Literais de ponto flutuante – decimais com fração (*double*)

- 2.0 , 3.14159 , ...
- 314159e-05 , 314159E-05

Literais booleanos (*boolean*)

- *true* e *false*

Literais de caracteres (*char*)

- Valores de 16 bits que podem ser manipulados como inteiros
- Par de apóstrofos (‘ ‘) ‘a’
- Seqüência de escape (\ ) ‘ \141’ octal = ‘ \u0061’ hex = ‘a’

A classe String

Representa qualquer seqüência de caracteres

Valores: “exemplo de valor literal string”

```
String nome = “João da Silva”;
```

```
nome.toUpperCase();  // mensagem para objeto nome (maiúsculo) .
```

Operador: + (concatenação)

```
nome = nome + “sauro”;
```

Observações

Strings são objetos, não são tipos primitivos, porém Java oferece facilidades de manipulação.

Strings são objetos que não mudam de valor

Principais métodos da classe String

boolean equals(String s)

retorna true se a string é igual a ***s***

boolean equalsIgnoreCase(String s)

retorna true se a string é igual a ***s*** independente de maiúsculo/minúsculo

int length()

retorna o tamanho da string

int indexOf(String s)

procura a ***s*** na string e retorna a posição, retorna -1 se não achou

char charAt(int i)

retorna o caracter na posição ***i*** da string (começa de 0)

Literais

Literais de string

Cria um objeto para cada literal de string

Texto arbitrário entre aspas (“ “)

“Hello World”

“duas\nlinhas”

Conversão de tipos

Tipos primitivos (**cast**)

```
char a;
int i = (int)a;
int x = (int)10L;
```

String em número

```
String piStr = "3.14159";
Float pi = Float.valueOf(piStr);
float pi = Float.parseFloat(piStr);
```

Operadores

Operadores para números e char

relacionais

`< > <= >= == !=` (resultado booleano)

aritméticos

`+ - * / %` (resultado numérico)

`++ --` (incremento/decremento)

`& (and) ^ (xor) | (or)` (para bits)

Operadores lógicos

booleanos

`== !=` (resultado lógico)

`&& (and) || (or)` ! (not) (resultado lógico)

Atribuição

`= += -=`

Expressões

Expressões avaliam (computam) o valor de uma sequência de variáveis, valores, operadores e chamada de métodos

Exemplos

- 2
- 2 * 4
- a == 3
- 5 + 2 * 3 - a
- 2 * Math.sqrt(9)

Precedência

- Explícita, utilizando parênteses
- Implícita: multiplicativo, aditivo, igualdade, &&, ||

Variáveis

Declaração

tipo nome [= *expressão*]

exemplos: `int a;  char c = 'a';  int d=3, e, f=5;`

pode ser feita em qualquer lugar do programa

Nome

Identificador válido em Java, diferente de palavras reservadas.

Válido = qualquer seqüência descritiva de caracteres de letras, números, caracteres de sublinhado e símbolo de cifrão. Não pode começar com número.

Por convenção, variáveis iniciam com letra minúscula

Variáveis

- Escopo
  - Classe (variável membro)
  - Método
- Tempo de vida
  - mesmo que escopo
- Visibilidade
  - método, classe, super-classe (nesta ordem)

2 – Estruturas de Seleção

- Uma estrutura de seleção permite a escolha de um grupo de ações e estruturas a ser executado quando determinadas condições , representadas por expressões lógicas, são ou não satisfeitas.

- Seleção Simples:

Sintaxe:

```
if (<Condição>)
    {
        <ações>;
    }
```

Seleção Simples

- <Condição> é uma expressão lógica que quando inspecionada, pode gerar um resultado falso ou verdadeiro.

  Se a condição for verdadeira o bloco de código é executado, caso contrário encerra o comando.

- Seleção Simples:

Sintaxe:

```
if (<Condição>)
    {
        <ações>;
    }
```

Seleção Composta

Sintaxe:

```
if (<Condição>)
    {
        <ações>;
    }
  else
        {
            <ações>;
        }
```

Seleção Encadeada

Sintaxe:

```
if (<Condição1>)
   {
        <ações>;
   }
  else
        if(<Condição2>)
           {
                <ações> ;
            }
        else
              if(<Condição3>)
                {
                     <ações>;
                }
              else  { <ação B> }
```

Seleção caso for

Sintaxe:

```
switch (<Condição1>)
   {
       case 1:  <ações>; break;
       case 2:  <ações>; break;
       case 3:  <ações>; break;
       default: <ações>;
   }
```

Controle de fluxo - Alternativa

if (*expressão*) *comando*
[**else** *comando*]

switch (*expressão*) {
  case *valor1*: *comando*;
[**break**];
  case *valor2*: *comando*;
[**break**];
  ....
  [**default:** *comando*;]
}

Repetição com teste no início *Enquanto*

Sintaxe:

```
while (<condição>)
 {
        <ação 1>;
        <ação 2>;
        <ação n>;
        <validação da condição>;
 }
```

Repetição com variável de controle

- A Linguagem de programação Java permite que o comando de repetição com variável de controle tenha três etapas: Inicialização, teste de condição para executar o laço e validação da variável de controle.

  Sintaxe exemplo:

```
for (int i = 0; i < 10; i++)
{     <ação 1>;
       <ação 2>;
       <ação n>;
 }
```

Repetição com variável de controle

Onde:

- `int i = 0` -> Declaração e inicialização da variável de controle, ponto de partida do laço;
- `i < 10` -> condição que permite executar o laço;
- `i++` -> validação da variável de controle.

Repetição com variável de controle

- A parte de inicialização pode ser optativa:

- Sintaxe exemplo:

```
...
i = 5;
for (; i < 10; i++)
      {     <ação 1>;
            <ação 2>;
            <ação n>;
       }
...
```

Repetição com variável de controle

- A parte de inicialização pode ser optativa:

- Sintaxe exemplo:

```
...
for (i = 5; i < 10; i++)
      {      <ação 1>;
             <ação 2>;
             <ação n>;
       }
...
```

Repetição com teste no final

```
boolean b = true;

do {
    float a = Float.parseFloat(JOptionPane
    .showInputDialog("Altura:"));
    if (a > 3)
      b = true;
    else
        b = false;
 }while (b);
```

Controle de fluxo - Repetição

```
while (expressão)
     comando

for (ini; cond; fim)
   comando

do {
    comandos1
} while (expressão);
```

break - sai do loop corrente

continue - volta para o teste do loop

Exemplo: I/O simples

```
import java.io.*;
class ContadorTeclas {
   public static void main(String[] args)
                  throws java.io.IOException
   {
      int conta = 0;
      while (System.in.read() != -1) {
         conta++;
      }
      System.out.println("\n Digitado " + conta +
" caracteres");
   }
}
```

Exemplo: I/O simples

```
import java.io.*;

class ExemploIO {
   public static void main(String[] args)
                throws java.io.IOException
   {
      BufferedReader teclado = new BufferedReader(
            new InputStreamReader(System.in));
      String s = teclado.readLine();
      String tudo = s;
      for (int cont = 0; cont < 5; cont ++) {
         s = teclado.readLine();
         tudo = tudo + s;
      }
      System.out.println("Digitado " + tudo);
}  }
```

I/O utilizando Swing

```
import javax.swing.*;
public class ExeSwing {

 public static void main(String[] args) {
   String nome = JOptionPane.showInputDialog(null,"Digite seu nome");
   int idade =  Integer.parseInt(JOptionPane.showInputDialog(null,"Digite
sua idade"));
   double valor =  Double.parseDouble(JOptionPane.showInputDialog(null,
                                                "Digite um valor"));
 }
}
```

I/O utilizando Swing

```
import javax.swing.*;
public class ExeSwing {

 public static void main(String[] args) {
   String nome = JOptionPane.showInputDialog(null,"Digite seu nome");
   int idade =  Integer.parseInt(JOptionPane.showInputDialog(null,"Digite
sua idade"));
   double valor =  Double.parseDouble(JOptionPane.showInputDialog(null,
                                                "Digite um valor"));
 }
}
```

Exercício

Faça um programa em java que verifique a idade de uma pessoa sendo informado o ano de seu nascimento.

Exercício

Faça um programa em java que entre com um valor qualquer e apresente como resultado o valor infomado mais 15,5%.

Exercício

Elabore um Programa em Java que entre com a idade de um nadador, mostre como resultado a categoria que ele pertence de acordo com a tabela a seguir:

Infantil A – 5 a 7 anos

Infantil B – 8 a 10 anos

Juvenil A – 11 a 13 anos

Juvenil B – 14 a 17 anos

Sênior maiores de 18 anos.

Exercício

A empresa XPTO Empreiteira Ltda pretende conceder um aumento de salários aos seus colaboradores. Sabendo que existem 3 faixas de salários e o aumento será concedido em reais conforme tabela abaixo:
Faça um um programa em Java que apresente como resultado o percentual de aumento para cada faixa salarial.

Faixa	Valor Salário em R$	Aumento em R$
1	1.000,00	193,56
2	1.001,00 até 2.000,00	174,34
3	Maior que 2.000,00	127,89

Exercício

- Construa um programa em Java que, dado um conjunto de valores inteiros e positivos, determine qual o menor e o maior valor do conjunto. O final do conjunto é conhecido através do valor 0 (zero) que não deve ser considerado.
- Elabore um programa em Java que, dado a idade e o sexo de um grupo de pessoas, apresente no final: a média de idade das pessoas; o percentual de pessoas do sexo masculino e feminino; a maior idade masculina e a menor idade feminina. Para finalizar o sexo deverá ser igual a N.

Exercício

- Faça um programa em Java que efetue o cálculo da quantidade de litros de combustível gastos em uma viagem, utilizando um automóvel que faz 12 Km por litro. Para obter o cálculo, o usuário deverá fornecer o tempo gasto na viagem e a velocidade média durante a mesma. Dessa forma será possível obter a distância percorrida com a fórmula: Distância = (Tempo * velocidade). Tendo o valor da distância, basta calcular a quantidade de litros gasto na viagem, da distância percorrida e da quantidade de litros utilizada na viagem.

Exercício

•Elabore um programa em Java que determine o grau de obesidade de uma pessoa, sendo fornecido o seu peso e a sua altura. O grau de obesidade é determinado pelo índice da massa corpórea (Massa = Peso / (Altura * Altura)) conforme tabela abaixo:

Massa Corpórea	Grau de Obesidade
< 26	Normal
>= 26 e <= 30	Obeso
> 30	Obeso Mórbido

Exercício

- Faça um programa em Java que calcule o valor de H, sendo que ele é determinado pela série:
  H = 1/1 + 3/2 - 5/3 -7/4 +9/5 + 11/6... 99/50.

- Elabore um programa que mostre como resultado a soma dos dez primeiros elementos da seguinte série: 2/500 - 5/450 + 2/400 - 5/350 +...

Exercício

Foi realizada uma pesquisa de algumas características físicas da população de determinada cidade, a qual coletou os seguintes dados referentes a cada habitante para serem analisados:

- Sexo (Masculino/Feminino);
- Cor dos olhos (azuis, verdes, castanhos);
- Cor dos cabelos (louros, castanhos, pretos);
- Idade.

Faça um programa em Java que apresente como resultado:

a. A média de idade dos habitantes;
b. Percentual de pessoas acima de 60 anos de idade em relação ao total de pessoas;
c. A quantidade de pessoas do sexo feminino cuja idade esta entre 17 e 36 anos e que tenham olhos verdes e cabelos louros;
d. O percentual de pessoas do sexo masculino com idade superior a 18 anos e que tenham cabelos pretos e olhos verdes ou castanhos.

Obs. Para finalizar N no lu

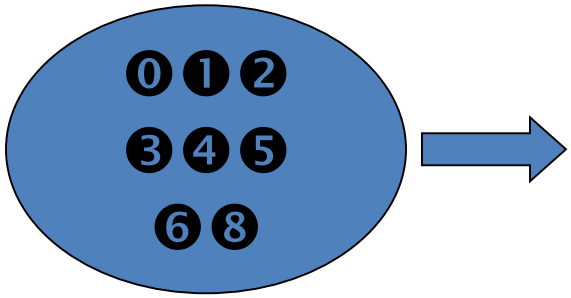
Estrutura de dados

- A quantidade de tipos de informação estipulados (tipos primitivos) não é suficiente para representar toda e qualquer informação que possa surgir.

- Construiremos novos tipos, denominados "tipos construídos" a partir da composição de tipos primitivos. Esses novos tipos tem um formato denominado ESTRURA DE DADOS, que define como os tipos primitivos estão organizados.

Variáveis compostas homogêneas

- Quando uma determinada estrutura de dados for composta de variáveis com o mesmo tipo primitivo, temos um conjunto homogêneo de dados.



Temos um conjunto de dados do tipo primitivo inteiro, ou seja, temos um conjunto homogêneo de dados.

Variáveis compostas unidimensionais

- Quando uma determinada estrutura de dados for composta de uma única dimensão (Vetor).

Dados	**Ana**	**Bruno**	**Carol**	**Dani**	**Edson**	**Fabio**	**Gino**
Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Variáveis compostas unidimensionais

- Declaração em java:

Tipo de dado [] identificador = new tipo de dado [quantidade];

Onde:

tipo de dado -> Representa qualquer tipo de variável;

identificador -> Representa o nome da estrutura;

quantidade -> a quantidade de elementos que o vetor pode manipular;

Variáveis compostas unidimensionais

- exemplos em java:

int [] numeros = new int[7]; -> cria um array com o nome **numeros** que contém 7 elementos do tipo int e seu índice varia de 0 a 6.

Dados	**500**	**456**	**678**	**345**	**3**	**16**	**234**
Índice	**0**	**1**	**2**	**3**	**4**	**5**	**6**

Variáveis compostas unidimensionais

- exemplos em java:

String [] diaSemanas = new String[7]; -> cria um array com o nome **diaSemanas** que contém 7 elementos do tipo String e seu índice varia de 0 a 6.

Dados	Dom	Seg	Ter	Qua	Qui	Sex	Sab
Índice	0	1	2	3	4	5	6

Variáveis compostas unidimensionais

- Para atribuir um valor a um elemento do array, deve-se indicar o índice desejado dentro dos colchetes, como nos exemplos a seguir:

```
numeros[0] = 100;    numeros[5] = 38;    numeros[2] = 4;
```

```
diaSemana[0] = “Dom”; diaSemana[3] = “Qua”;
```

Variáveis compostas unidimensionais Exemplo

```
1       public static void main(Strings[] args) {
2       int [] numeros = new int[10];
3       for (int i = 0; i < 10; i++) {
4           numeros[i] = (int) (Math.random()*100);
5           System.out.println(numeros[i]);
6       }
7        System.exit(0);
8     }
9  }
```

Linha 2: Declara o array unidimensional chamado numeros contendo dez elementos (índices de 0 a 9)

Linha 3: Contém um laço de repetição com a instrução for que faz com que as linhas 4 e 5 sejam executadas dez vezes . O valor da variável "I" inicia em zero e é incrementado em 1 até o limite estabelecido pelo laço.

Linha 4: Armazena no array numeros um valor gerado aleatoriamente. A cada ciclo de execução do laço o numero gerado é armazenado num elemento diferente do array.

Linha 5: Imprime em tela o número gerado que foi armazenado no elemento do array.

Variáveis compostas unidimensionais Exemplo

Os arrays podem ser criados e inicializados de outra maneira. Em vez de usar o operador new é possível definir os elementos do array entre chaves e separados por virgula:

```
1       public static void main(Strings[] args) {
2       int [] numeros = {234,345,567,54,3,2,5};
3       for (int i = 0; i < numeros.length; i++) {
4          System.out.println(numeros[i]);
5      }
7        System.exit(0);
8     }
9  }
```

Linha 2: Declara o array unidimensional chamado numeros contendo os elementos.

Linha 3: Contém um laço de repetição com a instrução for que faz com que as linha 4 seja executada enquanto existir elementos no vetor .

Variáveis compostas unidimensionais

- Faça um programa em java que carregue um vetor de 10 posições de valores inteiros e mostre como resultado o maior e o menor valor do conjunto.

- Faça um programa em java que carregue dois vetores de 5 posições de valores inteiros e mostre como resultado um terceiro vetor coma a soma dos valores lidos.

Variáveis compostas unidimensionais

- Faça um programa em java que que peça para o usuário informar a quantidade de alunos. Armazenar em um array de strings o nome dos alunos, para cada aluno carregar 3 notas e armazenar a média em um array de float. No final emitir uma lista com o nome do aluno e sua respectiva média.

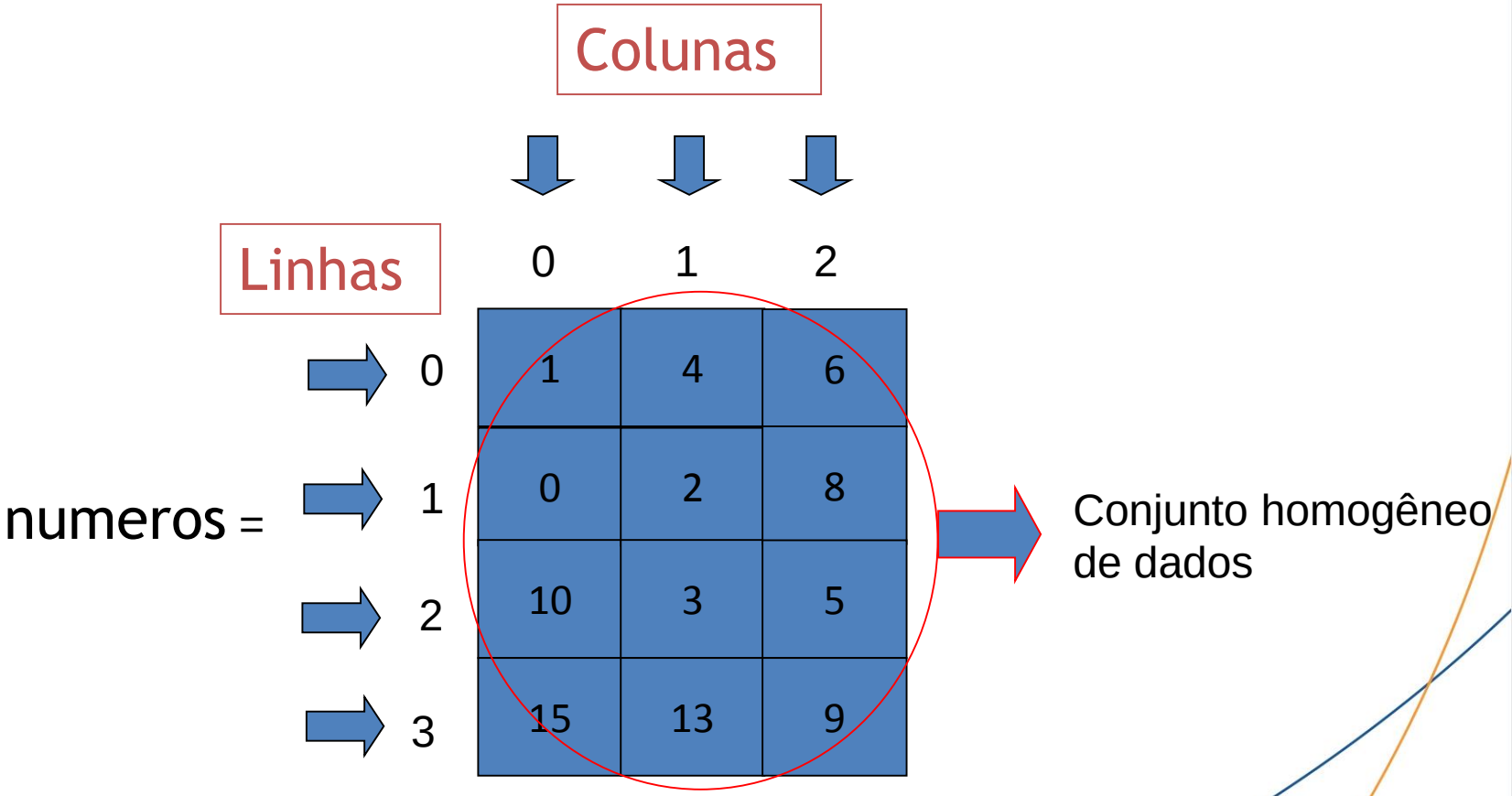
Variáveis compostas unidimensionais

- Faça um programa em java que carregue dois arrays A e B de float de 20 posições. Armazenar em um terceiro vetor C a multiplicação dos elementos dos vetores lidos seguindo a seguinte ordem:

  C[0]= A[0] * B[19]
  C[1]= A[1] * B[18]
  ...
  Ao final imprimir o array C.

Variáveis compostas multidimensionais

- Exemplo: Tipo de dado numeros [][] = new tipo de dado [4][3];



Variáveis compostas multidimensionais

- Para atribuir um valor a um elemento do array, deve-se indicar os índices desejados dentro dos colchetes, como nos exemplos a seguir:

```
numeros[0][1] = 100;    numeros[4][2]  = 38;

numeros[2][2]  = 4;     numeros[3][2]  = 20;
```

Variáveis compostas multidimensionais

- Exercício:

  Construa um algoritmo que carregue uma matriz 4X4 de inteiros e mostre como resultado:

  a) os valores da diagonal principal;

  b) A soma dos números impares;

Variáveis compostas multidimensionais

- Exercício:

Crie um programa em Java com uma matriz chamada *MatrizDeReais4x2*, com 4 x 2 números reais (*float*).

a) Receber os valores;
b) Retornar a multiplicação dos valores de uma dada coluna;
c) Retornar a soma dos valores de uma dada linha;
d) Retornar a soma de todos os elementos da matriz.

Variáveis compostas multidimensionais

- Exercício:

Considere uma matriz quadrada de inteiros de ordem N. Crie um programa em Java que resolva as seguintes questões:

a) a soma dos elementos da diagonal principal da matriz;
b) o menor valor par da matriz;
c) o valor do elemento [x , y];
d) uma nova matriz de ordem N, porém transposta.

Variáveis compostas multidimensionais

- Exercício:

Considere um vetor de String de tamanho N. Crie um programa em Java que contenha um vetor chamado de *VetorString* que tenha as seguintes funcionalidades:

a) o tamanho N é definido no momento da criação do objeto de *VetorString*;
b) possam ser adicionadas as Strings, uma de cada vez;
c) verifique se uma String está armazenada no *VetorString*;
d) retorne a posição da menor String.

Criação de Métodos em Java

- Métodos são trechos de código que permitem modularizar um sistema;
- Recebem um nome;
- Podem ser chamados várias vezes durante a execução do programa;
- Métodos podem ser usados (importados) por outros programas (classes).

Criação de Métodos em Java

- Um método pode ser invocado por outro, durante a execução do metodo1 pode ser necessária a execução do metodo2 que pode invocar o metodo3 e assim por diante.

- Todo método possui uma declaração e um corpo:

```
qualificador tipo-de-retorno nome-do-método(lista-de-parâmetro) {
código-do-corpo
}
```

- **Qualificador:** Define a visibilidade do método.
  - public: o método pode ser visível por qualquer classe;
  - private: visível apenas pela própria classe;
  - protected: visível apenas pela própria classe, por suas subclasses e pelas classes do mesmo pacote.

Criação de Métodos em Java

- qualificador tipo-de-retorno nome-do-método(lista-de-parâmetro) {
  código-do-corpo
  }

- **tipo-de-retorno** : Tipo de dado retornado pelo método. Métodos que não retornam valores devem possuir nesse parâmetro a palavra void.

- **nome-do-método** : Pode ser qualquer palavra ou frase, seguindo a regra de formação de identificadores.

- **lista-de-parâmetro**: Variáveis opcionais que podem ser recebidas pelo método para tratamento interno.

- **código-do-corpo** : Código implementado para a realização do método.

```
public class exemploMetodo {
public static void main(String[] args) {
imprimir();
int m = maior(10, 20, 30);
String dia = diaSemanaExtenso(3);
imprimirTexto("Maior = " + m + "\ndia = " + dia);
}

public static void imprimir(){
System.out.println("Aprendendo Java!!"); }

public static void imprimirTexto(String texto){
System.out.println(texto);  }

public static int maior(int a, int b, int c){
return Math.max(c, Math.max(a, b)); }

public static String diaSemanaExtenso(int dia){
switch (dia) {
  case 1: return "Domingo"; case 2: return "Segunda";
  case 3: return "Terça";   case 4: return "Quarta";
  case 5: return "Quinta";  case 6: return "Sexta";
  case 7: return "Sabado";  default: return "";  } }
}
```

- **Sobrecarga:** Métodos com o mesmo nome, desde que eles tenham assinaturas diferente, ou seja, número ou ordem de parâmetro recebido.
- Exedmplo:

```
public class exemploMetodoSobrecarga {
public static void main(String[] args) {
 System.out.println("Área de um quadrado = " + area(3));
 System.out.println("Área de um retângulo = " + area(3, 2));
 System.out.println("Área de um cubo = " + area(3, 2, 5));
 }
 public static double area (int x){
   return (x * x);
 }

 public static double area (int x, int y){
   return (x * y);
 }

 public static double area (int x, int y, int z){
   return (x * y * z);
 }
}
```

- Exercício:

- Faça um programe em Java que permita o usuário informar dois valores quaisquer . Apresentar como resultado o valor das quatro operações básicas da matemática, sendo que estes deverão ser implementados em métodos.

- Crie uma classe em Java que contenha três métodos com o nome “media” utilizando o conceito de sobrecarga. Os métodos devem calcular a média de dois, três ou quatro valores de entrada.

- Exercício:

- Elabore uma classe chamada “Buscas” contendo dois métodos que realizam operações com arrays, conforme a descrição seguinte:

a) um método que recebe um número inteiro e um array de inteiros, pesquise se esse número existe no array e retorna true (caso o número exista no array) ou false (caso não exista). A assinatura do método pode ser buscaNumero(int numero, int[ ] vetor).

b) um método que recebe um array de Strings, verifica se nos elementos desse array existe a palavra “computação” e retorne sim caso exista e não caso contrário. A assinatura do método pode ser buscaPalavra(String[ ] palavras).

c) Crie um terceiro método para testar os métodos anteriores.

Orientação a Objetos

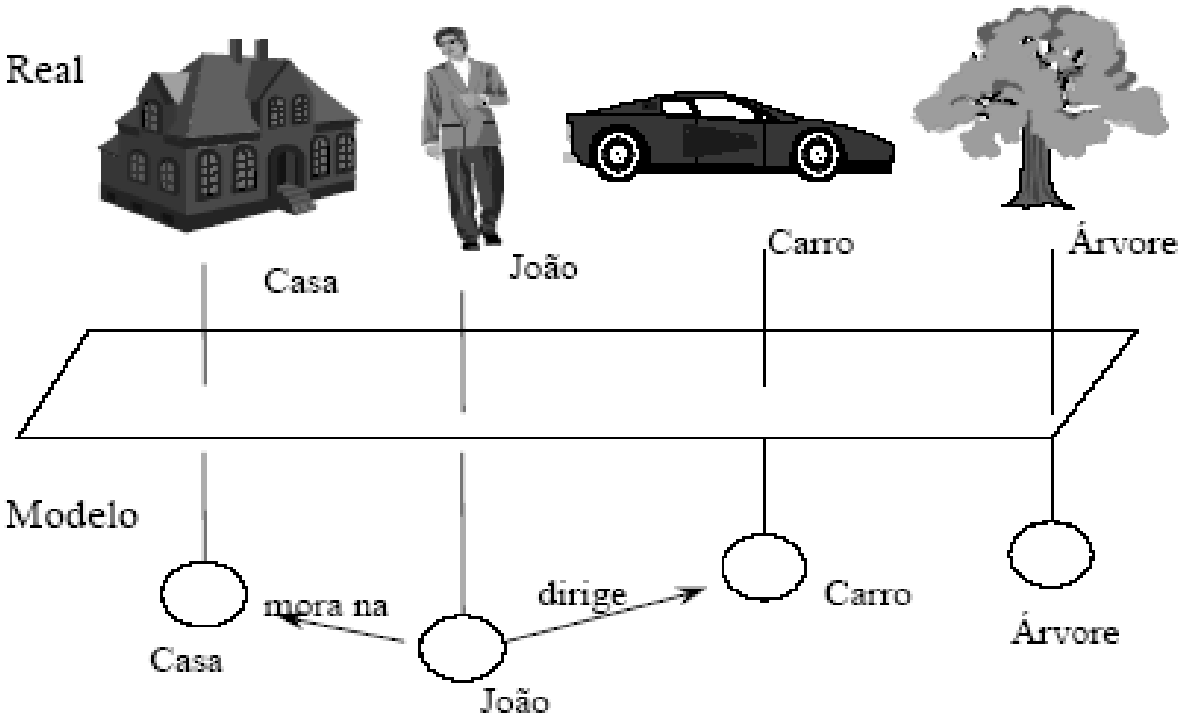
- Novo paradigma de desenvolvimento;
- Os sistemas são vistos como sendo uma coleção de objetos inter agentes;
- Envolve todas as atividade de desenvolvimento: (análise, projeto, programação, testes,...)

Fundamentos da OO

Na compreensão do mundo, os seres humanos utilizam-se de três métodos de organização dos pensamentos:

- Diferenciação;
- Distinção entre todo e parte
- Classificação

Diminuir a diferença semântica



Benefícios da Orientado a Objetos

- Reduz complexidade através da melhoria do grau de abstração;
- Melhoria de produtividade ao longo prazo;
- Reduz o esforço de codificação;
- Melhoria da confiabilidade;
- Facilita a manutenção

Classes

- Descreve um entidade real ou abstrata;
- Especificação de um objeto

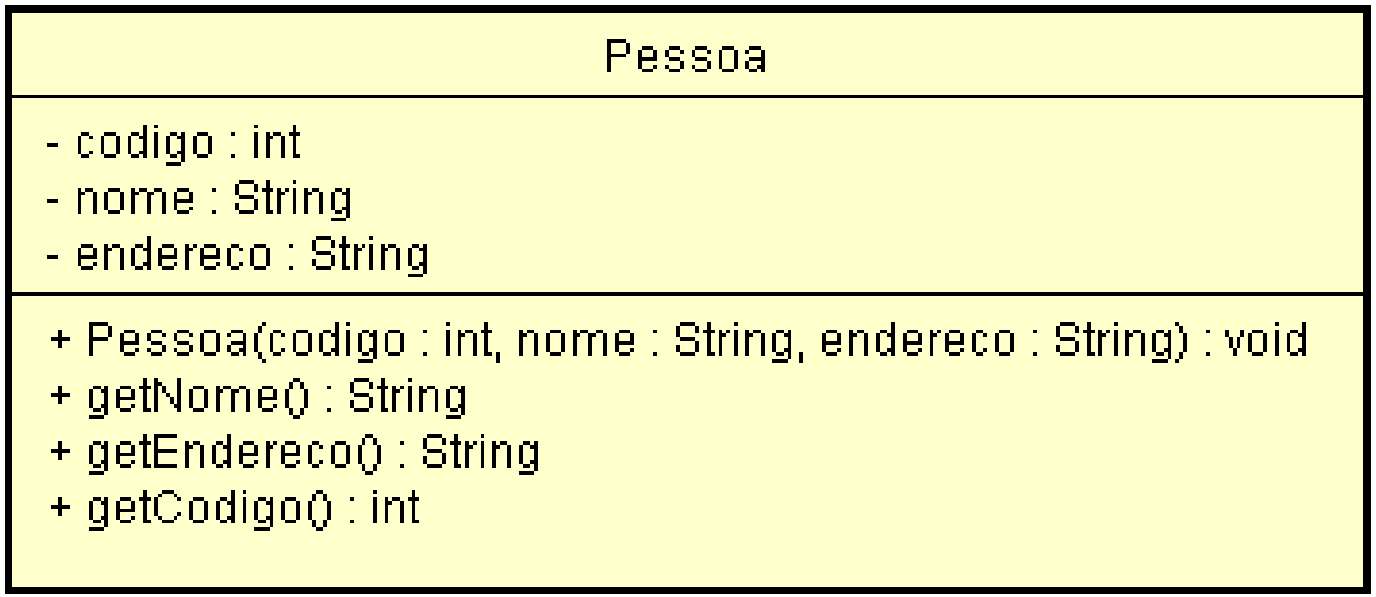
```
public class Classe01 {

  //atributos
  //metodos
  //outros membros

}
```

Construtores/Destrutores

- Construtores são procedimentos utilizados na construção de objetos;
- Destrutores executados antes do objeto ser finalizado

```
public class Classe01 {
 //construtores
 public Classe01() {
 }
 public Classe01(String nome) {
 }
 //destrutor
 public void finalize(){
 }
}
```

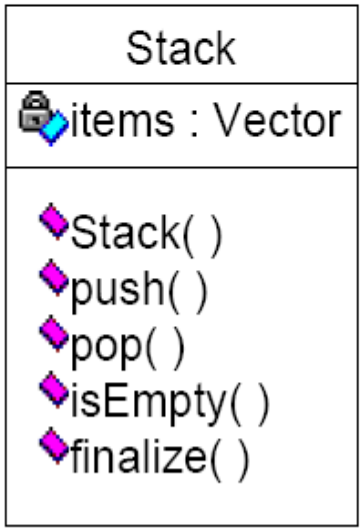
Exemplo UML (Unifed Modeling Language)



Exemplo

Representação em UML

(Unified Modeling Language)



Java →

Class Declaration

Class Body

Variable

Constructor

Methods

Cleanup

```
public class Stack
{
    private Vector items;
    public Stack() {
        items = new Vector(10);
    }
    public Object push(Object item) {
        items.addElement(item);
        return item;
    }
    public synchronized Object pop() {
        int len = items.size();
        Object obj = null;
        if (len == 0)
            throw new EmptyStackException();
        obj = items.elementAt(len - 1);
        items.removeElementAt(len - 1);
        return obj;
    }
    public boolean isEmpty() {
        if (items.size() == 0)
            return true;
        else
            return false;
    }
    protected void finalize() throws Throwable {
        items = null;
        super.finalize();
    }
}
```

Exemplo Java

```
public class Pessoa {
     //atributos
    private int codigo;
    protected String nome;
    protected String endereco;
    //construtor
    public Pessoa(int codigo,String nome, String endereco) {
      this.codigo = codigo;
      this.nome = nome;
      this.endereco = endereco;
    }
    //métodos
    public int getCodigo() {
      return codigo;
    }
    public String getNome() {
      return nome;
    }
    public String getEndereco() {
       return endereco;
    }
}
```

Outro exemplo

```
public class Classe01 {
 //atributos
 private final String constante = "QQ";
 private String nome = "Java";
 static int qtd;
 protected String endereco;

 //construtores
 public Classe01() {
 }

 final String fazAlgumaCoisa(int c, String s, double
d) {
     return "abc";
 }

 //destrutor
 protected void finalize(){
 }
}
```

Pacotes (package)

Conjunto de classes relacionadas

Sintaxe:

```
import pacote.classe;
```

Exemplos:

```
import java.util.ArrayList;
import java.sql.*;
```

Alguns pacotes da API Java

java.awt

java.io

java.util

java.lang

javax.swing

java.sql

java.math

Etc

Classe java.lang.Object

- Métodos
  - public boolean equals(Object obj)
  - public String toString()
  - public int hasCode()
  - protected Object close() throws
  - CloneNotSupportedException

Objetos

- É a materialização (instanciação) de uma Classe
- Possui estado ou características (atributos)
- Possui comportamento (métodos)
- Possui identidade (variável que contém sua referência)

Exemplo

```
import java.util.*;
...
  Date data = new Date();
  ArrayList lita = new ArrayList(15);
```

Encapsulamento

Impede que os atributos da classe possam ser manipulados externamente

- private
- protected

- Exercício:

Uma pessoa comprou 3 artigos em uma loja. Para cada artigo ela tem o nome, preço e o percentual de desconto sobre o preço. Faça um programa Java que imprima nome, preço, preço com desconto de cada artigo e o preço total a pagar, enquanto preço for diferente de 0 (zero).
Obs: Utilizar Estrutura de Classe.

- Exercício:

Faça um programa em Java que dado a idade e o sexo de um grupo de pessoas, apresente no final: o percentual de pessoas do sexo masculino e feminino; a maior idade masculina; a menor idade feminina.
Para finalizar informar N para o sexo.
Obs: Utilizar Estrutura de Classe.

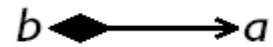
A secretaria de transportes de uma determinada cidade quer ter um controle melhor sobre os veículos que circulam na cidade, para tanto solicitou ao setor de TI uma solução para o problema, este resolveu fazer um levantamento dos veículos existentes na cidade. Foram informados para cada veiculo: A placa, ano, motor, chassi, cor e montadora.
Pede-se:

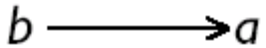
- A quantidade de veículos existente na cidade;
- A quantidade de veículos com mais de dez anos de circulação;
- A quantidade de veículos com mais de vinte anos de circulação;
- O percentual de veículos com menos de cinco anos de circulação em relação ao total de veículos;
- A quantidade de veículos cor branca e montadora FIAT;
- O percentual de veículos com motor 1.0 da montadora Renault cujo a cor seja prata e o ano seja superior 1995.
- Uma lista com todos os veículos cujo a primeira letra da placa seja vogal e o ultimo numero da placa seja par.

Relacionamento entre classes

Tipos de Relacionamento:

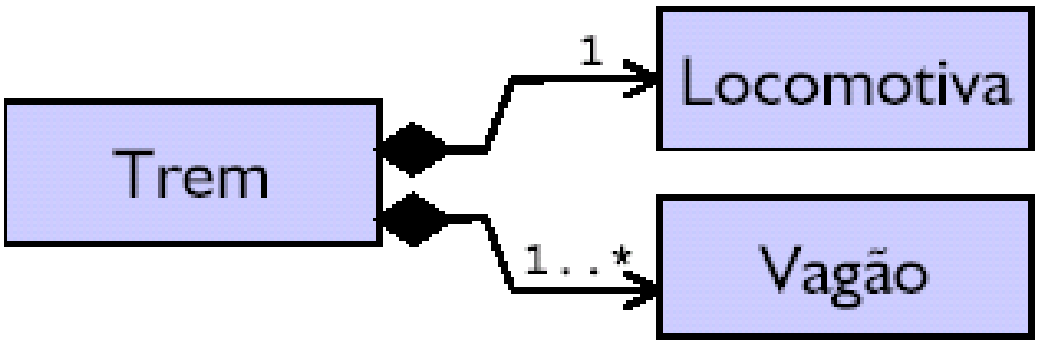
- Composição: Classe A é parte essencial da Classe B
- Agregação: Classe A é parte de B
- Associação: Classe A é usado por B





Composição

- Composição: um trem é formado por locomotiva e vagões.

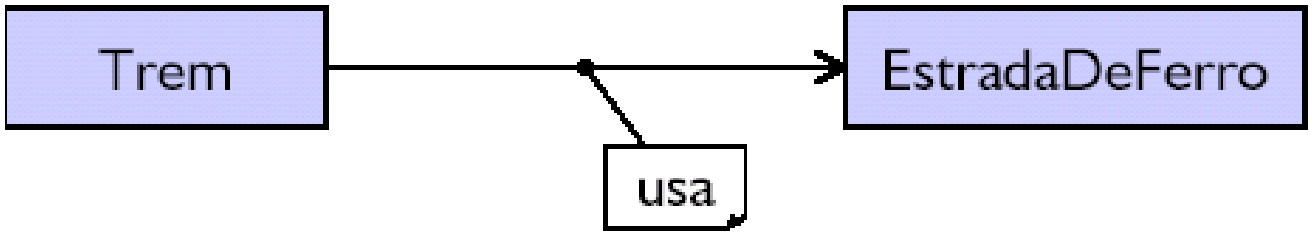


Agregação

- Agregação: uma locomotiva tem um farol (mas não vai deixar de ser uma locomotiva se não o tiver.

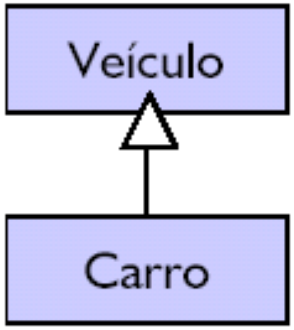
Associação

- Associação: o trem usa uma estrada de ferro.

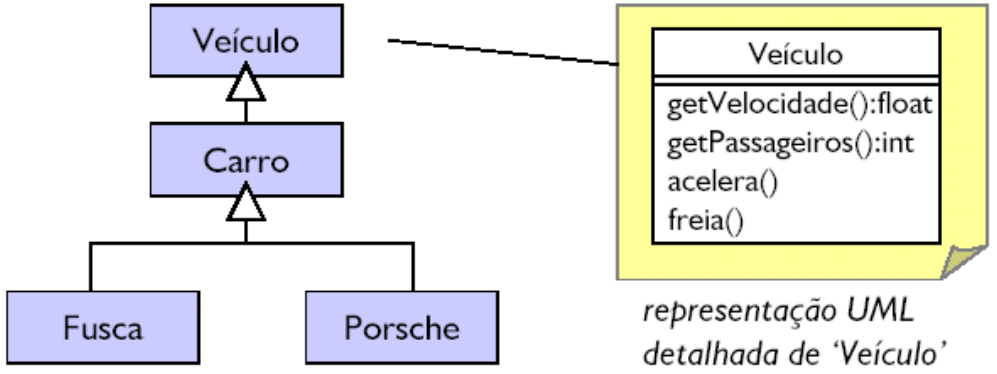


Herança

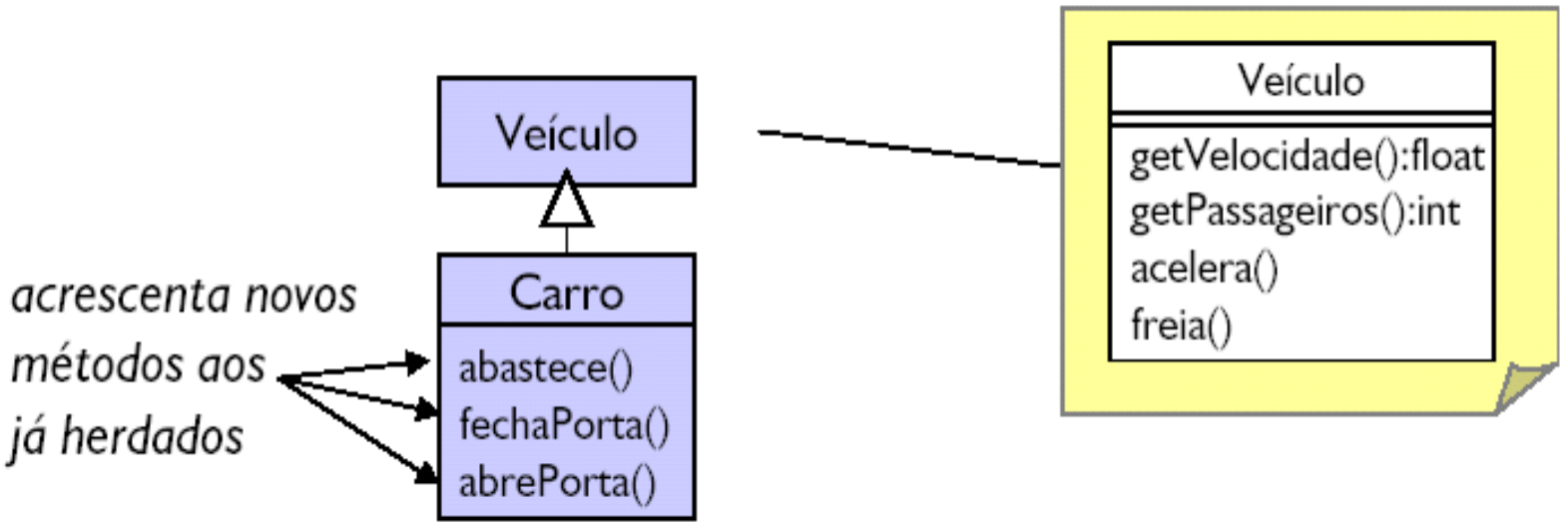
- Um carro é um veículo



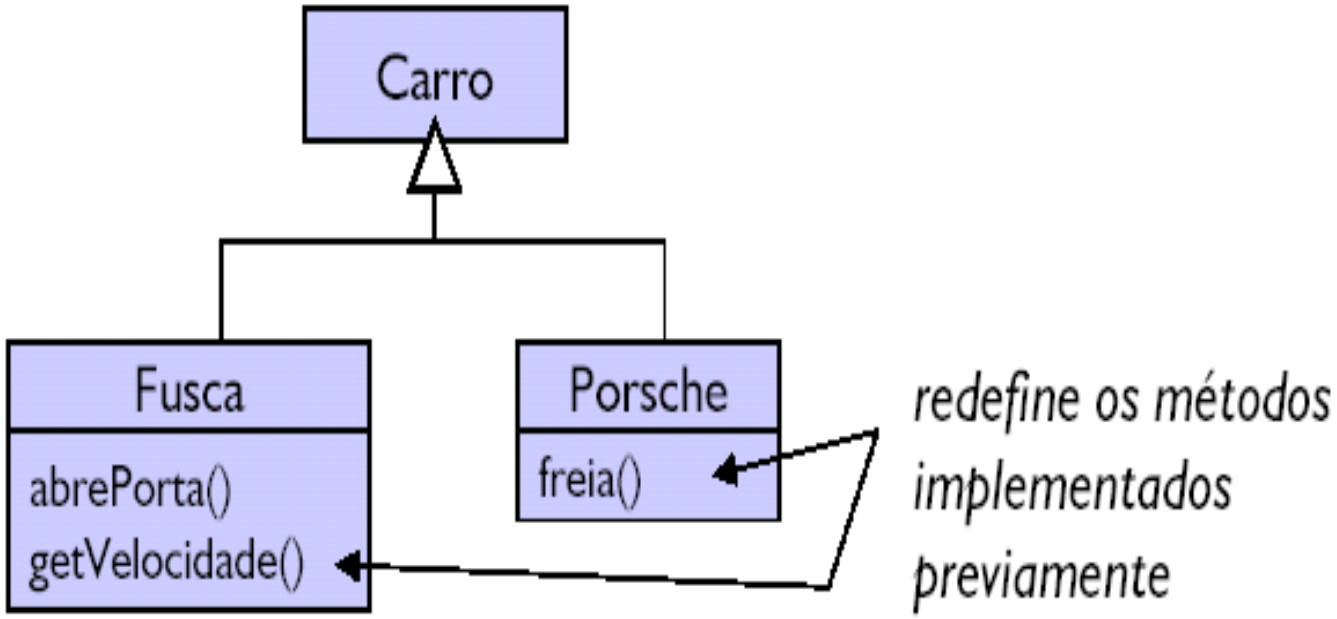
- Fuscas e Porches são carros



Extensão


Veículo
Carro
abastece()
fechaPorta()
abrePorta()
acrescenta novos
métodos aos
já herdados
Veículo
getVelocidade():float
getPassageiros():int
acelera()
freia()

Sobreposição



O que é polimorfismo?

Poli = muitos

Morfo = forma

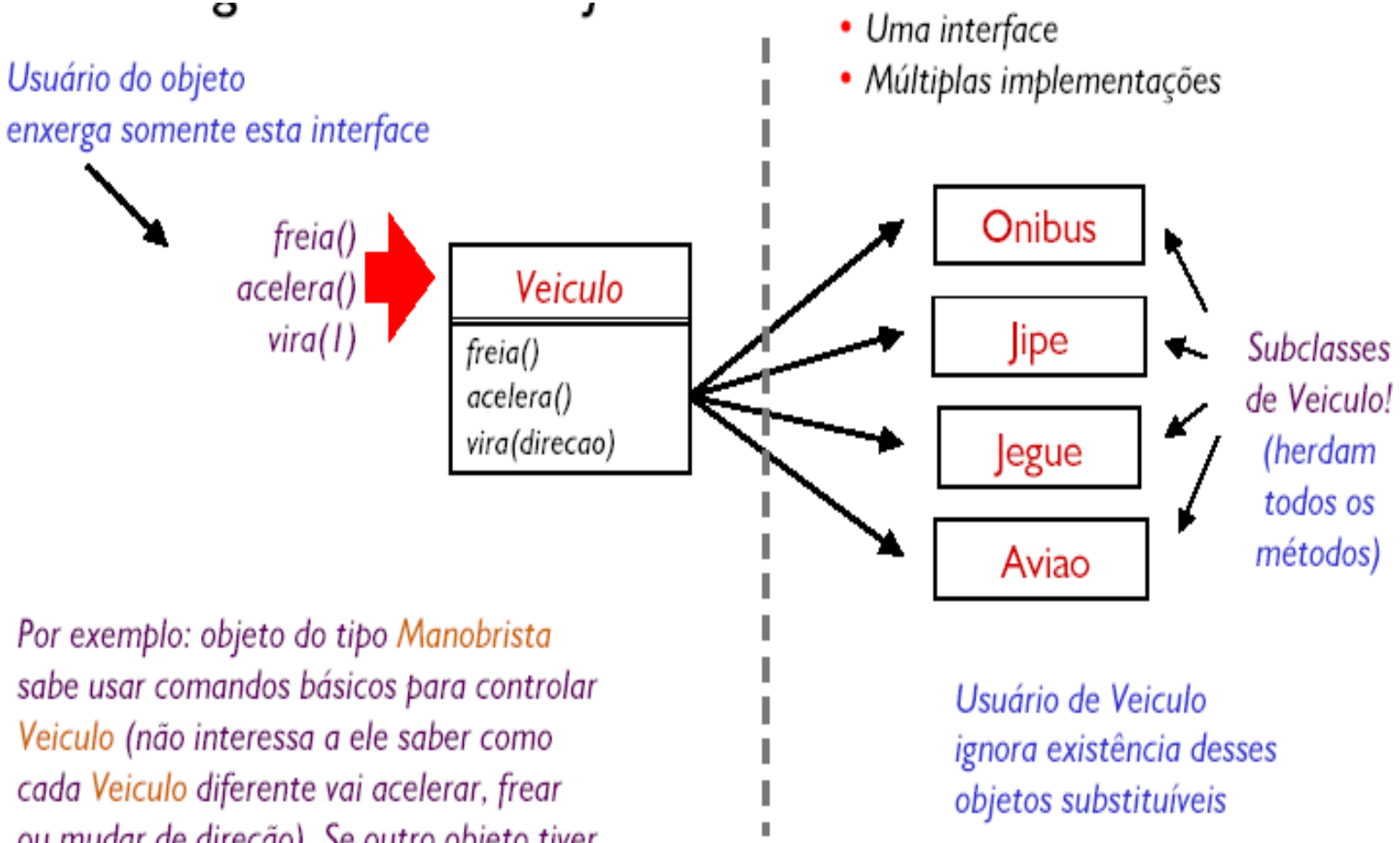
Como funciona?

- Um objeto que faz papel de interface serve de intermediário fixo entre o programa-cliente e os objetos que irão executar as mensagens recebidas
- O programa cliente não precisa saber a existência de outros objetos;
- Os objetos podem ser substituídos sem que os programas que usem a interface sejam afetados

Simplificando

Significa que um objeto pode ser utilizado no lugar de outro

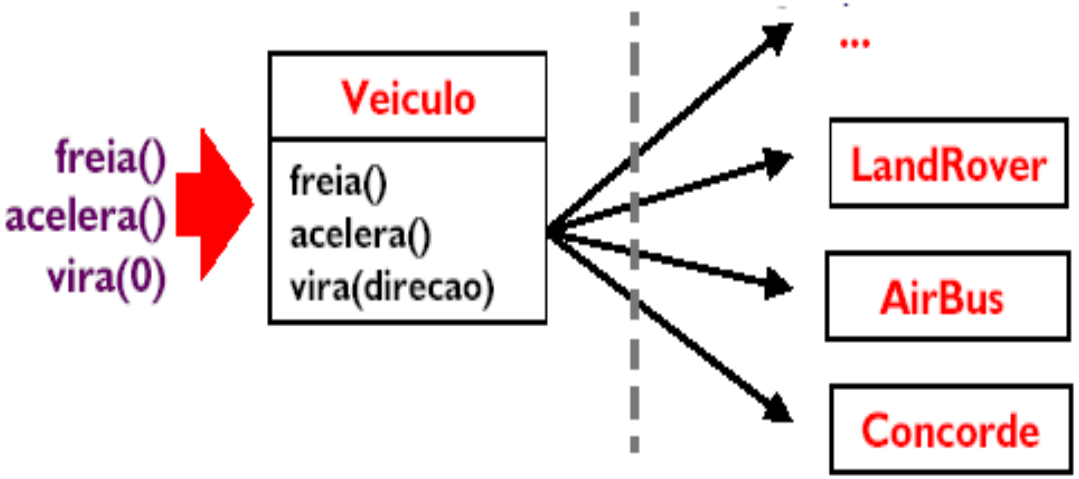
Exemplo

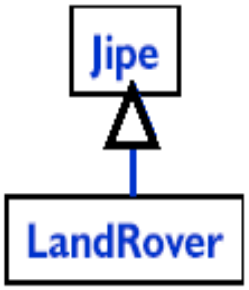

• Uma interface
• Múltiplas implementações
Usuário do objeto
enxerga somente esta interface
freia()
acelera()
vira(I)
Veiculo
freia()
acelera()
vira(direcao)
Onibus
Jipe
Jegue
Aviao
Subclasses
de Veiculo!
(herdam
todos os
métodos)
Por exemplo: objeto do tipo Manobrista
sabe usar comandos básicos para controlar
Veiculo (não interessa a ele saber como
cada Veiculo diferente vai acelerar, frear
ou mudar de direção). Se outro objeto tiver
Usuário de Veiculo
ignora existência desses
objetos substituíveis

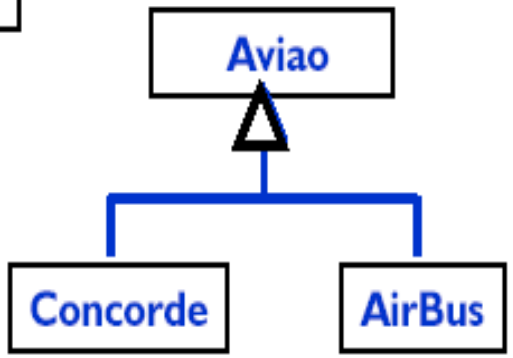
Programas extensíveis

Novos objetos podem ser usados em programas que não previam sua existência.

Exemplo







Mesmo nome. Implementações diferentes.

```
Veiculo v1 = new Veiculo();
Veiculo v2 = new Aviao();
Veiculo v3 = new Airbus();
v1.acelera(); // acelera Veiculo
v2.acelera(); // acelera Aviao
v3.acelera(); // acelera AirBus
```

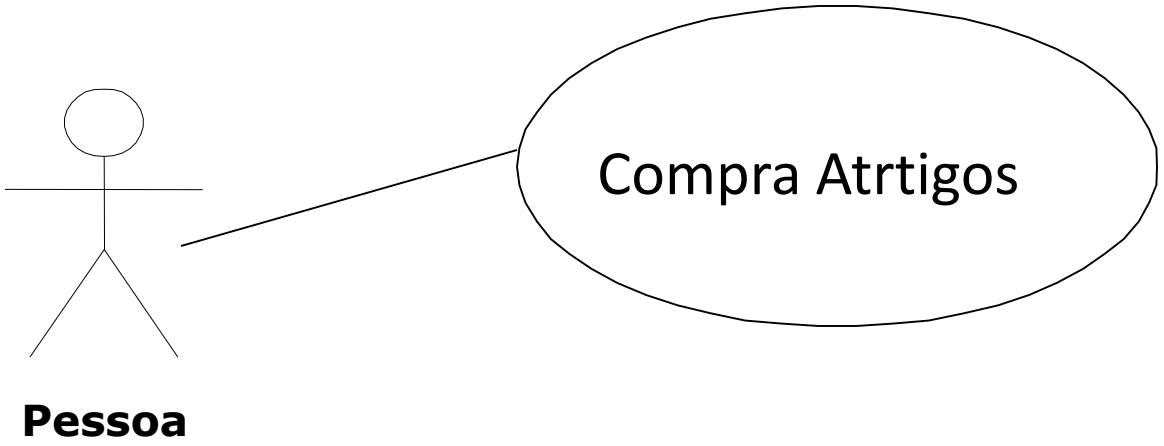
Características

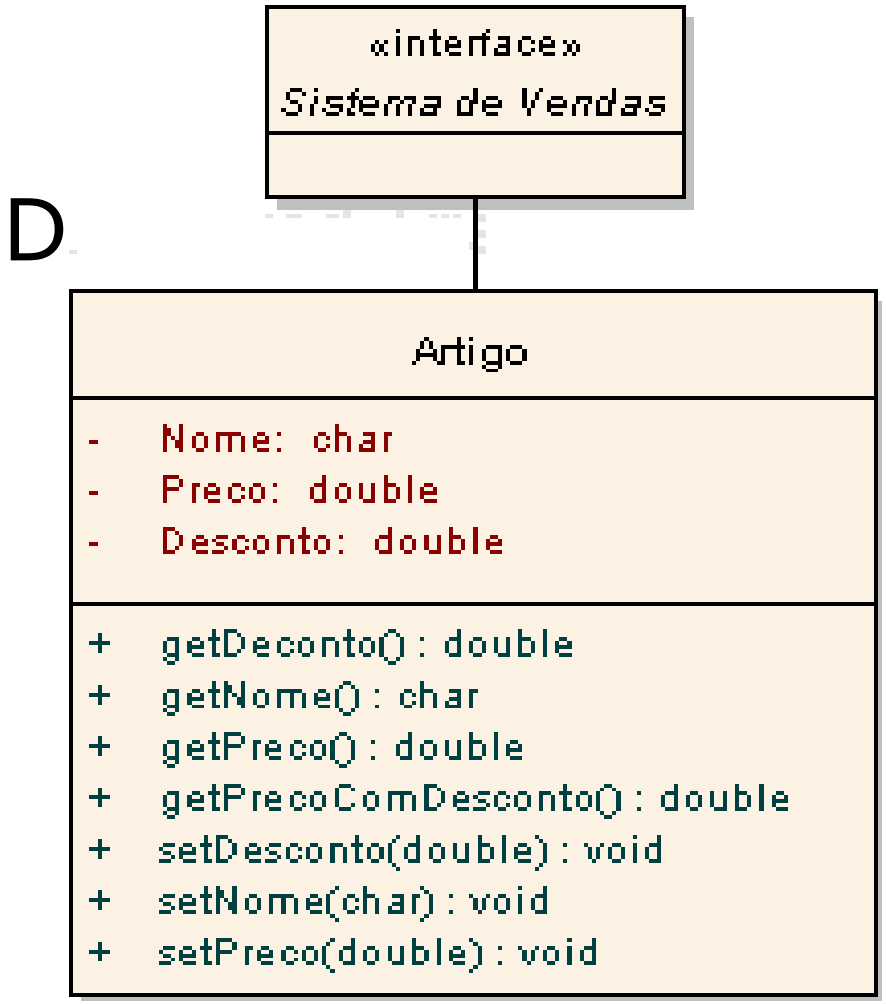
- Permite separar a interface da implementação;
- A classe base define a interface comum e não precisa dizer como isso vai ser feito;
- Diz apenas que métodos existem

Resolvendo estudo de caso

1. Uma pessoa comprou 3 artigos em uma loja. Para cada artigo ela tem o nome, preço e o percentual de desconto sobre o preço. O sistema deve gerar um relatório com: nome, preço, preço com desconto de cada artigo e o preço total a pagar.

Diagrama Use-case




«interface»
Sistema de Vendas
Artigo
- Nome: char
- Preco: double
- Desconto: double
+ getDeconto() : double
+ getNome() : char
+ getPreco() : double
+ getPrecoComDesconto() : double
+ setDesconto(double) : void
+ setNome(char) : void
+ setPreco(double) : void

Implementação em Java

```
package model;

public class Artigo {
private String nome;
private double preco;
private double percDesconto;

public void setNome(String nome){
this.nome=nome;
}
public void setPreco(double preco){
this.preco=preco;
}
```

Implementação em Java

```
public void setPercDesconto(double perc){
this.percDesconto=perc;
public String getNome(){
return this.nome;
}
public double getPreco(){
return this.preco;
}
public double getPercDesconto(){
return this.percDesconto;
}
public double getValorComDesconto(){
return getPreco()-(getPreco()*getPercDesconto()/100);
}
}
```

Implementação em Java

```
package view;

import javax.swing.JOptionPane;

import model.Artigo;

public class AppArtigo {

public static void main(String[] args) {

String nome = JOptionPane.showInputDialog("Informe o nome do Artigo");

double preco = Double.parseDouble(JOptionPane.showInputDialog("Informe o
Valor do ARtigo"));

double percDesconto =
Double.parseDouble(JOptionPane.showInputDialog("informe o valor do
desconto"));

Artigo novoArtigo = new Artigo();

novoArtigo.setNome(nome);

novoArtigo.setPreco(preco);

novoArtigo.setPercDesconto(percDesconto);

JOptionPane.showMessageDialog(null, "valor com desconto "

+novoArtigo.getValorComDesconto());}}
```

Resolvendo estudo de caso

2. Uma empresa oferece para seus clientes um determinado desconto de acordo com o valor da compra efetuada. O desconto é de 20% se o valor da compra for maior ou igual a R$ 200,00 e 15% se for menor. Supondo-se que a empresa tem 10 clientes, o sistema deverá fornecer o nome, endereço e quanto pagará cada cliente.

Resolvendo estudo de caso

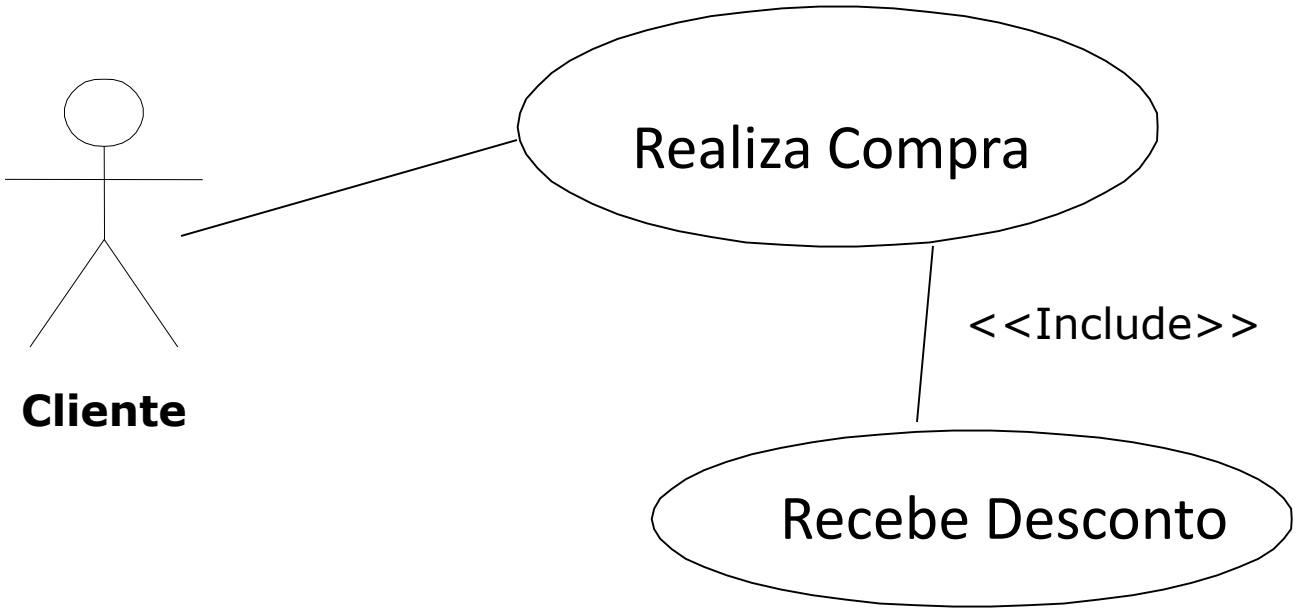
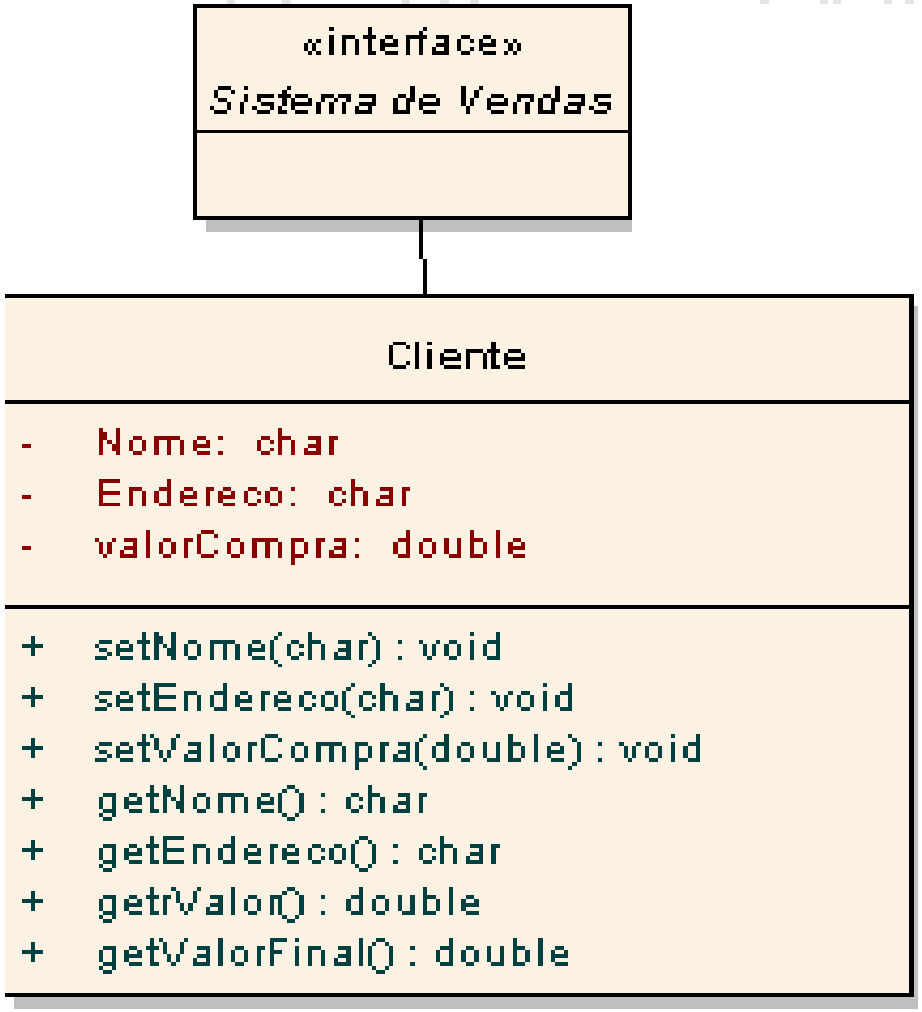


Diagrama de classe



Implementação em Java

```
package model;
public class Cliente {
private String nome;
private String endereco;
private double valorDaCompra;
public void setNome(String vnome){
this.nome=vnome;
}
public void gravaEndereco(String endereco){
this.endereco=endereco;
}
public void setValor(double valor){
this.valorDaCompra=valor;
}
```

Implementação em Java

```
public String getNome(){
return this.nome;
}
public String getEndereco(){
return this.endereco;
}
public double getValor(){
return this.valorDaCompra;
}
public double getValorFinal(){
if(getValor()>=200){
return getValor()-(getValor()*.2);
}
return getValor()-(getValor()*.15);
}}
```

Implementação em Java

```
package view;

import javax.swing.JOptionPane;

import model.Cliente;

public class AppArtigos {

public static void main(String[] args) {

Artigo novoArtigo = new Artigo();

novoArtigo.setNome(JOptionPane.showInputDialog("Informe o nome do
Cliente"));

novoArtigo.gravaEndereco(JOptionPane.showInputDialog("Informe o
endereco do cliente"));

novoArtigo.setValor(Double.parseDouble(JOptionPane.showInputDialog("i
nforme o valor da compra")));

JOptionPane.showMessageDialog(null, "O Cliente
"+novoArtigo.getNome()+" mora no endereco
"+novoArtigo.getEndereco()+" e pagará a quantia de
"+novoArtigo.getValorFinal());

}

}
```

Resolvendo estudo de caso

3.Considere que cada aluno de uma determinada disciplina tenha realizado três provas. Para cada aluno tem-se o nome e as notas das três provas. O sistema deverá fornecer os seguintes relatórios:

a) o nome dos alunos aprovados;

b) a média da turma em cada uma das provas;

c) o número de alunos aprovados e reprovados;

d) o nome do melhor aluno da turma;

e) o nome do melhor aluno em cada prova.

Obs.: Mínimo de alunos na turma: 10.

Diagrama Use-case

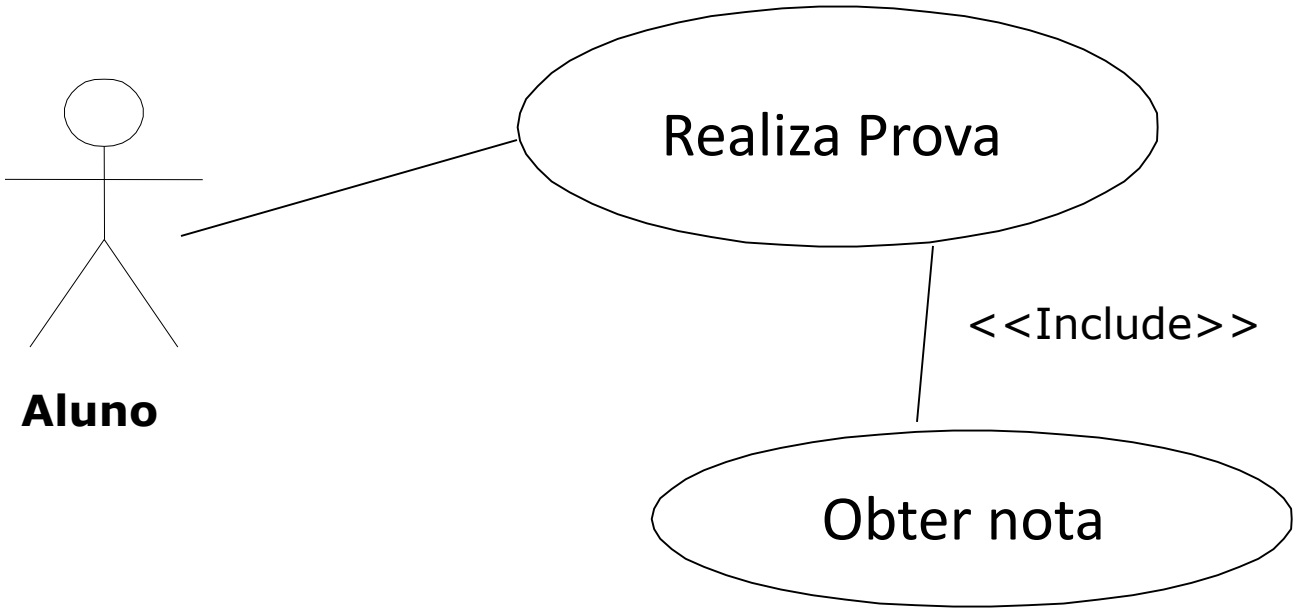


Diagrama Use-case

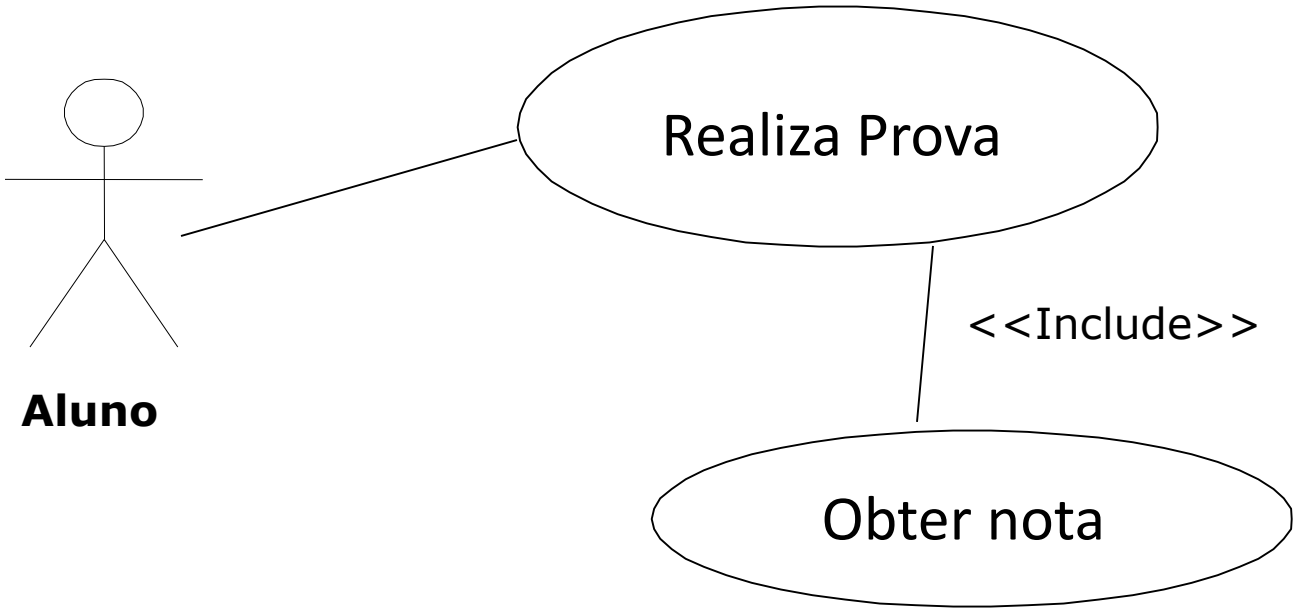
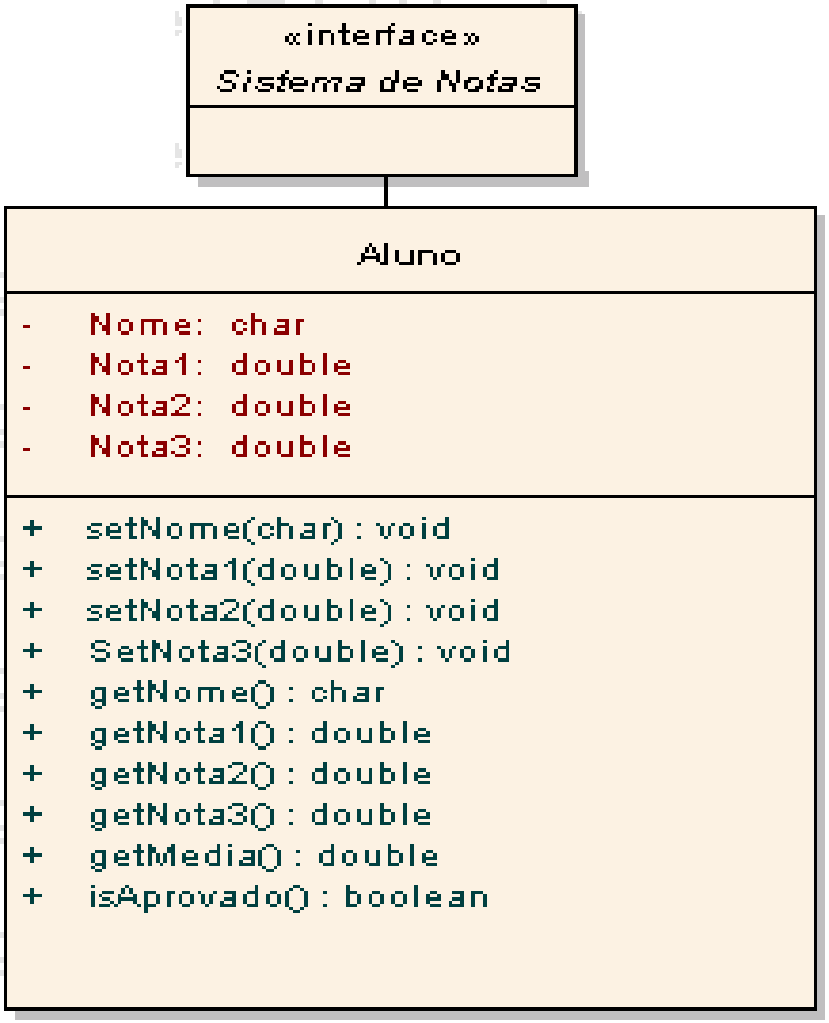


Diagrama de classe



Implementação em Java

```
package model;
public class Aluno {
private String nome;
private float nota1;
private float nota2;
private float nota3;

public String getNome() {
return nome;
}
public void setNome(String nome) {
this.nome = nome;
}
```

Implementação em Java

```
public float getNota1() {
return nota1;
}
public void setNota1(float nota1) {
this.nota1 = nota1;
}
public void setNota3(float nota3) {
this.nota1 = nota3;
}
public float getNota3() {
return nota3;
}
public float getNota2() {
return nota2;
}
```

Implementação em Java

```
public float getNota2() {
return nota2;
}
public void setNota2(float nota2) {
this.nota2 = nota2;
}
public float getMedia() {
return (this.getNota1() + this.getNota2() + this.getNota3()) / 3;
}
public boolean isAprovado() {
return this.getMedia() >= 7;
}
public String toString(){//parta mostrar os nome na mensagem da lista
return nome;
}}
```

Implementação em Java

```
package view;
import java.util.ArrayList;
import javax.swing.JList;
import javax.swing.JOptionPane;
import model.Aluno;

public class App {
public static void main(String[] args) {

ArrayList<Aluno> lista = new ArrayList<Aluno>();
String nome = JOptionPane.showInputDialog("Nome");
```

Implementação em Java

```
while (nome != null) {
float nota1 = Float
.parseFloat(JOptionPane.showInputDialog("Nota1"));
float nota2 = Float
.parseFloat(JOptionPane.showInputDialog("Nota2"));
float nota3 = Float
.parseFloat(JOptionPane.showInputDialog("Nota3"));
Aluno a = new Aluno();
a.setNome(nome);
a.setNota1(nota1);
a.setNota2(nota2);
a.setNota3(nota3);
lista.add(a); // adicionar na lista
```

Implementação em Java

```
JOptionPane.showMessageDialog(null, "nome: " + a.getNome()
+ "\n Média: " + a.getMedia());
nome = JOptionPane.showInputDialog("Nome");
}
for (int i = 0; i < lista.size(); i++) {
JOptionPane.showMessageDialog(null, "Nome: "
+ lista.get(i).getNome() + "\n Média: "
+ lista.get(i).getMedia());
}
for (Aluno aluno : lista){//para percorrer uma lista
JOptionPane.showMessageDialog(null, "Nome: "
+ aluno.getNome() + "\n Média: "
+ aluno.getMedia());
}
```

Implementação em Java

```
//uma lista dentro de uma mensagem
JList jList = new JList(lista.toArray());
JOptionPane.showMessageDialog(null,jList);
}
}
```

Resolvendo estudo de caso

4. Uma empresa de promoções culturais recebe diferentes tipos de obras para que sejam lançadas no mercado. Antes de realizar um lançamento, cada obra é avaliada por até três pessoas especializadas (pareceristas). Para controlar as obras submetidas à avaliação, está sendo desenvolvido um sistema.

Título	Autor	Pareceres
A escalada do terror	Fernandinho	Parecerista Data Conteúdo João de Abreu 10/10/2001 Livro que não pode ser publicado, pois é um manual do crime.
Confins do Sertão	Hector Babenco	Parecerista Data Conteúdo Lima Duarte 15/08/2002 Vídeo que demonstra a sensibilidade da alma nordestina. Vale a pena assistir.

Diagrama Use-case

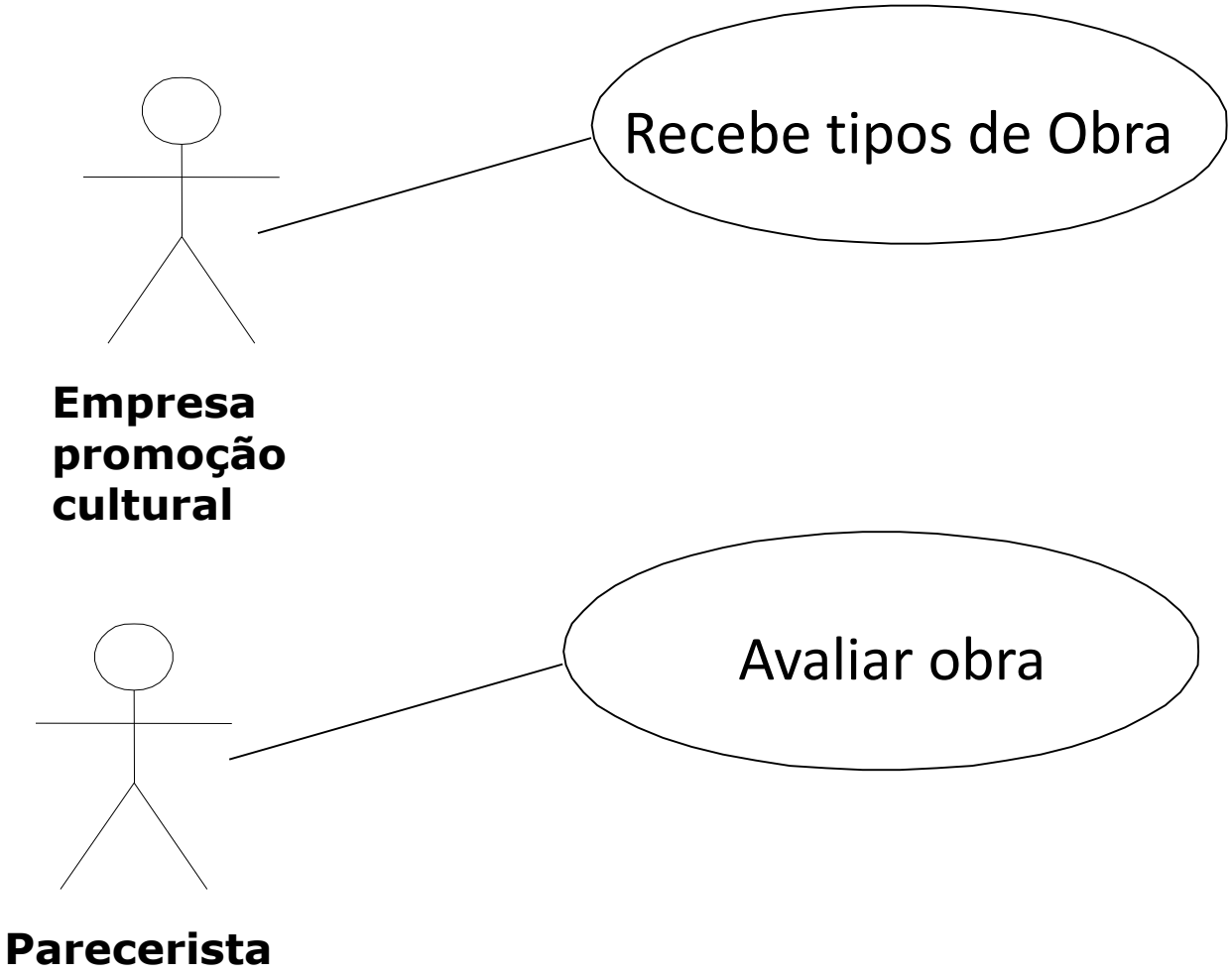
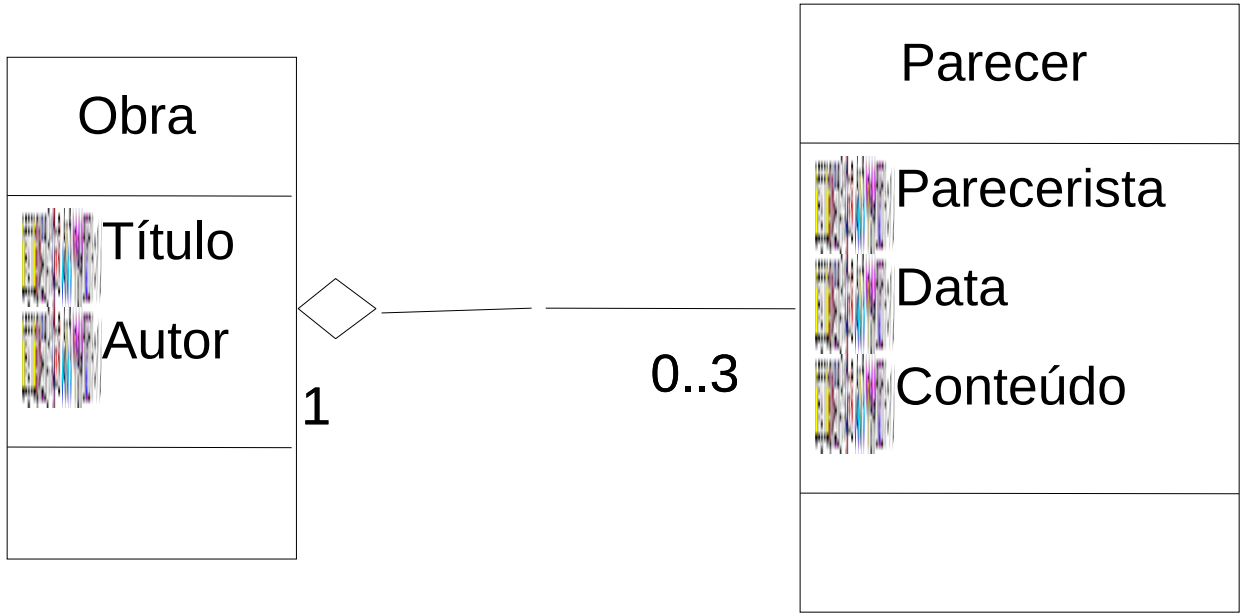


Diagrama de classe

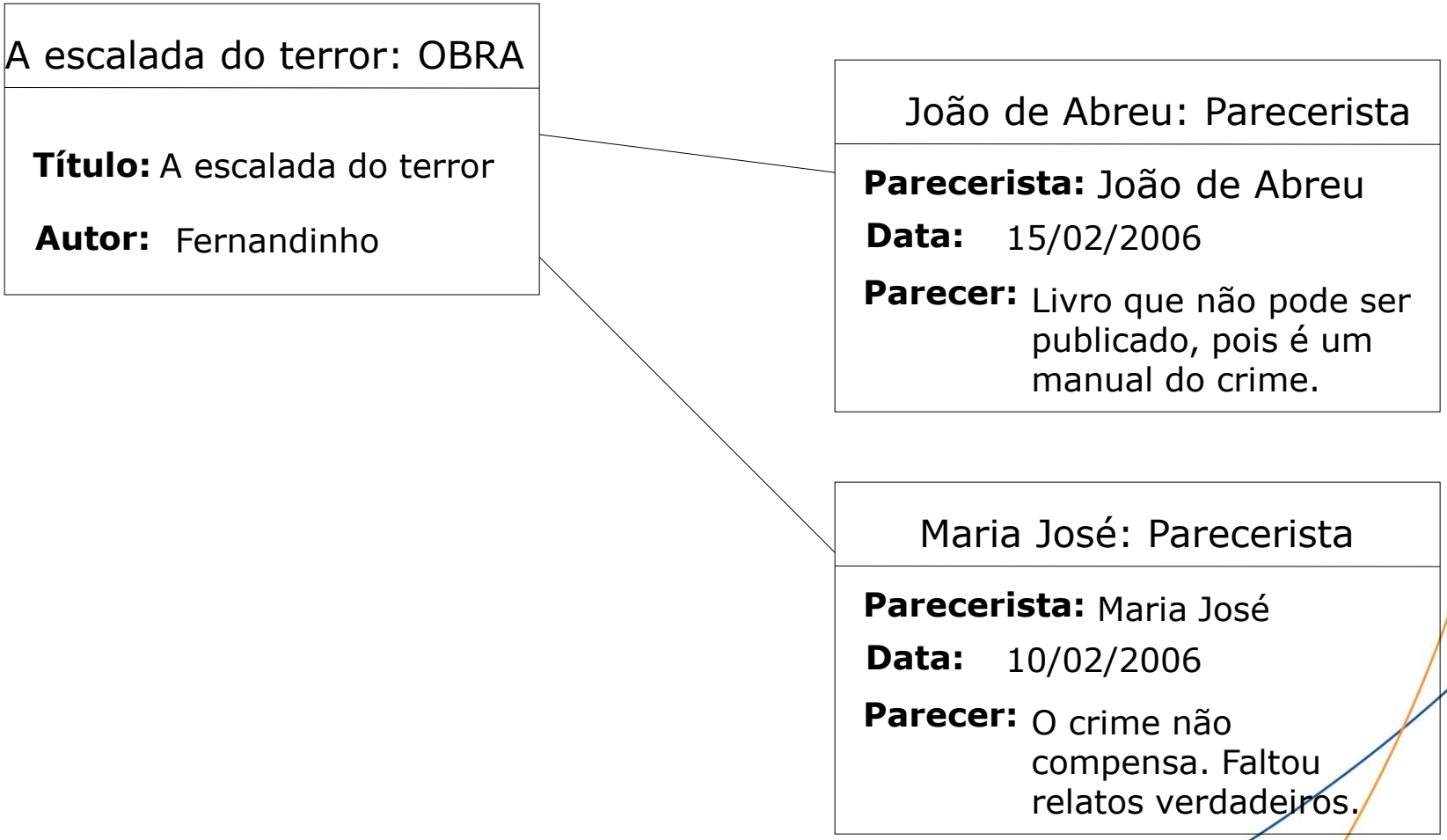


A classe Parecer é parte da classe Obra (agregação).

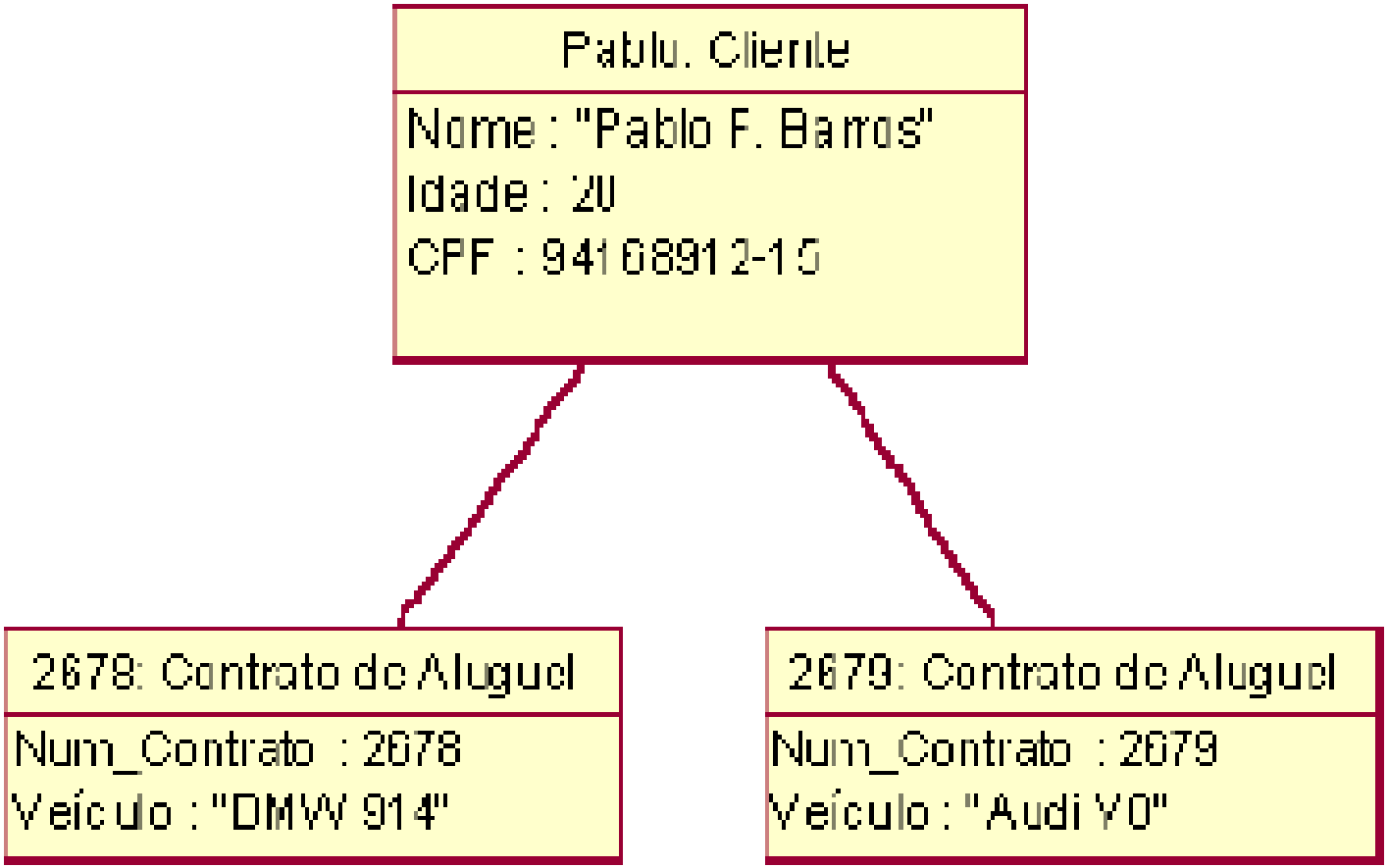
Diagrama de Objeto

- O diagrama de objetos é uma variação do diagrama de classes e utiliza quase a mesma notação.
- A diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes.
- O diagrama de objetos é como se fosse o perfil do sistema em um certo momento de sua execução.

Diagrama de Objeto



Exemplo Diagrama de Objeto



Herança

- Determinada classe passa a herdar características (variáveis e métodos) definidas em outra classe, especificada como sua ancestral ou superclasse.
- Possibilita o compartilhamento ou reaproveitamento de recursos definidos anteriormente em outra classe.
- A classe fornecedora dos recursos recebe o nome de superclasse e a receptora dos recursos, de subclasse.
- Especialização, uma vez que uma classe herda características de outra, ela pode implementar partes específicas não contempladas na classe original (superclasse).

Exemplo de Herança

```
Public class Pessoa {
    private String nome;

    public void setNome(String nome){
        this.nome = nome;
    }
    public String getNome(){
        return this.nome;
    }
}
```

A classe Pessoa possui um atributo nome e dois métodos para manipular seu conteúdo.

Exemplo de Herança

```
Public class PessoaFisica extends Pessoa
{
    private String rg;

    public void setRg(String rg){
        this.rg = rg;
    }
    public String getRg(){
        return this.rg;
    }
}
```

```
Public class PessoaJuridica extends
Pessoa {
    private String cnpj;

    public void setCnpj(String cnpj){
        this.cnpj = cnpj;
    }
    public String getCnpj(){
        return this.cnpj;
    }
}
```

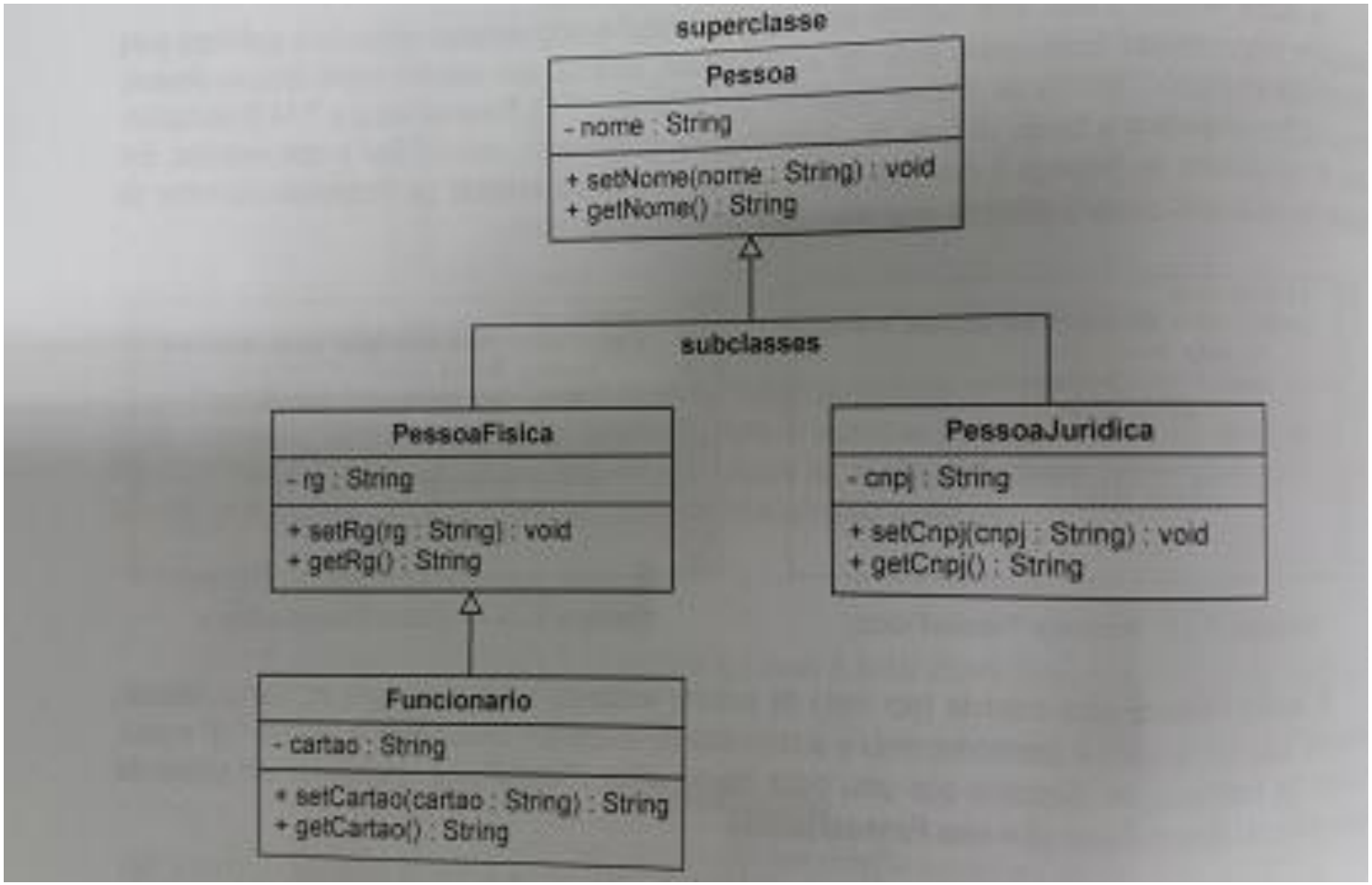
Por meio da palavra **extends** as classes **PessoaFisica** e **PessoaJuridica** estendem as funcionalidades da classe Pessoa.

Exemplo de Herança

```
Public class Funcionario extends PessoaFisica
{
    private String cartao;

    public void setCartao(String cartao){
        this.cartao = cartao;
    }
    public String getCartao(){
        return this.cartao;
    }
}
```

A classe Funcionario estende as funcionalidades da classe **PessoaFisica** especializando ainda mais a classe original Pessoa.

Exemplo de Herança



Exemplo de Herança

```
Public class UsarFuncionario  {
   public static void main (String args[]) {

     Funcionario funcionario = new Funcionario()
     funcionario.setNome(“Pedro”);
     funcionário,setRg(“1.238.874”);
     funcionário.setCartao(“RH123”);
     system.out.println(funcionário.getNome());
     system.out.println(funcionário.getRg());
     system.out.println(funcionário.getCartao());
   }
}
```

Polimorfismo

```
Public class PessoaPolimorfa  {
   public static void main (String args[]) {

     Pessoa pessoa = null;
     int tipo = Integer.parseint(
     JOptionPane.showimputDialog(
     "Forneça um número de 1 a 4"));
     switch (tipo) {
       case 1: pessoa = new Pessoa(); break;
       case 2: pessoa = new PessoaFisica(); break;
       case 3: pessoa = new PessoaJuridica(); break;
       case 4: pessoa = new Funcionario(); break;
       default: sytem.out.println("tipo não existe");
     }
   }
}
```