Algoritmo II
Classes e Objetos

Prof. Me. Rober Marcone Rosi
Unidade de Engenharia, Computação e Sistemas

Revisão – Nomenclatura SUN

- São recomendações da SUN para nomenclatura de classes, métodos e variáveis. Seu programa irá funcionar mesmo que você não siga estas convenções:
  I. Classes e interfaces: A primeira letra deve ser maiúscula e, caso o nome seja formado por mais de uma palavra, as demais palavras devem ter sua primeira letra maiúscula também (camelCase);
  II. Métodos: A primeira letra deve ser minúscula e após devemos aplicar o camelCase;
  III. Variáveis: Da mesma forma que métodos;
  IV. Constantes: Todas as letras do nome devem ser maiúsculas e caso seja formada por mais de uma palavra separada por underscore.

Classes	Métodos	Variáveis	Constantes
Carro	desligar	motor	COMBUSTIVEL
CursoJavaIniciante	iniciarModulo	quantidadeModulos	NOME_CURSO
Hotel	reservarSuiteMaster	nomeReservaSuite	TAXA_SERVICO

Orientação a Objetos

- O termo **orientação a objetos** indica que estamos organizando nosso sistema como uma coleção de objetos, que incorporam dados e operações e interagem para realizar as ações do sistema.
- A orientação a objetos objetiva a construção de um sistema com código mais fácil de manter, flexível e extensível.

Histórico de Orientação a Objetos

- A OO surgiu no final da década de 60, quando dois cientistas dinamarqueses criaram a linguagem Simula (Simulation Language)
- 1967 - Linguagem de Programação Simula-67- conceitos de classe e herança
- O termo Programação Orientada a Objetos (POO) é introduzido com a linguagem Smalltalk (1980)
- Inicio dos anos 90⇨ Paradigma de Orientação a Objetos
- abordagem poderosa e prática para o desenvolvimento de software

Vantagens da Orientação a Objetos

- Abstração de dados: os detalhes referentes às representações das classes serão visíveis apenas a seus atributos;
- Reutilização: o encapsulamento dos métodos e representação dos dados para a construção de classes facilitam o desenvolvimento de software reutilizável, auxiliando na produtividade de sistemas;
- Flexibilidade: as classes delimitam-se em unidades naturais para a alocação de tarefas de desenvolvimento de software;

Vantagens da Orientação a Objetos

- Extensibilidade: facilidade de estender o software devido a duas razões:
  - herança: novas classes são construídas a partir das que já existem;
  - as classes formam uma estrutura fracamente acoplada o que facilita alterações;
- Mesma notação é utilizada desde a fase de análise até a implementação.
- Manutenibilidade: a modularização natural em classes facilita a realização de alterações no software.

Exemplos de Linguagens Orientada a Objetos

FAESA
faça acontecer

- Existem diversas linguagens OO, tais como:
  - Smalltalk (1980)
  - Ada (1983)
  - Eiffel (~1985)
  - Object Pascal (1986)
  - Common Lisp (1986)
  - C++ (~1989)
  - Java

Conceitos Básicos de Orientação a Objetos

- Criou o conceito de **objeto**, que é um tipo de dado com uma estrutura e operações para manipular esta estrutura.
- Tipos definidos pelo usuário devem se comportar da mesma maneira de tipos pré-definidos (fornecidos pelo compilador).
- Os objetos trocam mensagens entre si.
- Essas mensagens resultam na ativação de métodos, os quais realizam as ações necessárias.
- Os objetos que compartilham uma mesma interface, ou seja, respondem as mesmas mensagens, são agrupados em classes.

Pilares da Orientação a Objetos

- **Abstração,**
- **Encapsulamento,**
- **Herança e**
- **Polimorfismo**.

ABSTRAÇÃO

- processo de representar um grupo de entidades através de seus atributos comuns
- feita a abstração, cada entidade particular (instância) do grupo é considerada somente pelos seus atributos particulares
- os atributos comuns às entidades do grupo são desconsiderados (ficam "ocultos" ou "abstraídos")

ABSTRAÇÃO

- A capacidade de determinar o problema de maneira geral, dando importância apenas aos seus aspectos relevantes e ignorando os detalhes é chamada de abstração.

  Existem dois tipos de abstração: **Abstração de dados e Abstração de Procedimentos**.

ABSTRAÇÃO

Abstração de dados

Abstração de dados é a definição de um tipo de dado por seu comportamento e estado, utilizando-se métodos. A manipulação desse dado é realizada apenas através de seus métodos.

Um exemplo é classificar uma lista a partir das operações aplicadas a ela, como inserção e remoção. Qualquer objeto do tipo lista só pode sofrer modificações através dessas operações.

ABSTRAÇÃO

Abstração de procedimentos

Consiste em considerar um procedimento com uma operação bem definida como algo único, mesmo que utilize outros procedimentos internos. Usamos abstração de procedimentos quando uma função divide-se em outras sub-funções, que por sua vez decompõem-se em outras funções.

ABSTRAÇÃO DE PROCESSOS

- o conceito de abstração de processos é um dos mais antigos no projeto de linguagens
- absolutamente crucial para a programação
- historicamente anterior à abstração de dados
- todos os subprogramas são abstrações de processo
- exemplo: chamadas **sort(array1, len1), sort(array2, len2), ...**

ENCAPSULAMENTO

- um agrupamento de subprogramas+dados que é compilado separada/independentemente chama-se uma **unidade de compilação** ou um **encapsulamento**
- um encapsulamento é portanto um sistema abstraído
- muitas vezes os encapsulamentos são colocados em bibliotecas
- exemplo: encapsulamentos C (não são seguros porque não há verificação de tipos de dados em diferentes arquivos de encapsulamento)

HERANÇA

- Herança é a criação de novas classes, chamadas classes derivadas, a partir de classes já existentes, as classes-base.
- Alguns dos benefícios da utilização de herança é a reutilização do código da classe base, além da possibilidade de implementação de classes abstratas genéricas.

POLIMORFISMO

- Polimorfismo caracteriza a criação de funções com um mesmo nome, mas códigos diferentes, facilitando a extensão de sistemas.
- Um tipo de polimorfismo é a redefinição de métodos para uma classe derivada. Para que isso aconteça, o método deve possuir o mesmo nome, tipo de retorno e argumentos do método sobrescrito.

Estrutura Básica

- Quatro elementos básicos:
  - Classes/Objetos;
  - Atributos;
  - Métodos;
  - **Pacotes**.

Classes e Objetos

- De acordo com o Dicionário Aurélio, **objeto** é algo que é manipulável, manufaturável, perceptível por um dos sentidos ou apreendido pelo conhecimento.
- Isto pode representar praticamente qualquer coisa, desde **itens concretos**, como um indivíduo, um carro, um barco, um avião, um cachorro ou uma árvore, até **itens abstratos**, como uma reserva, uma organização, um aluguel, uma festa ou um evento.

Exemplo de Classe e Objetos

- Objetos e Classes

Conceitos de Classe

- Classes:
  - É um tipo definido pelo usuário que contém o molde, a especificação para os objetos, assim como o tipo inteiro contém o molde para as variáveis declaradas como inteiros.
  - A classe envolve, associa, funções e dados, controlando o acesso a estes, definí-la implica em especificar os seus atributos (dados) e suas funções membro (código).
  - Todo objeto é uma instância de uma Classe.
  - Possuem propriedades (ATRIBUTOS) e comportamento (MÉTODOS).

Conceitos de Classe

- Apesar de serem diferentes, elas compartilham uma mesma estrutura e um mesmo comportamento. Se estas cadeiras forem objetos no nosso sistema, não há necessidade de modelar e representar separadamente cada cadeira.

Conceitos de Classe

- **Classe**: Basta definir, uma única vez, um modelo que descreva a estrutura e o comportamento destes objetos. A este modelo damos o nome de **classe.**

Conceitos de Classe

- No mundo real, agrupamos os objetos já existentes de acordo com a nossa necessidade, classificando-os. Por exemplo, em uma sala de aula podemos agrupar os alunos por sexo, criando as classes Homem e Mulher; por idade, criando as classes Adolescente, Adulto e Idoso; por entendimento do assunto, criando as classes Iniciante e Experiente; ou simplesmente agrupando todos na classe Aluno.

Conceitos de Classe

- Desta forma, podemos modelar o sistema com diferentes visões, definindo diferentes **classes conceituais** para um mesmo objeto do mundo real. As classes conceituais agrupam e modelam os objetos do mundo real.
- Após identificar os objetos do mundo real e modelá-los, devemos definir como estes objetos serão implementados. Normalmente, cada classe conceitual é transformada em uma **classe de software**.

Conceitos de Classe

- No exemplo abaixo, a **classe conceitual Aluno** é transformada na **classe de software de mesmo nome**.
- Repare na figura abaixo, que a diferença entre as duas é que na classe de software há uma preocupação com questões de implementação, como tipo de cada atributo, visibilidade, definição das operações, etc.

Objeto no mundo real	Classe Conceitual	Classe de Software	Objeto de Software
	**Aluno** mat nome	**Aluno** - mat: int - nome: String + matricular() + trancarMat()	**obj: Aluno** - mat = 1234 - nome = "José" + matricular() + trancarMat()

Conceitos de Classe

- Exemplos de Classes:
  - aluno
  - conta corrente
  - folha de cheque
  - automóvel
  - cliente
  - fornecedor
  - time de futebol
  - jogo de futebol

Conceito de Objetos

- Objetos:
  - Tudo em Orientação a Objetos é OBJETO.
  - Objeto, no mundo físico, é tipicamente um produtor e consumidor de itens de informação.
  - Definição (mundo do software)
    - “Qualquer coisa, real ou abstrata, a respeito da qual armazenamos dados e métodos que os manipulam” Martin, Odell (1995)
  - Abstração de uma entidade do mundo real e que possuí características (VALORES).
  - É uma instanciação de uma classe.

Conceito de Objetos

- Exemplos de objetos:
  - um aluno (“Carlos Alberto da Silva”)
  - uma conta corrente (“0123-003934-1”)
  - uma folha de cheque (“234564”)
  - um automóvel (“Honda Civic preto 06/06”)
  - um cliente (“Maria Antonia Guimaraes”)
  - um fornecedor (“Bosch”)
  - um time de futebol (“Palmeiras”)
  - uma partida de futebol (“Palmeiras x Santos”)

Conceito de Atributos

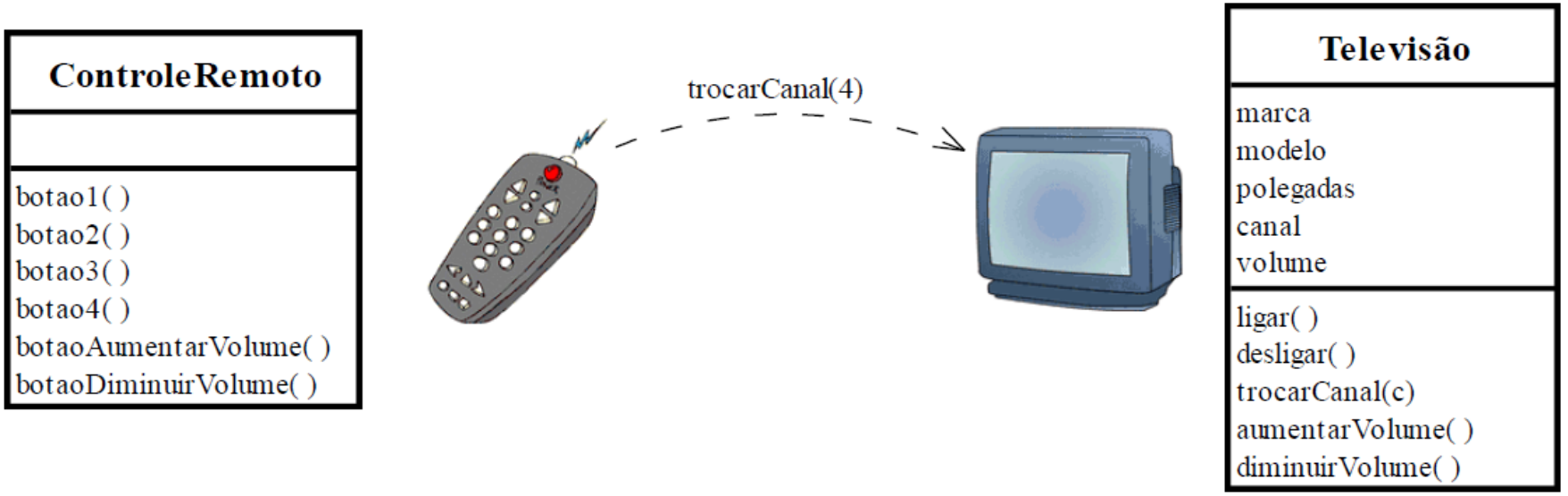
- Atributos:
  - Representam um conjunto de informações, ou seja, elementos de dados que caracterizam um objeto.
  - Descrevem as informações que ficam escondidas em um objeto para serem exclusivamente manipulado pelas operações daquele objeto.
  - São variáveis que definem o estado de um objeto, ou seja, são entidades que caracterizam os objetos.
  - Cada objeto possui seu próprio conjunto de atributos.

Conceitos Básicos

- Métodos:
  - Quando um objeto é mapeado dentro do domínio do software, os processos que podem mudar a sua estrutura de dados são denominados Operações ou Métodos.
  - Métodos são invocados por Mensagens.
  - Cada objeto possui seu próprio conjunto de métodos
  - Definições: São procedimentos ou funções definidos e declarados que atuam sobre um objeto.

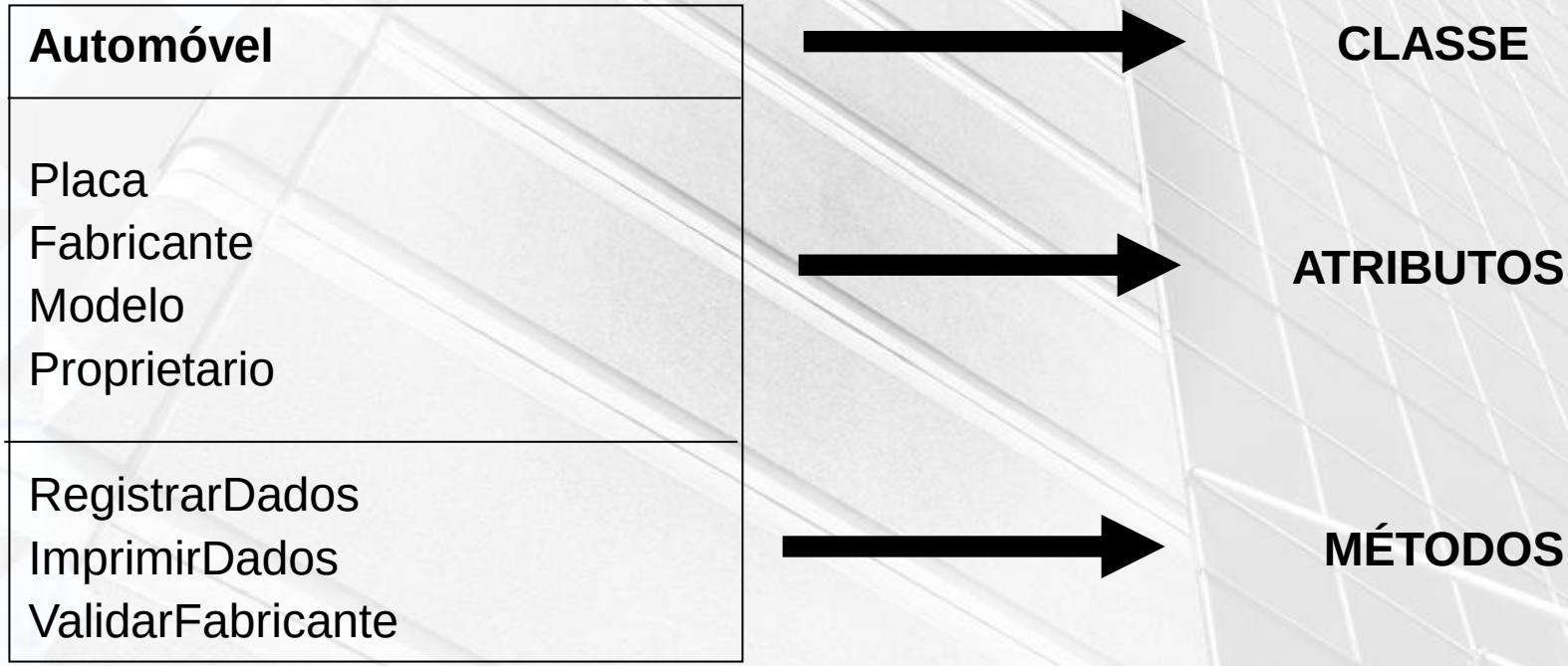
Conceitos Básicos

- Mensagem: São o meio de comunicação entre objetos e são responsáveis pela ativação de todo e qualquer processamento.



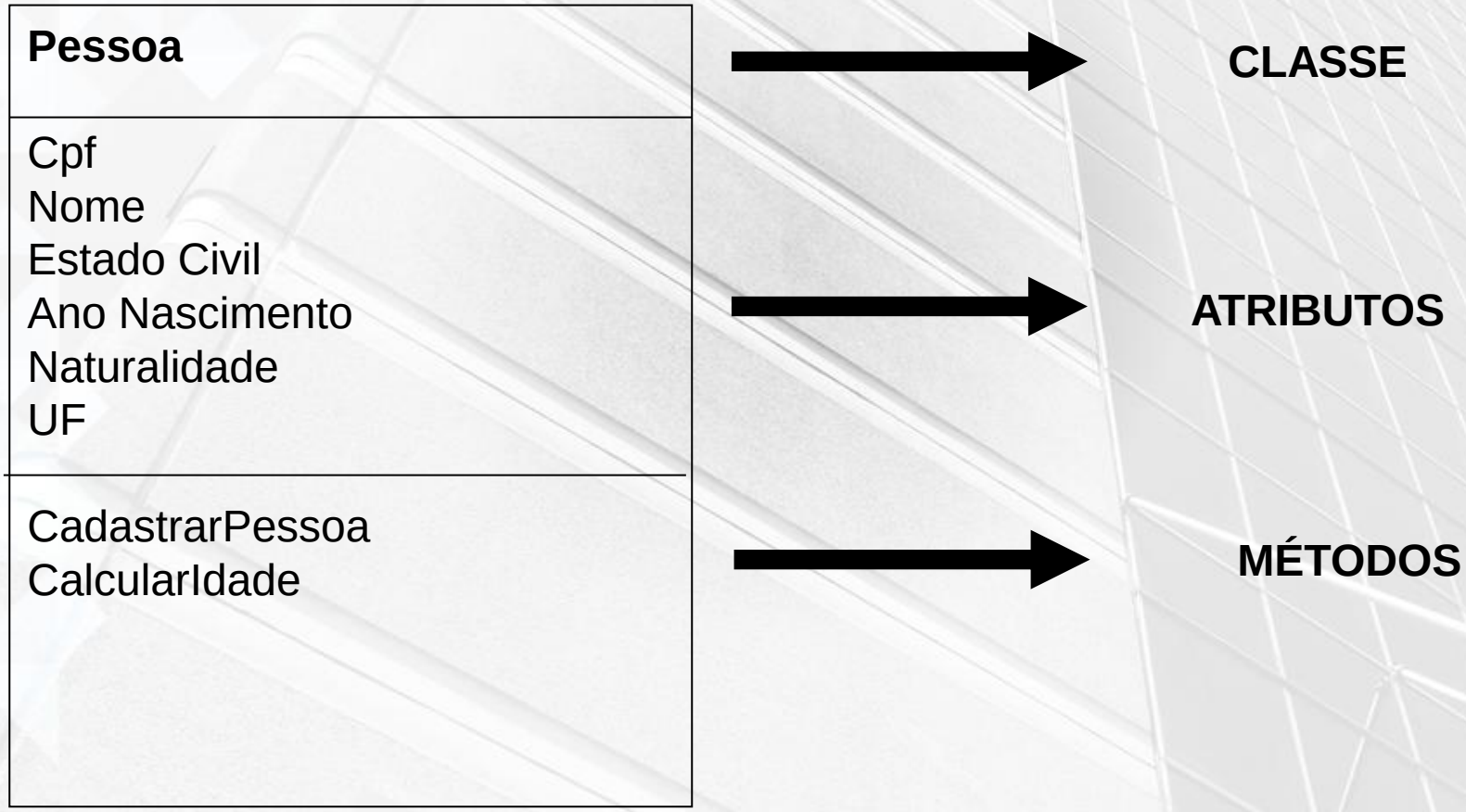
Notação de Classe em UML

- Atributos e Métodos:



Notação de Classe em UML

- Atributos e Métodos:



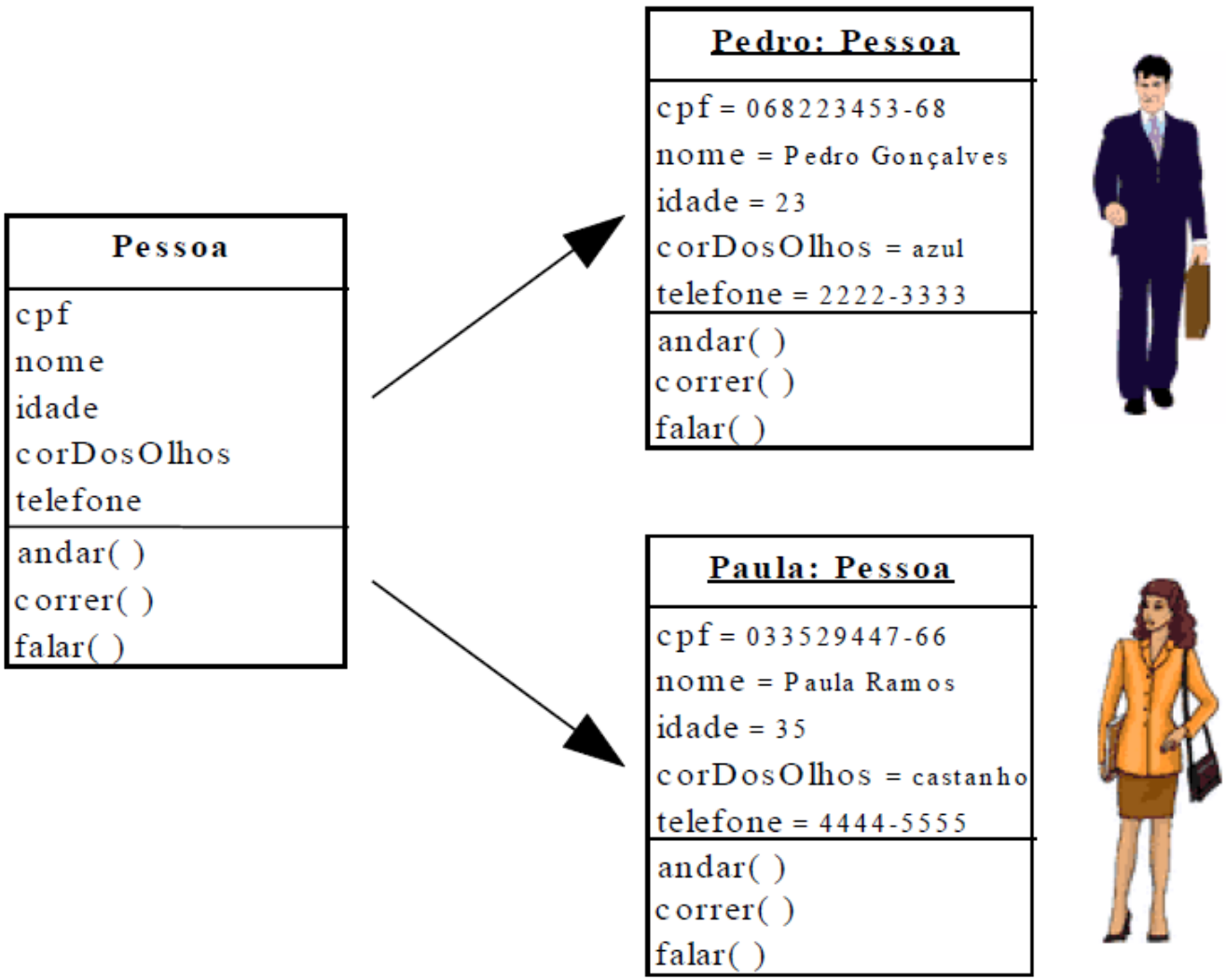
Instanciação

- A classe de software define a estrutura e o comportamento dos objetos. Por exemplo, a classe Pessoa, exemplificada na figura abaixo, define as informações e o comportamento de uma pessoa.
- Na mesma figura podemos notar dois objetos criados a partir desta classe. A classe funciona como uma espécie de molde para os objetos.

Instanciação

- **Instanciar** é o ato de criar um objeto a partir de uma classe de software. Ao instanciar um objeto, o computador aloca um espaço na memória da máquina para armazenar as características do objeto.
- Para fazer uma comparação, imprimir um formulário corresponde a instanciar um objeto, cujos valores podem ser preenchidos sem alterar o formulário inicial, que neste caso corresponde à classe. Uma classe pode instanciar vários objetos, ou eventualmente, nenhum.
- **Instância** é um sinônimo para objeto, que é uma instância de uma classe.

Instanciação

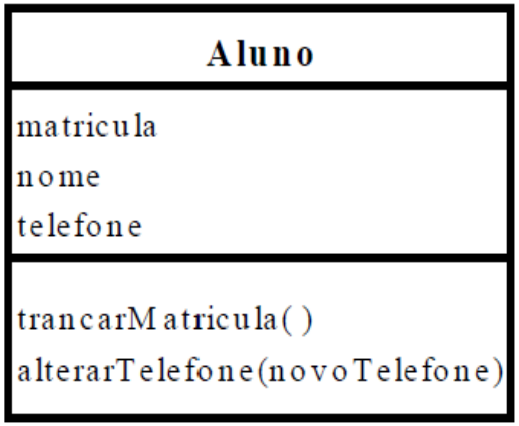


Exercício

- É possível ter objetos originados da mesma classe com uma quantidade de atributos diferentes? Por quê?
- É possível ter objetos originados da mesma classe com valores diferentes nos atributos?
- É possível ter objetos da mesma classe com implementações diferentes para uma mesma operação?
- Um objeto pode pertencer a mais de uma classe?
- Uma classe pode ter mais de um objeto?
- Se dois analistas identificarem a classe Aluno para dois sistemas distintos, obrigatoriamente a classe terá os mesmos atributos e operações nos dois sistemas? Por quê?

Implementação OO Java

Classe, Atributos e Métodos



atributos

métodos

```
class Aluno {
    String matricula;
    String nome;
    String telefone;

    public void trancarMatricula() {
        ...
    }

    public void alterarTelefone (String novoTelefone) {
        telefone = novoTelefone;
    }
}
```

Implementação OO Java

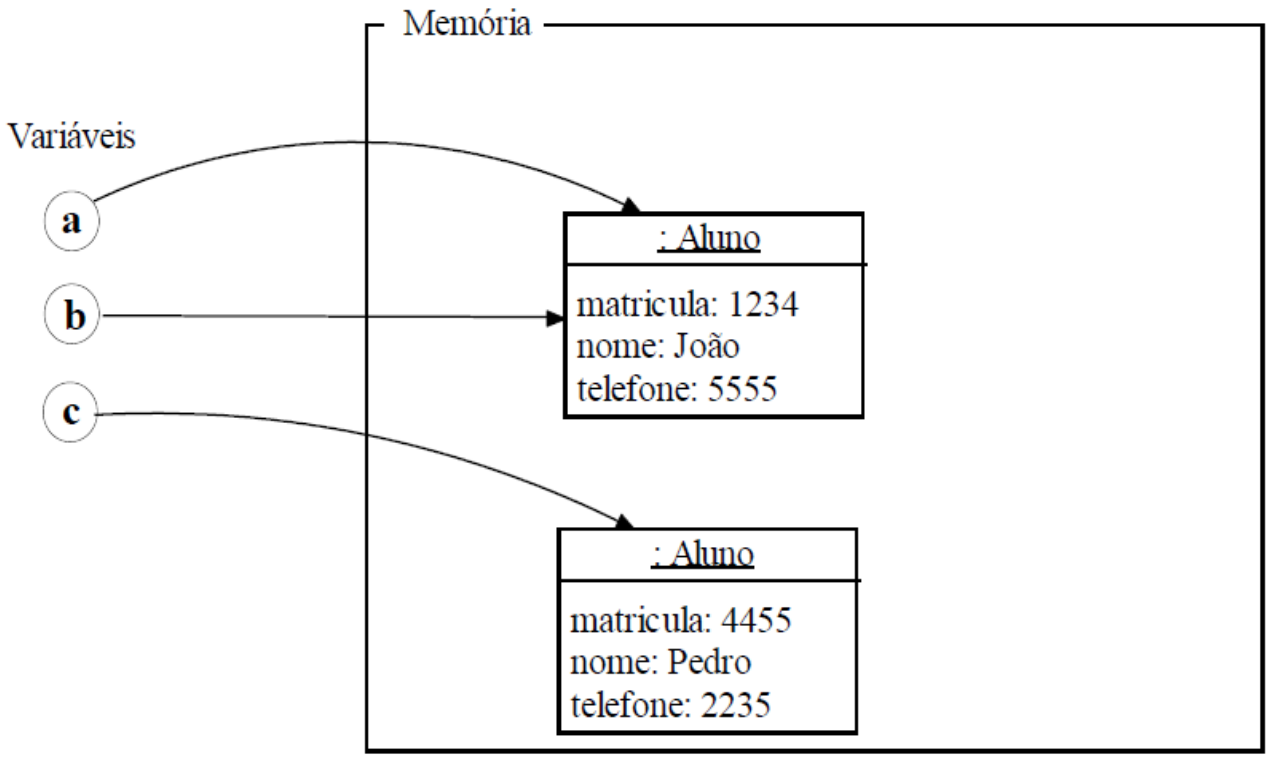
Classe, Atributos e Métodos

```
class Programa {
    public void main () {

        Aluno a;
        a = new Aluno() // cria um novo objeto
        print(a.nome);  // imprime o valor do atributo nome
        a.alteraTelefone("2222-3333"); // chama o método alteraTelefone, passando como parâmetro o novo tel.

    }
}
```

Implementação OO Java

Instanciação

```
Aluno a, b, c;
a = new Aluno();
a.matricula = 1234;
a.nome = "João";
a.telefone = "5555";

b = a;

c = new Aluno();
c.matricula = 4455;
c.nome = "Pedro";
c.telefone = "2235";
```



Implementação OO Java

Exercício:

O que será impresso?

```
static void Main(string[] args)
  {
      Aluno a, b, c;
      a = new Aluno();
      b = new Aluno();
      c = a;
      a.Telefone = "1111-2222";
      b.Telefone = a.Telefone;
      c.Telefone = "3323-4444";



      System.out.println(a.Telefone);
      System.out.println(b.Telefone);
      System.out.println(c.Telefone);
  }
```

Implementação OO Java

FAESA faça acontecer

Identidade dos objetos:

- Objetos se distinguem por sua própria existência e não pelos valores de seus atributos

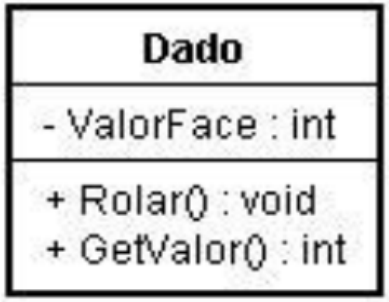
```
Vector a = new Vector();
a.add("Teste");
Vector b = new Vector();
b.add("Teste");

if (a == b) {
    print("ok 1");
}

if (a.equals(b)) {
    print("ok 2");
}
```

Implementação OO Java

FAESA
Faça acontecer

Exercício:

- Implemente a classe Dado conforme especificado abaixo:



- O método Rolar() deve atribuir ao atributo ValorFace um número aleatório de 1 a 6.
- O método GetValor() retorna o valor do atributo ValorFace
- Faça um programa que role dois dados e mostre o valor de cada um e a soma dos valores.

Implementação OO Java

Solução 1:

```
public class Dado {

    private int ValorFace;

    public void rolar() {
        int r = (int)(Math.random() * 6); // 0 - 5
        r = r + 1; // 1 - 6

        ValorFace = r;
    }

    public int getValor() {
        return ValorFace;
    }

}
```

Implementação OO Java

FAESA
Faça acontecer

Solução 2:

```
import java.util.Random;
public class Dado {
    private int valorFace;
    public void rolar()
    {
        Random gerador = new Random();

        valorFace = gerador.nextInt(6)+1;
    }
    public int getValor()
    {
        return valorFace;
    }
}
```

Implementação OO Java

FAESA faça acontecer

Solução:

```
public class Diversos {
    /**
     * @param args
     */
    public static void main(String[] args) {

        Dado d1 = new Dado();
        Dado d2 = new Dado();

        d1.rolar();
        d2.rolar();

        System.out.println("D1: " + d1.getValor());
        System.out.println("D2: " + d2.getValor());

        System.out.println("Soma: " + (d1.getValor() + d2.getValor()));
    }
}
```

Construtor

- O método construtor é invocado quando se cria um novo objeto para a classe.
- Em Java, o método construtor é aquele que tem o mesmo nome da classe e sem tipo de retorno.
- Pode haver mais de um construtor, desde que com tipos de argumentos diferentes
- Etapas executadas durante a instanciação:
  - 1.Alocação da área de memória.
  - 2.Inicialização dos atributos com os valores default.
  - 3.Execução do método construtor.

Construtor

```
class Quadro {
    int altura;
    int largura;
    Figura figura;

    public Quadro() {
        figura = new Figura();
        altura = 0;
        largura = 0;
    }
}
```

```
class Figura {
    public Figura () {
        println("Uma nova figura acaba de ser criada");
    }
}
```

Construtor: O que será impresso?

FAESA
Faça acontecer

```
class ClasseA {

    public ClasseA () {
        print("Oi!");
    }

    public ClassA (String str) {
        print("Oi "+str);
    }
}

class ClasseB {
    public ClasseB () {
        new ClasseA ("BB");
    }
}

class ProgramaPrincipal {
    public start() {
        ClasseA a = new ClasseA();
        ClasseB b = new ClasseB();
    }
}
```

Considere a classe a seguir:

```
public class Person
{
  private String name;
  public Person(String a)
  {
    name = a; //copia a para name
  }
  public String getName()
  {
     return name;
  }
  public String toString()
  {
    return "Nome: " + name;
  }
}
```

Encapsulamento

Person x = **new** Person("Maria"),
y = **new** Person("Andre");

System.*out.println("Nome: "+x.name);*

Estrutura Básica

Pacotes

- Classes que compõem o núcleo de funcionalidades Java.

- Package Java é um mecanismo para agrupar classes de finalidades afins ou de uma mesma aplicação.

Estrutura Básica

Pacotes

- Além de facilitar a organização conceitual das classes, permite localizar cada classe necessária durante a execução da aplicação.

- A principal funcionalidade de um pacote Java é evitar a explosão do espaço de nome.

Estrutura Básica

Pacotes

- Entre os principais pacotes oferecidos estão:
  - java.lang;
  - java.util;
  - java.io;
  - java.net.

Estrutura Básica

- Package é uma coleção de classes e interfaces relacionadas fornecendo proteção de acesso e gerenciamento de nomes de diretórios.
- Usamos a declaração ***import*** para acessar essas classes.

```
import java.lang.*;
```

Estrutura Básica

Pacotes

- Para colocarmos uma classe em um pacote, deve-se inserir a diretiva ***package*** com o nome do pacote no início da classe.

```
package br.com.cap.pdf.profiler;
```

Especificadores de acesso

- Determinam a visibilidade de um determinado membro da classe com relação a outras classes;
- Há quatro níveis de acesso:
  - Público (public);
  - Privado (private);
  - Protegido (protected);
  - Amigo ou privado ao pacote (friendly ou package-private).

Especificadores de acesso

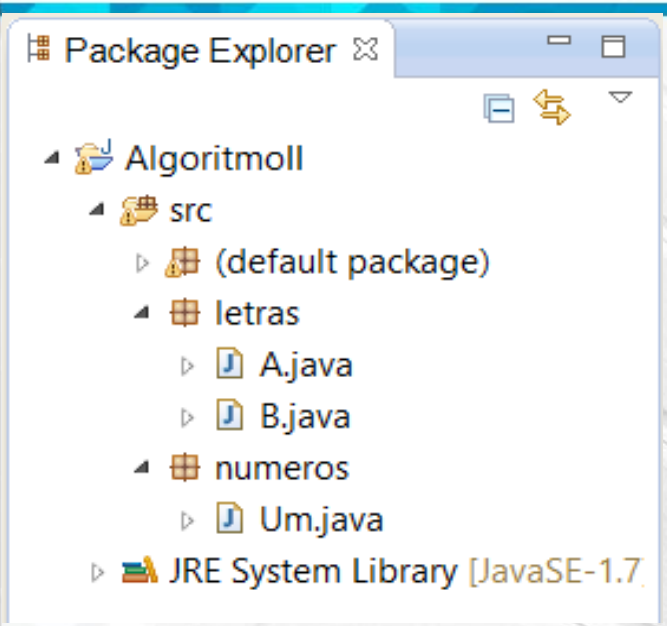
- Três palavras-chave especificam o acesso:
  - *public*
  - *private*
  - *protected*
- O nível de acesso **friendly** ou **package-private** é determinado pela ausência de especificador. Portanto o modificador de acesso nesse caso é padrão (default ou friendly)!
- O **modificador padrão** define que a classe só poderá ser acessada por outra classes dentro do mesmo pacote.
- Devem ser usadas antes do nome do membro que querem especificar;
- Não podem ser usadas em conjunto.

Membros públicos (public)

- **Classes**: Classes públicas* podem ser importadas por qualquer classe.
- **Atributos**: Atributos públicos podem ser lidos e alterados por qualquer classe.
- **Métodos**: Métodos públicos podem ser chamados por qualquer classe.

* Só pode haver uma classe pública por arquivo fonte e os nomes (da classe e do arquivo) devem ser iguais.

Membros públicos (public)




```java
package letras;

public class A {
    public int x = 10;
    public void print()
    {
        System.out.println(x);
    }
    void incr()
    {
        x++;
    }

}
```

```java
package letras;

public class B {
    public A a = new A();
    public void f()
    {
        a.x = 15;
        a.print();
    }

}
```

```java
package numeros;
import letras.B;

public class Um {
    B b = new B();
    public void g()
    {
//      b.a.incr();
        b.f();
    }

}
```

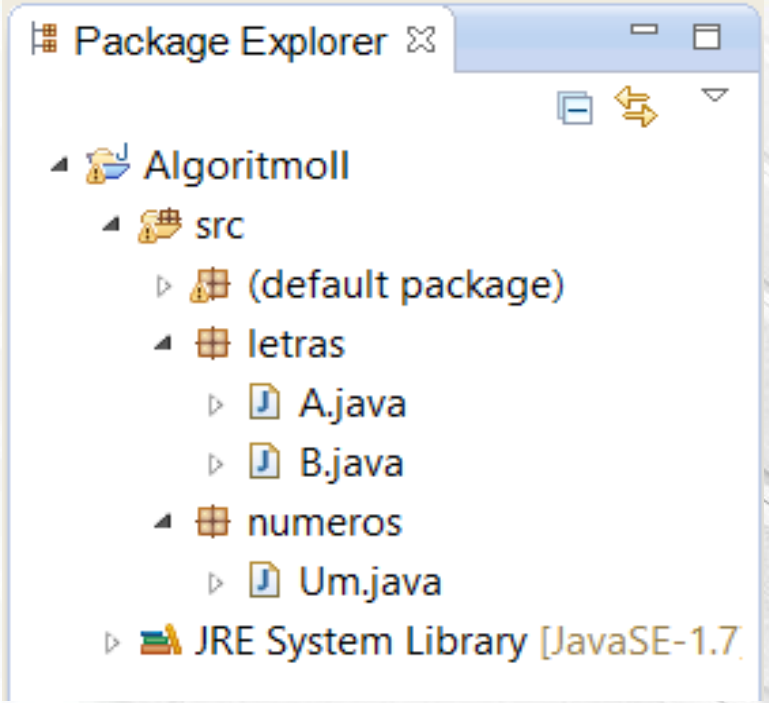
public static void main!

- O método main() é:
  - *public*, pois deve ser chamado pela JVM;
  - *static*, pois pertence à classe como um todo (a JVM não instancia um objeto para chamá-lo);
  - *void*, pois não retorna nada.
- A classe que possui o método main() deve ser:
  - *public*, pois deve ser acessível pela JVM.

Membros privados (private)

- **Classes**: Somente classes internas* podem ser declaradas privadas.
- **Atributos**: Atributos privados só podem ser lidos e alterados pela própria classe.
- **Métodos**: Métodos privados só podem ser chamados pela própria classe.
- * Tópico avançado. Não veremos em algoritmo II.

Membros privados (private)

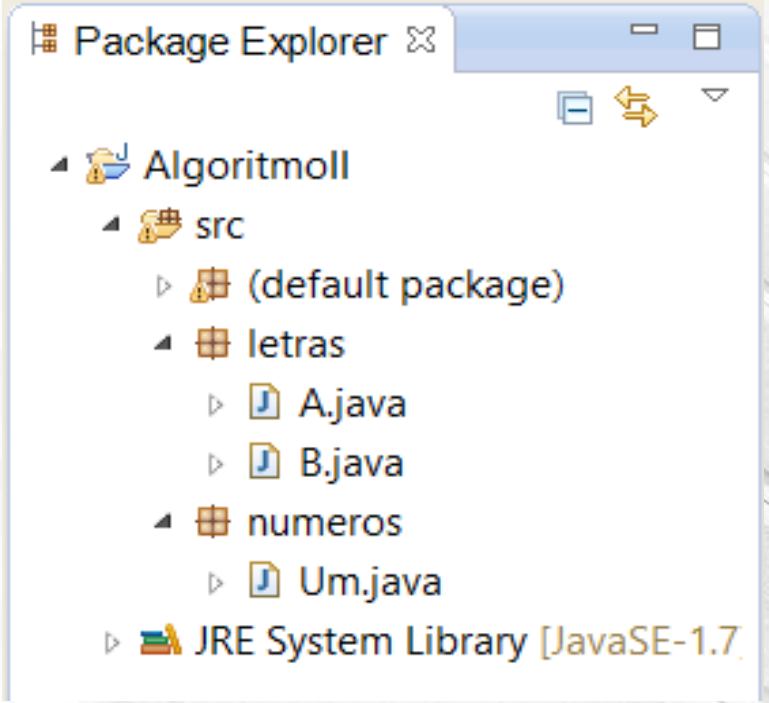



```
package letras;

public class A {
    private int x = 10;
    private void print()
    {
        System.out.println(x);
    }
    void incr()
    {
        x++;
    }

}
```

Membros amigos (friendly)

- **Classes**: Classes amigas só podem ser importadas por classes do mesmo pacote.
- **Atributos**: Atributos amigos só podem ser lidos e alterados por classes do mesmo pacote.
- **Métodos**: Métodos amigos só podem ser chamados por classes do mesmo pacote.

Membros amigos (friendly)




```java
package letras;

public class A {
    int x = 10;
    void print()
    {
        System.out.println(x);
    }
    void incr()
    {
        x++;
    }

}
```

Membros protegidos (protected)

- **Classes**: Somente classes internas* podem ser declaradas protegidas.
- **Atributos**: Atributos protegidos só podem ser lidos e alterados por classes do mesmo pacote ou subclasses*.
- **Métodos**: Métodos protegidos só podem ser chamados por classes do mesmo pacote ou subclasses*.
- * Tópico avançado. Não veremos em algoritmo II.

OBS: Os modificadores de acesso: (padrão, public, private e protected) nunca poderão ser combinados !

Resumo dos especificadores de acesso

Acesso	Público	Protegido	Amigo	Privado
Mesma classe	SIM	SIM	SIM	SIM
Classes no mesmo pacote	SIM	SIM	SIM	Não
Subclasses em diferentes pacotes	SIM	SIM	Não	Não
Não subclasses em diferentes pacotes	SIM	Não	Não	Não

Herança

- Herança é a criação de novas classes, chamadas classes derivadas, a partir de classes já existentes, as classes base.
- Alguns dos benefícios da utilização de herança é a reutilização do código da classe base, além da possibilidade de implementação de classes abstratas genéricas.

```
public class Student extends Person
{
  private String periodo; // período cursado
  private int classes; // aulas que frequenta
  ...

  public Student(String n, String g)
  {
    super(n); //chamada ao construtor da superclasse
    periodo = g;
    classes = 0;
  }

  public String toString()
  {
    return super.toString() + "\nPeríodo: " + periodo;
  }

} //fim da classe Student
```

Herança

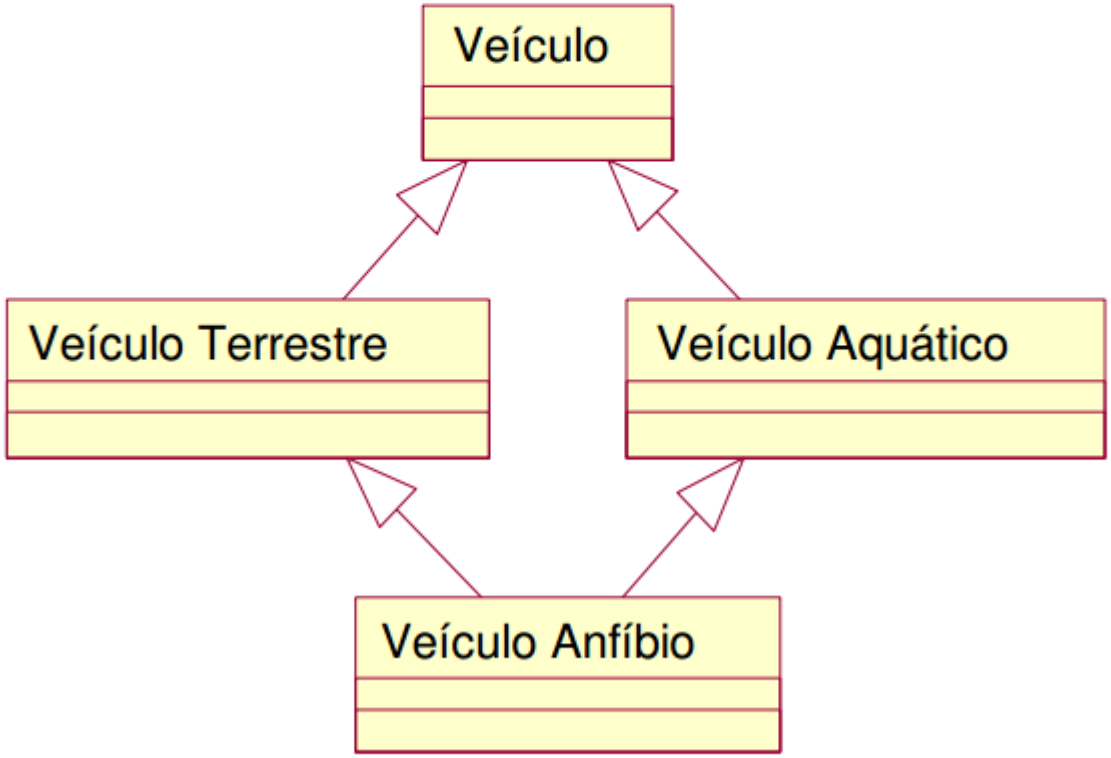
- A classe no slide anterior ilustra a criação de uma subclasse Student que herda os atributos e métodos de Person.
- Uma subclasse não herda os construtores de sua superclasse, mas pode invocá-los através do comando super.
- A chamada a essa função deve ser a primeira instrução no construtor da subclasse, caso contrário ocorrerá erro.
- Caso a chamada ao construtor base não tivesse sido feita explicitamente em Student(String n, String g), uma chamada ao construtor default (sem argumentos) da classe base seria realizada. Se não houver este construtor (o default), uma mensagem de erro será apresentada.

Herança Múltipla

- ***Herança múltipla:*** Uma classe pode ter mais de uma superclasse.
  - Tal classe herda de todas a suas superclasses.
- O uso de herança múltipla deve ser evitado.
  - Esse tipo de herança é difícil de entender.
  - Requer políticas para resolver conflitos de herança
  - Algumas LPs não dão suporte à implementação desse tipo de herança (Java e Smalltalk).

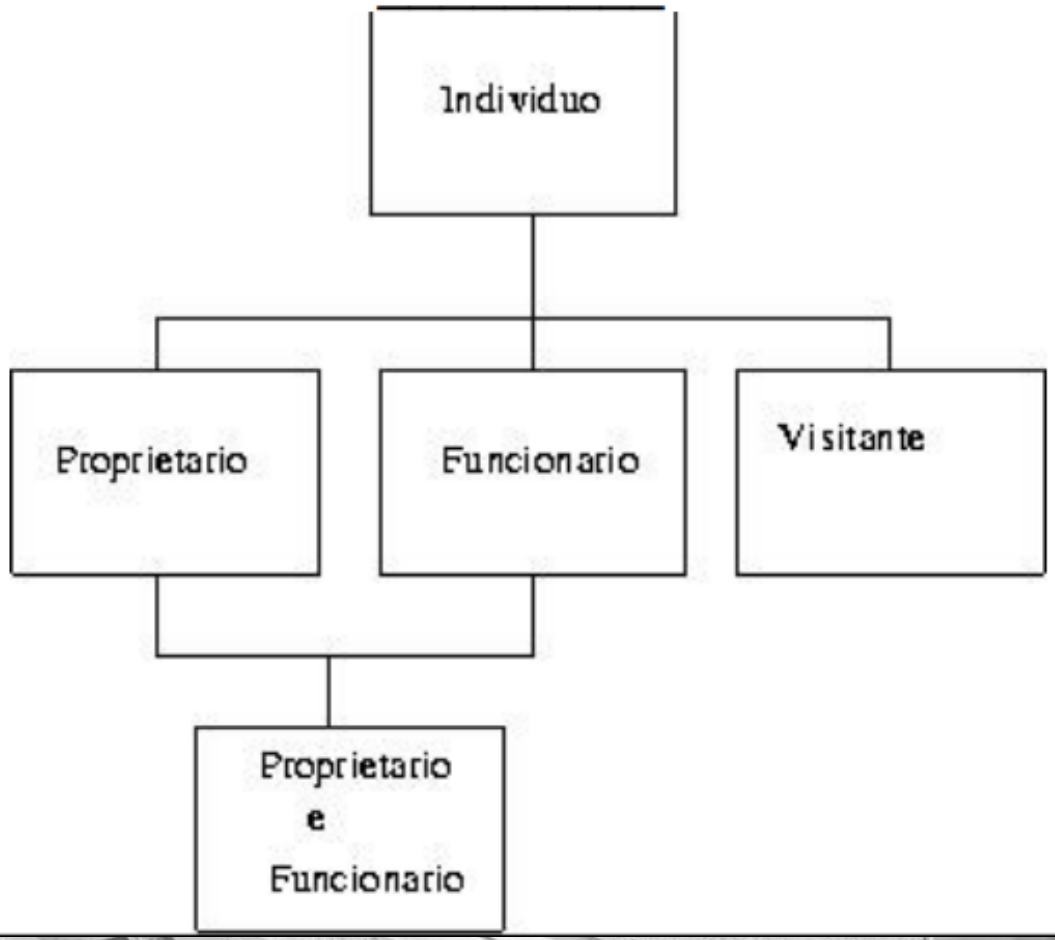
Herança Múltipla

Exemplo (Herança múltipla)

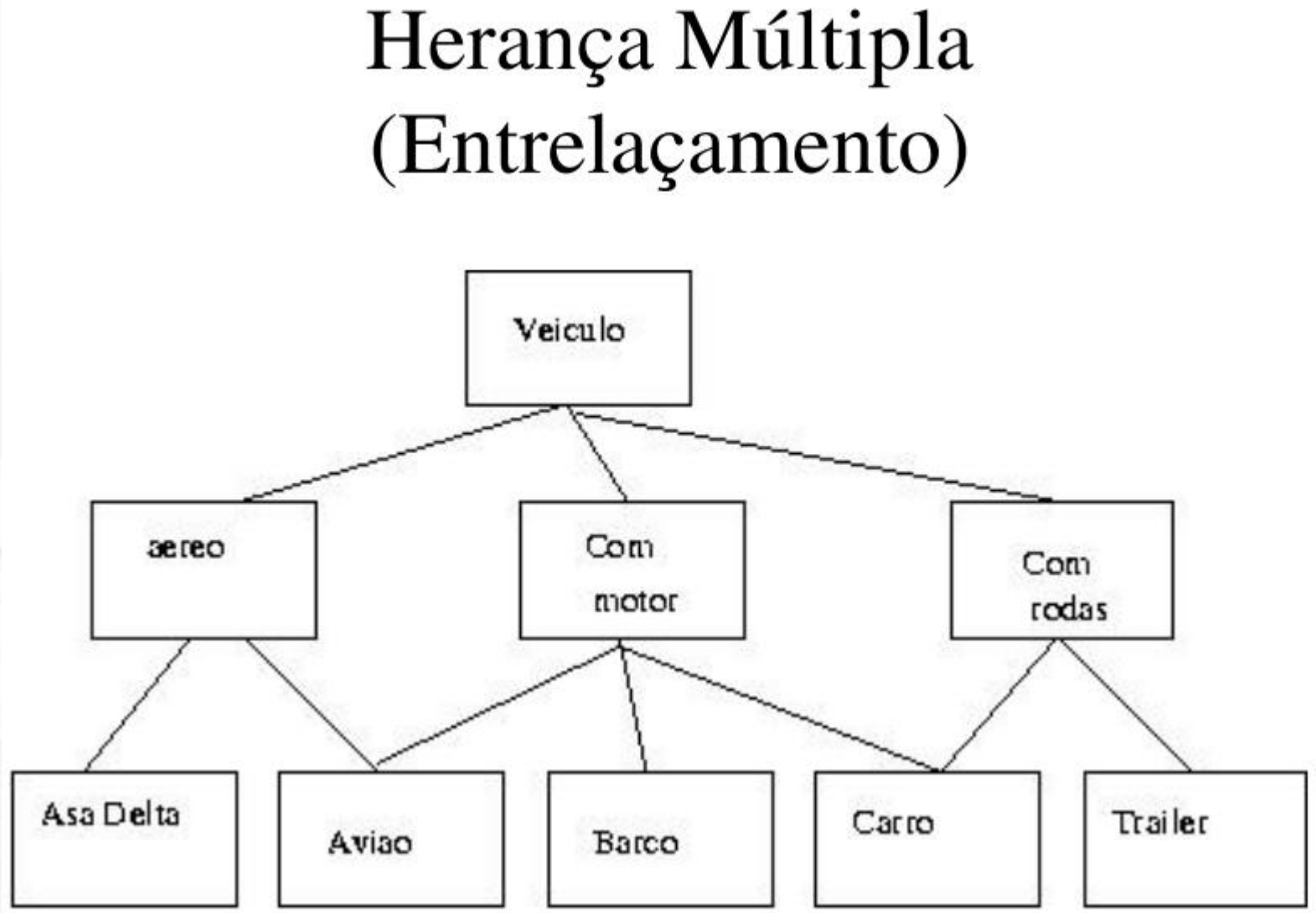


Herança Múltipla

Herança Múltipla



Herança Múltipla



Polimorfismo

- Polimorfismo caracteriza a criação de funções com um mesmo nome, mas códigos diferentes, facilitando a extensão de sistemas.
- Um tipo de polimorfismo é a redefinição de métodos para uma classe derivada.
- O acréscimo de um segundo construtor à classe Student, como mostra o exemplo, é um exemplo de sobrecarga de métodos.

Polimorfismo

```
public class Student extends Person
{
  private String grade; // serie cursada
  private int classes; // aulas que frequenta
  public Student(String n, String g)
  {
    super(n); //chamada ao construtor da superclasse
    grade = g;
    classes = 0;
  }
  public Student(String n, String g, int f)
  {
    super(n); //chamada ao construtor da superclasse
    grade = g;
    classes = f;
  }
} //fim da classe Student
```

Manipulação de objetos

- Quando declara-se uma variável cujo tipo é o nome de uma classe, como em:

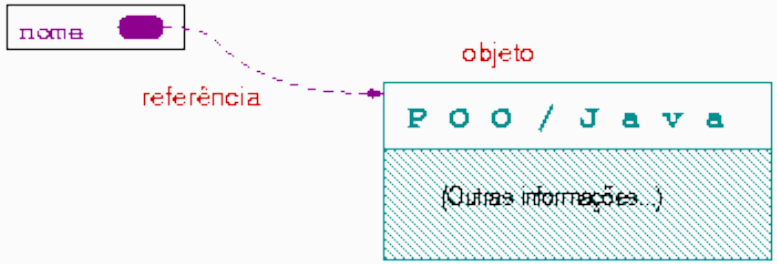
```
String nome;
```

não está se criando um objeto dessa classe, mas simplesmente uma **referência para um objeto** da classe String, a qual inicialmente não faz referência a nenhum objeto válido:

Quando um objeto dessa classe é criado, obtém-se uma referência válida, que é armazenada na variável cujo tipo é o nome da classe do objeto. Por exemplo, quando cria-se uma *string* como em:

```
nome = new String("POO/Java");
```

nome é uma variável que armazena uma referência para um objeto específico da classe String -- o objeto cujo conteúdo é "POO/Java":

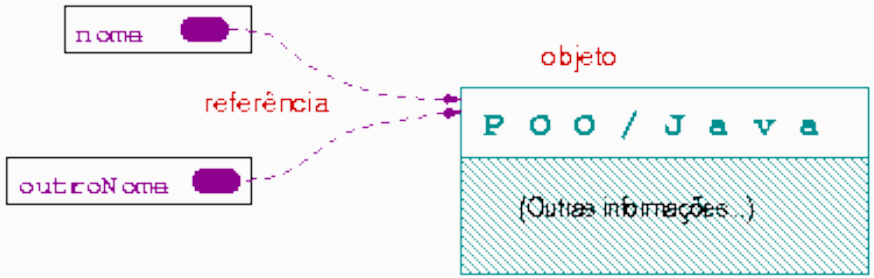


Manipulação de objetos

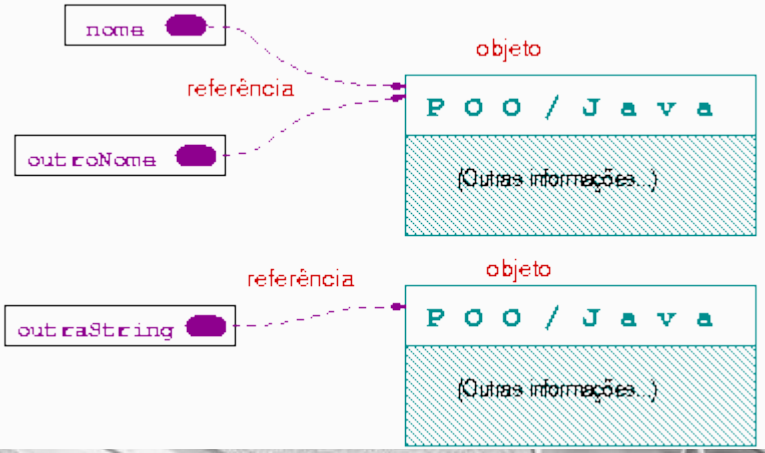
- É importante ressaltar que a variável nome mantém apenas a referência para o objeto e não o objeto em si. Assim, uma atribuição como:

```
String outroNome = nome;
```

não cria outro objeto, mas simplesmente uma outra referência para o mesmo objeto:



É possível "clonar" o objeto?



Clonagem de objetos em Java

FAESA faça acontecer

```
package clone;

public class CloneClass implements Cloneable
{
	int a; double b;
	// This method calls Object's clone().
	CloneClass getClone()
	{
		try // call clone in Object.
		{
			return (CloneClass) super.clone();
		} catch (CloneNotSupportedException e)
		{
			System.out.println ("Clonagem não
suportada." );
			return this;
		}
	}
}
```

Clonagem de objetos em Java

- No exemplo acima, o método *getClone* chama o método clone no objeto e retorna o objeto. É necessário notar-se aqui que o objeto que é retornado depois do mecanismo de clonagem deve ser tipado em seu tipo apropriado, neste caso é uma *CloneClass*.

- Se a classe não está implementando uma interface clonável, e tentarmos clonar este objeto, receberá uma mensagem de *CloneNotSupportedException*. No processo de clonagem, o construtor não é chamado, ao invés disso, uma cópia exata do dito objeto é criada. Mas o objeto clonado tem que implementar uma interface clonável.

- O método *clone()* da classe *Object* cria e retorna uma cópia do objeto, com a mesma classe e com os mesmos campos tendo os mesmos valores. No entanto, *Object.clone()* mostrará uma *CloneNotSupportedException* a não ser que o objeto seja uma instância de uma classe que implemente o marcador da interface *Cloneable*.

Outras Soluções: Construtor de cópia

```
1.public class Aluno
2.{
3.
4.     String nome;
5.     String segundoNome;
6.
7.     public Aluno (){} // construtor normal
8.
9.     public Aluno (Aluno other)
10.    {
11.      this.nome = other.nome;
12.      this.segundoNome = other.segundoNome;
13.    }
14.}
```

Sugestão de estudo: site Programmr

Programmr: Programmer´s Playground

- http://www.programmr.com/