

Programação Orientada por Objetos

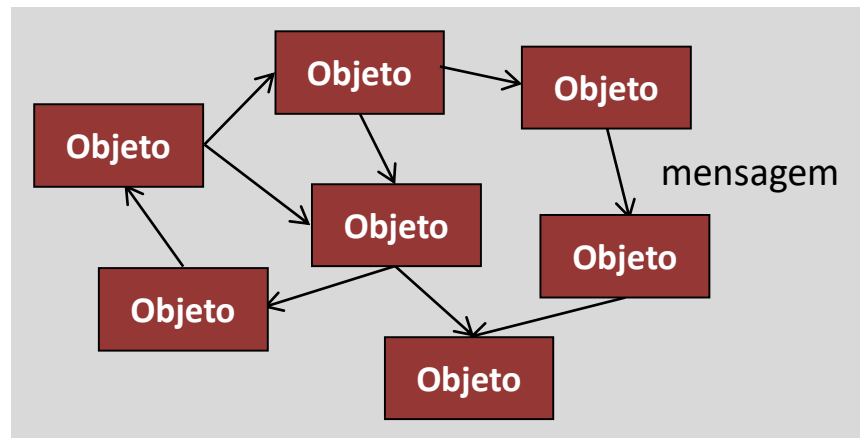
Abstração e Encapsulamento

Classes e Objetos

(Livro *Big Java, Late Objects* – Capítulo 8)

- [Programação Orientada por Objetos \(POO\)](#)
- [Linguagem Java](#)

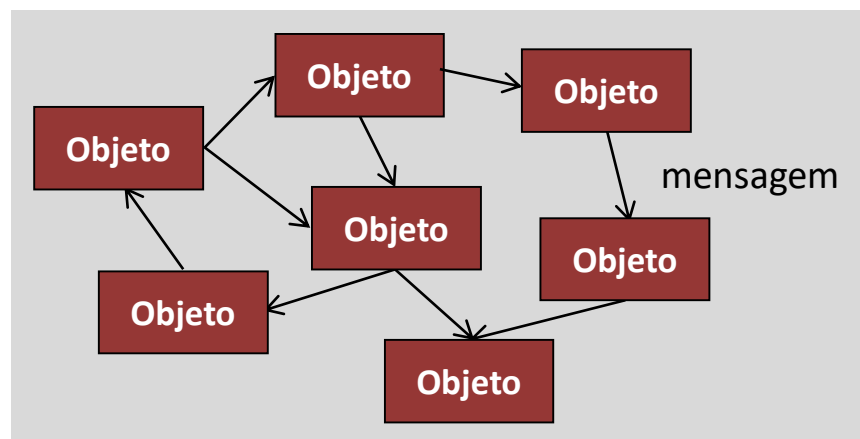
- Constituído por Objetos



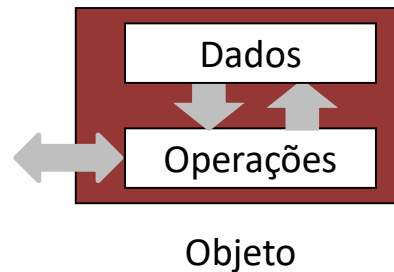
- **Objetos**

- São **estruturas** computacionais autónomas
- Trabalham de forma **cooperativa**
 - Cada um tem uma **responsabilidade** particular no programa
 - Cada um presta **serviços** a outros objetos
 - Solicitados através de **mensagens**
- **Comunicam**, entre si, através de **mensagens**

- É uma **abstração (representação)**
 - Entidade real ou Conceito ... que o programa processa
 - Exemplos:
 - Entidades reais:
 - Pessoa, Automóvel, Cliente // existência física
 - Conceitos:
 - Entrevista, Consulta Médica



- Constituição de um Objeto



- Um objeto **encapsula** (agrega), numa única **estrutura computacional**, ...

... os **dados** (atributos essenciais de uma abstração) e as **operações** que manipulam esses dados, ...

... de modo a ...

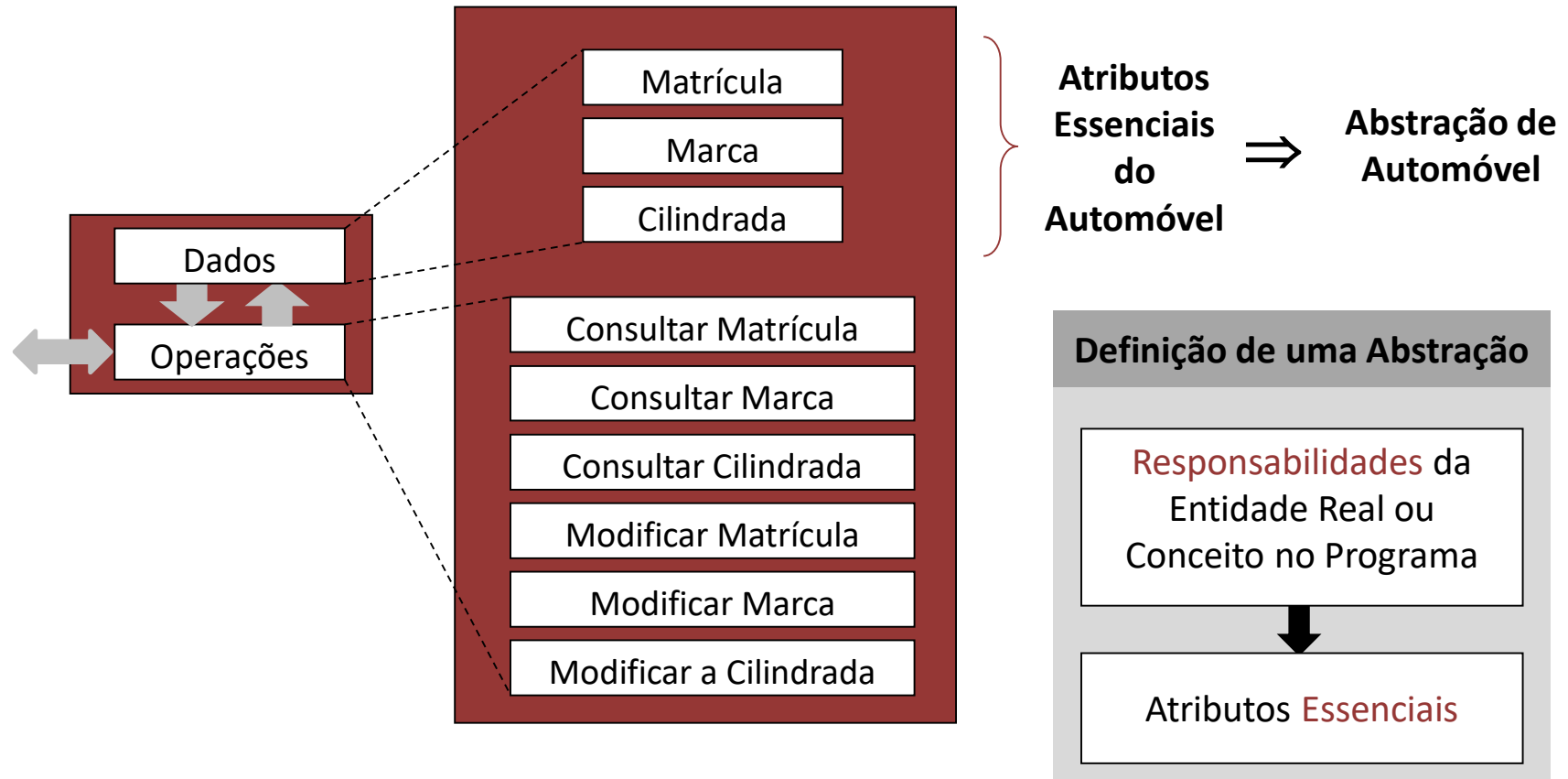
... permitir que os **dados** sejam acessíveis do exterior, **apenas** através de operações **próprias** da entidade

... **esconder** do exterior da entidade as,

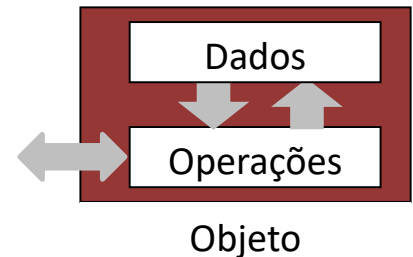
- estruturas de dados
- implementações das operações

Exemplo de Objeto

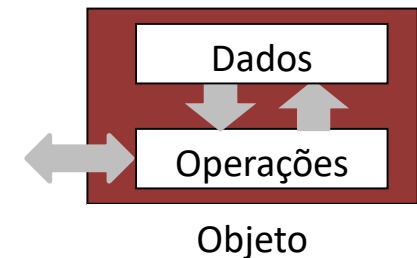
- Objeto Automovel // **Representação** abstrata de Automóvel (modelo simplificado)
// Representa apenas atributos essenciais do Automóvel para o programa

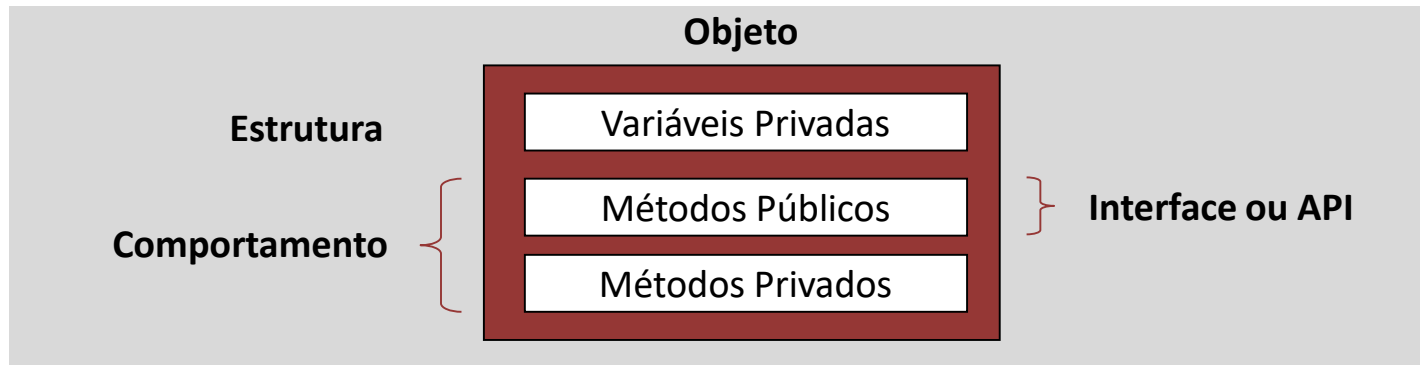


- **Dados acessíveis** do exterior só através de **operações** do próprio objeto
 - Permite controlar as modificações dos dados de modo a garantir a **integridade** dos dados
 - i.e., **dados consistentes** com a entidade real ou conceito representado
 - Exemplo do Objeto Automovel
 - Garantir cilindrada ≥ 0
 - Contribui para a obtenção de **programas** mais **robustos**
 - Sem erros de execução
- **Estruturas** de dados e **implementações** das operações **escondidas** do exterior (funciona como uma **caixa-negra**)
 - Permite alterações deste código de objeto sem implicar modificações dos programas que usam os objetos
 - Contribui para a longevidade dos programas



- **Objeto é uma estrutura autónoma**
 - Independente do contexto de utilização, ou seja, dos programas
 - Vantagens
 - Facilmente **reutilizáveis** em qualquer programa
 - Não precisam de qualquer adaptação
 - Importante para redução do custo de produção dos programas modernos
 - Programas caracterizados por dimensões grandes
 - Facilitam a **deteção de erros**
 - Pode ser testado de forma isolada
 - Importante para criação de código robusto
 - Capacidade de **atualização** sem afetar o código cliente
 - Quer das estruturas de dados
 - Quer das implementações das operações





- **Variáveis**
 - Privadas
 - // para guardar os dados
 - // para satisfazer princípio do encapsulamento
- **Métodos**
 - Públicos
 - // implementam as operações
 - // operações acessíveis do exterior
 - Privados
 - // métodos auxiliares para executarem cálculos intermédios
- **Definições**
 - Estrutura
 - conjunto de todas as variáveis do objeto
 - Estado
 - conjunto de todos os dados (conteúdos das variáveis) do objeto
 - Comportamento
 - conjunto de todos os métodos do objeto
 - Interface ou API
 - conjunto dos métodos acessíveis do exterior

- **Programa**

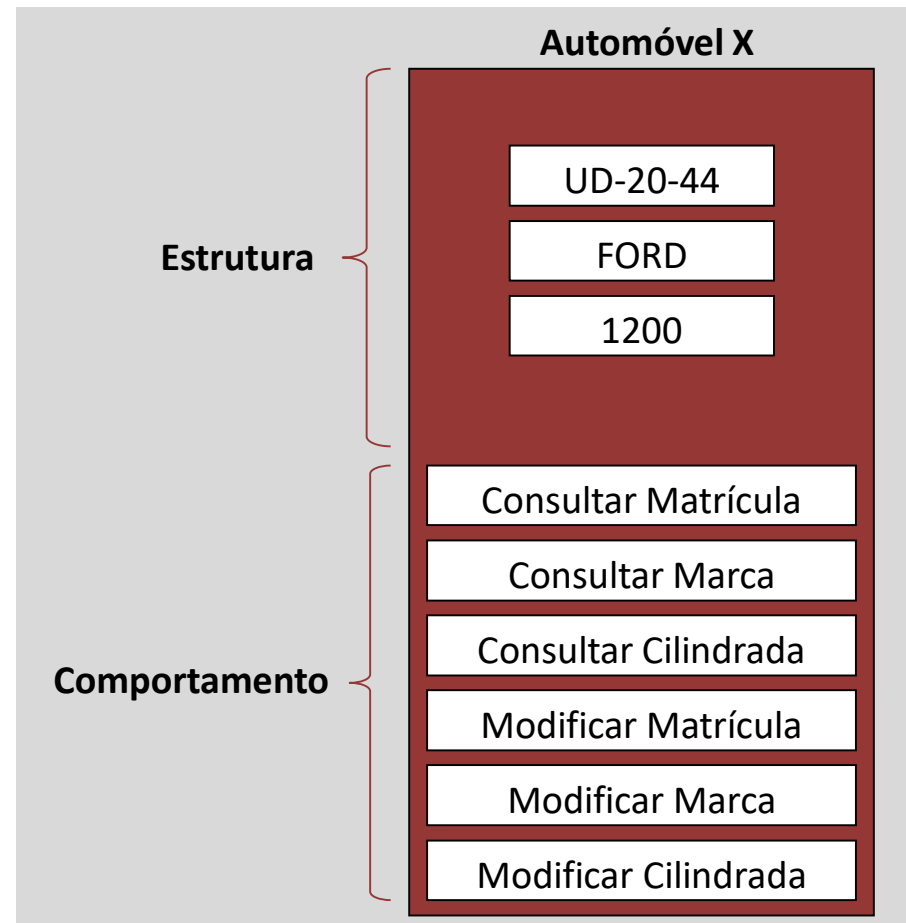
- Geralmente
 - Processa múltiplos **objetos similares** // da mesma espécie

- **Objetos Similares têm**

- Mesma Estrutura
- Mesmo Comportamento

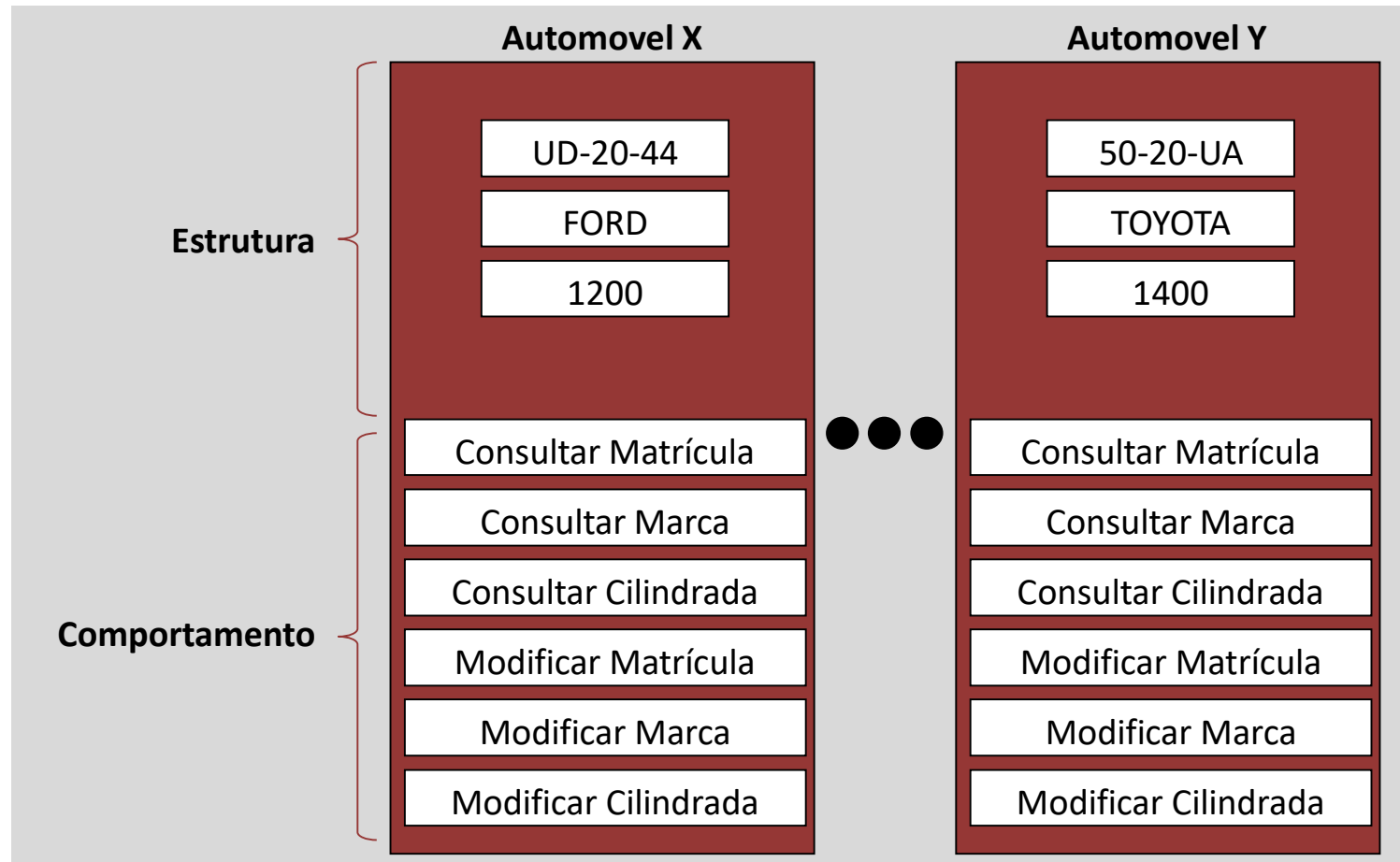
- **Exemplo**

- Programa de gestão de automóveis
- Processa **múltiplos** objetos **Automovel**
 - Estruturas iguais
 - Comportamentos iguais
 - Dados (Estados) diferentes
 - Dados **específicos** de cada objeto

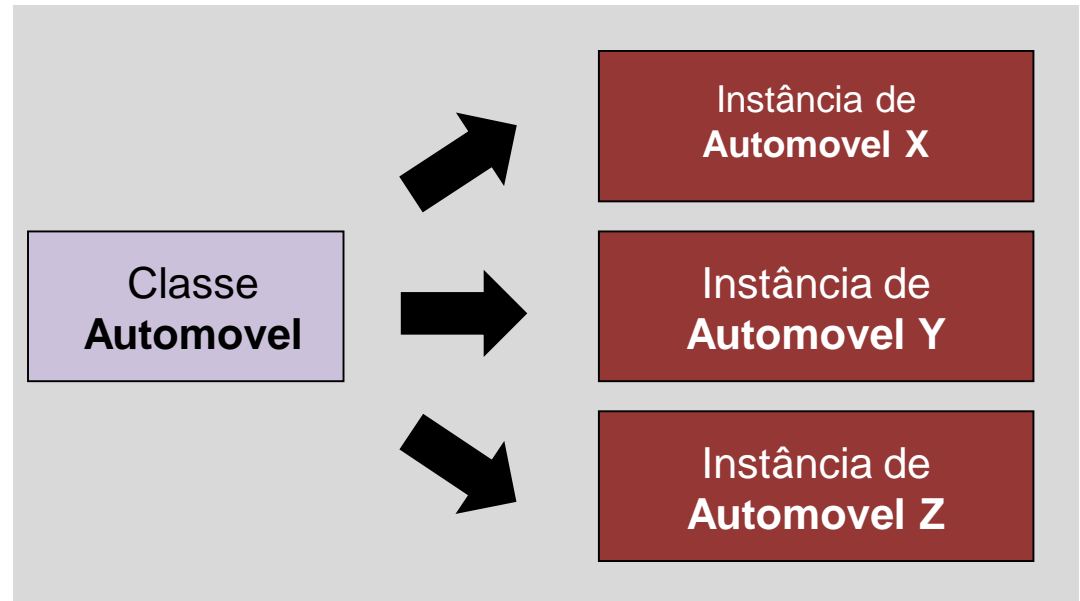


Exemplo

- Programa de gestão de automóveis
- Processa múltiplos objetos **Automovel**
 - Estruturas iguais
 - Comportamentos iguais
 - Dados (**Estados**) diferentes

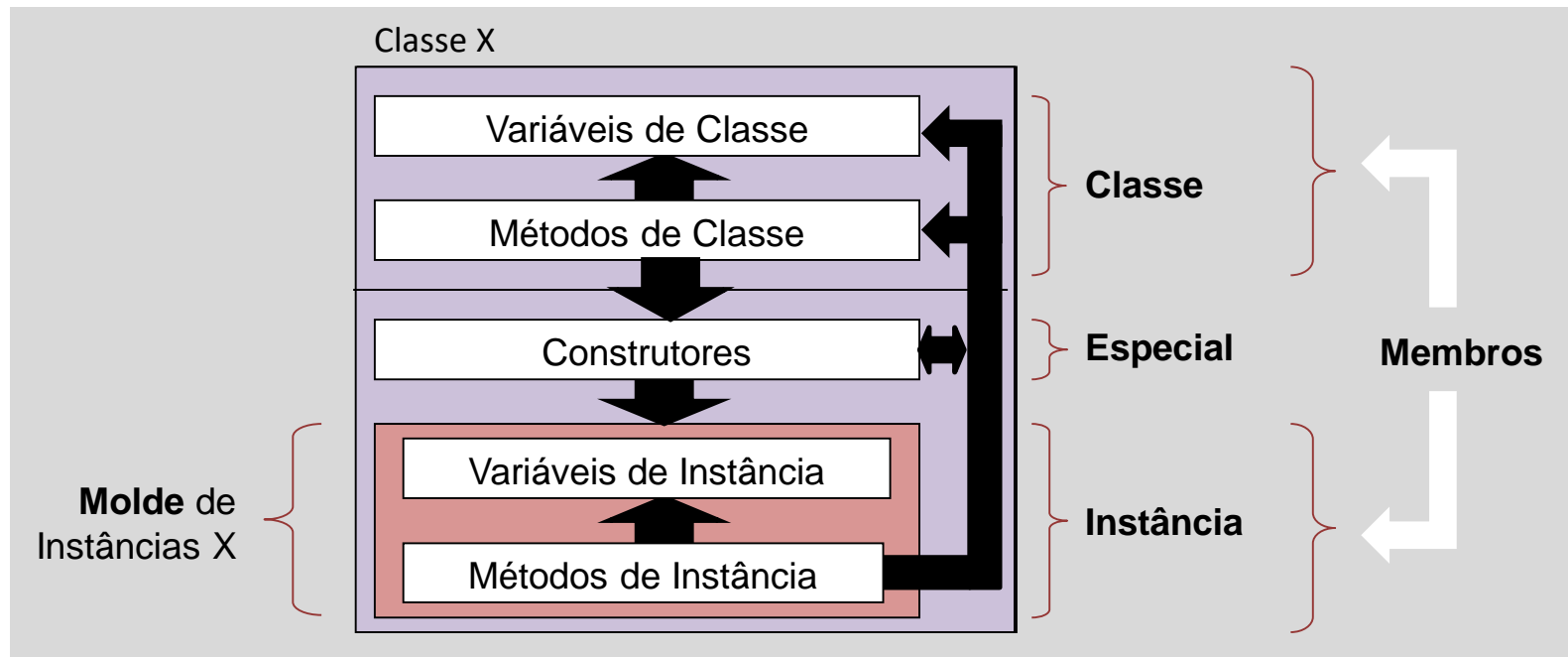


- Para criar múltiplos objetos similares
 - É preciso uma estrutura computacional que guarde a Estrutura (variáveis) e o Comportamento (métodos) desses objetos, para servirem de **molde** na sua construção
 - Essa estrutura é chamada **Classe**
- Tipos de Estruturas Computacionais
 - **Classe**
 - Tem **capacidade** para criar objetos similares
 - Chamados **instâncias** de classe
 - Pode funcionar como **fábrica** de instâncias
 - **Objeto**
 - Instância de uma classe
 - Criado por uma classe



■ Tipos de Membros

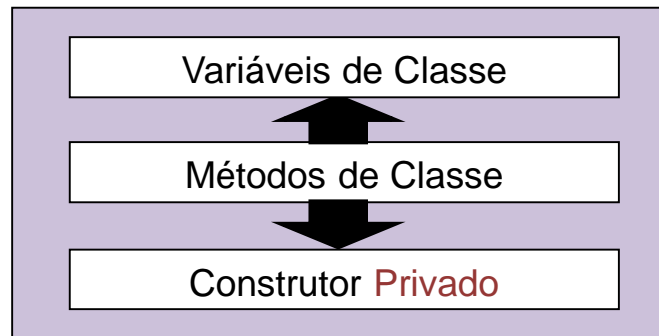
- Membros de Instância
 - Variáveis de instância // Definem a **Estrutura** para guardar o **Estado** das **instâncias** (dados)
 - Métodos de instância // Definem o **Comportamento** das **instâncias** (operações)
- Membros de Classe
 - Variáveis de classe // Definem a **Estrutura** para guardar dados da **classe** (dados **globais**)
 - Métodos de classe // Definem o **Comportamento** da **classe** (operações)
- Especiais
 - Construtores // Criam instâncias, **reproduzindo** as variáveis e mét. de instância



- Essencialmente
 - Podemos considerar dois tipos de classes
 - Classes **Instanciáveis** // **Fábricas** de instâncias (Ex: String)
 - Classes **Não-Instanciáveis** // **Prestadoras** de serviços (Ex: classe Math)
// não disponibilizam para o exterior nenhum construtor

- **Classes Não-Instanciáveis**

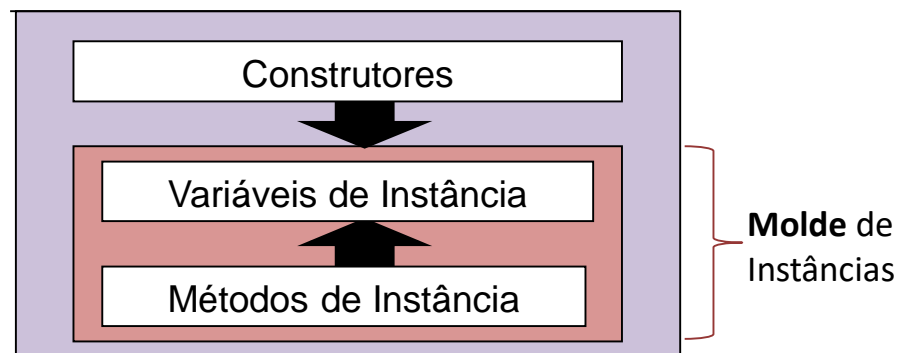
- Classes não geradoras de instâncias
 - Com construtor privado
 - Classe abstrata // abordaremos noutra aula
- Apenas prestam serviços
 - Designadas **Prestadoras de Serviços**
- Estrutura **Geral**



- Exemplo
 - Class **Math**
 - Só presta **serviços de matemática**
 - Exemplos
 - Funções tradicionais: `abs()`, `sin()`, `sqrt()`, etc.
 - Constantes tradicionais: `PI`, `e`

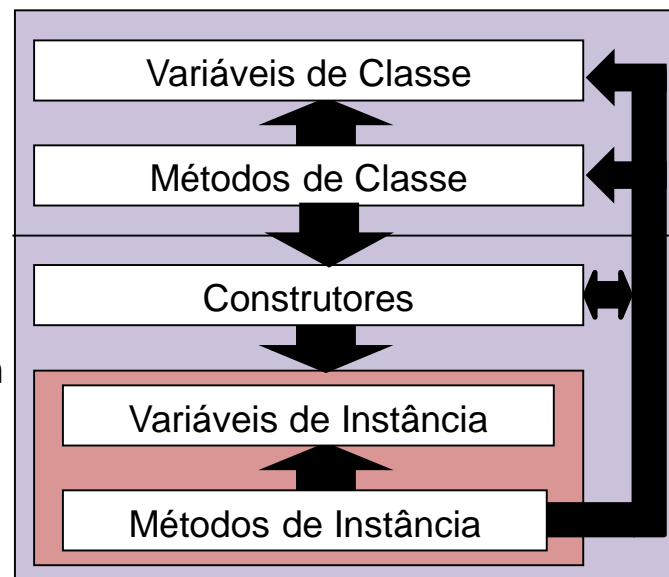
Classes Instanciáveis

- Geradoras de instâncias
 - Designadas **Fábricas de Instâncias**
- Estruturas possíveis
 - Sem variáveis e métodos de classe
 - Fábrica pura

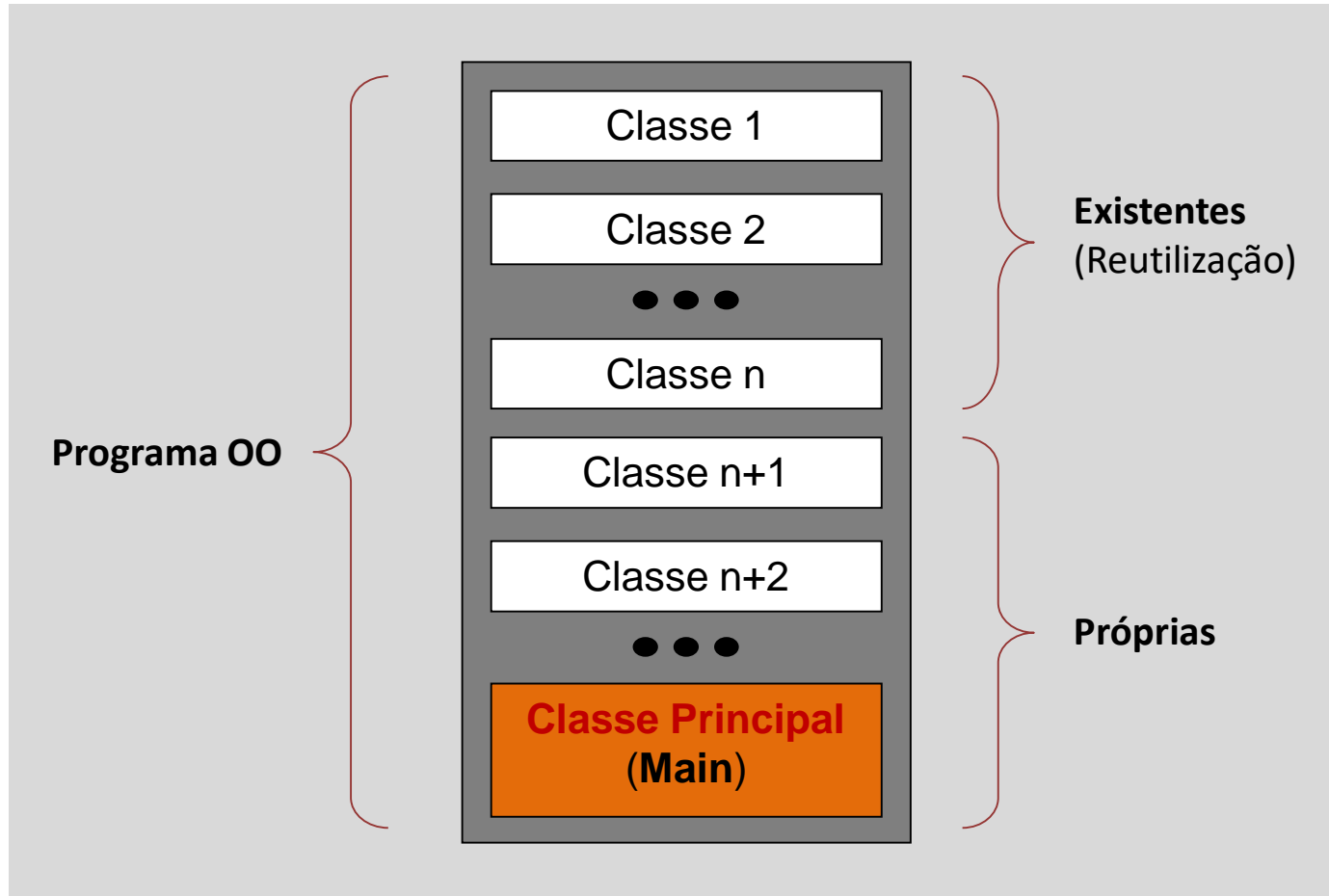


Com variáveis e/ou métodos de classe

- Variáveis de classe
 - Acessíveis a qualquer método da classe
 - Métodos de instância e de classe
 - Partilhadas** por todas as instâncias da classe
 - Interesse
 - Guardar dados **globais** das instâncias
 - Exemplos
 - Factor de conversão comum todas as instâncias
 - Guardar dados da classe
 - Ex: nº de instâncias criadas
- Exemplo
 - Classe **String** // método `format()` é método de classe

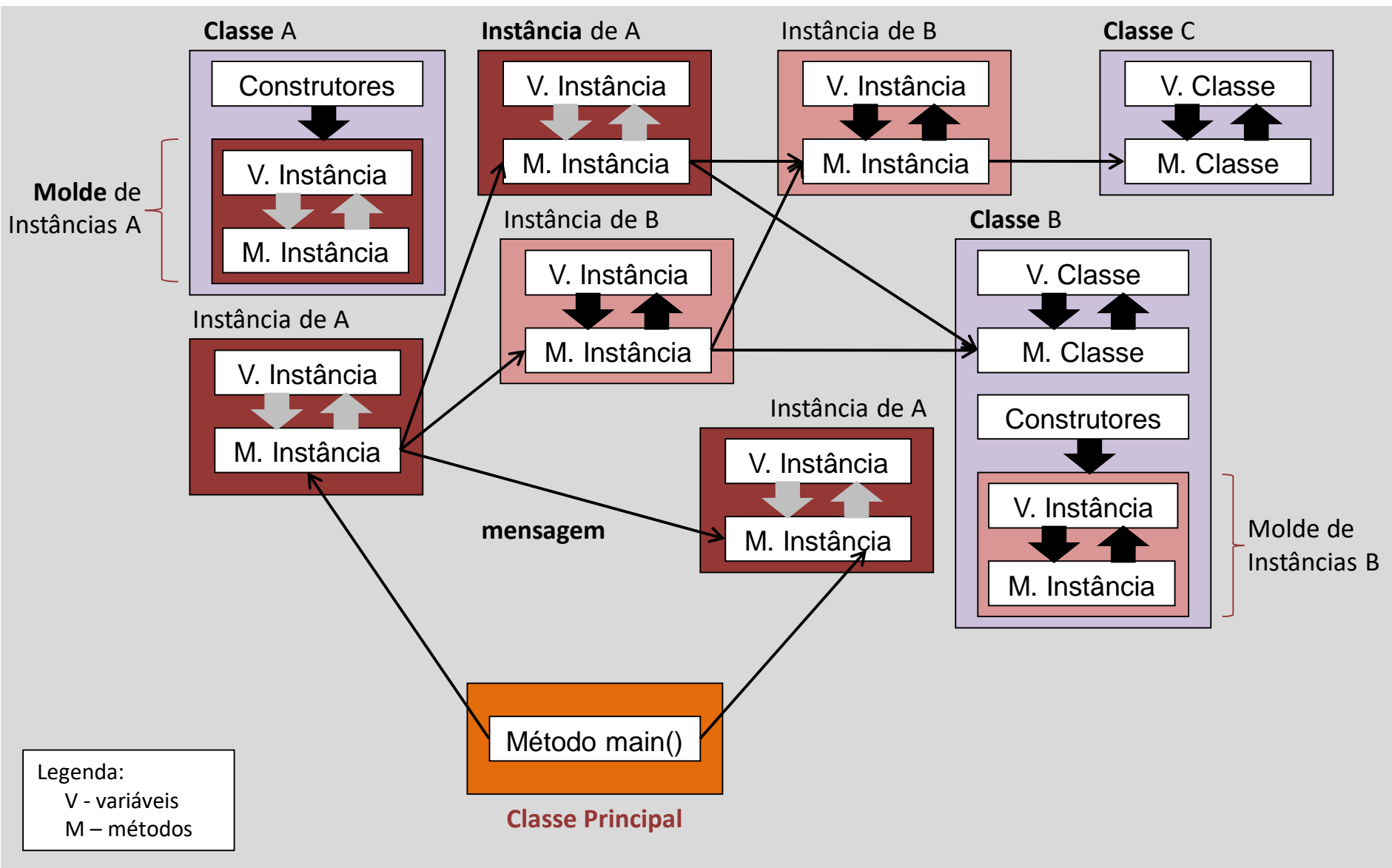


- Consiste num **conjunto de classes**



- Classe Principal (Main)**
 - Tem capacidade para **iniciar e controlar a execução** do programa ... através do método `main()`

Exemplo



- Um Tipo de Dados

- Define

- Um conjunto de dados
 - +
 - Um conjunto de operações sobre esses dados



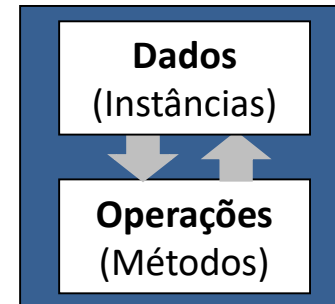
- Exemplo

- Tipo **booleano (bool)**
 - Conjunto de **dados**: { true, false } // **únicos** dados de variável tipo bool
 - Conjunto de **operações**: { AND, OR, NOT } // **únicas** operações sobre esses dados

- Classe

- Define

- Um conjunto de **instâncias**
 - +
 - Um conjunto de **métodos** que podem ser aplicados a essas **instâncias**



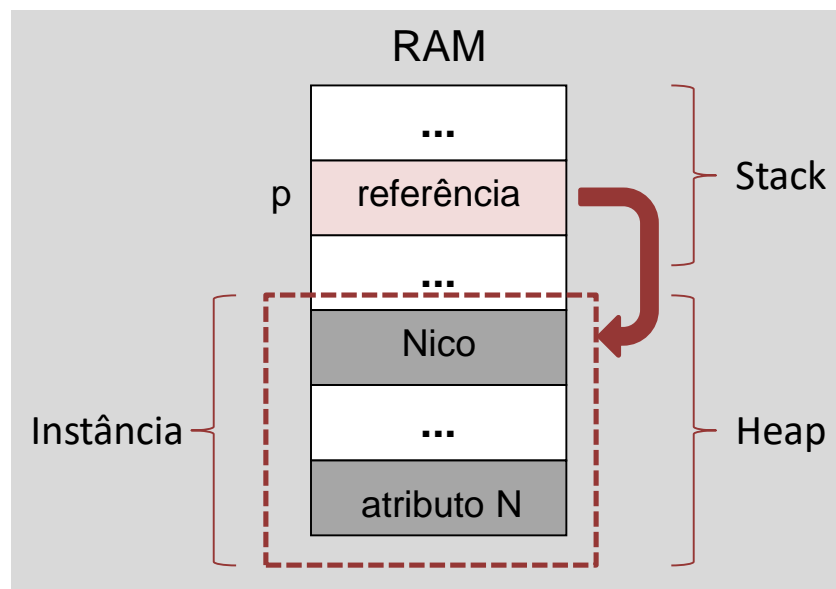
- Instâncias

- São dados dos programas
 - Logo
 - Classe considerada **tipo de dados ... definido pelo utilizador**

- Pode definir o tipo de uma variável

- Sintaxe: Classe nomeVariável;
 - Exemplo: Cliente cliente; // variável cliente para guardar instâncias da classe Cliente

- **Classe é Tipo Referência** (não-primitivo)
 - Variável de Tipo Referência
 - Guarda referência para instância (objeto) // localização de memória
 - Exemplo
 - Classe Pessoa
 - `Pessoa p = new Pessoa("Nico");` // usado na declaração do tipo de uma variável



- [Paradigmas da Programação](#)
- [Programação Orientada por Objetos \(POO\)](#)
- [Programação Java](#)



- [Package](#)
 - [Noção](#)
 - [Exemplos JAVA](#)
 - [Declaração](#)
 - [Importação de Classes](#)
- [Mecanismo de Controlo de Acesso \(Visibilidade\)](#)
 - [Classes](#)
 - [Membros da Classe](#)
- [Classe](#)
 - [Declaração](#)
 - [Organização dos Membros](#)
- [Tipos de Dados](#)
 - [Categorias](#)
 - [Primitivos](#)
 - [Referência](#)
- [Variáveis de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
- [Métodos de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
 - [Sobrecarga \(Overloading\)](#)
 - [Invocação](#)
- [Mecanismo de Mensagens](#)
 - [Tipos de Mensagens](#)
 - [Com e Sem Retorno](#)
 - [Sequência de Mensagens](#)
- [Referência this](#)
- [Métodos de Instância](#)
 - [Categorias](#)
 - [Consulta \(Gets\)](#)
 - [Modificadores \(Sets\)](#)
 - [Condicionados](#)
 - [Validação de Dados](#)
 - [Complementares](#)
 - [toString\(\)](#)
 - [Auxiliares](#)
- [Construtores](#)
 - [Declaração](#)
 - [Sobrecarregados](#)
 - [Invocação this\(\)](#)
 - [Construção de Instâncias](#)
- [Classe Principal de um Programa](#)
 - [Estrutura Básica](#)
- [Operador Condicional \(Ternário\)](#)

■ Interesse

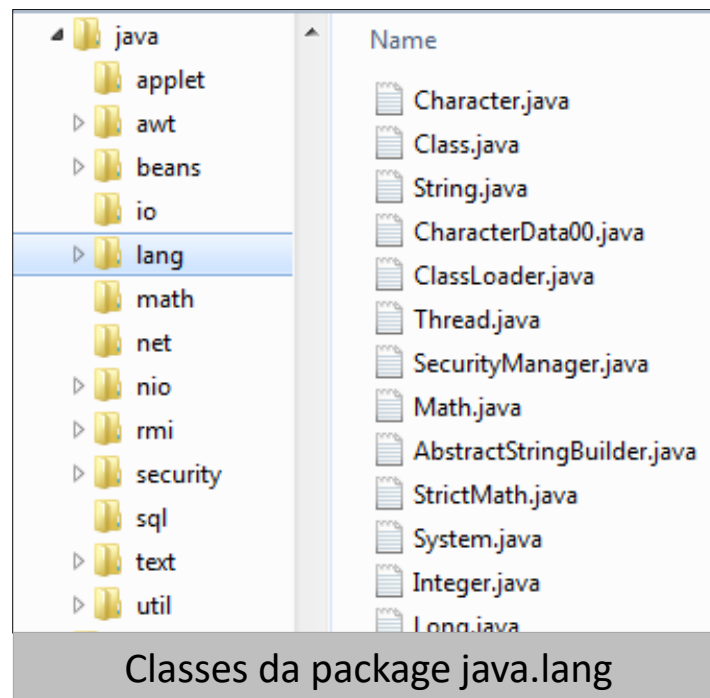
- Permitir **organizar** classes e interfaces Java ... para **facilitar a pesquisa**
- Interesse Análogo
 - Pastas de ficheiros num sistema operativo

■ Package

- Serve para **guardar**
 - Classes // ficheiros
 - Interfaces Java // ficheiros
 - Packages
- Concretamente
 - É **pasta de ficheiros** do sistema operativo

■ Exemplo

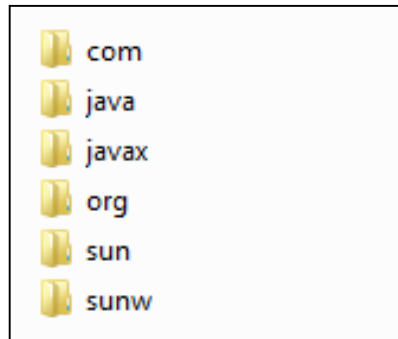
- java.lang
 - Package Java ... fornecida no JDK e JRE
 - Disponibiliza
 - Classes essenciais à execução de programas Java
 - Exemplos
 - ClassLoader // Carregar classes do programa
 - System // Ex: System.in.println(...)
 - String, Math



■ Fornecidas

- JDK
- JRE

■ Nível Superior

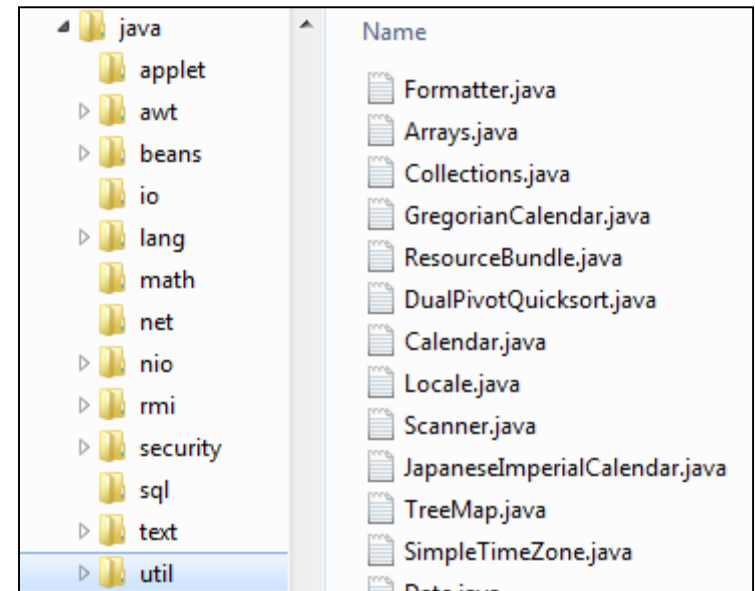
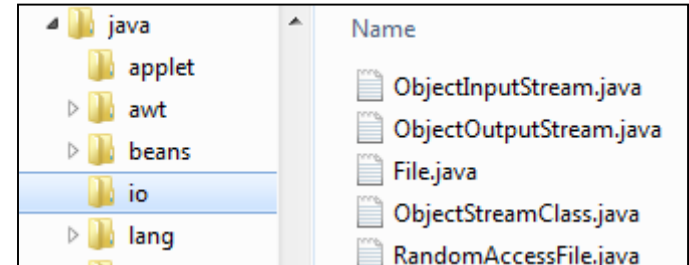


■ Classes Organizadas

- Por funcionalidade

■ Nomes

- Sugerem funcionalidades das suas classes
- Exemplos



Package	Disponibiliza
java.io	Classes relacionadas com operações de entrada/saída (input/output)
java.util	Classes que implementam estruturas e tipos de dados de grande utilidade geral

- Localização
 - Ficheiro `rt.jar`
- Microsoft Windows

Program Files

7-Zip

Common Files

iTunes

Java

jdk1.7.0_02

jre7

bin

lib

amd64

applet

cmm

deploy

ext

fonts

images

management

security

servicetag

zi

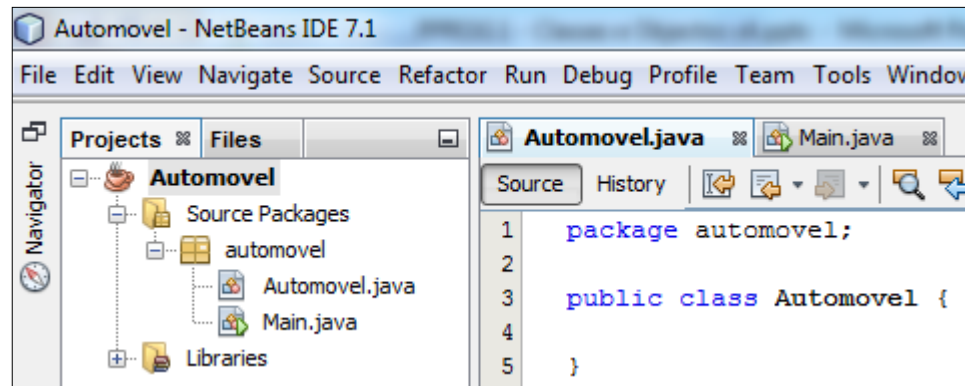
Microsoft Office

Netbeans 7.1

Name	Date modified	Type	Size
tzmappings	09-02-2012 04:24	File	8 KB
sound.properties	09-02-2012 04:24	PROPERTIES File	2 KB
rt.jar	09-02-2012 04:24	Executable Jar File	50.550 KB
resources.jar	09-02-2012 04:24	Executable Jar File	2.407 KB
psfontj2d.properties	09-02-2012 04:24	PROPERTIES File	11 KB
psfont.properties.ja	09-02-2012 04:24	JA File	3 KB
plugin.jar	09-02-2012 04:24	Executable Jar File	1.790 KB
net.properties	09-02-2012 04:24	PROPERTIES File	3 KB
meta-index	09-02-2012 04:24	File	3 KB
management-agent.jar	09-02-2012 04:24	Executable Jar File	1 KB
logging.properties	09-02-2012 04:24	PROPERTIES File	3 KB
jvm.hprof.txt	09-02-2012 04:24	TXT File	5 KB
jsse.jar	09-02-2012 04:24	Executable Jar File	508 KB
jce.jar	09-02-2012 04:24	Executable Jar File	107 KB
javaws.jar	09-02-2012 04:24	Executable Jar File	861 KB
fontconfig.properties.src	09-02-2012 04:24	SRC File	11 KB
fontconfig.bfc	09-02-2012 04:24	BFC File	4 KB
flavormap.properties	09-02-2012 04:24	PROPERTIES File	4 KB
deploy.jar	09-02-2012 04:24	Executable Jar File	3.637 KB
currency.data	09-02-2012 04:24	DATA File	5 KB
content-types.properties	09-02-2012 04:24	PROPERTIES File	6 KB

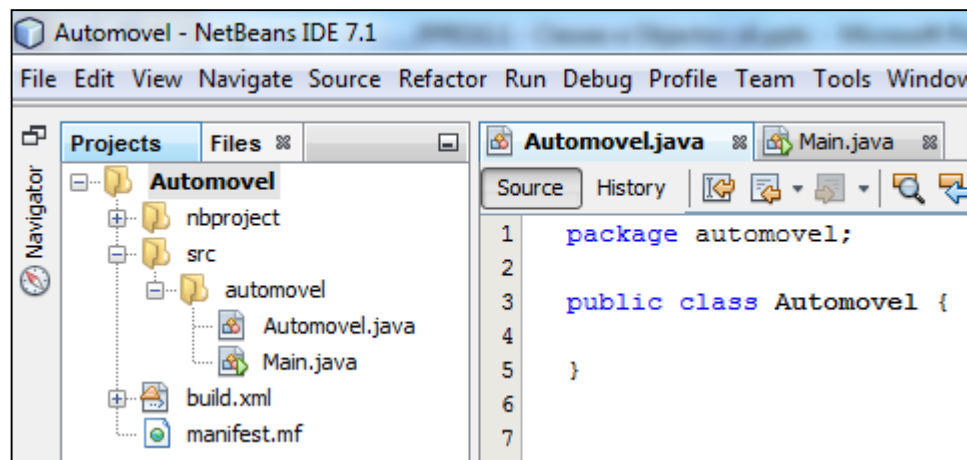
- Exemplo: Projeto Automovel

- Classe Automovel
 - Declarada pertencente à package automovel



Declaração da
package da classe

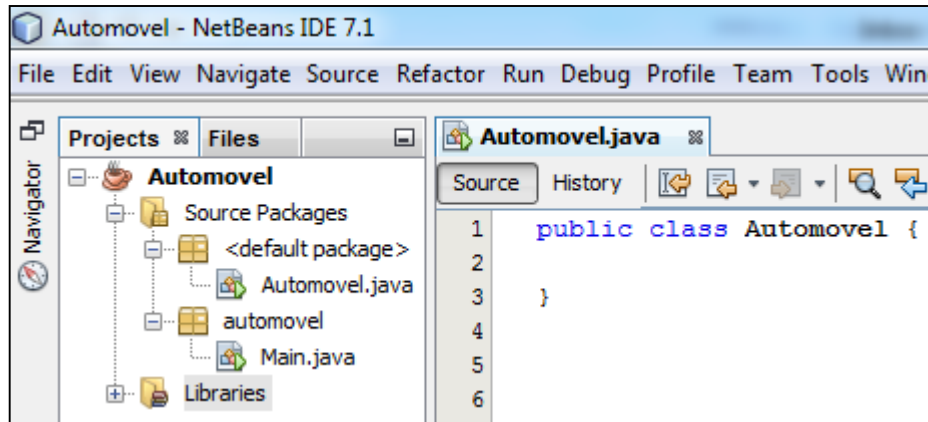
- Localização
 - Indicada na janela Files
 - Pasta: Automovel\src\automovel // pasta Automovel = pasta do projeto



- Exemplo: Projeto Automovel

- Classe Automovel

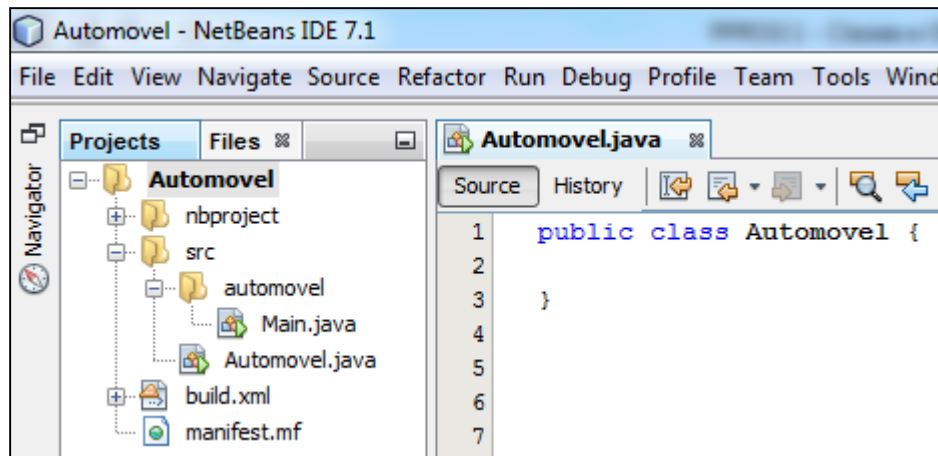
- Package não declarada \Rightarrow guardada na package por omissão (<default package>)



Package da classe
não declarada

- Localização

- Indicada na janela Files
 - Pasta: Automovel\src // pasta Automovel = pasta do projeto



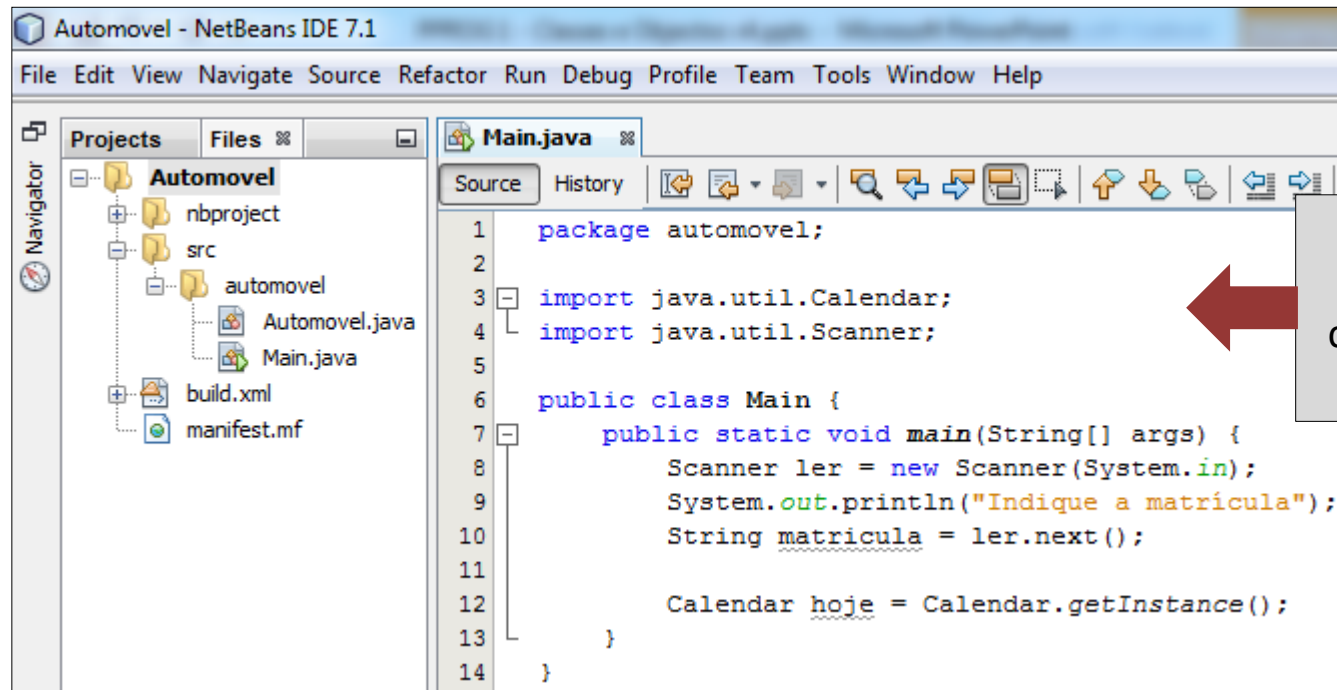
- **Interesse**

- Usar classes guardadas noutras packages // diferentes da package da classe a programar

- **Declaração de Importação**

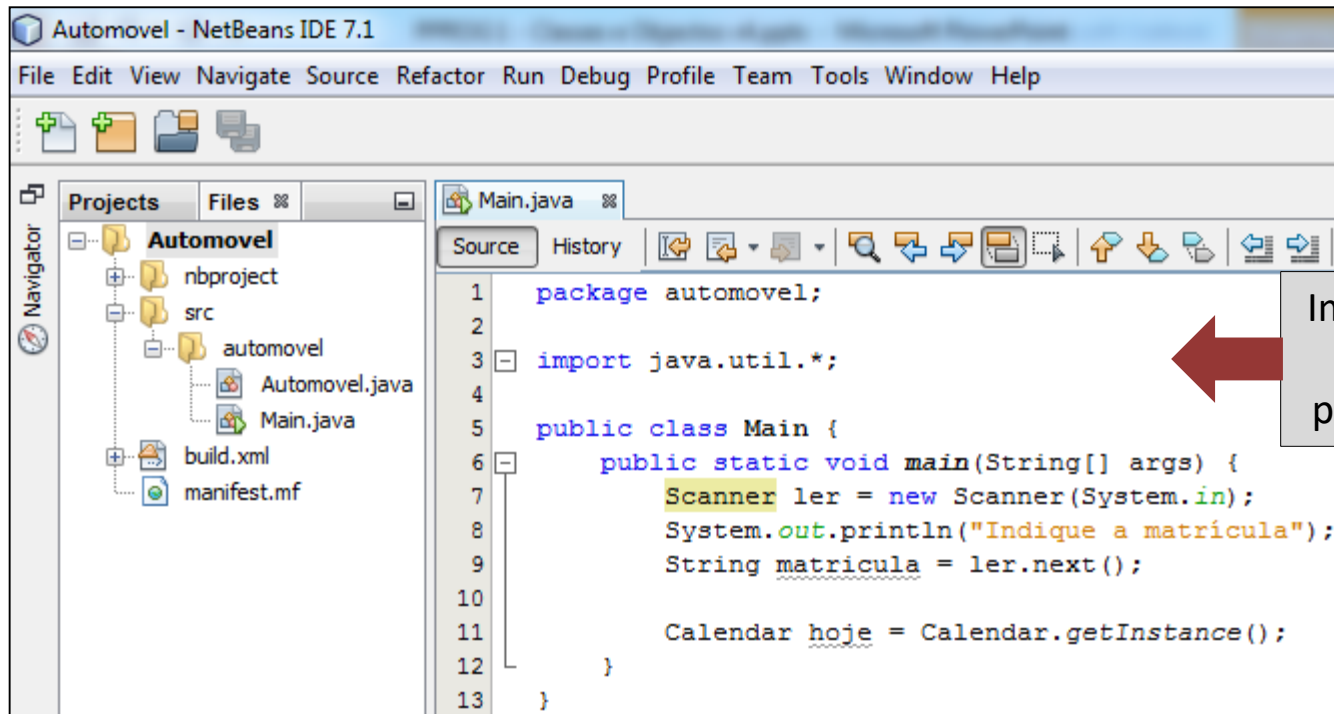
- Tipos de Importação
 - Individual
 - Geral

- Declaração de Importação
 - Importação Individual



Importação individual das classes Calendar e Scanner

- Declaração de Importação
 - Importação Geral



Importação geral
das classes da
package java.util

- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação
- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
 - Referência this
 - Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Condicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)

- Interesse

- Especificar quem tem acesso (**visibilidade**) às entidades definidas:
 - Classes
 - Membros das classes
 - Variáveis
 - Métodos
 - Construtores

- Tipos de acesso a uma entidade

Tipos de Acesso	Declaração (Modificador de Acesso)
Privado	private
Package	(nenhum)
Protegido	protected
Público	public



Acessibilidade

- Tipos de acesso a **membros** de uma **classe** (variáveis, métodos e construtores)

Tipos de Acesso	Membro acessível à
Privado	Própria classe
Package	Própria classe e classes dentro da sua package
Protegido	Própria classe, classes dentro da sua package e qualquer subclasse
Público	Qualquer classe

- Tipos de acesso a uma **classe**

Tipos de Acesso	Classe acessível a
Privado	
Protegido	
Package	Classes dentro da sua package
Público	Qualquer classe

São classes especiais (chamadas classes internas); serão abordadas mais tarde

- [Package](#)
 - [Noção](#)
 - [Exemplos JAVA](#)
 - [Declaração](#)
 - [Importação de Classes](#)
- [Mecanismo de Controlo de Acesso \(Visibilidade\)](#)
 - [Classes](#)
 - [Membros da Classe](#)
- [Classe](#)
 - [Declaração](#)
 - [Organização dos Membros](#)
- [Tipos de Dados](#)
 - [Categorias](#)
 - [Primitivos](#)
 - [Referência](#)
- [Variáveis de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
- [Métodos de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
 - [Sobrecarga \(Overloading\)](#)
 - [Invocação](#)



- [Mecanismo de Mensagens](#)
 - [Tipos de Mensagens](#)
 - [Com e Sem Retorno](#)
 - [Sequência de Mensagens](#)
- [Referência this](#)
- [Métodos de Instância](#)
 - [Categorias](#)
 - [Consulta \(Gets\)](#)
 - [Modificadores \(Sets\)](#)
 - [Condicionados](#)
 - [Validação de Dados](#)
 - [Complementares](#)
 - [toString\(\)](#)
 - [Auxiliares](#)
- [Construtores](#)
 - [Declaração](#)
 - [Sobrecarregados](#)
 - [Invocação this\(\)](#)
 - [Construção de Instâncias](#)
- [Classe Principal de um Programa](#)
 - [Estrutura Básica](#)
- [Operador Condicional \(Ternário\)](#)

▪ Sintaxe

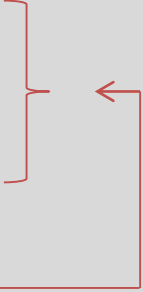
```
[modificador de acesso] [final] class NomeClasse [extends SuperClasse] [implements Interfaces] {  
    //membros da classe  
}
```

- [...] opcional
- **modificador de acesso** private, public, protected ou sem modificador = package
- **final** classe não pode ser herdada
considerada classe completa (não há especializações)
- **NomeClasse** letra inicial maiúscula
- **extends** aplica-se a classe que estende outra classe (herança)
- **implements** aplica-se a classe que implementa um ou mais interfaces

▪ Exemplo

```
public class Automovel{                                // Nome da classe iniciado com letra maiúscula  
    ...  
}
```

```
[modificador de acesso] [final] class NomeClasse [extends SuperClasse] [implements Interfaces] {  
    // variáveis de instância  
    // constantes de classe  
    // variáveis de classe  
  
    // membros públicos  
        // construtores  
        // métodos de instância  
            // métodos de consulta (gets)  
            // métodos de modificação (sets)  
            // métodos complementares e auxiliares  
        // métodos de classe  
        // organização  
  
    // outros membros privados  
        // métodos de instância  
        // métodos de classe  
}
```



- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação



- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
- Referência this
- Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Conicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)

■ Categorias

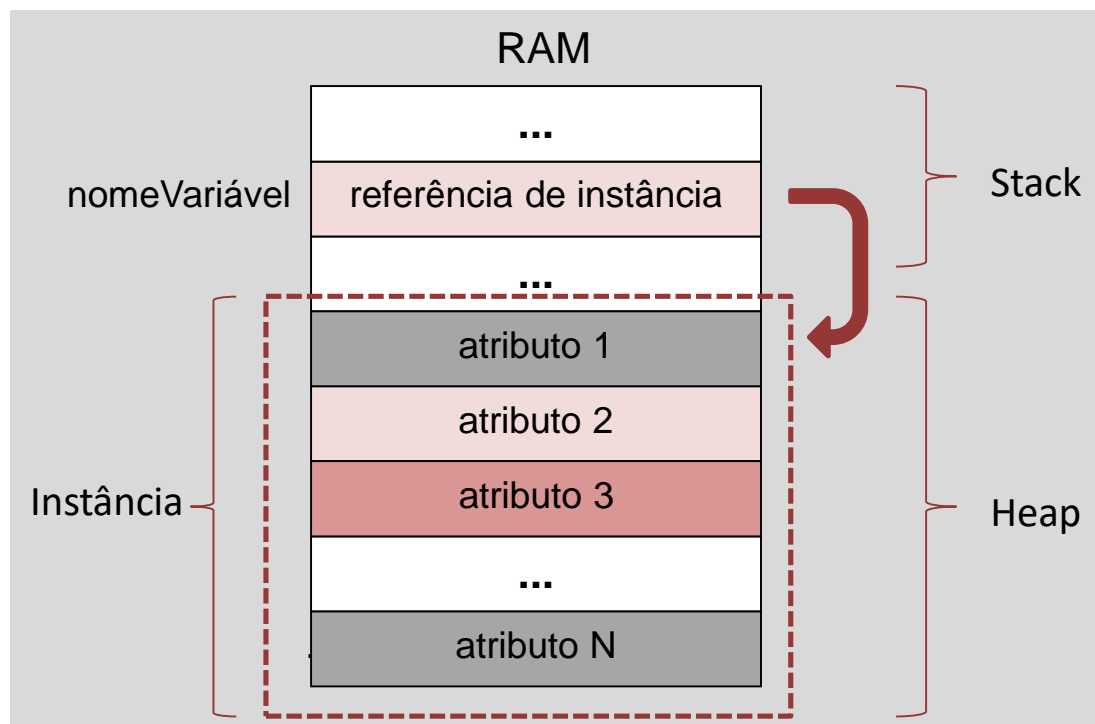
- Primitivos
- Referência

■ Tipos Primitivos

- Tipos Simples
- Variável de tipo primitivo guarda
 - Um **valor simples**

■ Referência

- Tipos Complexos
 - Classes
- Variável de tipo **referência** guarda
 - Localização de memória de instância



▪ Tipos Primitivos

- Inteiros:
 - `byte` 1 byte (-128, 127)
 - `short` 2 bytes (-32 768, 32 767)
 - `int` 4 bytes (-2 147 483 648, 2 147 483 647)
 - `long` 8 bytes (-9×10^{18} , 9×10^{18})
- Reais:
 - `float` 4 bytes ($-/+ 3.4 \times 10^{38}$)
 - `double` 8 bytes ($-/+ 1.7 \times 10^{308}$)
- Outros:
 - `char`
 - `boolean`

▪ Notas

- Nomes
 - Letras minúsculas
- Gammas de Valores
 - Não dependem da máquina
 - Ao contrário do C/C++
 - Interesse
 - Proporcionar portabilidade do código entre diferentes plataformas ou sistemas operativos

▪ Tipos Referência

// alguns exemplos

▪ Texto: String

// Memória Principal

▪ Ficheiro Texto

// Memória Secundária

▪ Ler: Scanner

▪ Escrever: Formatter

▪ Números

▪ Inteiro: Integer, Long

▪ Real: Double, Float

▪ Notas

▪ Nomes (de classes)

▪ Letra inicial maiúscula

- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação



- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
- Referência this
- Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Condicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)

- Interesse

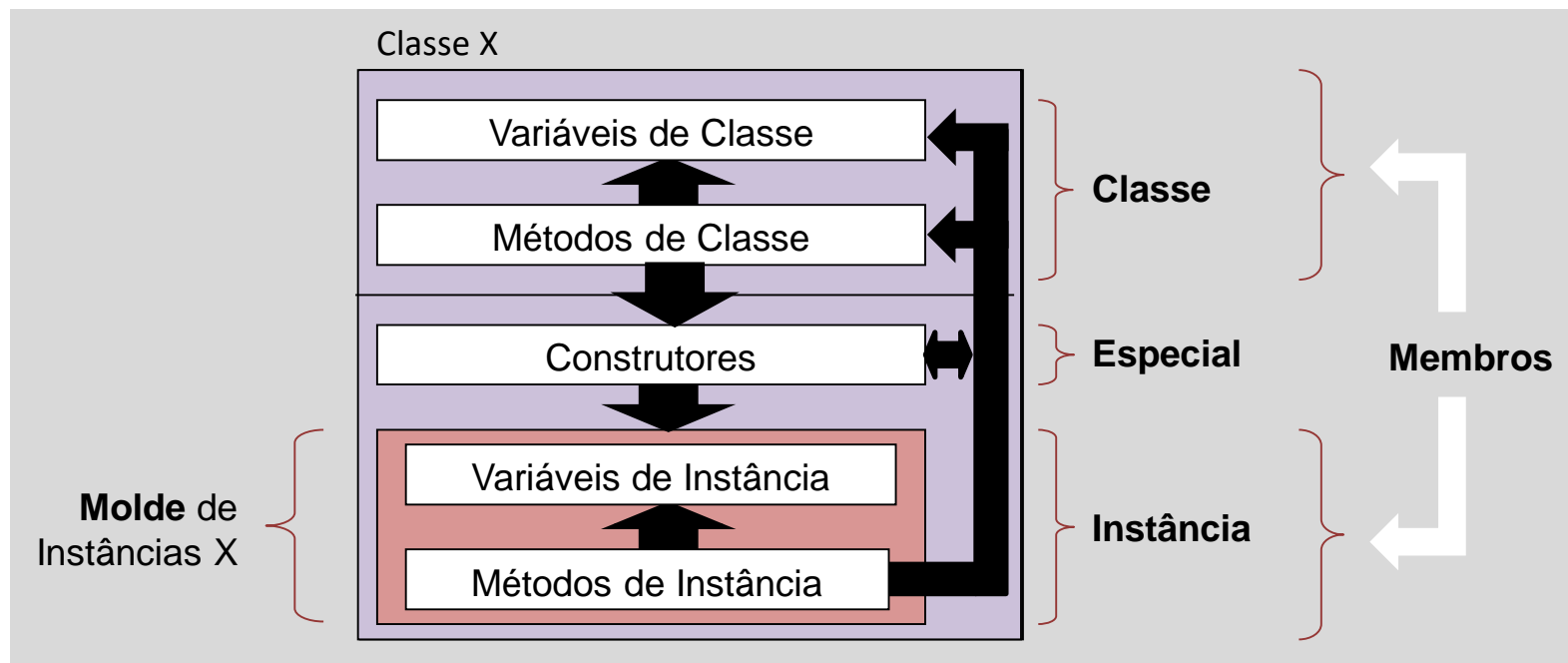
- Definem a **Estrutura** para guardar o **Estado** (dados) das instâncias
 - Dados específicos de cada instância

- Acessíveis a

- Construtores // para inicialização
- Métodos de Instância // para consulta, modificação

- Inacessíveis a

- Métodos de Classe

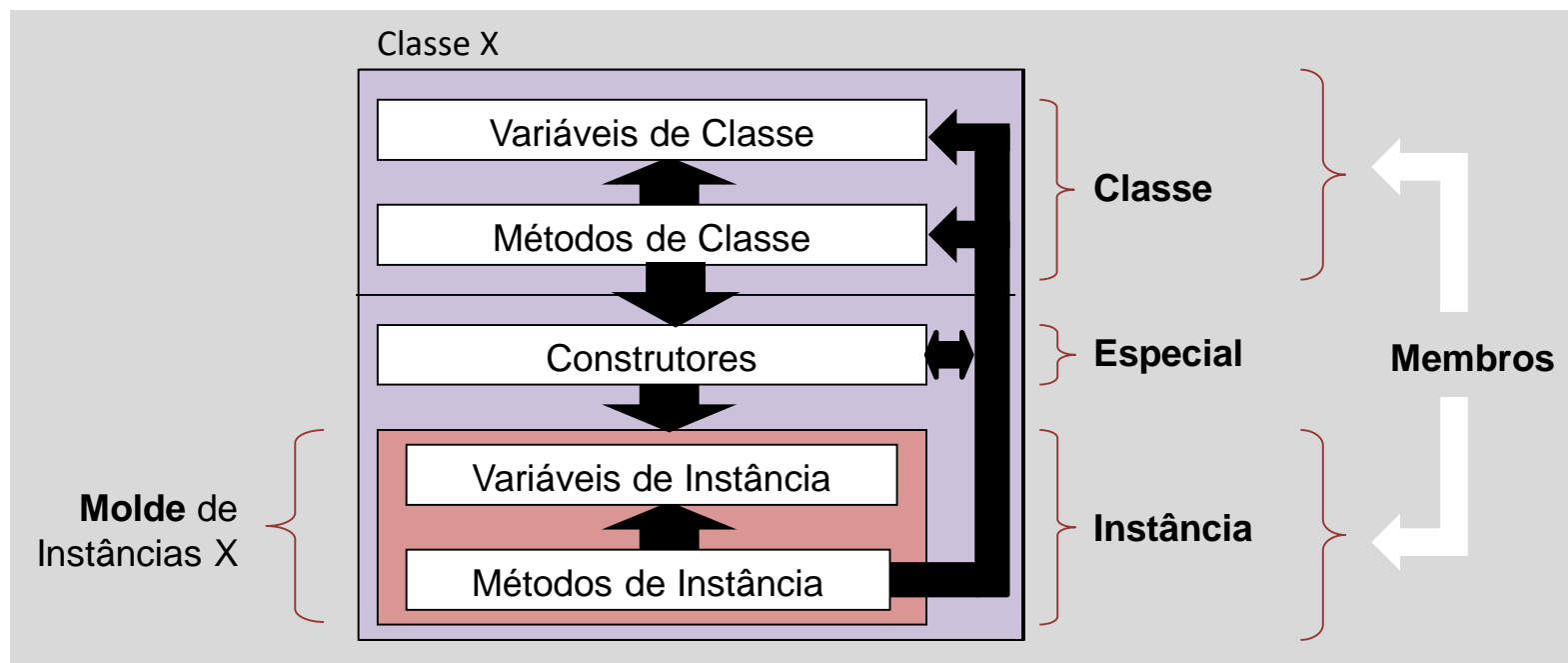


Interesse

- Definem a **Estrutura** para guardar os **dados** da **classe**
 - Dados **globais** da classe
 - Comuns** a todas as instâncias // Ex: taxa de juro das contas bancárias
 - Partilhados** por todas as instâncias da classe

Acessíveis a

- Métodos de Classe e de Instância
- Construtores



▪ **Sintaxe**

```
[modificador de acesso] [final] [static] tipo nomeVariável [=valor_inicial];
```

- [...] opcional
- **modificador de acesso** public, private, protected ; sem modificador = package
- **final** só pode ser feita uma atribuição; torna variável numa constante
- **static** **variável de classe**; sem static => **variável de instância**
- **tipo** primitivo ou referência
- **nomeVariável** nomes simples (não-compostos) devem ter apenas letras minúsculas
- **=valor_inicial** para atribuir o valor inicial

▪ **Exemplo**

```
public class Cliente {  
    // variáveis de instância  
    private String nome;  
    private String morada;  
    private int numero;  
    private int nif;  
    private int nbi;  
  
    // variáveis de classe  
    private static int quantidadeClientes = 0;  
    private static int proximoCliente = 0;  
}
```

- Declarações fora de métodos
- **Private** para garantir o princípio do encapsulamento
- Variáveis de **instância** **devem** ser **inicializadas pelos construtores** da classe

- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação



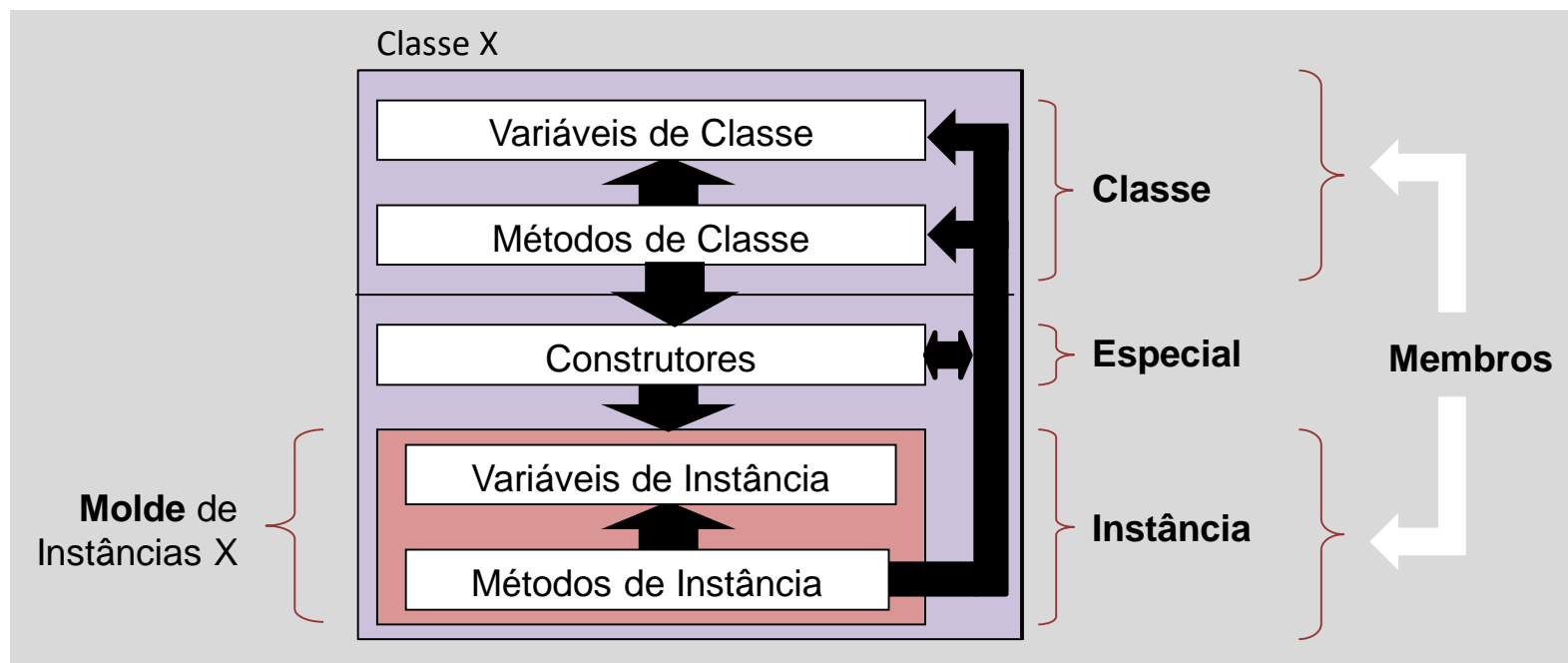
- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
- Referência this
- Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Condicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)

- Interesse

- Definem o **Comportamento** das **instâncias** // operações sobre os dados

- Têm acesso a

- **Variáveis** de Instância e Classe
 - **Métodos** de Instância e de Classe
 - Construtores

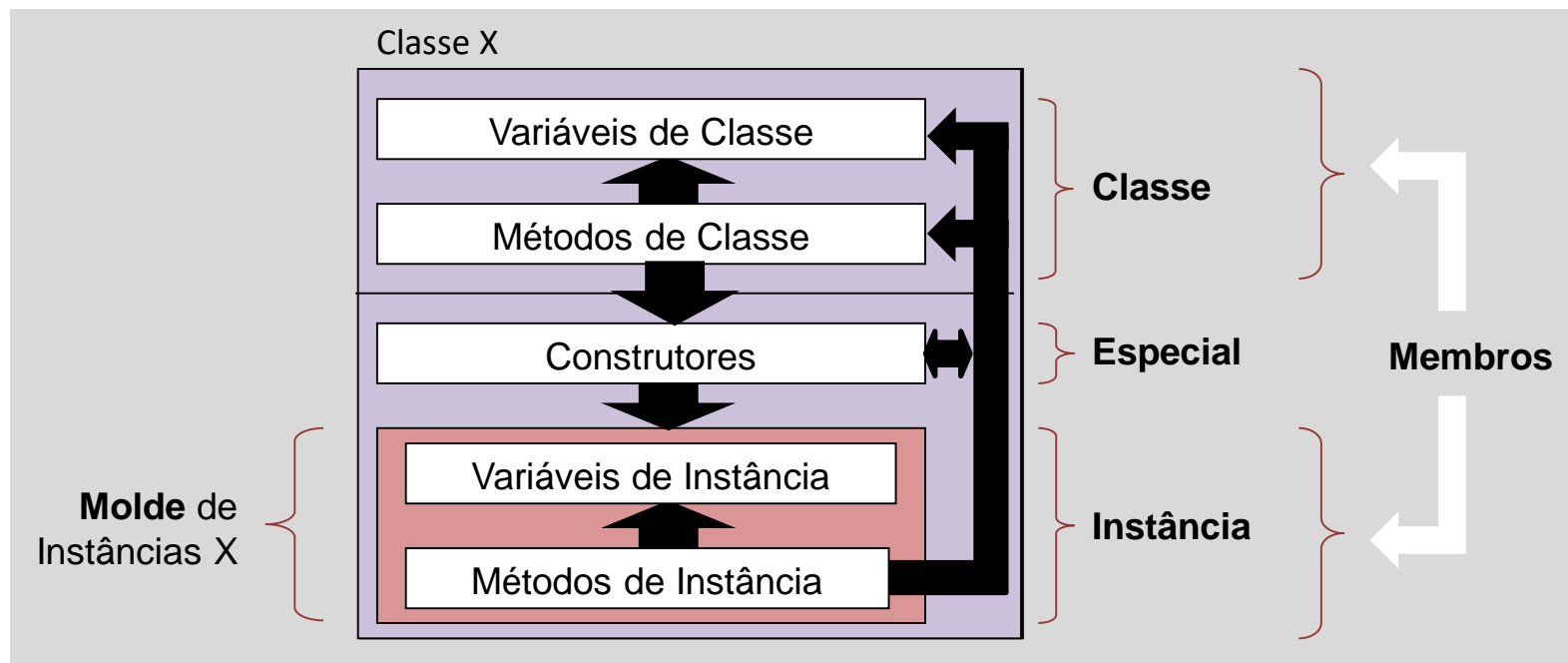


- Interesse

- Definem o Comportamento da classe // operações sobre dados globais da classe

- Têm acesso a

- Variáveis de Classe
 - Construtores



▪ Sintaxe

Cabeçalho
do método



```
[modificador de acesso] [final] [static] [tipo de retorno] nomeMétodo (lista de parâmetros) {  
    //corpo do método  
}
```

- [...] opcional
- **modificador de acesso** public, private, protected ; sem modificador = package
- **final** método não pode ser **reescrito** nas subclasses (será abordado noutra aula sobre Herança)
- **static** **método de classe**
sem static o **método é de instância**
- **tipo de retorno** primitivo ou referência
- **nomeMétodo** letra inicial deve ser **minúscula**
- Sintaxe da **lista de parâmetros** tipo1 nome1, tipo2 nome2, ..., tipoN nomeN

■ Exemplos

```
public class Cliente {  
    ...  
    // métodos de instância  
    public String getNome(){  
        return nome;  
    }  
    public void setMorada(String moradaCliente){  
        morada = moradaCliente;  
    }  
  
    // método de classe  
    public static int getQuantidadeClientes(){  
        return quantidadeClientes;  
    }  
    ...  
}
```

- **Variáveis Locais**

- Declaração
 - Em qualquer local do **corpo** do método
 - Mais perto da sua utilização
- Exemplo

```
public class Matematica {  
    ...  
    // método de classe  
    public static long fatorial(int n){  
        long r=1;                // variável local r  
        for(int i=n; i>0; i--){   // variável local i  
            r = r * i;  
        }  
        return r;  
    }  
    ...  
}
```

- **Assinatura de um método**

- nome(lista de tipos de parâmetros)
- Exemplo

```
public static int m(int x, String s){ ... }
```

- Assinatura

```
m(int, String)
```

- **Mecanismo de sobrecarga de métodos (de instância e de classe)**

- Permite a uma classe declarar **múltiplos métodos com o mesmo nome**, desde que as suas assinaturas sejam diferentes:
 - Em número de parâmetros
 - ou
 - Nos tipos de parâmetros homólogos

- **Exemplos de assinaturas de métodos m sobrecarregados**

```
m()
```

```
m(int)
```

```
m(float, String)
```

```
m(String, int)
```

```
m(int, String, int)
```

- **Tipos de Invocação de Métodos**
 - Mesma classe
 - Classes diferentes
- **Invocação de Métodos da Mesma Classe**
 - Acesso direto
 - Invocação
 - nome_método(lista_argumentos)
 - Exemplo
 - Método m2 da classe Exemplo
 - Invoca o método m1 da mesma classe
 - m1(10)
- **Invocação de Métodos de Outras Classes**
 - Método de Instância
 - Aplica-se a instância da respetiva classe
 - Exemplo
 - Método m2 da classe Exemplo
 - Invoca o método m3 da classe Outra
 - o.m3(10) // o é instância de Outra
 - Método de Classe
 - Aplica-se à respetiva classe
 - Exemplo
 - Outra.m4()

```
public class Exemplo {  
    ...  
    // métodos de instância  
    public String m1(int x){ ... }  
  
    public void m2(String s){  
        String s = m1(10);  
        int i = m4();  
        int r = Outra.m(2);  
  
        Outra o = new Outra();  
        int y = o.m3(10);  
        int z = Outra.m4();  
        ...  
    }  
    // método de classe  
    public static int m4(){ ... }  
}
```

```
public class Outra {  
    ...  
    // método de instância  
    public int m3(int n){ ... }  
  
    // método de classe  
    public static int m4(){ ... }  
    ...  
}
```



- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação

- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
- Referência this
- Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Conicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)

▪ Objetivo

- Suportar **comunicação de mensagens** entre objetos durante execução de programa, pedindo a invocação de métodos (prestação de serviços)

▪ Funcionamento do mecanismo

- **Objeto-emissor** envia uma mensagem a um **objeto-recetor**
- O **objeto-recetor**, após a recepção da mensagem:
 - caso seja possível: executa o método **associado** à mensagem
 - caso não seja possível: gerado um erro de execução



▪ Método executado

- Método cuja assinatura é igual à mensagem(arg1, arg2, ..., argN)
- Exemplo:
 - Mensagem: `obj1.getNome();`
 - Método executado: `getNome()` do objeto `obj1`

Operador ponto (.)

- **Sintaxe da Mensagem**

- Sem retorno // resultado da execução do método correspondente
 - **objeto-recetor.mensagem();**
 - Envia mensagem sem argumentos ao objeto-recetor
 - Exemplo: **obj1.incrementaContador();**
 - **objeto-recetor.mensagem(arg1, arg2, ..., argN);**
 - Envia mensagem com argumentos ao objeto-recetor
 - Exemplo: **obj2.setNome("Sofia");**
- Com retorno
 - **resultado = objeto-recetor.mensagem();**
 - Envia mensagem sem argumentos ao objeto-recetor
 - Exemplo: **String nome = obj3.getNome();**
 - **resultado = objeto-recetor.mensagem(arg1, arg2, ..., argN);**
 - Envia mensagem com argumentos ao objeto-recetor
 - Exemplo: **boolean res = obj3.equalsIgnoreCase("silva");**

- Java Permite Sequência de Mensagens

- Exemplo

- `obj.m1().m2()`

- Objeto-recetor é interpretado no sentido →

- `(obj.m1()) . m2()`

—————→

- A mensagem `m2()` é enviada ao objeto resultante do envio da mensagem `m1()` a `obj`

- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação

- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
- Referência this
- Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Conicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)



- **Referência especial da instância atual**

- Referência da instância recetora da mensagem a solicitar execução de serviço
- Exemplo - this.nome (ver figura)
 - this é **referência da instância** que executa o método setName(...)

- **Apenas pode ser usada dentro de**

- Métodos de instância // pertencem a instâncias
- Construtores // têm acesso a membros de instância

- **Permite indicar só membros de instância**

- **Variáveis de instância**

- Sintaxe: this.nomeVariável
 - Exemplo: this.nome

- **Métodos de instância**

- Sintaxe: this.nomeMétodo(lista de argumentos)
 - Exemplo: this.getNome()

- **Não pode ser usada em métodos de classe**

```
public class Cliente {  
    // variável de instância  
    private String nome;  
    ...  
    // método de instância  
    public void setName(String nome) {  
        this.nome = nome;  
    }  
    public String getNome() {  
        return nome;  
    }  
    ...  
}
```

Referência this (Exemplo)

- Usada principalmente para
 - Resolver conflitos entre identificadores de variáveis de instância e de parâmetros de entrada (ou variáveis locais) com o mesmo nome

- Exemplo

```
public class Cliente {  
    // variável de instância  
    private String morada;  
    ...  
    // método de instância  
    public void setMorada(String morada) { this.morada =  
        morada; }  
    ...  
}
```

Conflito criado pela
necessidade de tornar
o código mais legível

- Notas:

- Há distinção clara entre a variável de instância e o parâmetro de entrada
 - Após a receção da mensagem **obj.setMorada("Rua de ...")**, a instância obj executa o método setMorada(...), considerando **this = obj**
 - **this.morada** lê-se: "morada desta instância"
- Distinguir a invocação de métodos de instância próprios e externos à classe (herdados)

```
public class Exemplo {  
    private void m1() { ... }  
    public void m2() { this.m1(); }  
}
```

Considerada boa
prática de programação

- [Package](#)
 - [Noção](#)
 - [Exemplos JAVA](#)
 - [Declaração](#)
 - [Importação de Classes](#)
- [Mecanismo de Controlo de Acesso \(Visibilidade\)](#)
 - [Classes](#)
 - [Membros da Classe](#)
- [Classe](#)
 - [Declaração](#)
 - [Organização dos Membros](#)
- [Tipos de Dados](#)
 - [Categorias](#)
 - [Primitivos](#)
 - [Referência](#)
- [Variáveis de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
- [Métodos de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
 - [Sobrecarga \(Overloading\)](#)
 - [Invocação](#)

- [Mecanismo de Mensagens](#)
 - [Tipos de Mensagens](#)
 - [Com e Sem Retorno](#)
 - [Sequência de Mensagens](#)
- [Referência this](#)
- [Métodos de Instância](#)
 - [Categorias](#)
 - [Consulta \(Gets\)](#)
 - [Modificadores \(Sets\)](#)
 - [Condicionados](#)
 - [Validação de Dados](#)
 - [Complementares](#)
 - [toString\(\)](#)
 - [Auxiliares](#)
- [Construtores](#)
 - [Declaração](#)
 - [Sobrecarregados](#)
 - [Invocação this\(\)](#)
 - [Construção de Instâncias](#)
- [Classe Principal de um Programa](#)
 - [Estrutura Básica](#)
- [Operador Condicional \(Ternário\)](#)



- **Características Genéricas**
 - Responsáveis pelo **comportamento** das instâncias após pedidos de serviço (receção de mensagens)
 - Públicos
 - Ao contrário das variáveis de instância
 - Privadas
 - Garantir o princípio do encapsulamento dos dados
 - Acesso exterior aos dados das instâncias feito através de métodos próprios da instância
 - Fazem parte da API de uma classe

■ Categorias

■ Consulta ou Interrogadores (Gets)

- Permitirem a consulta exterior dos dados das instâncias
- Públicos

■ Modificadores (Sets)

- Permitem a modificação exterior dos dados das instâncias
- Públicos

■ Complementares

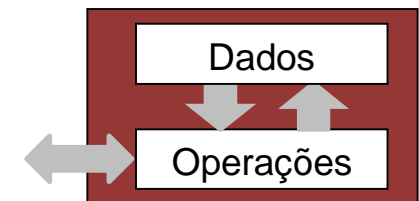
- Completam as responsabilidades dos métodos anteriores
- Exemplo
 - toString()
- Públicos

■ Auxiliares

- Auxiliam a implementação de outros métodos mais complexos, realizando cálculos intermédios, etc.
- Privados

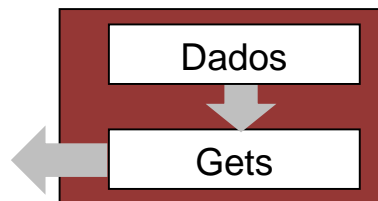
Responsáveis pelo **acesso exterior aos dados** das instâncias de classes.

Para garantir o princípio do encapsulamento



- Métodos de Consulta ou Interrogadores (Gets)

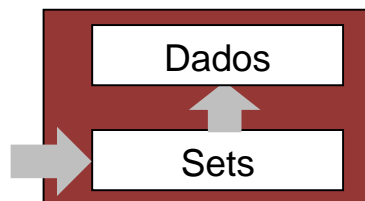
- Nomes começados por get
- Retornam os dados das instâncias



- Sintaxe: `public tipo_retornado getNomeVariávelDeInstância() { ...}`
- Exemplo: `public String getNome() { return nome; }`
- Opcionais
 - Nem sempre se deve definir um método get para cada variável de instância
 - Não faz sentido definir um método get para um dado útil apenas para a instância

- Métodos Modificadores (Sets)

- Nomes começados por set
- Controlam modificações exteriores dos dados das instâncias

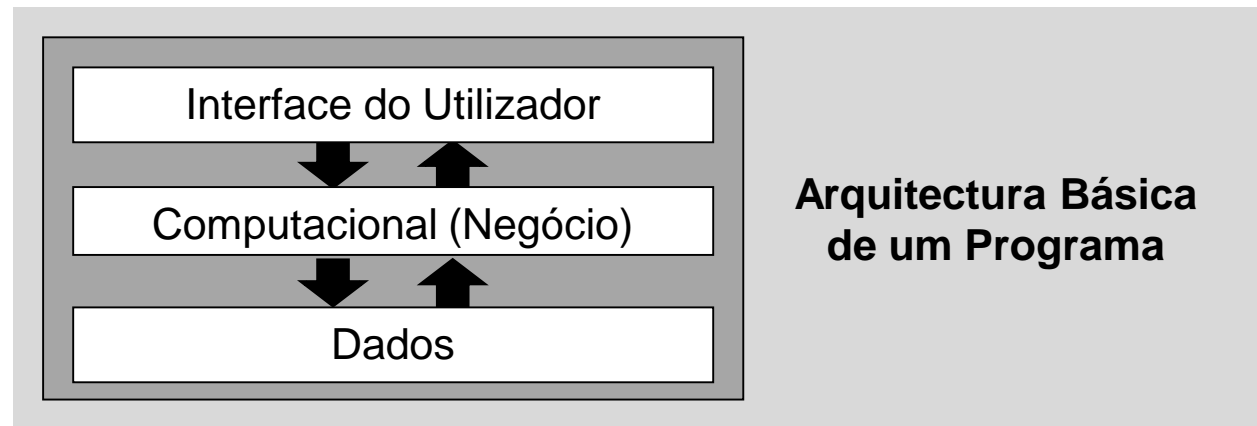


- Sintaxe: `public void setNomeVariávelDeInstância(tipo parâmetro){ ... }`
- Exemplo : `public void setMorada(String morada){ ... }`
- Opcionais
 - Nem sempre, se deve definir um set para cada variável de instância
 - Por exemplo
 - Não faz sentido definir um método `set` para um dado `constante`

- Métodos Modificadores (Set) Condicionados
 - Métodos cujas Execuções
 - Sujeitas a determinadas condições
 - Podem não ter sucesso
 - Exemplo
 - Método setIdade da classe Pessoa
 - Idade ≥ 0

▪ Métodos Modificadores (Set) Condicionados

- Em caso de **insucesso**
 - Objeto-emissor da mensagem deve ser notificado
 - Solução **mais desejável**
 - Lançar uma exceção // abordaremos noutra aula
 - Soluções **indesejáveis**
 - Usar **instruções de input/output** para enviar mensagem para ecrã
 - Viola princípio da separação das camadas // engenharia da programação
 - Computacional (Negócio)
 - Interface do Utilizador



- Misturando as duas camadas
 - Modificações feitas numa podem obrigar a alterações da outra
- Retorno de valor booleano
 - Viola definição de método set // tipo de retorno é void

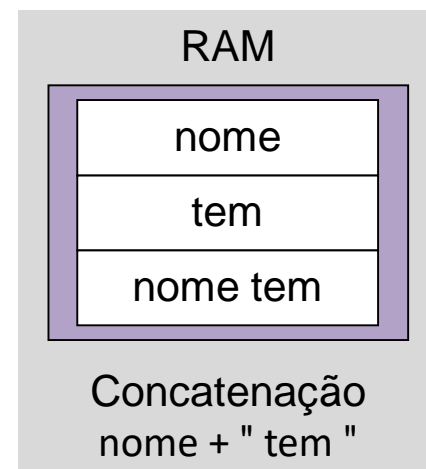
- **Método de instância complementar**
 - Típico de uma classe
- **Funcionalidade**
 - Retorna uma **representação textual legível** da instância sobre a qual é aplicado
 - Baseado na representação textual de cada uma das variáveis de **instância**
- **Interesse**
 - Apresentar a instância no **ecrã**
 - Gravar a instância num **ficheiro de texto**
- **Exemplo**

```
public class Pessoa{  
    // variáveis de instância  
    private String nome;  
    private int idade;  
    ...  
    // método de instância  
    public String toString(){  
        return String.format("%s tem %d anos.", nome, idade);  
    }  
}
```

- **Concatenação de Strings**
 - Operador de concatenação (+) é ineficiente // melhor: String.format() e StringBuilder

▪ Concatenação de strings Java

- Exemplo
nome + " tem " + idade + "anos. "
- É **pouco eficiente**
 - Porque as strings são imutáveis (constantes)
- Cada concatenação de 2 strings
 - Cria uma 3ª string
- Strings grandes
 - Requerem um grande trabalho de alocação de memória
- Pode provocar grandes perdas de desempenho



▪ Classe StringBuilder

- Torna a concatenação de strings mais eficiente
- Exemplo

```
public String toString() {  
    StringBuilder s = new StringBuilder(nome);  
    s.append(" tem ");  
    s.append(idade);  
    s.append(" anos.");  
    return s.toString();  
}
```

▪ Método format da classe String

- Usa esta classe StringBuilder

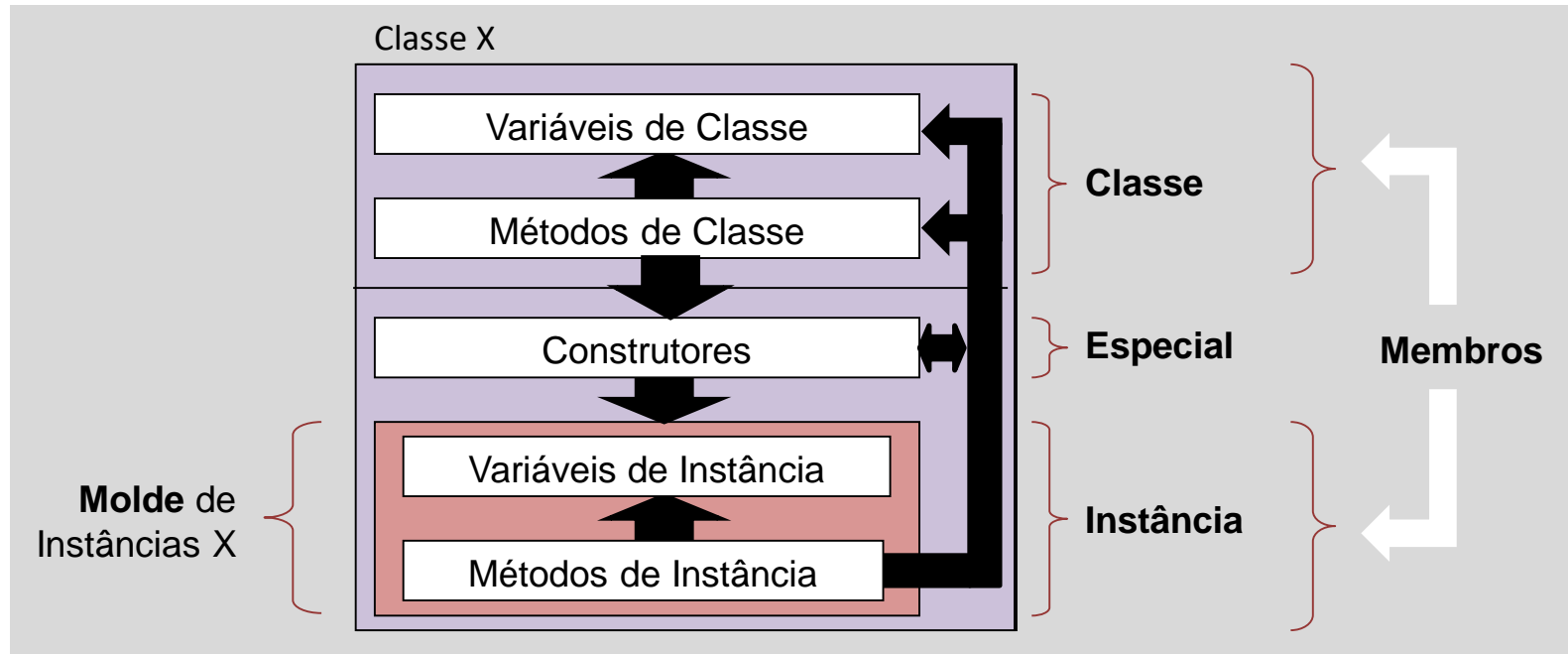
- Package
 - Noção
 - Exemplos JAVA
 - Declaração
 - Importação de Classes
- Mecanismo de Controlo de Acesso (Visibilidade)
 - Classes
 - Membros da Classe
- Classe
 - Declaração
 - Organização dos Membros
- Tipos de Dados
 - Categorias
 - Primitivos
 - Referência
- Variáveis de Instância e de Classe
 - Interesse
 - Declaração
- Métodos de Instância e de Classe
 - Interesse
 - Declaração
 - Sobrecarga (Overloading)
 - Invocação

- Mecanismo de Mensagens
 - Tipos de Mensagens
 - Com e Sem Retorno
 - Sequência de Mensagens
- Referência this
- Métodos de Instância
 - Categorias
 - Consulta (Gets)
 - Modificadores (Sets)
 - Condicionados
 - Validação de Dados
 - Complementares
 - toString()
 - Auxiliares
- Construtores
 - Declaração
 - Sobrecarregados
 - Invocação this()
 - Construção de Instâncias
- Classe Principal de um Programa
 - Estrutura Básica
- Operador Condicional (Ternário)



▪ Noção

- Membro especial de uma classe
- Permite
 - Criar instâncias de classes // reproduzindo as variáveis e métodos de instância
 - Inicializar o estado das instâncias // variáveis de instância



▪ Sintaxe

```
[modificador de acesso] nomeClasse(lista de parâmetros){ ... }
```

- **Modificador de acesso** opcional
public (em **geral**)
 - Permite criar instâncias de classes
 - Garante classe instanciável
 - Faz parte da API de uma classe instanciávelprivate
 - Não permite criar instâncias
 - Ex: classe Math
- **nomeClasse** obrigatorio = nome da respetiva classe
- **parâmetros** para receberem **dados iniciais** das instâncias criadas

▪ Nota

- Não faz sentido declarar o **tipo de retorno** de um construtor
 - Construtor só serve para
 - Criar instâncias
 - Inicializar os seus estados
 - Nunca devolverá um resultado da sua execução

- Exemplo

```
public class Pessoa{  
    // variáveis de instância  
  
    private String nome;  
    private int idade;  
  
    // construtor  
  
    public Pessoa(String nome,int idade){  
        this.nome = nome;  
        this.idade = idade;  
    }  
  
    ...  
}
```

- Estado tem de ser consistente

- Com a entidade real/conceito representado pelas instâncias
- Exemplo das instâncias da classe Pessoa
 - idade ≥ 0

- Construtor quando não recebe um dado inicial?

- Inicializa dado com valor por omissão
- Variáveis de instância do tipo Referência (classe)
 - Inicializadas com instâncias do tipo da variável
 - Em geral, estas instâncias são criadas pelos respetivos construtores sem parâmetros
 - Senão, são inicializadas a null \Rightarrow Estado inconsistente

```
public class Exemplo {  
    private Data data;          // variável de instância do tipo Data (referência)  
    // construtores  
    public Exemplo(){ data = new Data(); } // atribuída data válida;  
                                           // validada pela classe data  
}
```

- Variáveis de instâncias do tipo String

- Inicializadas com uma string ; p.ex: "sem nome"
- Senão, são inicializadas a null

- **Mecanismo de Sobrecarga (Overloading)**
 - Permite declarar **múltiplos construtores** com assinaturas diferentes
 - Listas de parâmetros diferentes
 - Em número de parâmetros
 - E/ou tipo de parâmetros

- **Exemplo**

```
public class Pessoa{  
    // variáveis de instância  
    private String nome;  
    private int idade;  
  
    // construtores  
    public Pessoa(String nome,int idade){           // construtor completo  
        this.nome = nome;  
        this.idade = idade;  
    }  
    public Pessoa(String nome){                     // idade inicializada por omissão  
        this(nome,0);  
    }  
    public Pessoa(Pessoa p){                        // construtor de cópia (ou clone)  
        this(p.getNome(),p.getIdade());  
    }  
    ...  
}
```

- Interesse

- Inicializar os estados das instâncias de diferentes maneiras

- Tipos de Construtores

- Sem parâmetros // não permite ao código cliente da classe inicializar o Estado da instância
// criado automaticamente, se a classe não declarar qualquer construtor
 - Com parâmetros // Estado das instâncias é inicializado com os parâmetros de entrada,
// definidos pelo código cliente

▪ Sintaxe

```
new nomeConstrutor(lista_parâmetros)    // operador new retorna referência de instância
```

▪ Exemplos

```
public class TestePessoa {  
    public static void main(String[] args){  
        Pessoa p1 = new Pessoa();           // criada instância de Pessoa  
                                              // Estado da instância inicializado com dados por omissão  
                                              // referência da instância guardada em p1 do tipo Pessoa  
  
        Pessoa p2 = new Pessoa( "Nico", 24 ); // cria instância de Pessoa  
                                              // Estado inicializado com dados passados por parâmetro  
                                              // referência dessa instância é guardada em p  
  
        Pessoa p3;                          // declara uma variável p3 do tipo Pessoa  
                                              // variável inicializada a null  
  
        p3 = new Pessoa( "Rita", 12 );      // criada instância de Pessoa  
                                              // Estado inicializado com dados passados por parâmetro  
                                              // referência dessa instância é guardada em p3  
    }  
}
```

▪ Sintaxe

```
this(lista de parâmetros);
```

▪ Invoca

- Um construtor dentro de outro construtor, da mesma classe
- Com o **mesmo número** de parâmetros e o **mesmo tipo** de parâmetros **homólogos**

▪ Apenas pode ser usada em construtores

- Tem de ser a 1ª instrução, **obrigatoriamente**

▪ Interesse

- Simplifica a programação de construtores

▪ Exemplo

```
public class Pessoa{  
    // variáveis de instância  
    private String nome;  
    private int idade;  
  
    // construtores  
    public Pessoa(String nome,int idade){  
        this.nome = nome;  
        this.idade = idade;  
    }  
    public Pessoa(String nome){ this(nome,0); }  
    public Pessoa(Pessoa p){ this(p.getNome(),p.getIdade()); }  
}
```

- [Package](#)
 - [Noção](#)
 - [Exemplos JAVA](#)
 - [Declaração](#)
 - [Importação de Classes](#)
- [Mecanismo de Controlo de Acesso \(Visibilidade\)](#)
 - [Classes](#)
 - [Membros da Classe](#)
- [Classe](#)
 - [Declaração](#)
 - [Organização dos Membros](#)
- [Tipos de Dados](#)
 - [Categorias](#)
 - [Primitivos](#)
 - [Referência](#)
- [Variáveis de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
- [Métodos de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
 - [Sobrecarga \(Overloading\)](#)
 - [Invocação](#)

- [Mecanismo de Mensagens](#)
 - [Tipos de Mensagens](#)
 - [Com e Sem Retorno](#)
 - [Sequência de Mensagens](#)
- [Referência this](#)
- [Métodos de Instância](#)
 - [Categorias](#)
 - [Consulta \(Gets\)](#)
 - [Modificadores \(Sets\)](#)
 - [Conicionados](#)
 - [Validação de Dados](#)
 - [Complementares](#)
 - [toString\(\)](#)
 - [Auxiliares](#)
- [Construtores](#)
 - [Declaração](#)
 - [Sobrecarregados](#)
 - [Invocação this\(\)](#)
 - [Construção de Instâncias](#)
- [Classe Principal de um Programa](#)
 - [Estrutura Básica](#)
- [Operador Condicional \(Ternário\)](#)



- Classe que contém o método principal do programa

- Declaração

```
public static void main(String[] args){  
    // corpo do método  
}
```

parâmetro **args** é obrigatório

- Primeiro método do programa executado pelo interpretador de JAVA

- Pode ser usada para testar novas classes (exemplo: Pessoa):

```
public class TestePessoa {  
    public static void main(String[] args){  
        Pessoa p;                                // declara uma variável p do tipo Pessoa  
        p = new Pessoa( "Rita", 12 );            // cria uma instância de Pessoa e inicializa o seu Estado  
                                                // referência dessa instância é guardada em p  
        System.out.println( "Nome" + p.getNome() ); // apresenta o nome da instância p  
                                                // envia mensagem getNome() à instância p  
        System.out.println( "Idade:" + p.getIdade() ); // apresenta a idade da instância p  
                                                // envia mensagem getIdade() à instância p  
        p.setIdade( 11 );                        // altera a idade da instância p para 11  
        System.out.println( p.toString() );      // apresenta a instância p  
        System.out.println( p );                 // equivalente à instrução anterior  
    }  
}
```


- [Package](#)
 - [Noção](#)
 - [Exemplos JAVA](#)
 - [Declaração](#)
 - [Importação de Classes](#)
- [Mecanismo de Controlo de Acesso \(Visibilidade\)](#)
 - [Classes](#)
 - [Membros da Classe](#)
- [Classe](#)
 - [Declaração](#)
 - [Organização dos Membros](#)
- [Tipos de Dados](#)
 - [Categorias](#)
 - [Primitivos](#)
 - [Referência](#)
- [Variáveis de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
- [Métodos de Instância e de Classe](#)
 - [Interesse](#)
 - [Declaração](#)
 - [Sobrecarga \(Overloading\)](#)
 - [Invocação](#)

- [Mecanismo de Mensagens](#)
 - [Tipos de Mensagens](#)
 - [Com e Sem Retorno](#)
 - [Sequência de Mensagens](#)
- [Referência this](#)
- [Métodos de Instância](#)
 - [Categorias](#)
 - [Consulta \(Gets\)](#)
 - [Modificadores \(Sets\)](#)
 - [Condicionados](#)
 - [Validação de Dados](#)
 - [Complementares](#)
 - [toString\(\)](#)
 - [Auxiliares](#)
- [Construtores](#)
 - [Declaração](#)
 - [Sobrecarregados](#)
 - [Invocação this\(\)](#)
 - [Construção de Instâncias](#)
- [Classe Principal de um Programa](#)
 - [Estrutura Básica](#)
- [Operador Condicional \(Ternário\)](#)



- **Sintaxe**

```
condição ? valor_1 : valor_2 // tem 3 operandos
```

- **Semântica**

Se condição = verdadeiro então
 retorna valor_1
senão
 retorna valor_2

- **Exemplo**

```
public class Circulo {  
    // variável de instância  
    private float raio;  
    ...  
    // método de instância  
    public void setRaio( float raio ){  
        this.raio = raio > 0 ? raio : 1 ;  
    }  
    ...  
}
```