



PROGRAMAÇÃO ORIENTADA A OBJETOS [POO]

Material 02 – POO_02

Prof. Mestre Marcos Roberto de Moraes [Maromo]

Introdução à linguagem Java

Introdução a Linguagem Java

- ❑ Histórico
- ❑ Características da Linguagem
- ❑ Plataforma Java
- ❑ Por que usar Java ?
- ❑ Exemplos
- ❑ Orientação a Objetos em Java
- ❑ Elementos Básicos da Linguagem
- ❑ Exercícios (Laboratório)



Histórico

- ❑ Projeto Green da Sun Microsystems (1991)
 - ▣ Patrick Naughton, Mike Sheridan e James Gosling.
- ❑ Primeira demonstração do projeto (1992)
 - ▣ Sistema em handheld com interface a touchscreen [Star seven].
- ❑ Lançamento da Plataforma Java (1995)
 - ▣ John Gage (Sun) e Marc Andreessen (Netscape), anunciaram a JVM e a API Java. Inserido no Netscape Navigator.

Histórico



John Gage

http://upload.wikimedia.org/wikipedia/commons/d/de/John_Gage.jpg



Marc Andreessen

http://farm4.static.flickr.com/3285/2960434767_4014c753f9_o.jpg



Patrick Naughton

http://s3.amazonaws.com/quarkbase_test.com/patrick-naughton-41297.jpg



James Gosling

<http://www.javaalone.com/wp-content/uploads/2010/06/james-gosling.jpg>

Características da Linguagem

- ❑ Simples
- ❑ Orientada a Objetos
- ❑ Multithread
- ❑ Interpretada
- ❑ Neutra de Arquitetura
- ❑ Portável
- ❑ Robusta
- ❑ Segura
- ❑ Alto desempenho

Simples

- ❑ Permite desenvolvimento de software em diferentes SOs e arquitetura de hardware. [Programador não se preocupa com a infraestrutura]
- ❑ Criadores de Java optaram por não implementar o uso do conceito de herança múltipla, de sobrecarga de operadores, ponteiros nem a operação aritmética com esse tipo de dado. [MENDES, 2009]

Orientada a Objetos

- Linguagem criada seguindo o paradigma da orientação a objetos (já visto).

Multithread

- ❑ Java permite a criação de programa que implementam o conceito multithread.
- ❑ Mecanismos de sincronização entre processos.
- ❑ Multithreading = Técnica de programação concorrente.

Interpretada

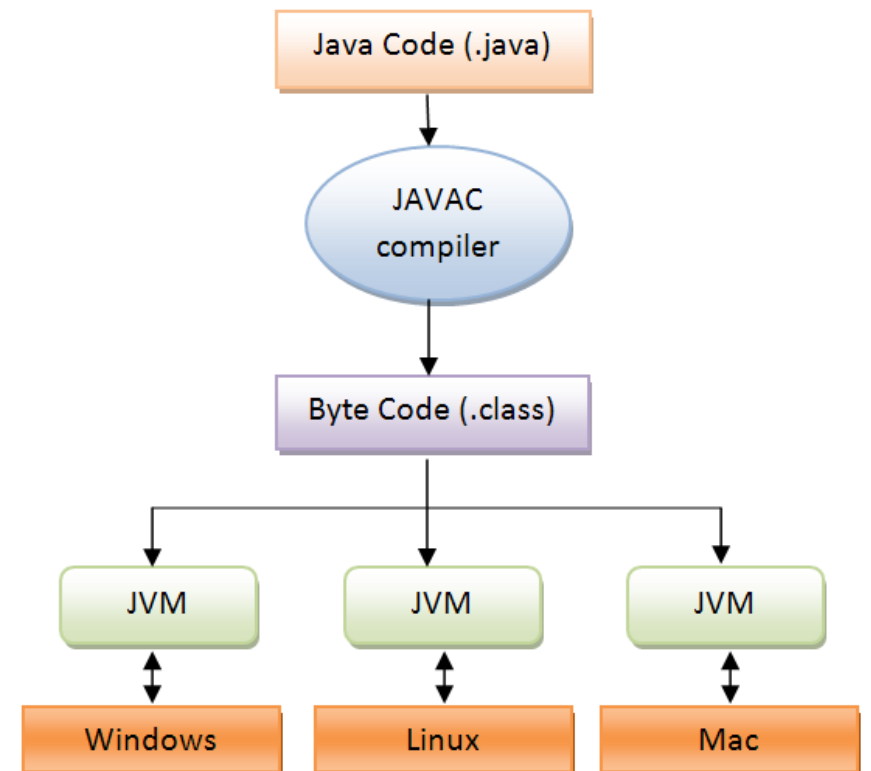
- Após compilação é gerado um arquivo intermediário (bytecode).
- Vantagem:
 - ▣ Este código poderá ser executado em qualquer arquitetura (Windows, Linux, Mac e Unix) que tenha uma máquina virtual Java instalada (JVM).

Independência de Arquitetura

- Projetada para suportar sistemas que serão implementado em plataformas heterogêneas (HW / SW)
 - ▣ Unix, Linux, Mainframe.

Portabilidade

- JVM – Java Virtual Machine – garante a portabilidade dos programas.
- Especificação na qual o compilador Java de cada plataforma irá se basear para gerar o código em bytecode.



Alto Desempenho

- ❑ Executa um código que foi previamente analisado e convertido para bytecodes.
- ❑ Recursos de garbage collector (executado em segundo plano).

Robusta

- ❑ Possui sistema de tratamento de exceções – (indicação ao programa que ocorreu um problema durante a sua execução).
- ❑ Estes recursos tornam os programas mais robustos e tolerantes a falhas. (DEITEL, 2005)

Segura

- ❑ Pode-se garantir que, em um ambiente de rede, nenhum programa Java permitirá que outro programa escrito em qualquer linguagem possa se esconder em um código Java a fim de se instalar automaticamente (MENDES, 2009).



Ambiente Java

Conhecendo o Ambiente Java

Ambientes de desenvolvimento Java

- ❑ JSE (Java Standard Edition)
 - ▣ Seu uso é voltado a PCs e servidores.
 - ▣ Contem todo o ambiente necessário para a criação e execução de aplicações desktop e web de pequeno e médio porte.
 - ▣ Pode-se dizer que essa é a plataforma principal, já que, o JEE e o JME tem sua base aqui.

Ambientes de desenvolvimento Java

- ❑ JEE (Java Enterprise Edition)

- ▣ Voltada para o desenvolvimento de softwares corporativos.
- ▣ Baseados em componentes que são executados em um servidor de aplicação.

- ❑ JME (Java Micro Edition)

- ▣ Ambiente de desenvolvimento para dispositivos móveis ou portáteis, como telefones celulares e palmtops.

Componentes básicos da linguagem Java

- ❑ JRE (Java Runtime Environment)
 - ▣ Significa Ambiente de Tempo de Execução
 - ▣ É um pacote de softwares, que é executado como um aplicativo do sistema operacional e que interpreta a execução de programas Java
 - ▣ A JRE é composta pela JVM somada ao conjunto de API's. (JVM + API's = JRE)

Componentes básicos da linguagem Java

- API (Application Programming Interface)
 - ▣ Significa Interface de Programação de Aplicativos
 - ▣ Biblioteca (ou uma série delas) com funções e procedimentos públicos que permitem aos programadores desenvolverem aplicações fazendo uso de recursos já definidos.

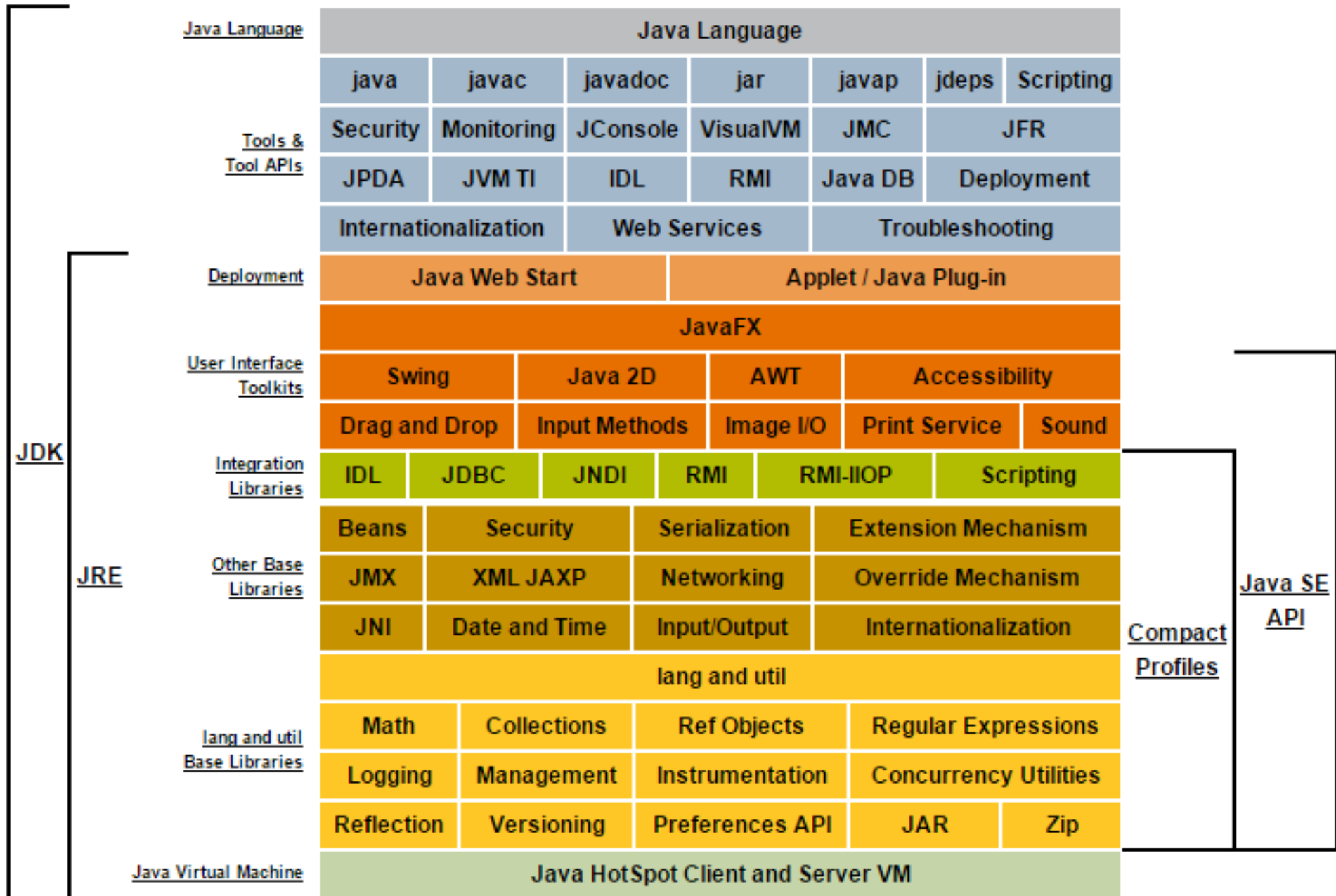
Componentes básicos da linguagem Java

- JVM (Java Virtual Machine)
 - ▣ Significa Máquina Virtual Java
 - ▣ Software que emula uma CPU e Memória para a execução de programas Java.

Componentes básicos da linguagem Java

- ❑ JDK (Java Development Kit) ou SDK (Software Development Kit)
 - ▣ Significa Kit de Desenvolvimento Java
 - ▣ Conjunto de ferramentas para a compilação, documentação e debug de aplicativos Java.
 - ▣ Composto pela JRE somada as ferramentas de desenvolvimento.

Versão 8
A versão 11 não integra algumas bibliotecas que não faziam parte do core



Tipos de programas em java

□ **Stand-Alone:**

- ▣ Aplicação baseada na JSE, que tem total acesso aos recursos do sistema (memória, disco, rede, dispositivos, etc)
- ▣ Um servidor pode executar uma aplicação Stand-Alone, por exemplo, um WebServer.
- ▣ Uma estação de trabalho pode executar uma aplicação de Automação Comercial.

Tipos de programas em java

□ **Java Applets:**

- ▣ Pequenas aplicações, que não tem acesso aos recursos de hardware e depende de um navegador que suporte a JSE para serem executados, geralmente usados para jogos, animações, teclados virtuais, etc.

□ **Java Servlets:**

- ▣ Programas escritos e preparados para serem executados dentro de servidores web baseados em JEE, geralmente usados para gerar conteúdo dinâmico de websites.

Tipos de programas em java

□ **Java Midlets:**

- ▣ Pequenas aplicações, extremamente seguras, e construídas para serem executadas dentro da JME, geralmente, celulares, Palm Tops, controladores eletrônicos, computadores de bordo, smart cards, tecnologia embarcada em veículos, etc.

□ **JavaBeans:**

- ▣ Pequenos programas, que seguem um padrão bem rígido de codificação, e que tem o propósito de serem aproveitados em qualquer tipo de programa Java, sendo reaproveitados, e podendo ser chamados a partir de: stand-alone, applets, servlets e midlets.

Equívocos em relação ao Java

- ❑ **É uma extensão do HTML**
 - ▣ Falso, o Java é uma linguagem completa derivada do SmallTalk e do C++;
- ❑ **O JavaScript é uma versão light do Java**
 - ▣ Falso, a Netscape aproveitou a onda de marketing e batizou sua tecnologia, LiveScript, de JavaScript;
- ❑ **É interpretado, muito lento para aplicações robustas**
 - ▣ O Java é interpretado sim, entretanto, a forma como a dupla compilador/interpretador tratam os programas garante uma performance muitas vezes equivalente ao do C++, com a facilidade de uma linguagem bem mais simples que o C++;
- ❑ **É difícil programar em Java**
 - ▣ Falso, a maior dificuldade está em assimilar os conceitos da Orientação a Objetos. A linguagem Java é muito simples;

Por que usar Java ?

Aug 2020	Aug 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	16.98%	+1.83%
2	1	▼	Java	14.43%	-1.60%
3	3		Python	9.69%	-0.33%
4	4		C++	6.84%	+0.78%
5	5		C#	4.68%	+0.83%

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

Linguagens mais populares do mundo.

Exemplo de Programa Java

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Parametro: " + args[0]);  
        System.out.println("Parametro: " + args[0]);  
        System.out.println("Ola, recebi dois argumentos");  
    }  
}
```

Comentário do Exemplo Anterior

- ❑ Linha 1: Representa o nome do pacote (Contêiner da aplicação)
- ❑ Linha 2: Comentário `//` ou `/* */`
- ❑ Linha 3: Classe Main
- ❑ Linha 4: Método principal main
- ❑ Linhas 5 a 7: Comando de Saída
- ❑ Não esqueça de passar os parâmetros durante a execução.

Referências

□ Bibliográficas:

- ▣ Mendes – Java com Ênfase em Orientação a Objetos
- ▣ Deitel – Java, como programar – 6º edição.
- ▣ Arnold, Gosling, Holmes – A linguagem de programação Java – 4º edição.
- ▣ Apostilas Caelum

□ Internet

- ▣ <http://java.sun.com>
- ▣ <http://www.guj.com.br>
- ▣ <http://www.portaljava.com>

Orientação a Objetos ênfase em Java

Java e OO

Orientação a Objetos

- ❑ Técnica de desenvolvimento de softwares que consiste em representar os elementos do mundo real (que pertencem ao escopo da aplicação) dentro do software.
- ❑ Em tese, é **uma forma mais natural de informatização**, já que leva em consideração os elementos como eles realmente existem.
- ❑ Vale ressaltar que a orientação a objetos não é exclusividade da linguagem Java. Outras linguagem C#, VB.net, PHP 5, entre outras, fazem uso de tais recursos.

Elementos básicos da programação orientada a objetos

□ **Classe**

- ▣ Na verdade, na linguagem Java, tudo está definido em classes.
- ▣ A estrutura da própria linguagem é organizada em classes.
- ▣ Porém, inicialmente vamos nos ater às duas representações básicas de classes em orientação a objetos:
 - A classe de modelagem (ou definição de tipo).
 - A classe que contém a inicialização da aplicação.

Elementos básicos da programação orientada a objetos

▣ Classe de modelagem

- Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.
- A classe de modelagem pode ser entendida como um molde, uma forma, um modelo que define as características e as funcionalidades dos futuros objetos que serão criados a partir dela.

```
Em Java:  
public class Carro{  
  
}
```

Elementos básicos da programação orientada a objetos

□ Objeto

- Em modelagem orientada a objetos, um objeto é uma instância de uma classe existente no sistema de software.
- Um objeto representa uma entidade do mundo real dentro da aplicação de forma individual, possuindo todas as informações e funcionalidades abstraídas na concepção da classe.

Em Java:

```
Carro c1 = new Carro();
```

Analógia

Classe X Objeto



Estrutura básica e uma classe de modelagem

□ **Atributos ou variáveis de instância**

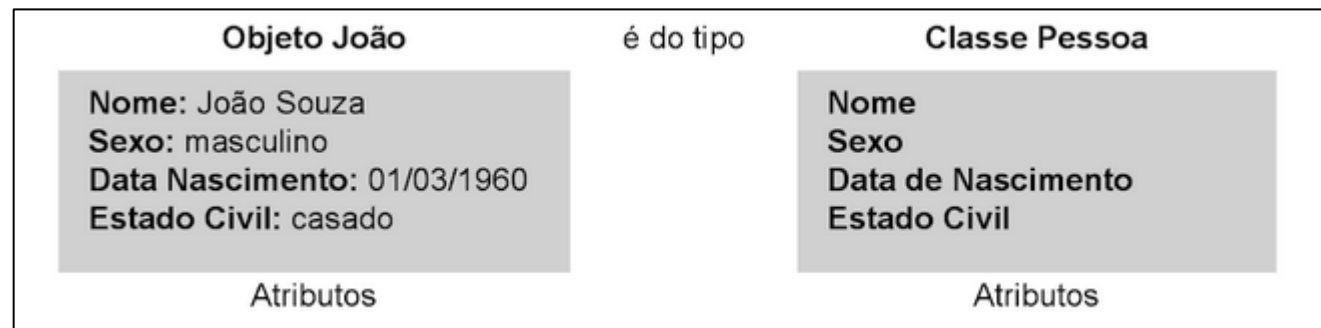
- ▣ São as informações, os dados, que serão armazenados nos objetos.

□ **Métodos**

- ▣ São as ações, as regras , as funcionalidades que serão executadas pelos objetos.

Estrutura básica e uma classe de modelagem

- ❑ **Classe de modelagem (definição de tipo)**
 - ▣ Atributos (dados)
 - ▣ Métodos (ações)
- ❑ **Objetos (instâncias)**



Fonte: Melo, A, C; 2010.

UML

- A partir do momento em que os elementos básicos da orientação a objetos são assimilados, podemos modelar classes nas especificações corretas utilizando a principal ferramenta de modelagem e documentação de aplicações orientadas a objeto existente no mercado: A UML (Unified Modeling Language ou Linguagem unificada de modelagem).

UML

- ❑ A UML não é uma metodologia de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como projetar seu sistema, mas ela lhe auxilia a visualizar seu desenho e a comunicação entre objetos.
- ❑ Permite que desenvolvedores visualizem os produtos de seu trabalho em diagramas padronizados.
- ❑ Os objetivos da UML são: especificação, documentação, e estruturação para sub-visualização e maior visualização lógica de um total desenvolvimento de um sistema de informação.

Classe de modelagem:

Usuario

- Abstração e modelagem da classe de usuário de uma aplicação.
 - ▣ Informações armazenadas em cada usuário (objeto).
 - nome
 - email
 - login
 - senha
 - ▣ Ações exercidas pelos (objetos do tipo) usuários na aplicação.
 - provar existência.

Classe de modelagem:

Usuario

- **Nome da classe**

- ▣ Usuario

- **Atributos**

- ▣ nome, email, login e senha

- **Métodos**

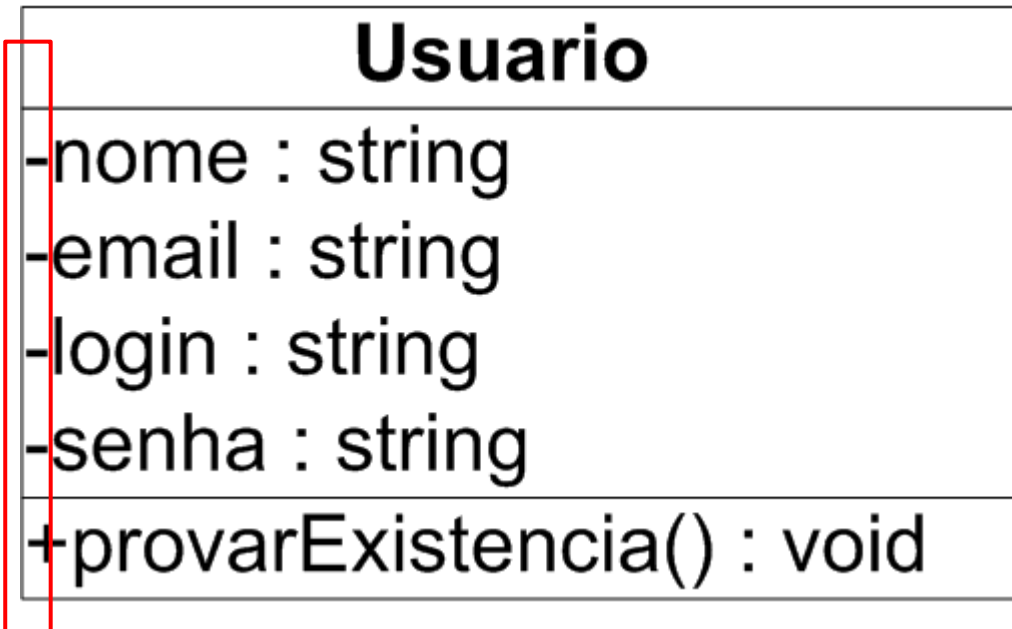
- ▣ provarExistencia

Classe de modelagem:

Usuario

□ Em UML:

Visibilidade
dos atributos e
dos métodos.



Visibilidade

- ▣ Private [-]

- Só fica visível dentro da classe em que foi implementado

- ▣ Public [+]

- Fica visível em toda a aplicação

- ▣ Protected [#]

- O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança

Codificação da classe

Usuario

```
/**...*/
```

```
public class Usuario {
```

```
//Atributos
```

```
public String nome;
```

```
public String email;
```

```
public String login;
```

```
public String senha;
```

Atributos

```
//Construtores
```

```
//Inicializa objeto com valores vazios
```

```
public Usuario () {
```

```
}
```

```
//Inicializa atributos com valores passados pelos parâmetros
```

```
public Usuario (String nome, String email, String login, String senha) {
```

```
    this.nome = nome;
```

```
    this.email = email;
```

```
    this.login = login;
```

```
    this.senha = senha;
```

```
}
```

```
}
```

Construtores

Encapsulamento

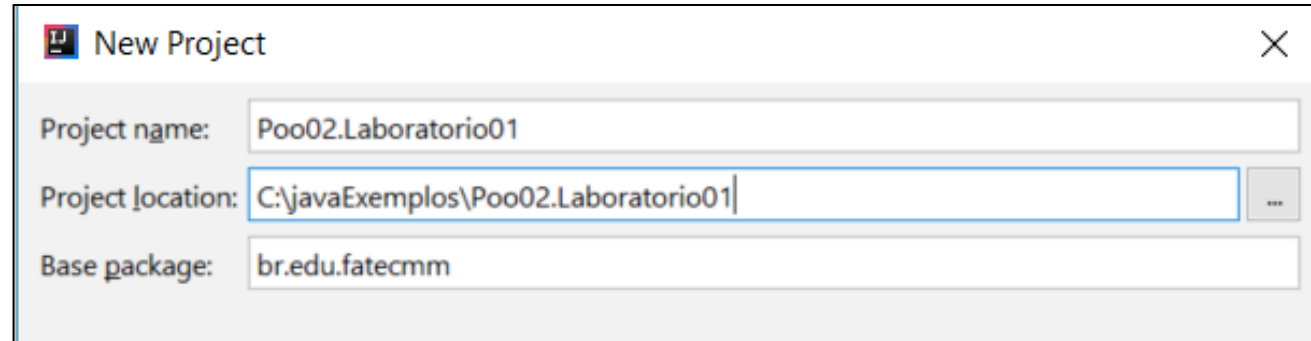
- ❑ **Consiste na proteção dos atributos de uma classe** (e posteriormente dos objetos) de acessos externos.
- ❑ Considerando que todas as regras referentes a classe estão contidas na própria classe (e nunca em outra parte da aplicação), o acesso aos atributos deverão ser feitos de modo a garantir que tais regras sejam cumpridas.

Prover acesso a campo protegido

- ❑ Métodos getters e setters.
- ❑ Cada atributo tem seus próprios métodos públicos getter e setter.
 - ▣ Getter: Lê o conteúdo de um atributo e retorna seu valor.
 - ▣ Setter: Recebe um valor por parâmetro e altera (escreve) tal valor no referente atributo.

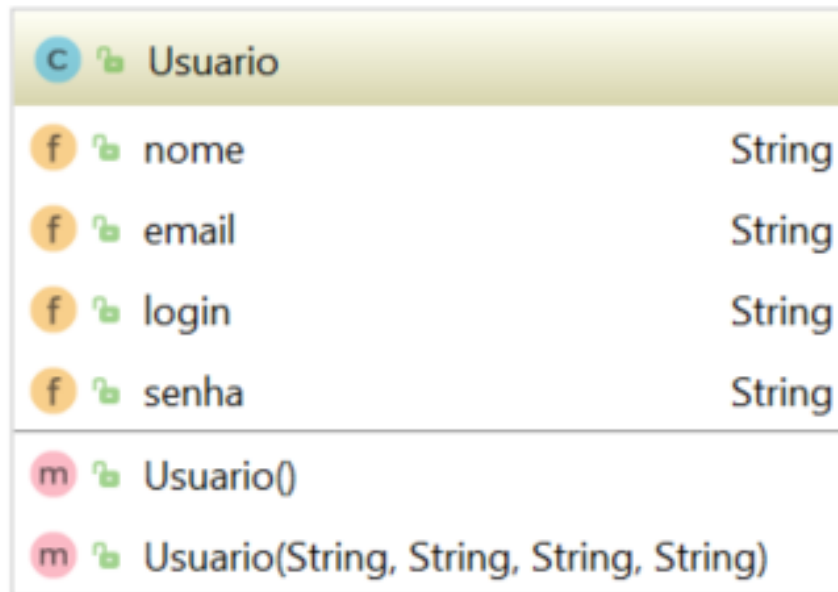
Projeto: **Poo02.Laboratorio01**

- ❑ Na IntelliJ IDEA modele e execute o exemplo a seguir.



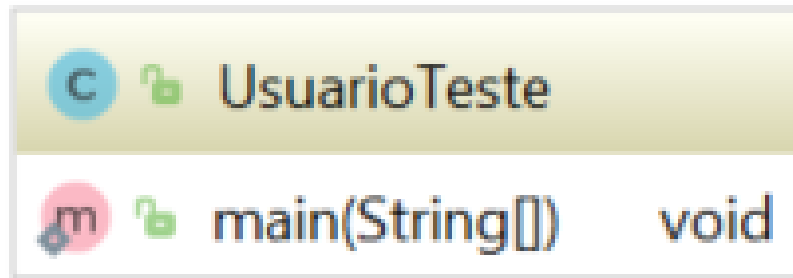
Classes Usuario

- ❑ Modelo a classe usuário como segue:



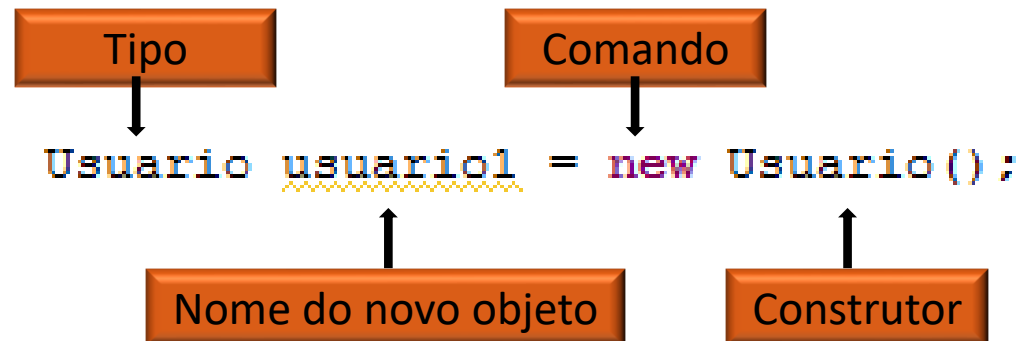
Classe: Main (Renomear >> UsuarioTeste)

- Neste exemplo, a classe **Main** será renomeada para **UsuarioTeste**, nela teremos o método main.



Como instanciar um objeto

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



Como realizar uma chamada a método








```
public static void main(String[] args) {  
  
    Usuario usuario1 = new Usuario();  
  
    usuario1.provarExistencia();  
  
}
```

Nome do novo objeto

Método

Codificação da classe Usuario

```
1 package br.edu.fatecmm;
2
3 public class Usuario {
4     //Atributos
5     public String nome;
6     public String email;
7     public String login;
8     public String senha;
9
10    //Construtores
11    public Usuario() {
12
13    }
14
15    public Usuario(String nome, String email, String login, String senha) {
16        this.nome = nome;
17        this.email = email;
18        this.login = login;
19        this.senha = senha;
20    }
21 }
```

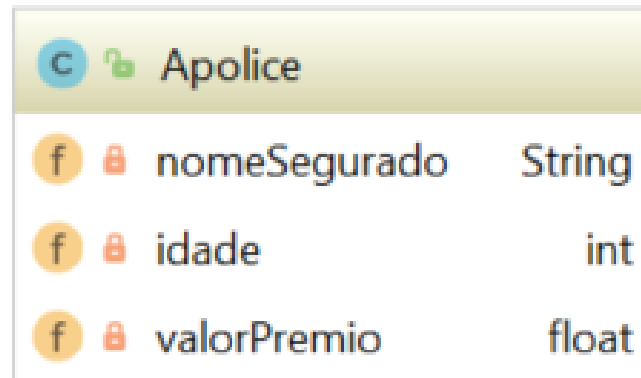
c  Usuario		
f 	nome	String
f 	email	String
f 	login	String
f 	senha	String
<hr/>		
m 	Usuario()	
m 	Usuario(String, String, String, String)	

Estrutura básica de um projeto Java

- **Projeto (Estrutura de pastas e arquivos que compõem a aplicação)**
 - ***Classe de modelagem (Definição de tipo)***
 - Atributos
 - Construtores
 - **Getters e setters (Veja como refatorar)**
 - Métodos específicos da classe
 - ***Classe de entrada do programa***
 - Contém o Método main()

Exercício 1

- ❑ A) Crie um projeto chamado prjApolice
- ❑ B) Defina uma classe Java chamada Apolice com os seguintes atributos: nomeSegurado, idade e valorPremio. Visibilidade dos atributos deve estar definida como private.



A screenshot of an IDE window showing the class `Apolice`. The class has three private attributes: `nomeSegurado` of type `String`, `idade` of type `int`, and `valorPremio` of type `float`. Each attribute is preceded by a yellow circle with the letter 'f' and a red padlock icon, indicating its visibility and type.

Apolice		
f	nomeSegurado	String
f	idade	int
f	valorPremio	float

Exercício 1

- C) Crie o seguintes método na Classe Apólice

Métodos	Descrição
imprimir()	Este método não retornar valor e deverá mostrar na tela todos os atributos da Classe Apólice

c	Apolice	
f	nomeSegurado	String
f	idade	int
f	valorPremio	float
m	imprimir()	void

Exercício 1

- E) Crie uma segunda classe chamada **ApoliceTeste** com a seguinte estrutura:
 - ▣ Criar método *main()* conforme o padrão da linguagem Java. Neste método, criar um objeto da classe **Apolice** usando o comando: **Apolice apolice = new Apolice();**. Para cada atributo da classe atribuir um valor corrente.
 - ▣ Executar o método *imprimir()* e analisar o que será impresso na tela.
 - ▣ Com a ajuda do professor, refatore a classe *Apolice* encapsulando todos os campos e criando os modificadores de acesso.

Referências

□ Bibliográficas:

- ▣ Mendes – Java com Ênfase em Orientação a Objetos
- ▣ Deitel – Java, como programar – 6º edição.
- ▣ Arnold, Gosling, Holmes – A linguagem de programação Java – 4º edição.
- ▣ Apostilas Caelum

□ Internet

- ▣ <http://java.sun.com>
- ▣ <http://www.guj.com.br>
- ▣ <http://www.portaljava.com>

FIM

Maromo