LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS

ETEC DE HORTOLÂNDIA

CURSO TÉCNICO EM INFORMÁTICA INTEGRADO AO ENSINO MÉDIO

PROF. RALFE DELLA CROCE FILHO

CONTEÚDO

- Classe (de modelagem) e Objeto
- Classe de modelagem (estrutura)
  - Atributo
  - Método
  - Método construtor
- Encapsulamento
- Eclipse (recursos básicos)
- Javadoc
- Objeto como atributo
- Objeto por parâmetro

PARADIGMA DA POO

- 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.

ABSTRAÇÃO

- Abstração é a habilidade de se concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais.
- Habilidade mental que permite aos seres humanos visualizarem os problemas do mundo real com vários graus de detalhe, dependendo do contexto corrente do problema.

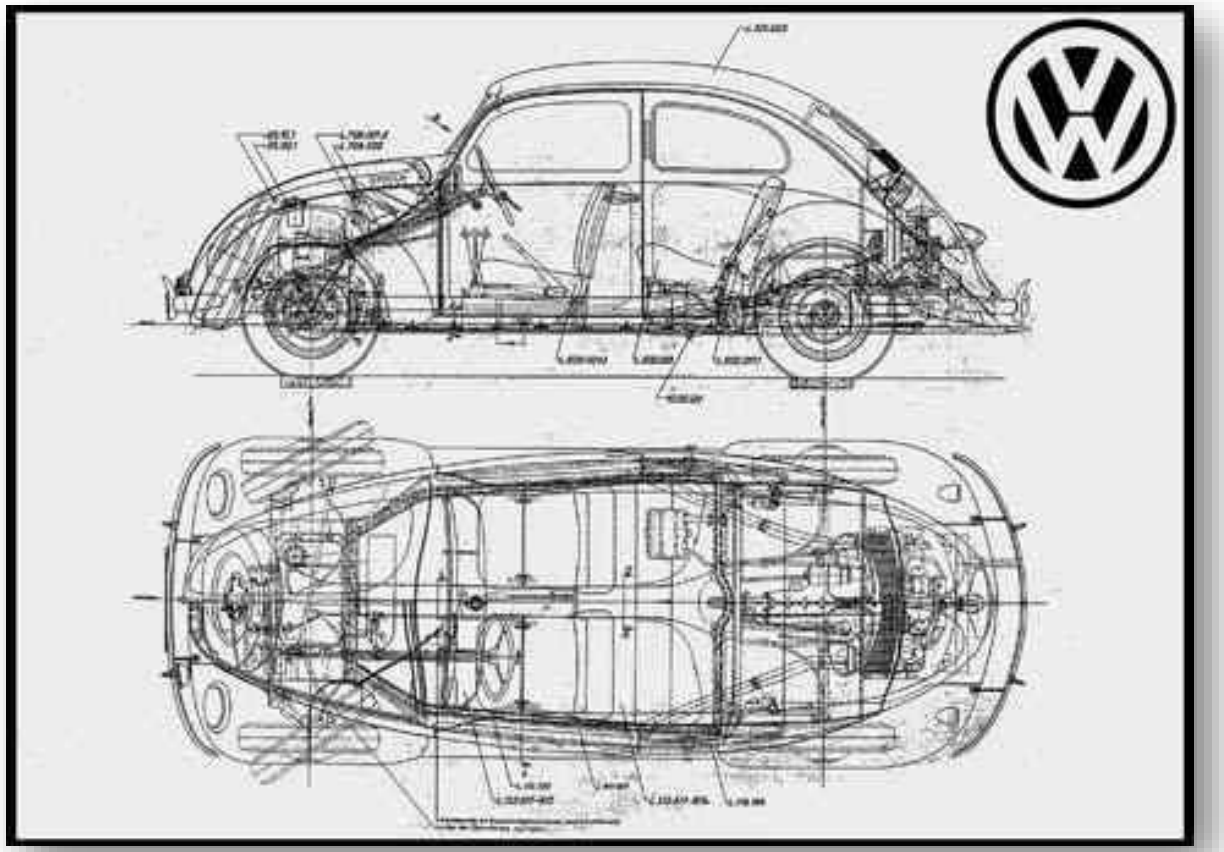
CLASSE

- É a unidade fundamental de programação da linguagem Java.
- Todos os recursos da linguagem e tudo que é desenvolvido em Java é estruturado em classes.
- As classes são estruturadas de acordo com a sua finalidade. Para entender o princípio da orientação a objetos é necessário entender o que são Entidades e a relação entre Classe de modelagem e Objeto.

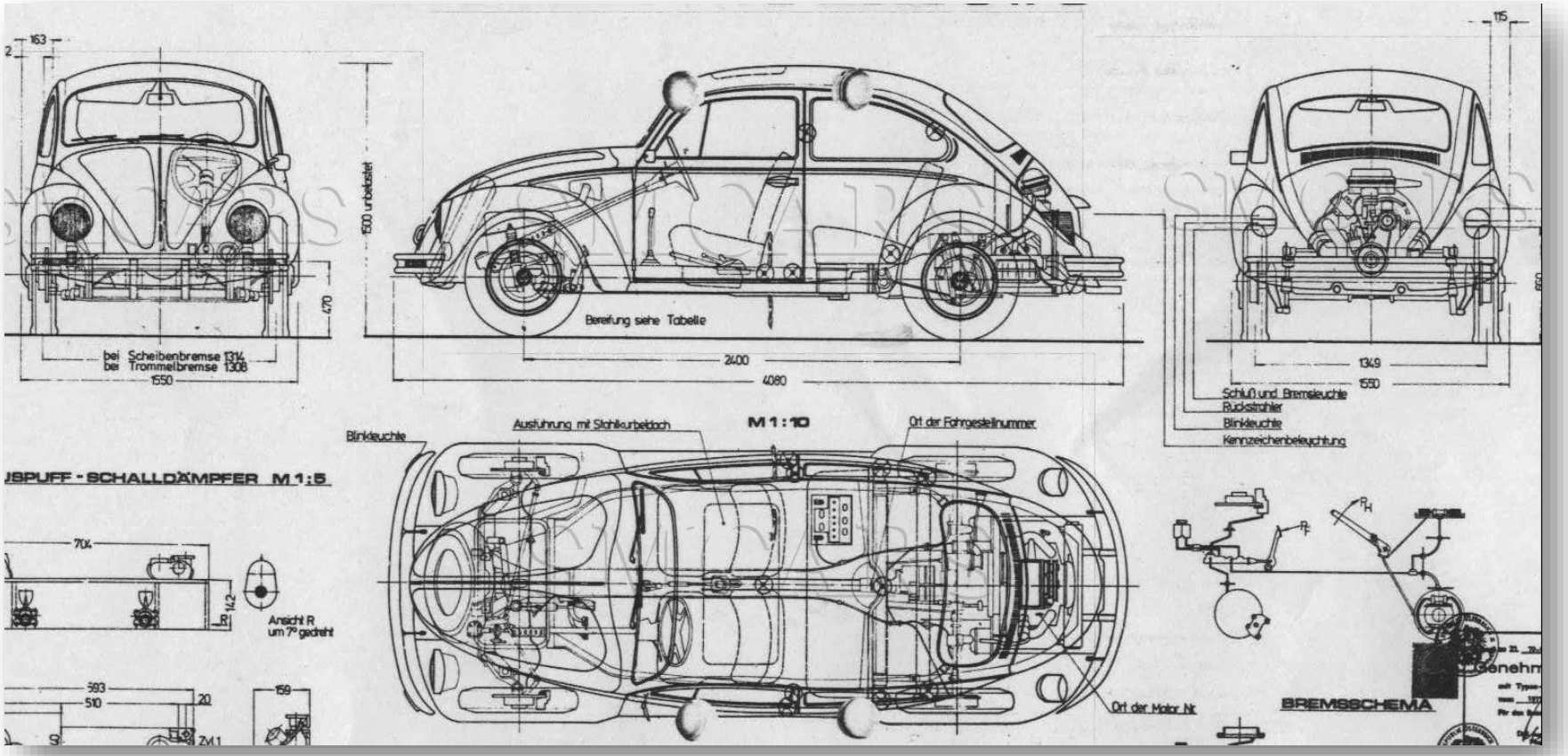
ENTIDADES

- Analisando um comércio qualquer (uma livraria, por exemplo) destacam-se alguns elementos chaves que concentram informações, executam ações e se relacionam na rotina de funcionamento da empresa. Por exemplo: funcionários, clientes, produtos, compra, venda, etc.
- No processo de analise e abstração identificam-se quais são esses elementos (e suas informações, ações e relacionamentos), ou seja, identificam-se as entidades do software que informatizará a livraria, por exemplo.

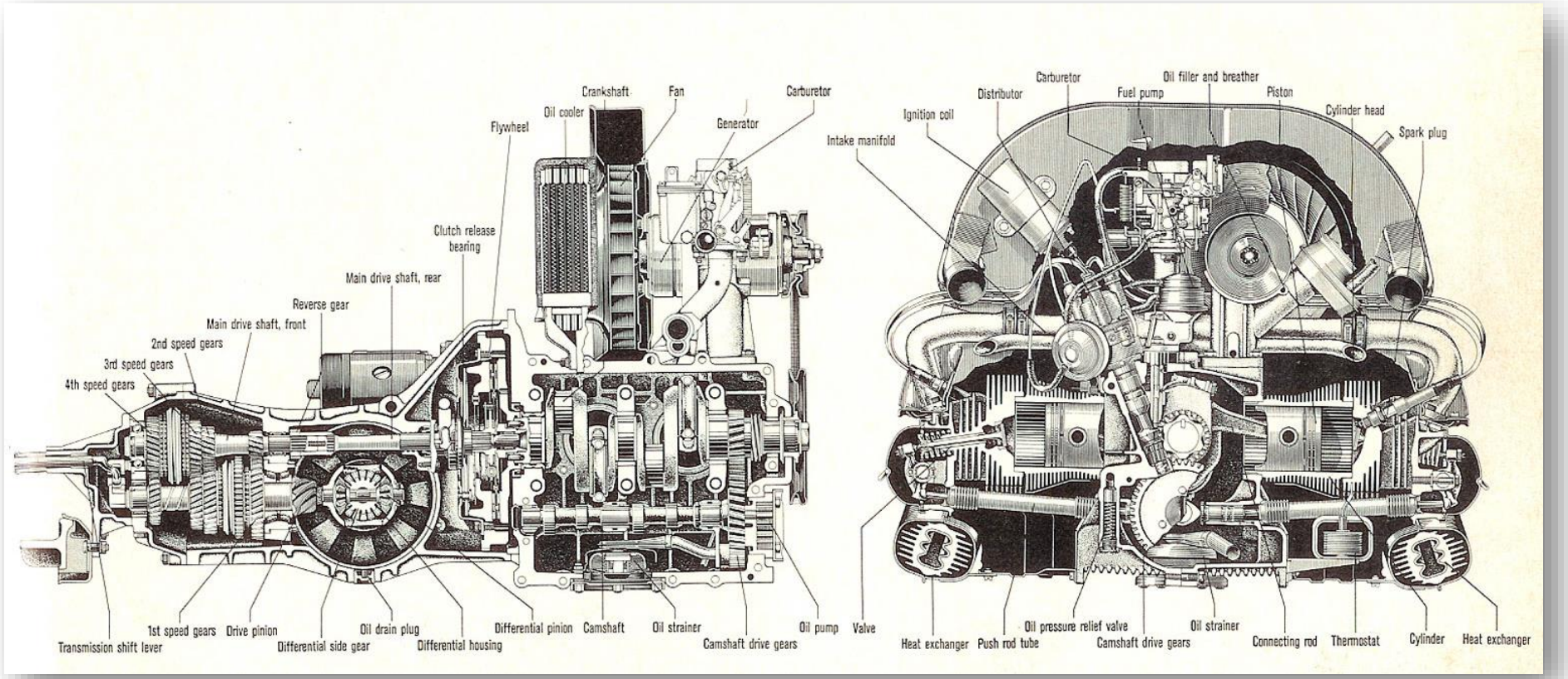
CLASSE DE MODELAGEM E OBJETOS - CONCEITO



CLASSE DE MODELAGEM E OBJETOS - CONCEITO



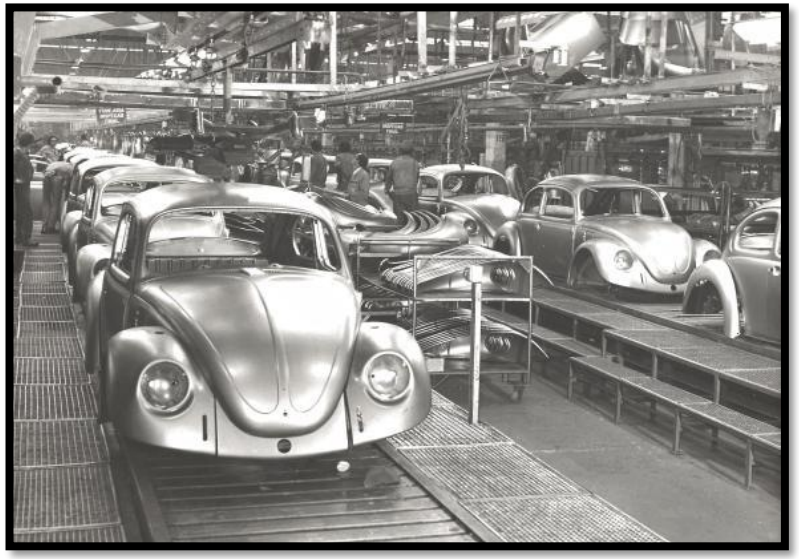
CLASSE DE MODELAGEM E OBJETOS - CONCEITO



CLASSE DE MODELAGEM E OBJETOS - CONCEITO

- Informações definidas no projeto que os futuros carros terão:
  - Modelo
  - Ano
  - Placa
  - Cor
- Ações definidas no projeto que os futuros carros poderão executar:
  - Acelerar
  - Frear
  - Trocar marcha
  - Virar (para esquerda ou para direita)
- Uma observação importante quanto as ações é que os detalhes técnicos (mecânicos, elétricos, hidráulicos) ficarão "escondidos" dos motoristas que apenas terão que aprender como usar esses recursos e não como eles funcionam internamente.

CLASSE DE MODELAGEM E OBJETOS - CONCEITO

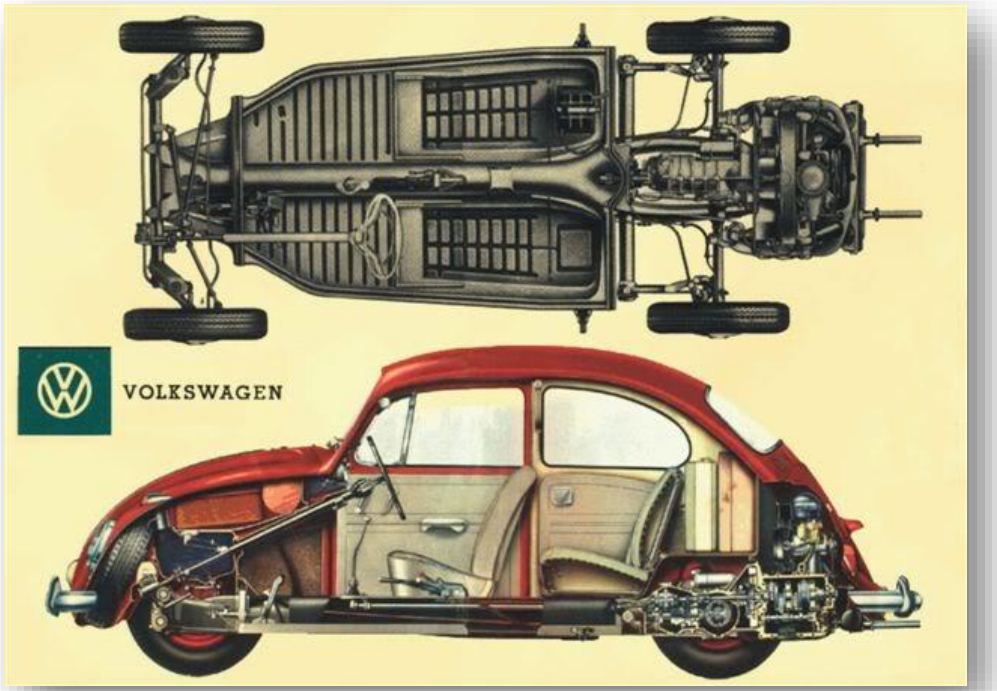


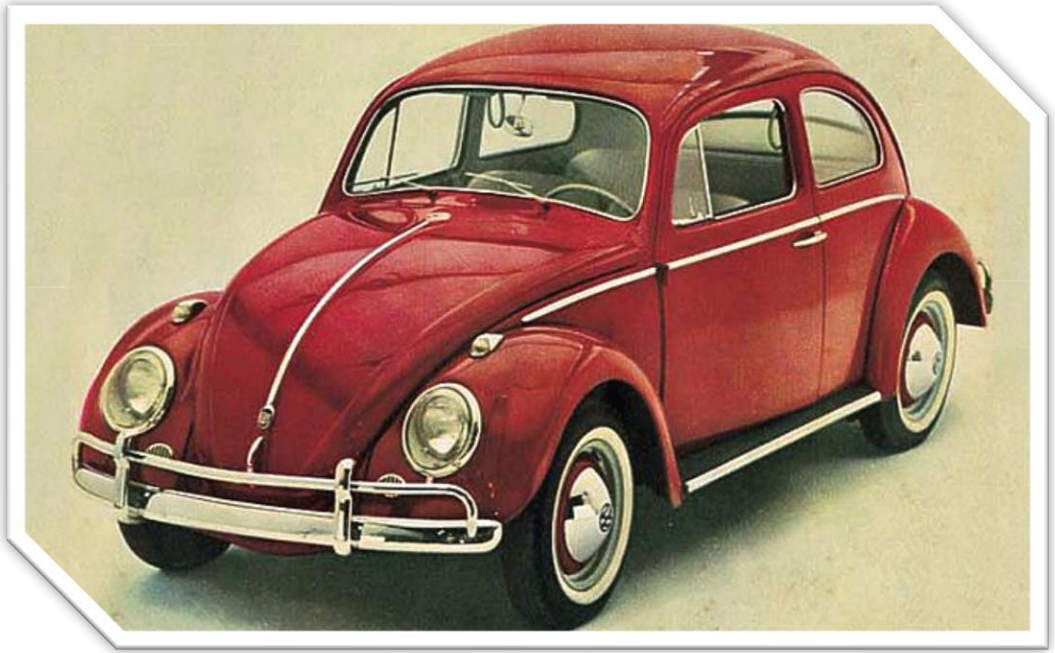
- No processo de fabricação o projeto é colocado em prática (seguido rigorosamente).

CLASSE DE MODELAGEM E OBJETOS - CONCEITO

- Podem ser fabricados quantos carros forem necessários a partir do mesmo projeto (e todos terão as mesmas especificações).

CLASSE DE MODELAGEM E OBJETOS - CONCEITO





CLASSE DE MODELAGEM

- É uma abstração de entidades existentes no domínio do software.
- 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.

Obs.: Utilizaremos esse termo para indicar que se trata de uma classe com a finalidade de dar origem a objetos, porém, essa **não é** uma terminologia adotada por convenção.

OBJETO

- É uma instância da classe de modelagem.
- Representa uma entidade do mundo real dentro da aplicação de forma individual, possuindo todas as informações e funcionalidades abstraídas na concepção da classe.

CLASSE DE MODELAGEM E OBJETOS

- Um objeto é criado (instanciado) a partir de uma classe de modelagem e, consequentemente, possui todos os elementos definidos nessa classe.
- É possível criar quantos objetos forem necessários a partir de uma classe de modelagem e todos terão as mesmas características e funcionalidades da classe que lhes deu origem.
- As alterações realizadas nas classes de modelagem refletem em seus objetos, conceitualmente, inclusive nos já criados. Por isso o termo instância, porque um objeto representa o instante atual de sua classe de modelagem.

ELEMENTOS DE 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.

RESUMINDO

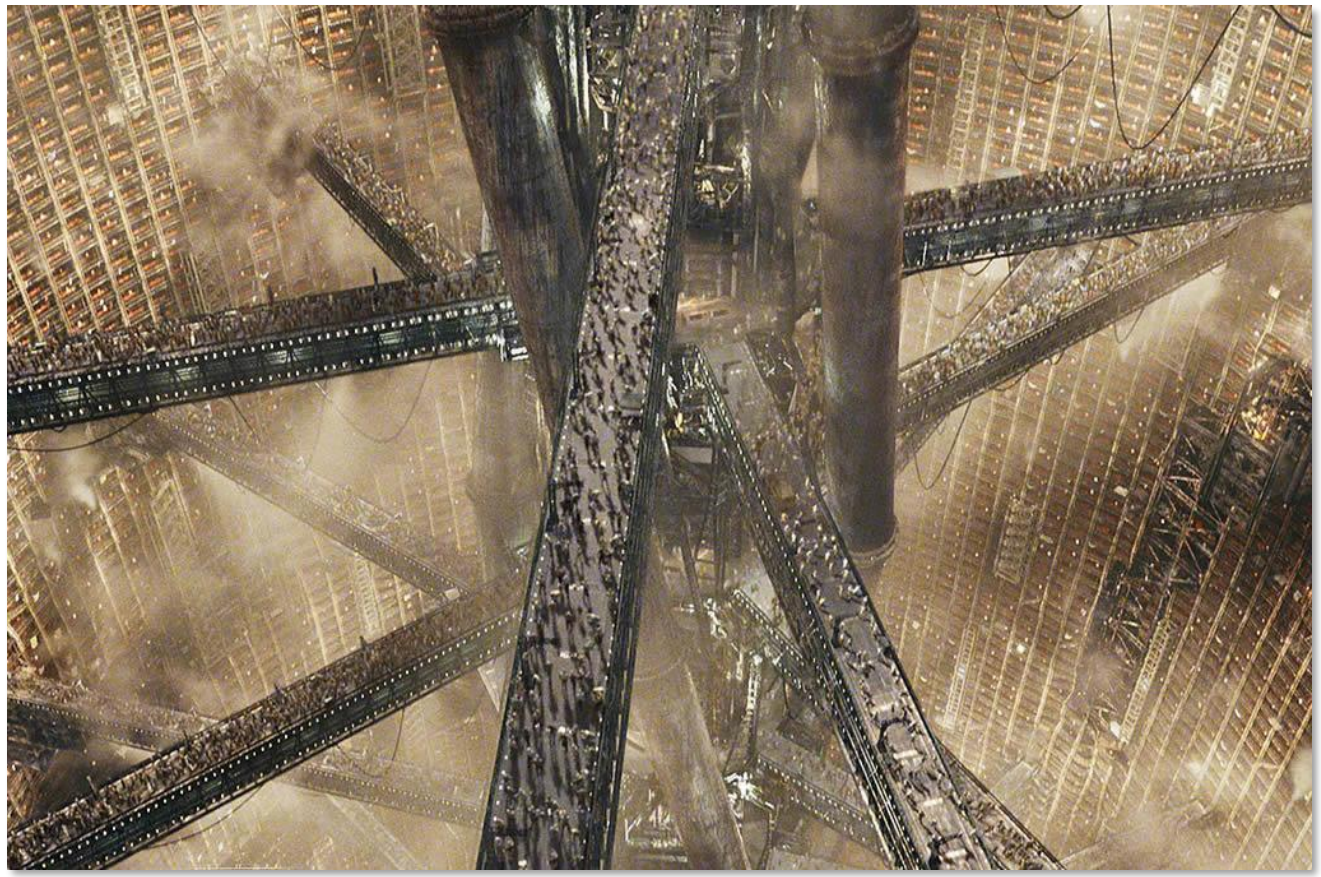
- Classe de modelagem (definição de tipo).
  - Atributos (dados)
  - Métodos (ações)
- Objetos (instâncias)

UML (UNIFIED MODELING LANGUAGE)

- É uma linguagem ou notação de diagramas para especificar, visualizar e documentar modelos de softwares orientados a objetos por meio de diagramas:
  - Diagrama de Caso de Uso: mostra atores (pessoas ou outros usuários do sistema), casos de uso (os cenários onde eles usam o sistema), e seus relacionamentos.
  - **Diagrama de Classe: mostra classes e os relacionamentos entre elas.**
  - Diagrama de Sequência: mostra objetos e uma sequência das chamadas do método feitas para outros objetos.
  - Diagrama de Colaboração: mostra objetos e seus relacionamentos, colocando ênfase nos objetos que participam na troca de mensagens.
  - Diagrama de Estado: mostra estados, mudanças de estado e eventos num objeto ou uma parte do sistema.
  - Diagrama de Atividade: mostra atividades e as mudanças de uma atividade para outra com os eventos ocorridos em alguma parte do sistema.
  - Diagrama de Componente: mostra os componentes de programação de alto nível.
  - Diagrama de Distribuição: mostra as instâncias dos componentes e seus relacionamentos.
  - Diagramas de Entidade-Associação: mostram os dados e as relações e as restrições entre os dados.

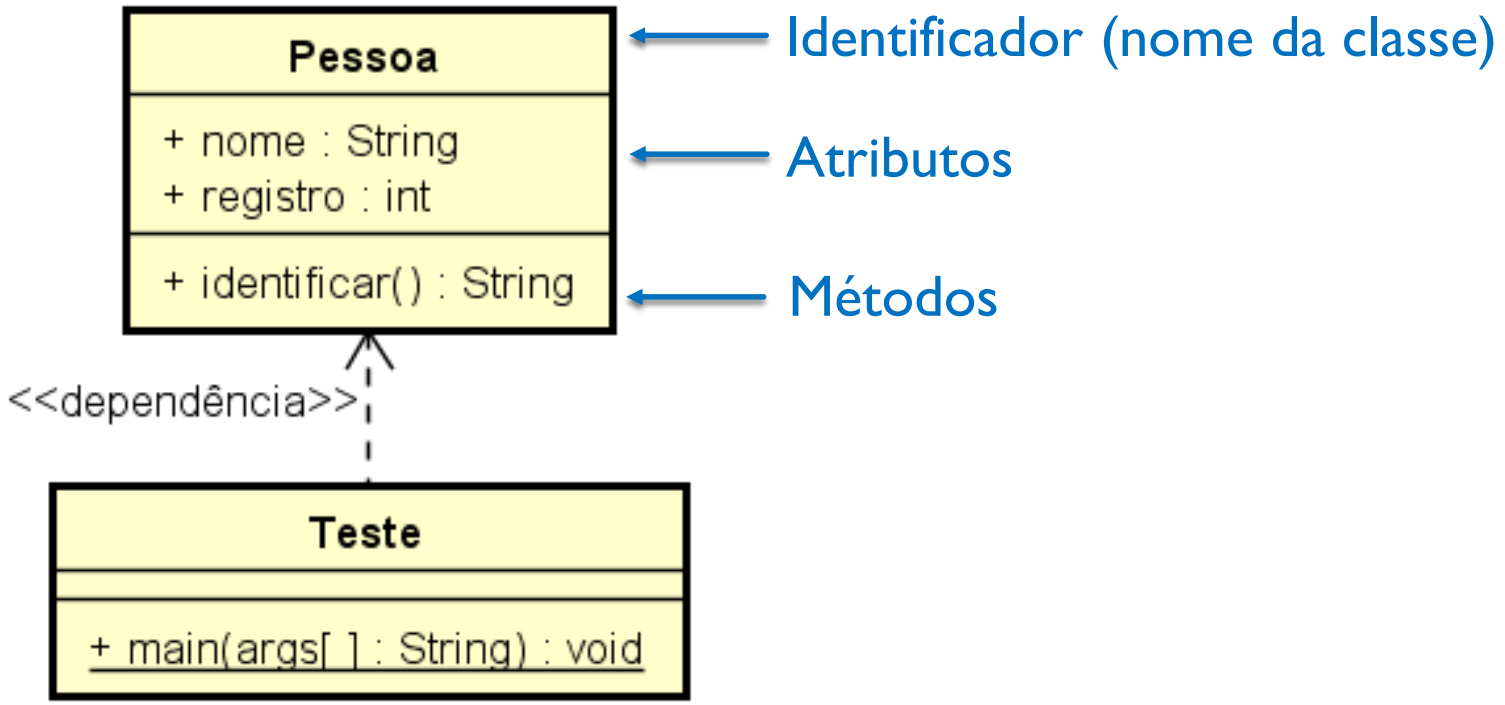
PROJETO ZION

- Projeto: Zion
  - Classe de modelagem: Pessoa
    - Atributos: nome e registro
    - Métodos: identificar
  - Classe (para teste): Teste



PROJETO ZION

- Diagrama de classe

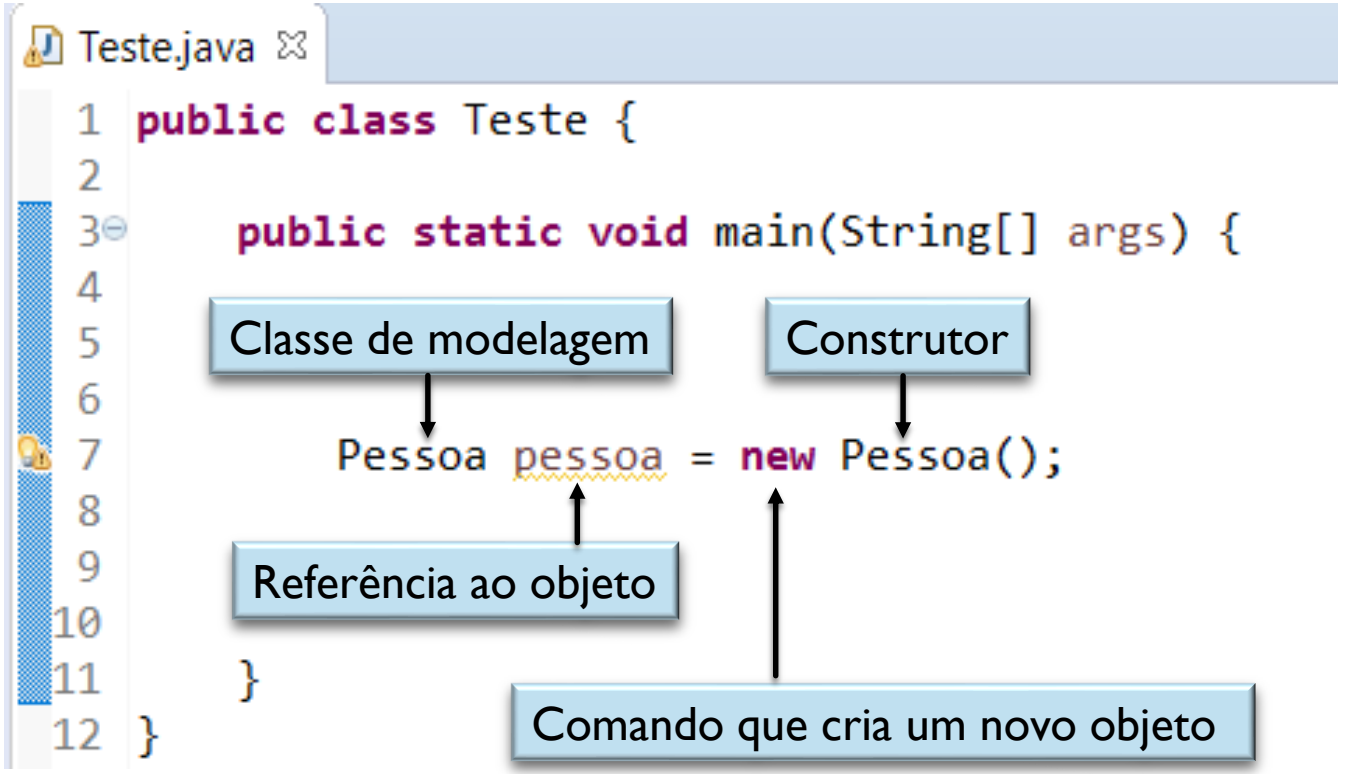


PROJETO ZION

Pessoa.java

```java
public class Pessoa {

    // Atributos
    String nome;
    int registro;


    // Método
    String identificar(){
        return "Nome: " + this.nome + "\nRegistro: " + this.registro;
    }
}
```

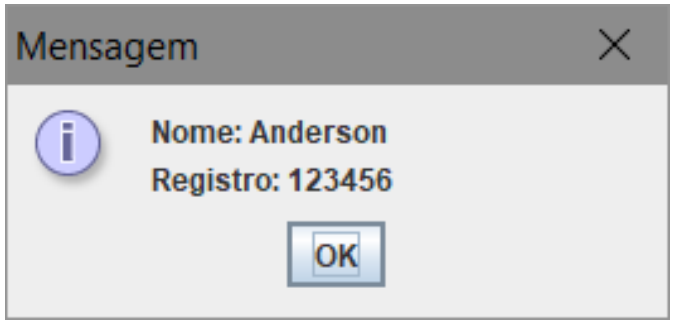
PROJETO ZION



PROJETO ZION

Teste.java

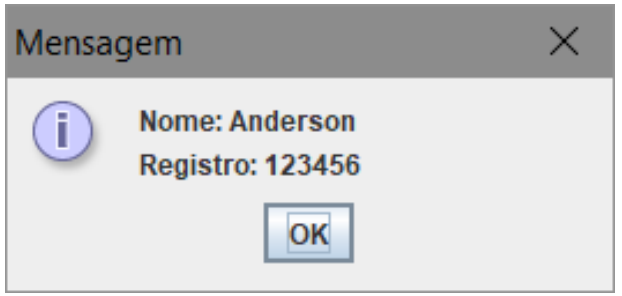
```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa();

        // Atribui valores aos atributos
        pessoa.nome = "Anderson";
        pessoa.registro = 123456;

        // Teste do método identificar
        // Invocação do método identificar e apresentação de seu retono
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```

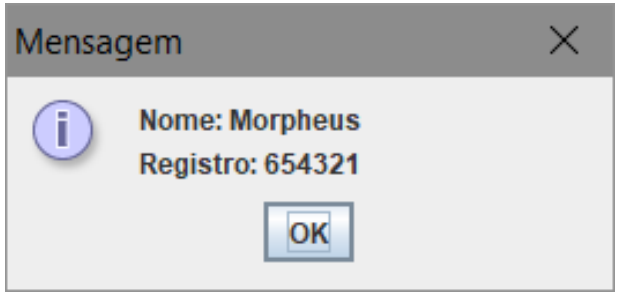


PROJETO ZION

Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objetos do tipo Pessoa
        Pessoa pessoa1 = new Pessoa();
        Pessoa pessoa2 = new Pessoa();

        // Atribui valores aos atributos
        pessoa1.nome = "Anderson";
        pessoa1.registro = 123456;

        pessoa2.nome = "Morpheus";
        pessoa2.registro = 654321;

        // Teste dos métodos identificar
        // Invocações dos métodos identificar e apresentações de seus retonos
        JOptionPane.showMessageDialog(null, pessoa1.identificar());
        JOptionPane.showMessageDialog(null, pessoa2.identificar());
    }
}
```





ENCAPSULAMENTO

- Consiste na proteção dos atributos de uma classe (e posteriormente dos objetos) de acessos indevidos.
- Considerando que todas as regras referentes a classe de modelagem 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.

ENCAPSULAMENTO

- O encapsulamento é implementado em duas etapas:
  - Restringindo a visibilidade do atributo utilizando o modificador de acesso private.
  - Criando métodos de acesso a esses atributos privados contendo as regras para leitura ou alteração dos dados armazenados.

ENCAPSULAMENTO - MODIFICADORES DE ACESSO

- private (-)
  - Visível somente dentro da classe em que foi implementado.
- public (+)
  - Visível em toda a aplicação.
- protected (#)
  - Visível dentro da classe em que foi implementado, em sua(s) subclasse(s) ou no mesmo pacote.

ENCAPSULAMENTO

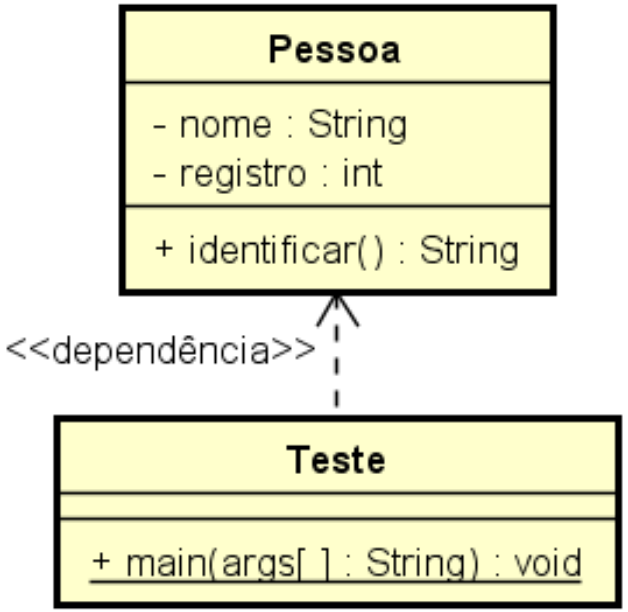
- Métodos de acesso.
  - Getters: métodos responsáveis pela leitura dos dados armazenados em atributos.
  - Setters: métodos responsáveis pela alteração dos dados armazenados em atributos.

ENCAPSULAMENTO

- As informações contidas nos objetos (seus atributos) poderiam ser protegidas de acordo com a necessidade (importância, confidencialidade), ou seja, deixar um determinado atributo disponível para leitura e alteração e restringir somente a leitura ou somente a alteração de outro (essa é a recomendação de alguns autores e é a prática comum de muitos desenvolvedores).
- Porém, com o intuito de definir um padrão (e por questões didáticas) adotaremos os seguintes procedimentos para a implementação do encapsulamento:
  - Todos os atributos serão definidos como privados.
  - Serão criados métodos de acesso para todos os atributos.
  - E serão implementadas as regras para leitura e alteração (codificação dos métodos de acesso) somente para os atributos e situações em que se fizer necessário.

PROJETO ZION

- Diagrama de classe



Obs.: Apesar da criação dos métodos de acesso não é necessário inseri-los no diagrama de classe. Por tratar-se de um conceito da orientação a objetos podemos considerá-los subentendidos.

Pessoa.java

```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;

    // Métodos de acesso
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getRegistro() {
        return registro;
    }
    public void setRegistro(int registro) {
        this.registro = registro;
    }

    // Funcionalidades dos objetos
    public String identificar(){
        return "Nome: " + this.getNome() + "\nRegistro: " + this.getRegistro();
    }
}
```

Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa();

        // Testes dos métodos de acesso
        pessoa.nome = "Anderson";
        The field Pessoa.registro is not visible = 123456;

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```

- A implementação padrão dos métodos de acesso não possuem regras para leitura ou alteração, mas, a estrutura dos métodos está pronta para recebe-las.

Pessoa.java

```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;

    // Métodos de acesso
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getRegistro() {
        return registro;
    }
    public void setRegistro(int registro) {
        this.registro = registro;
    }

    // Funcionalidades dos objetos
    public String identificar(){
        return "Nome: " + this.getNome() + "\nRegistro: " + this.getRegistro();
    }
}
```

Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa();

        // Testes dos métodos de acesso
        pessoa.setNome("Anderson");
        pessoa.setRegistro(123456);

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```

- A visibilidade privada permite o acesso direto ao atributo dentro da classe em que ele foi implementado, porém, por essa visão de encapsulamento, mesmo assim, os métodos de acesso deverão ser utilizados garantindo que, caso haja uma regra de acesso, ela seja sempre aplicada.

CONSTRUTORES

- São métodos específicos para a inicialização dos atributos (quando houver) de um objeto e é utilizado somente no momento da criação do objeto.
- Podem ser inicializados quantos atributos forem necessários, porém, quando definido um construtor com passagem de parâmetros deve-se implementar outro construtor "zerando" explicitamente os atributos.

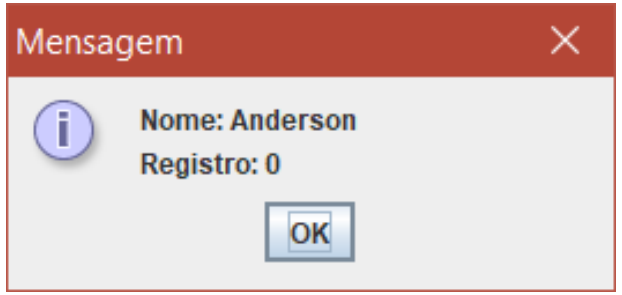
Obs.: Os métodos construtores também não precisam ser inseridos no diagrama de classes (pelo mesmo motivo dos métodos de acesso).

Pessoa.java

```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;

    // Construtor
    public Pessoa() {
        this.nome = "";
        this.registro = 0;
    }
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
    }
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
    }
```

Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa("Anderson");

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```

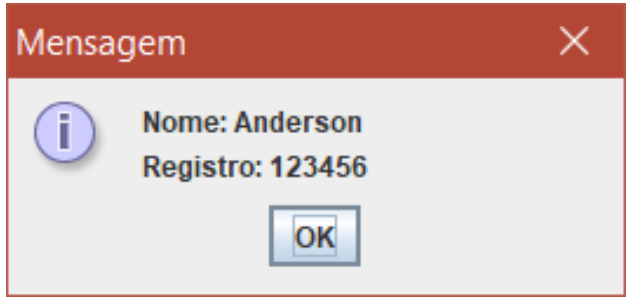


Pessoa.java

```java
public class Pessoa {
    // Atributos
    private String nome;
    private int registro;
    // Construtor
    public Pessoa() {
        this.nome = "";
        this.registro = 0;
    }
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
    }
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
    }
    // Métodos de acesso
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getRegistro() {
        return registro;
    }
    public void setRegistro(int registro) {
        this.registro = registro;
    }
    // Funcionalidades dos objetos
    public String identificar(){
        return "Nome: " + this.getNome() + "\nRegistro: " + this.getRegistro();
    }
}
```

Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa("Anderson", 123456);

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```



SOBRECARGA (OVERLOAD)

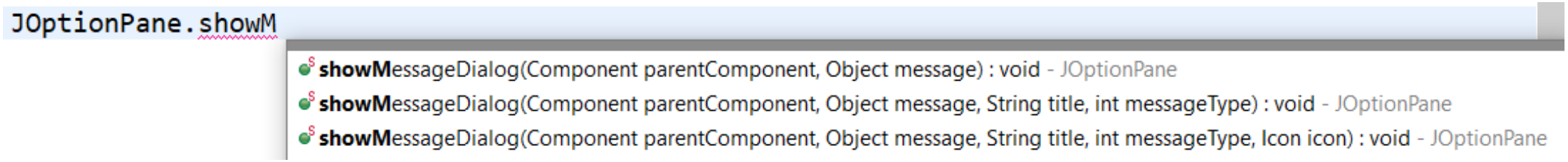
- É a possibilidade de implementar dois (ou mais) métodos com o mesmo nome desde que a passagem de parâmetros seja diferente como, por exemplo, os construtores.

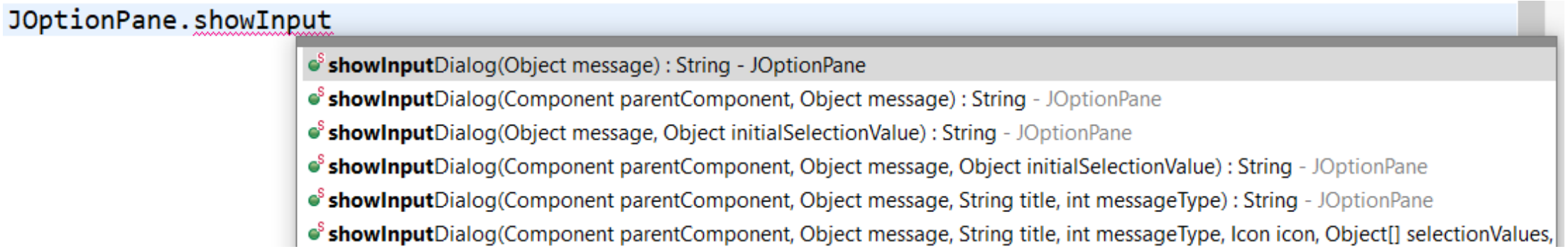
Pessoa.java

```
1  public class Pessoa {
2
3      // Atributos
4      private String nome;
5      private int registro;
6
7      // Construtor
8      public Pessoa() {
9          this.nome = "";
10         this.registro = 0;
11     }
12     public Pessoa(String nome) {
13         this.nome = nome;
14         this.registro = 0;
15     }
16     public Pessoa(String nome, int registro) {
17         this.nome = nome;
18         this.registro = registro;
19     }
```

SOBRECARGA (OVERLOAD)

- A sobrecarga de métodos é utilizada em várias classes da linguagem Java, por exemplo, na JOptionPane.





PROJETO ZION

Teste.java

```
1  import javax.swing.JOptionPane;
2
3  public class Teste {
4
5      public static void main(String[] args) {
6
7          // Criação de objeto do tipo Pessoa
8          Pessoa pessoa = new Pessoa();
9
10         // Testes dos métodos de acesso
11         pessoa.setNome("Anderson");
12         pessoa.setRegistro(123456);
13
14         // Teste do método identificar
15         JOptionPane.showMessageDialog(null, pessoa.identificar());
16     }
17 }
```

PROJETO ZION

Teste.java

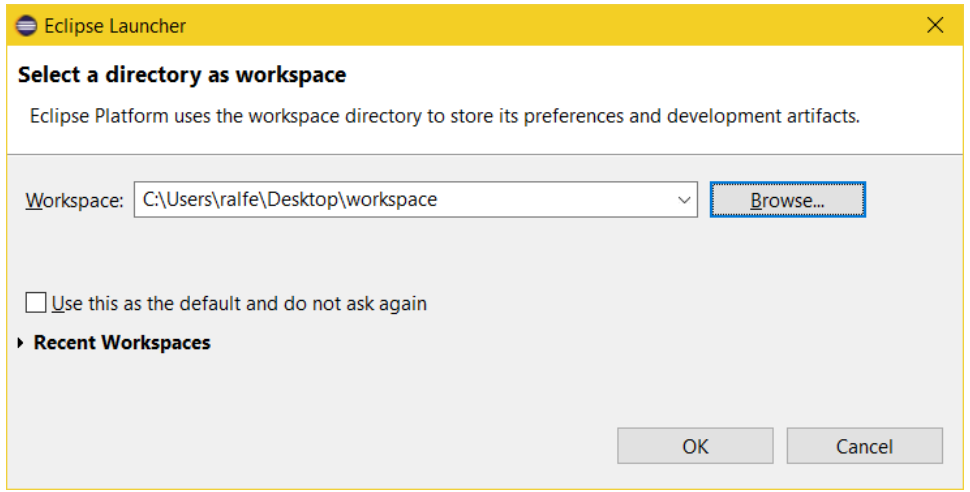
```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa1 = new Pessoa();
        Pessoa pessoa2 = new Pessoa();

        // Testes dos métodos de acesso
        pessoa1.setNome("Anderson");
        pessoa1.setRegistro(123456);

        pessoa2.setNome("Morpheus");
        pessoa2.setRegistro(654321);

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa1.identificar());
        JOptionPane.showMessageDialog(null, pessoa2.identificar());
    }
}
```

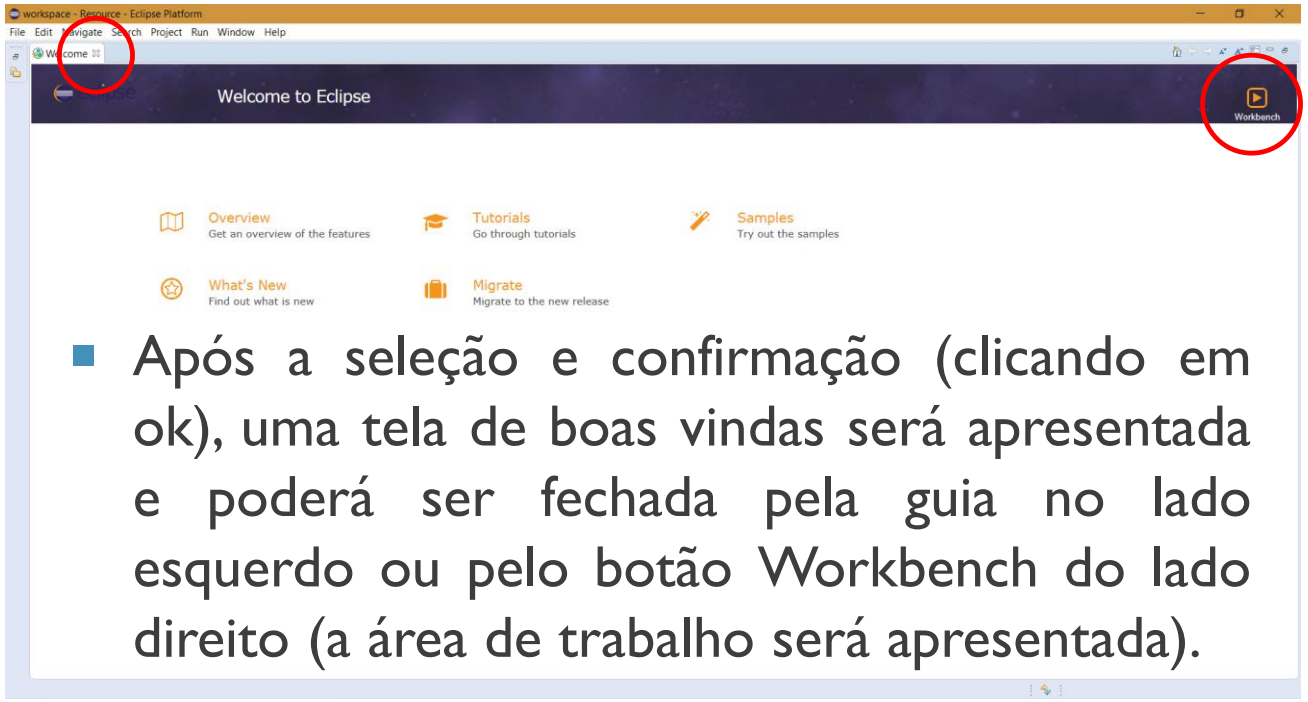
CRIAÇÃO DO PROJETO ZION NO ECLIPSE

- Projeto (Estrutura de pastas e arquivos que compõem a aplicação)
  - Classe de modelagem (Definição de tipo)
    - Atributos (definição de dados)
    - Construtores (métodos de inicialização de atributos)
    - Métodos de acesso (getters e setters)
    - Métodos específicos da classe (funcionalidades dos objetos)
  - Classe Teste
    - Método main

SELEÇÃO DA WORKSPACE

- Ao abrir o Eclipse será solicitado a Workspace (pasta que contém os projetos).

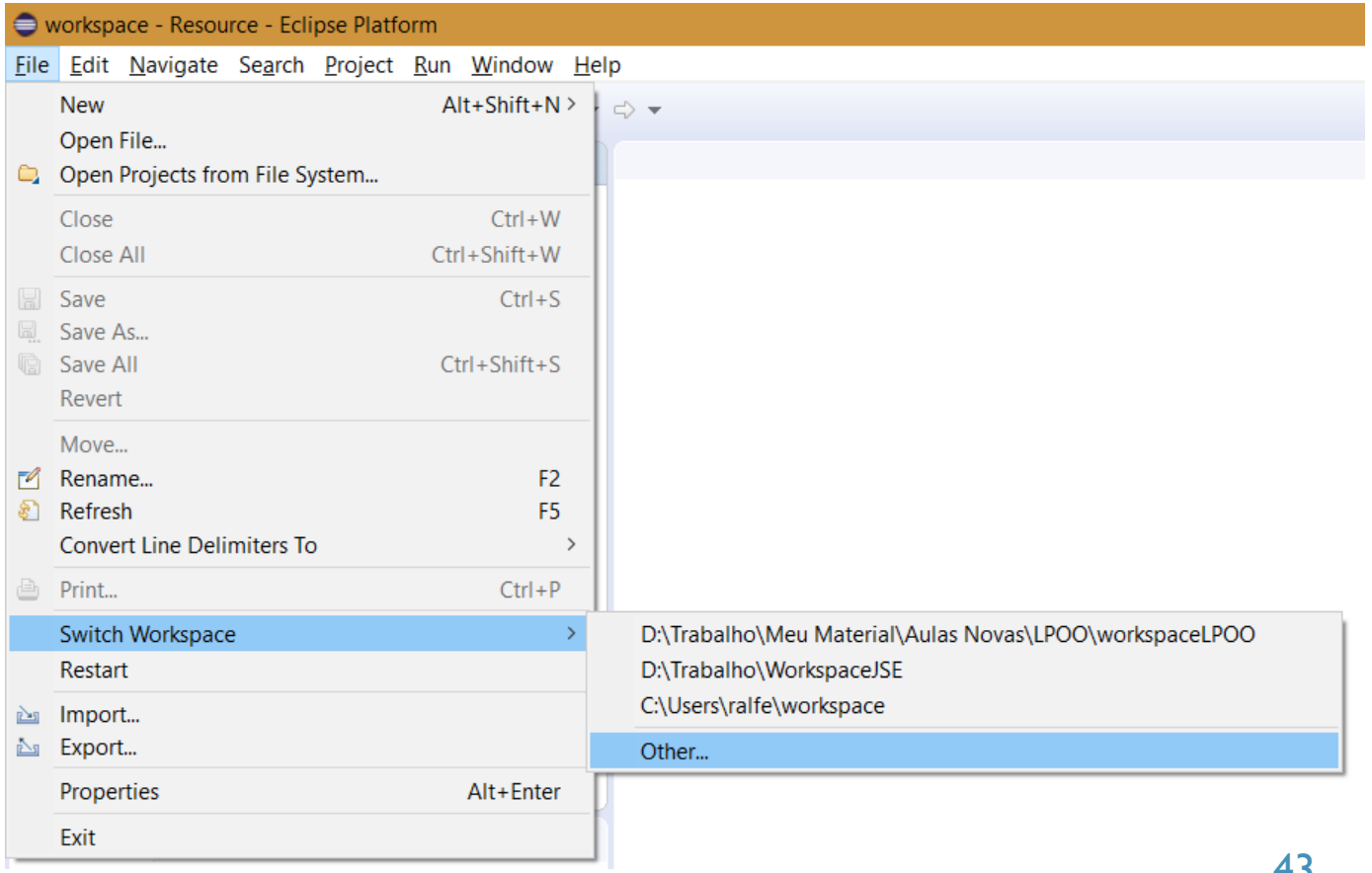




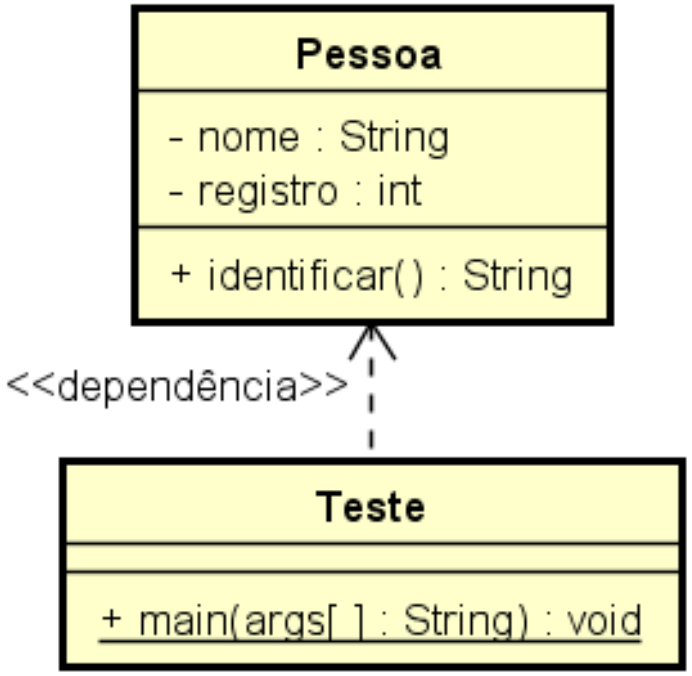
- Após a seleção e confirmação (clicando em ok), uma tela de boas vindas será apresentada e poderá ser fechada pela guia no lado esquerdo ou pelo botão Workbench do lado direito (a área de trabalho será apresentada).

SELEÇÃO DA WORKSPACE

- Caso a tela de seleção da Workspace não apareça na inicialização do Eclipse (o que acontece quando a opção "Use this as the default and do not ask again" é selecionada) vá no menu File > Switch Workspace > Other. A mesma tela inicial será apresentada.



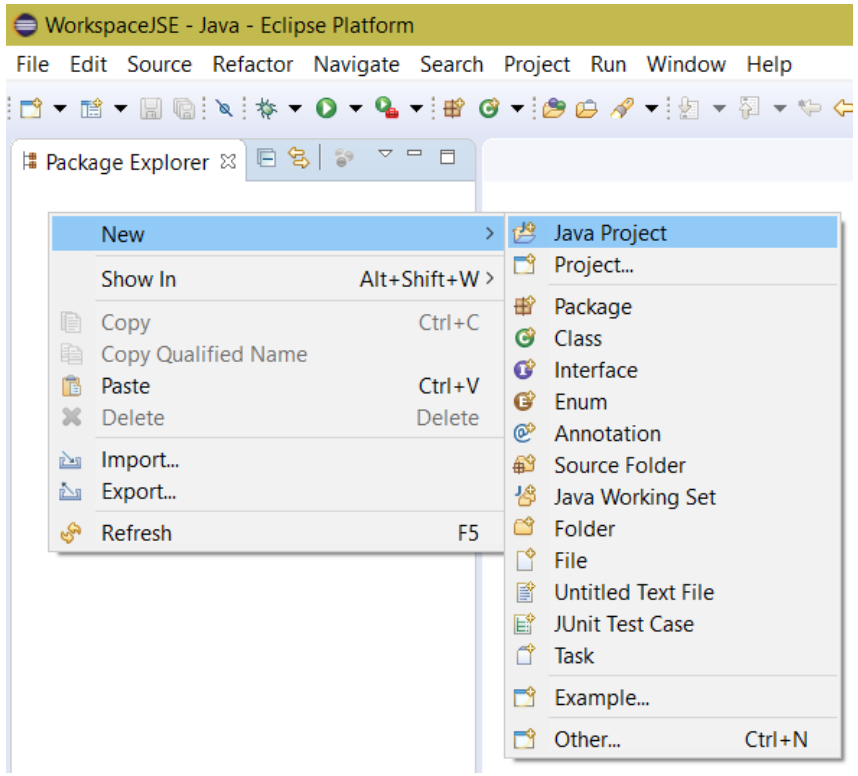
PROJETO ZION

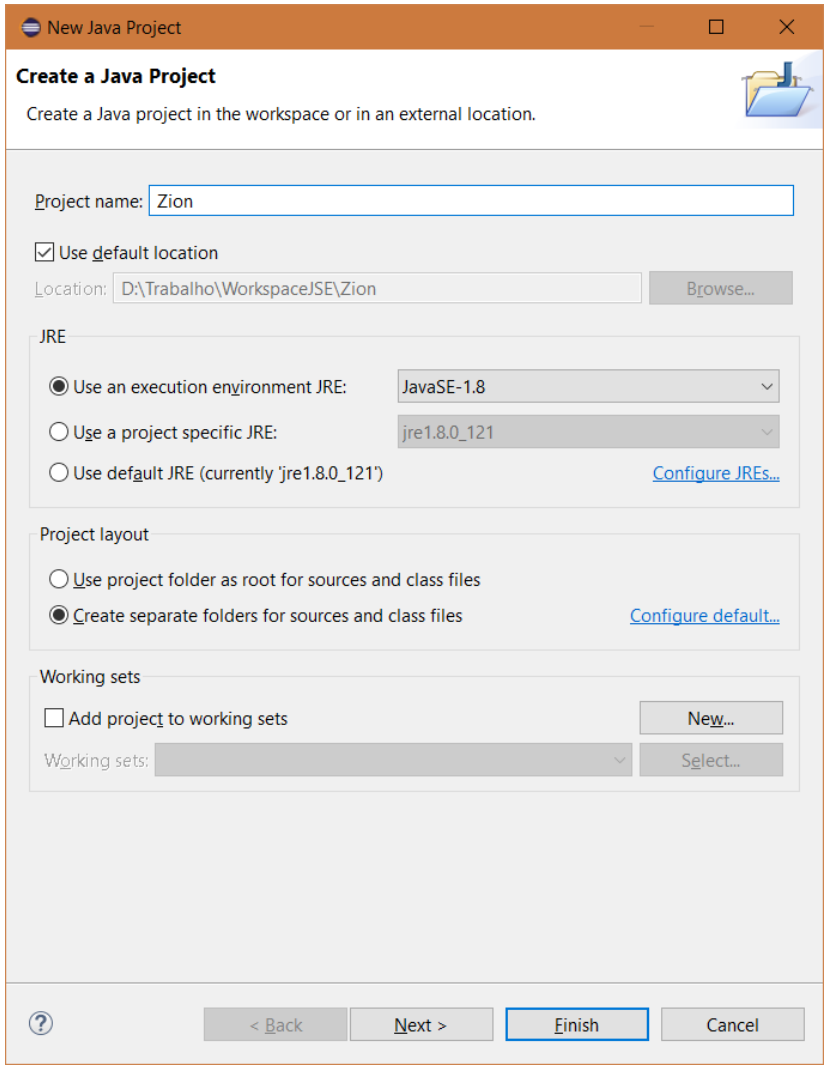


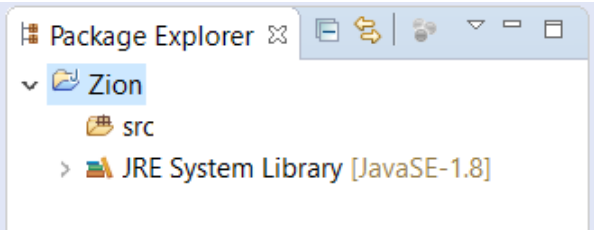
Classe Pessoa	
identificar	Monta e retorna uma String contendo os conteúdos dos atributos nome e registro.

Classe Teste	
main	Instancia um objeto do tipo Pessoa. Testa o método identificar. Testa a inserção de dados nos atributos. Testa os construtores.

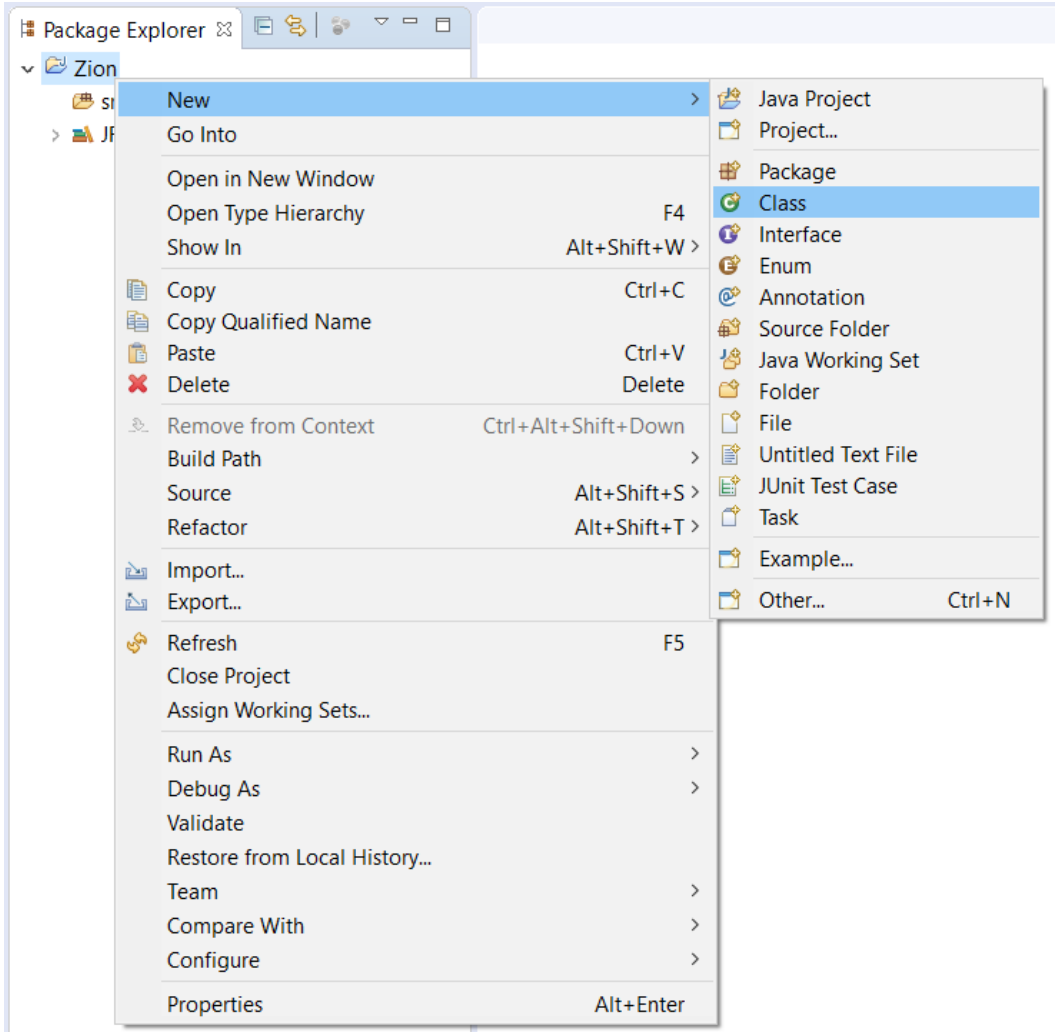
Criação do Projeto

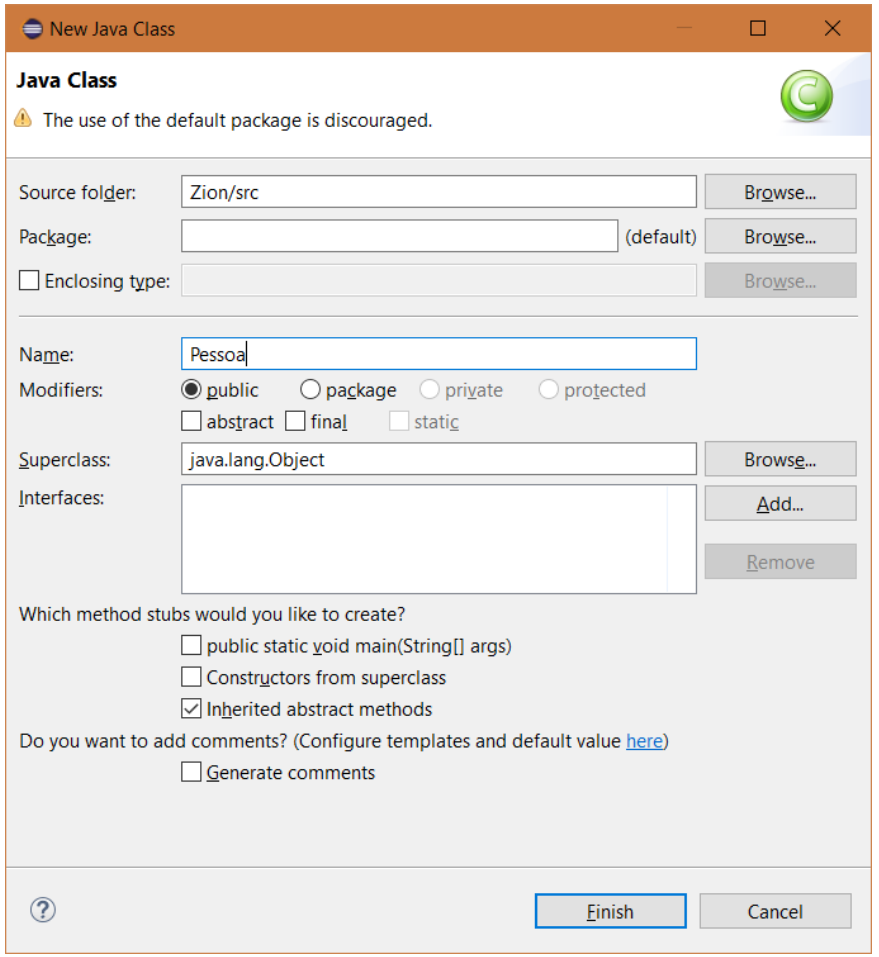




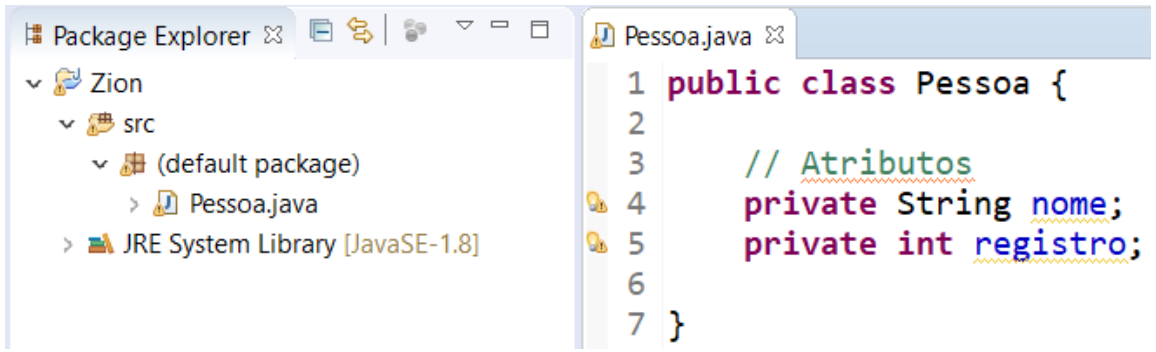


Criação da classe Pessoa

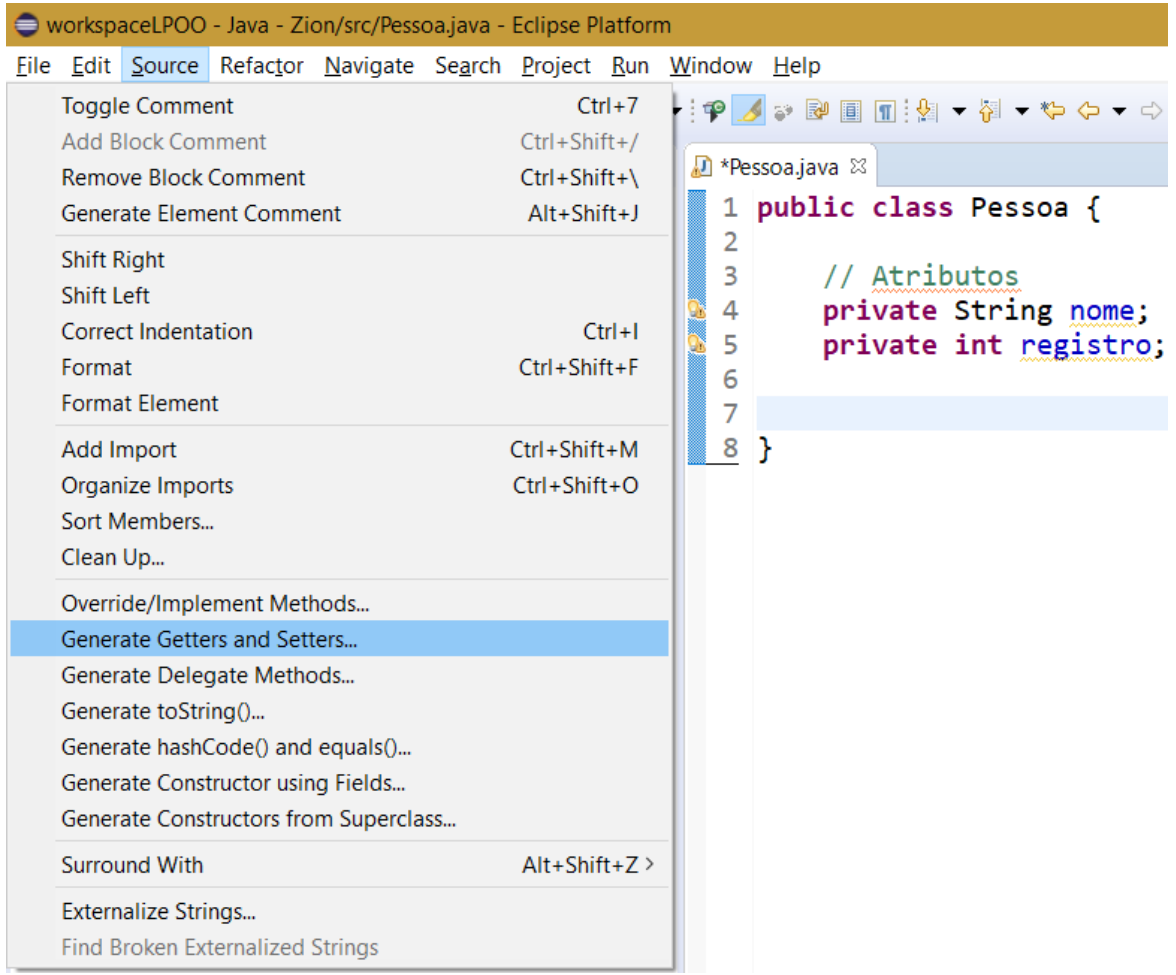




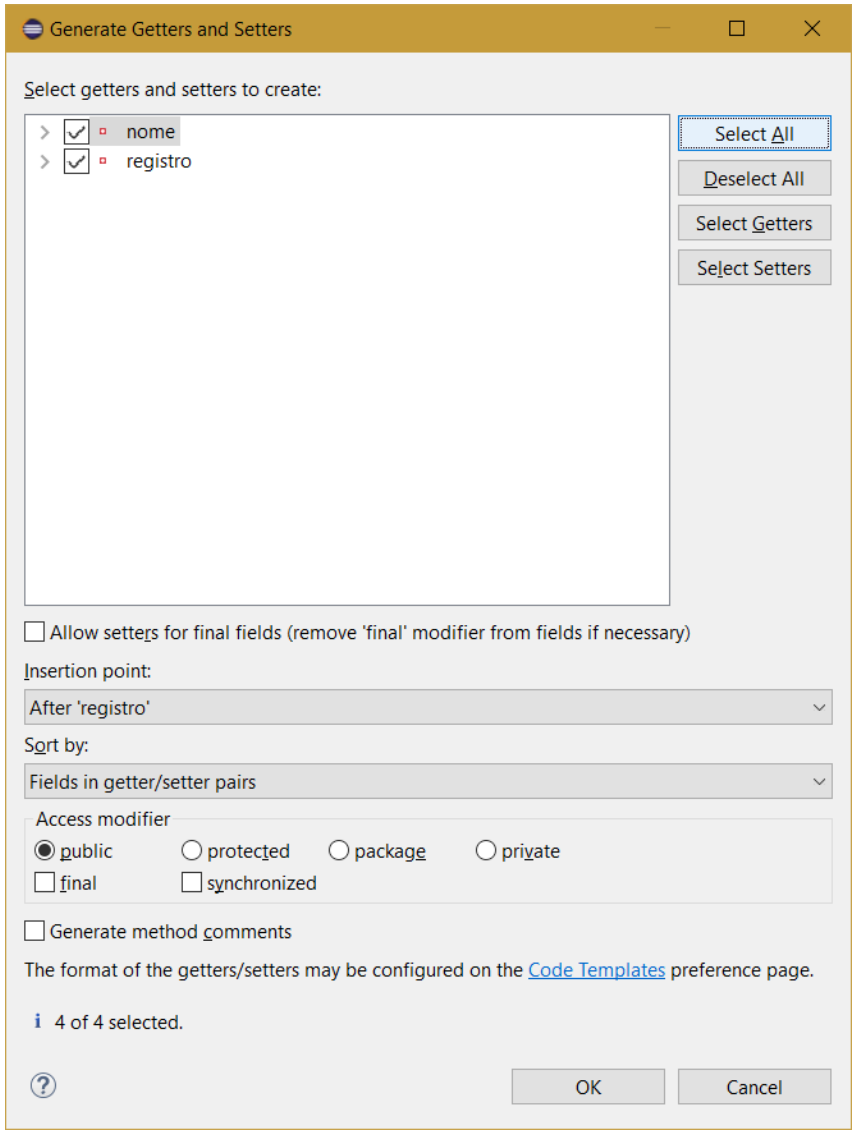
- Declaração de atributos



- Geração automática do métodos de acesso



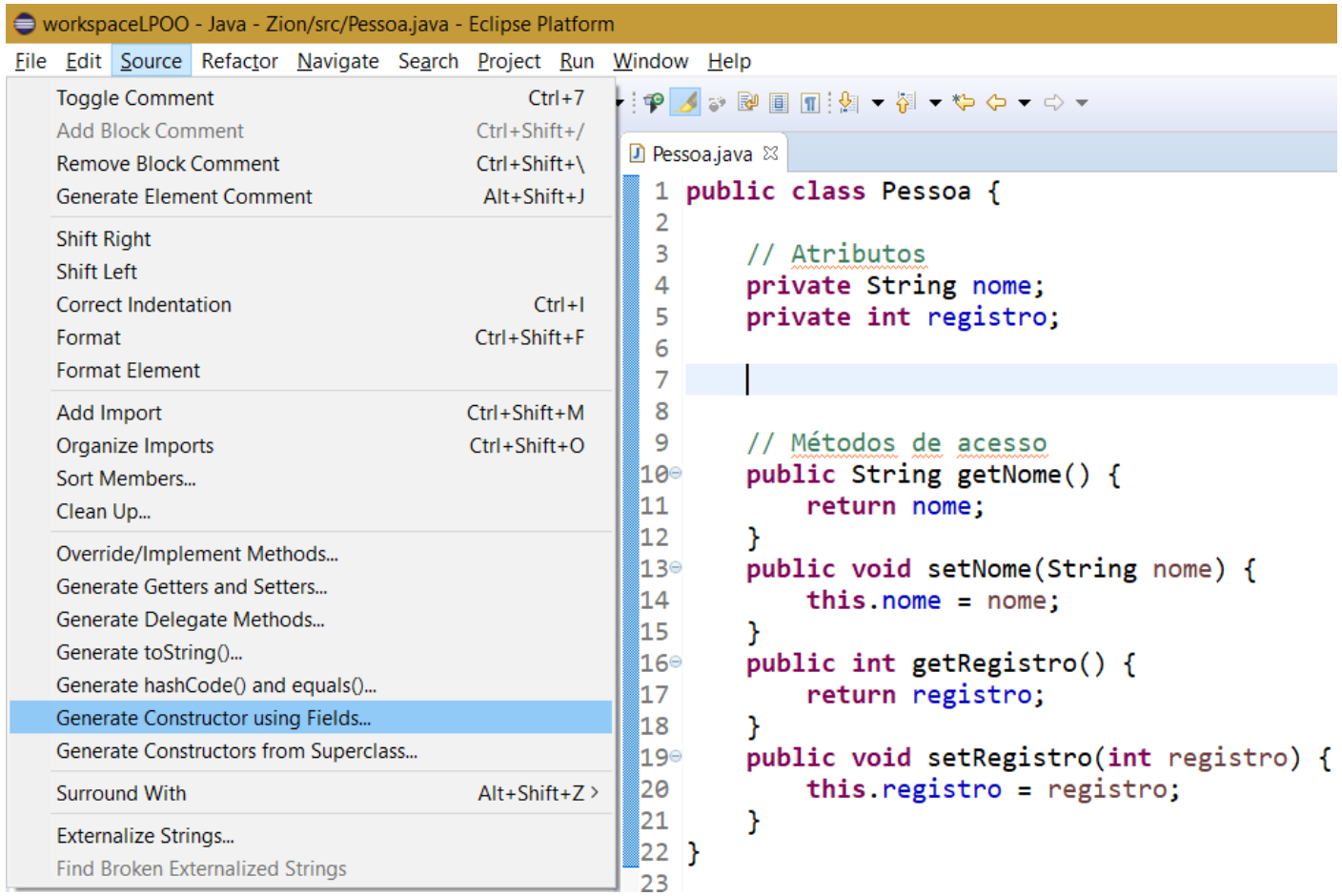
Métodos de acesso

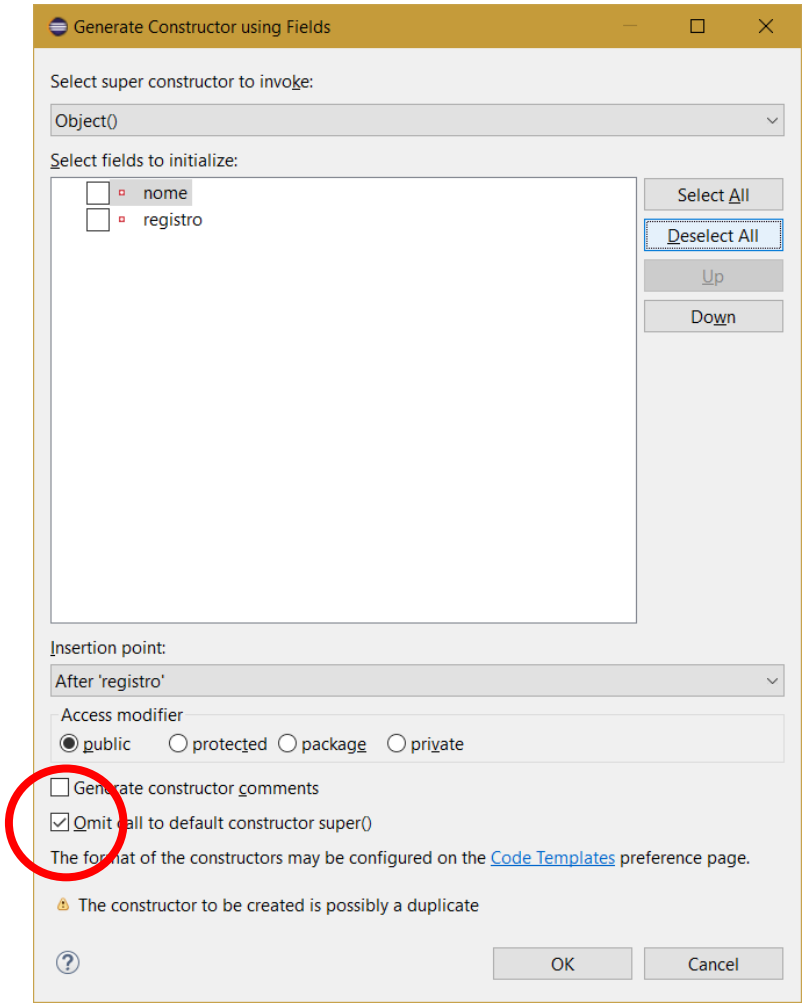


Pessoa.java

```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;

    // Métodos de acesso
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getRegistro() {
        return registro;
    }
    public void setRegistro(int registro) {
        this.registro = registro;
    }
}
```

Criação automática do construtor (sem passagem de parâmetro)

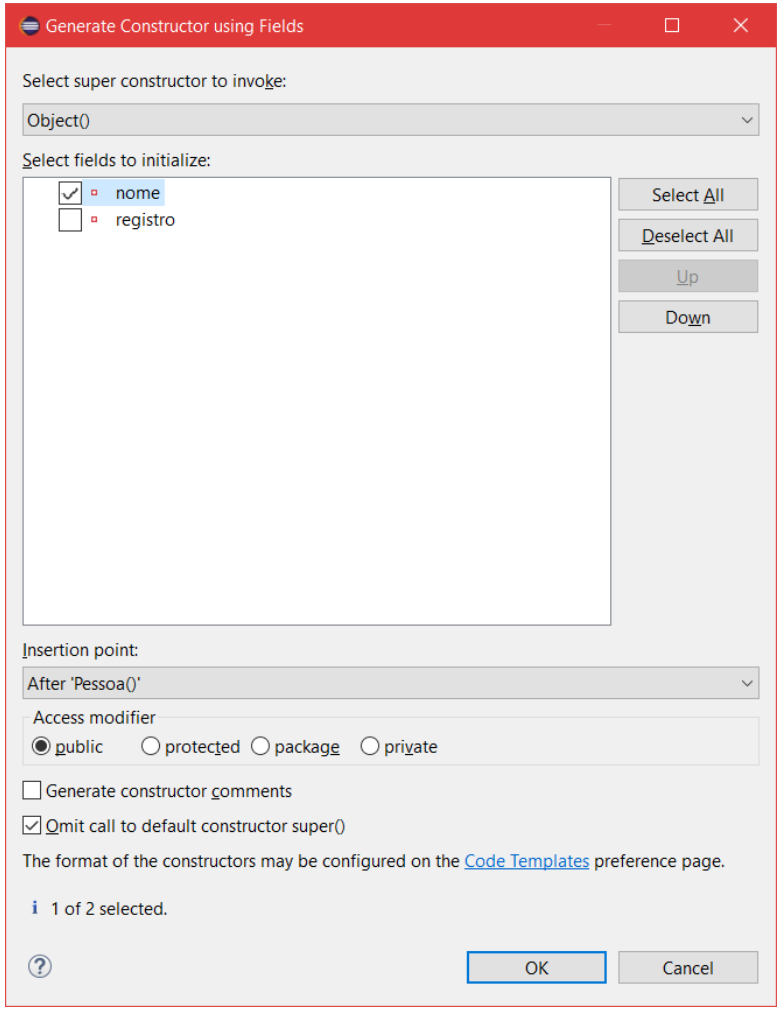




Criação automática do construtor (com passagem de parâmetro)

Pessoa.java

```
1  public class Pessoa {
2
3      // Atributos
4      private String nome;
5      private int registro;
6
7      // Construtores
8      public Pessoa() {
9      }
10
11
12
13     // Métodos de acesso
14     public String getNome() {
15         return nome;
16     }
17     public void setNome(String nome) {
18         this.nome = nome;
19     }
20     public int getRegistro() {
21         return registro;
22     }
23     public void setRegistro(int registro) {
24         this.registro = registro;
25     }
26 }
```



Construtores

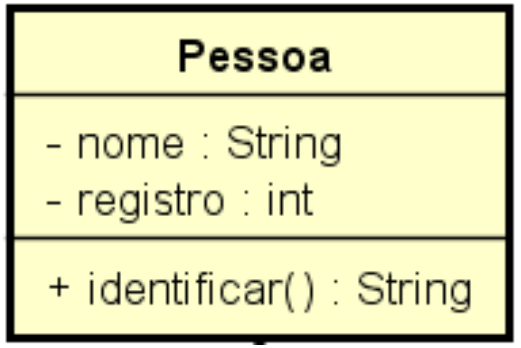
Pessoa.java

```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;

    // Construtor
    public Pessoa() {
        this.nome = "";
        this.registro = 0;
    }
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
    }
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
    }
```

- A inicialização dos atributos no construtor sem parâmetros é inserida manualmente e é importante atentar para os demais atributos que não recebem valores por parâmetro.

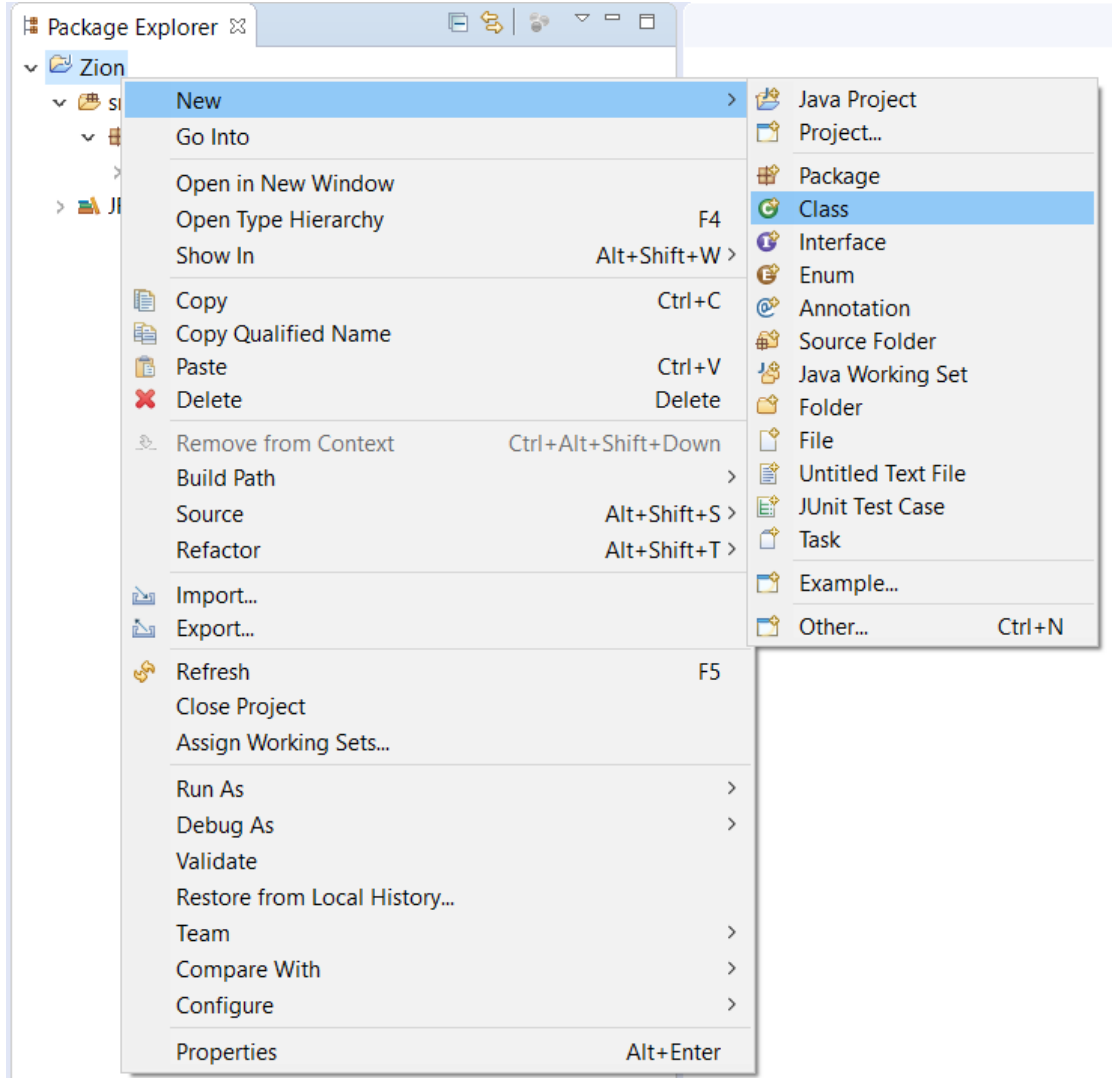
Pessoa.java

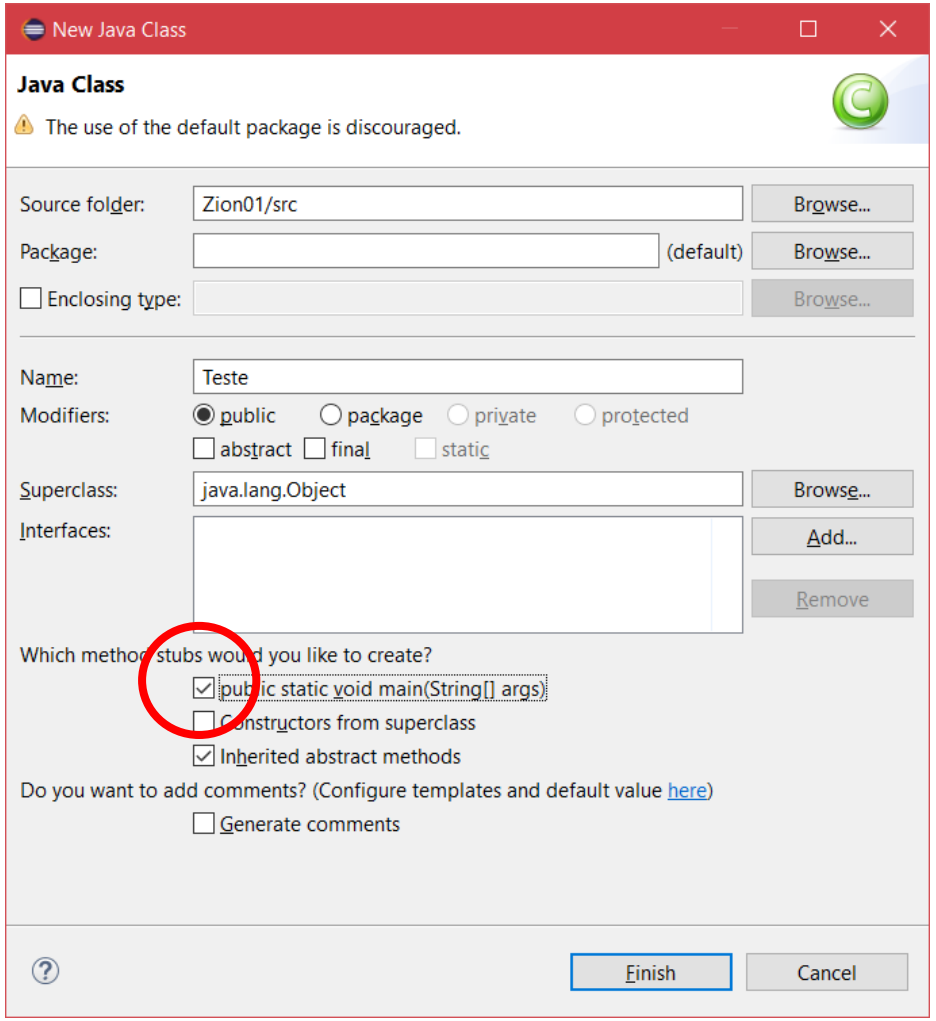
```java
public class Pessoa {
    // Atributos
    private String nome;
    private int registro;
    // Construtores
    public Pessoa() {
        this.nome = "";
        this.registro = 0;
    }
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
    }
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
    }
    // Métodos de acesso
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getRegistro() {
        return registro;
    }
    public void setRegistro(int registro) {
        this.registro = registro;
    }
    // Funcionalidades dos objetos
    public String identificar(){
        return "Nome: " + this.getNome() + "\nRegistro: " + this.getRegistro();
    }
}
```



Classe Pessoa	
identificar	Monta e retorna uma String contendo os conteúdos dos atributos nome e registro.

Criação da classe Teste

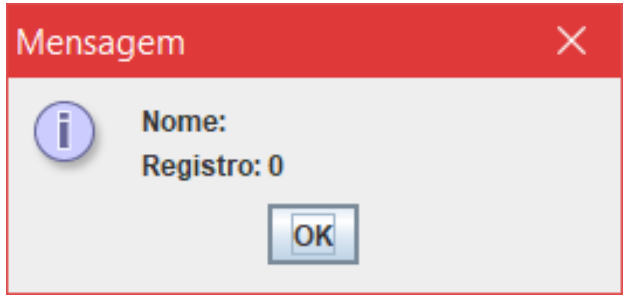




Testes

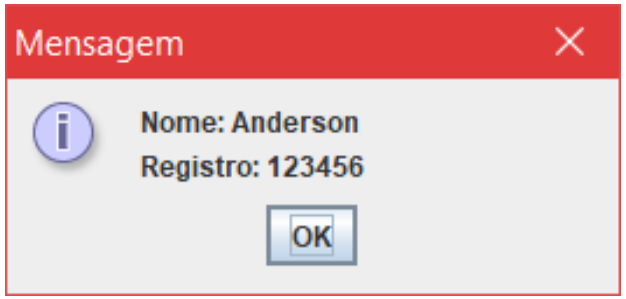
Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa();

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```

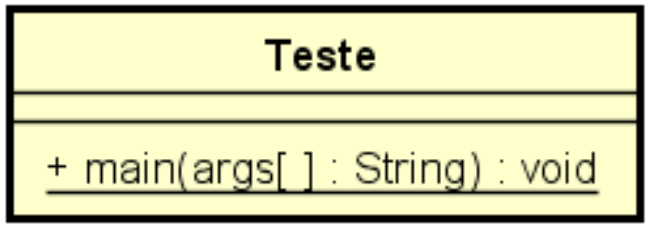


Teste.java

```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa();

        // Testes dos métodos de acesso
        pessoa.setNome("Anderson");
        pessoa.setRegistro(123456);

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```



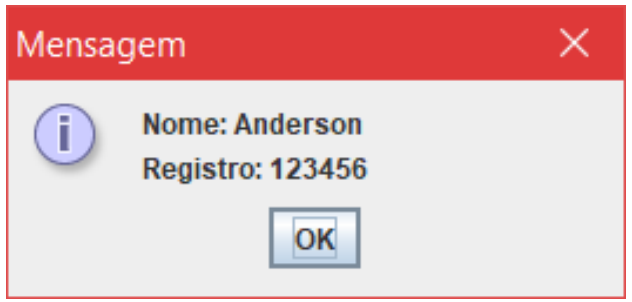
Testes

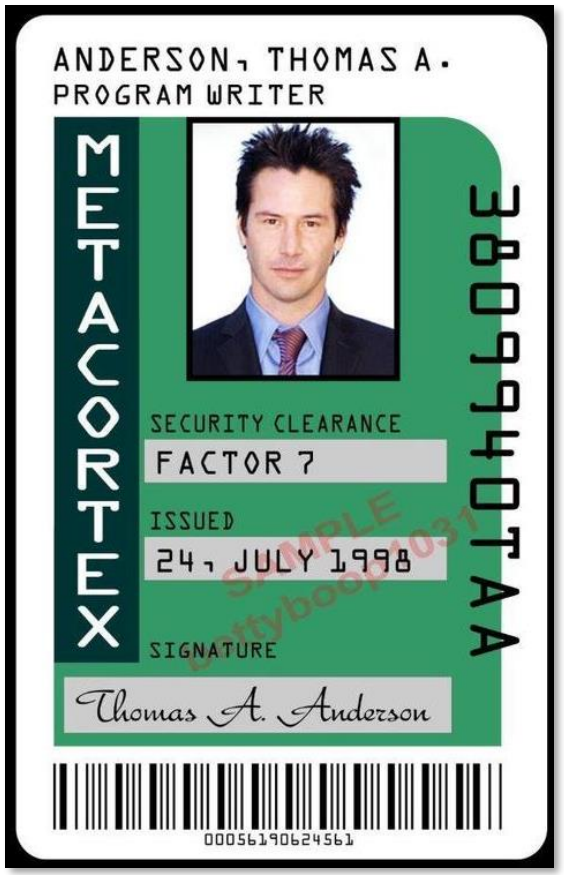


Classe Teste	
main	Instancia um objeto do tipo Pessoa. Testa o método identificar. Testa a inserção de dados nos atributos. Testa os construtores.

Teste.java

```
1  import javax.swing.JOptionPane;
2
3  public class Teste {
4
5      public static void main(String[] args) {
6
7          // Criação de objeto do tipo Pessoa
8          Pessoa pessoa = new Pessoa("Anderson", 123456);
9
10         // Testes dos métodos de acesso
11         // pessoa.setNome("Anderson");
12         // pessoa.setRegistro(123456);
13
14         // Teste do método identificar
15         JOptionPane.showMessageDialog(null, pessoa.identificar());
16     }
17 }
```





JAVADOC

- É a documentação da linguagem Java organizada e disponibilizada pela Oracle em formato HTML no endereço http://docs.oracle.com/javase/8/docs/api/
- O Eclipse possui um recurso para geração de documentação dos projetos desenvolvidos nos moldes do Javadoc (menu Project > Generate Javadoc) baseando-se nas descrições delimitadas por /** */
- O Javadoc contém tags que definem determinadas informações como @author, @version, @param, @return, etc. e organizam esses dados na documentação.

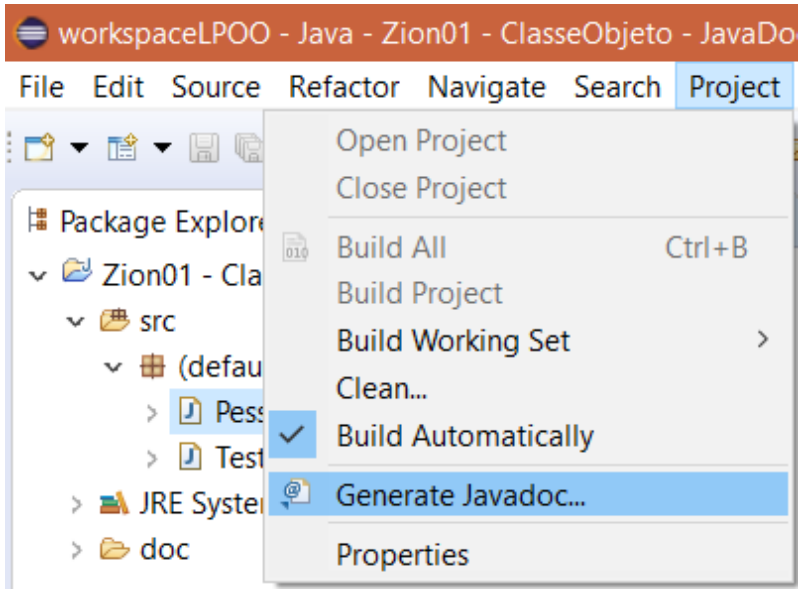
Pessoa.java

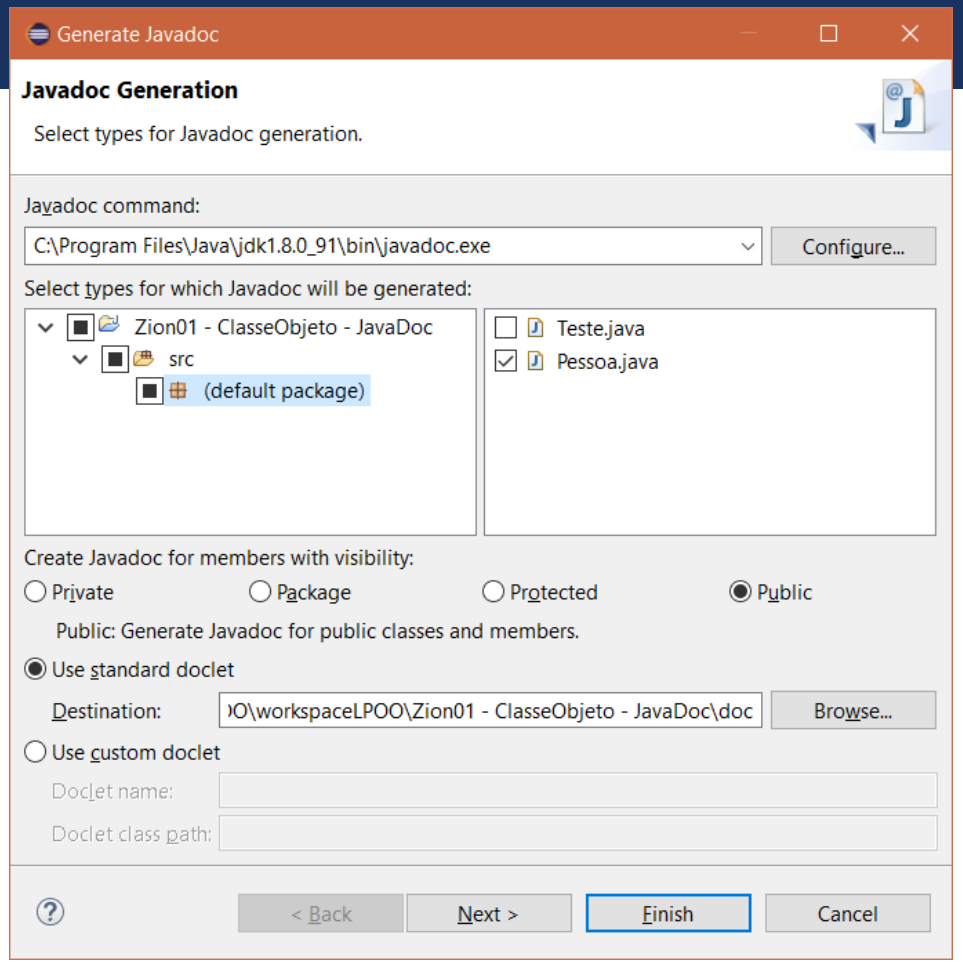
```java
/**
 * @author Prof. Ralfe
 * @version Última atualização: 12/07/2017
 */
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;

    // Construtores
    /**
     * Inicializa o atributo nome com espaço em branco e registro com zero)
     */
    public Pessoa() {
        this.nome = "";
        this.registro = 0;
    }
    /**
     * @param nome para inicialização do atributo nome
     * Inicializa o atributo registro com zero
     */
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
    }
    /**
     * @param nome para inicialização do atributo nome
     * @param registro para inicialização do atributo registro
     */
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
    }
```

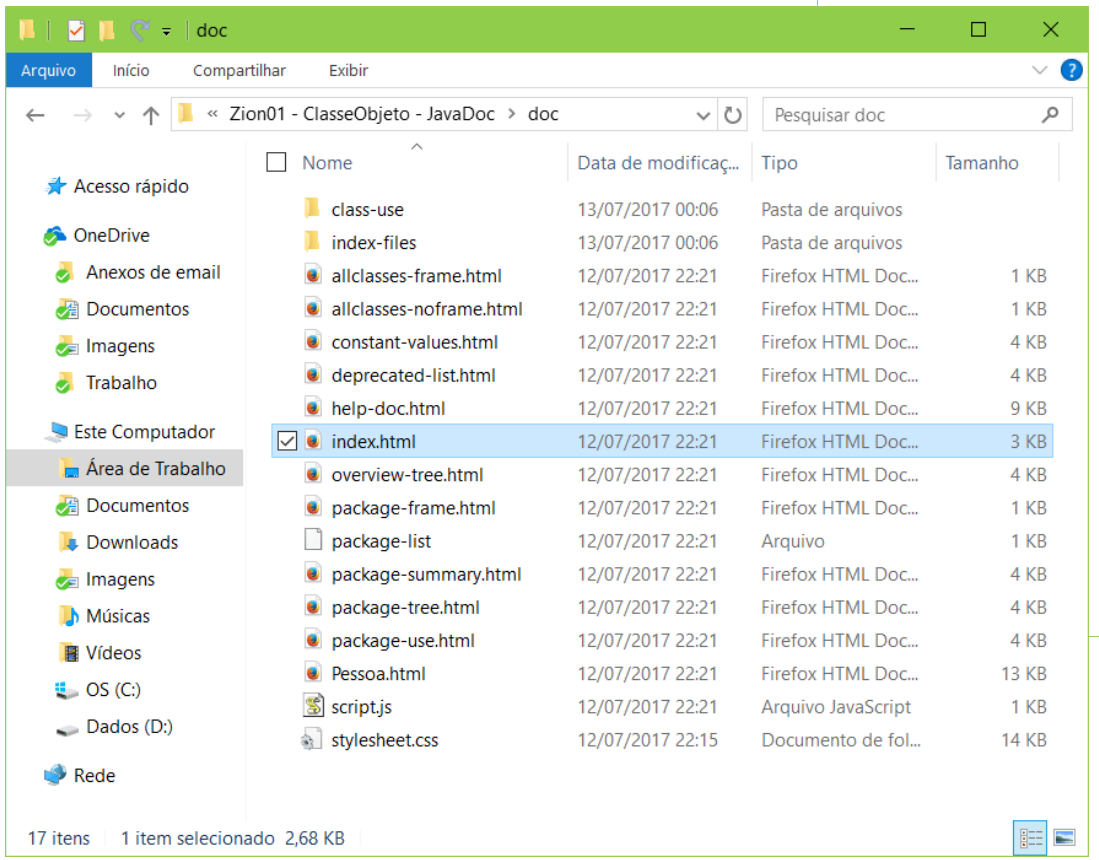
Pessoa.java

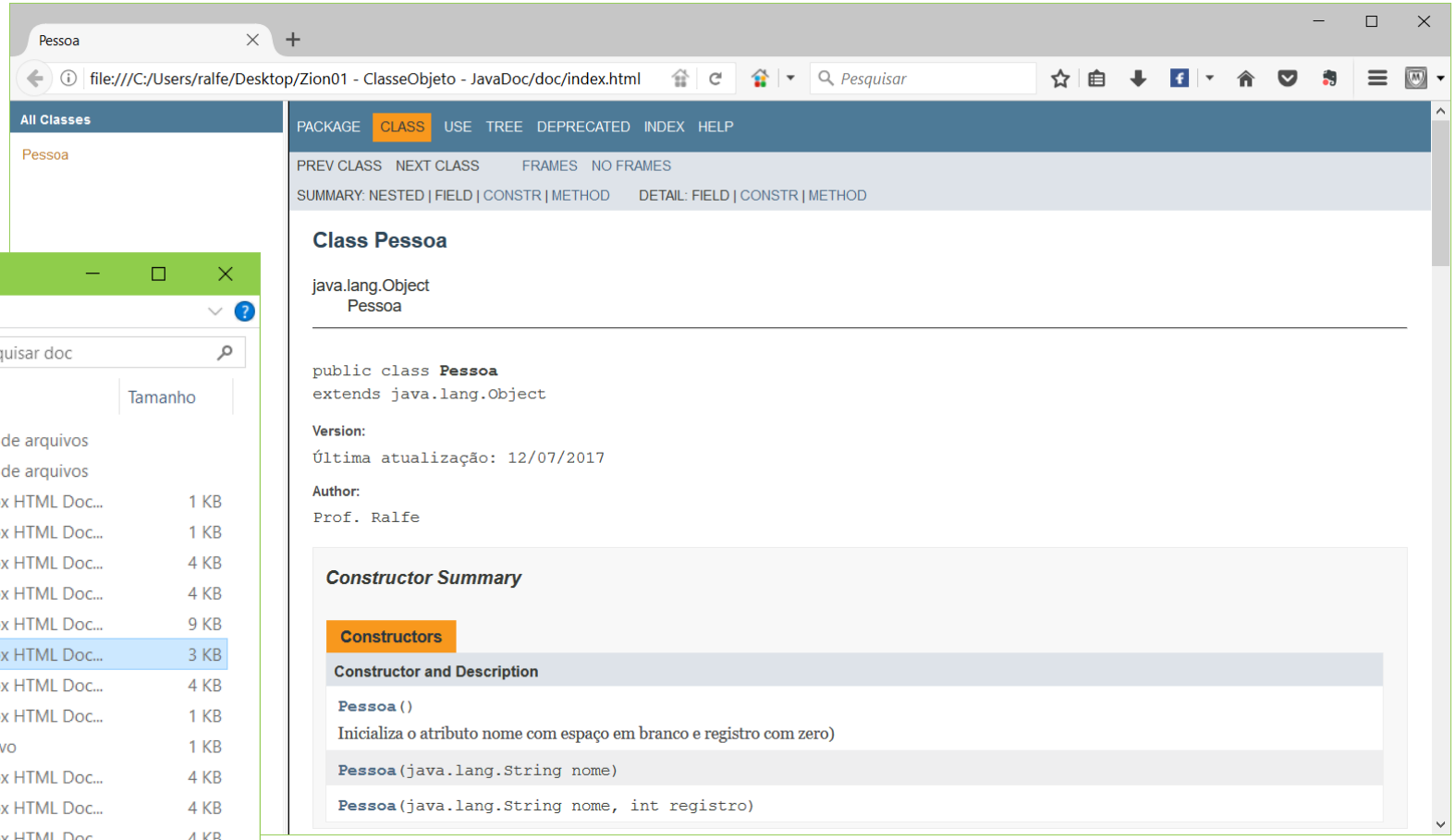
```java
    // Métodos de acesso
    /**
     * Define regras para leitura do atributo nome
     * @return o conteúdo do atributo nome
     */
    public String getNome() {
        return nome;
    }
    /**
     * Define regras para alteração do atributo nome
     * @param nome valor para alteração do atributo nome
     */
    public void setNome(String nome) {
        this.nome = nome;
    }
    /**
     * Define regras para leitura do atributo registro
     * @return o conteúdo do atributo registro
     */
    public int getRegistro() {
        return registro;
    }
    /**
     * Define regras para alteração do atributo registro
     * @param registro valor para alteração do atributo registro
     */
    public void setRegistro(int registro) {
        this.registro = registro;
    }
    // Funcionalidades dos objetos
    /**
     * Monta uma mensagem com o nome e o registro
     * @return a mensagem com o nome e o registro
     */
    public String identificar(){
        return "Nome: " + this.getNome() + "\nRegistro: " + this.getRegistro();
    }
```

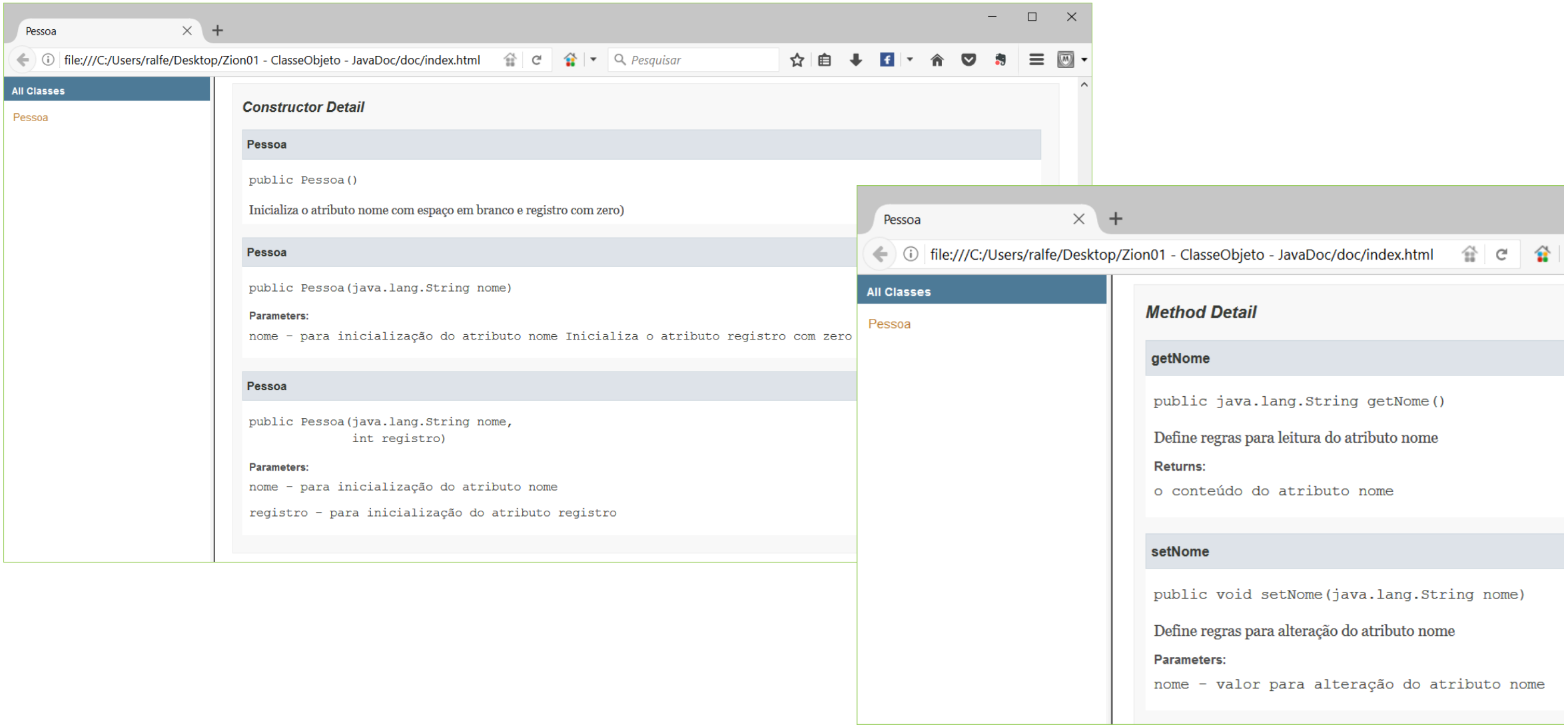
JAVADOC


workspaceLPOO - Java - Zion01 - ClasseObjeto - JavaDo
File Edit Source Refactor Navigate Search Project
Package Explore
Zion01 - Cla
src
(defau
Pess
Test
JRE Syste
doc
Open Project
Close Project
Build All Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Generate Javadoc...
Properties

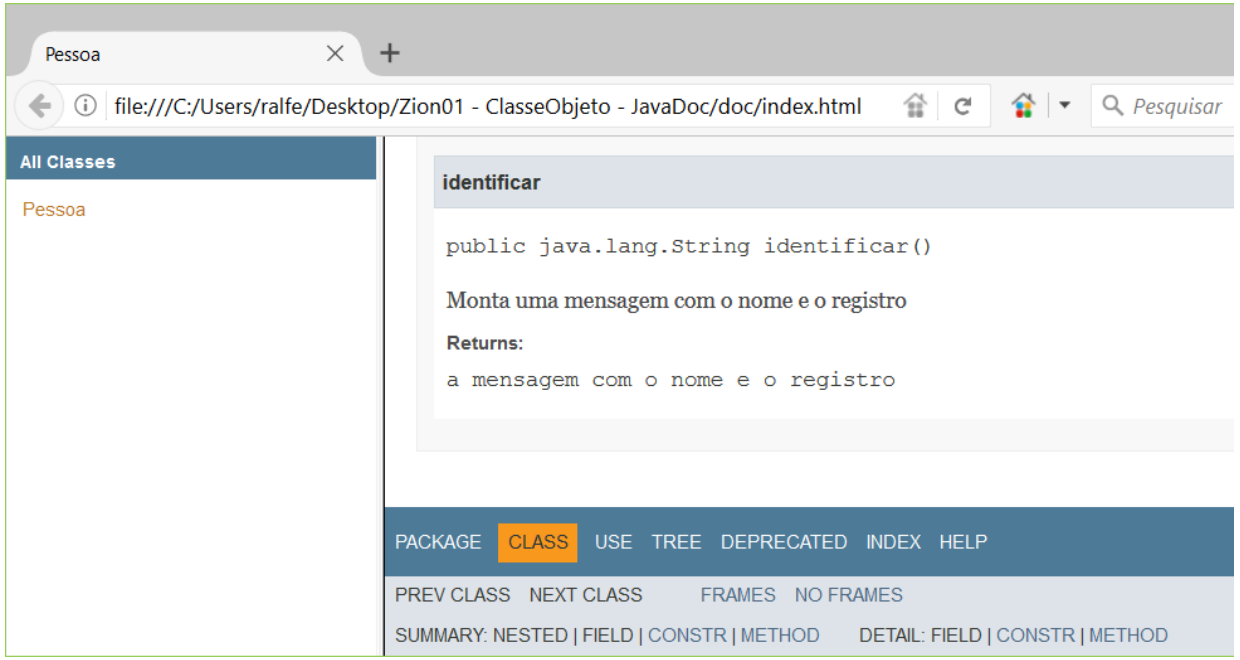

Generate Javadoc
Javadoc Generation
Select types for Javadoc generation.
Javadoc command:
C:\Program Files\Java\jdk1.8.0_91\bin\javadoc.exe
Configure...
Select types for which Javadoc will be generated:
Zion01 - ClasseObjeto - JavaDoc
src
(default package)
Teste.java
Pessoa.java
Create Javadoc for members with visibility:
Private
Package
Protected
Public
Public: Generate Javadoc for public classes and members.
Use standard doclet
Destination:
)O\workspaceLPOO\Zion01 - ClasseObjeto - JavaDoc\doc
Browse...
Use custom doclet
Doclet name:
Doclet class path:
< Back
Next >
Finish
Cancel


doc
Arquivo
Início
Compartilhar
Exibir
Zion01 - ClasseObjeto - JavaDoc › doc
Pesquisar doc
Acesso rápido
OneDrive
Anexos de email
Documentos
Imagens
Trabalho
Este Computador
Área de Trabalho
Documentos
Downloads
Imagens
Músicas
Vídeos
OS (C:)
Dados (D:)
Rede
Nome
Data de modificaç...
Tipo
Tamanho
class-use 13/07/2017 00:06 Pasta de arquivos
index-files 13/07/2017 00:06 Pasta de arquivos
allclasses-frame.html 12/07/2017 22:21 Firefox HTML Doc... 1 KB
allclasses-noframe.html 12/07/2017 22:21 Firefox HTML Doc... 1 KB
constant-values.html 12/07/2017 22:21 Firefox HTML Doc... 4 KB
deprecated-list.html 12/07/2017 22:21 Firefox HTML Doc... 4 KB
help-doc.html 12/07/2017 22:21 Firefox HTML Doc... 9 KB
index.html 12/07/2017 22:21 Firefox HTML Doc... 3 KB
overview-tree.html 12/07/2017 22:21 Firefox HTML Doc... 4 KB
package-frame.html 12/07/2017 22:21 Firefox HTML Doc... 1 KB
package-list 12/07/2017 22:21 Arquivo 1 KB
package-summary.html 12/07/2017 22:21 Firefox HTML Doc... 4 KB
package-tree.html 12/07/2017 22:21 Firefox HTML Doc... 4 KB
package-use.html 12/07/2017 22:21 Firefox HTML Doc... 4 KB
Pessoa.html 12/07/2017 22:21 Firefox HTML Doc... 13 KB
script.js 12/07/2017 22:21 Arquivo JavaScript 1 KB
stylesheet.css 12/07/2017 22:15 Documento de fol... 14 KB
17 itens
1 item selecionado 2,68 KB


Pessoa
file:///C:/Users/ralfe/Desktop/Zion01 - ClasseObjeto - JavaDoc/doc/index.html
Pesquisar
All Classes
Pessoa
PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
Class Pessoa
java.lang.Object
Pessoa
public class Pessoa
extends java.lang.Object
Version:
Última atualização: 12/07/2017
Author:
Prof. Ralfe
Constructor Summary
Constructors
Constructor and Description
Pessoa()
Inicializa o atributo nome com espaço em branco e registro com zero)
Pessoa(java.lang.String nome)
Pessoa(java.lang.String nome, int registro)


Pessoa
file:///C:/Users/ralfe/Desktop/Zion01 - ClasseObjeto - JavaDoc/doc/index.html
Pesquisar
All Classes
Pessoa
Constructor Detail
Pessoa
public Pessoa()
Inicializa o atributo nome com espaço em branco e registro com zero)
Pessoa
public Pessoa(java.lang.String nome)
Parameters:
nome - para inicialização do atributo nome Inicializa o atributo registro com zero
Pessoa
public Pessoa(java.lang.String nome,
int registro)
Parameters:
nome - para inicialização do atributo nome
registro - para inicialização do atributo registro
Pessoa
file:///C:/Users/ralfe/Desktop/Zion01 - ClasseObjeto - JavaDoc/doc/index.html
All Classes
Pessoa
Method Detail
getNome
public java.lang.String getNome()
Define regras para leitura do atributo nome
Returns:
o conteúdo do atributo nome
setNome
public void setNome(java.lang.String nome)
Define regras para alteração do atributo nome
Parameters:
nome - valor para alteração do atributo nome

JAVADOC

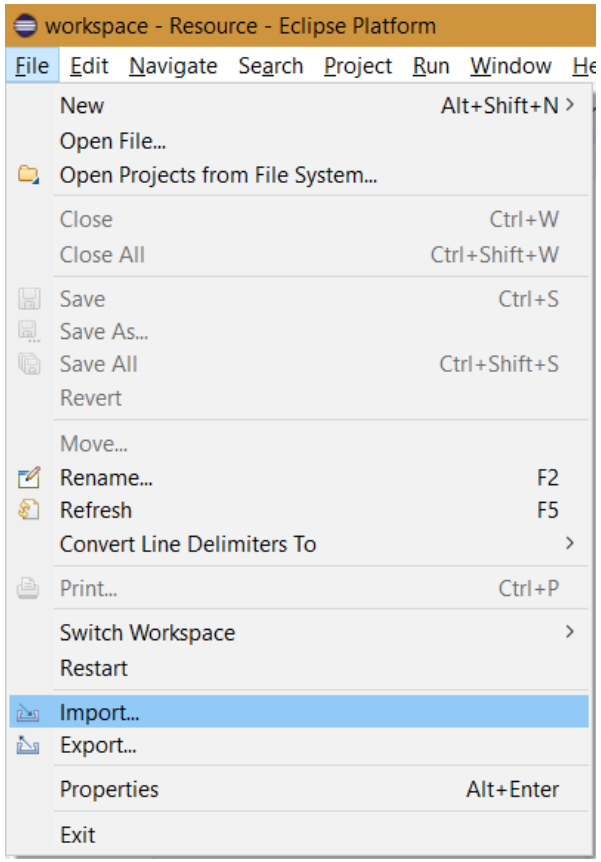


- Claro que esse exemplo tem uma quantidade desnecessária de comentários.
- A documentação deve detalhar somente as principais (e não óbvias) funcionalidades da classe.
- Perceba que é gerada documentação apenas para os elementos públicos da classe, ou seja, o que é necessário para sua utilização.

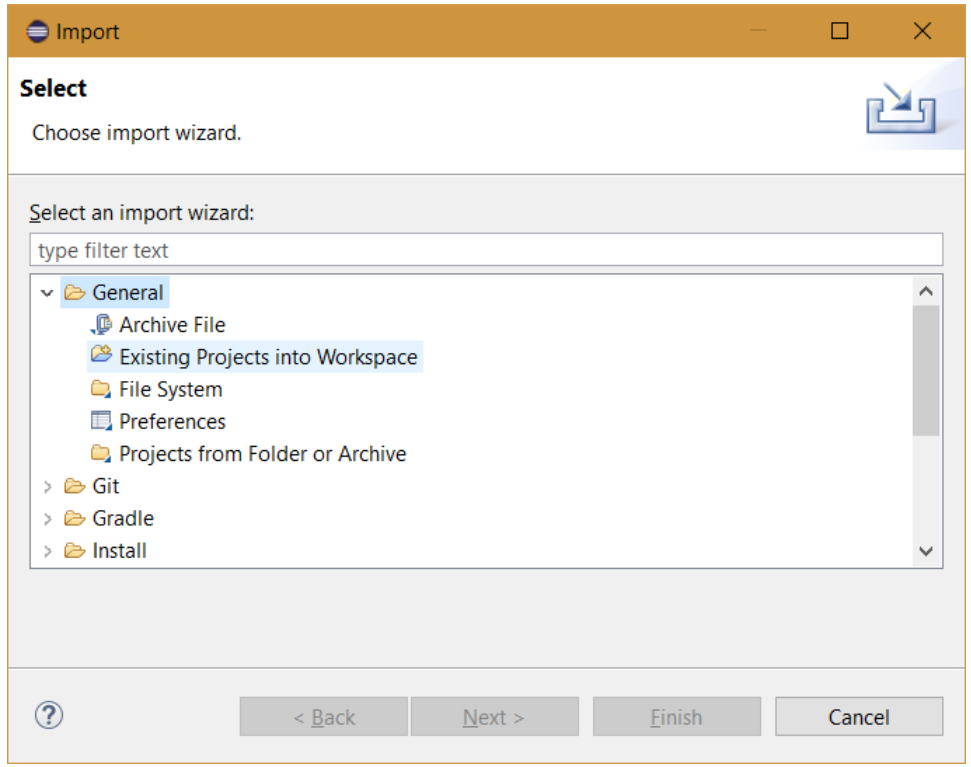
IMPORTAR UM PROJETO

- Para editar um Projeto Java já existente é necessário importá-lo no Eclipse.
- Antes da importação é preciso que:
  - O projeto já esteja na Workspace.
  - O Eclipse já esteja apontando para a Workspace (slides 46 e 47).

IMPORTAR UM PROJETO



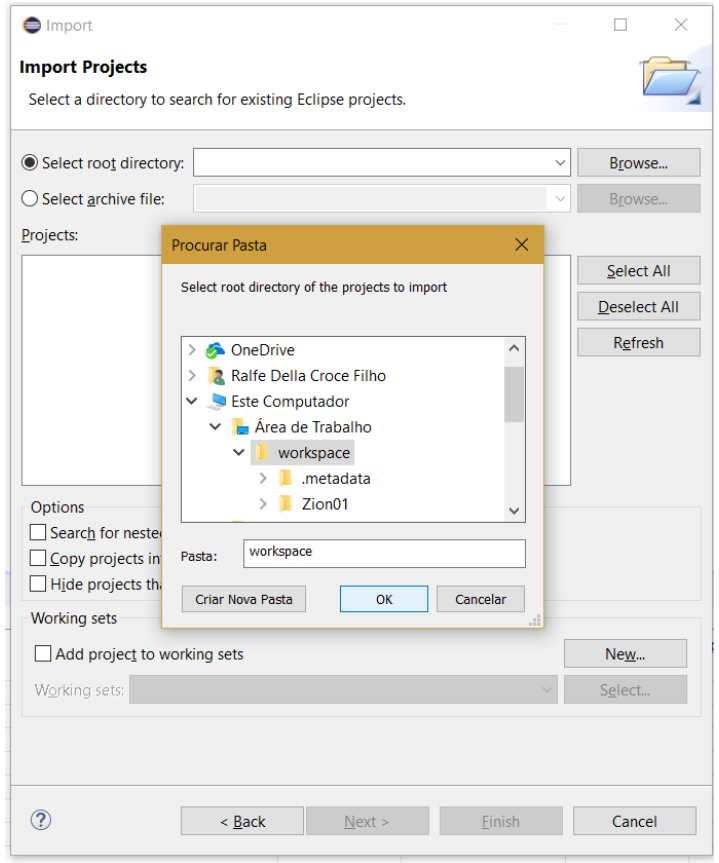
- Menu File > import

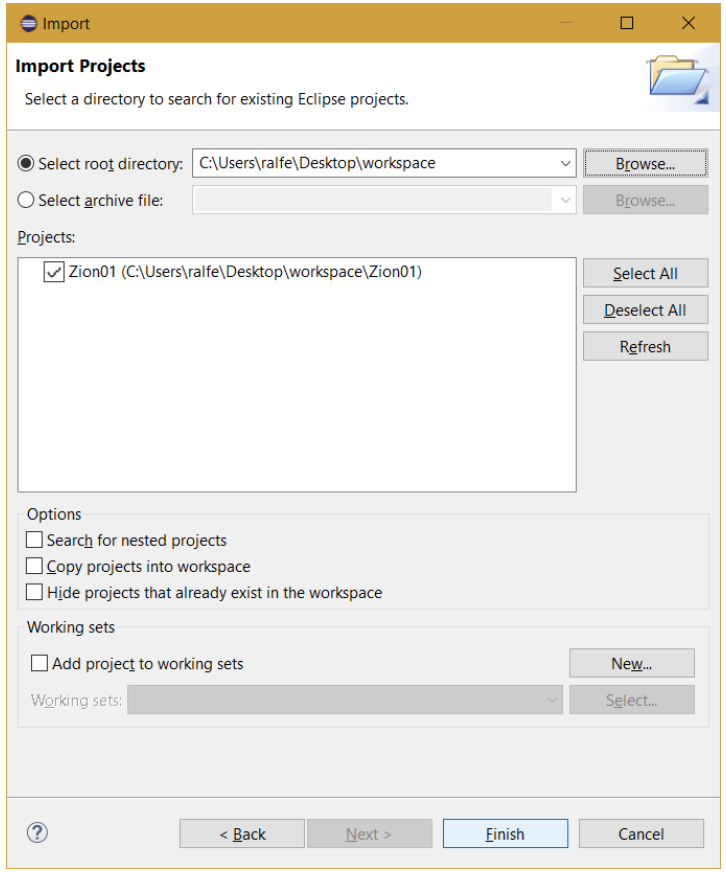


- General > Existing Projects into Workspace

IMPORTAR UM PROJETO

- Clique em Browse e em Ok na tela que será exibida (uma vez que a Worspace já foi selecionada previamente).
- Os projetos contidos na Workspace serão listados em Projects. Selecione o(s) projeto(s) desejado(s) e clique m Finish.



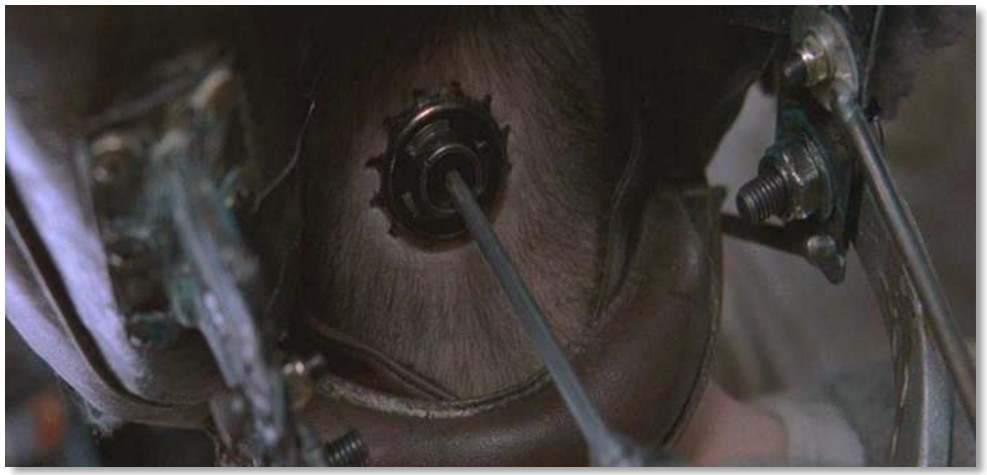


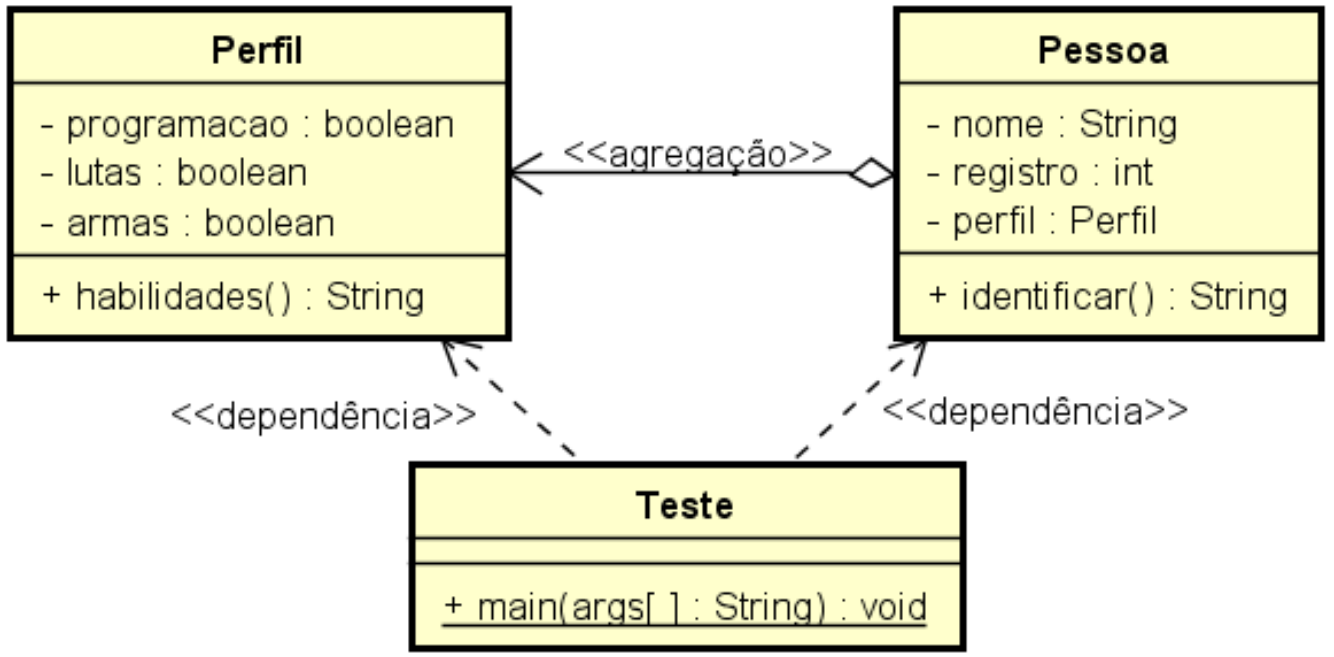
OBJETO COMO ATRIBUTO

- Os atributos podem ser definidos como tipos primitivos (como já visto), objetos ou vetores.
- Utilizar um objeto como atributo gera um relacionamento entre as classes de modelagem chamado de Associação.
- Existem dois tipos de Associação: Agregação e Composição.

PROJETO ZION

- Na agregação considera-se que as classes são independentes entre si e relacionam-se quando necessário.





Perfil.java

```java
public class Perfil {

    // Atributos
    private boolean programacao;
    private boolean lutas;
    private boolean armas;

    // Construtores
    public Perfil() {
        this(false, false, false);
    }

    public Perfil(boolean programacao, boolean lutas, boolean armas) {
        this.programacao = programacao;
        this.lutas = lutas;
        this.armas = armas;
    }
```

Pessoa.java

```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;
    private Perfil perfil;

    // Construtor
    public Pessoa() {
        this("", 0, new Perfil());
    }
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
        this.perfil = new Perfil();
    }
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
        this.perfil = new Perfil();
    }
    public Pessoa(String nome, int registro, Perfil perfil) {
        this.nome = nome;
        this.registro = registro;
        this.perfil = perfil;
    }
```

- Obs.: Se houver um construtor que recebe parâmetros para todos os atributos, o construtor que inicializa os atributos vazios pode realizar uma chamada a esse construtor pelo método this() com os valores vazios (linha 15 do exemplo acima).

Perfil.java

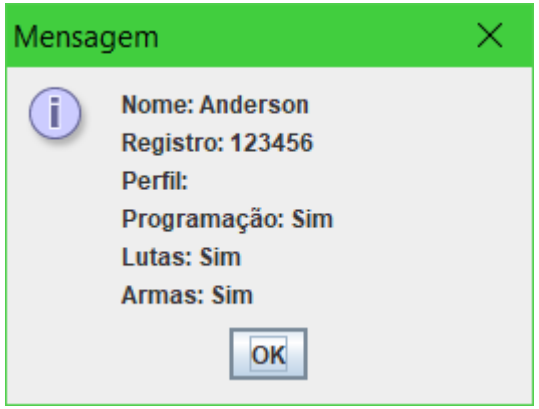
```
50      public String habilidades(){
51          /*
52          String mensagem = "";
53
54          if(this.isProgramacao()){
55              mensagem = mensagem + "\nProgramação: Sim";
56          }else{
57              mensagem = mensagem + "\nProgramação: Não";
58          }
59          if(this.isLutas()){
60              mensagem = mensagem + "\nLutas: Sim";
61          }else{
62              mensagem = mensagem + "\nLutas: Não";
63          }
64          if(this.isArmas()){
65              mensagem = mensagem + "\nArmas: Sim";
66          }else{
67              mensagem = mensagem + "\nArmas: Não";
68          }
69
70          return mensagem;
71          */
72
73          // Com o uso do operador condicional ternário a implementação
74          // do método ficará bem mais enxuta
75          return "Programação: " + (this.isProgramacao() ? "Sim" : "Nao") +
76                  "\nLutas: " + (this.isLutas() ? "Sim" : "Nao" )+
77                  "\nArmas: " + (this.isArmas() ? "Sim" : "Nao" );
78      }
```

Teste.java

```
1  import javax.swing.JOptionPane;
2  public class Teste {
3      public static void main(String[] args) {
4
5          // Criação de um objeto do tipo Perfil
6          Perfil perfil = new Perfil(true, true, true);
7
8          // Criação de um objeto do tipo Pessoa
9          Pessoa pessoa = new Pessoa("Anderson", 123456, perfil);
10
11         // Teste do método identificar
12         JOptionPane.showMessageDialog(null, pessoa.identificar());
13     }
14 }
```

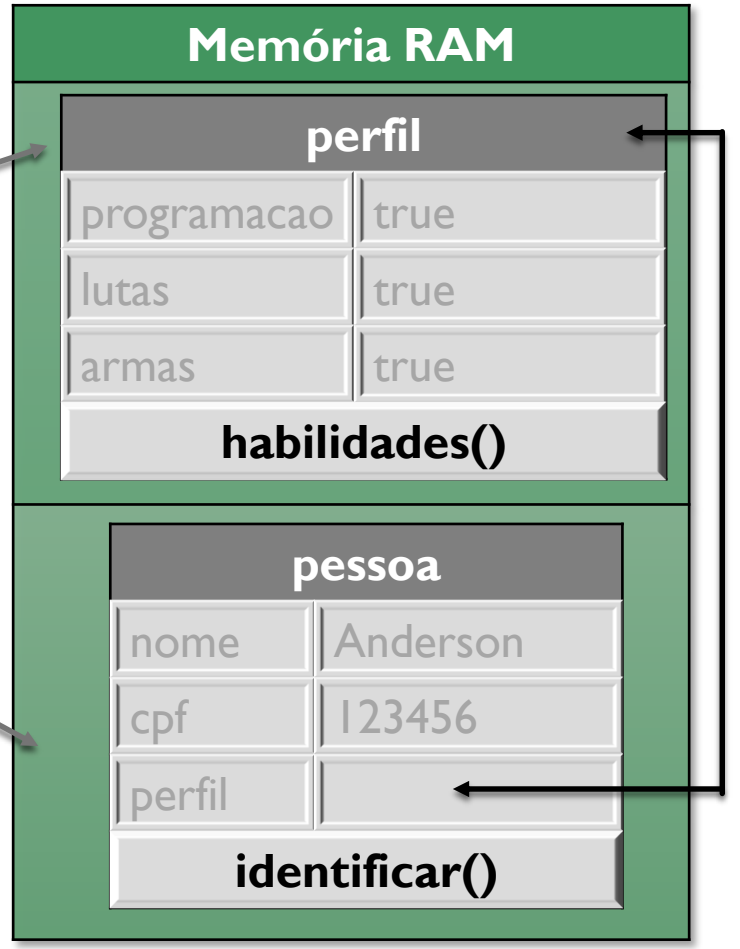
Pessoa.java

```
48      public String identificar(){
49          return "Nome: " + this.getNome() +
50                  "\nRegistro: " + this.getRegistro() +
51                  "\nPerfil:" +
52                  "\n" + this.perfil.habilidades();
53      }
```

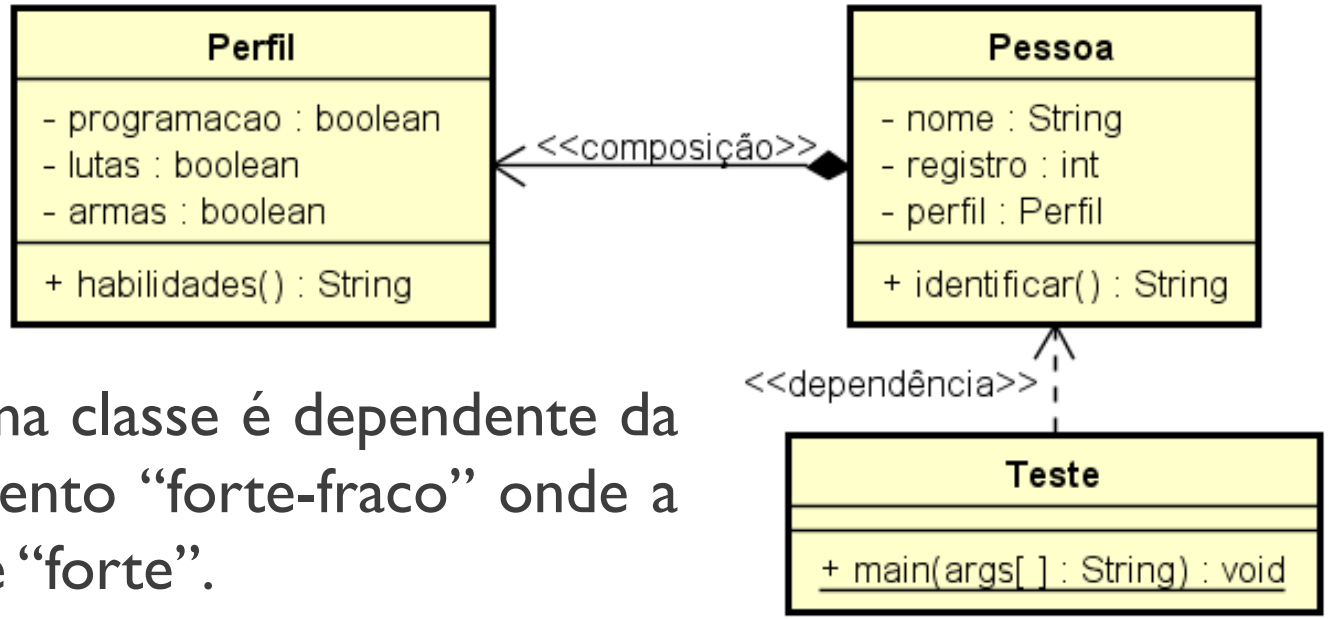


Teste.java

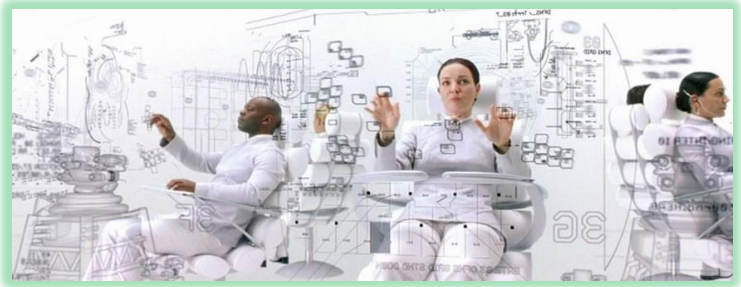
```java
import javax.swing.JOptionPane;
public class Teste {
    public static void main(String[] args) {

        // Criação de um objeto do tipo Perfil
        Perfil perfil = new Perfil(true, true, true);

        // Criação de um objeto do tipo Pessoa
        Pessoa pessoa = new Pessoa("Anderson", 123456, perfil);

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());
    }
}
```



PROJETO ZION



- Na composição considera-se que uma classe é dependente da outra, ou seja, existe um relacionamento "forte-fraco" onde a classe "fraca" não existe sem a classe "forte".
- Nesse exemplo esse relacionamento é mais coerente já que o perfil refere-se a uma pessoa e perde o sentido isoladamente.



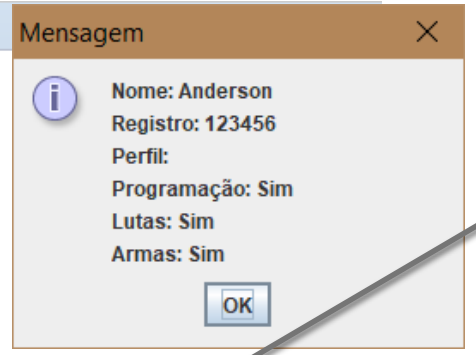
Pessoa.java

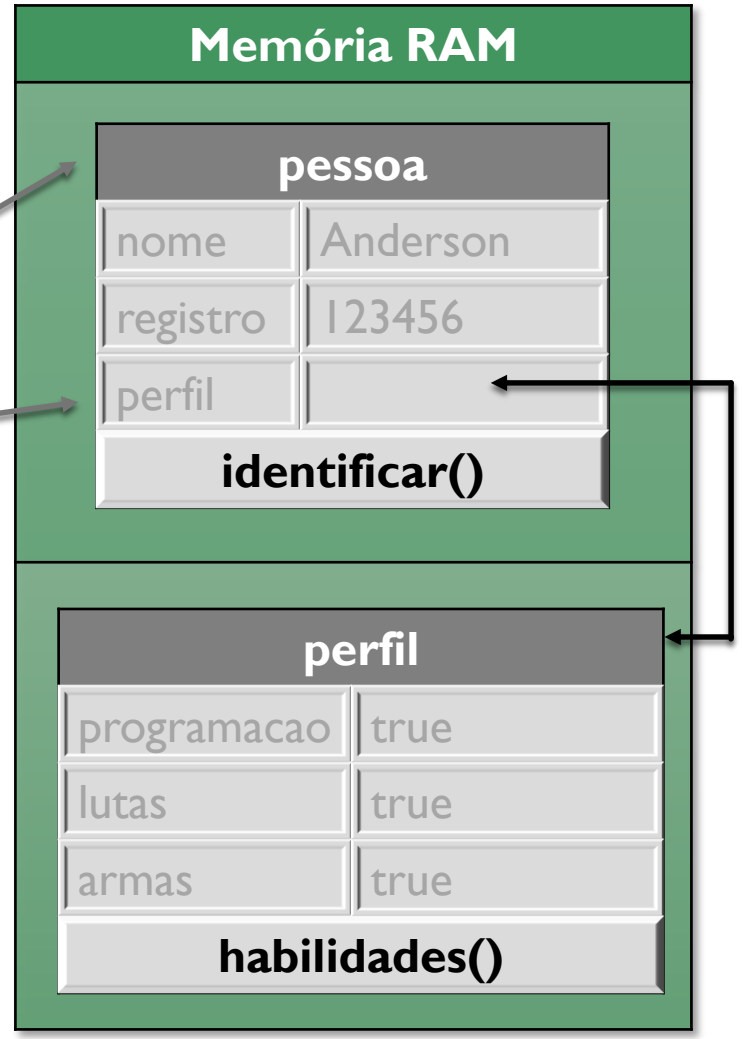
```java
public class Pessoa {

    // Atributos
    private String nome;
    private int registro;
    private Perfil perfil;

    // Construtor
    public Pessoa() {
        this.nome = "";
        this.registro = 0;
        this.perfil = new Perfil();
    }
    public Pessoa(String nome) {
        this.nome = nome;
        this.registro = 0;
        this.perfil = new Perfil();
    }
    public Pessoa(String nome, int registro) {
        this.nome = nome;
        this.registro = registro;
        this.perfil = new Perfil();
    }
    public Pessoa(String nome, int registro,
                  boolean programacao, boolean lutas, boolean armas) {
        this.nome = nome;
        this.registro = registro;
        this.perfil = new Perfil(programacao, lutas, armas);
    }
```

- Como um objeto Perfil agora será criado somente por meio da classe Pessoa as possíveis inicializações de seus atributos serão efetuadas por um construtor de Pessoa.

Teste.java

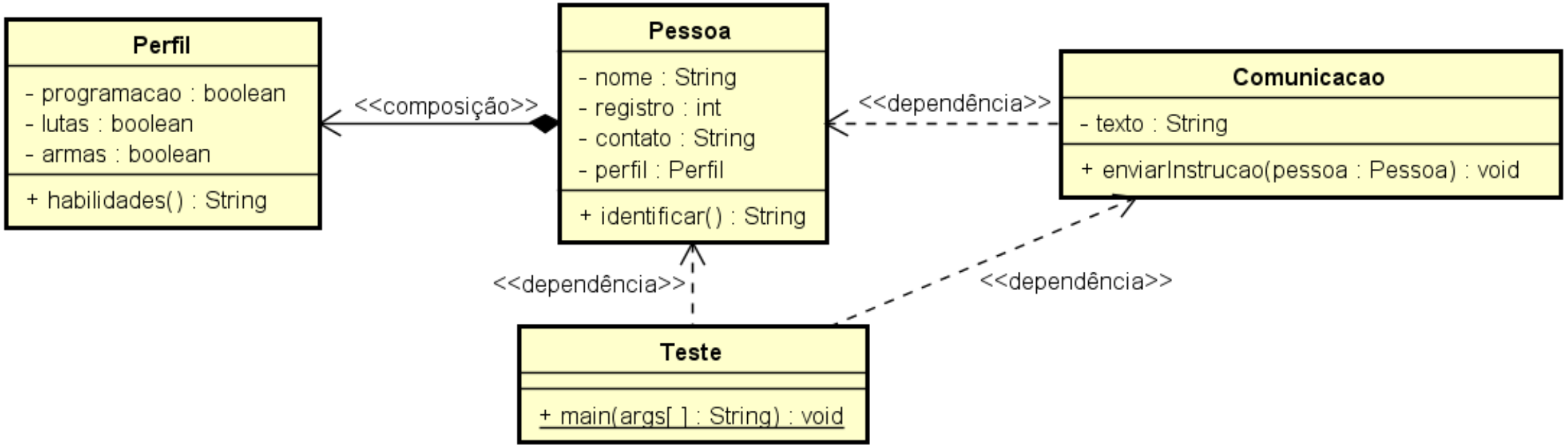
```java
import javax.swing.JOptionPane;

public class Teste {

    public static void main(String[] args) {

        // Criação de um objeto do tipo Pessoa
        // inicializando os atributos de pessoa e perfil
        Pessoa pessoa = new Pessoa("Anderson", 123456, true, true, true);

        // As alterações de perfil podem ser realizadas
        // por meio dos métodos setters do atributo perfil
        pessoa.getPerfil().setProgramacao(true);
        pessoa.getPerfil().setLutas(true);
        pessoa.getPerfil().setArmas(true);

        // Teste do método identificar
        JOptionPane.showMessageDialog(null, pessoa.identificar());

        // As leituras de perfil podem ser realizadas
        // por meio dos métodos getters do atributo perfil
        JOptionPane.showMessageDialog(null, pessoa.getPerfil().isProgramacao());
        JOptionPane.showMessageDialog(null, pessoa.getPerfil().isLutas());
        JOptionPane.showMessageDialog(null, pessoa.getPerfil().isArmas());
        // Assim como o método habilidades
        JOptionPane.showMessageDialog(null, pessoa.getPerfil().habilidades());
    }
}
```





OBJETO POR PARÂMETRO

- Projeto Zion

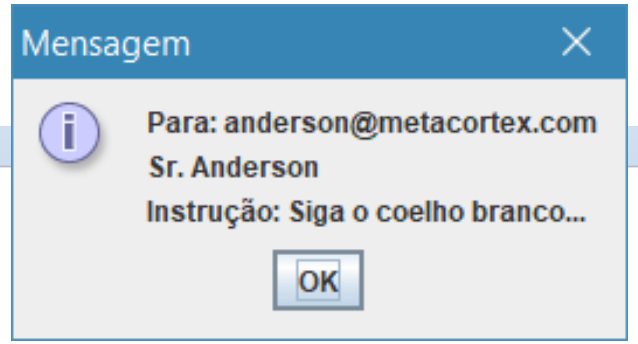


Comunicacao.java

```java
30    public String enviarInstrucoes(Pessoa pessoa){
31
32        String mensagem = "Para: " + pessoa.getContato();
33
34        if(pessoa.getPerfil().isProgramacao()){
35            mensagem = mensagem + "\nSr. " + pessoa.getNome() + "\nInstrução: " + this.getTexto();
36        }else{
37            mensagem = mensagem + "\nA ignorância é uma benção.";
38        }
39        return mensagem;
40    }
```

Teste.java

```java
6  import javax.swing.JOptionPane;
7  public class Teste {
8      public static void main(String[] args) {
9
10         // Criação de um objeto do tipo Pessoa
11         Pessoa pessoa = new Pessoa("Anderson", 123456, "anderson@metacortex.com", true, true, true);
12
13         // Criação de um objeto do tipo Informativo
14         Comunicacao comunicado = new Comunicacao("Siga o coelho branco...");
15
16         // Teste do método enviarPromocoes
17         JOptionPane.showMessageDialog(null, comunicado.enviarInstrucoes(pessoa));
18     }
19 }
```

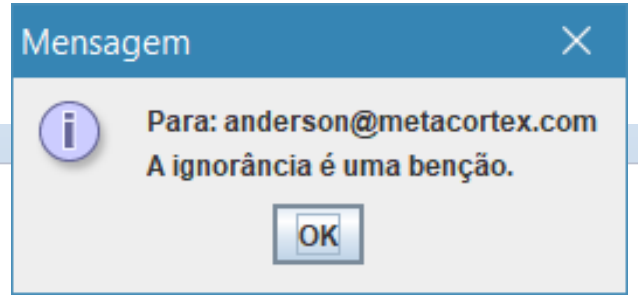


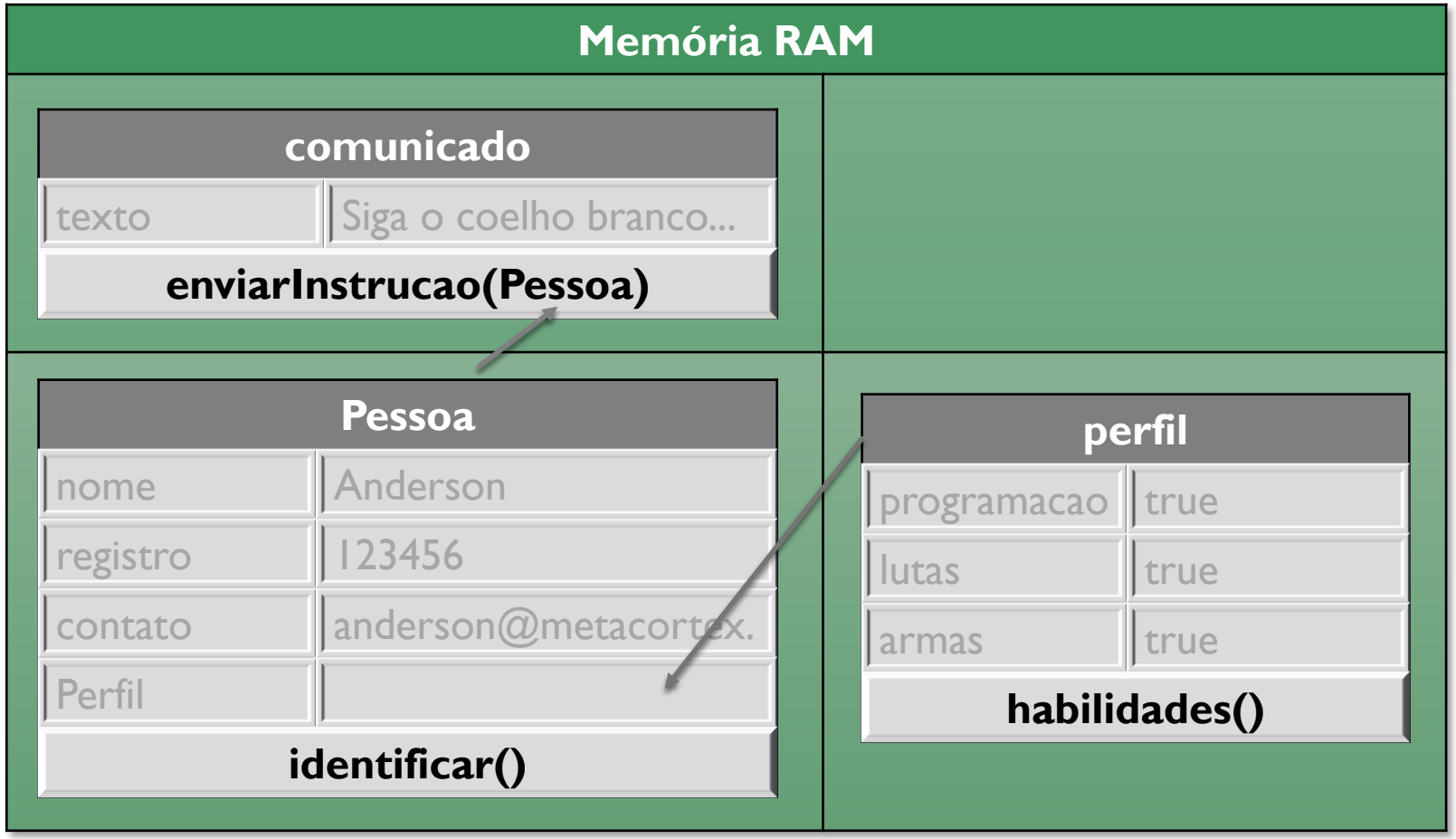
Comunicacao.java

```java
30    public String enviarInstrucoes(Pessoa pessoa){
31
32        String mensagem = "Para: " + pessoa.getContato();
33
34        if(pessoa.getPerfil().isProgramacao()){
35            mensagem = mensagem + "\nSr. " + pessoa.getNome() + "\nInstrução: " + this.getTexto();
36        }else{
37            mensagem = mensagem + "\nA ignorância é uma benção.";
38        }
39        return mensagem;
40    }
```

Teste.java

```java
6  import javax.swing.JOptionPane;
7  public class Teste {
8      public static void main(String[] args) {
9
10         // Criação de um objeto do tipo Pessoa
11         Pessoa pessoa = new Pessoa("Anderson", 123456, "anderson@metacortex.com", false, true, true);
12
13         // Criação de um objeto do tipo Informativo
14         Comunicacao comunicado = new Comunicacao("Siga o coelho branco...");
15
16         // Teste do método enviarPromocoes
17         JOptionPane.showMessageDialog(null, comunicado.enviarInstrucoes(pessoa));
18     }
19 }
```




Memória RAM
comunicado
texto
Siga o coelho branco...
enviarInstrucao(Pessoa)
Pessoa
nome
Anderson
registro
123456
contato
anderson@metacortex.
Perfil
identificar()
perfil
programacao
true
lutas
true
armas
true
habilidades()

ADENDO

CONTEXTUALIZAÇÃO

CONTEXTUALIZAÇÃO

- A Orientação a Objetos possui uma cadeia de conceitos que estruturam, organizam e padronizam softwares. Todos esses conceitos são implementáveis, ou seja, eles foram (e são) criados para aplicação prática no desenvolvimento de softwares de acordo com esse paradigma.
- Contextualizar a aplicação prática desses conceitos é, didaticamente, o melhor caminho para a análise e entendimento de seus efetivos objetivos e importância. Com essa intenção, utilizarei dois caminhos...

CONTEXTUALIZAÇÃO – MATRIX

- Na apresentação dos conceitos utilizarei exemplos baseados no universo da trilogia Matrix (1999, 2003 e 2003), filme dos irmãos Larry e Andy Wachowski (agora irmãs Lana e Lilly Wachowski). O motivo? Primeiro porque é um dos meus filmes favoritos, segundo porque, de forma extremamente visionária e competente, o filme aborda tanto a tecnologia (especificamente a área de softwares) quanto questões filosóficas fundamentais (pois é, se você achava que era "apenas" um filme de ação e ficção (ganhador de 4 óscares) acho que você não entendeu o filme).
- Obviamente, para o entendimento do conteúdo não é necessário assistir/conhecer o filme, mas, se você não assistiu, creio que quem está perdendo é você...

CONTEXTUALIZAÇÃO

- A outra forma de contextualização será por meio de vários exercícios utilizando simulações de softwares que gerenciam, por exemplo, uma livraria, um controle bancário, uma agência de turismo, um controle escolar e uma imobiliária.
- **Observação importante:** realizar as atividades práticas é fundamental e indispensável. Será MUITO mais difícil (senão impossível) a assimilação dos conteúdos sem a prática.

ESTEREÓTIPOS NO MODELO

- Um *stereotype* (anotação entre <<>>) é um elemento que identifica a finalidade de outros elementos do modelo. Existe um conjunto padrão de estereótipos que podem ser aplicados e são utilizados para refinar o significado de um elemento do modelo.
- Utilizarei os estereótipos fora dos padrões previstos na UML para identificar onde os conceitos de Orientação a Objetos estão sendo aplicados no modelo com intenções puramente didáticas.