AULA 05 – ORIENTAÇÃO A OBJETOS

Ao término dessa aula você terá aprendido:

- ✓ *Classes, objetos e métodos;*
- ✓ *Métodos construtores;*
- ✓ *O paradigma da orientação a objetos.*

Um dos grandes diferenciais da programação orientada a objetos, em relação a outros paradigmas de programação, que também permitem a definição de estruturas e operações sobre essas estruturas, está no conceito de herança, mecanismo através do qual definições existentes podem ser facilmente estendidas. Além da herança, outros conceitos importantes, como interfaces, classes abstratas, polimorfismo e encapsulamento, formam os pilares deste paradigma.

5.1 CLASSES, OBJETOS E MÉTODOS

De forma alegórica, tomando como base a confecção de pães, pode-se compreender uma ***classe*** como sendo a ***forma*** de determinado tipo de pão. Ela servirá de modelo/base para os pães que serão feitos. Já os ***objetos***, podem ser considerados como os ***pães*** que são gerados através da forma.

Classe ➔

Objeto ➔

Em Java, classes são definidas por meio do uso da palavra-chave ***class***. Para definir uma classe, utiliza-se a construção:

[modif] class NomeDaClasse {

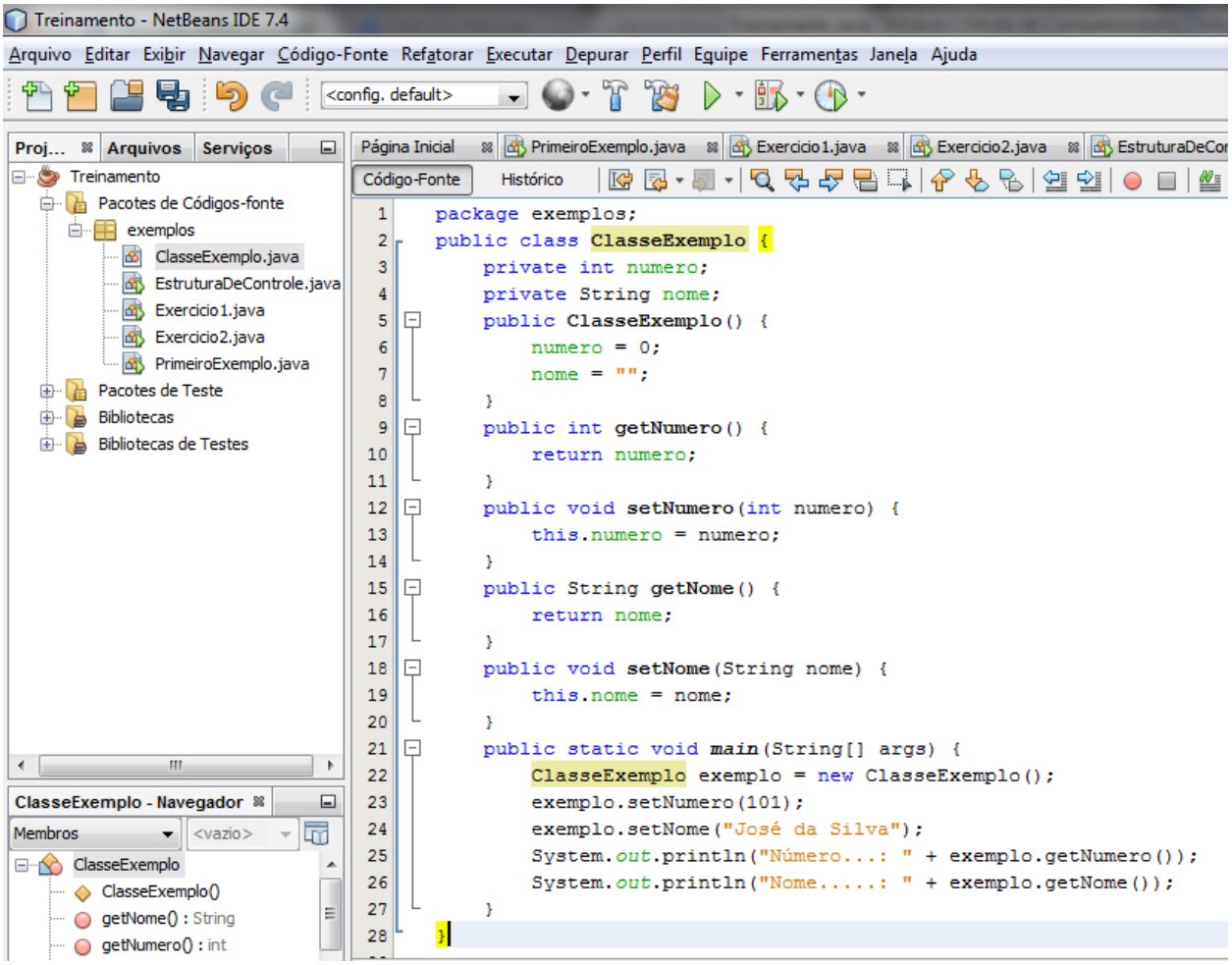
// corpo da classe...

}

A primeira linha é um comando que inicia a declaração da classe. Após a palavra-chave ***class***, segue-se o nome da classe, que deve ser um identificador válido para a linguagem. O modificador ***modif*** é opcional; se presente, pode ser uma combinação de ***public*** e ***abstract*** ou ***final***.

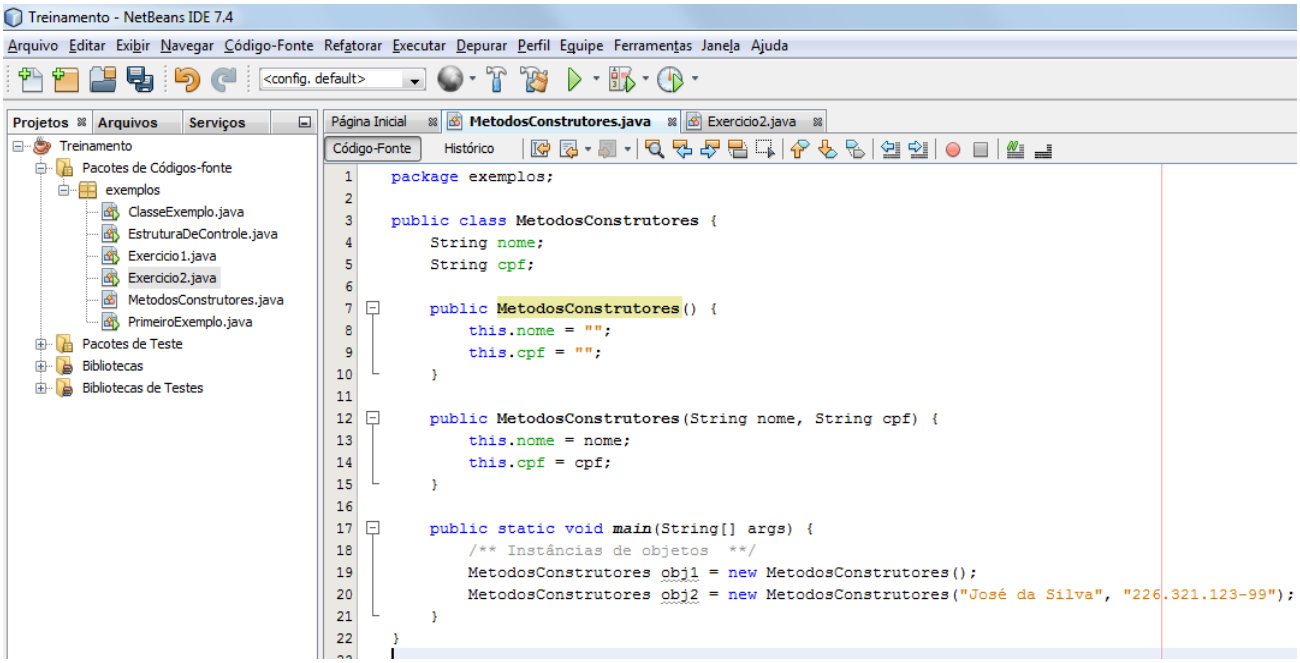
A definição da classe, propriamente dita, está entre as chaves ***{*** e ***}*** que delimitam blocos na linguagem Java. Este corpo da classe usualmente obedece à seguinte sequência:

1. As variáveis de classe (definidas como static), ordenadas segundo sua visibilidade: iniciando pelas **public**, seguidos pelas **protected**, pelas com visibilidade padrão (sem modificador) e, finalmente, pelas **private**;
2. Os **atributos** (ou variáveis de instância) dos objetos dessa classe, seguindo a mesma ordenação, segundo a visibilidade definida para as variáveis de classe;
3. Os **construtores** de objetos dessa classe;
4. Os **métodos** da classe, geralmente agrupados por funcionalidade.



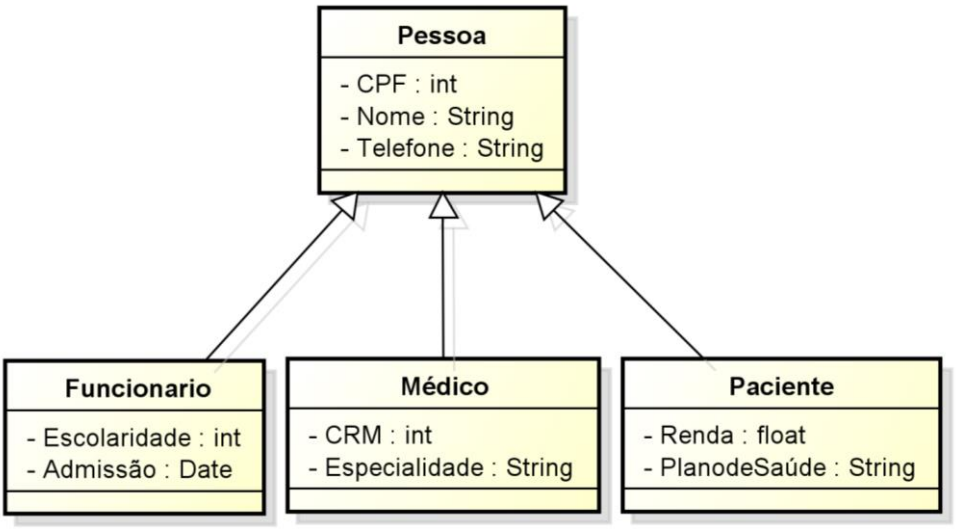
5.2 MÉTODOS CONSTRUTORES

Construtores são métodos (funções) que, ao instanciarmos (criarmos) um objeto de uma classe, são executados automaticamente. Esses métodos podem receber ou não parâmetros, e também podem ser sobrecarregados. Possuem, obrigatoriamente, o mesmo nome da classe que o definem, e não retornam valores. O exemplo abaixo cria uma classe ***MetodosConstrutores*** com dois construtores: um que não recebe parâmetros e outro que recebe dois parâmetros, que também são os atributos declarados na classe: ***nome*** e ***cpf***.



5.3 ABSTRAÇÃO

Na orientação a objetos, **abstração** consiste em generalizar as características (atributos) e comportamentos (métodos) de uma determinada classe, para ser usada como referência a demais classes que dela estender. Não é possível instanciar objetos de uma classe abstrata. Como exemplo, considere as classes ***Funcionario***, ***Medico*** e ***Paciente***. Essas classes possuem atributos em comum, como cpf, nome, telefone etc. Com isso, cria-se uma classe abstrata, por exemplo, ***Pessoa***, contendo tais atributos (generalização), e nas classes ***Funcionario***, ***Médico*** e ***Paciente***, cria-se apenas os atributos específicos (especialização). Esse conceito é demonstrado na próxima figura.

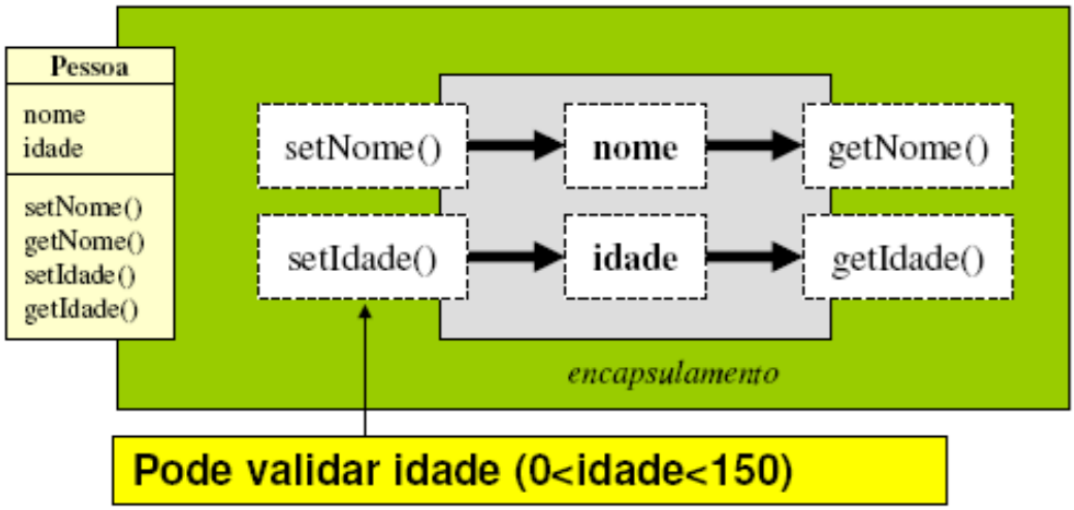


5.4 ENCAPSULAMENTO

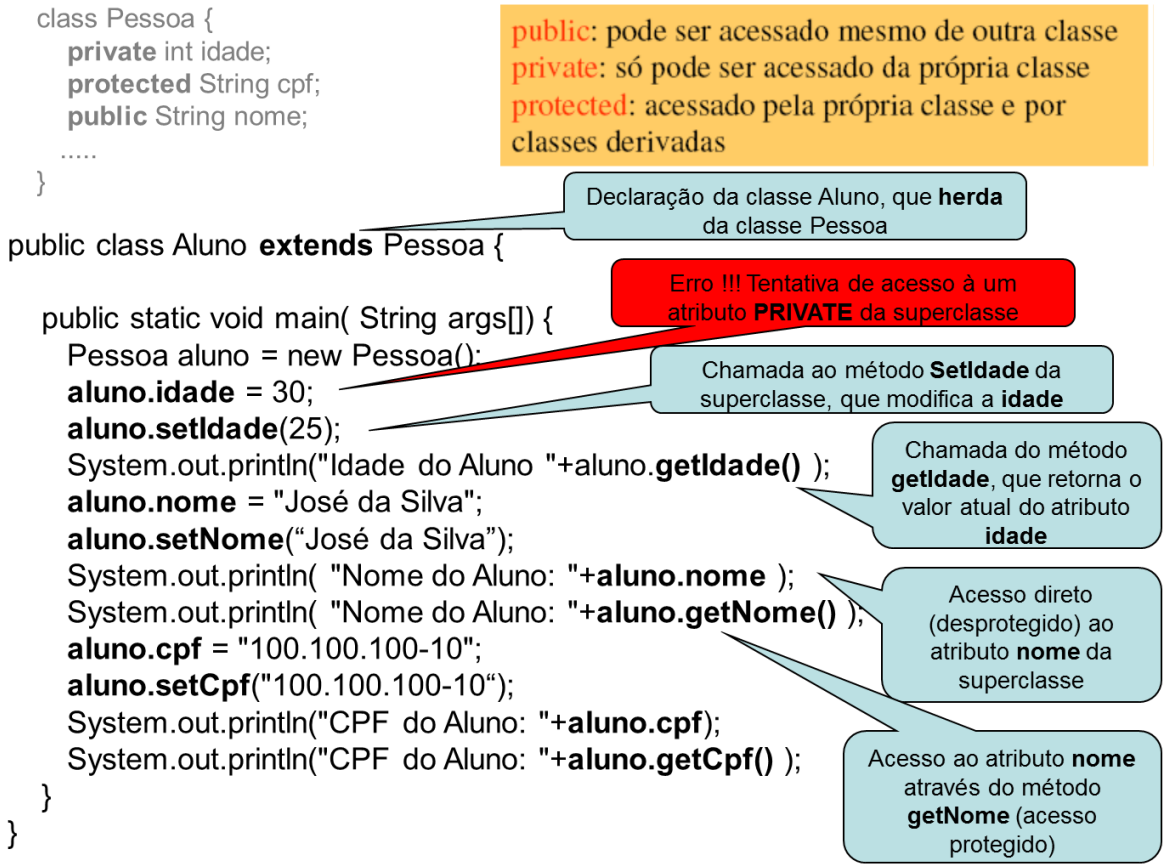
Encapsulamento, também referido como *esconder informação*, consiste em separar os aspectos externos de um objeto, que são acessíveis a outros objetos, dos detalhes internos de implementação do objeto, os quais permanecem escondidos dos outros objetos. O uso de encapsulamento evita que um programa torne-se tão interdependente, que uma pequena mudança tenha grandes efeitos colaterais.

Em um processo de encapsulamento os atributos das classes são do tipo **private**. Para acessar esses tipos de modificadores é necessário criar métodos **setters** e **getters**.

Por entendimento, os métodos ***setters*** servem para alterar a informação de uma propriedade de um objeto, e os métodos ***getters***, para retornar o valor dessa propriedade.



O próximo exemplo demonstra a utilização de encapsulamento:



5.5 HERANÇA

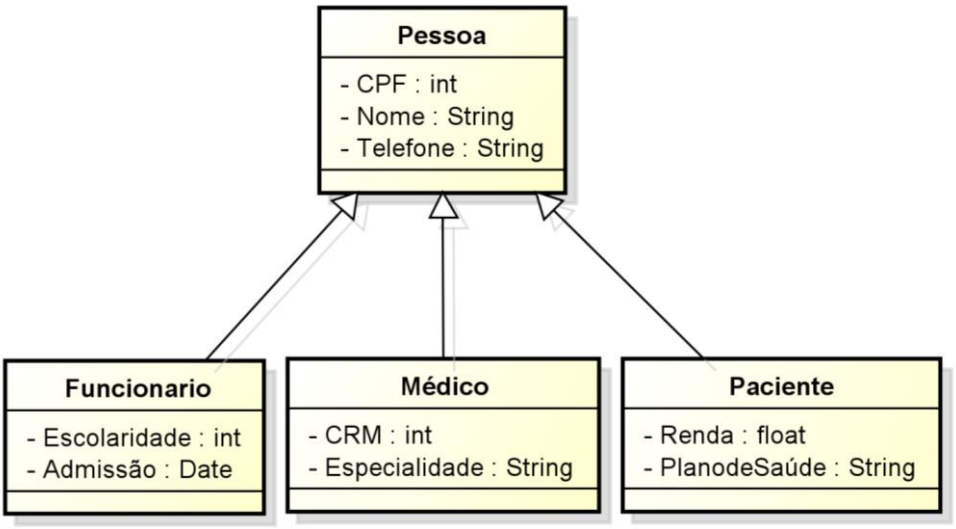
Herança é o mecanismo pelo qual elementos mais específicos incorporam a estrutura e comportamento de elementos mais gerais.

Uma classe derivada ***herda*** a estrutura de atributos e métodos de sua classe ***base***, mas pode seletivamente:

- ✓ Adicionar novos métodos;
- ✓ Estender a estrutura de dados;
- ✓ Redefinir a implementação de métodos já existentes.

Uma classe ***pai*** ou ***superclasse*** proporciona a funcionalidade, que é comum a todas as suas classes ***derivadas***, ***filhas*** ou ***subclasses***, enquanto uma classe derivada proporciona a funcionalidade adicional que especializa seu comportamento.

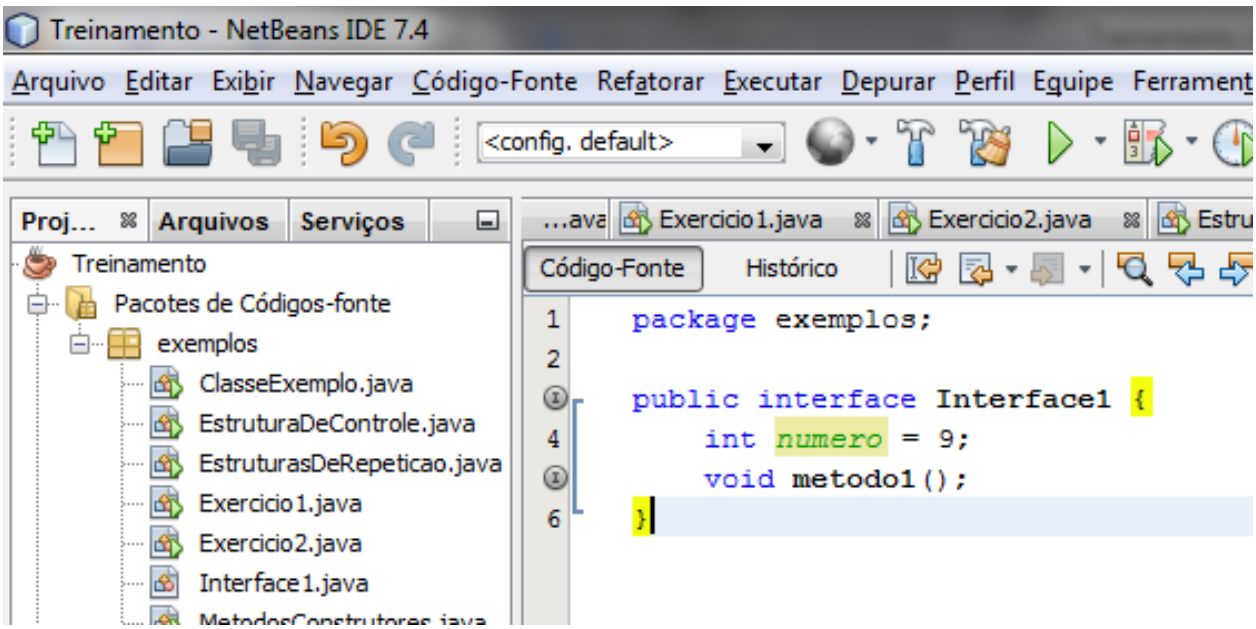
A próxima figura é a mesma utilizada no tópico 4.3 – “abstração”, e também exemplifica o conceito de herança.



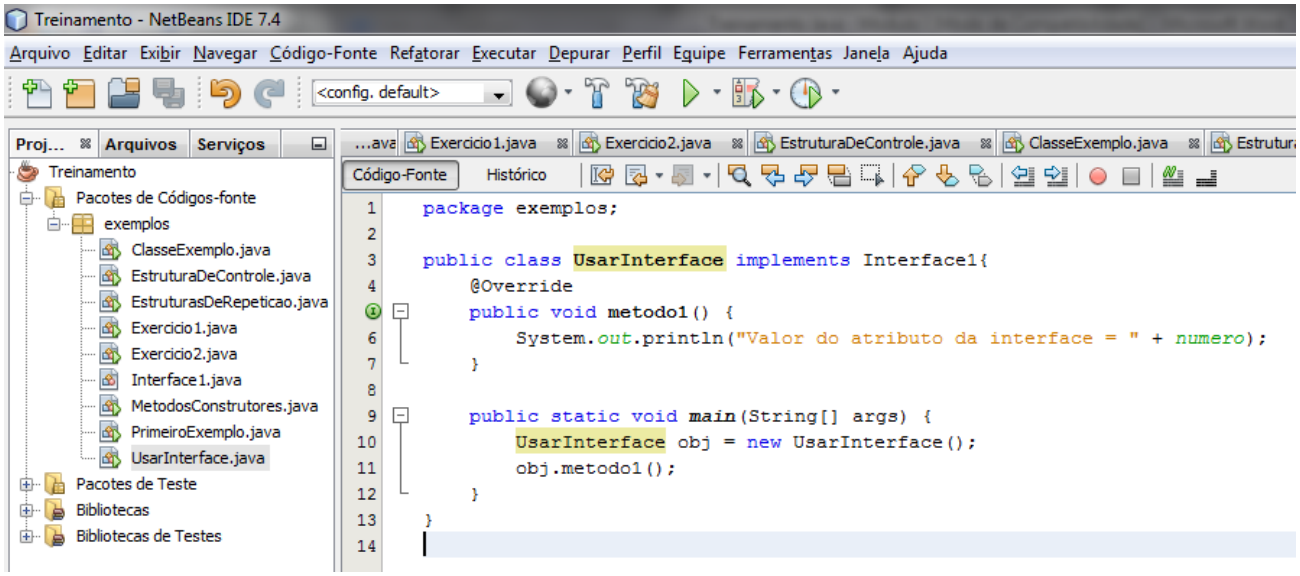
5.6 INTERFACES

Java também oferece outra estrutura, denominada interface, com sintaxe similar a de classes, mas possuindo apenas a especificação da funcionalidade que uma classe deve conter, sem determinar como essa funcionalidade deve ser implementada. Uma interface Java é uma classe abstrata para a qual todos os métodos são implicitamente ***abstract*** e ***public***, e todos os atributos são implicitamente ***static*** e ***final***. Em outros termos, uma interface Java implementa uma "classe abstrata pura".

A sintaxe para a declaração de uma interface é similar àquela para a definição de classes, porém seu corpo define apenas assinaturas de ***métodos*** e ***constantes***. Por exemplo, para definir uma interface ***Interface1,*** que declara um método ***metodo1***, sem argumentos e sem valor de retorno, e uma constante ***numero***, com valor igual a 9. A sintaxe é:



Para criar uma classe que implemente essa interface, a sintaxe é:

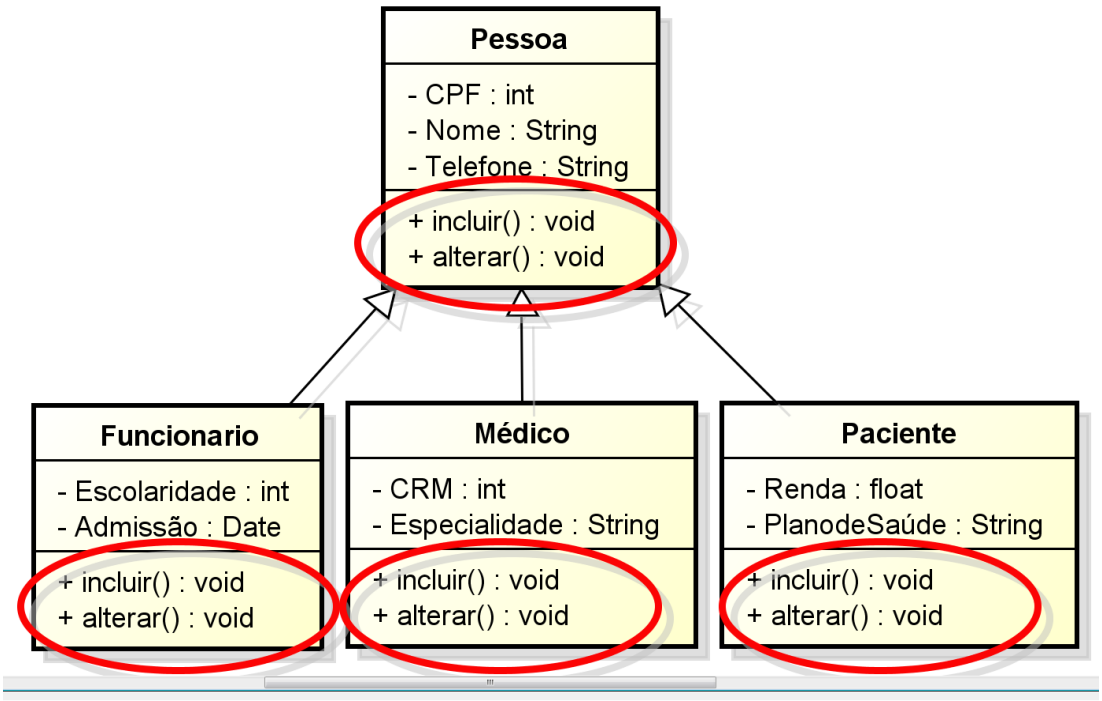


5.7 POLIMORFISMO

Polimorfismo é uma operação que pode assumir múltiplas formas. Trata-se da propriedade segundo a qual uma operação pode comportar-se diferentemente em classes diferentes.

O polimorfismo é o responsável pela extensibilidade em programação orientada a objetos, promovendo o reuso.

O próximo exemplo ilustra essa técnica. Em todas as classes têm-se dois métodos: ***incluir()*** e ***alterar()***, mas em cada uma delas esses métodos podem ter comportamentos (implementações) distintos.



5.8 EXEMPLO

As classes abaixo utilizam os pilares do paradigma de orientação a objetos: classe abstrata, polimorfismo, encapsulamento, construtores, herança e sobrecarga de métodos.

Classe Pessoa

```
1  package exemplos;
2
3  public abstract class Pessoa {
4      private int cpf;
5      private String nome;
6      private String telefone;
7
8      public int getCpf() {
9          return cpf;
10     }
11
12     public void setCpf(int cpf) {
13         this.cpf = cpf;
14     }
15
16     public String getNome() {
17         return nome;
18     }
19
20     public void setNome(String nome) {
21         this.nome = nome;
22     }
23
24     public String getTelefone() {
25         return telefone;
26     }
27
28     public void setTelefone(String telefone) {
29         this.telefone = telefone;
30     }
31
32     public void imprimir() {
33         System.out.println("CPF.............: " + getCpf());
34         System.out.println("Nome............: " + getNome());
35         System.out.println("Telefone........: " + getTelefone());
36     }
37 }
```

Classe Funcionário

```java
package exemplos;

public class Funcionario extends Pessoa {
    private String escolaridade;
    private String admissao;

    public Funcionario() {
    }

    public Funcionario(String escolaridade, String admissao) {
        this.escolaridade = escolaridade;
        this.admissao = admissao;
    }

    public String getEscolaridade() {
        return escolaridade;
    }

    public void setEscolaridade(String escolaridade) {
        this.escolaridade = escolaridade;
    }

    public String getAdmissao() {
        return admissao;
    }

    public void setAdmissao(String admissao) {
        this.admissao = admissao;
    }

    @Override
    public void imprimir() {
        super.imprimir();
        System.out.println("Escolaridade....: " + getEscolaridade());
        System.out.println("Admissao........: " + getAdmissao());
    }
}
```

Classe Médico

```java
package exemplos;

public class Medico extends Pessoa {
    private int CRM;
    private String especialidade;

    public Medico() {
    }

    public Medico(int CRM, String especialidade) {
        this.CRM = CRM;
        this.especialidade = especialidade;
    }

    public int getCRM() {
        return CRM;
    }

    public String getEspecialidade() {
        return especialidade;
    }

    public void setCRM(int CRM) {
        this.CRM = CRM;
    }

    public void setEspecialidade(String especialidade) {
        this.especialidade = especialidade;
    }

    @Override
    public void imprimir() {
        super.imprimir();
        System.out.println("CRM.............: " + getCRM());
        System.out.println("Especialidade...: " + getEspecialidade());
    }
}
```

Classe Paciente

```java
package exemplos;

public class Paciente extends Pessoa {
    private Double renda;
    private String planodesaude;

    public Paciente() {
    }

    public Paciente(Double renda, String planodesaude) {
        this.renda = renda;
        this.planodesaude = planodesaude;
    }

    public Double getRenda() {
        return renda;
    }

    public void setRenda(Double renda) {
        this.renda = renda;
    }

    public String getPlanodesaude() {
        return planodesaude;
    }

    public void setPlanodesaude(String planodesaude) {
        this.planodesaude = planodesaude;
    }

    @Override
    public void imprimir() {
        super.imprimir();
        System.out.println("Renda.....: " + getRenda());
        System.out.println("Plano de Saúde..: " + getPlanodesaude());
    }
}
```

Classe OrientacaoObjeto – com o método main para instanciar os objetos

```java
package exemplos;

public class OrientacaoObjeto {

    public static void main(String[] args) {
        Funcionario jose = new Funcionario();
        jose.setCpf(123456789);
        jose.setNome("José da Silva");
        jose.setTelefone("14 3232-0988");
        jose.setEscolaridade("Segundo grau completo");
        jose.setAdmissao("10/01/2000");

        Medico maria = new Medico();
        maria.setCpf(123456789);
        maria.setNome("Maria Mariano");
        maria.setTelefone("14 3232-0988");
        maria.setCRM(323456);
        maria.setEspecialidade("Pediatria");

        Paciente joao = new Paciente(678.00, "Unimed");
        joao.setCpf(76232123);
        joao.setNome("João de Oliveira");
        joao.setTelefone("14 99876-5432");

        /**
         * Imprimir dados do funcionário José *
         */
        System.out.println("*****************************************");
        System.out.println("Impressão do Funcionário José");
        System.out.println("*****************************************");
        jose.imprimir();

        /**
         * Imprimir dados da médica Maria *
         */
        System.out.println("*****************************************");
        System.out.println("Impressão da Médica Maria");
        System.out.println("*****************************************");
        maria.imprimir();

        /**
         * Imprimir dados do paciente João *
         */
        System.out.println("*****************************************");
        System.out.println("Impressão do Paciente João");
        System.out.println("*****************************************");
        joao.imprimir();
    }
}
```