

PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Josenalde oliveira

josenalde.oliveira@ufrn.br

<https://github.com/josenalde/apds>

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN



Aspectos de herança entre classes

```
public class Engenheiro {  
  
    String nome;  
    String crea;  
    float salarioBruto;  
  
    float calcularSalarioLiquido(float desconto, float bonus)  
    {  
        return (salarioBruto * desconto) + bonus;  
    }  
    //getters and setters  
  
}
```

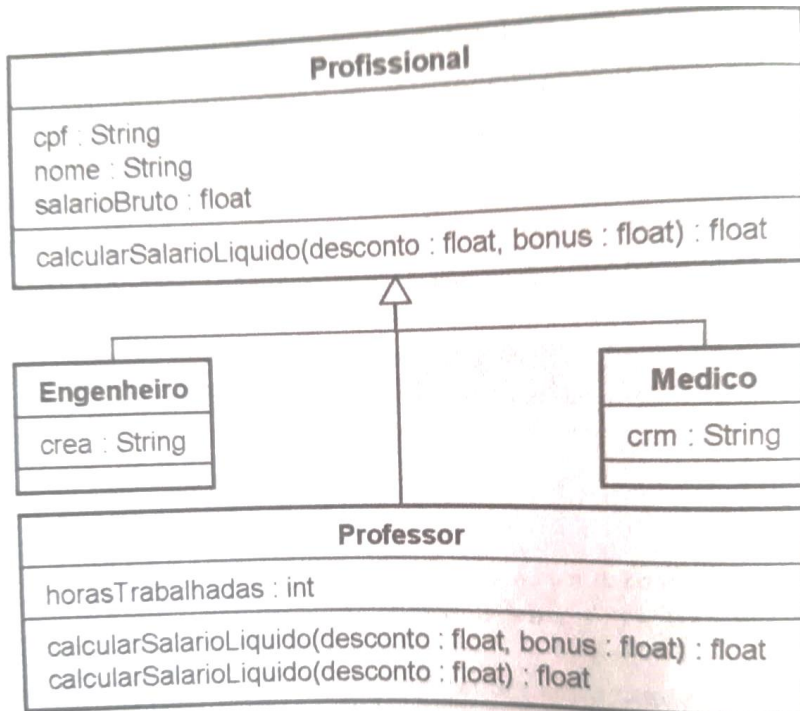
Engenheiro
nome : String salarioBruto : float crea : String
calcularSalarioLiquido() : float

- calcularSalarioLiquido(float,float):float

```
public class Principal {  
  
    public static void main(String[] args) {  
        Engenheiro e1 = new Engenheiro();  
        e1.setNome("José");  
        e1.setSalarioBruto(10000);  
        System.out.println("Salário Liquido: " +  
e1.calcularSalarioLiquido(0.8f, 400.70f));  
    }  
  
}
```



Aspectos de herança entre classes



- Professor, Engenheiro e Medico são subclasses, especializações da classe Profissional
- Engenheiro e Medico herdam atributos e método calcularSalarioLiquido
- Professor sobrescreve calcularSalarioLiquido(float,float):float e sobrecarrega calcularSalarioLiquido(float):float

```
public class Engenheiro extends Profissional {
    private String crea;
    public String getCrea() {
        return crea;
    }
    public void setCrea(String crea) {
        this.crea = crea;
    }
}
```

```
public class Medico extends Profissional {
    private String crm;
    public String getCrM() {
        return crm;
    }
    public void setCrM(String crm) {
        this.crm = crm;
    }
}
```



Aspectos de herança entre classes

```
public class Profissional { // classe Mãe

    private String cpf;

    private String nome;

    protected float salarioBruto;

    public Profissional() {

        salarioBruto = 0;

    }

    public Profissional(String cpf, String nome, float
salarioBruto) {

        this.cpf = cpf;

        this.nome = nome;

        this.salarioBruto = salarioBruto;

    }

    public String getCpf() {

        return cpf;

    }

    public final void setCpf(String cpf) {

        //código para validar o CPF

        this.cpf = cpf;

    }

    public String getNome() {

        return nome;

    }

}
```

```
    public void setNome(String nome) {

        this.nome = nome;

    }

    public float getSalarioBruto() {

        return salarioBruto;

    }

    public void setSalarioBruto(float salarioBruto) {

        if (salarioBruto >= 0) {

            this.salarioBruto = salarioBruto;

        }

    }

    public float calcularSalarioLiquido(float desconto, float
bonus) {

        return (salarioBruto * desconto) + bonus;

    }

}
```



Aspectos de herança entre classes

```
public class Professor extends Profissional {  
    private int horasTrabalhadas;  
    public Professor() {  
        this(0); //construtor da própria classe  
    }  
    public Professor(float salarioBruto) {  
        this.salarioBruto = salarioBruto;  
    }  
    public Professor(String cpf, String nome, float salarioBruto) {  
        super(cpf, nome, salarioBruto); //superclasse  
    }  
    public int getHorasTrabalhadas() {  
        return horasTrabalhadas;  
    }  
    public void setHorasTrabalhadas(int horasTrabalhadas) {  
        this.horasTrabalhadas = horasTrabalhadas;  
    }  
    @Override  
    public float calcularSalarioLiquido(float desconto, float bonus) {  
        //SOBRESCRITA (mesmo nome e parâmetros)  
        return ((getSalarioBruto() * horasTrabalhadas) * desconto) + bonus;  
    }  
}
```

```
//SOBRECARGA (mesmo nome do método, parâmetros diferentes)  
public float calcularSalarioLiquido(float desconto) {  
    return ((getSalarioBruto() * horasTrabalhadas) * desconto);  
}
```

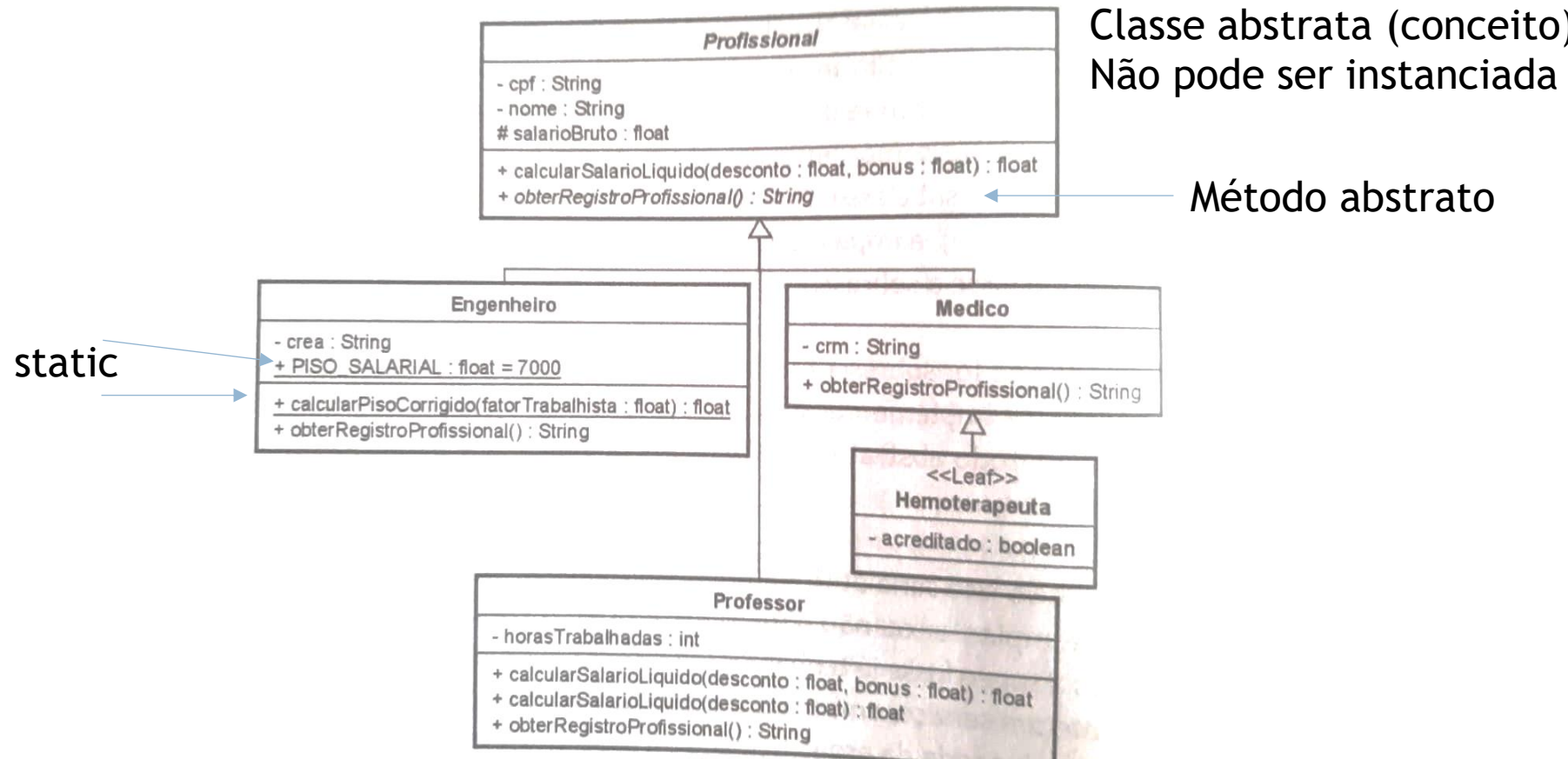


Aspectos de herança entre classes

```
public class Principal {  
    public static void main(String[] args) {  
        Engenheiro e1 = new Engenheiro();  
        e1.setNome("José");  
        e1.setSalarioBruto(10000);  
        System.out.println("Salário Líquido: " + e1.calcularSalarioLiquido(0.8f, 400.70f));  
        Medico m1 = new Medico();  
        m1.setNome("Vinicius");  
        m1.setSalarioBruto(8000);  
        System.out.println("Salário Líquido: " + m1.calcularSalarioLiquido(0.7f, 1500.70f));  
        Professor p1 = new Professor();  
        p1.setNome("Pablo");  
        p1.setHorasTrabalhadas(160);  
        p1.setSalarioBruto(50);  
        System.out.println("Salário Líquido: " + p1.calcularSalarioLiquido(0.6f, 500.30f));  
        System.out.println("Salário Líquido: " + p1.calcularSalarioLiquido(0.6f));  
        System.out.println("Salário Bruto: " + p1.getSalarioBruto());  
        Professor p2 = new Professor(55);  
        p2.setNome("Rodrigo");  
        p2.setHorasTrabalhadas(160);  
        Engenheiro eng = new Engenheiro();  
        System.out.println("Piso Salarial: " + eng.PISO_SALARIAL);  
        System.out.println("Piso Salarial: " + Engenheiro.PISO_SALARIAL);}}
```



Aspectos de herança entre classes



- **final** aplicado a atributos que deseja-se ser CONSTANTE, inicializado na declaração
- Na classe Engenheiro: **public final float PISO_SALARIAL = 7000;**
- Contudo, o melhor é este piso pertencer à CLASSE e não a uma instância dela. Usa-se static neste caso, bastando Engenheiro.PISO_SALARIAL para acessar
- **final** e **static** também se aplicam à métodos. No caso de classe, **final** implica não poder ser estendida em subclasses



Aspectos de herança entre classes

```
public static float calcularPisoCorrigido(float fatorTrabalhista) {  
    return PISO_SALARIAL * fatorTrabalhista;  
}
```

```
public abstract class Profissional {  
    //...  
    public abstract String obterRegistroProfissional();  
}
```

@Override

```
public String obterRegistroProfissional() {  
    return getCpf();  
}
```

PROFESSOR

@Override

```
public String obterRegistroProfissional() {  
    return crea;  
}
```

ENGENHEIRO

@Override

```
public String obterRegistroProfissional() {  
    return crm;  
}
```

MEDICO



Solução com classes abstratas

```
...
public static void main(String[] args) {
    Profissional p1 = new Professor("111.111.111.-11","josé",100);
    imprimirDados(p1);
}
public static void imprimirDados(Profissional profissional) {
    System.out.println("CPF: " + profissional.getCpf());
    System.out.println("Nome: " + profissional.getNome());
    System.out.println("Registro Profissional: " + profissional.obterRegistroProfissional());
}
```

Veja abaixo uma abordagem com problemas de modelagem

```
public static void imprimirDados(Profissional profissional) {
    System.out.println("CPF: " + profissional.getCpf());
    System.out.println("Nome: " + profissional.getNome());
    if (p instanceof Engenheiro) { ... "CREA: " + ((Engenheiro) p).getCrea(); }
    if (p instanceof Medico) { ... "CRM: " + ((Medico) p).getCrm(); }
}
```

