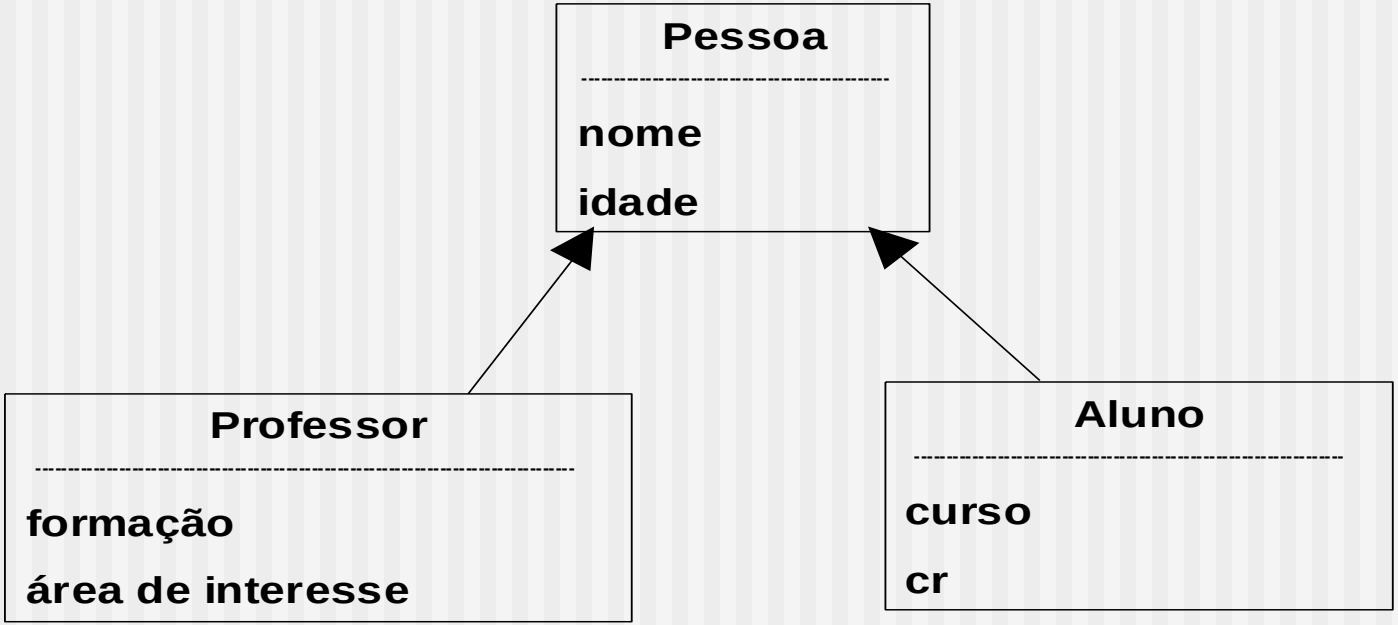
Programação Orientada a Objetos

Herança

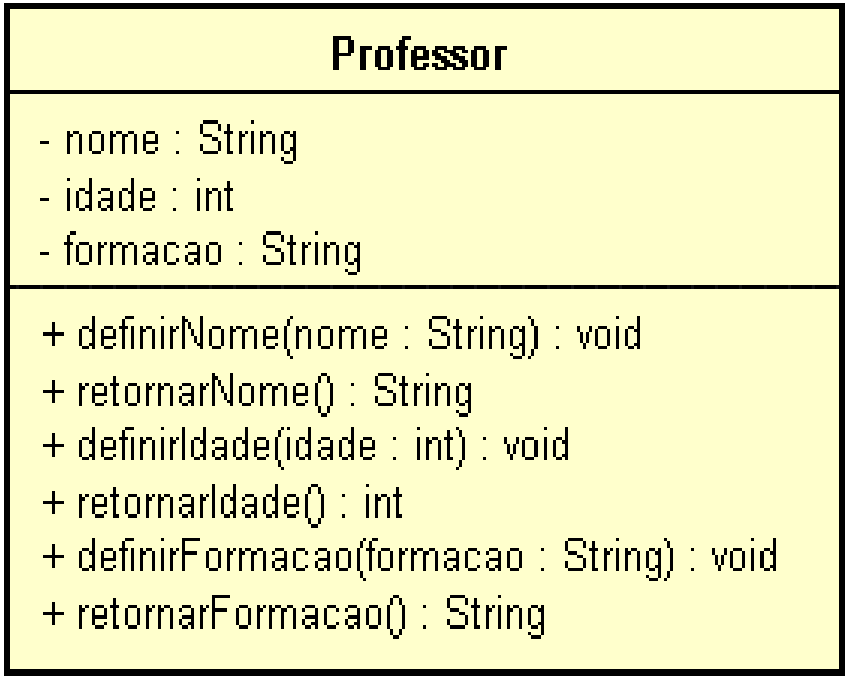
Herança

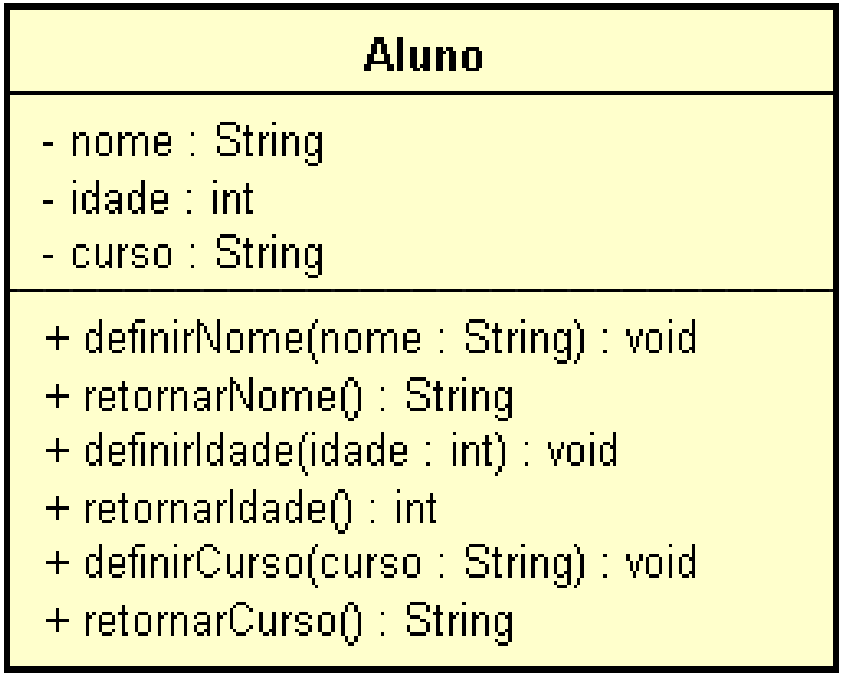
- Permite a uma classe herdar o estado (atributos) e o comportamento (métodos) de outra classe.
  - Superclasse
  - Subclasse
  - Ancestral
  - Descendente

Herança

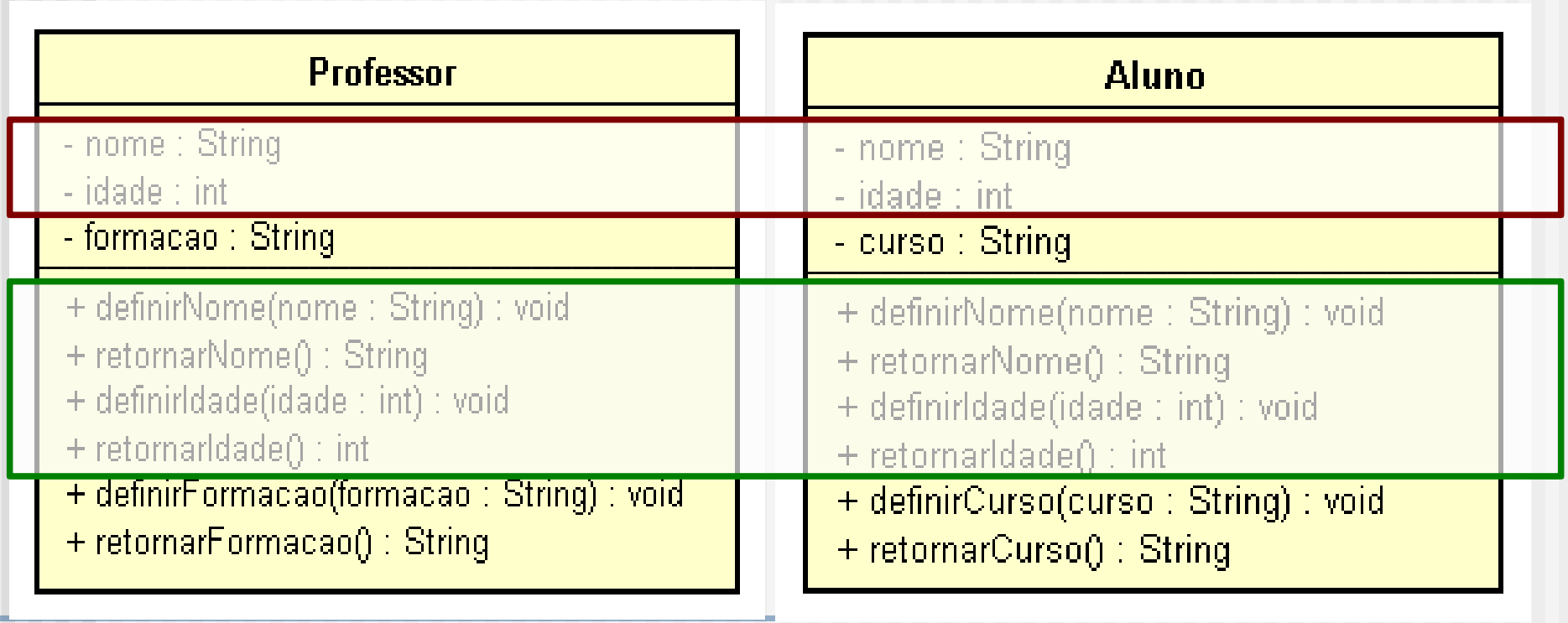

Pessoa
nome
idade
Professor
formação
área de interesse
Aluno
curso
cr

Herança

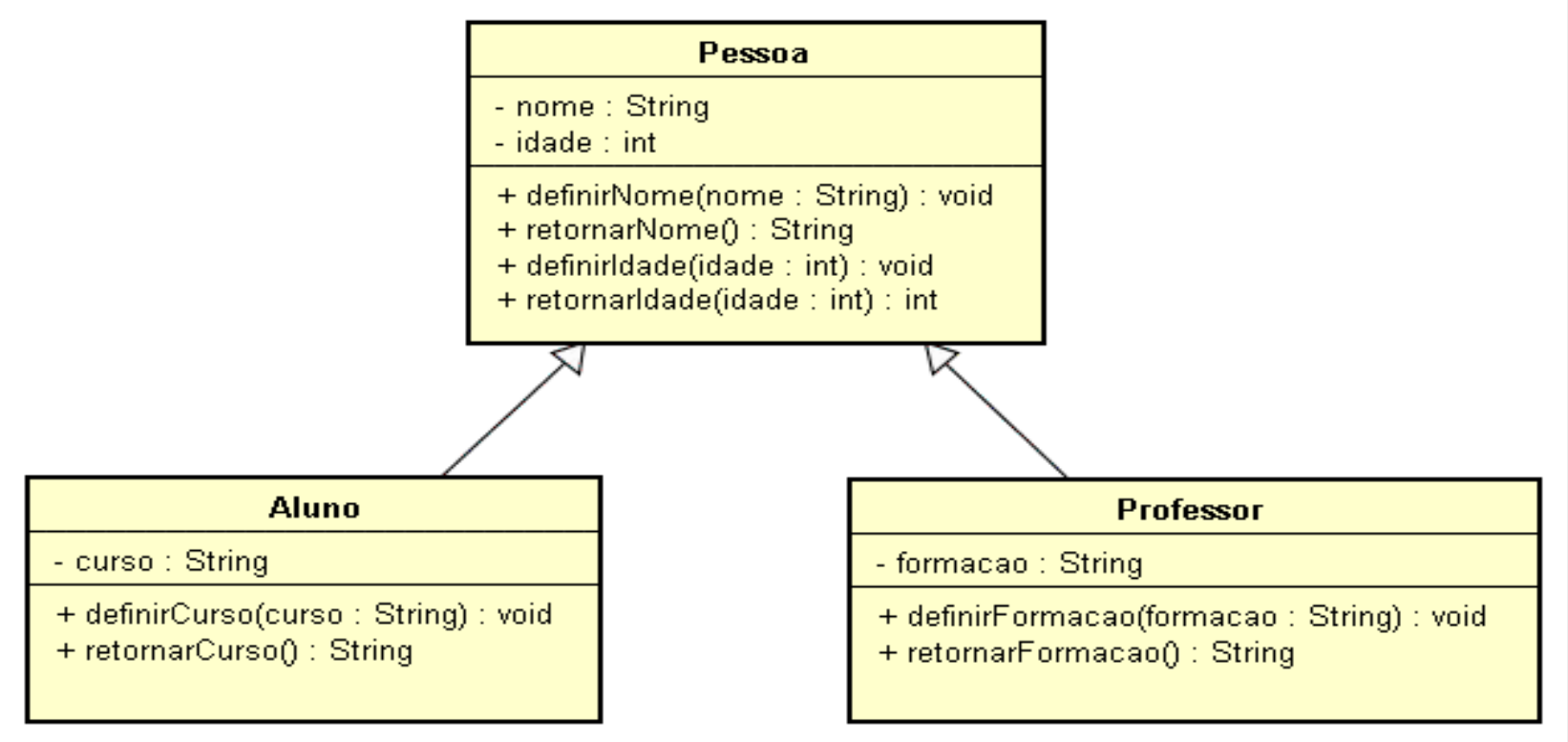

Professor
- nome : String
- idade : int
- formacao : String
+ definirNome(nome : String) : void
+ retornarNome() : String
+ definirIdade(idade : int) : void
+ retornarIdade() : int
+ definirFormacao(formacao : String) : void
+ retornarFormacao() : String

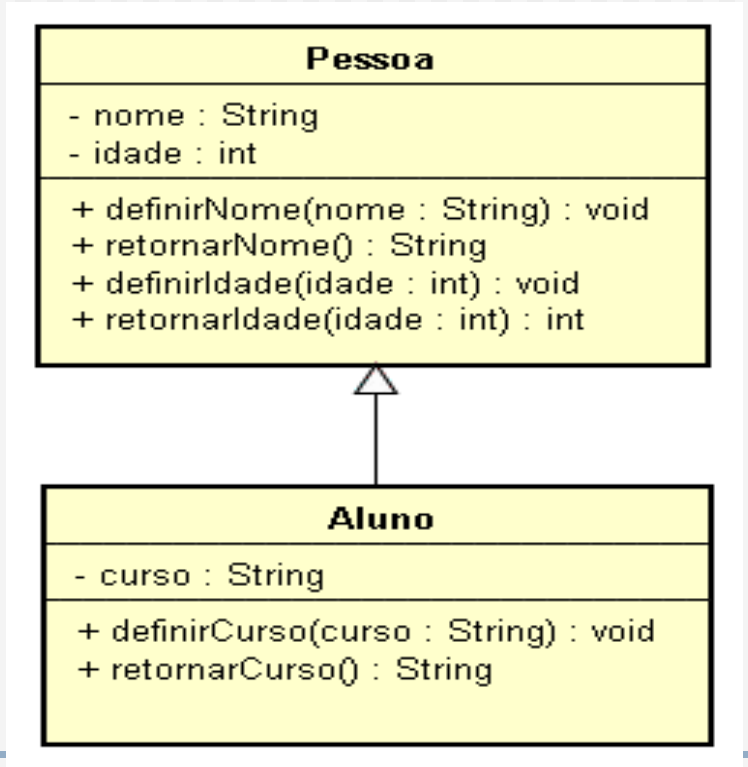

Aluno
- nome : String
- idade : int
- curso : String
+ definirNome(nome : String) : void
+ retornarNome() : String
+ definirIdade(idade : int) : void
+ retornarIdade() : int
+ definirCurso(curso : String) : void
+ retornarCurso() : String

Herança


Professor
- nome : String
- idade : int
- formacao : String
+ definirNome(nome : String) : void
+ retornarNome() : String
+ definirIdade(idade : int) : void
+ retornarIdade() : int
+ definirFormacao(formacao : String) : void
+ retornarFormacao() : String
Aluno
- nome : String
- idade : int
- curso : String
+ definirNome(nome : String) : void
+ retornarNome() : String
+ definirIdade(idade : int) : void
+ retornarIdade() : int
+ definirCurso(curso : String) : void
+ retornarCurso() : String

Herança


Pessoa
- nome : String
- idade : int
+ definirNome(nome : String) : void
+ retornarNome() : String
+ definirIdade(idade : int) : void
+ retornarIdade(idade : int) : int
Aluno
- curso : String
+ definirCurso(curso : String) : void
+ retornarCurso() : String
Professor
- formacao : String
+ definirFormacao(formacao : String) : void
+ retornarFormacao() : String



Instâncias de Aluno

João
25
Gestão da Informação

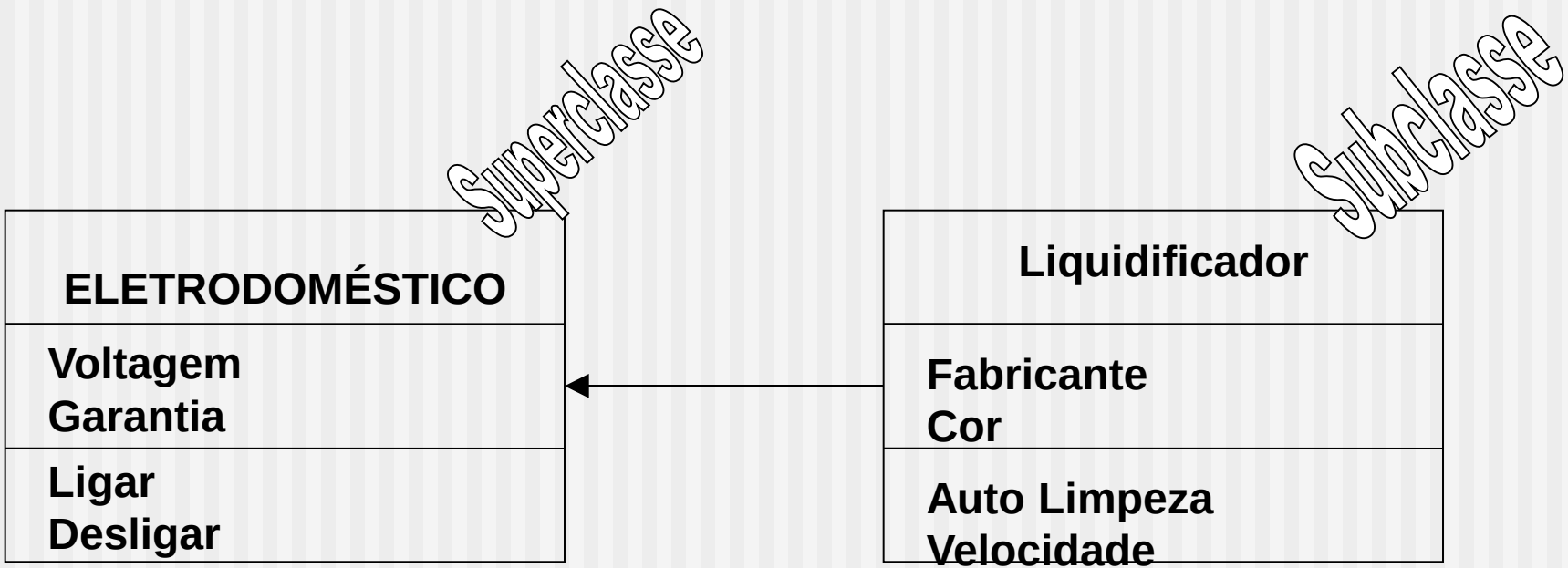
Maria
20
Gestão da Informação

```
// SuperClass.java
        public class SuperClass
        {
            // Atributos e métodos
        }
//  SubClass.java
        public class SubClass EXTENDS SuperClass
        {
            // Atributos e métodos
        }

class Aluno extends Pessoa {
...
}
```

```
class Pessoa {
    String  nome;
    int idade;

    void definirNome(String valor) {
        nome = valor;
    }

    String retornarNome() {
        return nome;
    }

    void definirIdade(int valor) {
        idade = valor;
    }

    int retornarIdade() {
        return idade;
    }
}
```

```
class Aluno extends Pessoa {
    String curso;

    void definirCurso(String valor) {
        curso = valor;
    }

    String retornarCurso() {
        return curso;
    }

}
```

```
Aluno joao = new Aluno();
joao.definirNome(“João”);
joao.definirIdade(25);
joao.definirCurso(“Sistemas de Informação”);

Aluno maria = new Aluno();
maria.definirNome(“Maria”);
maria.definirIdade(20);
maria.definirCurso(“Sistemas de Informação”);
```


Superclasse
Subclasse
ELETRODOMÉSTICO
Voltagem
Garantia
Ligar
Desligar
Liquidificador
Fabricante
Cor
Auto Limpeza
Velocidade

Exercícios: Conta Bancária

Elabore uma classe ContaBancaria, com os seguintes membros:

- ✓ atributo String cliente
- ✓ atributo int num_conta
- ✓ atributo float saldo
- ✓ método sacar (o saldo não pode ficar negativo)
- ✓ método depositar

Agora acrescente ao projeto duas classes herdadas de ContaBancaria: ContaPoupança e ContaEspecial, com as seguintes características a mais:

⇒ Classe ContaPoupança:
- ✓ atributo int dia de rendimento
- ✓ método calcularNovoSaldo, recebe a taxa de rendimento da poupança e atualiza o saldo.

⇒ Classe ContaEspecial
- ✓ atributo float limite
- ✓ redefinição do método sacar, permitindo saldo negativo até o valor do limite.

Após a implementação das classes acima, você deverá implementar uma classe Contas.Java, contendo o método main. Nesta classe, você deverá implementar:

a) Incluir dados relativos a(s) conta(s) de um cliente;
b) Sacar um determinado valor da(s) sua(s) conta(s);
c) Depositar um determinado valor na(s) sua(s) conta(s);
d) Mostrar o novo saldo do cliente, a partir da taxa de rendimento, daqueles que possuem conta poupança;
e) Mostrar os dados da(s) conta(s) de um cliente;

Exercício

- Uma loja tem 2 tipos de funcionários: vendedores e administrativos. Para ambos a empresa precisa ter o registro do nome e RG do funcionário.
- Os vendedores têm um salário base, mas ganham também comissão de suas vendas. Os administrativos têm um salário base, mas podem ganhar horas extras adicionais.
- Faça uma hierarquia de classes que tenha uma classe ancestral que implemente o que for comum aos dois tipos de funcionários e uma classe descendente para cada tipo.
- Os vendedores devem ter um método que acumule o total de vendas durante o mês e um método que retorne seu salário total, considerando que a comissão é de 5%. Para os administrativos as horas extras é que são acumuladas e pagas com o valor de um centésimo do salário por hora. Nos dois casos, o método que retorna o salário a receber zera os valores acumulados.
- Crie um programa principal que utilize um vetor de 5 vendedores e outro vetor de 5 administrativos. Receba os dados desses funcionários via usuário, e crie situações para testar os métodos criados.
- Crie outros métodos e atributos que sugiram características comuns às duas classes, e características específicas de cada classe.