

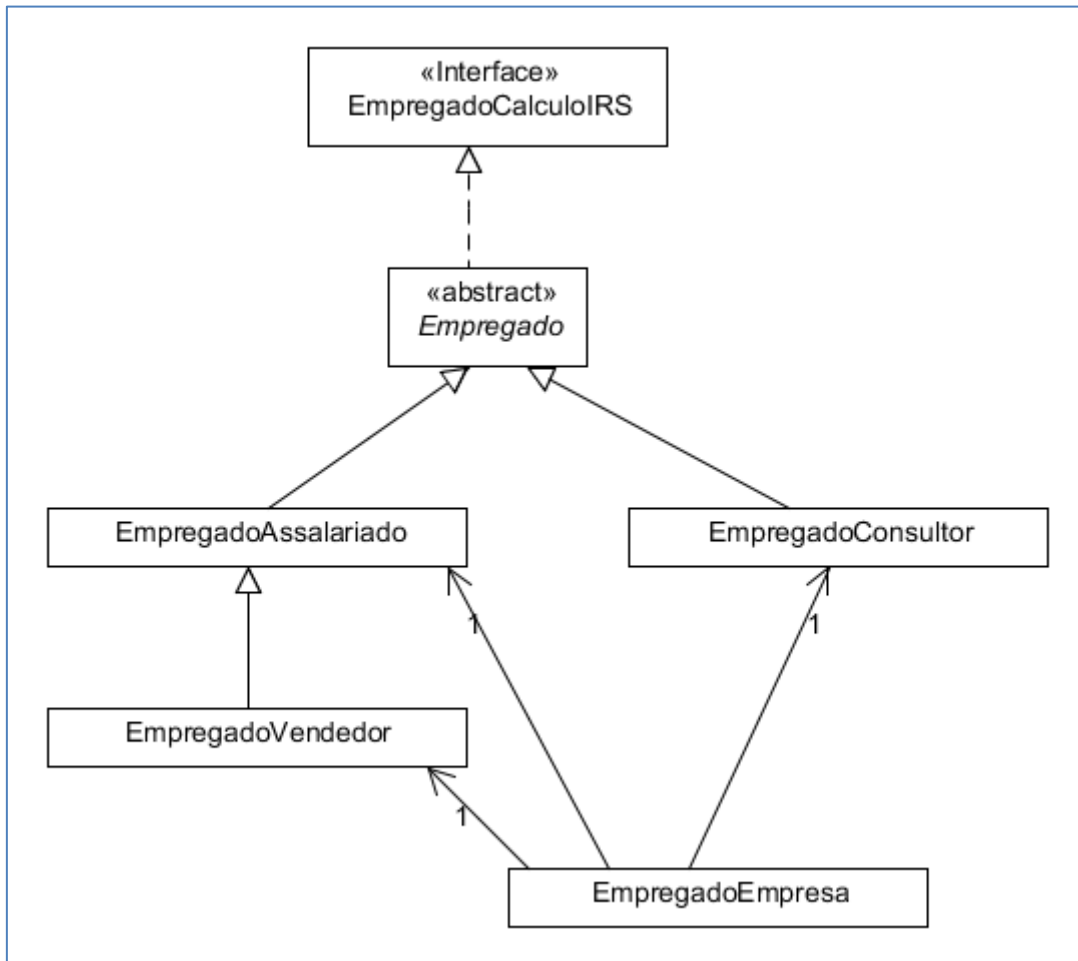
Programação em Java – Fundamentos

Capítulo 4

Exercícios

Exercício Empregado

Considere as classes criadas no capítulo anterior. Vamos acrescentar uma interface que será implementada por Empregado. O novo diagrama está representado abaixo:



Vamos começar por copiar essas classes para o package `capitulo4`. Em seguida vamos alterar as classes com o objetivo de praticar a criação e utilização de uma interface, assim como encapsular as variáveis que vão deixar de ter visualização pública.

Interface **EmpregadoCalculoIRS**

Vamos começar por definir uma interface de acordo com o código abaixo:



```
package capitulo4;
public interface EmpregadoCalculoIRS {
    public final float[][] escaloesIRS = {
        {5600f, 5.0f}, {14000f, 7.0f}, {40000f, 10.0f}, {0f, 15.0f}};
```

```

    public double calculoImposto(double salarioBruto);
    public double calculoSalarioLiquido(double salarioBruto);
}

```

Esta interface define um array com os escalões do IRS abaixo indicados:

Salário anual	%IRS
5600€	5%
14000€	7%
40000€	10%
Mais de 40000€	15%


Além disso esta interface obriga as classes que a implementarem a definir os métodos `calculoImposto()` e `calculoSalarioLiquido()`.

Classe Empregado

Vamos alterar a classe Empregado com os seguintes objetivos:

- 1) Implementar a interface `EmpregadoCalculoIRS` o que implica a definição dos métodos da interface;
- 2) Definir todas as variáveis como **private** o que as torna invisíveis fora da própria classe, por exemplo em `EmpregadoEmpresa`;
- 3) Criar os métodos de objeto abaixo indicados, com o qualificador de visualização **public**, para através deles permitir o acesso às variáveis:
 - a) `obterNumero()` – devolve o valor da variável de objeto `numero`;
 - b) `obterNome()` – devolve o valor da variável de objeto `nome`;
 - c) `alterarNome()` – recebe como parâmetro uma `String` e modifica o valor da variável de objeto `nome`;

O exemplo abaixo mostra uma possível implementação desta classe:



```

package capitulo4;
public abstract class Empregado implements EmpregadoCalculoIRS {

    private static long ultimoNumEmp=0;

    private String nome;
    private long numero;

    public Empregado(String s) {
        super();
        this.nome = s;
        this.numero = ++ultimoNumEmp;
    }

    public final long obterNumero() {
        return this.numero;
    }

    public final String obterNome() {

```

```

        return this.nome;
    }

    public final void alterarNome(String s) {
        this.nome = s;
    }

    //declaro método abstrat
    public abstract float calcularSalarioBruto();

    //método da interface
    @Override
    public double calculoImposto(double salarioBruto) {
        //à partida fica no maior escalão
        float percentagemIRS = escaloesIRS[escaloesIRS.length - 1][1];
        for (int i = 0; i < escaloesIRS.length - 1; i++) {
            if (salarioBruto <= escaloesIRS[i][0]) {
                percentagemIRS = escaloesIRS[i][1];
                break;
            }
        }
        return salarioBruto*(percentagemIRS/100);
    }
    //método da interface
    @Override
    public double calculoSalarioLiquido(double salarioBruto) {
        return salarioBruto - calculoImposto(salarioBruto);
    }
}

```

Classe EmpregadoAssalariado

Vamos alterar a classe EmpregadoAssalariado, que é uma extensão de Empregado, com os seguintes objetivos:

- 1) Alterar a visibilidade das variáveis de objeto para **private**:
 - a) numHoras - numero de horas de trabalho;
 - b) salHora – salário/hora;
- 2) Um construtor que recebe o nome do empregado, o número de horas de trabalho e o respetivo salário/hora. Na execução começa por invocar o construtor de Empregado (atenção que esta classe não tem construtor por omissão) e atribui os parâmetros recebidos às respetivas variáveis de objeto;
- 3) Os métodos de objeto com visualização **public**:
 - a) obterNumHoras() – devolve o valor da variável de objeto numHoras;
 - b) alterarNumHoras() – recebe um valor e modifica o valor da variável de objeto numHoras;
 - c) obterSalHora() – devolve o valor da variável de objeto salHora;
 - d) alterarSalHora() - recebe um valor e modifica o valor da variável de objeto salHora;
- 4) O método calcularSalarioBruto(), com visualização **public**, que devolve o produto entre o número de horas e o salário por hora. A definição deste método é essencial para que a classe não seja abstract;

Segue um exemplo de implementação:



```
package capitulo4;
public class EmpregadoAssalariado extends Empregado {

    private float numHoras;
    private float salHora;

    public EmpregadoAssalariado(String s, float numHoras, float salHora) {
        super(s); //evocar construtor superclass para inicializar dados
        comuns
        this.numHoras = numHoras;
        this.salHora = salHora;
    }

    public float obterNumHoras() {
        return numHoras;
    }

    public float obterSalHora() {
        return salHora;
    }

    public void alterarNumHoras(float novasHoras) {
        numHoras=novasHoras;
    }

    public void alterarSalHora(float novoSalHora) {
        salHora=novoSalHora;
    }

    @Override
    public float calcularSalarioBruto() {
        return (numHoras*salHora);
    }
}
```

Repare que o construtor de EmpregadoAssalariado tem que chamar o construtor de Empregado utilizando a instrução **super(s)**, visto que a classe Empregado não possui um construtor por omissão.

Se não redefinir o método calcularSalárioBruto() obterá um erro de compilação, alertando-o para a necessidade de colocar a classe como **abstract**. Isto é uma consequência de termos declarado esse método como **abstract** na super classe.

Classe EmpregadoVendedor

Vamos alterar a classe EmpregadoVendedor, que é uma extensão de EmpregadoAssalariado, com os seguintes objetivos:

- 1) Alterar a visibilidade das variáveis de objeto para private:
 - a) vendas – total de vendas feitas pelo vendedor, em euros;
 - b) comissão – valor da comissão que o vendedor vai receber, representado sob a forma de percentagem do total de vendas;
- 2) Um construtor que recebe o nome do empregado, o número de horas de trabalho, o respetivo salário/hora, a comissão nas vendas e o valor total de vendas. Na execução começa por invocar o construtor de EmpregadoAssalariado (atenção que esta classe não tem construtor por omissão) e atribui os parâmetros recebidos às respetivas variáveis de objeto;
- 3) Métodos de objecto com visualização public:

- a) obterComissão() - devolve o valor da variável de objeto comissão;
 - b) alterarComissão() recebe um valor e altera a variável de objeto comissão;
 - c) obterVendas(); - devolve o valor da variável de objeto vendas;
 - d) alterarVendas()- recebe um valor e altera a variável de objeto vendas;
- 4) Método calcularComissaoVendas() que devolve o produto entre o valor das vendas e a comissão das vendas (em percentagem);
- 5) Redefinir calcularSalarioBruto() pois agora há uma comissão de vendas que tem que ser adicionada ao salário base (numero de horas a multiplicar pelo salário hora);

Segue um exemplo de implementação:



```
package capitulo4;
public class EmpregadoVendedor extends EmpregadoAssalariado {

    private float vendas;
    private float comissao; //em % das vendas

    //construtor de vendedor, que chama o de assalariado,
    //que por sua vez chama o de empregado
    public EmpregadoVendedor(String nome, float numHoras,
        float salHora, float comissao, float vendas) {
        super(nome,numHoras,salHora);
        this.comissao = comissao;
        this.vendas = vendas;
    }

    public float obterComissao() {
        return comissao;
    }

    public void alterarComissao(float novaComissao) {
        comissao=novaComissao;
    }

    public float obterVendas() {
        return vendas;
    }

    public void alterarVendas(float novasVendas) {
        vendas=novasVendas;
    }

    public float calcularComissaoVendas() {
        return (vendas*comissao/100);
    }

    @Override
    public float calcularSalarioBruto() {
        return (super.calcularSalarioBruto()+calcularComissaoVendas());
    }
}
```

Repare na forma como foi implementado o método calcularSalarioBruto(), pois aproveita o método da super classe, sendo adicionada a comissão de vendas.

Classe EmpregadoConsultor

Vamos alterar a classe EmpregadoConsultor, que herda as características de Empregado, com os seguintes objetivos:

- 1) Adicionar uma variável de objeto com visualização private:
 - a) salMensal – representa o salário mensal fixo e bruto em euros;
- 2) Um construtor que recebe o nome do empregado e o respetivo salario mensal. Na execução começa por invocar o construtor de Empregado (atenção que esta classe não tem construtor por omissão) e atribui o outro parâmetro recebido à respetiva variável de objeto;
- 3) Redefinir calcularSalarioBruto() que devolve o salário mensal tendo em conta que este é uma quantia fixa:

Segue um exemplo de uma possível implementação



```
package capitulo4;
public class EmpregadoConsultor extends Empregado {

    private float salMensal;

    public EmpregadoConsultor(String s, float salMensal) {
        super(s); //evocar construtor superclass para inicializar dados
        comuns
        this.salMensal=salMensal;
        //this distingue parametro de construtor
    }

    public float obterSalMensal() {
        return salMensal;
    }

    public void alterarSalMensal(float novoSalHora) {
        salMensal=novoSalHora;
    }

    @Override
    public float calcularSalarioBruto() {
        return salMensal;
    }
}
```

Classe Empresa

Criar classe **Empresa** que cria objetos das classes anteriores e testa o seu funcionamento. Para isso deve escrever todas as suas variáveis, alterar os seus valores e escrever os novos valores. Deve também testar os métodos definidos em Empregado por causa da interface e herdados por todas as classes.



```
package capitulo4;
public class EmpregadoEmpresa {
    public static void main(String[] args) {
        System.out.println("===== EmpregadoAssalariado =====");
        EmpregadoAssalariado assal1=
```

```

        new EmpregadoAssalariado("Antônio", 176.5f, 3.5f);
System.out.println("Nome.....="+assall.obterNome());
System.out.println("Numero.....="+assall.obterNumero());
System.out.println("Numero de Horas="+assall.obterNumHoras());
System.out.println("Salario Hora...="+assall.obterSalHora());
System.out.println("Salario bruto..="+assall.calcularSalarioBruto());
System.out.println("IRS.....="+
        assall.calcularImposto(assall.calcularSalarioBruto()));
System.out.println("Salario liquido="+
        assall.calcularSalarioLiquido(assall.calcularSalarioBruto()));
System.out.println("=== Assalariado com novo nome, " + "
        mais horas e mais salario/hora");
assall.alterarNumHoras(200f);
assall.alterarSalHora(4.75f);
assall.alterarNome("Antônio Gonçalves");
System.out.println("Nome.....="+assall.obterNome());
System.out.println("Numero.....="+assall.obterNumero());
System.out.println("Numero de Horas="+assall.obterNumHoras());
System.out.println("Salario Hora...="+assall.obterSalHora());
System.out.println("Salario bruto..="+assall.calcularSalarioBruto());
System.out.println("IRS.....="+
        assall.calcularImposto(assall.calcularSalarioBruto()));
System.out.println("Salario liquido="+
        assall.calcularSalarioLiquido(assall.calcularSalarioBruto()));

System.out.println("===== EmpregadoVendedor =====");
System.out.println("\\"filho\\" de assalariado\\"neto\\" de empregado");
EmpregadoVendedor vendedor1=
        new EmpregadoVendedor("Carlos", 100f, 3.5f, 10f, 3000f);
System.out.println("Nome.....="+vendedor1.obterNome());
System.out.println("Numero.....="+vendedor1.obterNumero());
System.out.println("Numero de Horas.....="+vendedor1.obterNumHoras());
System.out.println("Salario Hora.....="+vendedor1.obterSalHora());
System.out.println("Comissao.....="+vendedor1.obterComissao());
System.out.println("Vendas.....="+vendedor1.obterVendas());
System.out.println("Salario em comissoes="+
        vendedor1.calcularComissaoVendas());
System.out.println("Salario bruto.....="+
        vendedor1.calcularSalarioBruto());
System.out.println("IRS.....="+
        vendedor1.calcularImposto(vendedor1.calcularSalarioBruto()));
System.out.println("Salario liquido.....="+
        vendedor1.calcularSalarioLiquido(vendedor1.calcularSalarioBruto()));
System.out.println("=== O vendedor trabalhou mais e foi aumentado");
vendedor1.alterarVendas(3500f);
vendedor1.alterarNumHoras(300f);
vendedor1.alterarSalHora(5.75f);
System.out.println("Nome.....="+vendedor1.obterNome());
System.out.println("Numero.....="+vendedor1.obterNumero());
System.out.println("Numero de Horas.....="+vendedor1.obterNumHoras());
System.out.println("Salario Hora.....="+vendedor1.obterSalHora());
System.out.println("Comissao.....="+vendedor1.obterComissao());
System.out.println("Vendas.....="+vendedor1.obterVendas());
System.out.println("Salario em Comissoes="+
        vendedor1.calcularComissaoVendas());
System.out.println("Salario bruto.....="+
        vendedor1.calcularSalarioBruto());
System.out.println("IRS.....="+
        vendedor1.calcularImposto(vendedor1.calcularSalarioBruto()));
System.out.println("Salario liquido.....="+
        vendedor1.calcularSalarioLiquido(vendedor1.calcularSalarioBruto()));

System.out.println("===== EmpregadoConsultor =====");
EmpregadoConsultor consult1=
        new EmpregadoConsultor("Carlos Muralhas", 2000f);
System.out.println("Nome.....="+consult1.obterNome());
System.out.println("Numero.....="+consult1.obterNumero());

```

```

        System.out.println("Salario Mensal."+consult1.obterSalMensal());
        System.out.println("Salario bruto."+
                                consult1.calcularSalarioBruto());
        System.out.println("IRS.....="+
                                consult1.calcularSalarioBruto());
        System.out.println("Salario liquido="+
                                consult1.calcularSalarioBruto());

        System.out.println("=== O consultor foi aumentado...");
        consult1.alterarSalMensal(3000f);
        System.out.println("Nome.....="+consult1.obterNome());
        System.out.println("Numero.....="+consult1.obterNumero());
        System.out.println("Salario Mensal="+consult1.obterSalMensal());
        System.out.println("Salario bruto="+consult1.calcularSalarioBruto());
        System.out.println("IRS.....="+
                                consult1.calcularSalarioBruto());
        System.out.println("Salario liquido="+
                                consult1.calcularSalarioBruto());
    }
}

```

Cotação:

Empregado – 10%
 EmpregadoAssalariado-20%
 EmpregadoVendedor-30%
 EmpregadoConsultor-10%
 EmpregadoEmpresa-30%