Programação Orientada a Objeto

A08 - Modificadores de Acesso e Interface

- modificadores de acesso static e final;
- Interface.

Dr. Márcia L. Aguena Castro
marcia@39dev.com

Me. Reinaldo de O. Castro
reinaldo@39dev.com

Modificador de Acesso static - Atributos

Atributo **`static`**

- São atributos **relativos à sua classe** e não aos objetos. São definidos pela palavra-chave `static`.
- Atributos estáticos têm o mesmo valor para todas as instâncias de uma classe. Todo **objeto dessa classe compartilhará** seu valor.
- Existem mesmo **antes de ser criado qualquer objeto** da classe.
- Não podemos utilizar `this` nos atributos estáticos.
- Se forem públicos, podem ser acessados através da classe, sem precisar ter um objeto para chamá-lo.

Modificador de Acesso static

Atributo **`static`**

- `contador` foi declarado estático e público.
- Cada vez que alguém nomeia um objeto da classe `Pessoa`, ele é atualizado.

```
1   public abstract class Pessoa{
2     private String nome;
3     public static int contador = 0;
4
5     public Pessoa() {}
6     public Pessoa(String $nome) {
7       this.nome=$nome;
8       contador++;
9     }
10    public String getNome() {
11      contador ++;
12      return this.nome;
13    }
```

Modificador de Acesso static

Atributo **static**

```
1  public class TestaStatic{
2    public static void main (String[] args) {
3
4      System.out.println("Pessoas nomeadas: "+ Pessoa.contador);
5
6      Pessoa p1 = new PessoaFisica("Marcia","123.456.678-00");
7      Pessoa p2 = new PessoaJuridica("39DEV","098.765.432/0001-00");
8
9      System.out.println(p1.getNome() + " - " + p1.getIdentificador());
10     System.out.println(p2.getNome() + " - " + p2.getIdentificador());
11     System.out.println("Pessoas nomeadas: "+ Pessoa.contador);
12     }
13 }
```

Chamada do atributo contador é feita através da classe, não tem objeto ligado a ele.

Modificador de Acesso static

Atributo **static**

```
1  public class TestaStatic{
2    public static void main (String[] args) {
3
4      System.out.println("Pessoas nomeadas: "+ Pessoa.contador);
5
6      Pessoa p1 = new PessoaFisica("Marcia","123.456.678-00");
7      Pessoa p2 = new PessoaJuridica("39DEV","098.765.432/0001-00");
8
9      System.out.println(p1.getNome() + " - " + p1.getIdentificador());
10     System.out.println(p2.getNome() + " - " + p2.getIdentificador());
11     System.out.println("Pessoas nomeadas: "+ Pessoa.contador);
12     Pessoa.contador = 30;
13     System.out.println("Pessoas nomeadas: "+ Pessoa.contador;
14 }
```

Sendo público, alguém pode manipular o contador diretamente

Modificador de Acesso static

Atributo **static**

- contador foi declarado estático e **privado**.
- É preciso de um método público para usá-lo fora da classe.

```
1  public abstract class Pessoa{
2      private String nome;
3      //public static int contador = 0;
4      private static int contador = 0;
5
6      public Pessoa() {}
7      public Pessoa(String $nome) {
8        this.nome=$nome;
9        contador++;
10     }
11     public String getNome() {
12       contador ++;
13       return this.nome;
14     }
15     public int getContador(){
16       return contador;
17     }
```

Modificador de Acesso static

Atributo **static**

```
1  public class TestaStatic{
2    public static void main (String[] args) {
3      //Com contador privado, esse comando daria erro
4      //System.out.println("Pessoas nomeadas: "+ Pessoa.contador);
5
6      Pessoa p1 = new PessoaFisica("Marcia","123.456.678-00");
7      Pessoa p2 = new PessoaJuridica("39DEV","098.765.432/0001-00");
8
9      System.out.println(p1.getNome() + " - " + p1.getIdentificador());
10     System.out.println(p2.getNome() + " - " + p2.getIdentificador());
11     System.out.println("Pessoas nomeadas: "+ p1.getContador());
12     //Pessoa.contador = 30; também daria erro
13     System.out.println("Pessoas nomeadas: "+ p1.getContador());
14 }
```

É preciso um objeto para chamar o método getContador(). Faz sentido?
E se eu quiser saber o valor de contador antes de criar objetos?

Modificador de Acesso static - Métodos

Método **`static`**

- É um **método de classe** e não de objetos. É definido pela palavra-chave `static`;
- Também existem **mesmo antes de ser criado qualquer objeto** da classe;
- Não podemos utilizar `this` nos métodos estáticos também;
- **Só podem manipular atributos estáticos**;

Modificador de Acesso static - Métodos

Método **`static`**

- Por ser um método ligado à classe, não pode ser abstrato: ele **deve ser implementado na classe**;
- Embora não possa ser sobrescrito, **pode ser sobrecarregado**.
- O método `main` é estático pois permite que a JVM invoque-o sem criar nenhuma instância de classe onde foi escrito.

Modificador de Acesso static - Método

Método **`static`**

- O método `getContador()` foi criado para acessar o atributo estático.

```
1  public abstract class Pessoa{
2    private String nome;
3    private static int contador = 0;
4
5    public Pessoa() {}
6    public Pessoa(String $nome) {
7      this.nome=$nome;
8      contador++;
9    }
10   public String getNome() {
11     contador ++;
12     return this.nome;
13   }
14   public static int getContador(){
15     return contador;
16   }
```

Modificador de Acesso static - Método

Atributo **static**

```
1  public class TestaStatic{
2    public static void main (String[] args) {
3
4      System.out.println("Pessoas nomeadas: "+ Pessoa.getContador());
5
6      Pessoa p1 = new PessoaFisica("Marcia","123.456.678-00");
7      Pessoa p2 = new PessoaJuridica("39DEV","098.765.432/0001-00");
8
9      System.out.println(p1.getNome() + " - " + p1.getIdentificador());
10     System.out.println(p2.getNome() + " - " + p2.getIdentificador());
11     System.out.println("Pessoas nomeadas: "+ Pessoa.getContador());
12 }
```

Chamada do método `getContador()` é feita através da classe, não tem objeto ligado a ele.

Modificador de Acesso static - Classe

Classe **`static`**

- Como `static` é um modificador relativo a **componentes de uma classe** e não de objetos, uma classe só pode ser definida como estática se estiver aninhada dentro de outra classe;
- A classe estática aninhada poderá ter objetos instanciados sem que seja necessário instanciar objetos da classe principal.

Exercício - Modificador de Acesso Static

- A empresa VED93 (em que calculamos as bonificações de funcionários com polimorfismo) precisa de mais informações sobre as bonificações de funcionários. A administração pede que criemos um mecanismo para somar o total de bonificações e contar quantos funcionários foram bonificados. Crie uma classe que seja alimentada com os valores das bonificações calculadas e forneça as informações pedidas à administração.

Modificador de Acesso Final

- Em **métodos**, o modificador `final` garante que ele não será **sobrescrito** nas subclasses;
- Em **atributos e variáveis**, o modificador `final` garante que ele terá seus valores **atribuídos somente uma vez**;
- Uma **classe** declarada final **não pode ter subclasses**, e todos os **métodos** dela são **implicitamente final** também.

Modificador de Acesso Final

- Um **método** declarado **estático** ou **privado**, implicitamente também é **final**;
- Para **variáveis e atributos constantes**, faça a atribuição logo **na declaração**.
- Atenção: Quando se declara uma **referência** como final, somente a referência não pode ser alterada, **a instância pode**.
- `String` é um exemplo de **classe final**.

Modificador de Acesso Final

```
1  public class ModificadorFinal{
2    private final int atributo1 = 100;
3    private final int atributo2;
4    private int atributo3;
5
6    public ModificadorFinal(){
7      this.atributo2 = 10; //erro se tirar a linha
8    }
9    public ModificadorFinal(int $atributo2){
10     this.atributo2 = $atributo2;//erro se tirar a linha
11   }
12   public final void TentaModificar(int $atributo3){
13     //atributo1 = 1000; erro
14     //atributo2 = 2000; erro
15     this.atributo3 = $atributo3;
16     System.out.println(atributo1 + ", "+ atributo2+ ", "+ atributo3);
17   }
18 }
```

Virou uma constante.

TEM que inicializar nos construtores

Modificador de Acesso Final

```
1  public class TestaFinal{
2    public static void main (String[] args){
3      final int a=0;
4      final int b;
5      final ModificadorFinal m1 = new ModificadorFinal(10);
6      ModificadorFinal m2 = new ModificadorFinal(20);
7      //a=11; erro
8      b = 10;
9      //b = 11; erro
10     System.out.println(a + ", " + b);
11     m1.TentaModificar(100);
12     m2.TentaModificar(200);
13     //m1 = m2;erro
14     m1.TentaModificar(300);//dá erro?
15   }
16 }
```

Virou uma constante.

Somente a referência de m1 não pode ser alterada

Interface

- Interface em Java é um **contrato de responsabilidade** de implementação que uma classe assume;
- Isso quer dizer que a interface define assinaturas de métodos e deixa para as classes implementarem esses métodos (qual conceito que se parece com esse?)
- Uma interface pode ser usada como um tipo de dado para declarar atributos, variáveis e parâmetros de outra classe

Interface

- Interfaces **não possuem construtores** (porque não possuem instâncias);
- Para declarar uma interface, usamos a palavra-chave `interface`;
- Para indicarmos que uma classe implementa os métodos de uma interface, usamos a palavra-chave `implements`;
- Diferentemente de **herança** em Java, em que uma classe pode **estender somente uma superclasse**, uma classe pode **implementar mais de uma interface**.
- Uma **interface** pode implementar **outras interfaces**;

Interface

- **Não existe estado** (ou seja, atributos) associados a uma interface, qualquer atributo declarado em uma interface dever ser **público, estático e final** (ou seja, uma constante);
- Todos os métodos declarados em uma interface devem ser **públicos e abstratos**;
- Java 8 permite que as interfaces implementem métodos **padrão** (`default`) e métodos **estáticos**;
- Uma classe que **não implementa** todos os métodos especificados na interface deve ser deve **obrigatoriamente declarada abstrata**.

Interface

```
1  public interface identificavel{
2    public static final String documentoPadrao = "xxxxxxxxxxx";
3    public abstract String getIdentificador();
4    public abstract void setIdentificador(String $identificador);
5  }
```

Atributos públicos, estáticos e final

Métodos públicos e abstratos

As interfaces são totalmente equivalentes. O compilador insere as declarações padrão que foram suprimidas.

```
1  public interface identificavel{
2    String documentoPadrao = "xxxxxxxxxxx";
3    String getIdentificador();
4    void setIdentificador(String $identificador);
5  }
```

Interface

- Uma interface é normalmente utilizada quando classes **não correlacionadas** precisam compartilhar métodos e constantes permitindo o **polimorfismo** entre elas;
- Uma interface é geralmente utilizada no lugar de uma **classe abstrata** quando não há nenhuma implementação padrão a herdar (**nem atributos, nem métodos**);
- "**Programe voltado a interface** e não à implementação." (Design Patterns);
- Interfaces trazem a vantagem de **diminuir o acoplamento** de classes.

Interface

```
1   public class      PessoaFisica
2         extends     Pessoa
3         implements  Identificavel,
4                     Empregavel{
5
6     private String cpf;
7
8     public PessoaFisica() {}
9 ›   public PessoaFisica(String $nome, String $cpf)
13    public String getIdentificador(){
14      return(this.getCpf());
15    }
16    public void setIdentificador(String $identificador){
17      this.setCpf($identificador);
18    }
```

Assina o contrato de implementação da interface

implementa os métodos

Interface

Interface também faz polimorfismo

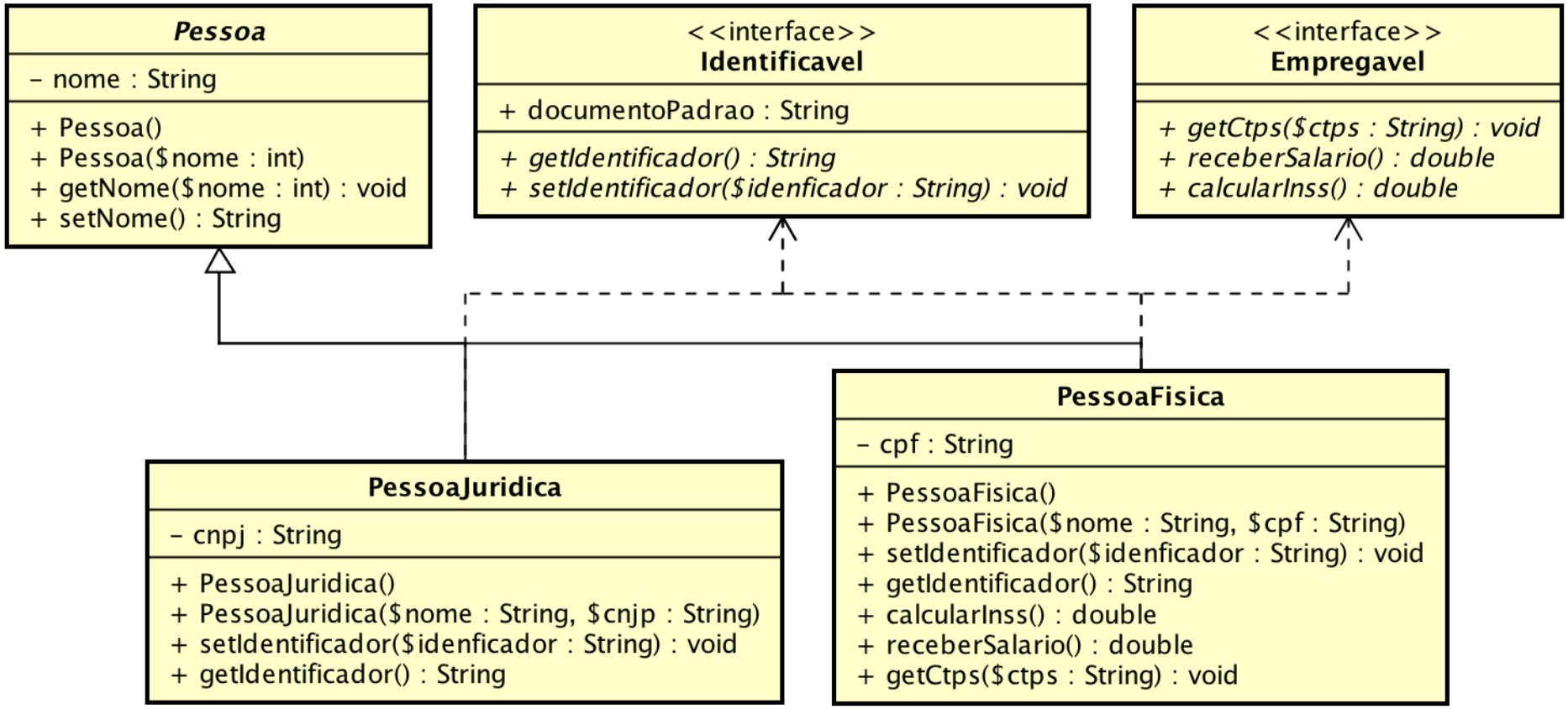
```
1  public class TestaInterface{
2    public static void main (String[] args) {
3
4      Identificavel p[] = new Identificavel[4];
5      p[0] = new PessoaFisica("John Snow","123.456.678-00");
6      p[1] = new PessoaFisica("Aira Stark","123.324.345-00");
7      p[2] = new PessoaJuridica("Patrulha da Noite","234.234.234/0001-22");
8      p[3] = new PessoaJuridica("Banco de Bravos", "567.567.567/0001-77");
9      for (int i=0;i<4;i++){
10       if ( p[i] instanceof PessoaJuridica){
11             System.out.println(((PessoaJuridica)p[i]).getNome()
12                                + " - " + p[i].getIdentificador());
13       }else{
14             System.out.println(((PessoaFisica)p[i]).getNome()
15                                + " - " + p[i].getIdentificador());
```

Interface

```
1  public class TestaInterface{
2    public static void main (String[] args) {
3
4      Identificavel p[] = new Identificavel[4];
5      p[0] = new PessoaFisica("John Snow","123.456.678-00");
6      p[1] = new PessoaFisica("Aira Stark","123.324.345-00");
7      p[2] = new PessoaJuridica("Patrulha da Noite","234.234.234/0001-22");
8      p[3] = new PessoaJuridica("Banco de Bravos", "567.567.567/0001-77");
9      for (int i=0;i<4;i++){
10       if ( p[i] instanceof PessoaJuridica){
11             System.out.println(((PessoaJuridica)p[i]).getNome()
12                                + " - " + p[i].getIdentificador());
13       }else{
14             System.out.println(((PessoaFisica)p[i]).getNome()
15                                + " - " + p[i].getIdentificador());
```

Interface também permite casting

Interface


Pessoa
- nome : String
+ Pessoa()
+ Pessoa($nome : int)
+ getNome($nome : int) : void
+ setNome() : String
<<interface>>
Identificavel
+ documentoPadrao : String
+ getIdentificador() : String
+ setIdentificador($idenficador : String) : void
<<interface>>
Empregavel
+ getCtps($ctps : String) : void
+ receberSalario() : double
+ calcularInss() : double
PessoaJuridica
- cnpj : String
+ PessoaJuridica()
+ PessoaJuridica($nome : String, $cnjp : String)
+ setIdentificador($idenficador : String) : void
+ getIdentificador() : String
PessoaFisica
- cpf : String
+ PessoaFisica()
+ PessoaFisica($nome : String, $cpf : String)
+ setIdentificador($idenficador : String) : void
+ getIdentificador() : String
+ calcularInss() : double
+ receberSalario() : double
+ getCtps($ctps : String) : void

Interface

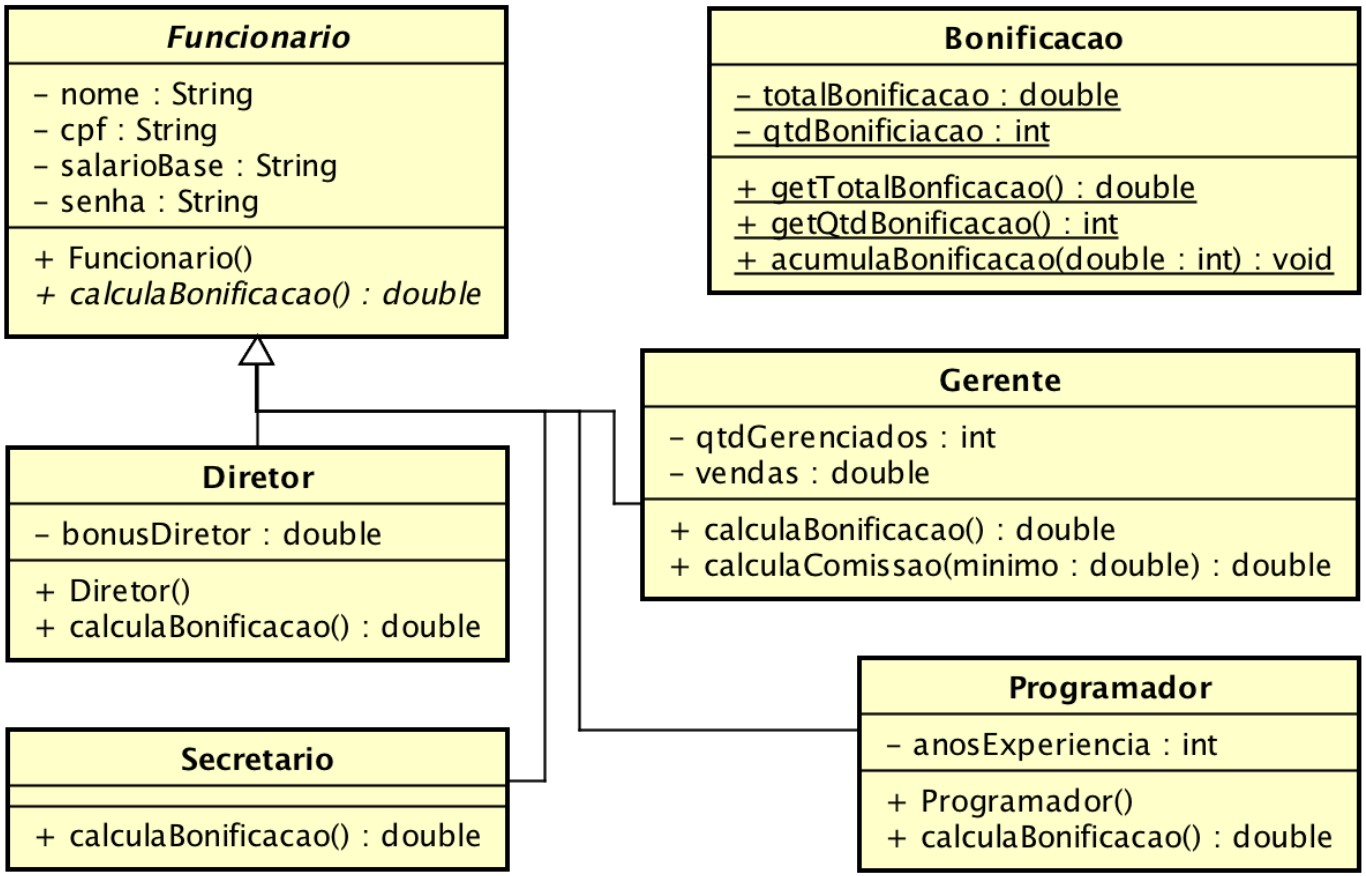
```
1  public interface Empregavel{
2    public String getCtps();
3    public double receberSalario();
4    public void calcularInss();
5  }
```

```
1  public interface Identificavel{
2    String documentoPadrao = "xxx.xxx.xxx.xxx";
3    String getIdentificador();
4    void setIdentificador(String $identificador);
5  }
```

```
1  public class PessoaFisica extends Pessoa
2                            implements Identificavel,
3                                       Empregavel{
```

Exercício - Interface

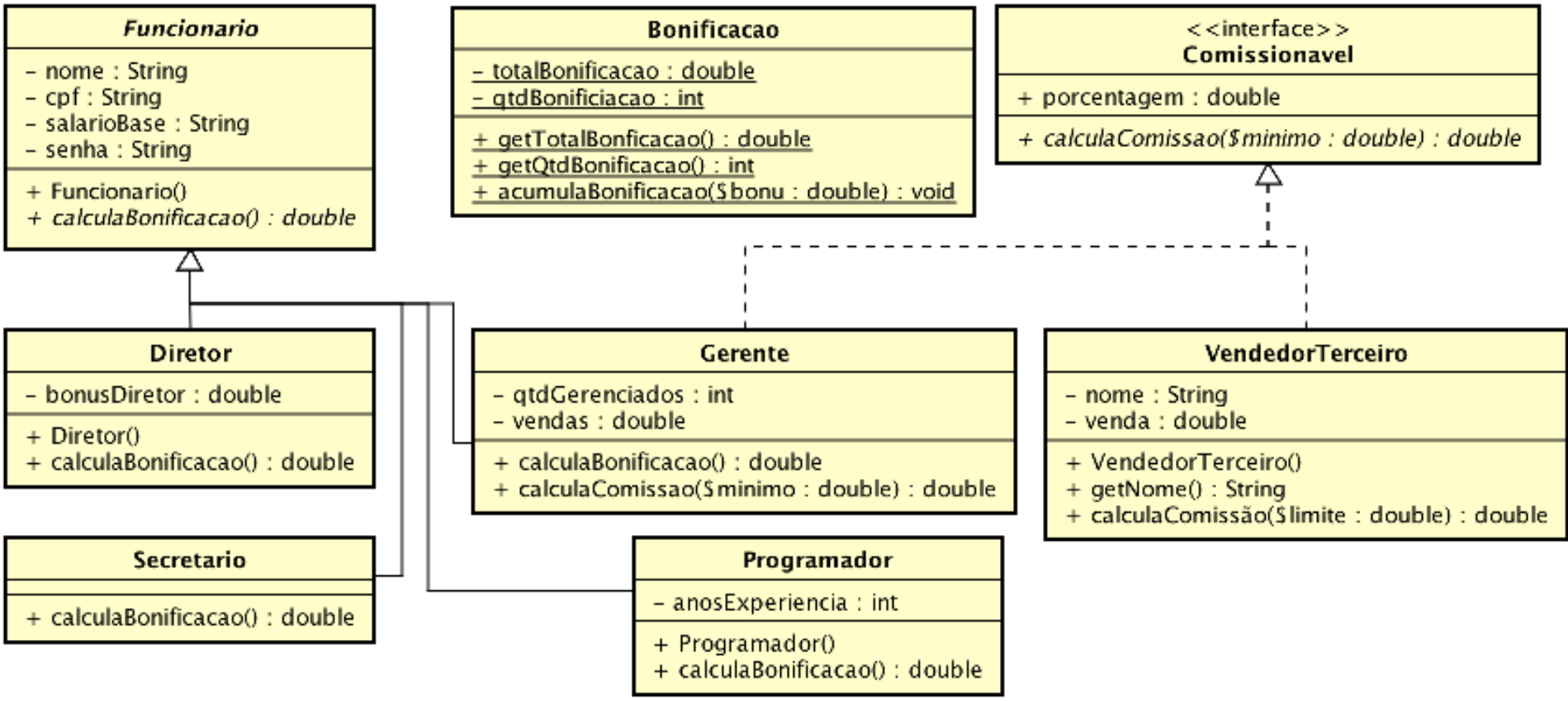
- A empresa VED93 tem seu sistema atual representado com o seguinte diagrama de classes:



Exercício - Interface

- A empresa VED93 além de bonificações também paga comissão sobre a venda de seus produtos aos gerentes e vendedores tercerizados (que não são funcionários).
- A comissão é paga da forma:
  - A porcentagem é fixa em 10% sobre a quantidade de vendas;
  - Existe um limite mínimo para as vendas. Se ele não for atingido, não há comissão;
  - Gerentes só recebem comissão se também tiverem pelo menos 5 funcionários gerenciados;
- Pense numa solução para este problema e então olhe o diagrama de classes a seguir e implemente-o.

Exercício - Interface



Exercícios : Interface

Atenção: Não copie ou cole nenhum exercício. A repetição é intencional para criar fluência na linguagem.

1. Crie uma interface `Tributavel`, com um método para calcular a tributação sobre `ContaInvestimento`, `Consorcio` e `Finaciamento`.
2. A tributação pode ter de 3 alíquotas:
   - 0 % (isento);
   - 15% (média);
   - 30% (alta);
3. Em `ContaInvestimento` . Se o investimento tiver menos de 3 meses, a tributação é alta. Se tiver entre 3 meses e 1 anos a tributação é média. Mais de um ano e um investimento é isento.
4. Em `Financiamento` e `Consórcio`, a tributação aplicada é média sobre o montante.

Exercícios : Interface

5. Na classe `AplicacaoFinanceira`:
    1. Crie um vetor `tributaveis` da interface `Tributavel` com 5 posições;
    2. Copie o vetor `produtos` para as 4 primeiras posições de `tributaveis`;
    3. Copie a posição 5 do vetor `contas` (a que possue o objeto da classe `ContaInvestimento`) para a 5a posição de `tributaveis`.
    4. Mostre a `tributacao()` de todos os elementos do vetor `tributaveis`.