Algoritmos e Estruturas de Dados

Conceitos Básicos de POO

Autores:
Carlos Urbano
Catarina Reis
José Magno
Marco Ferreira
(adaptado do original de Vitor Carreira)

Conceitos Básicos de POO

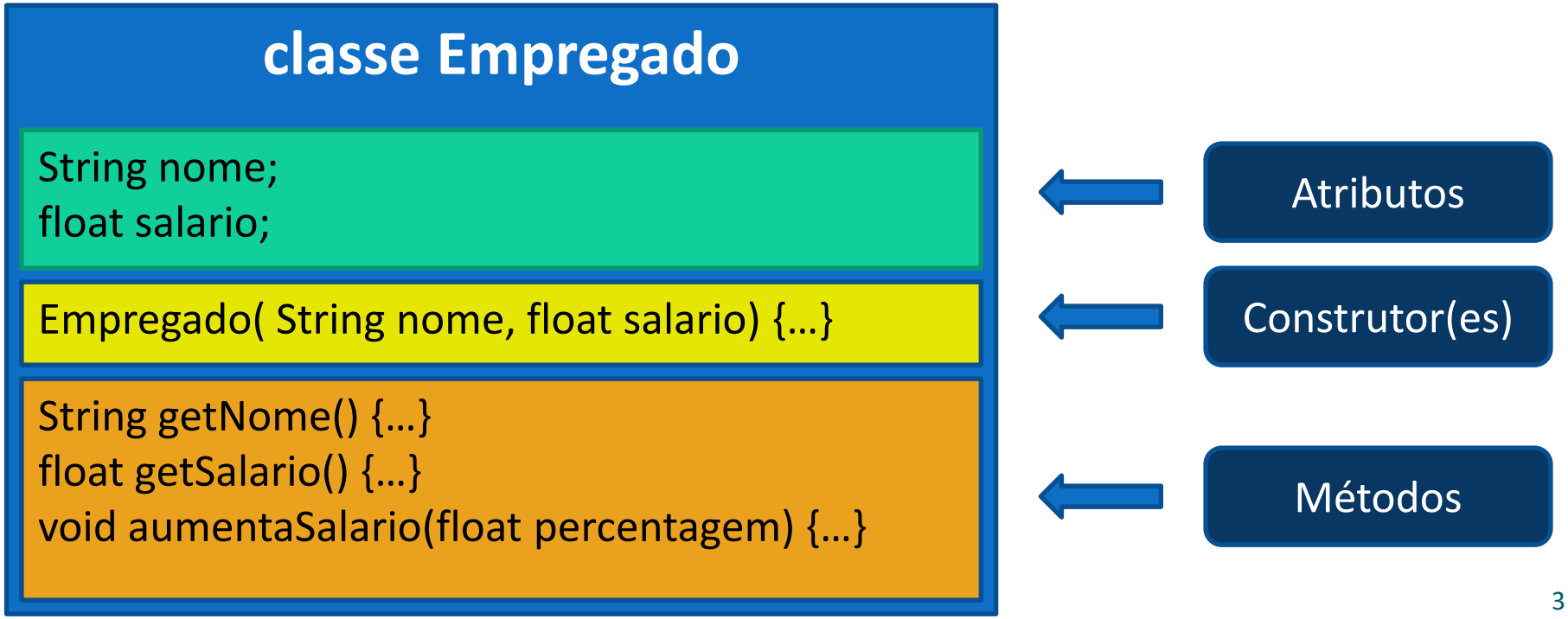
- Objeto
- Classe
- Encapsulamento
- Herança
- Polimorfismo
- Outros aspetos

Conceitos Básicos de POO

- **Objeto**
  - Combinação específica de **dados** e **métodos** que permitem processar e comunicar com esses dados
  - Os dados (**atributos**) definem o **estado** ou as **propriedades** do objeto e os **métodos** o seu **comportamento**
- **Classe**
  - Define o **tipo** de um objeto
  - Normalmente, aos objetos dá-se o nome de **instâncias da classe** que os define, uma vez que o seu **tipo** é o **nome** dessa classe

Conceitos Básicos de POO (objeto e classe)

- Exemplo
  - Consideremos a classe Empregado cujos atributos são `nome` e `salario` e métodos são `getNome()`, `getSalario()` e `aumentaSalario()`



Conceitos Básicos de POO (objeto e classe)

- A **classe** deve ser vista como um **molde** que permite construir instâncias (objetos) com estados (atributos) concretos.
- Os objetos abaixo são exemplos de instância da classe Empregado, onde cada um tem estados concretos

"Joaquim Fonseca"
1500

"Jorge Rodrigues"
1000

"Manuel da Silva"
2500

"Carlos Martins"
1500

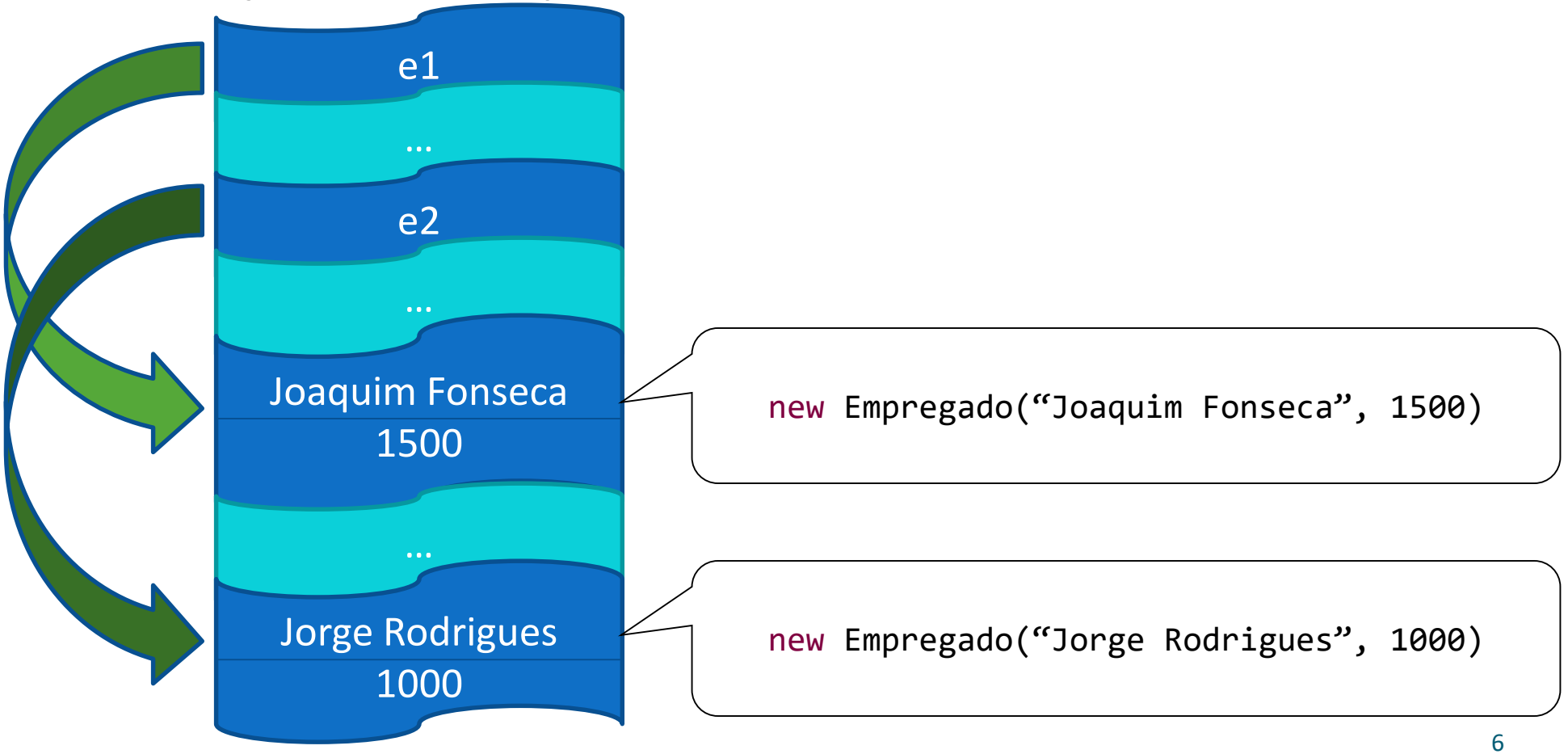
Conceitos Básicos de POO (objeto e classe)

- A **instanciação** (criação de objetos) é efetuada recorrendo ao operador **`new`** seguido do nome da classe. Desta forma, utiliza-se um construtor para construir um objeto.
- O operador **`new`** reserva um bloco de memória suficiente para guardar todos os dados de um objeto
- O programador não tem controlo sobre a localização do objeto. Assim, o conceito de ponteiro deixa de existir em Java
- Para construção dos objetos anteriores seria:

```
Empregado e1 = new Empregado("Joaquim Fonseca", 1500);
Empregado e2 = new Empregado("Jorge Rodrigues", 1000);
Empregado e3 = new Empregado("Manuel da Silva", 2500);
Empregado e4 = new Empregado("Carlos Martins", 1500);
```

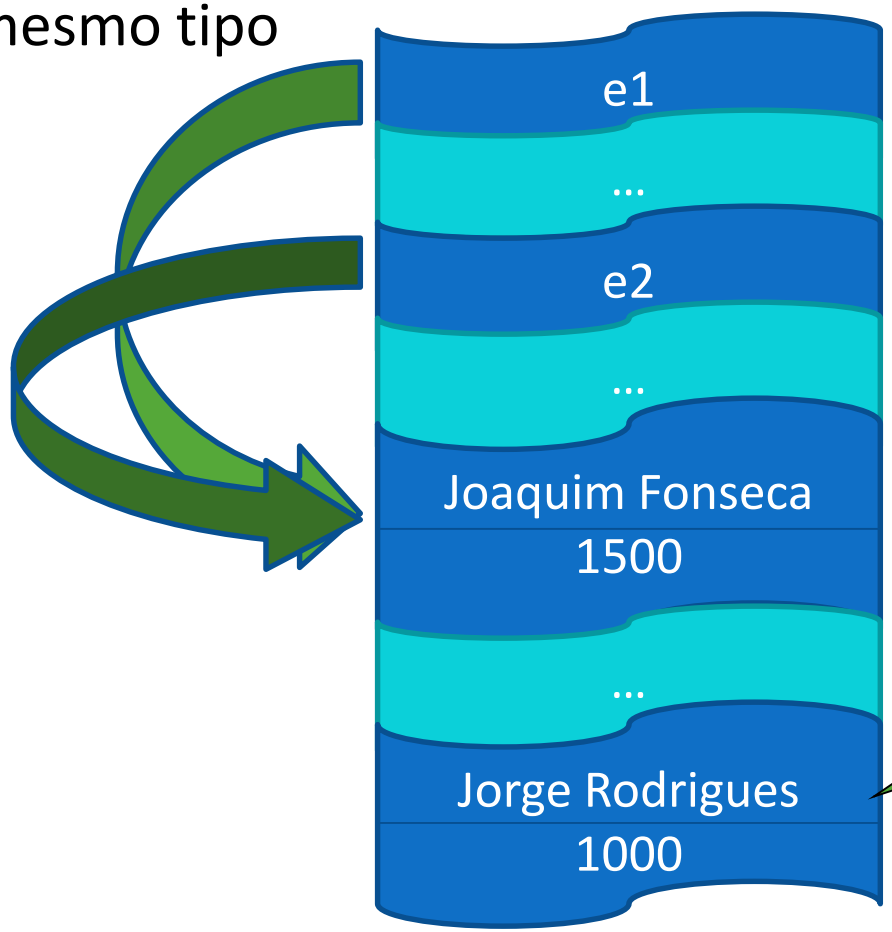
Conceitos Básicos de POO (objeto e classe)

- É importante distinguir a diferença entre uma **referência** (localização de um objeto) e uma **variável**



Conceitos Básicos de POO (objeto e classe)

- Durante a execução de um programa, uma variável pode guardar outras referências, desde que, as referências sejam do mesmo tipo



Por exemplo: `e2 = e1;`

Este objeto deixou de ser referenciado. Assim, deixou de ser um objeto válido.
O espaço ocupado pelo mesmo pode ser libertado, i.e., o objeto pode ser destruído.

Conceitos Básicos de POO (objeto e classe)

- Manipulação de objetos
  - Para aceder aos membros de um objeto é necessário executar uma instrução com a seguinte sintaxe:
    - Acesso aos atributos

      ```
      referência_objeto.nome_do_atributo
      ```

    - Invocar métodos

      ```
      referência_objeto.nome_do_método(lista de parâmetros)
      ```

  - Na terminologia POO, sempre que se invoca um método num objeto significa que se envia uma mensagem para um objeto. Ao objeto que recebe a mensagem dá-se o nome de recetor

Conceitos Básicos de POO

- **Encapsulamento**
  - Encapsulamento permite a um objeto proteger o acesso e o processamento da sua informação
  - Nas linguagens POO existem 3 níveis de acesso: público, protegido e privado (o Java acrescenta o acesso local)
  - Só é possível comunicar com o objeto através dos seus métodos públicos
  - O estado interno do objeto é protegido e apenas pode ser modificado por ele próprio

Conceitos Básicos de POO (objeto e classe)

- O exemplo anterior ficaria

```
public class Empregado {
  private String nome;    //ou protected
  private float salario; //ou protected

  public Empregado(String nome, float salario) {
    this.nome = nome;
    this.salario = salario;
  }

  public String getNome() {
    return nome;
  }

  public float getSalario() {
    return salario;
  }

  public void aumentaSalario(float percentagem) {
    salario += (salario * percentagem / 100);
  }
}
```

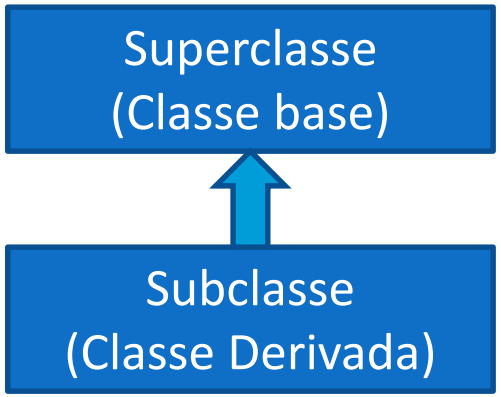
Conceitos Básicos de POO (encapsulamento)

- O acesso protegido permite, para além da própria classe, que todas as subclasses (herança), independentemente da *package* a que pertencem, possam aceder aos membros da classe declarados como protected. As classes pertencentes à mesma *package* também têm acesso aos membros protegidos

		Quem pode aceder?			
Tipo de acesso	Palavra reservada	Classe	Subclasse	Package	Todos
público	public	✓	✓	✓	✓
protegido	protected	✓	✓	✓	
privado	private	✓			
local	---	✓		✓	

Conceitos Básicos de POO

- **Herança**
  - O conceito de herança em POO consiste em criar uma classe reutilizando propriedades (atributos) e comportamentos (métodos) de uma classe já existente
  - Frequentemente, a classe que pretendemos criar é uma especialização de uma classe já existente. A herança permite aproveitar ao máximo o código da classe existente

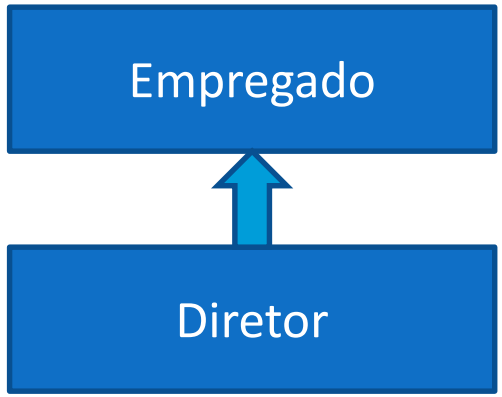


Superclasse - classe existente

Subclasse - classe criada a partir da classe existente e que herda as propriedades e comportamentos da classe base
O programador só tem de adicionar as propriedades e comportamentos específicos da nova subclasse

Conceitos Básicos de POO (herança)

- No exemplo anterior, vamos supor que se trata de uma empresa onde existem **vários tipos de empregados**
  - A empresa tem um quadro superior de **diretores** e que estes, para além do salário, possuem uma determinada **participação** na empresa (percentagem de ações).
  - Teríamos a seguinte hierarquia de classes



Conceitos Básicos de POO (herança)

- A implementação seria

```
public class Diretor extends Empregado {
  private float percentagemAcoes;

  public Diretor(String nome, float salario, float percentagemAcoes) {
    super(nome, salario);
    this.percentagemAcoes = percentagemAcoes;
  }
  public float getPercentagemAcoes() {
    return percentagemAcoes;
  }
  public void aumentaParticipacao(float percentagem) {
    percentagemAcoes *= (1 + percentagem / 100);
  }
}
```

- A subclasse herda da superclasse todos os membros da instância
- Neste exemplo, a subclasse Diretor herda da superclasse Empregado:
  - Os atributos: nome e `salario`
  - O método: `aumentaSalario`

Conceitos Básicos de POO (herança)

- **Membros herdados e acessos**
  - Uma subclasse acede diretamente (da superclasse)
    - a todos os membros da instância que são declarados como public ou protected
    - a todos os membros locais da instância (pertencentes à *package*), caso a subclasse pertença à mesma *package* da superclasse
  - Uma subclasse não consegue aceder:
    - aos membros da instância declarados como private
    - aos membros locais da instância, caso a subclasse pertença a uma *package* diferente da *package* da superclasse

Conceitos Básicos de POO

- **Polimorfismo**
  - Polimorfismo é um conceito teórico que significa que uma determinada variável pode representar múltiplos tipos de objetos desde que estejam relacionados entre si através de uma superclasse comum
  - Uma forma mais simples de explicar o conceito de polimorfismo é dizer que uma variável de uma dada classe, representa não só os objetos dessa classe mas também os objetos de todas as suas subclasses
    - Exemplo:
      - Se um Diretor é um Empregado, o conceito de polimorfismo diz que um objeto do tipo Empregado pode guardar um objeto do tipo Diretor

Conceitos Básicos de POO (polimorfismo)

- Seguindo o nosso exemplo, poderíamos ter

```
Empregado[] empregados = new Empregado[4];
empregados[0] = new Empregado(”Joaquim Fonseca", 1500);
empregados[1] = new Empregado(”Jorge Rodrigues", 1000);
empregados[2] = new Diretor(”Belmiro Azedo", 5000, 10);
empregados[3] = new Empregado(”Carlos Martins", 50000);
```

- Para aumentar o salário de todos os trabalhadores (empregados e diretores) bastaria utilizar o seguinte código:

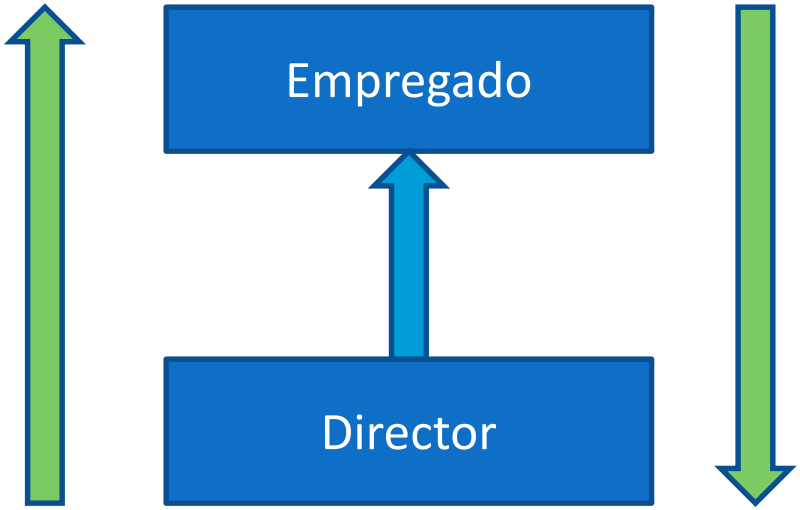
```
for (int i = empregados.length - 1; i >= 0; i--) {
        empregados[i].aumentaSalario(2.4F);
}
```

  - *Dynamic binding* (*binding* dinâmico) - durante a execução (*runtime*) é determinado o método a invocar e o objeto recetor

Conceitos Básicos de POO (polimorfismo)

- Upcast e downcast

Upcast
- Sempre seguro
- Automático (aplicação do polimorfismo)



Downcast
- Sempre verificado
- Explícito

```
for (int i = empregados.length-1; i >= 0; i--) {
    if (empregados[i] instanceof Diretor) {
        ((Diretor) empregados[i]).aumentaParticipacao(5);
    }
}
```

Conceitos Básicos de POO (polimorfismo)

- ***Overloading* e *Overriding* de métodos**
  - *Overloading* consiste na existência de vários métodos com o mesmo nome mas com assinaturas diferentes
    - São distinguidos em tempo de compilação (*static binding*)
  - *Overriding* consiste em reimplementar (redefinir), na subclasse, um método que já existe na superclasse (assinaturas iguais)
    - São distinguidos em tempo de execução (*dynamic binding*)

Conceitos Básicos de POO (polimorfismo)

- No nosso exemplo
  - Vamos supor que o salário de um diretor é calculado de forma diferente. Por exemplo, sempre que o salário aumente x%, a percentagem de ações aumenta 1/3*x%
    - Se a classe Diretor herda da classe Empregado o método que aumenta o salário, como é que se consegue atualizar apenas o salário do diretor?
  - A solução para o problema anterior consiste em redefinir (*override*) o método `aumentaSalario`
  - Ambas as classes (Diretor e Empregado) possuem o mesmo método mas com implementações diferentes

Conceitos Básicos de POO (polimorfismo)

- Teríamos então

```
public class Diretor extends Empregado {
    private float percentagemAcoes;

    public Diretor(String nome, float salario, float percentagemAcoes) {
        super(nome, salario);
        this.percentagemAcoes = percentagemAcoes;
    }

    public float getPercentagemAcoes() {
        return percentagemAcoes;
    }

    @Override
    public void aumentaSalario(float percentagem) {
        salario *= (1 + percentagem / 100);
        percentagemAcoes *= (1 + percentagem / 300);
    }

    public void aumentaParticipacao(float percentagem) {
        percentagemAcoes *= (1 + percentagem / 100);
    }
}
```

Conceitos Básicos de POO (polimorfismo)

- O método `aumentaSalario` da classe **Diretor** poderia, para aproveitar o código da superclasse, ser reescrito da seguinte forma:

```
@Override
public void aumentaSalario(float percentagem) {
    super.aumentaSalario(percentagem);
    aumentaParticipacao(2 * percentagem / 3);
}
```

- A palavra super é uma referência para a superclasse e pode ser utilizada para:
  - Chamar construtores da superclasse: `super(lista de parâmetros)`
  - Chamar métodos da superclasse: `super.nomeMetodo(lista de parâmetros)`
- A implementação anterior tem a vantagem de permitir concentrar o cálculo do salário de um empregado num único ponto

Conceitos Básicos de POO (outros aspetos)

- **Membros da instância**
  - Todos os membros (métodos e atributos) que constam do exemplo anterior são chamados membros da instância
  - Para aceder aos membros da instância é necessário ter sempre um objeto recetor, i.e., uma referência para o objeto
  - É por esse motivo que o acesso aos membros da instância tem a sintaxe:

```
referencia_objeto.nome_do_atributo
referência_objeto.nome_do_método(lista de parâmetros)
```

Conceitos Básicos de POO (outros aspetos)

- **Membros da classe**
  - Os membros de uma classe são atributos ou métodos partilhados por todos os objetos dessa classe
  - Para definir um membro de uma classe, coloca-se a palavra **`static`** no início da definição do método ou atributo
  - Uma denominação comum para membros da classe é a de membros estáticos
  - Para aceder aos membros de uma classe, não é necessário possuir uma referência para um objeto (objeto recetor)
  - O acesso aos membros de uma classe processa-se da seguinte forma:

```
Nome_da_Classe.nome_do_atributo
Nome_da_Classe.nome_do_método(lista de parâmetros)
```

Conceitos Básicos de POO (outros aspetos)

- **Classe Abstrata**
  - Uma classe abstrata representa conceitos abstratos
  - Não faz sentido (nem é possível) criar uma instância de uma classe abstrata
  - Utiliza-se a palavra `abstract` para indicar que uma classe é abstrata
  - Uma classe abstrata pode conter vários métodos abstratos (métodos sem implementação)
  - Também se utiliza a palavra `abstract` para indicar que um método não tem implementação
  - As subclasses que derivam de classes abstratas têm de implementar todos os métodos abstratos
    - Caso não o façam, as subclasses têm de ser declaradas como sendo abstratas

Conceitos Básicos de POO (outros aspetos)

- **Interface**
  - As interfaces estendem o conceito de classe abstrata
  - As classes abstratas podem possuir atributos, métodos abstratos e concretos
  - Uma interface comporta-se com uma classe abstrata pura. Apenas possui métodos abstratos (também pode incluir constantes). Não pode incluir atributos ou métodos concretos.
  - As interfaces permitem a interação entre classes não-relacionadas
  - Uma classe pode implementar várias interfaces
  - Uma interface pode derivar de múltiplas interfaces

Conceitos Básicos de POO (outros aspetos)

- **Interface**
  - Por omissão, numa interface:
    - Todos os métodos são públicos
    - Todos os atributos são constantes públicas
    - As seguintes definições são equivalentes:

```
public interface SuperficieFechada {
  double PI = 3.14159265359;
  double area();
  double perimetro();
}
```

```
public interface SuperficieFechada {
  public static final double PI = 3.14159265359;
  public double area();
  public double perimetro();
}
```