HERANÇA E POLIMORFISMO

Ramon Lummertz

RELEMBRANDO...

Classe e Objeto

Vantagens da orientação a objetos

Construtor

Polimorfismo

Encapsulamento

Métodos e atributos Estáticos

HERANÇA

Uma forma de reutilização de software em que uma nova classe é criada absorvendo membros de uma classe existente e aprimorada com capacidades novas ou modificadas.
Permite economizar tempo durante o desenvolvimento de um programa baseando novas classes no software existente testado, depurado e de alta qualidade.
Aumenta a probabilidade de que um sistema será implementado e mantido eficientemente.

HERANÇA

Ao criar uma classe, em vez de declarar membros completamente novos, você pode designar que a nova classe deve herdar membros de uma classe existente.

Classe existente é a superclasse.
Nova classe é a subclasse.

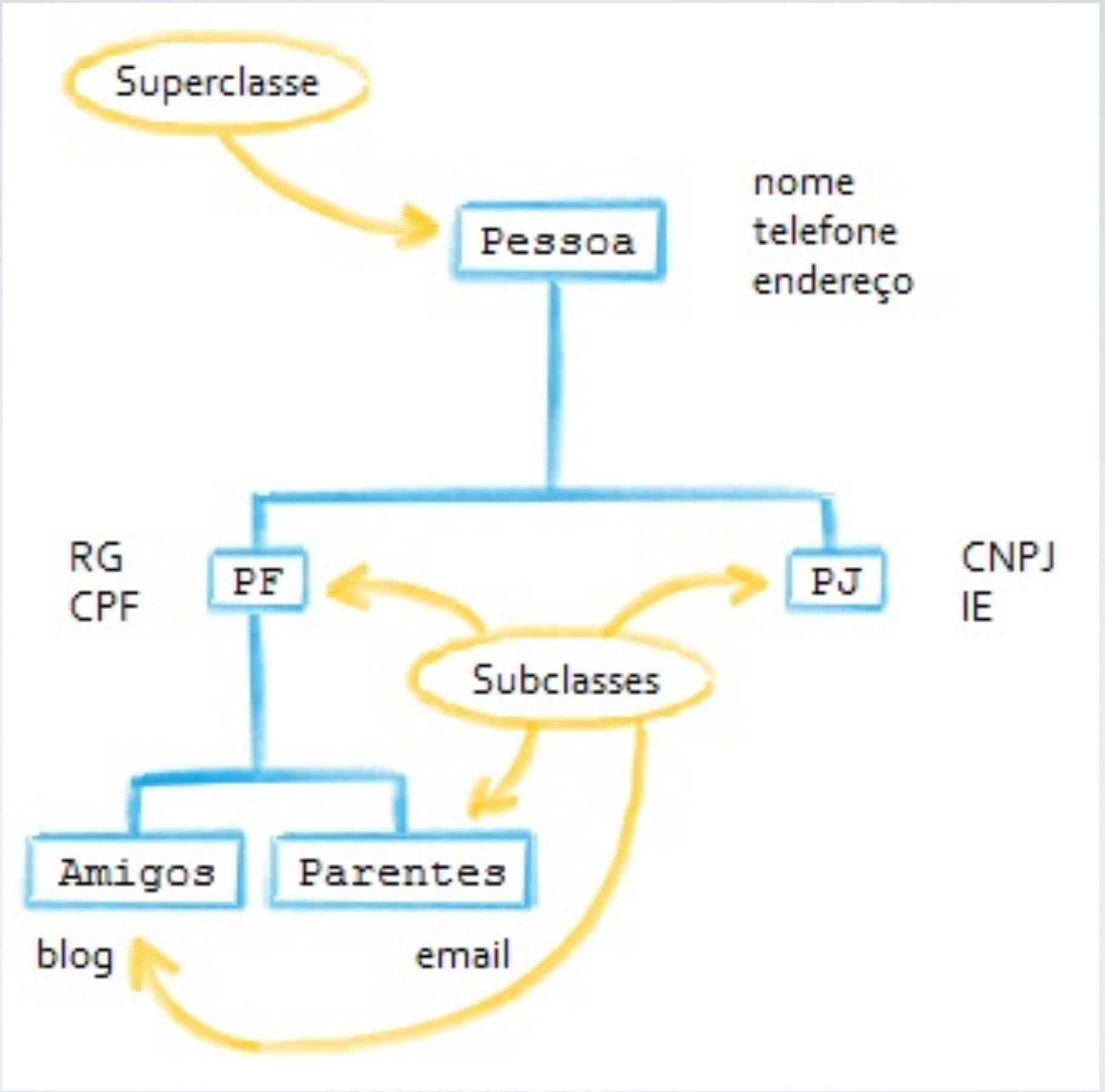
Cada subclasse pode ser uma superclasse de futuras subclasses.
Uma subclasse pode adicionar seus próprios campos e métodos.
Uma subclasse é mais específica que sua superclasse e representa um grupo mais especializado de objetos.
A subclasse expõe os comportamentos da sua superclasse e pode adicionar comportamentos que são específicos à subclasse.
É por isso que a herança é às vezes conhecida como especialização

HERANÇA

A hierarquia de classes inicia com a classe Object (no pacote java.lang)

Toda classe em Java estende (ou —"herda de") Object direta ou indiretamente.

O Java só suporta herança simples, na qual cada classe é derivada de exatamente uma superclasse direta.


Superclasse
Pessoa
nome
telefone
endereço
RG
CPF
PF
PJ
CNPJ
IE
Subclasses
Amigos
Parentes
blog
email

É UM

Um filho objeto é um objeto
Ex
O triangulo é uma forma
O gato é um animal
Banheira é um banheiro ?

HERANÇA

Os membros private de uma superclasse permanecem ocultos em suas subclasses.
Eles somente podem ser acessados pelos métodos public ou protected herdados da superclasse.

Os métodos de subclasse podem referenciar membros public e protected herdados da superclasse simplesmente utilizando os nomes de membro.

Quando um método de subclasse sobrescrever um método de superclasse herdado, o método de superclasse pode ser acessado a partir da subclasse precedendo o nome do método de superclasse com a palavra-chave super e um ponto (.) separador.

HERANÇA

Construtores não são herdados.
A primeira tarefa de um construtor de subclasse é chamar o construtor da sua superclasse direta, explícita ou implicitamente. Assegura que as variáveis de instância herdadas da superclasse sejam inicializadas adequadamente.

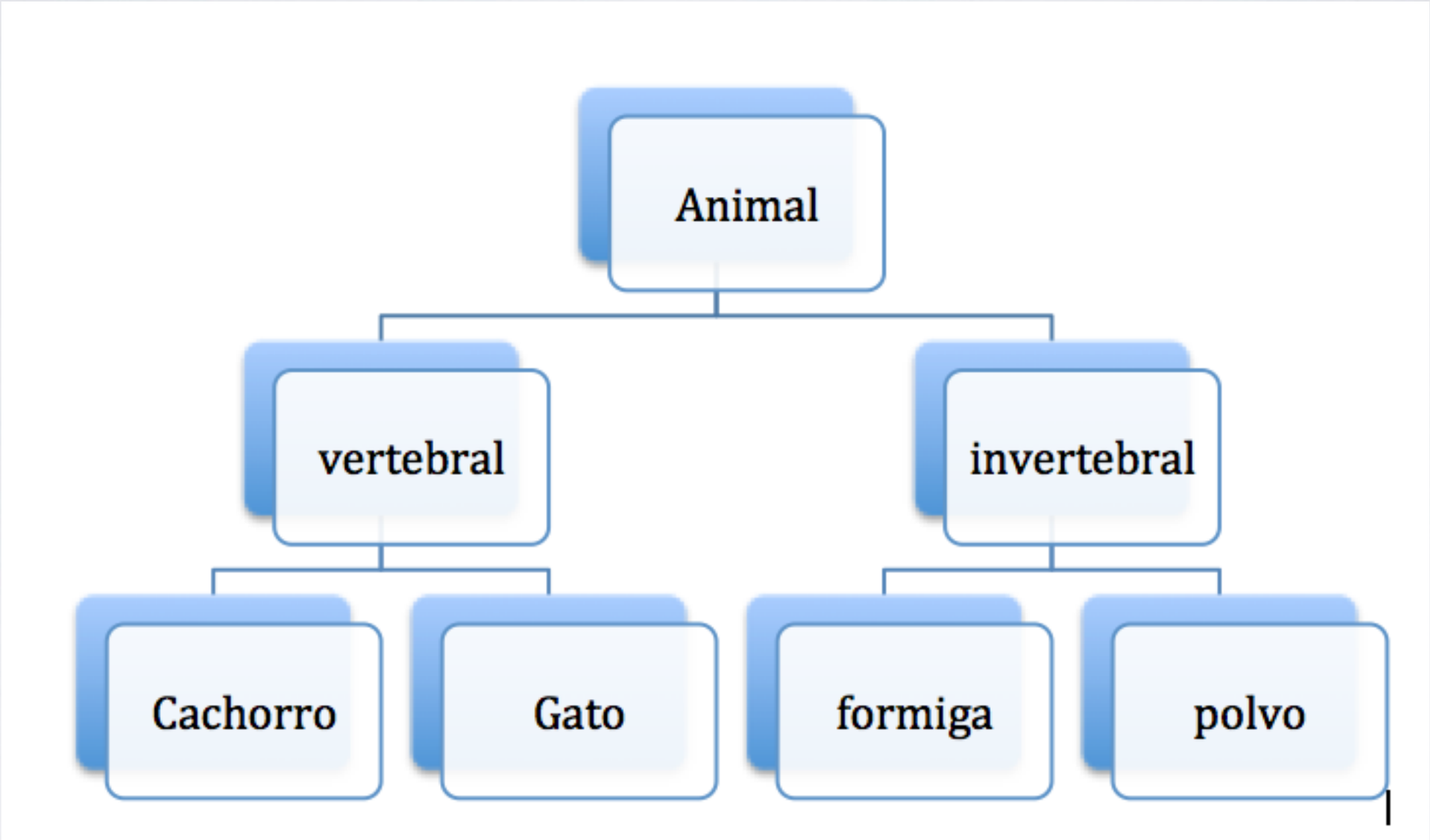
Se o código não incluir uma chamada explícita para o construtor de superclasse, o Java chama implicitamente o construtor padrão (ou construtor sem argumento) da superclasse.

O construtor padrão de uma classe chama o construtor padrão ou sem argumentos da superclasse.

HERANÇA

Quando um método ou construtor possui a mesma assinatura numa subclasse, temos o polimorfismo de sobrescrita.

UM EXEMPLO..


Animal
vertebral
invertebral
Cachorro
Gato
formiga
polvo

VOLTANDO A LIVROS...

HERANÇA

Um livro impresso é igual a um ebook?

HERANÇA

Um novo requisito apareceu!!!

Nossas vendas de livros, serão distintas para ebooks e livros. Afinal o livro fisico possui uma taxa adicional de impressão, que é 15% do valor do livro.

Qual a solução desse problema?

O PROBLEMA DOS EBOOKS..

Tentativa 1

```
public class Livro {
    private String nome;
    private String descricao;
    private double valor;
    private String isbn;
    private Autor autor;
    private boolean impresso;

public Livro(Autor autor) {
    this.autor = autor;
    this.isbn = "000-00-00000-00-0";
    this.impresso = true;
}

public boolean aplicaDescontoDe(double porcentagem) {
    if (porcentagem > 0.3) {
        return false;
    } else if (!this.impresso && porcentagem > 0.15) {
        return false;
   }
this.valor -= this.valor * porcentagem;
return true;
}
// outros métodos da classe
}
```

Tentativa 2

```
public class Ebook {
    private String nome;
    private String descricao;
    private double valor;
    private String isbn;
    private Autor autor;
    private String waterMark;
    public void setWaterMark(String waterMark) {
        this.waterMark = waterMark;
}

    public String getWaterMark() {
        return waterMark;
}
```

HERANÇA

Solução mais adequada

```
public class Ebook extends Livro {
    private String waterMark;
    public Ebook(Autor autor) {
        super(autor);
}

    public void setWaterMark(String waterMark) {
        this.waterMark = waterMark;
}

    public String getWaterMark() {
        return waterMark;
}}
```

Ao utilizar a palavra reservada extends, estamos dizendo que um Ebook (*subclasse*) **herda** tudo o que a classe Livro (*superclasse*) tem. Portanto, mesmo sem ter nenhum desses métodos declarados diretamente na classe Ebook, podemos usar os métodos na classe livro

delegate constructor

Como a classe Livro tinha um construtor obrigando a passagem de um Autor como parâmetro, ao herdar de um Livro, a classe Ebook também herdou essa responsabilidade. Repare que utilizamos a palavra super para delegar a responsabilidade para a *superclasse* que já tem esse comportamento bem definido.

```
public Ebook(Autor autor) {
    super(autor);
}
```

Herança múltipla

Uma regra importante da herança em Java é que nossas classes só podem herdar diretamente de **uma** classe pai. Ou seja, não há herança múltipla como na linguagem C++. Mas sim, uma classe pode herdar de uma classe que herda de outra e assim por diante. Você pode encadear a herança de suas classes.

CLASSES_ABSTRATAS

Pode-se dizer que as classes abstratas servem como "modelo" para outras classes que dela herdem, não podendo ser instanciada por si só. Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada herdando dela e então instanciar essa nova classe. Os métodos da classe abstrata devem então serem sobrescritos nas classes filhas.

Por exemplo, é definido que a classe "Animal" seja herdada pelas subclasses "Gato", "Cachorro", "Cavalo", mas ela mesma nunca pode ser instanciada.

CLASSES_ABSTRATAS

```java

abstract class Funcionario {

    public String nome;
    public double salario;
    public String departamento;


    public void calculaAumento(int p) {
        this.salario *=p;
    }

}

```

```java
public class Arigo extends Funcionario {

}
```

```java
public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Arigo f= new Arigo();

        f.calculaAumento(2);
    }

}
```

```java
public static void main(String[] args) {
    // TODO Auto-generated method stub

    Funcionario f= new Funcionario();

}
```

Não funciona

```java
public class Arigo extends Funcionario {

    @Override
    public void calculaAumento(int p) {
        // TODO Auto-generated method stub
        super.calculaAumento(p-1);

    }

}
```

METODOS ABSTRATOS

```
1
2  abstract class Funcionario {
3
4      public String nome;
5      public double salario;
6      public String departamento;
7
8
9      public void calculaAumento(int p) {
10         this.salario *=p;
11     }
12
13     public abstract void calcularPontos(){
14
15     }
16 }
17
```

Não funciona

```
14
15     private abstract void verHorarios();
16
17
18 }
19
```

```
    }

    public abstract void calcularPontos();
}
```

```
1
2  public class Arigo extends Funcionario {
3
4      @Override
5      public vo
6          // TO
7          super
8
9      }
10
11 }
```

Add unimplemented methods
Make type 'Arigo' abstract
Rename in file (⌘2 R)
Rename in workspace (⌥⌘R)

1 method to implement:
- Funcionario.calcularPontos()

```
2  public class Arigo extends Funcionario {
3
4      @Override
5      public void calculaAumento(int p) {
6          // TODO Auto-generated method stub
7          super.calculaAumento(p-1);
8
9      }
10
11     @Override
12     public void calcularPontos() {
13         // TODO Auto-generated method stub
14
15     }
16
17 }
18
```

CLASSES INTERFACE

As interfaces são padrões definidos através de contratos ou especificações. Um contrato define um determinado conjunto de métodos que serão implementados nas classes que assinarem esse contrato. Uma interface é 100% abstrata, ou seja, os seus métodos são definidos como abstract, e as variáveis por padrão são sempre constantes (static final).

Uma interface é definida através da palavra reservada "interface". Para uma classe implementar uma interface é usada a palavra "implements".

Como a linguagem Java não tem herança múltipla, as interfaces ajudam nessa questão, pois bem se sabe que uma classe pode ser herdada apenas uma vez, mas pode implementar inúmeras interfaces. As classes que forem implementar uma interface terão de adicionar todos os métodos da interface ou se transformar em uma classe abstrata, veja nos exemplos abaixo

```java
public interface Aluno {

    public abstract void AcessoPearson();
}
```

```java
public interface Funcionario {

    public abstract void calcularPonto();
}
```

```java
public class FuncionarioUlbra implements Aluno,Funcionario,Professor {
    @Override
    public void AcessoPearson() {

    }

    @Override
    public void acessoCapes() {

    }

    @Override
    public void calcularPonto() {

    }
}
```

REFERENCIAS

Turini, Rodrigo. **Desbravando Java e Orientação a Objetos:** um guia para o iniciante. Casa do Codigo