ARA0066 - PROGRAMAÇÃO ORIENTADA A OBJETOS EM JAVA

ARA0075

Aula - 04

Formato: 3 horas de aulas no formato presencial.

Carga horária: 80

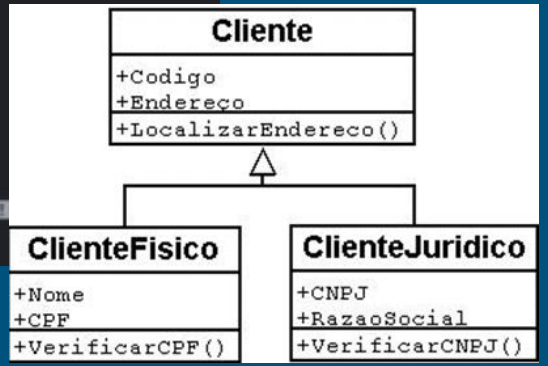
Objetivos - Herança

Herança é um dos pontos chave de programação orientada a objetos (POO). A ideia de herança é facilitar a programação. Uma classe A deve herdar de uma classe B quando podemos dizer que A é um B. Por exemplo, imagine que já exista uma classe que defina o comportamento de um dado objeto da vida real, por exemplo, animal.

Wikibooks
https://pt.wikibooks.org › wiki › Herança

Programação Orientada a Objetos/Herança - Wikilivros

Sobre trechos em destaque



Herança

Neste estudo, examinaremos os conceitos de herança e polimorfismo. Começaremos pelos aspectos elementares de herança e, em seguida, vamos aprofundar tal discussão, criando condições para analisarmos o polimorfismo.

O termo **herança em OO** define um tipo de relação entre objetos e classes, baseado em uma hierarquia.

Dentro dessa relação hierárquica, classes podem herdar características de outras classes situadas acima ou transmitir suas características às classes abaixo.

Uma classe situada hierarquicamente acima é chamada de **superclasse**, enquanto aquelas situadas abaixo chamam-se **subclasses**.

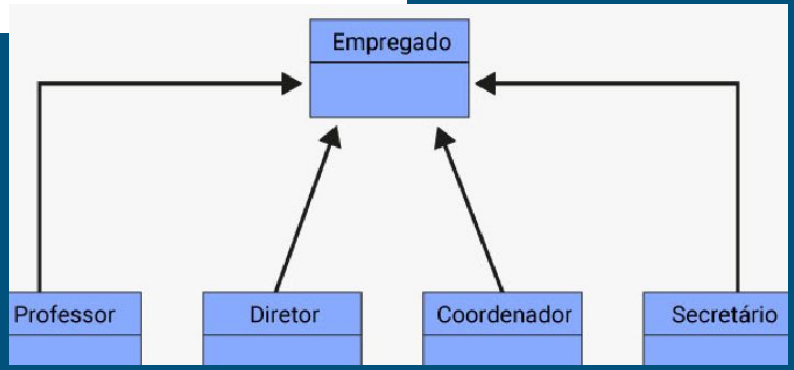
Essas classes podem ser, também, referidas como classe base ou pai (superclasse) e classe derivada ou filha (subclasse).

Herança

A herança nos permite reunir os métodos e atributos comuns numa superclasse, que os leva às classes filhas. Isso evita repetir o mesmo código várias vezes.

Outro benefício está na manutenibilidade: caso uma alteração seja necessária, ela só precisará ser feita na classe pai, e será automaticamente propagada para as subclasses.

A imagem a seguir apresenta um diagrama UML que modela uma hierarquia de herança. Nela, vemos que “Empregado” é pai (superclasse) das classes “Professor”, “Diretor”, “Coordenador” e “Secretario”. Essas últimas são filhas (subclasses) da classe “Empregado”.



Herança - Herança múltipla

Essa situação é denominada herança múltipla e, apesar de a POO aceitar a herança múltipla, a linguagem Java não a suporta para classes. Veremos as implicações da herança múltipla em outra ocasião, quando explorarmos mais a fundo o conceito de herança.

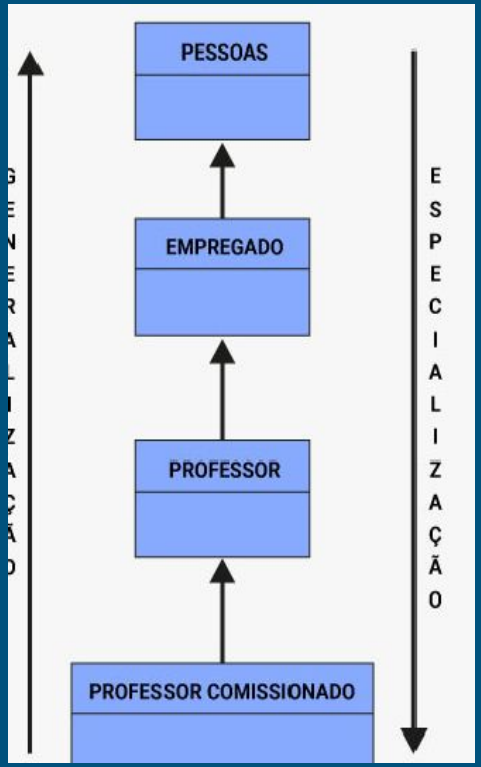
Comentário

Apesar de não permitir herança múltipla de classes, a linguagem permite que uma classe herde de múltiplas interfaces. Aliás, uma interface pode herdar de mais de uma interface pai. Diferentemente das classes, uma interface não admite a implementação de métodos. Esta é feita pela classe que implementa a interface.

Herança - Genealogia de classes

Examinando a genealogia das classes da imagem seguinte, podemos notar que, à medida que descemos na hierarquia, lidamos com classes cada vez mais específicas. De forma oposta, ao subirmos na hierarquia, encontramos classes cada vez mais gerais.

Essas características correspondem aos conceitos de generalização e especialização da OO.



Herança - Interfaces

A interface é um recurso muito utilizado em Java, bem como na maioria das linguagens orientadas a objeto, para "obrigar" a um determinado grupo de classes a ter métodos ou propriedades em comum para existir em um determinado contexto, contudo os métodos podem ser implementados em cada classe de uma maneira diferente. Pode-se dizer, a grosso modo, que uma interface é um contrato que quando assumido por uma classe deve ser implementado.

```java
public interface Veiculo {

    public String getNome();
    public String getId();
}
```

```java
public class Carro implements Veiculo,Motor{

    @Override
    public String getId() {

    }

    @Override
    public String getNome() {

    }

    @Override
    public String getFabricante() {

    }

    @Override
    public String getModelo() {

    }

  }
}
```

```java
public interface Motor {

    public String getModelo();
    public String getFabricante();
}
```

Herança - Sintaxe básica

Java

```java
public class ProfessorComissionado extends Professor {
//..
}
```

A classe **"Professor"** deve declarar **"Empregado"** como sua superclasse.

"Empregado" deve declarar **"Pessoa"** como sua superclasse.

Herança - Sintaxe básica

A sintaxe é análoga para o caso das interfaces, exceto que nesse caso pode haver mais de um identificador de superinterface. O código a seguir mostra um exemplo baseado na imagem Diagrama de classes – herança, considerando que "ProfessorComissionado", "Professor" e "Diretor" sejam interfaces.

Nesse exemplo, a herança múltipla pode ser vista pela lista de superinterfaces ("Professor" e "Diretor") que se segue ao modificador *"extends"*.

Java

```
1 public interface ProfessorComissionado extends Professor, Diretor {
2 //…
3 }
```

Herança - Detalhando

A classe **“Professor”** deve declarar **“Empregado”** como sua superclasse.

“Empregado” deve declarar **“Pessoa”** como sua superclasse.

A sintaxe é análoga para o caso das interfaces, exceto que nesse caso pode haver mais de um identificador de superinterface. O código a seguir mostra um exemplo baseado na imagem Diagrama de classes – herança, considerando que “ProfessorComissionado”, “Professor” e “Diretor” sejam interfaces.

Nesse exemplo, a herança múltipla pode ser vista pela lista de superinterfaces (“Professor” e “Diretor”) que se segue ao modificador *“extends”*.

Java

```
1 public interface ProfessorComissionado extends Professor, Diretor {
2 //…
3 }
```

Herança - Detalhando

Algo interessante de se observar é que em Java todas as classes descendem direta ou indiretamente da classe “Object”. Isso torna os métodos da classe “Object” disponíveis para qualquer classe criada. O método *“equals ()”*, da classe “Object”, por exemplo, pode ser usado para comparar dois objetos da mesma classe.

Se uma classe for declarada sem estender nenhuma outra, então o compilador assume implicitamente que ela estende a classe “Object”.

Se ela estender uma superclasse, como no código, então ela é uma descendente indireta de “Object”.

Isso fica claro se nos remetermos à imagem **Herança com vários níveis de ancestralidade**. Nesse caso, a classe “Pessoa” estende implicitamente “Object”, então os métodos desta são legados às subclasses até a base da hierarquia de classes. Logo, um objeto da classe “ProfessorComissionado” terá à sua disposição os métodos de “Object”.

Herança - Visibilidade

Quando dizemos que a classe “Pessoa” é uma generalização da classe “Empregado”, isso significa que ela reúne atributos e comportamento que podem ser generalizados para outras subclasses. Esses comportamentos podem ser especializados nas subclasses. Isto é, as subclasses podem sobrescrever o comportamento modelado na superclasse. Nesse caso, a assinatura do método pode ser mantida, mudando-se apenas o código que o implementa.

Neste momento, vamos entender o funcionamento desse mecanismo para compreendermos o que é polimorfismo.

Herança - Visibilidade

Primeiramente, precisamos compreender como os modificadores de acesso operam. Já vimos que tais modificadores alteram a acessibilidade de classes, métodos e atributos. Há quatro níveis de acesso em Java:

DEFAULT

Assumido quando nenhum modificador é usado. Define a visibilidade como restrita ao pacote.

PRIVADO

Declarado pelo uso do modificador *"private"*. A visibilidade é restrita à classe.

Herança - Visibilidade

Primeiramente, precisamos compreender como os modificadores de acesso operam. Já vimos que tais modificadores alteram a acessibilidade de classes, métodos e atributos. Há quatro níveis de acesso em Java:

PROTEGIDO

Declarado pelo uso do modificador *"protected"*. A visibilidade é restrita ao pacote e a todas as subclasses.

PÚBLICO

Declarado pelo uso do modificador *"public"*. Não há restrição de visibilidade.

Herança - Visibilidade

Os modificadores de acessibilidade ou visibilidade operam controlando o escopo no qual se deseja que os elementos (classe, atributo ou método) aos quais estão atrelados sejam visíveis.

O maior escopo é o **global**, que, como o próprio nome indica, abarca todo o programa. Outro escopo é definido pelo pacote.

Saiba mais

O modificador *"private"* é o mais restrito, pois limita a visibilidade ao escopo da classe. Isso quer dizer que um atributo ou método definido como privado não pode ser acessado por qualquer outra classe senão aquela na qual foi declarado. Isso é válido mesmo para classes definidas no mesmo arquivo e para as subclasses.

Herança - Pacote

Um pacote define um espaço de nomes e é usado para agrupar classes relacionadas.

Esse conceito contribui para a melhoria da organização do código de duas formas: permite organizar as classes pelas suas afinidades conceituais e previne conflito de nomes.

Comentário

Evitar conflitos de nomes parece fácil quando se trabalha com algumas dezenas de entidades, mas esse trabalho se torna desafiador num software que envolva diversos desenvolvedores e centenas de entidades e funções.

Em Java, um pacote é definido pela instrução *"package"* seguida do nome do pacote inserida no arquivo de código-fonte. Todos os arquivos que contiverem essa instrução terão suas classes agrupadas no pacote. Isso significa que todas essas classes, isto é, classes do mesmo pacote, terão acesso aos elementos que tiverem o modificador de acessibilidade *"default"*.

Herança - Pacote

O acesso aos métodos e atributos da superclasse pode ser concedido pelo uso do modificador *"protected"*. Esse modificador restringe o acesso a todas as classes do mesmo pacote. Classes de outros pacotes têm acesso apenas mediante herança.

Finalmente, o modificador de acesso *"public"* é o menos restritivo. Ele fornece acesso com escopo global. Isto é, qualquer classe do ambiente de desenvolvimento pode acessar as entidades declaradas como públicas.

Herança - Pacote

A tabela a seguir sumariza a relação entre os níveis de acesso e o escopo.

	default	public	private	protected
Subclasse do mesmo pacote	sim	sim	não	sim
Subclasse de pacote diferente	não	sim	não	sim
Classe (não derivada) do mesmo pacote	sim	sim	não	sim
Classe (não derivada) de pacote diferente	não	sim	não	não

Herança - Pacote

As restrições impostas pelos modificadores de acessibilidade são afetadas pela herança da seguinte maneira:

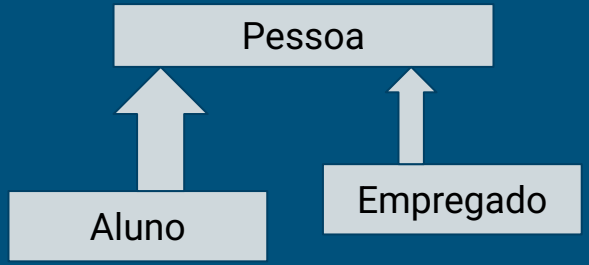
1) Métodos (e atributos) declarados públicos na superclasse devem ser públicos nas subclasses.
2) Métodos (e atributos) declarados protegidos na superclasse devem ser protegidos ou públicos nas subclasses. Eles não podem ser privados.

Atenção!

Métodos e atributos privados não são accessíveis às subclasses, e sua acessibilidade não é afetada pela herança.

Herança - Exemplo

Como exercício vamos implementar as classes, conforme diagrama ao lado.



Herança - Exemplo - Main

Main.java × Pessoa.java Aluno.java Empregado.java

```java
import java.util.Date;
public class Main {
    public static void main(String[] args) {
        // Criando uma instância de Aluno
        Aluno aluno = new Aluno( nome: "João", cpf: "123.456.789-00", new Date(), matricula: 1001);
        aluno.setNome("João da Silva"); // Alterando nome do aluno
        System.out.println("Aluno: " + aluno.getNome() + ", Matrícula: " + aluno.getMatricula());

        // Criando uma instância de Empregado
        Empregado empregado = new Empregado( nome: "Maria", cpf: "987.654.321-00", new Date(), salario: 2500.00);
        System.out.println("Empregado: " + empregado.getNome() + ", Salário: " + empregado.getSalario());
    }
}
```

Herança - Exemplo - Pessoa

Main.java | Pessoa.java | Aluno.java | Empregado.java

```java
import java.util.Date;

// Classe Pessoa
2 usages  2 inheritors
class Pessoa {
    3 usages
    private String nome;
    3 usages
    private String cpf;
    3 usages
    private Date dataDeNascimento;

    // Construtor
    2 usages
    public Pessoa(String nome, String cpf, Date dataDeNascimento) {
        this.nome = nome;
        this.cpf = cpf;
        this.dataDeNascimento = dataDeNascimento;
    }
```

Herança - Exemplo - Pessoa

```
15
16      // Getters e Setters
        2 usages
17      public String getNome() {
18          return nome;
19      }
20
        1 usage
21      public void setNome(String nome) {
22          this.nome = nome;
23      }
24
        no usages
25      public String getCpf() {
26          return cpf;
27      }
28
```

Herança - Exemplo - Pessoa

```
29      public void setCpf(String cpf) {
30          this.cpf = cpf;
31      }
32
        no usages
33      public Date getDataDeNascimento() {
34          return dataDeNascimento;
35      }
36
        no usages
37      public void setDataDeNascimento(Date dataDeNascimento) {
38          this.dataDeNascimento = dataDeNascimento;
39      }
40  }
41
```

Herança - Exemplo - Aluno

Main.java | Pessoa.java | Aluno.java | Empregado.java

```java
import java.util.Date;
// Classe Aluno
2 usages
class Aluno extends Pessoa {
    3 usages
    private int matricula;

    // Construtor
    1 usage
    public Aluno(String nome, String cpf, Date dataDeNascimento, int matricula) {
        super(nome, cpf, dataDeNascimento);
        this.matricula = matricula;
    }

    // Getter e Setter específico para matrícula
    1 usage
    public int getMatricula() { return matricula; }
```

Herança - Exemplo - Aluno

```
14          return matricula;
15      }
16
        no usages
17  >   public void setMatricula(int matricula) { this.matricula = matricula; }
20  }
21
22
```

Herança - Exemplo - Empregado

Main.java | Pessoa.java | Aluno.java | Empregado.java

```java
import java.util.Date;
// Classe Empregado
2 usages
class Empregado extends Pessoa {
    3 usages
    private double salario;

    // Construtor
    1 usage
    public Empregado(String nome, String cpf, Date dataDeNascimento, double salario) {
        super(nome, cpf, dataDeNascimento);
        this.salario = salario;
    }

    // Getter e Setter específico para salário
    1 usage
    public double getSalario() {
        return salario;
    }
```