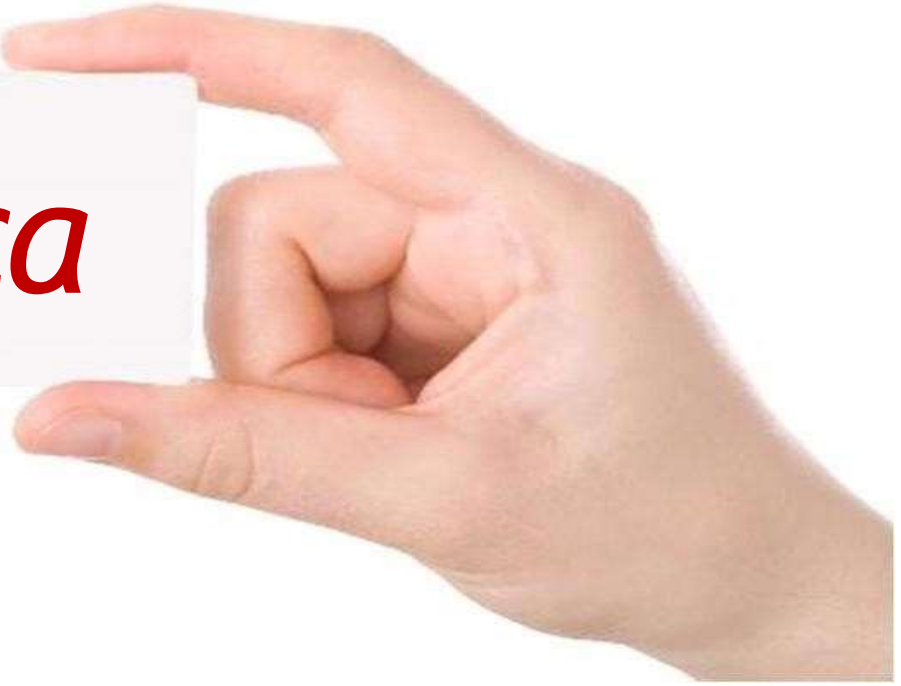
Herança



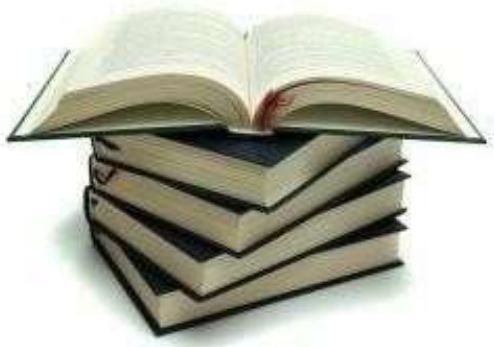
Programação Aplicada

Prof. Me. Gustavo Molina

HERANÇA SEGUNDO O DICIONÁRIO

• Aquilo que se recebeu dos pais das gerações anteriores;

•Bem, direito ou obrigação transmitido por via de sucessão ou por testamento.



Herança em Programação Orientada a Objeto

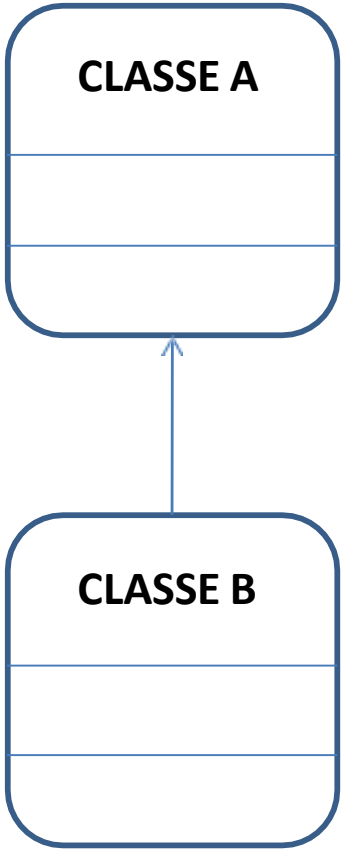
•Refere-se ao fato de que uma classe pode “herdar” toda ou parte da estrutura e comportamento de outra classe.

•Suponha que a classe B é herdeira da estrutura/ comportamento da classe A, então:

➔A classe B é chamada de: subclasse de A ou classe filha;

➔A classe A é chamada de superclasse de B ou classe mãe;

Graficamente



Superclasse = classe mãe.

Subclasse = classe filha.

EXEMPLO PRÁTICO

```
class Funcionario {
      String nome;
      String cpf;
      double salario;
      // métodos da classe
}
```

Ao se definir uma classe com estes atributos, estamos afirmando que todo funcionário tem `nome`, `cpf` e `salario`. Mas existem algumas exceções: um gerente é um funcionário da mesma empresa, porém ele tem também o atributo senha de acesso e o premio mensal. Como diferenciar?

EXEMPLOPRÁTICO

```
class Gerente{
        String nome;
        String cpf;
        double salario;
        int senha;
        double premioMensal;

        boolean autentica (int senha) {
                if (this.senha == senha){
                        System.out.println("Acesso permitido");
                        return true;
                else {
                        System.out.println("Acesso negado");
                        return false;
        }
}
```

Será que esta classe é necessária?

Solução

Relacionar a classe `Gerente` à classe `Funcionario`, de forma que ela "herde" todos os atributos que são comuns. Assim, ela passa a ser uma extensão de `Funcionario` e desta forma, ela pode ter seus próprios atributos e métodos.

```
class Gerente extends Funcionario {
        int senha;
        double premioMensal;

        boolean autentica (int senha) {
            if (this.senha == senha){
                System.out.println("Acesso permitido");
                return true;
            else {
                System.out.println("Acesso negado");
                return false;
        }
}
```

As classes Funcionário e Gerente

```
class Funcionario {
        String nome;
        String cpf;
        double salario;

        // métodos da classe
}
```

```
class Gerente extends Funcionario {
        int senha;
        double premioMensal;

        boolean autentica (int senha) {
            if (this.senha == senha){
                System.out.println("Acesso permitido");
                return true;
            else {
                System.out.println("Acesso negado");
                return false;
        }
}
```

Implicações

- Toda vez que se criar um objeto do tipo `Gerente`, este objeto terá também os atributos definidos na classe `Funcionario` ➔ Todo gerente é um funcionário.

- A classe `Gerente` herda todos os atributos e métodos da classe mãe (`Funcionario`). Herda também os atributos e métodos privados, porém, não consegue utilizá-los diretamente.

OUTRO EXEMPLO

Todo veículo tem obrigatoriamente um documento onde consta o número do renavam e o nome do proprietário entre outras coisas (independentemente do veículo ser um carro ou um caminhão)

Como fica a Classe Veículo

```
class Veiculo {
	int renavan;
	String proprietario;

	Veiculo(int renavan, String proprietario) {
		this renavan = renavan;
		this proprietario = proprietario;
	}

	void transfereProprietario(String novoProprietario){
		proprietario = novoProprietario;
	}
}
```

O QUE UM CARRO TEM ALÉM DE renavam e proprietario?

```
class Carro extends Veiculo {
        int qtdePortas;
        . . .
}
```

O QUE UM CAMINHÃO TEM ALÉM DE renavam e proprietario?

```
class Caminhao extends Veiculo {
        int qtdeEixos;
        . . .
}
```

NO MÉTODO MAIN....

```
Carro meuCarro = new Carro();

meuCarro.qtdePortas = 4;
meuCarro.renavan = 1809358684;
meuCarro.proprietario = “Jorge”;
meuCarro.transfereProprietario(“Pedro”);
```

Herança e o Método Construtor

Quando um objeto de uma subclasse é criado e seu construtor é executado, temos que lembrar que primeiramente o construtor da superclasse é executado de forma automática. Só então, é que o método construtor da subclasse é executado. No caso da subclasse Carro:

```
class Carro extends Veiculo {
      int qtdePortas;

      Carro(int qtdePortas){
            super (111245687, "Alfredo");
            this qtdePortas = qtdePortas;
      }
}
```

MODIFICADOR DE ACESSO → PROTECTED

→ As classes filhas têm acesso à todos os atributos e métodos declarados como `public` da classe mãe.

→Os atributos declarados como `private` (encapsulados) na classe mãe, não são acessíveis diretamente na classe filha (uso de setters/getters).

→O modificador `protected` permite que os atributos da classe mãe sejam visíveis à própria classe e às classes herdeiras dela.

MODIFICADORES DE ACESSO

MODIFICADOR	LIMITE DEVISIBILIDADE
`private`	Este é o nível de encapsulamento mais restritivo. A visibilidade das declarações limita-se à classe onde está definido o atributo/método.
`protected`	A visibilidade das declarações limita-se à própria classe e às classes herdeirasdela.
"sem especificação de modificador" (**`package`**)	A visibilidade das declarações limita-se à própria classe e às classes do mesmo package, mas não às classes herdeiras. Classes herdeiras não precisam ser do mesmo package.
`public`	Estas declarações são sempre acessíveis.

Reescrita de métodos (Override)

É possível que uma subclasse tenha um método com o mesmo nome e assinatura de um método da superclasse.

Durante a execução do programa, o tipo do objeto que está sendo manipulado é quem definirá qual método será executado.

EXEMPLO

```
class Funcionario {
    String nome;
    String cpf;
    double salario;

    void calculaSalario (double percentual){
        salario = salario + (salario * percentual/100);
    }
}

class Gerente extends Funcionario {
    int senha;
    double premio;

    void calculaSalario (double percentual) {
        salario = salario + (salario * percentual/100) + premio;
    }
}
```

Como as duas classes têm o mesmo método (override), significa que cada classe pode ter sua particularidade no cálculo do salário.