Herança em Java

1

Herança

- Herança é utilizada para criar uma classe nova a partir de uma classe existente;
- Ela é útil quando a classe existente contém apenas parte das características necessárias para um novo tipo de objeto;
- A nova classe contém todos os atributos e métodos da classe anterior;
- A nova classe pode adicionar atributos e métodos, e redefinir métodos existentes herdados da classe existente.

2

Herança em Java

- A **herança** possibilita o reuso de **atributos** e **métodos** de uma **classe pai** por suas **classes filhas**.
- A **classe filha** (chamada **subclasse**) herda os atributos e métodos da **classe pai** (chamada **superclasse**).
- Em Java, uma subclasse só pode herdar de uma única superclasse, já uma superclasse pode ter ilimitado número de subclasses.
- Java não permite herança múltipla.

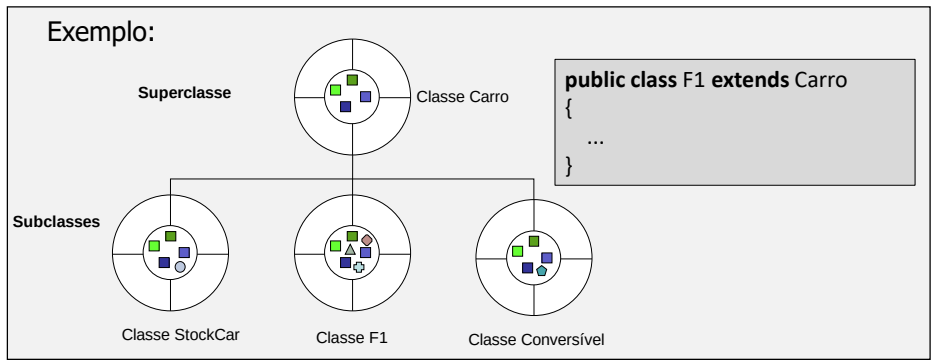
3

3

Sintaxe de Definição de Herança

- A sintaxe de definição de Herança em Java:

public class NomeSubClasse **extends** NomeSuperClasse **{...}**



4

4

Características

- A subclasse não tem acesso aos atributos e métodos da superclasse declarados como *private*;
- Atributos e métodos da superclasse declarados como *protected*, são acessíveis apenas pelas subclasses,
- Já os declarados como *public* são acessíveis por qualquer classe;
- Construtores não são herdados.

5

5

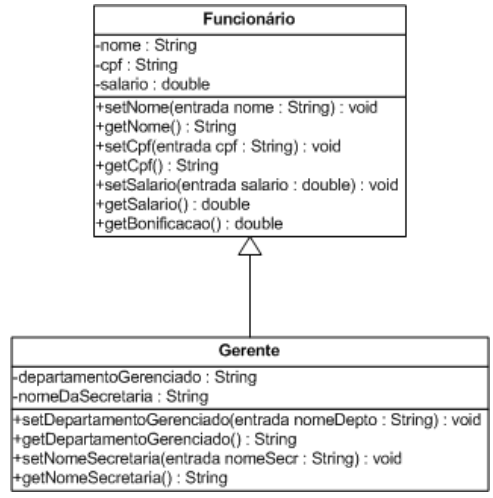
Características

- Uma subclasse pode redefinir um método da superclasse se ele não for *private*;
- Por exemplo, a classe Carro calcula o deslocamento apenas levando em consideração apenas a velocidade. Já a classe Formula1 precisa levar mais atributos em consideração, como a pressão aerodinâmica dos aerofólios. Então, ela deve reimplementar o método de cálculo do deslocamento;
- Pode-se usar a palavra chave *super* para chamar métodos e construtores da superclasse.

6

6

Modelagem de Herança em Java



7

7

Especificação da Classe Funcionário

```java
public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;

    public Funcionario() {
    }

    public Funcionario(String none, String cpf, double salario) {
        this.nome = nome;
        this.cpf = cpf;
        this.salario = salario;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
    ...
}
```

8

8

Herança na Classe Gerente

```
1  public class Gerente extends Funcionario {
2      private String departamentoGerenciado;
3      private String nomeDaSecretaria;
4
5      public Gerente() {
6      }
7
8      public Gerente(String none, String cpf, double salario, String depto, String secr) {
9          super(nome, cpf, salario);
10         departamentoGerenciado = depto;
11         nomeDaSecretaria = secr;
12     }
13
14     public void setDepartamentoGerenciado(String nomeDepto) {
15         departamentoGerenciado = nomeDepto;
16     }
17
18     public String getDepartamentoGerenciado() {
19         return departamentoGerenciado;
20     }
21     ...
22 }
```

Herança da superclasse "Funcionário"

Chamada do construtor da superclasse "Funcionário"

9

9

Utilização de métodos herdados em uma Classe

```
1  public class TestaGerente {
2      public static void main(String [] args) {
3          Gerente gerente = new Gerente();
4          gerente.setNome("João da Silva");
5          gerente.setDepartamentoGerenciado("Financeiro");
6      }
7  }
```

Utilização de um método herdado da classe Funcionário

Utilização de um método da classe Gerente

10

10

Redefinição de métodos (Overriding)

- A herança permite que métodos existentes na superclasse sejam reimplementados na subclasse;
- A partir da redefinição, o método assume o comportamento implementado na subclasse;
- Caso o comportamento do método definido na superclasse seja necessário, ele deve ser invocado explicitamente;
- A superclasse é representada pela palavra reservada **`super`**;

11

11

Exemplo de Redefinição (Overriding)

```java
public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;

    ...

    public double getBonificacao() {
        return salario * 0.10;
    }
}
```

A bonificação de todos os Funcionários é igual a 10% do seu salário

```java
public class Gerente extends Funcionario {
    private String departamentoGerenciado;
    private String nomeDaSecretaria;

    ...

    public double getBonificacao() {
        return super.getBonificacao() + 1000;
    }
}
```

Sobrescrita

Os gerentes têm um acréscimo de R$1000,00 ao bônus de 10%

Invocação do método da classe pai

12

12

Polimorfismo

- Define que instâncias de subclasses de uma mesma superclasse podem invocar métodos com mesma assinatura, mas com comportamentos distintos.
- Isso é feito usando uma referência a um objeto do tipo da superclasse.
- A decisão sobre qual implementação será executada é tomada em tempo de execução, por meio do mecanismo de **ligação tardia.**
- A Ligação tardia significa que o método a ser invocado é definido durante a compilação do programa

13

13

Polimorfismo - Exceção

- Em Java, métodos declarados como **final** não podem ser redefinidos e, portanto, não têm invocação polimórfica;
- Métodos declarados como **private** são implicitamente finais, portanto, não têm invocação polimórfica.

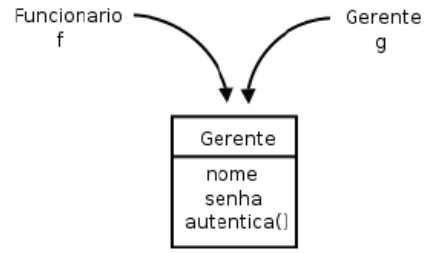
14

14

Polimorfismo - Exemplo

- Através da herança, todo o Gerente é um Funcionário;
- Assim, podemos nos referir a um gerente como um funcionário.

```
Gerente gerente = new Gerente();
Funcionario funcionario = gerente;
funcionario.setSalario(5000.0);
```



15

15

Exemplo de Utilização de Polimorfismo

Classe que controla o total de bonificação

```
class ControleDeBonificacoes {
    private double totalDeBonificacoes = 0;

    public void registra(Funcionario funcionario) {
        this.totalDeBonificacoes += funcionario.getBonificacao();
    }

    public double getTotalDeBonificacoes() {
        return this.totalDeBonificacoes;
    }
}
```

Método que soma a bonificação dos func.

em algum lugar da minha aplicação (ou no main se for apenas para testes):

```
ControleDeBonificacoes controle = new ControleDeBonificacoes();
Gerente funcionario1 = new Gerente();
funcionario1.setSalario(5000.0);
controle.registra(funcionario1);

Funcionario funcionario2 = new Funcionario();
funcionario2.setSalario(1000.0);
controle.registra(funcionario2);

System.out.println(controle.getTotalDeBonificacoes());
```

Criação de um gerente

Criação de um funcionário

Gerente sendo usando como se fosse um funcionário

16

16

Resumo

- Herança serve para implementar novas classes a partir de classes existentes;
- Os métodos herdados podem ser redefinidos na subclasse;
- Para manter o comportamento definido na superclasse em métodos redefinidos é necessário invoca-lo explicitamente;
- Na subclasse a superclasse é representada pela palavra reservada **`super`**;
- O polimorfismo é a invocação de implementações diferentes para o mesmo método redefinido nas subclasses.

17