Programação Orientada a Objetos

Aula 07 - Herança

Apresentação

Hoje, você vai aprender sobre mais um pilar de sustentação da Programação Orientada a Objetos: a **herança**. Ela é considerada um dos conceitos mais importantes na POO. A herança trouxe para a programação um novo olhar sobre a maneira de se programar, até então não atendido pela Programação Estruturada.

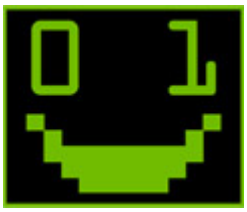
Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Entender o que é herança.
- Conhecer quais os tipos de herança existentes em POO.
- Entender o funcionamento da herança durante a execução do programa.
- Programar em Java usando a herança.

Herança



Calma, você não recebeu uma herança em bens dos seus parentes!!! Mas, a Herança na POO tem uma certa ligação com grau de parentesco. Vamos descobrir que os objetos podem ter Mãe!

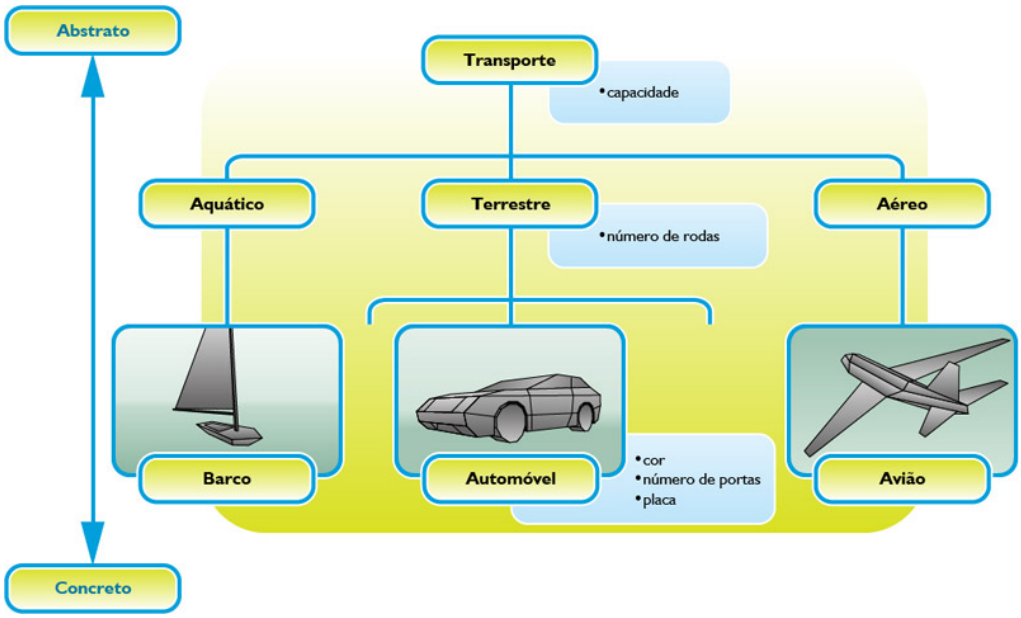
Eu sou um objeto?!?! Eu quero a minha Mãe!!!

Você já ouviu falar sobre herança relacionada à programação? Estranho não é? A Figura 1 mostra os pilares da orientação objetos com destaque para a herança.

Figura 01 - Orientação a objetos

Vamos começar pela definição de herança. Herança é o mecanismo que permite a uma classe herdar todos os atributos e métodos de outra classe. Ela permite definir a implementação de uma nova classe na definição de uma classe previamente implementada. Como exemplo, observe a Figura 2 a seguir.

Figura 02 - Exemplo de herança entre classes



Considere como classes as seguintes abstrações: **Transporte, Aquático, Terrestre, Aéreo, Barco, Automóvel** e **Avião**. Considere **capacidade** como um atributo da classe Transporte, que indica a quantidade de pessoas que o transporte em questão pode transportar. E **número de rodas** como um atributo da classe **Terrestre**. E **cor, número de portas** e **placa** como atributos da classe Automóvel.

A **Figura 2** apresenta um exemplo de herança, onde as classes **Aquático**, **Terrestre** e **Aéreo** herdam da classe Transporte.

A classe **Barco** herda da classe **Aquático**. A classe **Automóvel** herda de **Terrestre**. E, finalmente, **Avião** e **Aéreo**.

Um aspecto importante que também podemos observar na figura é que toda classe que herda de uma outra acaba herdando também, como consequência, os seus atributos. Por exemplo, de acordo com a **Figura 2**, a classe **Transporte** possui um atributo chamado **capacidade**. Para a classe **Aquático**, como ela herda de **Transporte**, pode-se dizer que também possui o atributo **capacidade**. E Barco, como herda de **Aquático**, também possui o atributo **capacidade**. Com essa lógica, percebemos que na herança os atributos (e também os métodos) são herdados naturalmente pelas classes subsequentes na hierarquia.

Hierarquia de Classe ou de Herança: é o mapeamento do tipo árvore de relacionamentos que se formam entre as classes como resultado da herança. Exemplo: a Figura 2 representa uma hierarquia de herança ou hierarquia de classe.

Usando a mesma lógica, responda: Quantos e quais são os atributos da classe **Terrestre**? E da classe **Automóvel**?

Respondendo: a classe **Terrestre** possui dois atributos: **capacidade** (que é herdado de Transporte) e **número de rodas**. Já a classe **Automóvel** possui cinco atributos: **capacidade** (herdado de Transporte), **número de rodas** (herdado de Terrestre), cor, **número de portas** e **placa**.

Observe na **Figura 2** que, quanto mais alta na hierarquia está a classe, mais ela tende a ser **abstrata** em comparação com as suas subsequentes. Ou seja, quanto mais alta na hierarquia, menos definida (abstrata) é a classe, e assim ela define menos atributos e métodos. Isso também garante que a classe tenha mais chances de ser reusada por outras classes que herdem da mesma. Entendeu? Não, então, vamos ao exemplo da figura. Suponha que a classe **Transporte**, além do atributo **capacidade**, possuísse também o atributo **número de rodas**. Ou seja, ela passa a ser uma classe mais **concreta**, menos abstrata. Mas, nesse caso, consequentemente, não seria interessante nem faria sentido para as classes **Aquático** e **Aéreo** herdar os atributos de **Transporte** (ou herdar de Transporte),pois o atributo **número de rodas** não é desejado por tais classes.

Vídeo 02 - Conceitos de Herança

Atividade 01

1. Defina alguns atributos adicionais para as classes **Aéreo** e **Avião** da Figura 2. Em seguida, baseado na hierarquia de herança da qual elas fazem parte, indique quantos e quais são os atributos de cada uma delas.

Termos Usados para Herança

Na literatura, são encontrados diversos termos para nomear tanto as classes que fornecem a herança quanto as classes que herdam de alguma outra. Veja exemplos de termos usados no Quadro 1 a seguir.

Classes que Fornecem a Herança	Classes que Herdam de Outras
Superclasse	Subclasse
Mãe	Filha
Tipo	Subtipo

Quadro 1 - Termos usados na herança

Nota: a classe filha não pode remover os atributos e métodos da classe mãe. Abaixo, são apresentados outros termos que são também comumente usados.

- **Ancestral:** é uma classe que aparece na hierarquia de classes em uma posição acima da progenitora (mãe).
- **Descendente:** dada uma classe, toda classe que aparece abaixo dela na hierarquia é uma descendente da classe dada.
- **Raiz (ou Base):** é a classe topo da hierarquia.
- **Folha:** é uma classe sem filhas.

Atividade 02

1. Considerando a hierarquia de herança entre classes da L, dê exemplos de:
    a. Classes superclasse e subclasse.
    b. Classe raiz e classe folha.
    c. Classe ancestral da classe **Automóvel**, que não seja superclasse dela.
    d. Classe descendente da classe **Veículo**, que não seja subclasse dela.

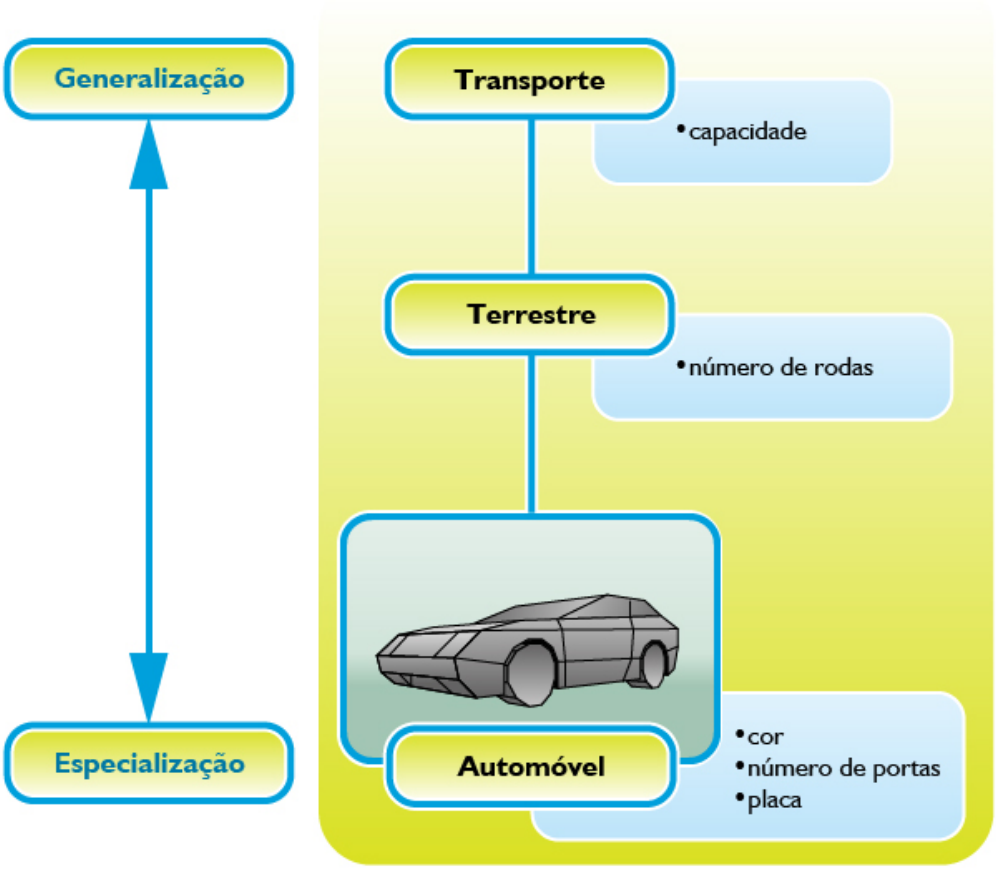
Especialização e Generalização

Assim como os conceitos de classes **Abstratas** e **Concretas**, tem-se também nos extremos da árvore hierárquica de herança, os conceitos de **Generalização** e **Especialização**. Na **Generalização**, como o próprio nome sugere, há classes mais genéricas e abstratas disponíveis, as quais podem ser usadas para outras

descenderem delas. Já a **Especialização** é usada para indicar que classes que estão numa posição inferior na hierarquia possuem estado e comportamento mais especializados, ou seja, com mais detalhes de informações.

A **Figura 3** ilustra tais conceitos dentro da hierarquia de classes de transportes, apresentada anteriormente. Como pode ser observado, classes em posição inferior na hierarquia, tal como a classe Automóvel, representam **especializações** de classes em posição superior (Transporte, Terrestre) na hierarquia. Já classes em posição superior, como a classe Transporte, representam **generalizações** de classes em posições inferiores (Automóvel, Terrestre).

Figura 03 - Generalização e Especialização



Herança Múltipla e Simples

Herança Múltipla: é a capacidade de uma classe possuir mais de uma superclasse e herdar as variáveis e métodos combinados de todas as superclasses.

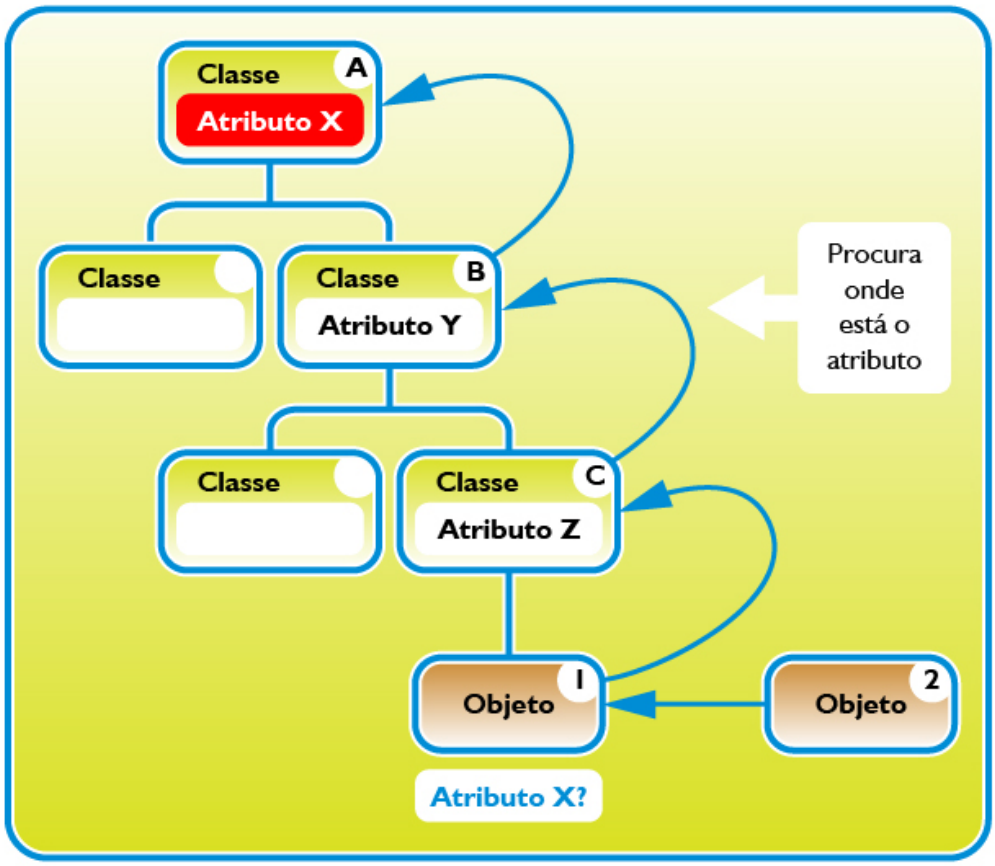
Herança Simples: Cada classe pode ter apenas uma superclasse, embora uma superclasse possa ter várias subclasses.

Na linguagem Java, a herança é simples e na codificação usa-se a palavra reservada *extends* para declarar que uma classe é herdeira de outra. Para simular a herança múltipla em Java, usa-se *Interfaces*.

Funcionamento da Herança

Você viu os conceitos sobre herança, mas é importante também que possamos entender o que acontece com uma classe que utiliza a herança durante sua execução. Para melhor entendermos o funcionamento da herança durante a execução do programa, vejamos a **Figura 4**.

Figura 04 - Herança em ação

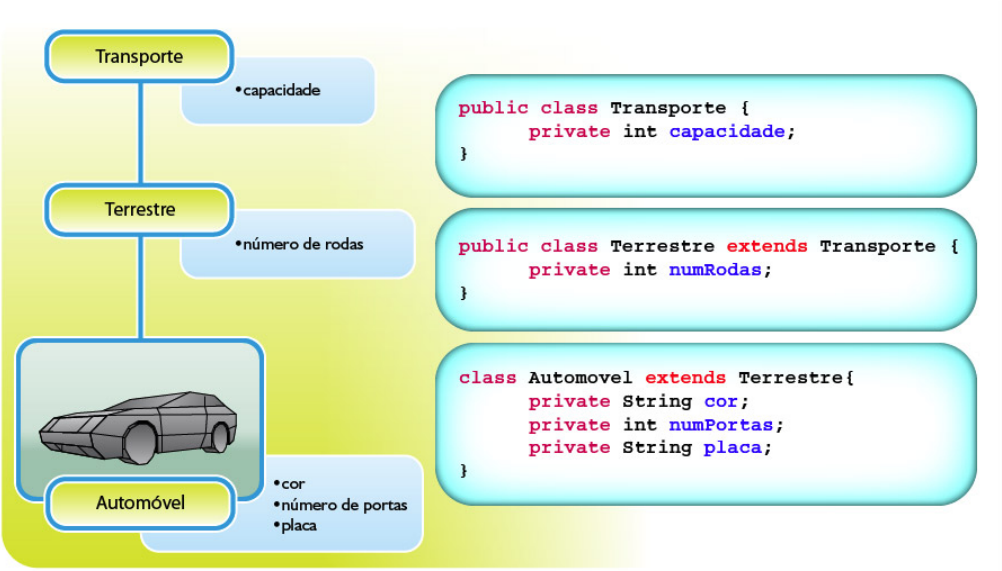


Sabemos que quando criamos um objeto, temos uma instância concreta da classe a qual esse objeto representa. Na **Figura 3**, esse objeto é representado pelo **Objeto1**. Agora, vamos supor que o **Objeto2** quer saber o valor do **atributoX** desse objeto, através do envio de uma mensagem. Para obter o valor do atributo X, o **Objeto1** sai consultando sua árvore hierárquica de classes até encontrar o atributo solicitado pelo **Objeto2**. No caso da **Figura 4**, o objeto consulta a classe da qual ele foi instanciado (**ClasseC**), não encontrando, consulta a classe mãe dessa mesma (**ClasseB**), e assim sucessivamente até encontrar o atributo e o valor. Vale ressaltar que o **Objeto1** não percorre toda a árvore hierárquica, percorre o caminho da classe filha em direção à classe mãe.

Outro ponto importante é que o objeto não acumula todos os atributos das classes que fazem parte da hierarquia de herança, mas apenas os atributos das classes ancestrais a ele. O usuário (programador ou outros objetos, tal como o Objeto 2 na **Figura 4**) de um determinado objeto não sabe (nem precisa saber) se o atributo é dele ou se o atributo é herdado de uma outra classe ancestral dele.

Herança em Java

Depois de termos visto os conceitos sobre herança, vamos finalmente ver a herança nas linhas de código Java. Como mencionamos anteriormente, em Java a palavra-chave extends é usada para indicar que uma dada classe herda de outra. A Listagem 1 apresenta exemplos de código em Java, indicando que **Terrestre** herda de **Transporte** e que **Automóvel** herda de **Terrestre**.



Listagem 1 - Exemplo de Herança em Java

Vale lembrar que quando uma classe herda de outra, todos os atributos (e métodos) da outra classe passam a fazer parte dessa mesma. Isso significa que, para o exemplo acima, o programador (ou outro objeto) pode solicitar à classe Automóvel os atributos de **Transporte** e **Terrestre**. Os atributos de **Terrestre** são acessíveis na classe **Automóvel** porque a segunda herda da primeira. E os atributos de **Transporte** são acessíveis a **Automóvel** devido ao fato de **Terrestre** herdar de **Transporte**.

Uma dúvida que pode estar surgindo é: como os atributos são acessíveis se eles estão declarados como private?

Respondendo: considere que para os códigos das classes da Listagem 1 foram declarados os métodos **get** e **set** para cada um dos atributos. Através desses métodos, que são também herdados pelas subclasses, podemos acessar facilmente qualquer um dos atributos herdados. Eles só não foram escritos para ressaltarmos a construção extends de Java e definir a herança entre classes. Veremos códigos mais completos na próxima aula.

Vídeo 03 - Representação UML

Vídeo 04 - Codificando Herança

Anote as Dicas!

1. A classe **Object**: todas as classes em Java descendem de uma classe, chamada Object, mesmo que a declaração **extendsObject** seja omitida, a classe Object é considerada a classe raiz da hierarquia de todas as classes Java, sendo, portanto, ancestral de todas as classes da linguagem.

2. Quando uma classe usa a relação de herança, podemos dizer que essa classe possui um relacionamento chamado **"É um"** com a classe da qual ela herda. Tal relação também indica que uma classe é do mesmo tipo que outra. Assim, nos exemplos anteriores, podemos dizer que **Automóvel** "é um" transporte **Terrestre**, assim como que **Terrestre** "é um" (ou tipo de) **Transporte**.

Leitura Complementar

Não faz parte desta disciplina falar sobre interfaces. Também não é obrigatório que você já domine a simulação de herança múltipla usando interface em Java. Para isso, sugerimos que você primeiro domine a herança simples para no futuro procurar saber sobre o assunto.

Como leitura complementar, propomos o próprio tutorial Java da Sun, que pode ser acessado pelo seguinte endereço:

<http://java.sun.com/docs/books/tutorial/index.html>

Artigo da web que fala sobre herança:

<http://www.tiexpert.net/programacao/java/heranca.php>

Resumo

Nesta aula, você aprendeu que **herança** é a capacidade que uma classe tem de herdar as características (atributos) e comportamentos (métodos) de outra classe. Conheceu como funciona a herança durante a execução do programa e quais são os tipos em que ela é classificada. Você viu também pequenos exemplos de como aplicar a herança na prática. Na próxima aula, continuaremos abordando esse assunto, apresentando mais exemplos interessantes do uso de herança, através da sobreposição de métodos construtores e os modificadores na herança.

Autoavaliação

1. Sem consultar o material responda: o que você entendeu por **herança**?
2. Qual a diferença entre Herança simples e Herança múltipla?

3. Como funcionaria o processo de busca pelos atributos na herança, se fosse solicitado ao objeto **Automóvel** a sua capacidade? (Dica: releia a explicação da **Figura 3** da nossa aula).

4. Considerando o código das classes **Transporte, Terrestre e Automóvel** apresentados na **Figura 4**, finalize a implementação delas, adicionando os métodos **get() e set()** para cada um de seus atributos. Em seguida, crie uma classe **Principal** com um método **main()** que cria um objeto da classe **Automóvel**, e chama todos os métodos set() e get() criados, inclusive das classes **Transporte e Terrestre**. Observe no seu exemplo, que é possível chamar todos os métodos get() e set() herdados pela classe Automóvel.

5. Das opções abaixo, qual se refere ao conceito de herança?

    a. Herança é a capacidade de reaproveitar outras classes para compor uma nova classe.

    b. Herança é a característica da OO de ocultar partes da implementação interna de classes do mundo exterior.

    c. Herança é a habilidade de objetos de classes diferentes responderem à mesma mensagem de diferentes maneiras.

    d. Herança é o mecanismo que permite a uma classe herdar todos os atributos e métodos de outra classe.

    e. As letras a e d estão corretas.

Referências

BARNES, David J.; KÖLLING, Michael. **Programação orientada a objetos com Java**. São Paulo: Pearson Prentice Hall, 2004.

DEITEL, H. M.; DEITEL, P. J. **Java como programar**. Porto Alegre: Bookman, 2003.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Editora Campus, 2003.

SINTES, anthony. **Aprenda a programar orientada a objeto em 21 dias**. Tradução: João Eduardo Nóbrega Tortello. São Paulo: Pearson EducationdoBrasil, 2002.

THE JAVA tutorials.**What is inheritance?** Disponível em: http://java.sun.com/docs/books/tutorial/java/concepts/inheritance.html>. Acesso em: 15 maio 2010.

_____. **Inheritance**. Disponível em: <http://java.sun.com/docs/books/tutorial/java/IandI/subclasses.html>. Acesso em: 15 maio 2010.