Programação Orientada a Objetos

Aula 08 - Herança II

Apresentação

Na aula passada, foram dados os primeiros passos sobre o assunto herança, um dos pilares da POO. Considerando a importância desse tema, continuaremos a abordá-lo na aula de hoje, focalizando, sobretudo, a herança de comportamento (método). Serão também apresentados outros exemplos mais sofisticados de implementação de herança na linguagem Java.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Relacionar a herança e o modificador de acesso protected.
- Entender o uso da palavra-chave super no contexto de herança entre classes.
- Saber como funciona a herança para os métodos, durante a execução do programa.

A Herança e o Protected

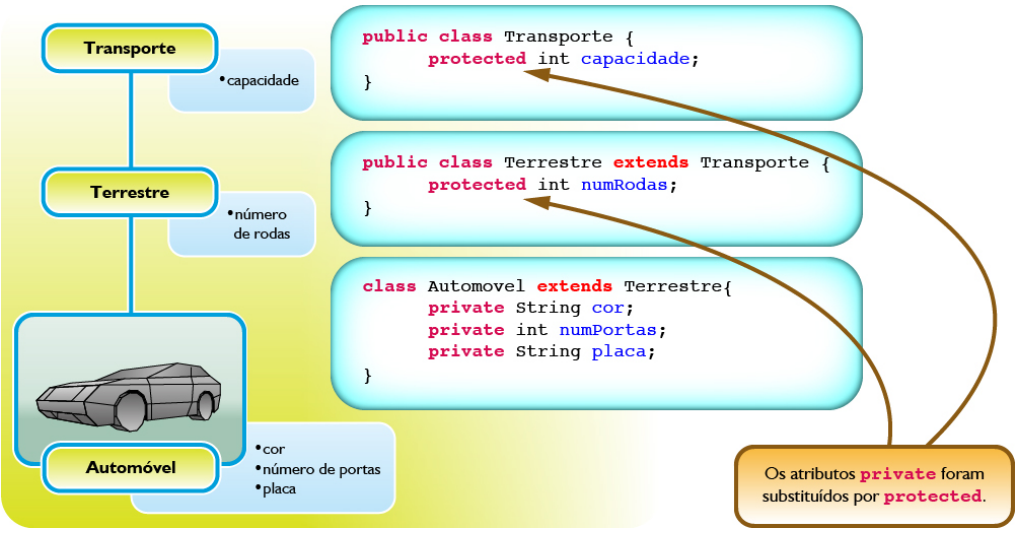
Olá, aqui estamos novamente. Descobrimos na aula passada que objetos (como eu) também têm ancestrais e descendentes!!! Mas alguns pontos importantes ainda precisam ser abordados sobre herança. Por isso, hoje iremos continuar falando sobre esse assunto.

Na Aula 05 (Encapsulamento), você viu os tipos de modificadores de acesso: *public, private e protected*. Esse último, ficamos de explicar melhor durante as aulas sobre herança.

Naquela aula, nós mencionamos que o modificador *protected* funciona como o *private*, exceto que as classes filhas também terão acesso ao atributo ou método declarado como *protected*. Isso significa que apenas as classes descendentes de uma determinada classe poderão ter acesso aos atributos e métodos declarados com esse modificador.

Veja o exemplo abaixo:

Figura 01 - Esquema de herança. **Listagem 1** - Herança em Java proteced e private

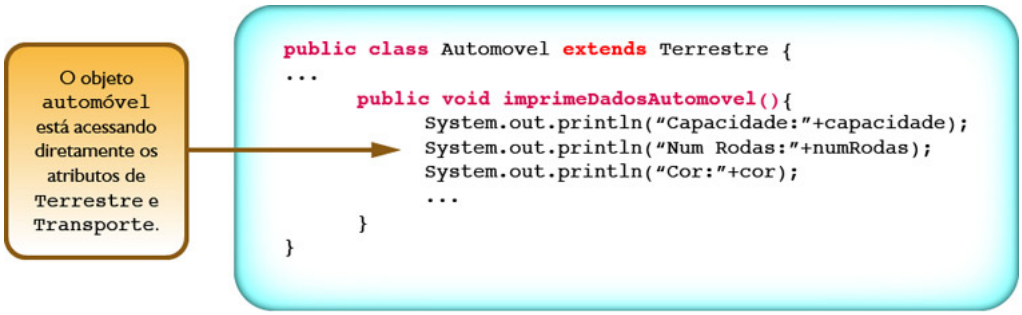


A Figura 1 e a Listagem 1 apresentam o exemplo da aula anterior com algumas pequenas modificações.

As classes **Transporte** e **Terrestre** tiveram seus atributos modificados para serem **protected**, isso significa que apenas classes que pertencem à hierarquia de herança podem acessar diretamente esses atributos. Em outras palavras, apenas subclasses (ou classes descendentes) de Transporte e Terrestre terão acesso aos seus atributos **protected**.

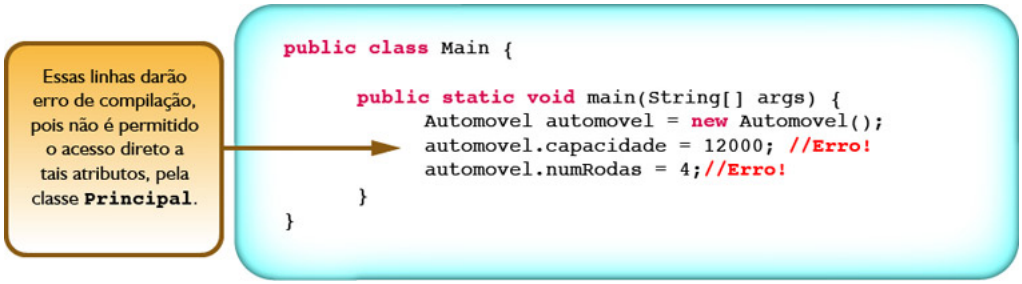
Vídeo 02 - Modificador de Acesso "Protected"

A Listagem 2 abaixo mostra a definição de mais um método na classe **Automóvel**, o qual faz acesso aos atributos **protected, capacidade** e **numRodas**, definidos nas classes ancestrais **Transporte** e **Terrestre**, respectivamente. Observe que tais atributos são acessados e usados livremente no método **imprimeDadosAutomovel()**. Isso só é possível porque agora eles foram declarados como **protected**.



Listagem 2 - Acesso a atributos protected

A Listagem 3 mostra uma tentativa frustrada de acesso aos atributos capacidade e numRodas herdados pela classe Automóvel, dentro de um método main(). Nesse caso, não é possível acessar tais atributos, porque a classe **Principal** não herda da classe Terrestre e, portanto, não tem direito a acessar os atributos protected.



Listagem 3 - Acesso a atributos protected

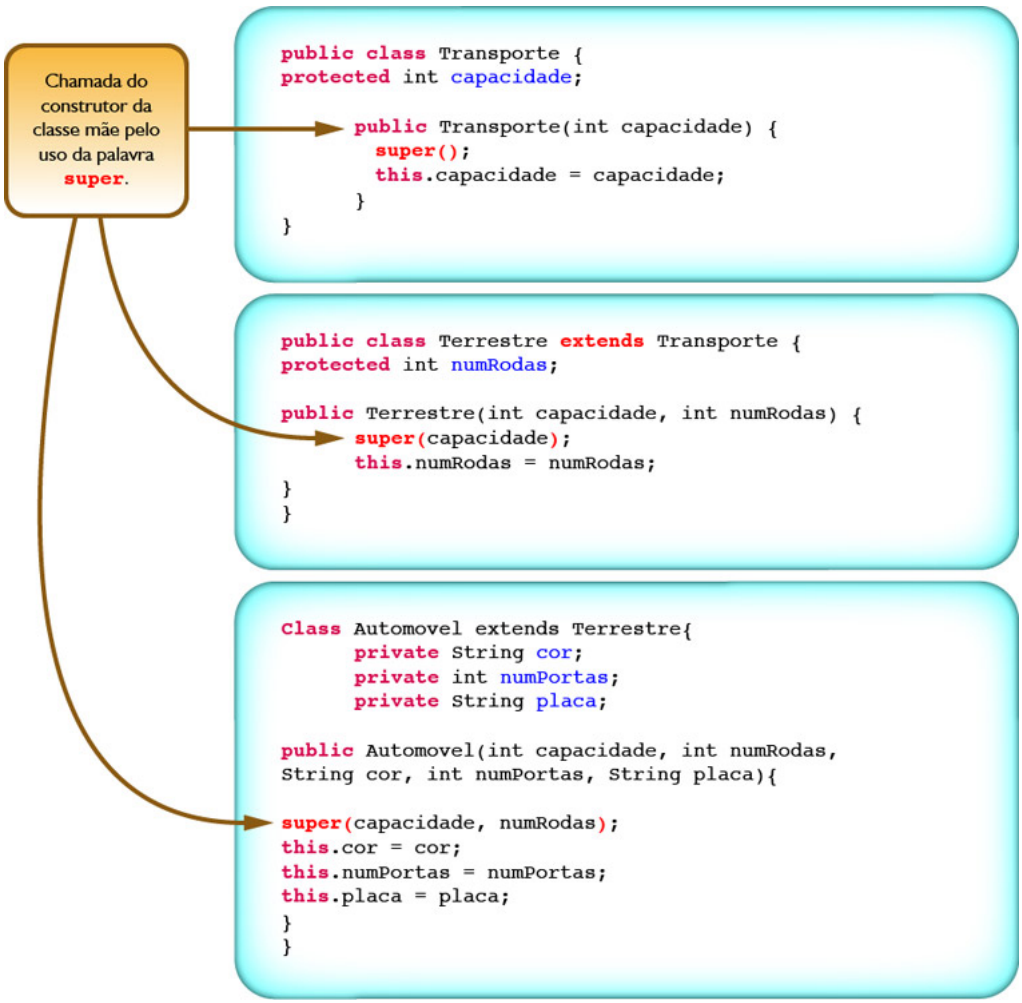
Atividade 01

1. Modifique as classes **Transporte**, **Terrestre** e **Automóvel** da Listagem 1 para que tenham todos os seus atributos declarados como **public**.
2. Em seguida, tente novamente compilar e executar tais classes, juntamente com a classe **Principal** da Listagem 3.
3. O que acontece?
4. A compilação e execução funcionaram corretamente? Se sim, explique o que você acha que aconteceu.

Herança e o Super

Em suas pesquisas em outras fontes (tutoriais na internet, livros), além do material desta aula, você pode ter encontrado o uso da palavra (ou operador) **super** nos códigos exemplos.

A palavra **super** é mais uma das palavras reservadas da linguagem Java que tem uma forte ligação com a herança. A palavra **super** refere-se à classe ancestral imediata da classe, ou seja, à classe mãe ou super-classe. Ela é usada nos construtores para chamada de construtores em cascata das classes mães.



Listagem 4 - Herança em Java com uso do operador super

A Listagem 4 ilustra um exemplo de herança em Java, que faz uso da palavra-chave *super*. Vamos então entender o que acontece quando usamos o comando **super** em tais contextos.

Primeiro, observe que o **super** é usado para chamar o método construtor da classe mãe. Na classe **Transporte**, como a classe mãe de **Transporte** é **Object** (vimos na aula passada que Object é o ancestral de todas as classes), o **super** não tem parâmetro. A chamada de super equivale a uma chamada explícita ao construtor sem parâmetro da classe Object.

Já no construtor da classe **Terrestre**, o **super** tem como parâmetro a **capacidade** exigida no construtor da classe **Transporte**. Portanto, a chamada a super na classe **Terrestre** é no fundo a chamada ao construtor de **Transporte**. Já no construtor da classe **Automóvel**, o **super** tem como parâmetros a **capacidade** e o **numRodas** exigidos pelo construtor da classe **Terrestre**.

Observando a Listagem 4, dá para observar que uma chamada a **super** ocasiona a invocação do construtor da classe mãe. Isso acaba permitindo uma chamada em cadeia dos construtores de classes ancestrais, permitindo assim a configuração de todos os atributos herdados da classe.

Vale ressaltar que se o construtor não possui parâmetro, o compilador Java aceita a omissão do **super** sem parâmetro, porque durante o processo de compilação ele insere explicitamente tal chamada. Para o nosso exemplo, se quiséssemos, poderíamos omitir o **super** do construtor da classe **Transporte**.

Vídeo 03 - Super

Anote as Dicas

1. Apenas comentários são permitidos antes da palavra **super** nos construtores. Assim, não é possível incluir nenhum comando antes de super() no código de métodos construtores de classes.

2. Da mesma maneira que se usa a palavra-chave **this** para acessar os atributos (ou métodos) da própria classe, pode-se usar **super** para acessar os atributos (ou métodos) da classe mãe.

Atividade 02

1. Hoje, os celulares estão cada vez mais sofisticados, mp3, mp4, acesso web, câmera digital... Crie uma hierarquia de classes que possui no topo da hierarquia o celular mais básico, aquele que simplesmente liga e atende ligações, e que vai sendo refinada com várias outras classes que definem celulares mais sofisticados e modernos.

   Para isso, use a hierarquia e nos construtores das classes use o operador *super*. Na hierarquia, defina no mínimo três classes.

Um Exemplo Completo

Vimos até agora diversos exemplos do uso de herança para acessar atributos das classes ancestrais. Vamos mostrar agora a herança utilizando os métodos dessas classes.

```
1  public class Transporte{
2        protected int capacidade
3
4        public Transporte(int capacidade){
5           super();
6           this.capacidade = capacidade;
7        }
8        public int getCapacidade(){
9           return capacidade;
10       }
11       public void setCapacidade(int capacidade){
12          this.capacidade = capacidade;
13       }
14 }
```

Listagem 5 - Classe Transporte

```
1  public class Terrestre extends Transporte{
2     protected int numRodas;
3     public Terrrestre(int capacidade, int num Rodas){
4        super(capacidade);
5        this.numRodas = numRodas;
6     }
7     public int getNumRodas(){
8        return numRodas;
9     }
10    public void setNumRodas(int numRodas){
11       this.numRodas = numRodas;
12    }
13 }
```

Listagem 6 - Classe Terrestre

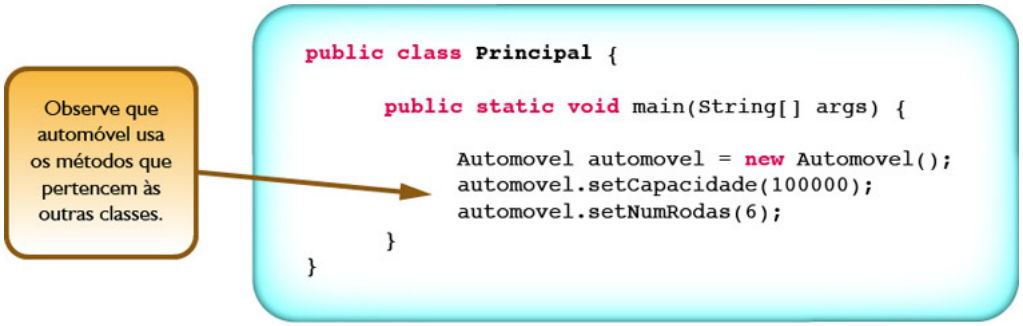
```
public class Automovel extends Terrestre{
   private String cor;
   private int numPortas;
   private String placa;

   public Automovel(){
      super(5,4);
   }

   public Automovel(int capacidade, int numRodas, String cor, int numPortas, String placa){
      super(capacidade,numRodas);
      this.cor= cor;
      this.numPortas = numPortas;
      this.placa= placa;
   }

   public String getCor(){
      return cor;
   }

   public void setCor(String cor){
      public int getNumPortas;
   }

   public void setNumPortas(int numPortas){
      this.numPortas = numPortas;
   }

   public void setPlaca(String placa){
      this.placa = placa;
   }
   public String getPlaca(){
      return placa;
   }
}
```

Listagem 7 - Classe Automóvel

As Listagens 5, 6 e 7 apresentam o código das classes Veículo, Terrestre e Automóvel, incluindo a implementação dos métodos **get** e **set** para cada atributo das classes.

Cada um dos métodos declarados na classe Veículo são herdados pelas classes Terrestre e Automóvel, assim como os métodos declarados pela classe Terrestre são herdados pela classe Automóvel, exatamente como ocorre com os atributos.

A Listagem 8 apresenta um método main() que cria um objeto da classe Automóvel, em seguida, chama dois métodos setCapacidade() e setNumRodas(). Observe pelas Listagens 5 , 6 e 7 que tais métodos não pertencem à classe Automóvel, mas são na verdade herdados de suas ancestrais. Dessa forma, percebe-se claramente que uma vez declarado um método (público ou protected) em uma das classes ancestrais, aquele método é visível em objetos das classes filhas.



Listagem 8 - Classe Principal

Uma pergunta: será que a herança também permite o uso de outros métodos diferentes dos **get** e **para as classes descendentes?**

A resposta é sim, a herança serve para todos os métodos, desde que a classe mãe permita. Se um método estiver com modificador **private**, esse método não estará acessível para classes filhas. Assim, apenas estão acessíveis para classes filhas, os métodos declarados como **public** e **protected** na classe mãe.

Vejamos um exemplo do uso de outros métodos. Vamos acrescentar agora à classe **Terrestre** um método que calcula o número de pneus reservas.

```
public class Terrestre extends Transporte {
      protected int numRodas;

        ... // Considere todos os códigos anteriores

      public int calculaNumPneusReserva() {
        return numRodas - 4;
      }
}
```

Listagem 9 - Classe Terrestre com método calculaNumPneusReserva()

Na Figura 10, vemos a classe Principal modificada para criar um objeto Automóvel e acessar diretamente o método calculaNumPneusReserva() que foi herdado da classe Transporte. Como era de se esperar, mesmo o método sendo da classe **Terrestre** a classe **Automóvel** usa sem restrição.

```
public class Main {
      public static void main(String[] args) {
            Automovel automovel = new Automovel();
            int n = automovel.calculaNumSteps();
            System.out.println("num steps:"+ n);
      }
}
```

Listagem 10 - Método main modificado

Atividade 03

1. Continuando a Atividade 02, acrescente agora os métodos que darão as funcionalidades dos telefones móveis (celulares).

   Uma observação é que os métodos ligar() e atender() não precisam ser criados novamente nas classes filhas. Uma vez que ele deve estar presente na classe Mãe da hierarquia (celular mais simples).

Funcionamento da Herança para os Construtores

Vimos que os construtores são invocados em tempo de execução quando o programador digita *new* para instanciar alguma classe, como no exemplo da classe carro acima. Mas, o que realmente acontece quando fazemos *new* Carro()?

Digamos que a classe Carro *extends* Veiculo e que Veiculo *extends Object*, como já vimos anteriormente, a palavra-chave *extends* é utilizada para determinar que uma classe herde de outra. Entraremos em mais detalhes sobre herança nos capítulos seguintes. Por agora, basta entender que Carro é uma subclasse de Veiculo e que Veículo é uma subclasse de Object. Exemplo:

```
1 class Carro extends Veiculo {}
```

```
1 class Veiculo extends Object {}
```

Agora, o que o acontece quando invocamos *new* Carro()?

1. O construtor de carro será invocado. Todo construtor invoca o construtor da sua superclasse com uma chamada implícita para *super()*, a menos que a classe invoque um construtor sobrecarregado da mesma classe (falaremos sobre isso depois).

2. O construtor da classe Veiculo será chamado, que por sua vez chamará o construtor da classe Object.

3. Como a classe Object é a última super classe de todas as classes, podemos concluir que a classe Veiculo *extends* Object mesmo que você não declare a chamada *extends* explicitamente. Chegando ao topo da pilha, o construtor da classe Object será executado primeiramente antes de todos os outros. Os seus atributos de instância serão inicializados e finalizamos o seu construtor.

4. Fazendo o caminho de volta, então, é chamado o construtor da classe Veículo e os seus atributos de instância agora terão seus valores inicializados. Finalizamos o construtor da classe Veículo.

5. E por fim, o construtor da classe Carro também será chamado e os valores de seus atributos de instancia serão inicializados. E finalizamos o construtor de Carro.

A figura a seguir demonstra como os construtores funcionam na pilha.

4. Object()
3. Veiculo chama **super**Object()
2. Carro chama **super** Veiculo()
1. main() chama **new** Carro()

O que faz o operador super?

- Ele chamará o construtor da superclasse para que essa possa realizar as suas inicializações antes mesmo que todas as suas classes filhas.
- Uma chamada para super() poderá ser sem parâmetros ou com parâmetros, dependendo dos argumentos especificados no construtor da superclasse.

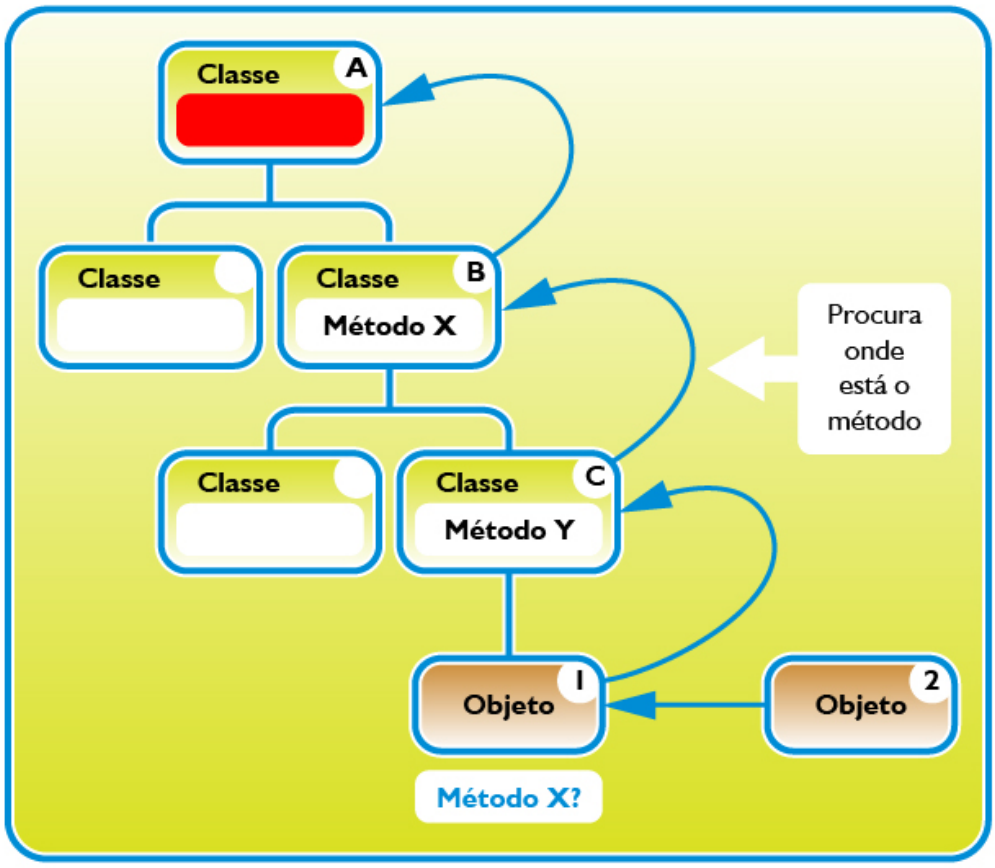
Você não pode fazer uma chamada para um método de instância ou um atributo de instância, até que o construtor da sua superclasse tenha finalizado.

Funcionamento da Herança para os Métodos

A herança funciona para os métodos durante a execução do programa, de maneira similar ao funcionamento para os atributos (ver Aula 10 – Coleções em Java). A Figura 2 ilustra tal situação.

Quando o **Objeto2** solicita a execução do **metodoX()** ao **Objeto1**, esse último inicialmente procura pelo método na própria classe C que o define. Caso não encontre, ele sai percorrendo sua árvore hierárquica. Nesse caso, em particular, o metodoX() é encontrado logo na primeira classe ancestral visitada, a **ClasseB**.

Figura 02 - Herança e os métodos



Vídeo 04 - Herdando Métodos

Leitura Complementar

A seguir, temos dois links para artigos da web que tratam sobre Herança:

<https://www.caelum.com.br/apostila-java-orientacao-objetos/heranca-reescrita-e-polimorfismo/#7-7-exercicios-heranca-e-polimorfismo>

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

Resumo

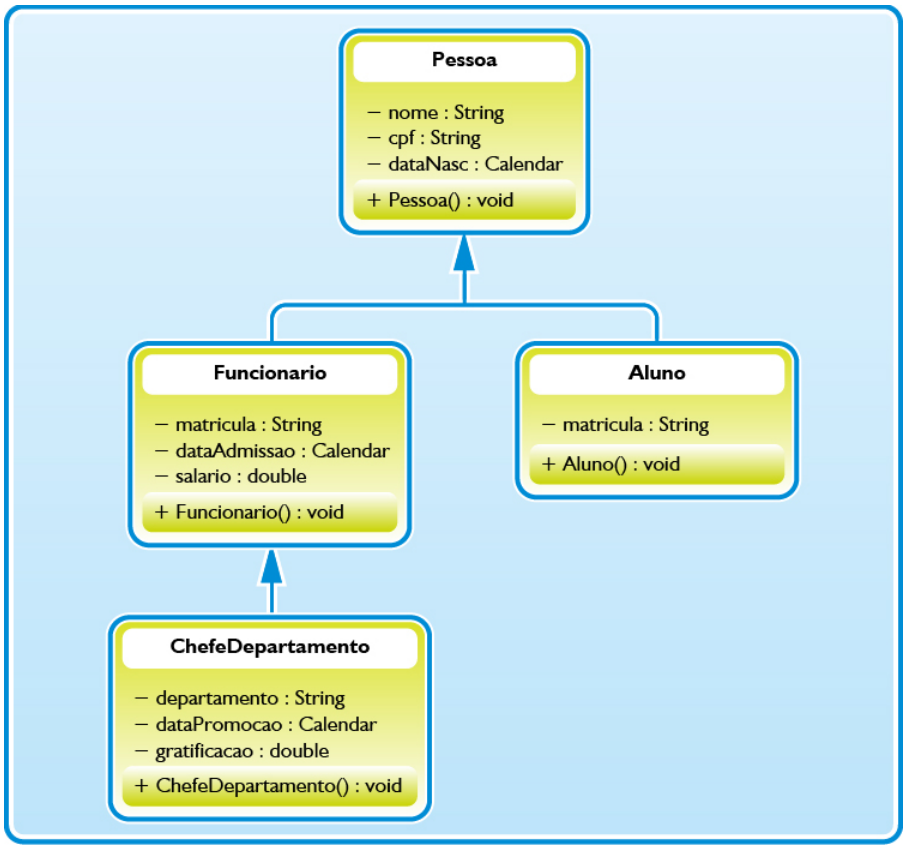
Nesta aula, você aprendeu que a Herança é a capacidade que uma classe tem de herdar as características e comportamentos (atributos e métodos) de outra classe. Em particular, enfatizamos a herança de métodos entre classes. Você viu que um método de uma classe mãe para ser herdado em Java por uma classe filha não pode ser declarado como **private**.

Autoavaliação

1. Para que serve o modificador de acesso **protected**? Como ele funciona no caso de herança entre classes?
2. Para que serve a palavra-chave **super**? Dê um exemplo concreto do seu funcionamento.
3. Existe alguma diferença entre o funcionamento da herança para os atributos e métodos?
4. Descreva o que acontece com o acesso aos atributos e métodos quando são do tipo:
    a. public
    b. private

c. protected

5. Crie as classes utilizando o princípio da herança, obedecendo à hierarquia da figura abaixo (obs.: para facilitar, substitua na figura o tipo Calendar por String).



a. Acrescente aos construtores a lista de parâmetros necessária para instanciar o objeto. Por exemplo, a classe Pessoa deve ter nome, CPF e dataNasc. E essa lista é acumulativa, ou seja, o construtor da classe Funcionário deve ter a lista de seus atributos mais os atributos necessários para a classe **Pessoa**. Dica: não deixe de usar a palavra-chave super em cada um dos construtores para chamar o construtor da classe mãe, passando os atributos que são mantidos por ela e seus ancestrais.

b. Insira os seguintes métodos para apresentar os valores dos atributos das classes, mostrarPessoa(), mostrarFuncionario(), mostrarChefe() e mostrarAluno(), respectivamente, às classes **Pessoa, Funcionário, ChefeDepartamento** e **Aluno**. Para imprimir os atributos, use o método System.out.println() em cada um dos métodos.

c. Crie uma classe **TestaTudo** com um método main(), que instancia um objeto de cada uma das classe e exibe os valores dos atributos através de chamadas aos métodos mostrarPessoa(), mostrarFuncionario(), mostrarChefe() e mostrarAluno().

Referências

SANTOS, Rafael. **Introdução à Programação Orientada a Objetos Usando Java**. Rio de Janeiro: Editora Campus, 2003.

THE JAVA Tutorials. Disponível em: http://java.sun.com/docs/books/tutorial/>. Acesso em: 16 maio 2010.