# Linguagem de Programação II

Prof. Marc Antonio Vieira de Queiroz

Ciência da Computação - Sistemas de Informação - UNIFIL
*marc.queiroz@unifil.br*

06/06/2013

# Roteiro da aula I

1. 8.3 - Herança

2. 8.3.1 A palavra-chave super

# 8.3 Herança I

O mecanismo de reaproveitamento por delegação ou composição permite o reuso de classes já existentes como instâncias de novas classes. As classes originais ficam assim contidas na nova classe.

Reuso de classes via o mecanismo de delegação é útil quando consideramos que a classe que reusa instâncias de outras é composta das outras classes. Um bom exemplo é o da classe DataHora, que é composta das classes Data e Hora. Outros exemplos seriam dados pelas classes que simplesmente utilizam uma ou mais instâncias da classe Data juntamente com outros dados.

Nem sempre o mecanismo de delegação é o mais natural para reutilização de classes já existentes, embora seja simples. Em especial, quando queremos usar uma classe para servir de base à criação de outra mais especializada, a relação de composição imposta

# 8.3 Herança II

pelo uso do mecanismo de delegação acaba por criar soluções pouco naturais.

Como exemplo consideremos as classes Pessoa e Funcionario: uma instância de Pessoa pode ser declarada dentro da classe Funcionario para representar os dados da pessoa/funcionário. Se criássemos uma classe ChefeDeDepartamento considerando que um chefe de departamento é um funcionário que é responsável por um departamento, poderíamos declarar uma instância de Funcionario dentro da classe ChefeDeDepartamento e acrescentar alguns campos que diferenciam ChefeDeDepartamento de Funcionario.

A relação entre as classes ChefeDeDepartamento, Funcionario e Pessoa seria, então, de com posição: pela declaração dos seus campos veríamos que um ChefeDeDepartamento contém um Funcionario, que por sua vez contém uma Pessoa. Apesar de ser

# 8.3 Herança III

possível descrever as classes assim, a solução é pouco natural: embora possamos dizer que a classe DataHora contém uma data e uma hora, declarar que a classe ChefeDeDepartamento contém um funcionário soa meio artificial - na verdade, um chefe de departamento é um tipo de funcionário, que deve ter campos adicionais para representar dados que são específicos de um chefe de departamento, e métodos para manipular estes campos.

Java oferece outra maneira de reutilizar classes, através do mecanismo de herança, que permite que criemos uma classe usando outra como base e descrevendo ou implementando as diferenças e adições da classe usada como base, reutilizando os campos e métodos não-privados da classe-base. O mecanismo de herança é o mais apropriado para criar relações é-um-tipo-de entre classes.

# 8.3 Herança IV

Com o mecanismo de herança, podemos declarar a classe Funcionario como sendo um tipo de Pessoa, e a classe Funcionario herdará todos os campos e métodos da classe Pessoa, não sendo necessária a sua redeclaração. Evidentemente uma classe herdeira pode acrescentar campos e métodos à classe original. Um primeiro exemplo de herança é visto a seguir, com as classes Pessoa, Funcionario e ChefeDeDepartamento.

# Exemplo listagem 8.7 I

Listagem 8.7: A classe Pessoa, que encapsula os dados de identificação de uma pessoa. A classe Pessoa (mostrada na listagem 8.7) contém os campos para representar uma pessoa (com finalidade de identificação). Todos os campos são privados, devendo ser acessados através dos métodos da classe. Esta classe será usada como base para a classe Funcionario, mostrada na listagem 8.8.
Listagem 8.8: A classe Funcionario, que encapsula os dados básicos de um funcionário de uma empresa e herda da classe Pessoa.
Outro exemplo de herança é dado pela classe ChefeDeDepartamento, que herda da classe Funcionario e é mostrada na listagem 8.9.

# Ilustrando o processo I

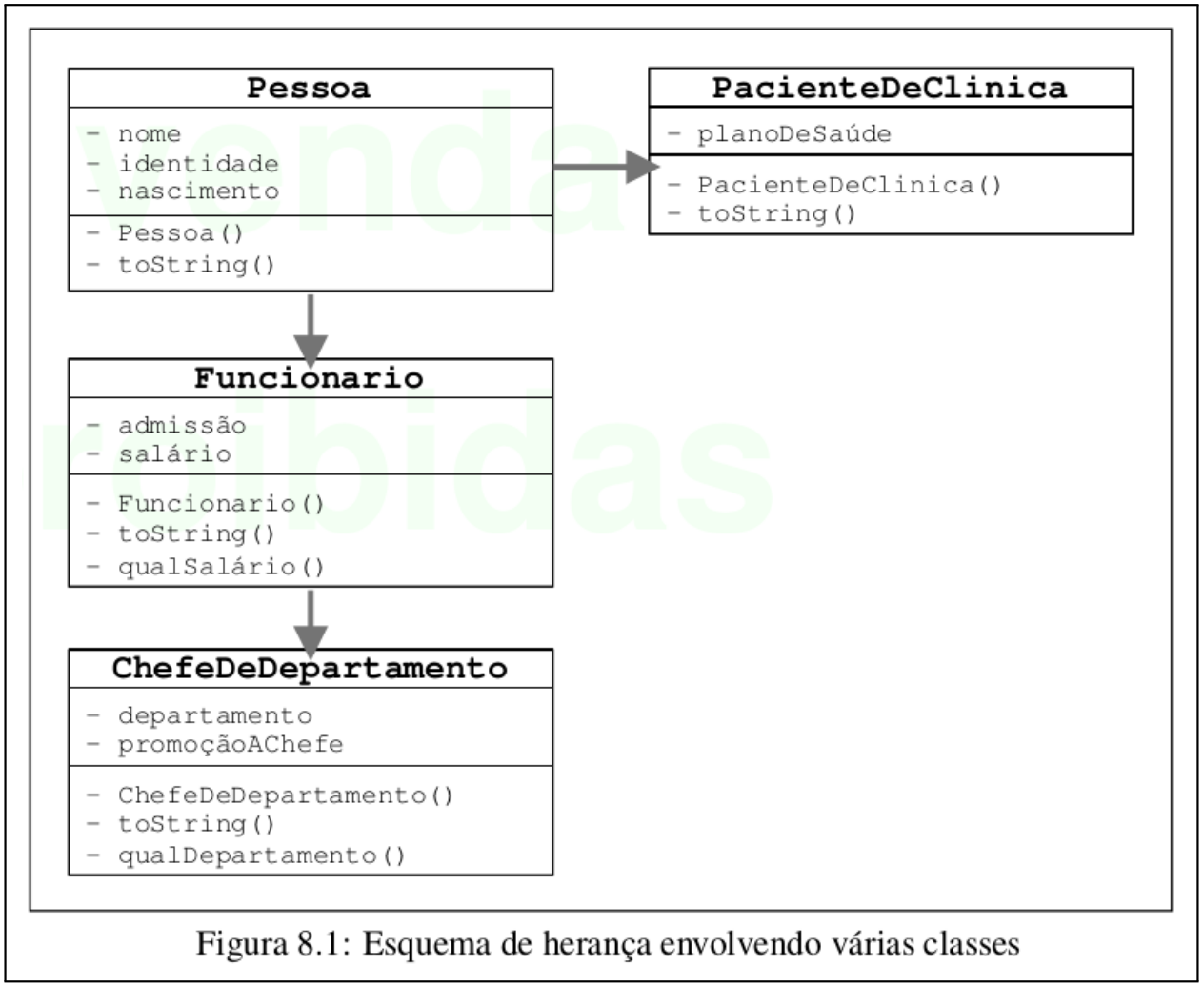


Figura 8.1: Esquema de herança envolvendo várias classes

Todas as classes de Java descendem de uma classe chamada Object - mesmo que a declaração extends Object seja omitida de classes criadas pelo usuário, elas implicitamente herdarão da classe Object. Esta classe por si não contém campos ou métodos úteis, não devendo ser usada diretamente, servindo mais para declarar métodos genéricos (como, por exemplo, toString) que todas as classes devem implementar através da sobreposição.

# A palavra-chave super I

Nos exemplos mostrados nas classes Funcionario e ChefeDeDepartamento (listagens 8.8 e 8.9) vimos que classes derivadas ou subclasses podem ter acesso a métodos das superclasses, usando a palavra-chave super. O acesso a métodos de classes ancestrais é útil para aumentar a reutilização de código: se existem métodos na classe ancestral que podem efetuar parte do processamento necessário, devemos usar o código que já existe (e, esperançosamente, funciona) em vez de reescrevê-lo.

Existem duas maneiras de se reutilizar métodos de classes ancestrais que não tenham sido declarados como private: se a execução do método for a mesma para a superclasse e a subclasse, então instâncias da subclasse podem chamar diretamente o método como se fosse delas mesmas - é o caso do método qualSalário, que é declarado na classe Funcionario e pode ser executado tanto por

# A palavra-chave super II

instâncias da classe Funcionario quanto por instâncias da classe ChefeDeDepartamento, uma vez que esta classe herda da classe Funcionario. Como o método não teve que ser reescrito ou mesmo declarado na classe ChefeDeDepartamento, houve uma economia no desenvolvimento da classe descendente.

A segunda maneira de executar métodos da classe ancestral é mais complexa: partimos do pressuposto que não existe na classe ancestral um método que faça, para a classe descendente, exatamente o que queremos, mas existem métodos que executam parte da tarefa ou resolvem parte do problema. Desta maneira, uma classe descendente poderia executar a parte que resolve o problema parcial chamando o método correspondente da classe ancestral e depois executar comandos que completassem a função desejada.

Isto frequentemente ocorre em construtores: em muitos casos a função dos construtores é inicializar os campos das classes. Consideremos como exemplo a classe Ponto2D, que contém os campos x e y e seu construtor, que tem como finalidade inicializar estes campos com argumentos passados ao construtor. Consideremos a classe Ponto3D que herda da classe Ponto2D e contém o campo z - a classe Ponto3D também contém (via herança) os campos x e y. Se o construtor de Ponto3D for inicializar os campos x, y e z, ele pode

delegar a inicialização de x e y ao construtor da classe Ponto2D e inicializar somente o campo z.

Um método ou construtor de uma subclasse pode chamar um método ou construtor de uma superclasse usando a palavra-chave super, cujo funcionamento é similar ao da palavra-chave this (seção 4.3.1) exceto que super se refere à classe ancestral imediata, e this à própria classe.

Algumas regras para uso da palavra-chave super para chamar métodos de classes ancestrais como subrotinas são:

Construtores são chamados simplesmente pela palavra-chave super seguida dos argumentos a serem passados para o construtor entre parênteses. Se não houverem argumentos, a construtores chamada deve ser feita como super(). Métodos são chamados pela palavra-chave super seguida de um ponto e do nome do super e método. Se houverem argumentos a serem passados para o método, estes devem estar métodos entre parênteses, após o nome do

método, caso contrário (chamada de métodos que não recebem argumentos) os parênteses devem estar vazios.

Construtores de superclasses só podem ser chamados de dentro de construtores de subclasses, e mesmo assim, somente se forem declarados na primeira linha de código do construtor da subclasse (comentários não são considerados como código). Em outras palavras, quando um construtor de uma superclasse for chamado de dentro de um construtor de uma subclasse, a linha que faz a chamada deve ser a primeira do corpo do construtor da subclasse, e métodos não podem chamar construtores de superclasses.

Somente os métodos e construtores da superclasse imediata podem ser chamados usando a palavra-chave super - não existem construções como super.super que permitam a execução de métodos e construtores de classes ancestrais da classe ancestral.

Caso seja necessário executar o construtor de uma classe ancestral da própria classe ancestral, os construtores podem ser

escritos em cascata, de forma que se a classe C herda da classe B que por sua vez herda da classe A, o construtor da classe C pode chamar o construtor da classe B que por sua vez pode chamar o construtor da classe A. Um exemplo desta técnica é mostrado nos construtores das classes ChefeDeDepartamento e Funcionario(listagens 8.9 e 8.8).

Se um método de uma classe ancestral for herdado pela classe descendente, ele pode ser chamado diretamente sem necessidade da palavra super. Por exemplo, se a classe Robo tem um método andaParaAFrente e a classe MiniRobo herda da classe Robo, a não ser que a classe MiniRobo sobreponha o método andaParaAFrente da classe Robo, ela poderá chamá-lo como se tivesse sido declarado na própria classe MiniRobo. O conceito de sobreposição será visto nas próximas aulas.

Um ponto particularmente curioso (e que causa confusão) é que o construtor de uma subclasse sempre chama o construtor de sua superclasse, mesmo que a chamada não seja explícita. Se a classe Ponto3D herda da classe Ponto2D o construtor de Ponto3D deve chamar o construtor da classe Ponto2D. Se a chamada não for explícita (através da palavra-chave super), o construtor da classe Ponto3D tentará chamar o construtor vazio (isto é, sem argumentos) da classe Ponto2D- se este construtor não estiver definido, um erro de compilação ocorrerá.

Desta forma é sempre importante lembrar que se uma classe não tem um construtor sem argumentos mas tem um com argumentos e uma classe herdeira é criada, esta classe herdeira deverá obrigatoriamente chamar o construtor com argumentos da classe ancestral.

Um exemplo mais detalhado que ilustra os tópicos discutidos anteriormente é dado pelas classes Automovel, AutomovelBasico e

AutomovelDeLuxo, mostradas respectivamente nas listagens 8.10, 8.11 e 8.12.

A classe Automovel representa um automóvel à venda, com campos que descrevem os seus dados e métodos para manipulação destes dados. Consideraremos que um automóvel representado por esta classe não tem nenhum acessório ou equipamento adicional em especial. As classes AutomovelBasico e AutomovelDeLuxo encapsulam dados que representam automóveis com acessórios e equipamentos, além dos dados básicos encapsulados pela classe Automovel.

Listagem 8.10: A classe Automovel, que encapsula os dados de um automóvel simples à venda.

A classe Automovel é usada como base para a criação da classe AutomovelBasico, que além de conter os campos e métodos não-privados de Automovel ainda tem campos que representam

acessórios e opcionais de um automóvel. A classe AutomovelBasico é mostrada na listagem 8.11.

Listagem 8.11: A classe AutomovelBasico, que encapsula os dados de um automóvel básico à venda, e que herda da classe Automovel.

Para reforçar mais ainda os conceitos de herança e uso da palavra-chave super, consideremos a classe AutomovelDeLuxo, que herda diretamente da classe AutomovelBasico e indiretamente da classe Automovel. A classe AutomovelDeLuxo é mostrada na listagem 8.12.

Listagem 8.12: A classe AutomovelDeLuxo, que encapsula os dados de um automóvel de luxo à venda, e que herda da classe AutomovelBasico.

A classe DemoAutomoveis, mostrada na listagem 8.13, exemplifica o uso de instâncias das classes Automovel, AutomovelBasico e AutomovelDeLuxo. Listagem 8.13: A classe DemoAutomoveis, que demonstra instâncias das classes Automovel, AutomovelBasico e AutomovelDeLuxo.