Linguagem de Programação II

Prof. Marc Antonio Vieira de Queiroz

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

11/04/2013

Roteiro da aula I

Introdução I

Nas aulas anteriores vimos que podemos criar aplicações que utilizam instâncias de classes definidas pelo usuário ou já existentes. Para criar instâncias precisamos usar a palavra-chave **new** que criará a instância da clas se, e, em geral, associara a instância recém-criada a uma referência, para que métodos da instância possam ser executados.

Até agora, além de inicializar a instância da classe com a palavra-chave **new**, usamos métodos para inicializar os campos da instâncias. O uso destes métodos é de responsabilidade do programador usuário das classes - para criar as instâncias o compilador obriga o uso da palavra-chave **new**, mas não obriga o uso dos métodos de inicialização. Desta maneira, por esquecimento, um programador usuário pode criar a instância de uma classe mas não inicializar os seus dados.

Introdução II

Para exemplificar os erros que podem ocorrer quando instâncias são inicializadas mas os dados encapsulados não o são, consideremos as classes RegistroAcademicosemconstrutor (listagem 4.1) que encapsula os dados de um registro acadêmico simples (basicamente para cálculo da mensalidade) e a classe DemoRegistroAcademicosemconstrutor (listagem 4.2) que demonstra usos de instâncias desta classe.

Introdução I

Podemos ver que na listagem 4.2 uma instância da classe RegistroAcademicoSemConstrutor não teve seus dados inicializados, o que não impede o compilador de compilar corretamente a classe, nem a máquina virtual Java de a interpretar. O resultado do programa na listagem 4.2 é mostrado a seguir:

- A mensalidade do Michael é 500.0
- A mensalidade do Roberto é NaN

A mensalidade do primeiro aluno foi calculada corretamente: a mensalidade de seu curso é 500 reais e o percentual de cobrança é 100% (sem bolsa). No caso do segundo aluno, os dados simplesmente não foram informados - para a sua instância os valores numéricos foram considerados como zero, e a mensalidade calculada como $0 * 100.0/0.0$ (veja a linha 47 da listagem 4.1). A divisão de zero por

Introdução II

zero (quando ambos são valores de ponto flutuante) é representada por Java como o valor NaN (Not a Number, não é um número).

Em muitas situações será necessária forçar o programador usuário a passar dados para as instâncias criadas em classes e programas para que estas tenham sentido. Isto pode ser feito usando construtor: . Este capítulo discutirá a criação e uso de construtores e de sobrecarga de métodos, que permite que criemos métodos com nomes iguais mas funções diferentes.

O que são construtores? I

Construtores são métodos especiais, que são chamados automaticamente quando instâncias são criadas através da palavra-chave **new**. Através da criação de construtores, podemos garantir que o código que eles contém será executado antes de qualquer outro código em outros métodos, já que uma instância de uma classe só pode ser usada depois de ter sido criada com **new**, o que causará a execução do construtor.

Construtores são particularmente úteis para inicializar campos de instâncias de classes para garantir que, quando métodos destas instâncias forem chamados, elas contenham valores específicos e não os *default*. Caso os campos de uma instância não sejam inicializados, os seguintes valores serão adotados:

O que são construtores? II

- Campos do tipo *boolean* são inicializados automaticamente com a constante *false*.
- Campos do tipo *char* são inicializados com o caracter cujo código Unicode é zero e que é impresso como um espaço.
- Campos de tipos inteiros (*byte*, *short*, *long*, *int*) ou de ponto flutuante (*float*, *double*) são automaticamente inicializados com o valor zero, do tipo do campo declarado. Instâncias de qualquer classe, inclusive da classe *String*, são inicializadas automaticamente com *null*.

As diferenças básicas entre construtores e outros métodos são:

- Construtores devem ter **exatamente** o mesmo nome da classe a que pertencem, inclusive considerando maiúsculas e minúsculas.

O que são construtores? III

- Construtores não podem retornar nenhum valor, nem mesmo *void*, portanto devem ser declarados sem tipo de retorno.
- Construtores não devem receber modificadores como *public* ou *private*, e serão públicos se a classe for pública. Não há muito sentido em declarar um construtor como sendo *private*, a não ser quando um construtor privado for chamado de outro construtor público. Veremos em breve que é possível termos mais de um método ou construtor com o mesmo nome através de sobrecarga.

A razão pela qual os construtores tem regras mais fixas para nomenclatura do que métodos é que quando uma instância de uma classe que tem construtores for inicializada com a palavra-chave *new*, o compilador executará automaticamente o construtor, precisando então saber exatamente qual é o nome deste. Uma outra diferença

O que são construtores? IV

significativa entre construtores e métodos comuns é que o programador não pode chamar construtores diretamente - somente quando a instância for inicializada com *new*.

Para exemplificar o papel e importância de construtores em uma classe, consideramos a classe *EventoAcademico*, na listagem 4.3, que representa um evento acadêmico como congresso, simpósio ou reunião.

Classe EventoAcademico I

Alguns pontos interessantes da classe *EventoAcademico* são:

- O construtor da classe, declarado na linha 25, é bem similar a um método para inicialização dos dados encapsulados na classe, mas seguem as regras de nomenclatura de construtores. Quando houverem instâncias de outras classes sendo usadas como campos de uma classe, estas instâncias podem também ser inicializadas no construtor. Desta maneira, o construtor vai criando e inicializando todos os campos necessários.

Classe EventoAcademico II

- Na classe *EventoAcademico* existem duas referências a instâncias da classe Data que devem ser inicializadas explicitamente, ao invés de simplesmente copiadas. A inicialização é feita através da palavra-chave *new*, e o método *inicializaData* da classe *Data* é chamado, usando como argumentos os valores obtidos da instância passada como argumento para o construtor. Desta forma teremos uma cópia exata dos dados da data passada como argumento, e não uma referência apontando para a mesma instância.

Para exemplificar o uso de instâncias da classe *EventoAcademico*, usamos a aplicação na classe *DemoEventoAcademico*, mostrada na listagem 4.4.

Resumo:

Classe EventoAcademico III

- Instâncias da classe EventoAcademico só podem ser inicializadas com a palavra-chave *new* se argumentos forem passados para seu construtor. Desta forma, para esta classe, não é possível criar instâncias que tenham valores não-inicializados.
- Duas instâncias temporárias da classe Data são criadas na classe DemoEventoAcademico (linhas 24 e 25) somente para que sejam passadas como argumentos para os construtores da classe EventoAcademico. Estas instâncias são reaproveitadas ou reusadas com valores diferentes, mas não existem problemas de cópias de referências uma vez que, internamente na classe EventoAcademico, novas instâncias internas são criadas. Em resumo, mesmo depois que os conteúdos das instâncias data1 e data2 são modificados (linhas 33 e 34 da listagem 4.4) as datas

Classe EventoAcademico IV

representadas internamente na instância SBED1998 da classe EventoAcademico não são modificadas.

- Podemos ver a praticidade de se ter um método toString na classe: podemos pedir ao método System out.println que imprima a instância da classe EventoAcademico, e System.out.println executará automaticamente o método toString, imprimindo a String resultante.

Construtor padrão I

Mesmo quando as classes criadas pelo programador não tem um construtor declarado explicitamente, o compilador Java cria um construtor padrão, que não recebe argumentos nem executa nenhum código. Quando o programador de classes cria um ou mais construtores, o compilador não inclui o construtor padrão.