

Mais sobre Expressões

Dalton Serey
© 2004 DSC/UFCG

Expressões

- Uma expressão é um **trecho de código** que, em tempo de execução, é avaliado para **produzir** alguma **coisa** de algum **tipo**.
- Literais são as expressões mais simples:
 - exemplos: 5, 2.0, 'c', "ooprog" false
- Como construímos expressões mais complexas?
 - combinando coisas de acordo com as operações que seus tipos permitem

Avaliação de expressões

- Ao executarmos o programa, Java deve **avaliar** as expressões
- Ou seja, **calcular algo a partir delas**
- Por exemplo, avaliar 5 + 13 significa que Java deve produzir 18
 - o valor *inteiro* 18; não o *literal* 18
- Em um programa, **todo valor** é produzido pela avaliação de uma expressão!

Avaliação de expressões

- Há diferentes tipos de expressões
- Para cada tipo, Java determina
 - **sintaxe**: regras que determina como escrever tais expressões corretamente
 - **semântica**: regras sobre como Java deve avaliar expressões sintaticamente corretas
- Exemplos:
 - 5 + 2 (sintaxe ok; valor produzido: 7)
 - 3+ - 4*'a' (sintaxe errada! **sem semântica!**)

Tipos de Expressões

- Tipos de expressões:
 - literais (sintaxe já vista!)
 - operações com literais (também já vistas!)
 - nomes (atributos, variáveis e parâmetros)
 - serviços de objetos
 - criação de objetos
 - comparação de objetos
- Ao longo da aula, veremos a sintaxe e a semântica de cada tipo de expressão

Nomes são expressões

- Nomes podem ser usados como expressões
- Há duas categorias de nomes:
 - os usados para valores primitivos
 - os usados para objetos
- A sintaxe é a mesma:
 - o nome deve ser escrito como foi declarado
 - só deve ser usado dentro do seu **escopo**
- O escopo determina a "validade" de um nome e onde pode ser usado no código

Nomes são expressões

- Escopo de atributos/campos:
 - validade: tempo de vida do objeto
 - dentro do código da classe
- Escopo de parâmetros:
 - validade: tempo de execução do método
 - dentro do código do método
- Escopo de variáveis:
 - validade: tempo de execução do bloco
 - dentro do código do bloco

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Nomes são expressões

- A semântica de nomes depende do tipo:
 - nomes usados para tipos primitivos produzem os próprios **valores**
 - nomes usados para tipos complexos produzem **referências** (ou rótulos) para objetos
- Qual a diferença?

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Nomes são expressões

- Exemplo:

```
int a = 0, b;  
b = a;  
a = a + 1;  
System.println(a + ", " + b);
```

```
MeuContador a, b;  
b = a;  
a.conte();  
System.println(a + ", " + b);
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Serviços são expressões

- **Invocações de métodos** são expressões
- Sintaxe:
 - *expObjeto*.*nomeDoServiço*(*argumentos*)
 - observe que *expObjeto* é uma **expressão** cuja avaliação deve produzir um objeto!
 - da mesma forma, *argumentos* é uma **lista de expressões** que produzem valores dos tipos esperados
 - os tipos esperados dos argumentos são os tipos declarados para os parâmetros na assinatura do método

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Serviços são expressões

- Semântica
 - *expObjeto*.*nomeDoServiço*(*argumentos*)
 - primeiro, Java avalia cada sub-expressão
 - determina-se o objeto a partir de *expObjeto*
 - e os valores dos *argumentos*
 - definido o objeto, o método *nomeDoServiço* do referido objeto é executado com os parâmetros ligados aos valores avaliados
 - o valor e o tipo de retorno do método são usados como o valor e tipo finais da expressão

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Serviços são expressões

- Exemplos:

```
MeuContador a;  
while ( a.valor() != 0 )  
    a.conte();  
System.println( a.valor() );
```

```
a.reset(b.valor())
```

```
"ooprog".substring(2, 4)
```

```
(meuMundo.getClosestBall(this)).getX()
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Expressões Compostas

- Expressões podem ser **compostas**
- Elas consistem na combinação de diversas sub-expressões para formar uma maior
- Para avaliá-las, a regra é:
 - avaliar primeiro todas as sub-expressões
 - combinar os valores obtidos para avaliar expressões de mais alto nível
 - repetir até avaliar a expressão completa

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Efeitos Colaterais

- Algumas expressões de serviços são usadas pelo seu **efeito colateral** e não pelo valor que retornam...
 - interessa o efeito sobre os objetos e as coisas
 - não o valor de retorno
- Exemplos:
 - atribuições
 - serviços sem valor de retorno esperado

```
a = 0;  
contador.conte();
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Tipo de retorno void

- Para os casos em que um serviço não deve retornar nada, Java define o tipo **void**
 - indica que o método não retorna nada
 - só pode ser usado como tipo de retorno!

```
public interface Contador {  
    ...  
    public void reset();  
    public void conte();  
    ...  
}
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Acesso a atributos

- Eis outro tipo de expressão que já usamos
- Sintaxe
 - *expObjeto*.nomeDoAtributo
- Semântica
 - produz o valor do atributo *nomeDoAtributo* armazenado pelo objeto *expObjeto*
 - deve respeitar escopo e visibilidade!
 - na prática funcionam como nomes complexos
 - podemos usá-los do lado esquerdo de atribuições

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Acesso a atributos

- Exemplos

```
aluno.nome = "João";  
aluno.idade = 20;  
System.out.println(aluno.pai.nome);
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Instanciação

- A criação de objetos é outro tipo de expressão que já vimos
- Sintaxe:
 - **new** NomeDaClasse (*argumentos*)
- Semântica:
 - efeito colateral: cria um novo objeto da classe *NomeDaClasse* usando os valores obtidos da lista de *argumentos* (expressões)
 - retorna uma referência para o objeto criado com o tipo da classe indicada

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Avaliação de Tipo

- Em Java há uma expressão para avaliar o tipo de um objeto
- Sintaxe:
 - `expObjeto instanceof NomeDoTipo`
- Semântica:
 - avalia se o objeto calculado pela expressão `expObjeto` é do tipo `NomeDoTipo`
 - retorna um valor-verdade
- Útil quando precisamos de informação mais específica do tipo dos objetos manipulados!

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Às vezes é conveniente tratar coisas por tipos diferentes dos oficialmente declarados
- Por exemplo, como resolver a atribuição `double a = 10`?
- O problema é que o tipo oficial do literal `10` não é um `double`, é um `int`!
- Java, contudo, não reclama disso!
- O que ocorre é chamado de **coerção**

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Coerção é o mecanismo de “enxergar” uma coisa como sendo de um tipo diferente do oficialmente declarado
 - no exemplo, o `int 10` foi interpretado por Java como o `double 10.0`
- Outro exemplo:
 - pode ser conveniente tratar um objeto da classe `MeuContador` como sendo do tipo `Contador`
 - ou o contrário...

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Em alguns casos, a coerção implica em **perda de informação**
 - por exemplo, pode ser conveniente tratar certo `int` como um `short`
 - problema: a faixa de valores dos tipos
 - `[int] >> [short]`
- No caso inverso, certamente não há problema

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Quando a coerção necessária preserva a informação, Java a faz automaticamente
 - é o caso de `double a = 10`
 - ou de `Contador c = new MeuContador()`
- Caso contrário, Java delega ao programador a decisão de fazer ou não a coerção
- Para isso, usamos expressões de **cast**

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Um **cast** faz Java “enxergar” um valor ou objeto como sendo de um tipo não oficial, **mesmo que haja perda de informação**
- Por exemplo,
 - é o caso de `int a = 10.0` não compila
 - mas, `int a = (int)10.0` sim!
 - o `(int)` indica que Java deve fazer a coerção
 - mesmo com a possível perda de informação!

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- A sintaxe do cast é:
 - `(TipoAUsar) expressão`
 - onde `TipoAUsar` indica o tipo segundo o qual, Java deve interpretar o valor obtido pela avaliação da `expressão`
- Importante:
 - uma coerção não muda o valor ou o objeto
 - apenas uma nova “visão” dele é assumida

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Um último exemplo

```
BancoDeDados bd;  
Pessoa p;  
int cpf;  
int matricula;
```

```
p = bd.getPessoa(cpf);  
matricula = p.getMatricula();
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Um último exemplo

```
BancoDeDados bd;  
Pessoa p;  
int cpf;  
int matricula;
```

```
p = bd.getPessoa(cpf);  
matricula = ((Aluno)p).getMatricula();
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Coerções e Casts

- Lembre: mesmo com um cast, a coerção só funciona se o objeto for realmente do tipo indicado!
- Se houver alguma dúvida quanto ao tipo, é indicado fazer uma verificação do tipo

```
p = bd.getPessoa(cpf);  
if ( !(p instanceof Aluno) )  
    // tratamento da situação de erro  
matricula = ((Aluno)p).getMatricula();
```

Versão 1.0

© 2004 Dalton Serey DSC/UFG

Exercício

Versão 1.0

© 2004 Dalton Serey DSC/UFG