TREINA RECIFE

DESENVOLVIMENTO GERENCIAL E PROFISSIONAL

Prof. Rodrigo M. M. Lyra

Curso de Lógica de Programação

Conversão de Tipos Primitivos

A conversão dita implícita consiste em uma atribuição DIRETA de uma constante variável ou expressão de um certo tipo a uma variável de outro tipo. Como o JAVA é fortemente **tipado**, em uma operação de atribuição, tipos não compatíveis incorrem em erro de compilação.

Conversão de Tipos Primitivos

- Quando os tipos da variável que recebe a atribuição e a constante/variável/expressão atribuída são os mesmos.

- Quando os tipos da variável que recebe a atribuição e da constante/variável/expressão atribuída são diferentes, mas o tipo desta última “cabe” no tipo da variável. Quando isto ocorre, há uma conversão implícita do tipo da constante/variável/expressão para o tipo da variável.

- Uma constante/variável/expressão do tipo **boolean** só pode ser atribuída a outro **boolean**.

- O tipo **char** guarda internamente o código UNICODE (um valor inteiro) de um caractere, portanto aceita atribuição de constantes inteiras dentro da sua faixa (0 a 65535), mas não é recomendável fazer este tipo de operação.

Conversão de Tipos Primitivos

Desta forma, podemos dizer que as conversões implícitas permitidas serão comumente realizadas entre tipos numéricos que representam números inteiros e decimais. Quando a atribuição envolve tipos diferentes, o tipo da constante/variável/expressão deve “caber” no tipo da variável. Isso significa que a faixa de valores do tipo atribuído deve ser mais restrita do que a faixa de valores da variável que recebe a atribuição.

É simples saber “quem cabe em quem” no universo de tipos primitivos inteiros e reais do JAVA. É só olhar para a sequência abaixo. Da esquerda para a direita, temos os tipos mais largos e os tipos mais curtos.

double - float - long - int - short - byte

Conversão de Tipos Primitivos

Em uma atribuição **DIRETA** envolvendo tipos diferentes, só há duas opções: ou o código **COMPILA** ou **NÃO COMPILA**. Para inferir isso, basta olhar para os tipos da esquerda e da direita na atribuição e verificar na sequência, se estes se encontram, respectivamente, à esquerda e à direita. Se sim, a atribuição compila. Se não, a atribuição não compila.

`int a = 2;` `double d = a; //esta linha!`	Tipo da esquerda: double (o tipo da variável de destino "d") Tipo da direita: int (o tipo da variável "a") Sequência da atribuição: **double -> int** Sequência de referência: **double** – float – long – **int** – short – byte Uma outra forma de pensar: estou atribuindo um int a um double e a faixa de valores suportada por um double é mais ampla do que a do int, logo, um int "cabe" em um double. **A atribuição COMPILA!**
`float f = 39.22;`	Tipo da esquerda: float (o tipo da variável de destino "f") Tipo da direita: double (o tipo da constante literal 39.22) Sequência da atribuição: **float -> double** Sequência de referência: **double** – **float** – long – int – short – byte Uma outra forma de pensar: estou atribuindo um double a um float e a faixa de valores suportada por um float é mais restrita do que a do double, logo um float "não cabe" em um double. **A atribuição NÃO COMPILA!**
`final long VALOR = 2012L;` `float z = VALOR; //esta linha!`	Tipo da esquerda: float (o tipo da variável de destino "z") Tipo da direita: long (o tipo da constante "VALOR") Sequência da atribuição: **float -> long** Sequência de referência: double – **float** – **long** – int – short – byte Uma outra forma de pensar: estou atribuindo um long a um float e a faixa de valores suportada por um float é mais ampla do que a do long, logo, um long "cabe" em um float. **A atribuição COMPILA!**
`int k = 333;` `short s = k; //esta linha!`	Tipo da esquerda: short (o tipo da variável de destino "s") Tipo da direita: int (o tipo da variável k) Sequência da atribuição: **short -> int** Sequência de referência: double – float – long – **int** – **short** – byte Uma outra forma de pensar: estou atribuindo um int a um short e a faixa de valores suportada por um short é mais restrita do que a do int, logo, um int "não cabe" em um short. **A atribuição NÃO COMPILA!**

Conversão de Tipos Primitivos

Valores possíveis					
Inteiro	**Primitivo**	**Menor**	**Maior**	**Valor Padrão**	**Tamanho**
Inteiro	byte	-128	127	0	8 bits
	short	-32768	32767	0	16 bits
	int	-2.147.483.648	2.147.483.647	0	32 bits
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits
	double	-4,94E-307	1.79769313486232570E + 308	0	64 bits
Caractere	char	0	65535	\0	16 bits
Booleano	boolean	false	true	false	1 bits

Conversão de Tipos Primitivos

A conversão dita explícita consiste em uma atribuição envolvendo tipos diferentes, mediada por um operador chamado CASTING, que significa “moldagem” em tradução livre. Tal operação é necessária para forçar a conversão de tipos mais “largos” em tipos mais “curtos”, evitando assim o erro de compilação que observamos quando tentamos realizar esta conversão de forma implícita e direta em uma atribuição.

Na conversão de um valor de tipo mais “largo”, para um tipo mais “curto”, pode haver perda de informação. Por exemplo, se um programador tenta converter um valor real com decimal diferente de zero em um inteiro, a parte decimal será descartada. Não há como armazenar internamente, nem mesmo representar casas decimais em uma variável de um tipo inteiro.

***Por outro lado, se o programador sabe de antemão, que o valor a ser convertido não tem casa decimal significativa, há uma chance maior desta conversão dar certo. De uma maneira geral, uma conversão explícita dará certo, se o valor a ser convertido estiver na faixa de valores do tipo da variável de destino que está recebendo o valor convertido. Vamos a alguns exemplos conceituais.**

Conversão de Tipos Primitivos

Atribuir 22112L a uma variável do tipo int	22112L é uma constante literal do tipo long, em princípio, o tipo long não “cabe” no tipo int, e de acordo com as regras de conversão implícita, não é possível realizar uma atribuição direta. Mas, se convertermos 22112L em int através da operação de CASTING, podemos realizar tal atribuição. A pergunta é: nesta conversão, o valor 22112 é mantido? É sim, pois 22112 está dentro da faixa de valores do tipo int, que vai de -2147483648 a 2147483647. Esta conversão pode ser feita sem perda de informação.
Atribuir 255L a uma variável do tipo byte	255L é uma constante literal do tipo long, em princípio, o tipo long não “cabe” no tipo byte, e de acordo com as regras de conversão implícita, não é possível realizar uma atribuição direta. Mas, se convertermos 255L em byte através da operação de CASTING, podemos realizar tal atribuição. A pergunta é: nesta conversão, o valor 255 é mantido? Não é, pois 255 não está dentro da faixa de valores do tipo byte, que vai de -128 a 127. Esta conversão pode ser feita, mas a informação original será perdida, pois o VALOR em questão está fora da faixa do tipo que o está recebendo. Neste caso, o JAVA faz uma conversão interna e o valor final da variável do tipo byte seria de -1, totalmente sem sentido.
Atribuir 3.11 a uma variável do tipo short	3.11 é uma constante literal do tipo double, em princípio, o tipo double não “cabe” no tipo short, e de acordo com as regras de conversão implícita, não é possível realizar uma atribuição direta. Mas, se convertermos 3.11 em short através da operação de CASTING, podemos realizar tal atribuição. A pergunta é: nesta conversão, o valor 3.11 é mantido? Não é, pois 3.11 possui casas decimais significativas e um tipo inteiro (no caso o short) não as suporta. Esta conversão pode ser feita, mas as casas decimais serão perdidas. Neste caso, o JAVA as despreza e atribui ao short apenas a parte inteira do double. Aí, uma segunda análise deve ser feita: o valor da parte inteira está na faixa do short? Sim, 3 está dentro da faixa de valores do short, que vai de -32768 a 32767. Neste caso, o JAVA faz uma conversão interna e o valor final da variável do tipo short seria de 3.
Atribuir 255.1F a uma variável do tipo byte	255.1F é uma constante literal do tipo float, em princípio, o tipo float não “cabe” no tipo byte, e de acordo com as regras de conversão implícita, não é possível realizar uma atribuição direta. Mas, se convertermos 255.1F em short através da operação de CASTING, podemos realizar tal atribuição. A pergunta é: nesta conversão, o valor 255.1 é mantido? Não é, pois 255.1 possui casas decimais significativas e um tipo inteiro (no caso o byte) não as suporta. Esta conversão pode ser feita, mas as casas decimais serão perdidas. Neste caso, o JAVA as despreza e atribui ao byte apenas a parte inteira do float. Aí, uma segunda análise deve ser feita: o valor da parte inteira está na faixa do byte? Não, 255 está fora da faixa de valores do byte, que vai de -128 a 127. Neste caso, o JAVA faz uma conversão interna e o valor final da variável do tipo byte seria de -1, totalmente sem sentido.

Conversão de Tipos Primitivos

Situações possíveis, respectivamente, em uma operação de CASTING:

- Conversão sem nenhuma perda.
- Conversão com perda da parte inteira.
- Conversão com perda da parte decimal.
- Conversão com perda das partes decimal e inteira.

Uma operação de CASTING consiste, normalmente, em uma linha de atribuição, em realizar uma conversão explícita de tipos no lado direito da atribuição, usando o operador de casting “(TIPO)”

Conversão de Tipos Primitivos

A atribuição em questão ocorre em duas etapas, sendo a primeira a conversão especificada no casting, e a segunda, a atribuição em si. Dessa forma, é necessário saber se o tipo final resultante do casting é compatível com o tipo da variável de destino (eles podem ser diferentes, nada impede isso), pois é ele que será atribuído de forma direta à variável de destino, o que, na prática, pode ser uma conversão implícita, se o tipo da variável de destino e o tipo do casting forem diferentes.

- A linha compila, se o tipo do casting for igual ao tipo da variável de destino ou se o tipo do casting for compatível com o tipo da variável de destino, segundo as regras de conversão implícitas.
- Se a linha não compilar, a verificação para por aqui, pois quem não compila não é executado.
- Quando a linha é executada, deve-se checar se:
    - quando a conversão é de tipos decimais para tipos inteiros, o valor a ser convertido contém conteúdo decimal significativo (casas decimais diferentes de zero). Se houver, há perda da parte decimal.
    - a parte inteira do valor a ser convertido está fora da faixa suportada pelo tipo do casting. Se estiver, há perda da parte inteira.

Operadores Aritiméticos

Em expressões aritméticas, a precedência é a padrão de qualquer linguagem de programação e segue a mesma regra da matemática básica: expressões entre parêntesis (dos mais internos para os mais externos), resto da divisão, divisão, multiplicação, subtração, soma e operações mais à esquerda. Há uma certa confusão na precedência dos operadores unários (a depender da posição do sinal no operando), de forma que é altamente aconselhável não se usar tais operadores em expressões aritméticas.

Tal prática mistura conceitos de matemática (operações aritméticas) e de lógica de programação (incremento), o que torna o código confuso, às vezes ambíguo, e sempre há uma saída programática para livrar as expressões aritméticas dentro de um programa de operadores unários de incremento e de decremento.

Operadores Aritiméticos

Operação	Operador	Tipo	Resultado
Soma	+	Binário	A soma dos dois operandos
Subtração	-	Binário	A subtração dos dois operandos
Multiplicação	*	Binário	O produto dos dois operandos
Divisão	/	Binário	A divisão dos dois operandos, que pode ser inteira ou normal
Resto da divisão	%	Binário	O resto da divisão dos dois operandos
Incremento unário	++	Unário	O incremento do operando (obrigatoriamente uma variável)
Decremento unário	--	Unário	O decremento do operando (obrigatoriamente uma variável)
Soma e atribuição	+=	Binário duplo	A atribuição ao operando à esquerda da soma dele, mesmo com o outro operando
Subtração e atribuição	-=	Binário duplo	A atribuição ao operando à esquerda da subtração dele, mesmo do outro operando
Multiplicação e atribuição	*=	Binário duplo	A atribuição ao operando à esquerda da multiplicação dele, mesmo pelo outro operando
Divisão e atribuição	/=	Binário duplo	A atribuição ao operando à esquerda da divisão dele, mesmo pelo outro operando

Operadores Relacionais

Representam as operações básicas da lógica booleana, são sempre aplicados a operandos do tipo **boolean** e são muito similares aos encontrados em outras linguagens de programação, especialmente no C. O quadro a seguir mostra, para cada operador, a operação, a sintaxe, o tipo do operador e o resultado.

Operadores Relacionais

Operação	Operador	Tipo	Resultado
Igual	==	Binário	Retorna **true**, se os dois operandos forem iguais, e **false**, caso contrário
Diferente	!=	Binário	Retorna **true**, se os dois operandos forem diferentes, e **false**, caso contrário
Maior que	>	Binário	Retorna **true**, se o operando da esquerda for maior que o da direita, e **false**, caso contrário
Maior ou igual	>=	Binário	Retorna **true**, se o operando da esquerda for maior ou igual ao da direita, e **false**, caso contrário
Menor que	<	Binário	Retorna **true**, se o operando da esquerda for menor que o da direita, e **false**, caso contrário
Menor ou igual	<=	Binário	Retorna **true**, se o operando da esquerda for menor ou igual ao da direita, e **false**, caso contrário

Operadores Lógicos

A precedência destes operadores em expressões booleanas é a padrão e segue a mesma regra da lógica booleana: parênteses (dos mais externos para os mais internos) NOT, XOR, AND, OR e operações mais à esquerda.

Operação	Operador	Tipo	Resultado
Conjunção	&&	Binário	Resultado do AND aplicado aos dois operandos
Disjunção	||	Binário	Resultado do OR aplicado aos dois operandos
Negação	!	Unário	Resultado do NOT aplicado ao operando
Disjunção exclusiva	^	Binário	Resultado do XOR aplicado aos dois operandos

Operadores Lógicos

Para efeito de revisão, são dadas abaixo as tabelas verdade das operações lógicas acima relacionadas, usando a nomenclatura do JAVA (operadores e representação dos valores lógicos).

Operação AND - &&		
Operando 1	Operando 2	Resultado
false	false	false
false	true	false
true	false	false
true	true	true

Operação OR - ||		
Operando 1	Operando 2	Resultado
false	false	false
false	true	true
true	false	true
true	true	true

Operação NOT - !	
Operando	Resultado
false	true
true	false

Operação XOR - ^		
Operando 1	Operando 2	Resultado
false	false	false
false	true	true
true	false	true
true	true	false

Obrigado!

+55 81 9.8136-8793

@rodrigo_lyra

http://br.linkedin.com/rodrigolyrati

rlyrastreaming@outlook.com