Tipos, Literais, Operadores

Identificadores

- São palavras utilizadas para **nomear** variáveis, métodos e classes
- Na linguagem Java, o identificador **sempre começa** por **letra**, **sublinhado**(_) ou **cifrão** (**$**)
  - Não podem começar por números
  - São “case sensitive”
- Do segundo caractere em diante, pode conter qualquer seqüência de letras, dígitos, sublinhados ou cifrões

Identificadores

- Exemplos de identificadores válidos:
  - nomeAluno
  - saldo
  - lâmpada // não recomendável
  - User_name // fora do padrão
  - _sys_var1
  - Class // não recomendável
- Java utiliza o padrão **Unicode** (16 bits)

Palavras reservadas em Java

abstract	boolean	break	byte	case
catch	char	class	continue	default
do	double	else	extends	
final	finally	float	for	if
implements	import	instanceof	int	interface
long	native	new	return	package
private	protected	public	synchronized	short
static	super	switch	this	try
throw	throws	transient	void	volatile
while	enum			

true, **false** e **null** **não** são palavras reservadas e sim literais

OBS: Todas as palavras reservadas são escritas com letras minúsculas

Palavras reservadas

- Modificadores de Acesso
  - private, protected, public
- Modificadores de classe, método e variável
  - abstract, class, extends, final, implements, interface, native, new, static, synchronized, transient, volatile
- Controle de fluxo
  - break, case, continue, default, do, else, for, if, instanceof, return, switch, while

Palavras reservadas

- Tratamento de Erros
  - catch, finally, throw, throws, try, assert
- Controle de Pacotes
  - import, package
- Tipos de Dados
  - boolean, byte, char, double, float, int, long, short, void (somente retorno)

Palavras reservadas

- Exemplo de códigos que não compilam:

```
class Teste {
   public void go() { }
   public int break(int b) { }
}
```

```
class LiteralTest {
   public static void main(String [] args) {
        int true = 100; //isto causa erro
   }
}
```


Tipos de dados primitivos

- Java, assim como Pascal, C e C++, é uma linguagem **fortemente tipada** -- todas as variáveis devem ter um tipo

- Diferentemente de C e C++, os tipos primitivos em Java são **portáteis** entre todas as plataformas

Tipos primitivos em Java

- Armazenados na pilha (acesso rápido)
- Não são objetos. Classe 'wrapper' faz transformação, se necessário (encapsula valor em um objeto)

Tipo	Tamanho	Mínimo	Máximo	Default	'Wrapper'
boolean	—	—	—	false	Boolean
char	16-bit	Unicode 0	Unicode 2^{16} - 1	\u0000	Character
byte	8-bit	-128	+127	0	Byte
short	16-bit	-2^{15}	$+2^{15}-1$	0	Short
int	32-bit	-2^{31}	$+2^{31}-1$	0	Integer
long	64-bit	-2^{63}	$+2^{63}-1$	0	Long
float	32-bit	IEEE754	IEEE754	0.0	Float
double	64-bit	IEEE754	IEEE754	0.0	Double
void	—	—	—	—	Void

Exemplos

- *Literais de caracter:*
```
char c = 'a';
char z = '\u0041'; // em Unicode
```
- *Literais inteiros*
```
int i = 10;  short s = 15;  byte b = 1;
long hexa = 0x9af0L; int octal = 0633;
```
- *Literais de ponto-flutuante*
```
float f = 123.0f;
double d = 12.3;
double g = .1e-23;
```
- *Literais booleanos*
```
boolean v = true;
boolean f = false;
```
- *Literais de string (não é tipo primitivo - s é uma referência)*
```
String s = "abcde";
```
- *Literais de vetor (não é tipo primitivo - v é uma referência)*
```
int[] v = {5, 6};
```

Caracteres especiais

SEQÜÊNCIA	VALOR DO CARACTERE
\b	Retrocesso (backspace)
\t	Tabulação
\n	Nova Linha (new line)
\f	Alimentação de Formulário (form feed)
\r	Retorno de Carro (carriage return)
\"	Aspas
\'	Aspa
\\	Contra Barra
nnn	O caractere correspondente ao valor octal *nnn*, onde *nnn* é um valor entre 000 e 0377.
\u*nnnn*	O caractere Unicode *nnnn*, onde *nnnn* é de um a quatro dígitos hexadecimais. Seqüências Unicode são processadas antes das demais seqüências.

Operadores

OPERADOR	FUNÇÃO	OPERADOR	FUNÇÃO
+	Adição	~	Complemento
-	Subtração	<<	Deslocamento à esquerda
*	Multiplicação	>>	Deslocamento à direita
/	Divisão	>>>	Desloc. a direita com zeros
%	Resto	=	Atribuição
++	Incremento	+=	Atribuição com adição
--	Decremento	-=	Atribuição com subtração
>	Maior que	*=	Atribuição com multiplicação
>=	Maior ou igual	/=	Atribuição com divisão
<	Menor que	%=	Atribuição com resto
<=	Menor ou igual	&=	Atribuição com AND
==	Igual	|=	Atribuição com OR
!=	Não igual	^=	Atribuição com XOR
!	NÃO lógico	<<=	Atribuição com desl. esquerdo
&&	E lógico	>>=	Atribuição com desloc. direito
||	OU lógico	>>>=	Atrib. C/ desloc. a dir. c/ zeros
&	AND	? :	Operador ternário
^	XOR	(*tipo*)	Conversão de tipos (cast)
|	OR	instanceof	Comparação de tipos

Precedência

- A *precedência* determina em que ordem as operações em uma expressão serão realizadas.
  - Por exemplo, operações de multiplicação são realizadas antes de operações de soma:

```
int x = 2 + 2 * 3 - 9 / 3;  // 2+6-3 = 5
```

- *Parênteses* podem ser usados para sobrepor a precedência

```
int x = (2 + 2) * (3 - 9) / 3; // 4*(- 6)/3 = - 8
```

- A maior parte das expressões de mesma precedência é calculada da *esquerda para a direita*

```
int y = 13 + 2 + 4 + 6; // (((13 + 2) + 4) + 6)
```

  - Há exceções. Por exemplo, atribuição.

Tabela de Precedência

ASSOC	TIPO DE OPERADOR	OPERADOR
D a E	separadores	`[] . ; , ()`
E a D	operadores unários	`new (cast) +expr -expr ~ !`
E a D	incr/decr pré-fixado	`++expr --expr`
E a D	multiplicativo	`* / %`
E a D	aditivo	`+ -`
E a D	deslocamento	`<< >> >>>`
E a D	relacional	`< > >= <= instanceof`
E a D	igualdade	`== !=`
E a D	AND	`&`
E a D	XOR	`^`
E a D	OR	`|`
E a D	E lógico	`&&`
E a D	OU lógico	`||`
D a E	condicional	`?:`
D a E	atribuição	`= += -= *= /= %= >>= <<= >>>= &= ^= !=`
E a D	incr/decr pós fixado	`expr++ expr--`

Atribuição

- A atribuição é realizada com o operador '='
  - '=' serve apenas para atribuição – não pode ser usado em comparações (que usa '==')!
  - Copia o valor da variável ou constante do lado direito para a variável do lado esquerdo.

```
x = 13;  // copia a constante inteira 13 para x
y = x;   // copia o valor contido em x para y
```

- A atribuição copia valores
  - O valor armazenado em uma variável de tipo primitivo é o valor do número, caractere ou literal booleana (true ou false)
  - O valor armazenado em uma variável de tipo de classe (referência para objeto) é o ponteiro para o objeto ou null.
  - Conseqüentemente, copiar referências por atribuição não copia objetos mas apenas cria novas referências para o mesmo objeto!

Operadores Matemáticos

- `+` adição
- `-` subtração
- `*` multiplicação
- `/` divisão
- `%` módulo (resto)

- Operadores unários
  - `-n` e `+n` (ex: -23) (em uma expressão: 13 + -12)
  - Melhor usar parênteses: 13 + (-12)
- Atribuição com operação
  - +=, -=, *=, /=, %=
  - `x = x + 1` equivale a `x += 1`

Incremento e Decremento

- *Exemplo*

```
int a = 10;
int b = 5;
```

- *Incrementa ou decrementa antes de usar a variável*

```
int x = ++a; // a contém 11, x contém 11
int y = --b;   // b contém 4, y contém 4
```

  - *A atribuição foi feita DEPOIS!*

- *Incrementa ou decrementa depois de usar a variável*

```
int x = a++; // a contém 11, x contém 10
int y = b--;   // b contém 4, y contém 5
```

  - *A atribuição foi feita ANTES!*

Operadores Relacionais

- == *igual*
- != *diferente*
- < *menor*
- <= *menor ou igual*
- > *maior*
- >= *maior ou igual*

- *Sempre produzem um resultado booleano*
  - `true` *ou* `false`
  - *Comparam os valores de duas variáveis ou de uma variável e uma constante*
  - *Comparam as referências de objetos (apenas == e !=)*

Operadores Lógicos

- `&&` *E (and)*
- `||` *Ou (or)*
- `!` *Negação (not)*

- *Produzem sempre um valor booleano*
  - *true ou false*
  - *Argumentos precisam ser valores booleanos ou expressões com resultado booleano*
  - *Por exemplo:* `(3 > x) && !(y <= 10)`
- *Expressão será realizada até que o resultado possa ser determinado de forma não ambígua*
  - *"short-circuit"*
  - *Exemplo: (*`false && <qualquer coisa>`*)*
  - *A expressão <qualquer coisa> não será calculada*

Operadores orientados a bit

- `&` *and*
- `|` *or*
- `^` *xor (ou exclusivo)*
- `~` *not*

- *Para operações em baixo nível (bit por bit)*
  - *Operam com inteiros e resultados são números inteiros*
  - *Se argumentos forem booleanos, resultado será igual ao obtido com operadores booleanos, mas sem 'curto-circuito'*
  - *Suportam atribuição conjunta: &=, |= ou ^=*

Operadores de Deslocamento

- << *deslocamento de bit à esquerda (multiplicação por dois)*
- >> *deslocamento de bit à direita (divisão truncada por dois)*
- >>> *deslocamento à direita sem considerar sinal (acrescenta zeros)*
- *Para operações em baixo nível (bit a bit)*
  - *Operam sobre inteiros e inteiros longos*
  - *Tipos menores (short e byte) são convertidos a int antes de realizar operação*
  - *Podem ser combinados com atribuição: <<=, >>= ou >>>=*

Operador Ternário (*if-else*)

- *Retorna um valor ou outro dependendo do resultado de uma expressão booleana*
  - `variavel = expressão ? valor, se true : valor, se false;`
- *Exemplo:*

```
int x = (y != 0) ? 50 : 500;
String tit = (sex == 'f') ? "Sra." : "Sr
num + " pagina" + (num != 1) ? "s" : ""
```

- *Use com cuidado*
  - *Pode levar a código difícil de entender*

Operador de Concatenação

- *Em uma operação usando "+" com dois operandos, se um deles for String, o outro será convertido para String e ambos serão concatenados*
- *A operação de concatenação, assim como a de adição, ocorre da direita para a esquerda*

```
String s = 1 + 2 + 3 + "=" + 4 + 5 + 6;
```

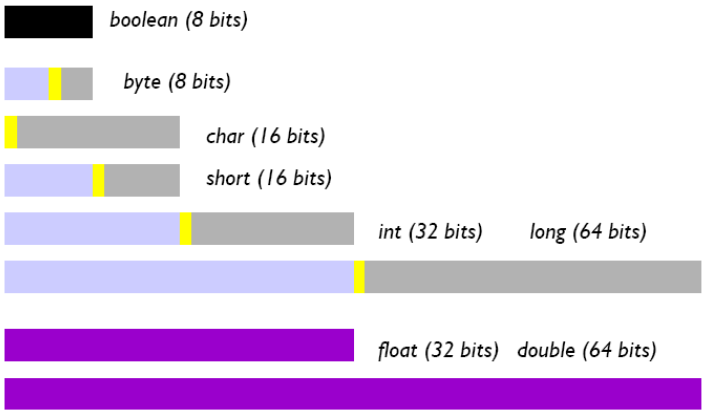
- *Resultado: s contém a String* `"6=456"`

instanceof

- *instanceof é um operador usado para comparar uma referência com uma classe*
  - *A expressão será true se a referência for do tipo de uma classe ou subclasse testada e false, caso contrário*
  - *Sintaxe: referência instanceof Classe*
- *Exemplo:*

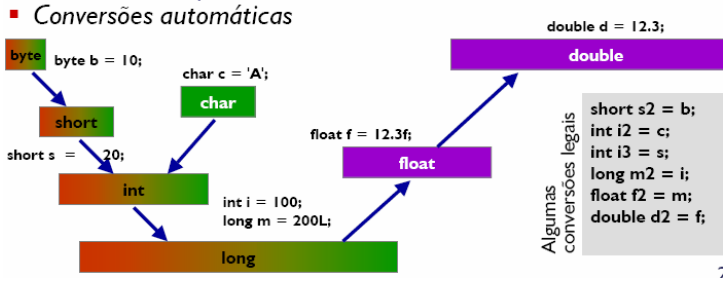
```
if (obj instanceof Point) {
  System.out.println("Descendente de Point");
}
```

Tipos de dados



Conversão de tipos primitivos

- *Java converterá um tipo de dados em outro sempre que isto for apropriado*
- *As conversões ocorrerão automaticamente quando houver garantia de não haver perda de informação*
  - *Tipos menores em tipos maiores*
  - *Tipos de menor precisão em tipos de maior precisão*
  - *Tnteiros em ponto-flutuante*
- *Conversões automáticas*



Conversão de referências

- *Pode-se atribuir uma referência A a uma outra referência B de um tipo diferente, desde que*
  - *B seja uma superclasse (direta ou indireta) de A: Qualquer referência pode ser atribuída a uma referência da classe Object*
  - *B seja uma interface implementada por A: mais detalhes sobre interfaces em aulas futuras*

```
class Carro extends Veiculo {...}

class Veiculo implements Dirigivel {}

class Porsche extends Carro {...}
```

Algumas conversões legais

```
Carro c = new Carro();
Veiculo v = new Carro();
Object o = new Carro();
Dirigivel d = new Carro();
Carro p = new Porsche();
```


Conversão de tipos primitivos

- A linguagem Java não suporta a coerção arbitrária dos tipos de variáveis, exceto no caso de concatenação de strings

- Exemplo
  - `long bigval = 6;  // 6 é um int, OK`
  - `int smallval = 99L; // 99 é um tipo long, ilegal`
  - `float x = 12.414F;  // 12.414F é um tipo float, OK`
  - `float y = 12.414;  // é um tipo double, ilegal`

Conversão automática

- Qualquer operação com dois ou mais operandos de tipos diferentes sofrerá **promoção** (conversão automática) ao tipo mais abrangente, que pode ser:
  - O maior ou mais preciso tipo da expressão (até double)
  - O tipo int (para tipos menores que int)

```
double d = 10 + 50L + 4.6;
    // tudo é promovido para double
```


Conversão automática

- As regras para conversão com tipos primitivos podem ser assim resumidas:
  - O tipo boolean **não** pode ser convertido para nenhum outro tipo
  - Os tipos não-booleanos podem ser convertidos para outros tipos não-booleanos, contanto que não haja perda de precisão. Isto é, podem ser convertidos apenas para tipos que suportem um limite igual ou maior ao seu

Coerção (*cast*)

- As variáveis podem ser convertidas em tipos maiores de maneira automática, mas não em tipos menores
  - Desta forma, uma expressão int pode ser tratada como long, mas uma expressão long **não poderá** ser tratada como int sem que haja uma coerção explicita
- Uma coerção é utilizada para persuadir o compilador a reconhecer uma atribuição
- Esse procedimento pode ser adotado, por exemplo, para "comprimir" um valor long em uma variável int
- Na coerção, o programador assume os riscos da conversão de dados

Coerção

- Não há risco de perda de informação na atribuição a seguir:
  - long L = 120;
    int i = L;
- Mas o compilador acusará erro porque um long não pode ser atribuído a um int
  - Solução: *int i = (int) L;*
- O operador de coerção tem **maior** precedência que os demais operadores

Coerção - Exemplo

```
public class Test {
  public static void main(String args[
     int i = 16777473;
     short s = (short) i;
     byte b = (byte) i;
     System.out.println("Valor int:" + i);
     System.out.println("Valor short:" + s);
     System.out.println("Valor byte:" + b);
  }
}
```

A execução de código provoca duas conversões. Em ambas ocorre perda de informação

```
i = 00000001 00000000 00000001 00000001
      (int (4 bytes)  - valor: 16777473)
s = 00000001 00000000 00000001 00000001
      (short (2 bytes) - valor: 257)
b = 00000001 00000000 00000001 00000001
      (byte (1 byte) - valor: 1)
```


Coerção – Exemplo com objetos

```
Array v = new Array( );
v.add("Hello");
String s = (String)v.get(0);
```

- Como o método **get** sempre retorna um elemento do tipo **Object**, que não pode ser atribuído a uma **String**, torna-se necessário fazer o **cast** antes da atribuição
- Caso fosse feita a atribuição direta, teríamos um erro de compilação:

```
String s = v.get(0);   // erro
```