DIN Departamento de Informática

UEM Universidade Estadual de Maringá

Java - Estruturas Fundamentais

Lilian Passos Scatalon
lpscatalon2@uem.br

Linguagem Java - Estruturas fundamentais

1. Tipos primitivos
2. Variáveis
3. Expressões e operadores
4. Strings
5. Entrada e Saída
6. Fluxo de controle

(1) Tipos primitivos

Inteiros

Números sem parte decimal

Table 1-1 Java Signed Integer Types

Type	Storage requirement	Range (inclusive)
`byte`	1 byte	−128 to 127
`short`	2 bytes	−32,768 to 32,767
`int`	4 bytes	−2,147,483,648 to 2,147,483,647 (just over 2 billion)
`long`	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

(1) Tipos primitivos

Ponto flutuante

Números com parte decimal

Table 1–2 Floating-Point Types

Type	Storage requirement	Range
`float`	4 bytes	Approximately ±3.40282347E+38F (6–7 significant decimal digits)
`double`	8 bytes	Approximately ±1.79769313486231570E+308 (15 significant decimal digits)

(1) Tipos primitivos

Caractere

Tipo `char`

Código Unicode, no formato UTF-16

Literais são delimitados por aspas simples

‘J’ é um literal com valor 74

Os caracteres especiais ‘\n’, ‘\t’, ‘\r’, ‘\b’ denotam nova linha, retorno de carro, tabulação e backspace

(1) Tipos primitivos

Booleano

O tipo `boolean` tem dois valores, `false` e `true`

Em Java, o tipo `boolean` não é um tipo numérico. Não há relação entre valores booleanos e os inteiros 0 e 1.

(2) Variáveis

Declaração

```
int total = 0;

int total = 0, count; // count é um inteiro não inicializado

Random generator = new Random();
```

(2) Variáveis

Nomes

Devem começar com uma letra e consistir de quaisquer letras, dígitos e os símbolos _ e $

Case sensitive: count e Count são nomes diferentes

Não é possível usar palavras-chave como nome, por exemplo double

Por convenção, adota-se o camel case, por exemplo countOfInvalidInputs

(2) Variáveis

Inicialização

É preciso inicializar as variáveis antes de usá-las

```
int count;
count++; // Erro
```

É uma boa prática declarar as variáveis quando o seu valor inicial está disponível

```
Scanner in = new Scanner(System.in);
System.out.println("Qual a sua idade?");
int idade = in.nextInt();
```

(2) Variáveis

Constantes

A palavra-chave final denota um valor que não pode ser alterado após a inicialização

```
final int DIAS_POR_SEMANA = 7;


final int DIAS_EM_FEVEREIRO;
if (anoBissexto) {
        DIAS_EM_FEVEREIRO = 29;
} else {
        DIAS_EM_FEVEREIRO = 28;
}
```

Por convenção, usa-se letras em caixa alta para nomes de constantes

(3) Expressões e operadores

Table 1-3 Java Operators

Operators	Associativity
[] . () (method call)	Left
! ~ ++ -- + (unary) - (unary) () (cast) new	Right
* / % (modulus)	Left
+ -	Left
<< >> >>> (arithmetic shift)	Left
< > <= >= instanceof	Left
== !=	Left
& (bitwise and)	Left
^ (bitwise exclusive or)	Left
| (bitwise or)	Left
&& (logical and)	Left
|| (logical or)	Left
? : (conditional)	Left
= += -= *= /= %= <<= >>= >>>= &= ^= |=	Right

(3) Expressões e operadores

Classe Math

Math.pow(x,y)

Math.sqrt(x)

Math.min(x,y)

Math.max(x,y)

Math.PI

(3) Expressões e operadores

Conversões de Tipo

Quando um operador combina operandos de tipos diferentes, os números são convertidos a um tipo comum

Em Java, a conversão é sempre legal se não há perda de informação

- De `byte` para `short`, `int`, `long`, `double`
- De `short` e `char` para `int`, `long`, `double`
- De `int` para `long`, `double`

As seguintes conversões são legais, mas podem perder informação

- De `int` para `float`
- De `long` para `float`, `double`

(3) Expressões e operadores

Conversões de Tipo

Para fazer uma conversão que não esteja entre as permitidas, usa-se o operador de coerção (*cast*)

```
double x = 3.75;
int n = (int) x;  //3
```

Nesse caso, outra opção é o arredondamento para o inteiro mais próximo

```
int n = (int) Math.round(x);  //4
```

(3) Expressões e operadores

Operadores relacionais e lógicos

`==` e `!=` testam igualdade

Também há os habituais operadores relacionais `<` (menor que), `>` (maior que), `<=` (menor ou igual a) e `>=` (maior ou igual a)

É possível combinar expressões do tipo boolean com os operadores lógicos `&&` (E lógico), `||` (OU lógico) e `!` (NÃO lógico)

(3) Expressões e operadores

Operadores relacionais e lógicos

Avaliação em curto-circuito

```
//curto-circuito com E lógico: a 1ª condição é falsa
n != 0 && s + (100 - s) / n < 50

//curto-circuito com OU lógico: a 1ª condição é verdadeira
n == 0 || s + (100 - s) / n >= 50
```

(3) Expressões e operadores

Operadores relacionais e lógicos

Operador condicional

Três operandos: uma condição e dois valores

```
time < 12 ? "am" : "pm"
```

O resultado é o 1º valor se a condição for verdadeira, caso contrário é o 2º valor

(4) Strings

Uma string em Java é uma sequência de caracteres Unicode

Strings literais são delimitadas por aspas duplas `"Java"`

String é uma classe

```
String local = "world";
```

(4) Strings

Concatenação

Operador +

```
String location = "Java";
String greeting = "Hello " + location;
```

Ao concatenar uma string com um valor numérico, o valor é convertido para uma string

```
int idade = 42;
String saida = idade + " anos";
```

(4) Strings

Substrings

Para fatiar strings, usa-se o método substring

```
String greeting = "Hello, World!";
String location = greeting.substring(7, 12); // "World"
```

Os parâmetros são as posições inicial e final (não inclusa) da substring a ser extraída

(4) Strings

Comparação

Para verificar se duas strings são iguais, usa-se o método equals

```
location.equals("World")
```

Não se usa o operador == para comparar strings, pois retornaria verdadeiro apenas se as strings comparadas fossem o mesmo objeto na memória

(4) Strings

Comparação (cont.)

Como qualquer objeto, uma variável String pode ser null

```
String middleName = null;

if (middleName == null)
```

Note que null não é o mesmo que uma string vazia “”

(4) Strings

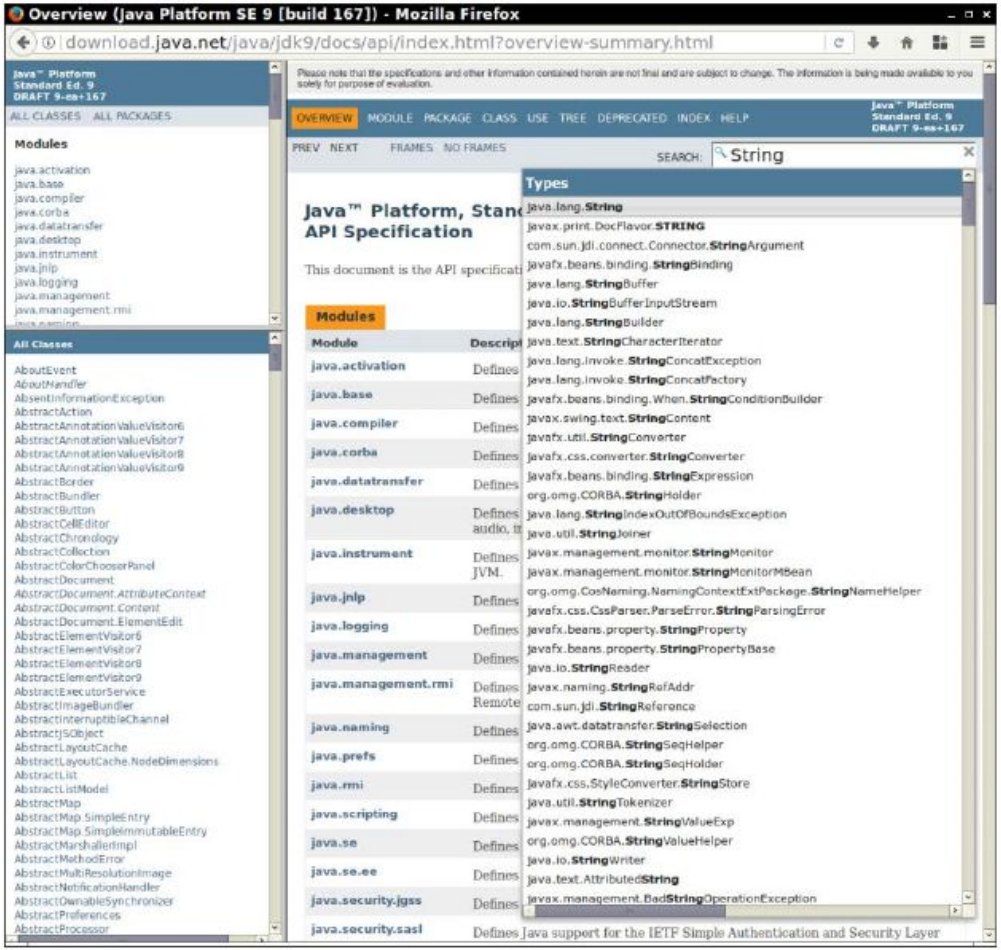
Convertendo entre números e strings (exemplo com inteiro)

Método toString()

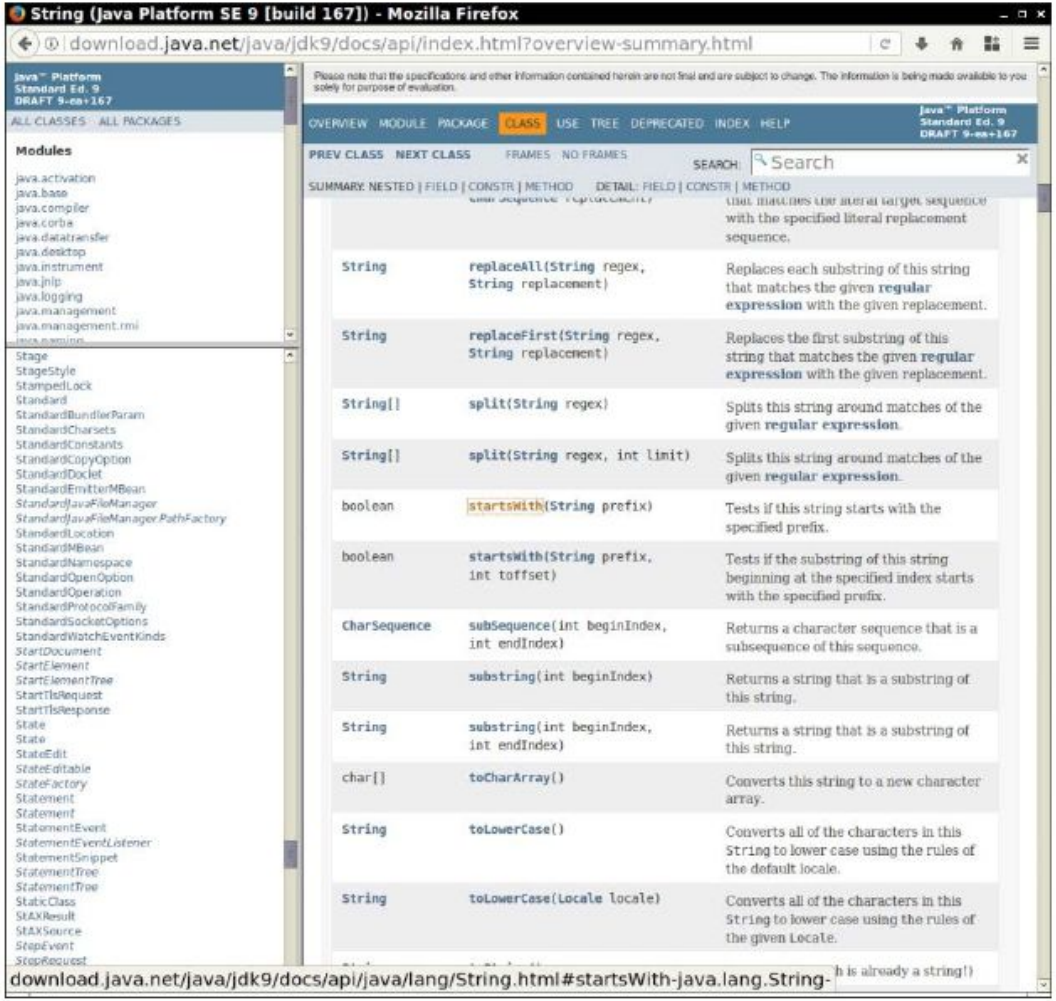
```
int n = 42;
String str = Integer.toString(n); // "42"
```

Método parseInt()

```
String str = "101010";
int n = Integer.parseInt(str); // Altera n para 101010
```



https://docs.oracle.com/en/java/javase/17/docs/api/index.html


String (Java Platform SE 9 [build 167]) - Mozilla Firefox
download.java.net/java/jdk9/docs/api/index.html?overview-summary.html
Java™ Platform
Standard Ed. 9
DRAFT 9-ea+167
ALL CLASSES ALL PACKAGES
Modules
java.activation
java.base
java.compiler
java.corba
java.datatransfer
java.desktop
java.instrument
java.jnlp
java.logging
java.management
java.management.rmi
Stage
StageStyle
StampedLock
Standard
StandardBundleParam
StandardCharsets
StandardConstants
StandardCopyOption
StandardDoclet
StandardEmitterMBean
StandardJavaFileManager
StandardJavaFileManagerPathFactory
StandardLocation
StandardMBean
StandardNamespace
StandardOpenOption
StandardOperation
StandardProtocolFamily
StandardSocketOptions
StandardWatchEventKinds
StartDocument
StartElement
StartElementTree
StartTlsRequest
StartTlsResponse
State
State
StateEdit
StateEditable
StateFactory
Statement
Statement
StatementEvent
StatementEventListener
StatementSnippet
StatementTree
StatementTree
StaticClass
StAXResult
StAXSource
StepEvent
StepRequest
Please note that the specifications and other information contained herein are not final and are subject to change. The information is being made available to you solely for purpose of evaluation.
OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
Java™ Platform
Standard Ed. 9
DRAFT 9-ea+167
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SEARCH: Search
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
with the specified literal replacement sequence.
String replaceAll(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.
String replaceFirst(String regex, String replacement) Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[] split(String regex) Splits this string around matches of the given regular expression.
String[] split(String regex, int limit) Splits this string around matches of the given regular expression.
boolean startsWith(String prefix) Tests if this string starts with the specified prefix.
boolean startsWith(String prefix, int toffset) Tests if the substring of this string beginning at the specified index starts with the specified prefix.
CharSequence subSequence(int beginIndex, int endIndex) Returns a character sequence that is a subsequence of this sequence.
String substring(int beginIndex) Returns a string that is a substring of this string.
String substring(int beginIndex, int endIndex) Returns a string that is a substring of this string.
char[] toCharArray() Converts this string to a new character array.
String toLowerCase() Converts all of the characters in this String to lower case using the rules of the default locale.
String toLowerCase(Locale locale) Converts all of the characters in this String to lower case using the rules of the given Locale.
h is already a string!)
download.java.net/java/jdk9/docs/api/java/lang/String.html#startsWith-java.lang.String-

(5) Entrada e Saída

Lendo Entrada

System.in é o stream de entrada padrão (com métodos para ler bytes)

```
import java.util.Scanner;

//...

Scanner in = new Scanner(System.in);
System.out.println("Qual é o seu nome?");
String nome = in.nextLine();          // nextLine() lê uma linha

String primeiroNome = in.next();      // next() lê uma única palavra (delimitada por espaço)

System.out.println("Qual é a sua idade?");
int idade = in.nextInt();             // nextInt() lê um inteiro
```

(5) Entrada e Saída

Saída Formatada

Método println() começa uma nova linha após imprimir a string

Método print() apenas imprime a string e mantém o cursor no lugar

```
System.out.print("Sua idade: "); // Não é println
int idade = in.nextInt();
```

(5) Entrada e Saída

Saída Formatada

Método println() começa uma nova linha após imprimir a string

Método print() apenas imprime a string e mantém o cursor no lugar

```
System.out.print("Sua idade: "); // Não é println
int idade = in.nextInt();
```

(5) Entrada e Saída

Saída Formatada

Método printf() permite controlar o formato dos números e fornecer múltiplos argumentos

```
System.out.print(1000.0 / 3.0);            // 333.3333333333333
System.out.printf("%8.2f", 1000.0 / 3.0);  // 333.33
System.out.printf("Olá, %s. No ano que vem você terá %d anos.\n", nome, idade+1);
```

Cada especificador de formato que começa com % é substituído pelo respectivo argumento

Table 1–5 Conversion Characters for Formatted Output

Conversion Character	Purpose	Example
`d`	Decimal integer	`159`
`x` or `X`	Hexadecimal integer	`9f` or `9F`
`o`	Octal integer	`237`
`f`	Fixed floating-point	`15.9`
`e` or `E`	Exponential floating-point	`1.59e+01` or `1.59E+01`
`g` or `G`	General floating-point: `e`/`E` if the exponent is greater than the precision or < −4, `f`/`F` otherwise	`15.9000` at the default precision of 6, `2e+01` at precision 1
`a` or `A`	Hexadecimal floating-point	`0x1.fccdp3` or `0X1.FCCDP3`
`s` or `S`	String	`Java` or `JAVA`
`c` or `C`	Character	`j` or `J`
`b` or `B`	`boolean`	`false` or `FALSE`
`h` or `H`	Hash code (see Chapter 4)	`42628b2` or `42628B2`
`t` or `T`	Date and time (obsolete; see Chapter 12 instead)	—
`%`	The percent symbol	`%`
`n`	The platform-dependent line separator	—

Table 1-6 Flags for Formatted Output

Flag	Purpose	Example
`+`	Prints sign for positive and negative numbers	`+3333.33`
space	Adds a space before positive numbers	`_3333.33`
`-`	Left-justifies field	`3333.33___`
`0`	Adds leading zeroes	`003333.33`
`(`	Encloses negative values in parentheses	`(3333.33)`
`,`	Uses group separators	`3,333.33`
`#` (for `f` format)	Always includes a decimal point	`3333.`
`#` (for `x` or `o` format)	Adds `0x` or `0` prefix	`0xcafe`
`$`	Specifies the index of the argument to be formatted; for example, `%1$d %1$x` prints the first argument in decimal and hexadecimal.	`159 9f`
`<`	Formats the same value as the previous specification; for example, `%d %<x` prints the same number in decimal and hexadecimal.	`159 9f`

(6) Fluxo de controle

Desvios

```
if (count > 0) {
        double average = sum / count;
        System.out.println(average);
}
```

```
if (count > 0) {
        double average = sum / count;
        System.out.println(average);
} else {
        System.out.println(0);
}
```

(6) Fluxo de controle

Desvios

```
switch (count){
    case 0:
          output = "None";
          break;
    case 1:
          output = "One";
          break;
  case 2:
  case 3:
  case 4:
  case 5:
          output = Integer.toString(count);
          break;
  default:
          output = "Many";
          break;
}
```

(6) Fluxo de controle

Loops

```
while (sum < target) {
        int next = generator.nextInt(10);
        sum += next;
        count++;
}
```

```
int next;
do {
        next = generator.nextInt(10);
        count++;
} while (next != target);
```

```
for (int i = 1; i <= 20; i++) {
        int next = generator.nextInt(10);
        sum += next;
}
```

(6) Fluxo de controle

Break e continue

```
while (true) {
        String input = in.next();
        if (input.equals("Q")) break; // Sai do loop
        // Processa entrada
}
// break desvia para cá

while (in.hasNextInt()) {
        int input = in.nextInt();
        if (input < 0) continue; // Desvia para o teste de in.hasNextInt()
        // Processa entrada
}

for (int i = 1; i <= target; i++) {
        int input = in.nextInt();
        if (n < 0) continue; // Desvia para i++
        // Processa entrada
}
```

(6) Fluxo de controle

Escopo

Uma variável local é aquela declarada em um método, incluindo seus parâmetros

O escopo de uma variável é a parte do programa em que é possível acessá-la

Inicia na declaração e se estende até o fim do bloco atual

(6) Fluxo de controle

Escopo

```
while (...) {
        System.out.println(...);
        String input = in.next(); // Escopo de input começa aqui
        ...
        // Escopo de input termina aqui
}
```

Inicia na declaração e se estende até o fim do bloco atual

(6) Fluxo de controle

Escopo

```
for (int i = 0; i < n; i++) { // i está no escopo para o teste e update
        ...
}
// i não está definido aqui


int i;
for (i = 0; !found && i < n; i++) {
        ...
}
// i ainda disponível
```

Bibliografia

H. Schildt. **Java: A Beginner's Guide**. McGraw-Hill Education, 8th edition, 2018.

C. S. Horstmann. **Core Java SE 9 for the Impatient**. Addison-Wesley, 2nd Edition, 2018.

Caelum. Apostila **Java e Orientação a Objetos**. Disponível em https://www.alura.com.br/apostila-java-orientacao-objetos