

A Linguagem Java

A LINGUAGEM JAVA

- Instrução em Java é composta de uma ou mais linhas terminadas por ponto-e-vírgula

- Simples:

```
System.out.println("Hello world");
```

- Bloco:

```
{
```

```
    System.out.print("Hello ");
```

```
    System.out.println("world");
```

```
}
```

A LINGUAGEM JAVA

- Identificadores:
 - devem começar com
 - uma letra,
 - um underscore “_”,
 - ou um sinal de cifrão “\$”.
 - As letras podem estar tanto em maiúsculo quanto em minúsculo.
 - Caracteres subsequentes podem usar números de 0 a 9.

A LINGUAGEM JAVA

■ Identificadores:

- representações de nomes de variáveis, métodos, classes, etc
- case-sensitive
 - Hello diferente de hello.
- Não podem ter nomes iguais às palavras-chave ou palavras reservadas do Java, como: class, public, void, int, etc

A LINGUAGEM JAVA

■ Palavras-chave:

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

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0

Estrutura básica de classes em Java

```
/**
 * A menor classe em Java não precisa de
 * variáveis, nem de operações, embora possa ser
 * compilada.
 */

class MenorClasseEmJava {

    /* Corpo da classe, onde se definem variáveis e
       operações
    */

} //Fim da declaração da classe 'MenorClasseEmJava'
```

Estrutura básica de classes em Java

- Uma classe é declarada com a palavra-reservada **class** seguida do nome da classe e de seu corpo entre chaves

```
class NomeDaClasse { ... }
```

- Caracteres maiúsculos e minúsculos são diferentes
 - As palavras Class, ClAss, ClaSs e class são distintas
 - Somente a última opção pode ser usada para declarar uma classe
 - As outras provocam erros em tempo de compilação

Estrutura básica de classes em Java

- O corpo da classe é delimitado pelos caracteres abre chave **{** e fecha chave **}**

```
/**
 * A menor classe em Java não precisa de
 * variáveis, nem de operações, embora possa ser
 * compilada.
 */

class MenorClasseEmJava {

    /* Corpo da classe, onde se definem variáveis e
     * operações
     */

} //Fim da declaração da classe 'MenorClasseEmJava'
```

Nome da classe

Início do corpo da classe

Fim do corpo da classe

Corpo da classe

Entendimento e a modificação de programas

A LINGUAGEM JAVA

■ Comentários

```
/**
```

```
 * A menor classe em Java não precisa de  
 * variáveis, nem de operações, embora possa ser  
 * compilada.
```

Início do bloco de comentário

```
*/
```

Fim do bloco de comentário

```
class MenorClasseEmJava {
```

```
    /* Corpo da classe, onde se definem variáveis e  
       operações */
```

```
} // Fim da declaração da classe 'MenorClasseEmJava'
```

Início do comentário de linha única

■ `/** documentacao */`

indica um comentário para documentação. Utilizado pela ferramenta javadoc

A LINGUAGEM JAVA

■ Exemplo de classe executável diretamente

■ Aplicação Java

```
/**
 * A classe 'AloMundo' implementa uma aplicação
 * Java que simplesmente imprime na saída
 * padrão (console) o texto 'Alô Mundo!'
 */

class AloMundo {
    public static void main(String[] args) {
        System.out.println("Alo Mundo!"); //Mostra o texto entre aspas.
    }
}
```

Cabeçalho da operação especial main

Início do corpo da operação

Início do corpo da operação

Declaração de operações

- Operações executam ações sobre o estado de um objeto
 - São também denominadas de métodos em Java
 - Nomes de operações seguem as mesmas regras de nomes de classes
 - Métodos (operações) não podem ser criados dentro de outras operações, nem fora do corpo da classe à que pertencem
 - Não podem existir métodos isolados!

Declaração de operações

■ Estilo

– Nomes de operações refletem ações que são

- O tipo **void** indica que a operação não retorna nenhum valor quando executada
- Ela apenas consulta ou modifica o estado do objeto!
- **Void**, em inglês significa "vazio", "nulo"
- Logo, o tipo **void** é um tipo que não tem nenhum valor, sendo formado pelo conjunto vazio

```
/**  
 * A classe  
 * controlar  
 */
```

```
class Lampa  
/**  
 * Variáv  
 * se ela  
 */
```

boolean

tipo de retorno da operação

```
/**  
 * A oper  
 * altera  
 */
```

void

Nome da operação

Lista de parâmetros da operação
() indica lista vazia.

```
apaga  
aceso = false; // Apaga a lâmpada  
}
```

Corpo da operação

Declaração de operação

Declaração de operações

- Neste caso, a operação deve ter em seu corpo a palavra-chave **return** seguida de um valor ou variável com mesmo tipo que o tipo de retorno da operação
- No caso específico da operação 'estaApagada', o tipo do valor de retorno é **boolean**
- A execução do **return** resulta no encerramento da execução da operação
- Operações com tipo de retorno **void** não precisam de um **return**, mas podem usá-lo para encerrar arbitrariamente a execução a sua execução
 - No caso, o **return** não precisa ser seguido de valor ou variável, pois não retorna nada, apenas encerra a operação

```
...  
/**  
 * A op  
 * alte  
 */  
void ac  
  aceso  
}  
  
/**  
 * A op  
 * se a  
 * o va  
 * a op  
 */  
boolean  
  if (a  
    ret  
  }  
  else  
    return false;  
  }  
}  
  
} //Fim da classe lâmpada
```

ue'

Referências e criação de objetos

■ Referência

- Variável cujo conteúdo (valor) é um endereço de uma posição de memória onde reside um objeto, ou então o valor nulo – **null** em Java

```
Lampada lamp1;
```

- Quando declarada sem inicialização, o seu valor default é nulo, isto é, não está apontando – referenciando – nenhum objeto
- A declaração de referência anterior é equivalente a seguinte declaração

```
Lampada lamp1 = null;
```

- Qualquer tentativa de uso de uma referência apontando para **null** resulta num erro
 - Antes, deve-se inicializar a referência com o endereço de um objeto existente

Referências e criação de objetos

- Criar
- C
- u
- Agora, tanto `lamp1`, quanto `lamp2` estão apontando (referenciando) para um mesmo objeto, na posição 2 de memória
- O objeto na posição 2 também pode ser manipulado através da referência `lamp2`

```
Lampada lamp1 = null;
```

```
lamp1 = new Lampada();
```

```
lamp1.acende();
```

```
Lampada lamp2 = lamp1;
```

```
lamp2.apaga();
```

```
lamp2 = new Lampada();
```

```
lamp2.acende();
```

Memória

1	lamp1	2
2	aceso	false
3	lamp2	4
4	aceso	true

Int ...

■ U
r
p

```
lamp1.acende();           //Acende a lâmpada 1
System.out.print("lamp1 esta apagada? ");
System.out.println(lamp1.estaApagada());
System.out.println();

lamp1.apaga();            //Apaga a lâmpada 1
System.out.print("lamp1 esta apagada? ");
System.out.println(lamp1.estaApagada());
System.out.println();

lamp2.apaga();            //Apaga a lâmpada 2
System.out.print("lamp2 esta apagada? ");
System.out.println(lamp2.estaApagada());
System.out.println();

}

} //Fim da classe 'CriadorDeLampadas'
```

sse

de-se

Modificadores de acesso

- Permitem que se controle a visibilidade externa das variáveis e operações de um objeto

- Tipos

- ▶ **public**

- Uma variável ou operação declarada como pública pode ser acessada no ambiente externo à classe onde está declarada

- ▶ public int saldo = 0;

- ▶ public int retornaSaldo();

- ▶ **private**

- Uma variável ou operação declarada como privada **não** pode ser acessada no ambiente externo à classe onde está declarada, ou seja, só pode ser acessada dentro da própria classe

- ▶ private int saldo = 0;

- ▶ private int retornaSaldo();

Modificadores de acesso

■ Tipos

▶ **protected**

- ▶ Uma variável ou operação declarada como protegida pode ser acessada no ambiente externo à classe onde está declarada, desde que o acesso seja feito a partir de uma classe declarada no mesmo diretório que esta ou quando herdada por outra classe

- ▶ `protected int saldo = 0;`

- ▶ `protected int retornaSaldo();`

▶ **default (quando omitido, isto é, não especificado)**

- ▶ Uma variável ou operação declarada sem modificador de acesso (ou seja, o modificador de acesso default) pode ser acessada no ambiente externo à classe onde está declarada, desde que o acesso seja feito a partir de uma classe declarada no mesmo diretório que esta

- ▶ `int saldo = 0;`

- ▶ `int retornaSaldo();`

Exemplo

- Considere o seguinte problema: um cartão de crédito simplificado
 - Representar um cartão de crédito que mantém informações sobre o saldo do cartão, seu limite máximo e o total de pontos de bônus, que poderão ser trocados posteriormente por prêmios
 - No cartão poderão ser realizadas operações de compra (aumento do saldo) e de pagamento do cartão (dedução no saldo)
 - O limite do cartão é fixado em 10.000
 - A cada compra realizada, o valor da compra é adicionado ao saldo, desde que não ultrapasse o limite do cartão

Exemplo

- Considere o seguinte problema: um cartão de crédito simplificado
 - Cada real de compras vale um ponto de bônus, sendo portanto, o valor comprado adicionado ao total de pontos de bônus
 - A cada pagamento realizado, o valor do pagamento é deduzido do saldo.
 - Cada real pago vale um ponto de bônus, sendo portanto, o valor pago adicionado ao total de pontos de bônus
 - O total de bônus é limitado ao limite de crédito, ou seja, 10.000 pontos

Exemplo

- Considere o seguinte problema: um cartão de crédito simplificado
 - Deve-se ter operações para retornar o saldo do cartão e o total de bônus
 - Os valores do saldo e o total de pontos de bônus somente podem ser alterados através das operações para compra e pagamento
 - Nenhum usuário de objetos da classe “CartaoDeCredito” pode alterar diretamente o saldo ou o total de pontos do bônus

Exemplo

- Cartão de crédito simplificado – propriedade essenciais
 - Nome da classe: CartaoDeCredito
 - Estado
 - Três variáveis do tipo inteiro para representar o saldo, o limite e o total de pontos de bônus do cartão
 - `private int saldo = 0;`
 - `private int limite = 10000;`
 - `private int bonus = 0;`
 - São privadas porque somente podem ser acessadas/modificadas dentro da classe, através de suas operações, que respeitam as regras de limite de crédito e de pontuação de bônus
 - Caso contrário, alguém, que não a classe, poderia modificar diretamente o valor do saldo, do limite ou do bônus para cometer uma fraude, por exemplo.

Exemplo

■ Cartão de crédito simplificado – propriedade essenciais

■ Comportamento

- Determinado pela execução das seguintes operações – apenas os cabeçalhos são mostrados
 - `public void compra(int valor)`
 - `public void paga(int valor)`
 - `public int retornaSaldo()`
 - `public int retornaBonus()`
- e pela operação auxiliar abaixo
 - `private void creditaBonus(int valor)`

Exemplo

- Cartão de crédito simplificado
 - As operações que foram declaradas como públicas respeitam as regras estipuladas para o cartão de crédito. Mais do que isso, devem ser visíveis, isto é, poder ser acessada externamente à classe para que possamos dispor de sua funcionalidade essencial
 - A operação `creditaBonus` foi definida como privada porque é utilizada internamente na classe, para auxiliar na implementação das operações compra e paga
 - Caso fosse definida como pública, seria possível alterar o valor do total de bonus sem precisar efetuar uma compra ou um pagamento, violando, portanto, as regras especificadas para o cartão de crédito

Exemplo

- Contador de crédito simplificado
- Tal tentativa de acesso provoca um erro em tempo de compilação porque a variável `bonus` foi declarada como privada (`private`), só podendo ser acessada, portanto, dentro da classe onde foi declarada ou seja a classe

Recomendação

- Toda variável ou operação de uma classe que não precise ser pública deve ser declarada como privada
 - O objetivo é evitar a violação das regras especificadas para a classe

350
10000
1150

3	
4	
	25

Tipos de Valores Primitivos

Tipos de valores primitivos

- Definição de tipo: conjunto de valores e se caracteriza por possuir operações associadas que atuam uniformemente sobre estes valores
 - **boolean** = {**false**, **true**}
 - **int** = {-2147483648,..., -2, -1, 0, +1, +2, ..., +2147483647}
- Modalidades de tipo de valores
 - Tipo primitivo (ou elementar, ou básico)
 - É aquele cujos valores são atômicos, i. e., não podem ser decompostos em valores mais simples
 - Exemplos: booleanos, inteiros, ponto flutuante
 - Tipo composto
 - É aquele cujos valores são compostos de valores mais simples
 - Exemplos: arrays, classes

Tipos de valores primitivos

- Predefinidos na linguagem
- Não são instâncias de classes
- **boolean** = {**true**, **false**}
 - Representa valores lógicos (booleanos), especialmente para indicar um estado (verdadeiro ou falso) em expressões lógicas ou condicionais
- **char** = {..., '0', '1', '2', ..., 'a', 'b', 'c', ...}
 - Caracteres armazenados em 16 bits (até 65535 caracteres), com suporte multilingüe (padrão Unicode)
 - Caracteres literais são expressos entre aspas simples
 - 'A' 'b' ',' '~' 'á' 'o' 'U' '@'
 - Cada caractere é representado internamente por um número inteiro, expresso na notação hexadecimal
 - A letra 'A' é representada por '\u0041'
 - A letra 'a' é representada por '\u0061'

Tipos de valores primitivos

- **byte** = $\{-128, \dots, -2, -1, 0, 1, 2, \dots 127\}$
 - Denota valores inteiros com 8 bits de precisão (um byte), com sinal, variando de -128 a 127, inclusive
- **short** = $\{-32768, \dots, -2, -1, 0, 1, 2, \dots 32767\}$
 - Denota valores inteiros com 16 bits de precisão (dois bytes), com sinal, variando de -32768 a 32767, inclusive
- **int** = $\{-2147483648, \dots, -2, -1, 0, 1, 2, \dots 2147483647\}$
 - Denota valores inteiros com 32 bits de precisão (quatro bytes), com sinal, variando de -2147483648 a 2147483647.

Tipos de valores primitivos

- **long** = $\{-9223372036854775808, \dots, -2, -1, 0, 1, 2, \dots, 9223372036854775807\}$
 - Denota valores inteiros com 64 bits de precisão (oito bytes), com sinal, variando de -9223372036854775808 a 9223372036854775807 .
- Valores inteiros literais são expressos da forma usual
 - 1 10 76 99 102
 - Porém, literais são sempre considerados do tipo int
- Inteiros do tipo long podem ser expressos como literais acrescentando-se L ou l no final do valor
 - 1l 10L 76l 99l 102L

Tipos de valores primitivos

- **float** = {..., 1.401298464324817E-45, ...
3.4028234663852886E38}
 - Denota valores numéricos de ponto flutuante, com precisão simples (quatro bytes), com sinal, onde 1.401298464324817E-45 é o menor número positivo diferente de zero e 3.4028234663852886E38 é o maior número positivo finito
- **double** = {..., 4.9E-324, ...
1.7976931348623157E308}
 - Denota valores numéricos de ponto flutuante, com precisão dupla (oito bytes), com sinal, onde 4.9E-324 é o menor número positivo diferente de zero e 1.7976931348623157E308 é o maior número

Tipos de valores primitivos

■ float e double

- Valores são representados da seguinte forma
 - $\text{valor} = \text{mantissa} \times 10^{\text{expoente}}$
- Valores de ponto flutuante são expressos da seguinte forma
 - $\langle \text{mantissa} \rangle e[+|-]\langle \text{expoente} \rangle$ ou apenas a parte da mantissa para valores de pequena grandeza
 - 2.998e+6 1.987E-23 -1.223e2 -3747.2820221 2737372.89
- Porém, literais são sempre considerados do tipo double
- Pontos flutuantes do tipo float podem ser expressos como literais acrescentando-se F ou f no final do valor
 - 2.998e+6F -3747.2820221f 1.987E-23f 2737372.89F

A classe String

- **String** = {"", "a", "b", ..., "aa", "ab", ..., "aaa", "aab", ..., "gustavo", ..., "java", ...}
- Usada para descrever sequências arbitrárias de caracteres (strings) de tamanho finito, limitado à memória disponível
- String literais são expressos entre aspas duplas
 - "" "Silvio" "cliente do banco"
 - "" denota o string vazio, isto é, o string que não possui caracteres. Não confundir com " ", que é o string com o caractere de espaço em branco
- A principal operação com instâncias da classe String é a concatenação, realizada com o operador +
 - "java" + "cliente do banco" = "javacliente do banco"
- Outras operações com strings serão vistas mais a frente

Variáveis

- espaço na memória usado para armazenar o estado de um objeto
- deve ter um nome e um tipo
- tipo da variável indica o tipo de dado que ela pode conter
- nome das variáveis deve seguir as mesmas regras de nomenclatura que os identificadores.

Variáveis

■ Definição

- Uma variável é uma entidade que armazena um valor, que pode ser inspecionado ou modificado sempre que desejado

■ Declaração

- Deve-se indicar o tipo da variável seguido do nome e de uma cláusula de inicialização, que é opcional

```
String nomeDoAluno = "";  
int matriculaAluno;  
boolean éBolsista = false;  
double media = 0.0;  
Lampada lamp = new Lampada();
```

- Embora opcional, jamais deixe de inicializar uma variável quando de sua declaração

Declaração de variáveis

- `<tipo do dado> <nome> [= valor inicial];`
`int i = 10;`
`char c;`
`c = 'X';`
- Exibindo o valor de uma Variável
 - `System.out.println(value);`
 - `System.out.print("O valor de i é ");`
 - `System.out.println(i);`
 - `System.out.print("O valor de i é "+ i);`
- Variáveis finais (conteúdo não pode ser alterado)
 - `final float pi = 3.14159;`
(Vale também para métodos)

Declaração de variáveis

- Pode-se declarar múltiplas variáveis de um só tipo de uma única vez
 - `String nomeDoAluno = "", nomeDoCurso = "";`
 - `double nota1 = 0.0, nota2 = 0.0, nota3 = 0.0, media = 0.0;`
- Regras para nomes de variáveis
 - Não podem conter espaços
 - Deve ser iniciado por uma letra ou `'_'` ou `'$'`
 - Pode-se utilizar acentos sem restrições
 - Pode conter números
 - Não pode ser uma palavra reservada da linguagem Java
- Estilo para nomes de variáveis
 - Devem iniciar com caracteres minúsculos, alternando com maiúsculos entre cada palavra ou então com o caractere `'_'`
 - Devem ser descritivas e expressivas

Operadores aritméticos

■ Operações sobre valores numéricos em Java (operadores aritméticos)

- + (unário): modifica o sinal de um valor numérico para positivo
 - +10 denota que +10 é um valor positivo
- - (unário): modifica o sinal de um valor numérico para negativo
 - -10 denota que -10 é um valor negativo
- * (binário): multiplicação de dois números
 - $10 * -5 = -50$ $10 * 5.5 = 55.0$ ————— **Tipo double**
- / (binário): divisão de dois números
 - $9 / 4 = 2$ **Tipo int** $-9.0 / -4 = 2.25$
- % (binário): módulo (resto) da divisão - $(a / b) * b + (a \% b) = a$
 - $9 \% 5 = 4$ $-9 \% 5.0 = -4.0$ $9 \% -5.0 = 4.0$

Tipos de valores primitivos

- Quando valores numéricos de dois tipos diferentes são combinados numa expressão, o tipo resultante é aquele **mais abrangente**, ou seja, o tipo mais específico é convertido implicitamente para o tipo mais geral
 - byte converte para short, char, int, long, float, double
 - short converte para int, long, float, double
 - char converte para int, long, float, double
 - int converte para long, float, double
 - long converte para float, double
 - float converte para double
- As conversões de um tipo mais geral para um tipo mais específico podem acarretar perdas de valor e devem ser explicitadas pelo programador

Tipos de valores primitivos

- Para dividir um valor do tipo `int` por um valor do tipo `double`, considerando o valor do tipo `double` como sendo `int`, deve-se fazer da seguinte forma
 - $9 / (\text{int}) 4.6 = 2$
- Quando se prefixa um valor com o nome de um tipo entre parênteses, está-se convertendo o tipo do valor à direita para o tipo explicitado
- o `double 4.6` foi convertido para o `int 4` antes da operação de divisão, daí o resultado ser 2 e não 1, ou seja, houve perda
- Esta operação de conversão explícita se denomina **Type casting**

Recapitulando...

■ Tipos de dados

Representação	Tipo de Dado	Dado Primitivo
lógico	Boolean	boolean
inteiro	Integer e Character	char, byte, short e int
inteiro longo	Integer	long
número fracionário	Float-point	float e double

Recapitulando...

- Lógico: falso ou verdadeiro

`boolean resultado = true;`

- Inteiro: pode ser representado de 5 formas:

- decimal, octal, hexadecimal, ASCII e Unicode

`2` // valor 2 em decimal

`077` // 0 indica que ele está representado em octal.

`0xBACC` // 0x indica que ele está representado em hexadecimal.

`'a'` // representação ASCII

`'\u0061'` // representação Unicode

- char é um inteiro especial, exclusivamente positivo e representa um único Unicode

`char c = 97;` // representa o símbolo 'a'

`byte b = 'a';` // em inteiro representa o número 97

- qualquer operação efetuada entre eles terá sempre como resultado um tipo int

`byte b1 = 1;`

`byte b2 = 2;`

`int resultado = b1 + b2;`

Recapitulando...

- **Inteiro Longo:**
 - Os inteiros têm por padrão o valor representado pelo tipo primitivo int.
 - Pode-se representá-los como long adicionando, ao final do número, um "l" ou "L".
- Os tipos de dados inteiros assumem valores nas seguintes faixas:

<i>Tamanho em memória</i>	<i>Dado primitivo</i>	<i>Faixa</i>
8 bits	byte	-2^7 até 2^7-1
16 bits	char	0 até $2^{16}-1$
16 bits	short	-2^{15} até $2^{15}-1$
32 bits	int	-2^{31} até $2^{31}-1$
64 bits	long	-2^{63} até $2^{63}-1$

Recapitulando...

■ Números fracionários:

- tipo ponto-flutuante possuem o valor **double** como padrão
- possuem um ponto decimal ou um dos seguintes caracteres:
 - E ou e // expoente
 - F ou f // float
 - D ou d // double

■ Exemplos,

- 3.14 // tipo double
- 6.02E23 // double com expoente
- 2.718F // float
- 123.4E+306D // double

■ Faixa de Valores:

<i>Tamanho em memória</i>	<i>Dado primitivo</i>	<i>Faixa</i>
32 bits	float	-10^{38} até $10^{38}-1$
64 bits	double	-10^{308} até $10^{308}-1$

Variáveis e expressões

Endereçamento

Endereçamento de variáveis

- As variáveis de “valor”, ou primitivas, são aquelas que armazenam dados no exato espaço de memória onde a variável está.
- As variáveis de “referência” são aquelas que armazenam o endereço de memória onde o dado está armazenado
 - Ex: `int num = 10;`
 `String nome = “Hello”;`

Endereço de memória	Nome da variável	Dado
1001	num	10
:		:
1563	nome	Endereço (2000)
:		:
:		:
2000		"Hello"

Variáveis

■ Comando de atribuição

- Permite modificar o valor armazenado na memória por uma variável
- Denotado pelo operador binário = usado da seguinte forma
variável = expressão, onde
- variável denota uma referência para a posição de memória onde o valor da variável a ser modificado é armazenado
- expressão denota uma expressão que, quando avaliada, resulta no valor a ser armazenado na variável

Memória

```
int contador = 1;  
contador = contador + 1;
```

De me re o
Denota o valor armazenado, no caso 1, e o valor
ou seja, neste contexto, a variável **modificando**
contador é usada para **inspecionar** a
memória. Resulta no valor 2, após avaliação

1	contador	2
2		
3		

Escopo das variáveis

- O escopo de uma variável corresponde ao bloco do programa (ou parte dele apenas) onde a variável foi declarada

```
class Triangulo {  
    float lado1;  
  
    boolean éEquilátero()  
    {  
        boolean igualdade12 = (lado1 == lado2);  
        boolean igualdade23 = (lado2 == lado3);  
        if (igualdade12 && igualdade23)  
            resultado = true;  
        else  
            resultado = false;  
        return resultado;  
    }  
  
    float calculaPerímetro()  
    {  
        float resultado = 0;  
        return resultado;  
    }  
  
    float lado2, lado3;  
}  
// fim da classe Triangulo
```

- Variáveis em escopos disjuntos (não relacionados), mesmo tendo o nome iguais, são variáveis diferentes
 - No exemplo, as variáveis `resultado` declaradas nas operações `éEquilátero` e `calculaPerímetro` são diferentes, apesar de terem o mesmo nome
- Variáveis declaradas num mesmo escopo devem ter nomes diferentes
- Uma variável declarada dentro de um escopo mais interno sobrepõe a definição de uma outra variável, com mesmo nome, declarada num escopo mais externo
 - Por exemplo, caso uma variável com nome `lado1` fosse definida como local à operação `calculaPerímetro`, então ela é que prevalece dentro do escopo desta operação, em detrimento da variável de instância de mesmo nome declarada na classe

Variáveis com valores constantes

- Variáveis podem ser declaradas com um valor inicial que não poderá ser alterado posteriormente durante a execução do programa

```
private final int LIMITE = 10000;
```

- O uso do modificador **final** assegura que o valor de inicialização da variável não poderá ser modificado em outro local dentro do programa, ou seja, o valor da variável é constante durante a execução do

- O bom estilo recomenda que se declare as constantes com todas letras maiúsculas

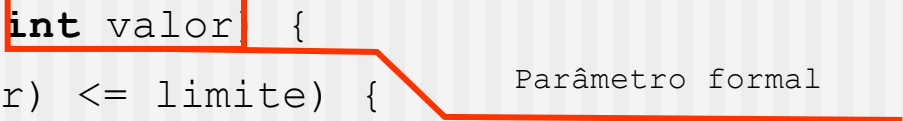
de compilação

- Útil para explicitar que o valor da variável não deve ser modificado

Operações parametrizadas

- Permitem variar os valores com os quais operam cada vez que são executadas, mas que não se sabe no momento em que são declaradas

```
public void compra(int valor) {  
    if ((saldo + valor) <= limite) {  
        saldo = saldo + valor;  
        creditaBonus(valor);  
    }  
    else {  
        System.out.println("Limite de crédito não ...");  
    }  
}
```



- A variável declarada como parâmetro de uma operação é denominada de parâmetro formal e denota um valor não sabido no momento da declaração da operação, mas que será fornecido no momento de sua execução

Operações parametrizadas

- No momento em que é executada, o valor da variável é passado como parâmetro, sendo denominado de parâmetro real ou argumento da operação

Memória

```
CartaoDeCredito cartao = new CartaoDeCredito();  
int valorCompra = 1203;  
cartao.compra(valorCompra);  
cartao.compra(394);
```

Parâmetros reais


1	
2	
3	
4	



Operações parametrizadas

- Podem ter mais de um parâmetro formal definido

```
public void compra(int valor, double juros, int parcelas) {  
    if ((saldo + valor) <= limite) {  
        ...  
    }  
    else {  
        System.out.println("Limite de crédito não ...");  
    }  
}
```

- A ordem, os tipos e a quantidade de parâmetros reais passados devem casar com os dos parâmetros formais

```
...  
cartao.compra(2933, 1.2, 10); 
```

```
...  
cartao.compra(2933, 10,  12); cartao.compra(2933,  10);
```

Operações parametrizadas

- Podem existir duas ou mais operações com o mesmo nome numa classe, desde que o número ou o tipo de seus parâmetros sejam diferentes

```
class CartaoDeCredito {  
    ...  
    public void compra(double valor) {  
        ...  
    }  
    public void compra(int valor) {  
        ...  
    }  
    ...  
    public void compra(int valor, double juros, int parcelas) {  
        ...  
    }  
    ...  
}
```

Expressões

- Definição

- trecho de programa que é avaliado para resultar num valor

- Modalidades de expressões

- Aritméticas
 - Condicionais
 - Lógicas (booleanas)

Expressões aritméticas

- Expressões que, quando avaliadas, resultam num valor numérico

Como é a avaliação? * é efetuada

primeiro que a +, ou vice-versa?

$x / y * z$ é avaliado como $(x / y) * z$ 10

- Re ou $x / (y * z)$ méticas

1. As subexpressões parentesadas têm prioridade de avaliação. Nas subexpressões parentesadas aninhadas, a avaliação é efetuada a partir da subexpressão parentesada mais interna

Expressões aritméticas

■ Regras para avaliação de expressões aritméticas

2. Ordem de precedência de operadores: operadores numa mesma subexpressão são avaliados na seguinte ordem:
 - $+$, $-$ unários: 1º
 - $*$, $/$, $\%$ binários: 2º
 - $+$, $-$ binários: 3º
3. Associatividade: os operadores numa mesma subexpressão e com mesma ordem de precedência são avaliados numa das seguintes formas:
 - Da direita para a esquerda: operadores unários $+$ e $-$
 - Da esquerda para a direita : operadores binários $*$, $/$, $\%$, $+$ e $-$

Operadores de Incremento e Decremento

`count = count + 1; // incrementa o em 1`
é equivalente a,
`count++;`

<i>Operador</i>	<i>Uso</i>	<i>Descrição</i>
<code>++</code>	<code>op++</code>	Incrementa op em 1; Avalia a expressão antes do valor ser acrescido
<code>++</code>	<code>++op</code>	Incrementa op em 1; Incrementa o valor antes da expressão ser avaliada
<code>--</code>	<code>op--</code>	Decrementa op em 1; Avalia a expressão antes do valor ser decrescido
<code>--</code>	<code>--op</code>	Decrementa op em 1; Decrementa op em 1 antes da expressão ser avaliada

Exemplos...

```
int i = 10,  
int j = 3;  
int k = 0;  
k = ++j + i; //resultará em k = 4+10 = 14
```

```
int i = 10,  
int j = 3;  
int k = 0;  
k = j++ + i; //resultará em k = 3+10 = 13
```

Precedência de Operadores

Ordem	Operador
1	() parênteses
2	++ pós-incremento e -- pós-decremento
3	++ pré-incremento e -- pré-decremento
4	! negação lógica
5	* multiplicação e / divisão
6	% resto da divisão
7	+ soma e - subtração
8	< menor que, <= menor ou igual, > maior que e >= maior ou igual
9	== igual e != não igual
10	& e binário
11	ou binário
12	^ ou exclusivo binário
13	&& e lógico
14	ou lógico
15	?: condicional
16	= atribuição

Expressões aritméticas

■ Exemplos

■ Fórmula para cálculo da área de um círculo

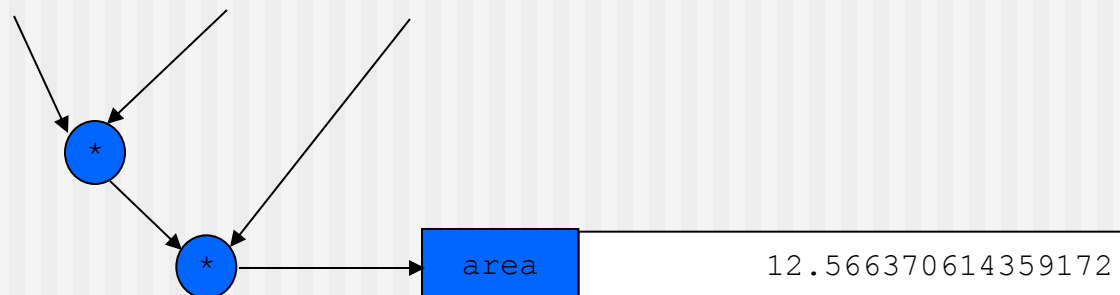
- $a = \pi \times r^2$

■ Em Java, têm-se

```
double area = 0, raio = 2;
```

```
...
```

```
area = Math.PI * raio * raio;
```



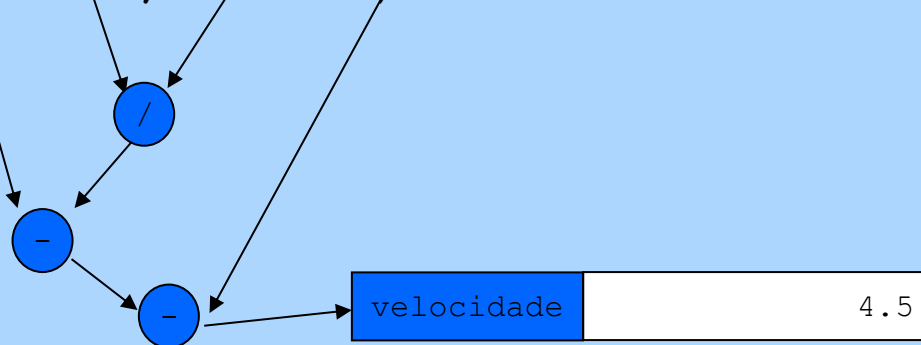
Expressões aritméticas

- Fórmula para cálculo da velocidade média

$$v = \frac{s_2 - s_1}{t_2 - t_1}$$

- A inserção de parênteses afeta a ordem normal da avaliação de operadores, por exemplo a avaliação da expressão sem parênteses é

`velocidade = s2 - s1 / t2 - t1;`



- O que resulta numa velocidade incorreta

Parênteses devem ser sempre usados para explicitar a ordem de avaliação nas expressões, mesmo quando isto não afetar o resultado

Expressões aritméticas

■ Exemplos

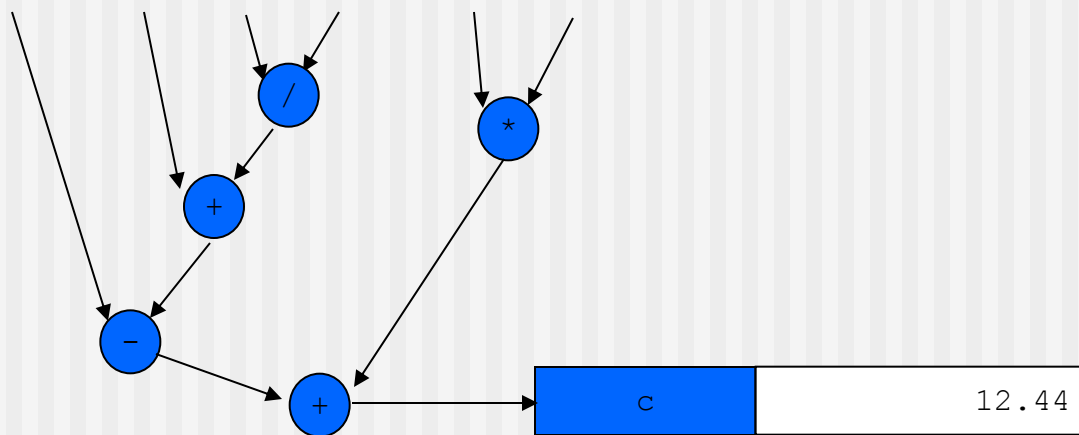
$$z - (a + b / 2) + w - y$$

■ Em Java, têm-se

```
double c = 0, z = 14, w = 1.5, y = 2.66, a = 3.4,  
b = 4.3;
```

...

```
c = z - (a + b / 2) + w * y;
```



Exemplos de fórmulas matemáticas

■ Fórmula

$$b^2 - 4ac$$

$$a + b - c$$

$$\frac{a + b}{c + d}$$

$$\frac{1}{1 + x^2}$$

$$a \times -(b + c)$$

■ Expressão em Java

$$b * b - 4 * a * c$$

$$a + b - c$$

$$(a + b) / (c + d)$$

$$1 / (1 + x * x)$$

$$a * -(b + c)$$

Funções matemáticas

- Disponíveis na classe **Math** da biblioteca de classes de Java
 - A classe **Math** provê métodos para realizar operações numéricas básicas, como cálculo de exponencial, logaritmo, raiz quadrada, funções trigonométricas
 - A documentação da classe pode ser encontrada em:

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html>

Funções matemáticas

- Exemplo: cálculo das raízes de equações de 2o grau

$$ax^2 + bx + c$$

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

```
double x = 0, a = 3.5, b = 14, c = 1.5;  
double discriminante = Math.pow(b, 2) - (4 * a * c);  
if (discriminante > 0) {  
    double r1 = (-b + Math.sqrt(discriminante)) / (2 * a);  
    double r2 = (-b - Math.sqrt(discriminante)) / (2 * a);  
}
```

Funções matemáticas

- Exemplo: função de arredondamento
 - **round**: retorna o inteiro mais próximo do argumento do tipo ponto flutuante

```
double a = 3.2, b = 3.5, c = 3.7;  
System.out.println("a = "+Math.round(a)+"", "+  
                  "b = "+Math.round(b)+"", "+  
                  "c = "+Math.round(c) );
```

- Resultado

```
a = 3, b = 4, c = 4
```

Funções matemáticas

- Exemplo: função de truncagem
 - **floor**: retorna um valor do tipo double correspondente a parte inteira do double passado como argumento

```
double a = 3.2, b = 3.5, c = 3.7;  
System.out.println("a = "+Math.floor(a)+"", "+  
                  "b = "+Math.floor(b)+"", "+  
                  "c = "+Math.floor(c));
```

- Resultado

```
a = 3.0, b = 3.0, c = 3.0
```

Funções matemáticas

■ Exemplo: função valor absoluto

- **abs**: retorna o valor absoluto do argumento

```
double a = -0.01, b = -3.7; int c = 3;  
System.out.println("a = "+Math.abs(a)+"", "+  
                    "b = "+Math.abs(b)+"", "+  
                    "c = "+Math.abs(c));
```

■ Resultado

```
a = 0.01, b = 3.7, c = 3
```

Funções matemáticas

- Exemplo: função randômica

- **random**: retorna um valor pseudoaleatório maior ou igual a 0.0 e menor que 1.0

```
System.out.println("a = "+Math.random()+"", "+  
                    "b = "+Math.random()+"", "+  
                    "c = "+Math.random() );
```

- Possível resultado

```
a = 0.9360168313500755, b = 0.8460201346813774, c = 0.3053851563688338
```

Operadores Relacionais

- Usados para comparar dois valores e determinar o relacionamento entre eles.
- O resultado será um valor lógico: **true** ou **false**

<i>Operador</i>	<i>Uso</i>	<i>Descrição</i>
>	op1 > op2	op1 é maior do que op2
>=	op1 >= op2	op1 é maior ou igual a op2
<	op1 < op2	op1 é menor do que op2
<=	op1 <= op2	op1 é menor ou igual a op2
==	op1 == op2	op1 é igual a op2
!=	op1 != op2	op1 não igual a op2

Expressões condicionais

■ Definição

- Uma expressão condicional é aquela que tem várias subexpressões, mas que apenas uma delas é escolhida para ser avaliada
- Java permite a elaboração de expressões condicionais com o operador condicional ?
 - (condição) ? expressão-esquerda : expressão-direita
(ternário): avalia a expressão à esquerda do sinal de dois pontos caso a condição seja avaliada verdadeira, caso contrário, avalia a expressão à direita

Expressões condicionais

■ Exemplos

■ Maior dentre duas variáveis double

```
double a = -0.01, b = -3.7;  
double max = (a > b) ? a : b;
```

■ Cálculo da média final

```
double mediaNotas = 0.0, notaProvaFinal = 0.0;  
...  
double mediaFinal = (mediaNotas >= 7.0) ?  
    mediaNotas :  
    ((6*mediaNotas + 4*notaProvaFinal) / 10);
```


Operadores Lógicos

- Avaliam um ou mais operandos lógicos que geram valor final **true** ou **false** como resultado.
- São seis os operadores lógicos:
 - && (e lógico),
 - & (e binário),
 - || (ou lógico),
 - | (ou binário),
 - ^ (ou exclusivo binário) e
 - ! (negação).
- A operação básica para um operador lógico é:
 - $x1 \text{ op } x2$
x1 e x2 podem ser expressões, variáveis ou constantes lógicas, e op pode tanto ser &&, &, ||, | ou ^.

Expressões lógicas

■ Definição

- Expressões que, quando avaliadas, resultam num valor verdadeiro ou falso

`salarioBruto > limiteIsenção`

`(temperatura <= 25) && (umidade < 65)`

- Regras para avaliação de expressões lógicas

1. As subexpressões parenteadas têm prioridade de avaliação. Nas subexpressões parenteadas aninhadas, a avaliação é efetuada a partir da subexpressão parenteada mais interna

Expressões lógicas

■ Regras para avaliação de expressões aritméticas

2. Ordem de precedência de operadores: operadores numa mesma subexpressão são avaliados na seguinte ordem:

- +, -, ! unários: 1º
- *, /, % binários: 2º
- +, - binários: 3º
- >, <, <=, >= binários: 4º
- ==, != binários: 5º
- && binário: 6º
- || binário: 7º
- ? : ternário: 8º

Expressões lógicas

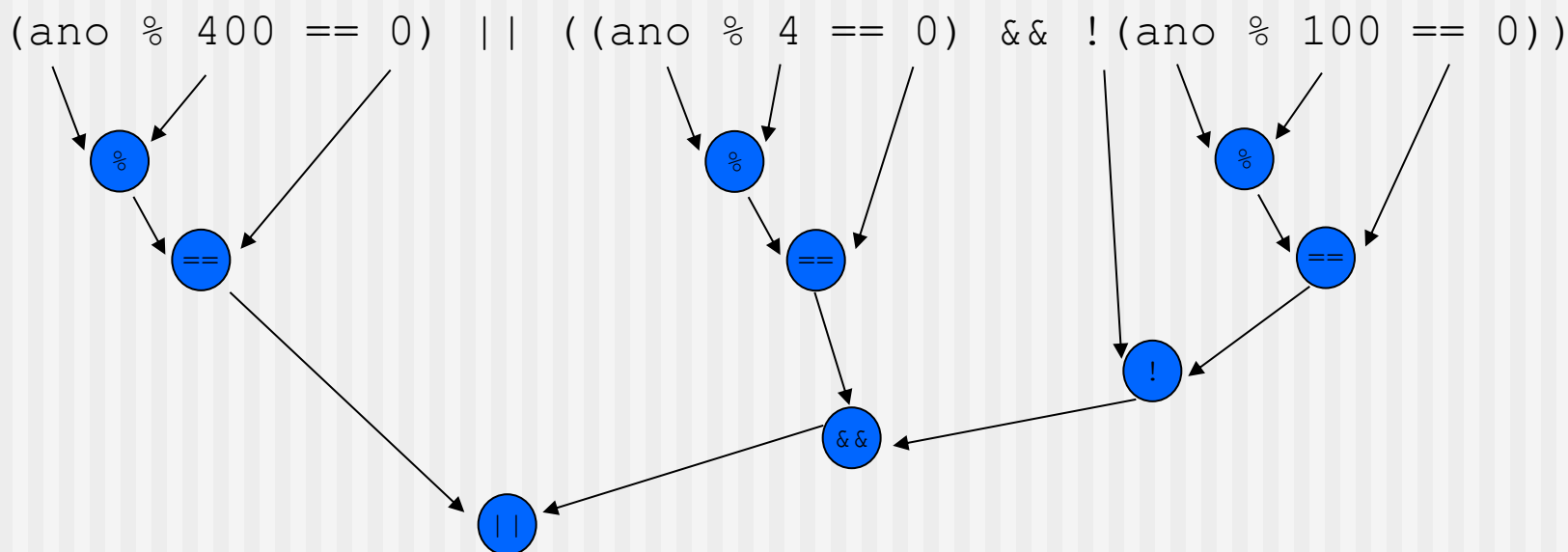
■ Regras para avaliação de expressões aritméticas

2. Associatividade: os operadores numa mesma subexpressão e com mesma ordem de precedência são avaliados numa das seguintes formas:

- Da direita para a esquerda: operadores unários $+$, $-$ e $!$
- Da esquerda para a direita : os demais operadores
- Exemplos
 - $a < b + c$ é equivalente a $a < (b + c)$
 - $a \geq b \ || \ a < b$ é equivalente a $(a \geq b) \ || \ (a < b)$

Expressões lógicas

- Exemplo: expressão para determinar se um ano é bissexto
 - Um ano é bissexto se for divisível exatamente por quatrocentos ou divisível por quatro e não divisível exatamente por 100



Expressões lógicas

- Os operadores lógicos são avaliados em “curto-circuito”

- Caso do operador &&

Para que as avaliações dos operadores sejam integrais, também chamadas de avaliações estritas, deve-se usar os operadores

- & no lugar de &&
- | no lugar de ||
- Caso a expressão do lado esquerdo seja avaliada true, então a avaliação da expressão do lado direito é desprezada e o resultado final é true
 - $(2000 \% 4 == 0) || ((1 / 0) > 1)$ é igual a true
 - $((1/0) > 1) || (2000 \% 4 == 0)$ provoca um erro em tempo de execução, embora sejam logicamente equivalentes

Expressões lógicas

- && (e lógico) e & (e binário)

<i>x1</i>	<i>x2</i>	<i>Resultado</i>
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

- A diferença básica do operador && para & é":
 - && suporta uma avaliação de curto-circuito (ou avaliação parcial), enquanto que o & não.
- Ex:
i = 0;
j = 10;
teste = (i > 10) && (j++ > 9); // j = 10
teste = (i > 10) & (j++ > 9); // j = 11 (realiza segunda operação)

Expressões lógicas

- `||` (ou lógico) e `|` (ou binário)

<i>x1</i>	<i>x2</i>	<i>Resultado</i>
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

- A diferença básica entre os operadores `||` e `|` é semelhante ao operador `&&`: `||` também suporta a avaliação parcial.
- Ex:
 `i = 0;`
 `j = 10;`
 `teste = (i < 10) || (j++ > 9); // j = 10`
 `teste = (i < 10) | (j++ > 9); // j = 11 (realiza segunda operação)`

Expressões lógicas

- \wedge (ou exclusivo binário)
 - O resultado de uma expressão usando o operador ou exclusivo binário terá um valor true somente se uma das expressões for verdadeira e a outra falsa.
 - Note que ambos os operandos são necessariamente avaliados pelo operador

<i>x1</i>	<i>x2</i>	<i>Resultado</i>
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

Expressões lógicas

■ ! (negação)

- inverte o resultado lógico de uma expressão, variável ou constante
 - o verdadeiro será falso e vice-versa.

<i>x1</i>	<i>Resultado</i>
VERDADEIRO	FALSO
FALSO	VERDADEIRO

Exercícios

- 1. Dada a tabela abaixo, declare as variáveis que se seguem de acordo com seus tipos correspondentes e valores iniciais. Exiba o nomes e valor das variáveis.

<i>Nome das Variáveis</i>	<i>Tipo do dado</i>	<i>Valor inicial</i>
number	integer	10
letter	character	a
result	boolean	true
str	String	hello

Exercícios

- 2. Crie um programa que obtenha a média de 3 números. Considere o valor para os três números como sendo 10, 20 e 45. O resultado esperado do exercício é:
 - número 1 com o valor 10
 - número 2 com o valor 20
 - número 3 com o valor 45
 - A média é 25

Exercícios

- 3. Dados três números, crie um programa que exiba na tela o maior dentre os números informados. Use o operador ? (dica: use dois operadores ?: para se chegar ao resultado). Por exemplo, dados os números 10, 23 e 5, o resultado esperado do exercício deve ser:
 - número 1 com o valor 10
 - número 2 com o valor 23
 - número 3 com o valor 5
 - O maior número é 23

Exercícios

- 4. Dadas as expressões abaixo, reescreva-as utilizando parênteses de acordo com a forma como elas são interpretadas pelo compilador.

1. $a / b \wedge c \wedge d - e + f - g * h + i$

2. $3 * 10 * 2 / 15 - 2 + 4 \wedge 2 \wedge 2$

3. $r \wedge s * t / u - v + w \wedge x - y++$

Estruturas de Controle

Estruturas de controle de decisão

- Permitem que blocos específicos de código sejam escolhidos para serem executados, redirecionando determinadas partes do fluxo do programa. Podem ser:
 - Declaração if
 - Declaração if-else
 - Declaração if-else-if
 - Declaração switch

Declaração IF

- especifica que uma instrução ou bloco de instruções seja executado se, e somente se, uma expressão lógica for verdadeira

```
if (expressão_lógica)  
    instrução;
```

ou

```
if (expressão_lógica) {  
    instrução1;  
    instrução2  
    ...  
}
```

Declaração IF

■ Exemplos

```
int grade = 68;  
if (grade > 60)  
    System.out.println("Congratulations!");
```

```
int grade = 68;  
if (grade > 60) {  
    System.out.println("Congratulations!");  
    System.out.println("Você foi aprovado!");  
}
```

Declaração if-else

- usada quando queremos executar determinado conjunto de instruções se a condição for verdadeira e outro conjunto se a condição for falsa

```
if (expressão_lógica)
    instrução_caso_verdadeiro;
else
    instrução_caso_falso;
```

ou

```
if (expressão_lógica) {
    instrução_caso_verdadeiro1;
    instrução_caso_verdadeiro2;
    ...
} else {
    instrução_caso_falso1;
    instrução_caso_falso2;
    ...
}
```

Declaração if-else

```
int grade = 68;
if (grade > 60)
    System.out.println("Congratulations! You passed!");
else
    System.out.println("Sorry you failed");
```

```
int grade = 68;
if (grade > 60) {
    System.out.print("Congratulations! ");
    System.out.println("You passed!");
} else {
    System.out.print("Sorry ");
    System.out.println("you failed");
}
```

Declaração if-else-if

- **else** pode conter outra estrutura if-else. Este cascadeamento de estruturas permite ter decisões lógicas muito mais complexas

```
if (expressão_lógica1)
    instrução1;
else if (expressão_lógica2)
    instrução2;
else
    instrução3;
```

Declaração if-else-if

```
if (grade >= 90) {  
    System.out.println("Excellent!");  
} else if((grade < 90) && (grade >= 80)) {  
    System.out.println("Good job!");  
} else if((grade < 80) && (grade >= 60)) {  
    System.out.println("Study harder!");  
} else {  
    System.out.println("Sorry, you failed.");  
}
```

Erros comuns no uso do "if"

1. A condição não avalia um valor lógico.

```
int number = 0;
if (number) {
    // algumas instruções aqui
}
```

2. Usar = (atribuição) em vez de == (igualdade) para comparação.

```
int number = 0;
if (number = 0) {
    // algumas instruções aqui
}
```

3. Escrever elseif em vez de else if.

```
if (number == 0) {
    // algumas instruções aqui
} elseif (number == 1) {
    // algumas instruções aqui
}
```

Declaração switch

- permite que uma única variável inteira tenha múltiplas possibilidades de finalização

```
switch (variável_inteira) {  
    case valor1:  
        instrução1; //  
        instrução2; // bloco 1  
        ... //  
        break;  
    case valor2:  
        instrução1; //  
        instrução2; // bloco 2  
        ... //  
        break;  
    default:  
        instrução1; //  
        instrução2; // bloco n  
        ... //  
        break;  
}
```


Declaração switch

```
int grade = 92;
switch(grade) {
    case 100:
        System.out.println("Excelente!");
        break;
    case 90:
        System.out.println("Bom Trabalho!");
        break;
    case 80:
        System.out.println("Estude um pouco mais!");
        break;
    default:
        System.out.println("Você foi reprovado!");
}
```

Estruturas de controle de repetição

- são comandos que permitem executar partes específicas do código determinada quantidade de vezes
- São eles:
 - **while**,
 - **do-while** e
 - **for**

Declaração while

- executa repetidas vezes um bloco de instruções enquanto uma determinada condição lógica for verdadeira

```
while (expressão_lógica) {  
    instrução1;  
    instrução2;  
    ...  
}
```

Declaração while

```
int i = 4;
while (i > 0) {
    System.out.print(i);
    i--;
}
```

```
// laço infinito
while (true)
    System.out.println("hello");
```

```
// a instrução do laço não será executada
while (false)
    System.out.println("hello");
```

Declaração do-while

- Similar ao while. As instruções dentro do laço do-while serão executadas pelo menos uma vez.

```
do {  
    instrução1;  
    instrução2;  
    ...  
} while (expressão_lógica);
```

Declaração do-while

```
int xint x = 0;
do {
    System.out.println(x);
    x++;
} while (x<10);

// laço infinito
do {
    System.out.println("hello");
} while(true);

// Um laço executado uma vez
do
    System.out.println("hello");
while (false);
```

Declaração for

- permite a execução do mesmo código uma quantidade determinada de vezes

```
for (declaração_inicial; expressão_lógica; salto)
{
    instrução1;
    instrução2;
    ...
}
```

Exemplo

```
for (int i = 0; i < 10; i++) {
    System.out.print(i);
}
```

Declarações de Interrupção

- permitem o redirecionamento do fluxo de controle do programa
- São ele:
 - **break**
 - **unlabeled** (não identificado)
 - **labeled** (identificado)
 - **continue**
 - **unlabeled** (não identificado)
 - **labeled** (identificado)
 - **return**

Declaração unlabeled break

- **break** encerra a execução de um **switch** e o controle é transferido imediatamente para o final deste. Podemos também utilizar a forma para terminar declarações **for**, **while** ou **do-while**.

```
String names[] = {"Zé", "Ana", "Bel", "Nico", "Iza"};
String searchName = "Iza";
boolean foundName = false;
for (int i=0; i < names.length; i++) {
    if (names[i].equals(searchName)) {
        foundName = true;
        break;
    }
}
if (foundName) {
    System.out.println(searchName + " found!");
} else {
    System.out.println(searchName + " not found.");
}
```

Declaração labeled break

- encerra um laço que é identificado por um label especificado na declaração break

```
int[][] numbers = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int searchNum = 5;
boolean foundNum = false;
searchLabel: for (int i=0; i<numbers.length; i++) {
    for (int j=0; j<numbers[i].length; j++) {
        if (searchNum == numbers[i][j]) {
            foundNum = true;
            break searchLabel;
        }
    } // final do laço j
} // final do laço i
if (foundNum) {
    System.out.println(searchNum + " found!");
} else {
    System.out.println(searchNum + " not found!");
}
```

Declaração unlabeled continue

- salta as instruções restantes de um laço e avalia novamente a expressão lógica que o controla

```
String names[] = {"Beah", "Bianca", "Lance", "Beah"};
int count = 0;
for (int i=0; i < names.length; i++) {
    if (!names[i].equals("Beah")) {
        continue; // retorna para a próxima condição
    }
    count++;
}
System.out.println(count + " Beahs in the list");
```

Declaração labeled continue

- interrompe a repetição atual de um laço e salta para a repetição exterior marcada com o label indicado

```
outerLoop: for (int i=0; i<5; i++) {  
    for (int j=0; j<5; j++) {  
        System.out.println("Inside    for(j)    loop");    //  
        mensagem1  
        if (j == 2)  
            continue outerLoop;  
    }  
    System.out.println("Inside    for(i)    loop");    //  
    mensagem2  
}
```

Declaração return

- Utilizada para sair de um método. O fluxo de controle retorna para a declaração que segue a chamada do método original.
- possui dois modos:
 - o que retorna um valor
 - `return ++count;`
 - `return "Hello";`
 - o que não retorna nada.
 - `return;`