

Abstract geometric lines in the top-left corner of the slide, consisting of several overlapping, irregular polygons and lines in a light gray color.

A SINTAXE BÁSICA DE JAVA

(MATERIAL ADAPTADO DO PROF. DR.
JOBSON MASSOLAR)



PRIMEIRO PROGRAMA

PRIMEIRO PROGRAMA EM JAVA

- Todo programa Java deve ter um **método main** que determina o início da execução do programa.
- Entretanto, por ser uma linguagem Orientada a Objetos, os métodos devem vir, obrigatoriamente, dentro de uma classe.
- Assim, um programa mínimo em Java deve ter uma **classe** e, dentro dela, um método chamado **main**.

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */  
  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Os comentários seguem o mesmo padrão do C/C++:
/* e */ para comentários multilinhas
// para comentários em uma linha

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */
```

```
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Todo código Java deve ficar dentro de uma classe.

Para definir uma classe usamos a palavra **class** seguida do **nome da classe**.

Os delimitadores { e } definem onde a classe inicia e onde ela termina.

A palavra **public** estabelece a visibilidade da classe (veremos isso mais a frente).

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

Repare que, em um determinado arquivo, devemos declarar somente uma classe.

O nome do arquivo deve ter o mesmo nome da classe com a extensão .java.

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

Dentro de uma classe podemos definir vários **métodos**.

Devemos definir, um e somente um, método **main**, que indica por onde a execução se iniciará.

String [] args são os parâmetros que o método poderá receber quando e se o programa for chamado via linha de comando.

A palavra **static** estabelece a forma de acesso ao método (veremos isso mais a frente).

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

Da mesma forma como fazemos nas funções em C/C++, antes do nome do método definimos o seu tipo de retorno.

O tipo de retorno do método **main** é sempre **void**, ou seja, não retorna nada.

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

O parâmetro **args** indica os argumentos do programa.

Os argumentos são os dados passados na linha de execução do programa (linha de comando).

String é o tipo de dado do parâmetro, neste caso será um **array** de String.

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

Assim como o C/C++ tem suas **funções** organizadas em um conjunto de **bibliotecas**, o Java possui uma série de **métodos** organizados em **classes**.

Nesse caso temos a classe **System**, que fornece, dentre outras coisas, os arquivos de entrada e saída padrão: **in** e **out**, que são atributos da classe **System**.

Aqui temos:

System é uma **classe**;

out é um **objeto** que pertence à classe **System** e que representa a saída padrão;

println é um **método** do objeto **out**.

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main

} // fim da classe AloMundo
```

Para acessarmos os elementos
dentro de uma classe ou objeto
usamos o operador **ponto**.

PRIMEIRO PROGRAMA EM JAVA

```
/* Arquivo: AloMundo.java */

public class AloMundo {
    public static void main(String [] args)
    {
        System.out.println("Alo mundo!");
    } // fim do método main
} // fim da classe AloMundo
```

O método **println** simplesmente imprime a string passada como parâmetro na saída padrão e pula uma linha.

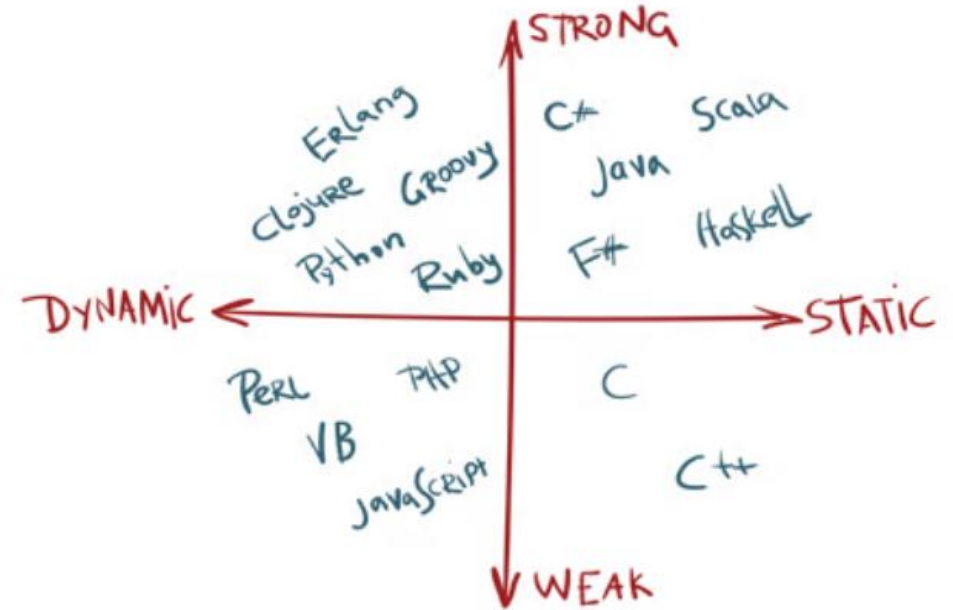
Repare que, assim como no C/C++, todo comando deve terminar com ;



CARACTERÍSTICAS DA SINTAXE

TIPOS DE DADOS

- Java é classificada como uma linguagem **fortemente e estaticamente tipada**.
- Isso quer dizer que todos os dados básicos manipulados por um programa Java estão associados a um determinado tipo (tipagem estática) e não há conversão implícita entre diferentes tipos (fortemente tipada).
- Quando realizamos qualquer operação com diversos dados diferentes, o Java usa o tipo de cada dado para verificar se a operação é válida.
- Qualquer incompatibilidade entre os tipos de dados, o compilador Java acusa como um erro.



TIPOS PRIMITIVOS DE DADOS

- Java possui 8 tipos primitivos de dados:

Tipo	Bits	Valor mínimo	Valor máximo
boolean	1	true ou false	
char	16	até 65.536 caracteres	
byte	8	-128	127
short	16	-32.768	32.767
int	32	-2.147.483.648	2.147.483.647
long	64	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
float	32	~ -1.4e-45	~ 3.4e38
double	64	~ -4.9e-324	~ 1.7e308

O tamanho dos tipos de dados independem da arquitetura da máquina, ou seja, o tipo int terá sempre 32 bits, independente da máquina onde o programa Java está sendo executado

VALORES PARA TIPOS PRIMITIVOS DE DADOS

Tipo	Valores	Exemplos
boolean	São representados por duas palavras.	true false
char	É uma letra ou símbolo em aspas simples.	'a' '.' '5'
byte	Composto de dígitos 0 a 9; Podem ser iniciadas com sinal negativo.	19 -3
short	“byte maior”.	190 -30
int	“short maior”.	1900 -300
long	“int maior”; Deve ser terminado em L maiúsculo.	19000L -3000L
float	Deve conter sempre o ponto (para diferenciar de valores inteiros); Deve ser terminado em f minúsculo.	4.93f
double	“float maior”.	-12.765

String consiste de uma sequência de zero ou mais caracteres entre aspas duplas.

Exemplo: “Oba !” | “Rio de Janeiro”

String não é um tipo primitivo de Java.

IDENTIFICADORES

- Identificadores são usados para nomear os elementos do nosso programa: variáveis, constantes, classes, métodos, atributos, etc.
- Os identificadores devem começar sempre com uma letra, \$, ou underscore (_).
- Após o primeiro caracter, são permitidos letras, \$, _ , ou dígitos.
- Não há limite de tamanho para o identificador.
- Identificadores em Java são *case-sensitive*, ou seja, **pessoa**, **PESSOA** e **Pessoa** são identificadores diferentes.
- Apesar de ser possível, não é recomendável o uso de caracteres de acentuação nos identificadores (ç, á, é, í, ó, ú, â, ê, ô, à, ã, õ).

IDENTIFICADORES

- Algumas palavras reservadas não são permitidas como identificadores:

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

VARIÁVEIS

- A declaração de variáveis tem a mesma sintaxe do C/C++:

```
int _a;  
double $c;  
char c1, c2, c3;  
long nomeBastanteExtensoParaMinhaVariavel;  
int i, j;  
boolean achou;
```

OPERADOR DE ATRIBUIÇÃO

- Em Java o operador de atribuição é responsável por colocar o resultado da expressão à direita na variável à esquerda:

`variavel = expressão`

- Podemos encadear várias atribuições a partir de uma única expressão:

`variavel1 = variavel2 = variavel3 = ... = expressão`

- Exemplos:

```
int i, j, k;  
double max, min;  
i = j = k = 1;    // Todas as variaveis = 1  
max = min = 0.0;  // Todas as variaveis = 0.0
```

VARIÁVEIS

- A declaração de variáveis tem a mesma sintaxe do C/C++:

```
double $c = 0.1;
```

```
char c1 = 'A', c2, c3;
```

```
long nomeBastanteExtensoParaMinhaVariavel = 1000L;
```

```
int i = 0, j = 1;
```

```
boolean achou = false;
```

Variáveis não inicializadas têm valor indefinido.

Se você tentar usar uma variável antes de definir seu valor, o compilador Java vai acusar um **erro**.

CONSTANTES

- A declaração e inicialização de constantes tem a mesma sintaxe da declaração e inicialização de variáveis, acrescidas da palavra **final**:

```
final int MAXIMO = 100;
```

```
final float PI = 3.14159f;
```

```
final char FIM = '$';
```

```
final boolean OK = true;
```

Constantes não podem ter seu valor alterado durante o programa.

Caso haja uma tentativa de alterar um desses valores o compilador Java acusará um **erro**.

CONVENÇÕES

- Apesar de não ser obrigatório, Java possui algumas convenções de nomenclatura que os programadores seguem:
1. Nomes de variáveis, atributos e métodos: utilizar o formato **camelCase** (primeira letra minúscula e primeira letra das demais palavras em maiúscula)
 - nomeCliente
 - matriculaAluno
 - cpf
 - dataNascimento
 2. Nomes de constantes: devem ser definidas em **caixa alta** e usar o **underscore** como separador:
 - LIMITE_SUPERIOR
 - MAX

CONVENÇÕES

- Apesar de não ser obrigatório, Java possui algumas convenções de nomenclatura que os programadores seguem:
-
3. Nomes de classes: utilizar o formato PascalCase (todas as primeiras letras das palavras em maiúsculo).
 - AloMundo
 - Cliente
 - Disciplina
 - SistemaAcademico
 - AlunoBolsista
 - UnidadeEnsino

OPERADORES ARITMÉTICOS

- Os operadores aritméticos previstos são:

Operador	Ação	Tipos
+	Soma	Inteiro e ponto flutuante
-	Subtração	Inteiro e ponto flutuante
*	Multiplicação	Inteiro e ponto flutuante
/	Divisão	Inteiro e ponto flutuante
%	Resto da divisão	Inteiro
++	Incremento	Inteiro e ponto flutuante
--	Decremento	Inteiro e ponto flutuante

OPERADORES ARITMÉTICOS

- Ordem de precedência dos operadores aritméticos

Operadores	Associação
++ --	Direita para esquerda
* / %	Esquerda para direita
+ -	Esquerda para direita

- Para alterar essa ordem devemos usar parênteses nas expressões.

```
float a, b, i = 10, j = 30, k = 40;
```

```
a = i + j / k;    // a = 10.75
```

```
b = (i + j) / k;  // b = 1
```

OPERADORES ARITMÉTICOS

Importante:

- O operador `/` pode ser aplicado tanto a números inteiros quanto a números de ponto flutuante.
- Quando todos os argumentos desse operador são inteiros então o resultado será um número inteiro, ou seja, a parte decimal é desprezada.

```
int i = 6, j = 3, k = 4;
```

```
i / j          resulta em 2
```

```
i / k          resulta em 1 e não em 1.5
```

- Para resolver esse problema será usado o operador de *casting*, que será visto mais adiante.

INTERFACE COM O USUÁRIO

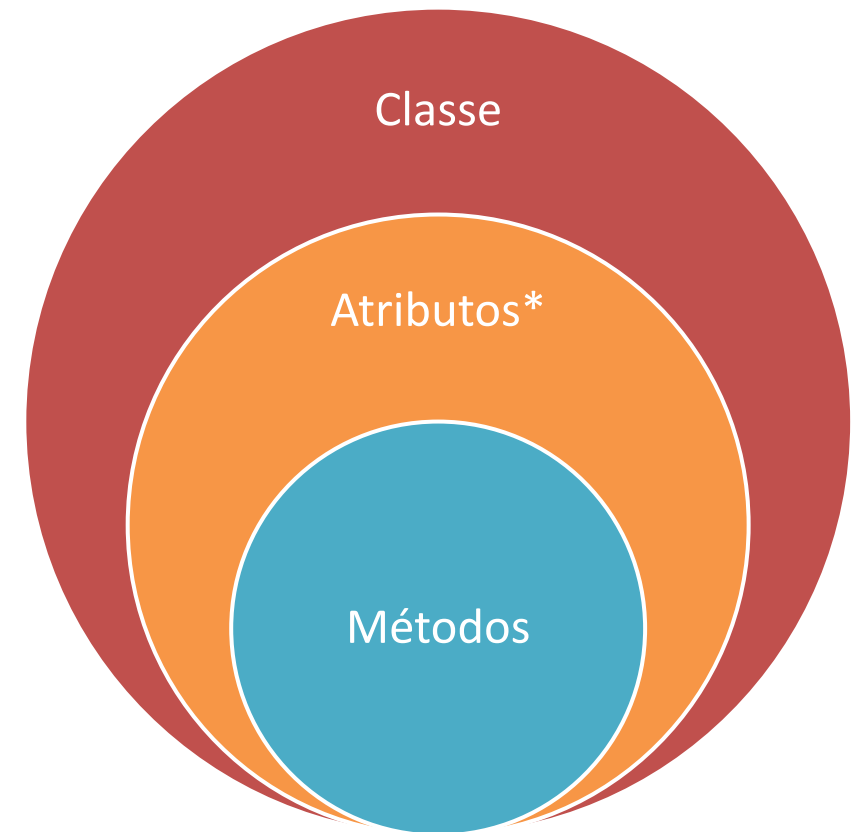
- O desenvolvimento da interface com o usuário em Java depende muito do tipo de aplicação a ser desenvolvida:
 - Para aplicações Desktop, as interfaces podem ser construídas com as bibliotecas AWT ou Swing do próprio Java.
 - Para as aplicações Web, as interfaces podem usar HTML/JavaScript/CSS ou um conjunto de componentes específicos de acordo com a tecnologia adotada. Por exemplo: componentes RichFaces para desenvolvimento usando a tecnologia Java Server Faces (JSF).
 - Para aplicações Móveis, normalmente são adotados componentes específicos dependendo do SO. Por exemplo: para o Android existem componentes específicos desenvolvidos pela Google.

Cada uma dessas tecnologias requer um tempo de aprendizado. Por isso, nesse curso vamos adotar somente a entrada/saída padrão via console (teclado e monitor em modo texto).

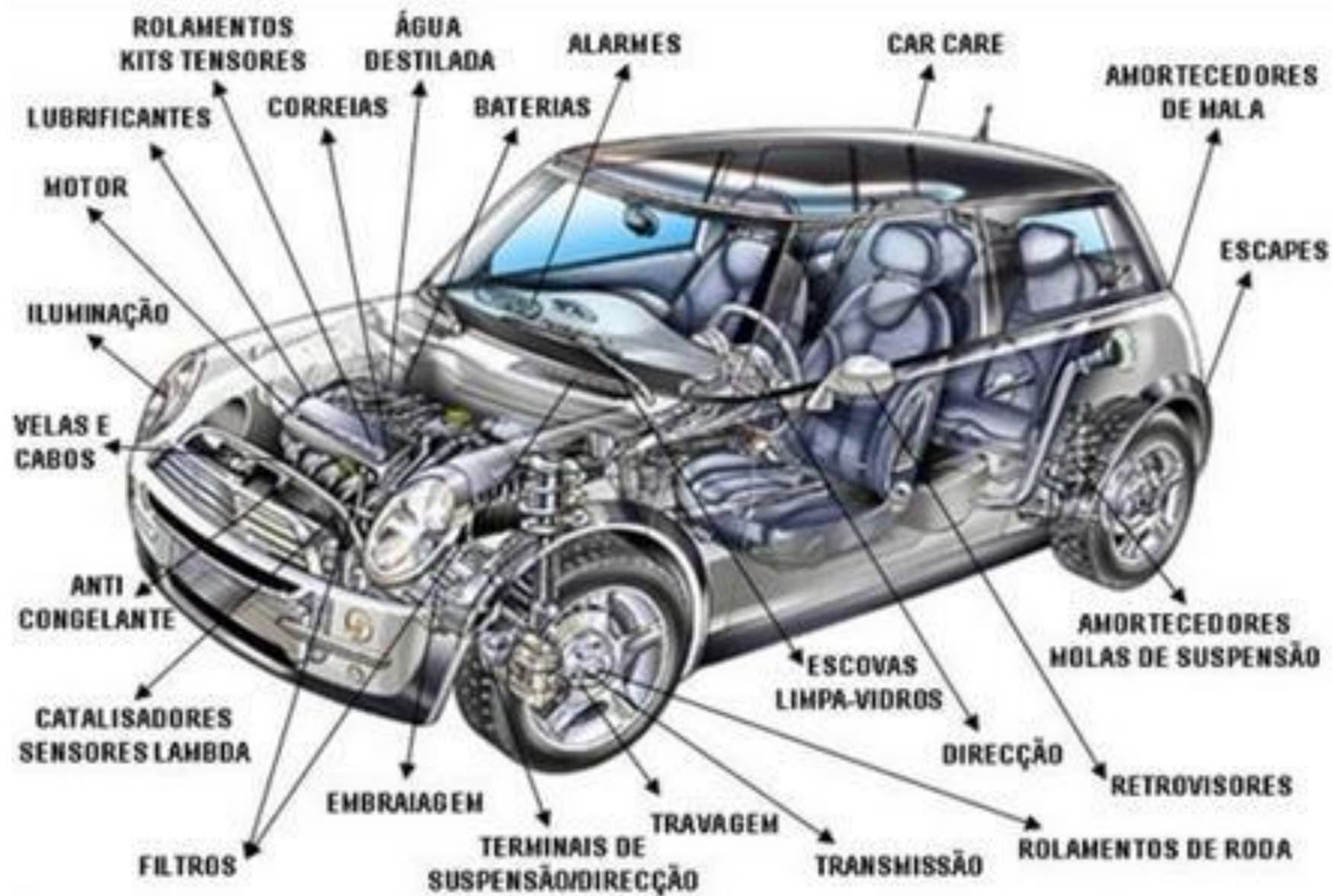
ENTRADA E SAÍDA VIA CONSOLE

- Classe System:
 - Pertence à biblioteca padrão do Java chamada `java.lang`.
 - A classe `System` define, dentre outras coisas, os arquivos de entrada e saída padrão.
 - O arquivo `out` representa a saída de vídeo.
 - O arquivo `in` representa a entrada via teclado.
 - Note que `in` e `out` são objetos e, por isso, podemos acessar seus métodos.

Para realizar a saída de dados via console (vídeo) usamos os métodos `print`, `println` ou `printf` do objeto `out`.



*Atributos são tipados. Podem ser tipos primitivos ou objetos de alguma classe.



ENTRADA E SAÍDA VIA CONSOLE

- Classe Scanner:
 - É usada para realizar a entrada de dados.
 - Pertence à uma biblioteca do Java chamada java.util.
 - Por não estar definido em uma biblioteca padrão, para usar a classe Scanner precisamos informar onde essa ela se encontra. Isso é feito através do comando import:

```
import java.util.Scanner;
```


ENTRADA E SAÍDA VIA CONSOLE

- Classe Scanner:
 - Existem vários métodos na classe Scanner para fazer a entrada de dados:
 - ler um int: `nextInt()`
 - ler um double: `nextDouble()`
 - ler um float: `nextFloat()`
 - ler um character: `nextLine().charAt(0)`
 - ler um long: `nextLong()`
 - ler uma string: `nextLine()`
 - Vamos ver o uso da classe Scanner com um exemplo.

ENTRADA E SAÍDA VIA CONSOLE

```
/* Arquivo: Leitura.java */
```

```
import java.util.Scanner;
```

```
public class Leitura {  
    public static void main(String [] args) {  
        int idade;  
        Scanner teclado = new Scanner(System.in);  
  
        System.out.println("Qual a sua idade ? ");  
        idade = teclado.nextInt();  
  
        System.out.printf("Idade = %d\n", idade);  
    } // fim do metodo main  
} // fim da classe
```

Importa a classe **Scanner** para avisar ao compilador Java onde ela está definida. Nesse caso ela está definida no pacote **java.util**.

ENTRADA E SAÍDA VIA CONSOLE

```
/* Arquivo: Leitura.java */

import java.util.Scanner;

public class Leitura {
    public static void main(String [] args) {
        int idade;
        Scanner teclado = new Scanner(System.in);

        System.out.println("Qual a sua idade ? ");
        idade = teclado.nextInt();

        System.out.printf("Idade = %d\n", idade);
    } // fim do metodo main
} // fim da classe
```

Cria um objeto da classe **Scanner**.
O operador **new** é usado para criar esse objeto.
A variável **teclado** passa a referenciar esse objeto.
Ao criar o objeto da classe Scanner, informamos de onde serão lidos os dados: **System.in**

ENTRADA E SAÍDA VIA CONSOLE

```
/* Arquivo: Leitura.java */
```

```
import java.util.Scanner;
```

```
public class Leitura {  
    public static void main(String [] args) {  
        int idade;  
        Scanner teclado = new Scanner(System.in);  
  
        System.out.println("Qual a sua idade ? ");  
        idade = teclado.nextInt();  
  
        System.out.printf("Idade = %d\n", idade);  
    } // fim do metodo main  
} // fim da classe
```

System.in é um objeto que referencia a entrada padrão, que no nosso caso é o **teclado**.

ENTRADA E SAÍDA VIA CONSOLE

```
/* Arquivo: Leitura.java */

import java.util.Scanner;

public class Leitura {
    public static void main(String [] args) {
        int idade;
        Scanner teclado = new Scanner(System.in);

        System.out.println("Qual a sua idade ? ");
        idade = teclado.nextInt();

        System.out.printf("Idade = %d\n", idade);
    } // fim do metodo main
} // fim da classe
```

Utilizamos o método **nextInt** para ler um número inteiro da entrada padrão (teclado).

O número lido é armazenado na variável **idade**.

ENTRADA E SAÍDA VIA CONSOLE

```
/* Arquivo: Leitura.java */

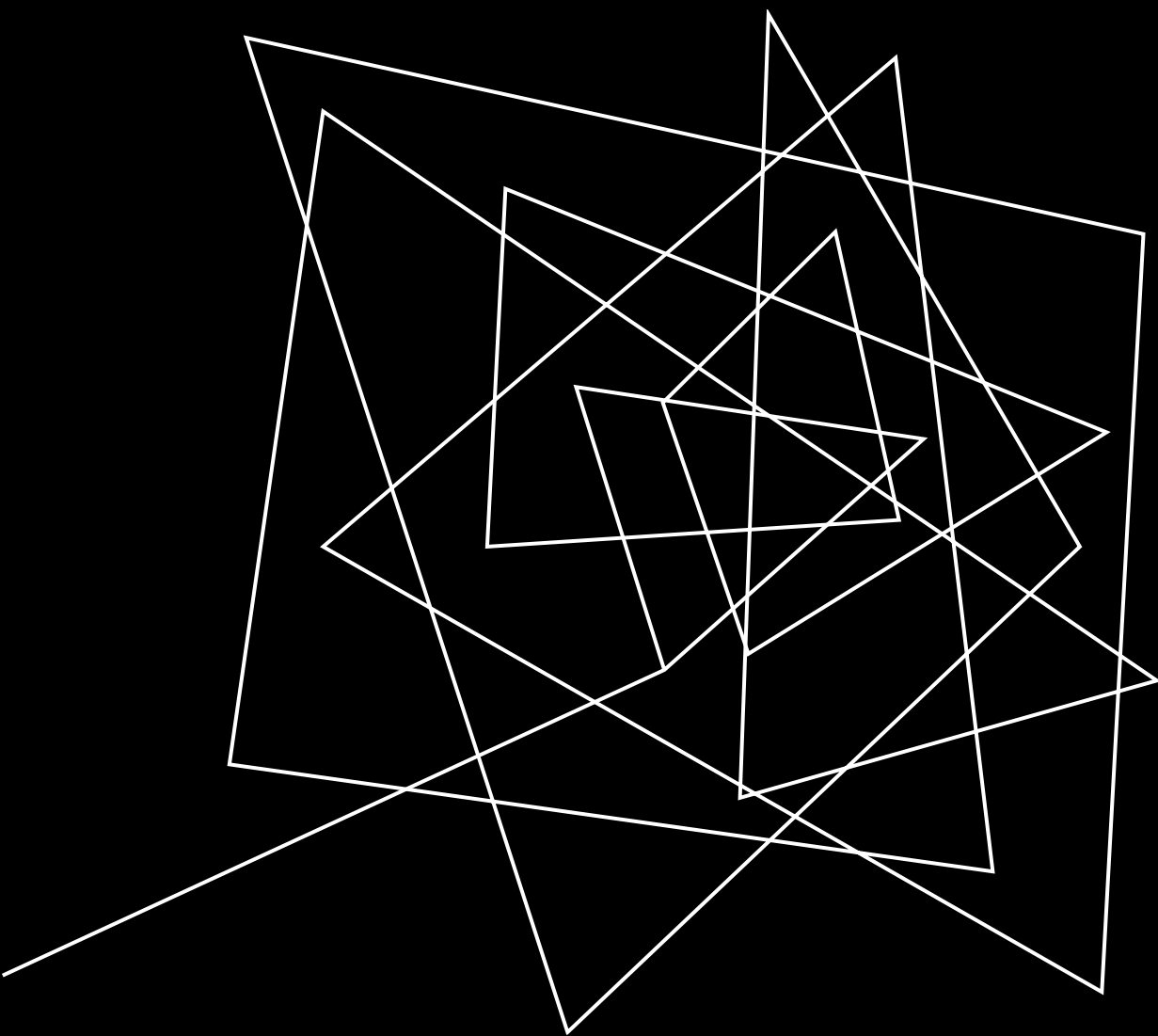
import java.util.Scanner;

public class Leitura {
    public static void main(String [] args) {
        int idade;
        Scanner teclado = new Scanner(System.in);

        System.out.println("Qual a sua idade ? ");
        idade = teclado.nextInt();

        System.out.printf("Idade = %d\n", idade);
    } // fim do metodo main
} // fim da classe
```

O método **printf()** do Java usa os mesmos formatadores da função **printf()** do C/C++.



PRATICANDO

PRATICANDO

1. Leia duas variáveis inteiras e imprima a soma, subtração, multiplicação e divisão entre elas.
2. Altere o tipo das duas variáveis do exercício 1 para ponto flutuante.
3. Leia o salário e o percentual de aumento. Em seguida, aplique o percentual de aumento sobre o salário e imprima o novo salário.
4. Leia o raio de um círculo. Em seguida imprima o perímetro ($2\pi R$) e a área (πR^2) do círculo com esse raio.

PRATICANDO

```
import java.util.Scanner;

public class Ex01 {
    public static void main(String [] args)
    {
        Scanner teclado = new Scanner(System.in);

        /* Coloque o resto do codigo aqui ! */
    }
}
```



+ CARACTERÍSTICAS DA SINTAXE

INCREMENTO E DECREMENTO

- Os operadores ++ e -- são muito utilizados em Java.
- Os operadores ++ e -- podem ser prefixados ou pósfixados. Assim:

x++

usa o valor de x e depois incrementa x

++x

incrementa x e depois usa o valor já incrementado

- Qual o valor final de i, j e k nas expressões abaixo?

```
int i, j, k;
```

```
i = 10;
```

```
j = i++;
```

```
k = ++j;
```

INCREMENTO E DECREMENTO

- Os operadores ++ e -- são muito utilizados em Java.
- Os operadores ++ e -- podem ser prefixados ou pósfixados. Assim:

x++

usa o valor de x e depois incrementa x

++x

incrementa x e depois usa o valor já incrementado

- Qual o valor final de i, j e k nas expressões abaixo?

```
int i, j, k;
```

```
i = 10;
```

```
j = i++;
```

```
k = ++j;
```

Esse trecho pode ser reescrito como:

```
i = 10;
```

```
j = i;
```

```
i++;
```

```
++j;
```

```
k = j;
```

Assim, i, j e k terão valor 11.

OPERADOR DE ATRIBUIÇÃO ARITMÉTICA

- Em programação é comum encontrar expressões como:

`quantidade = quantidade + 10;`

variável variável expressão

`salario = salario - salario * percentual / 100;`

variável variável expressão

OPERADOR DE ATRIBUIÇÃO ARITMÉTICA

- Em Java, é muito comum combinar os operadores aritméticos com o operador de atribuição.
- A sintaxe da atribuição aritmética é:

Expressão normal	Expressão
<code>var = var + expressão</code>	<code>var += expressão</code>
<code>var = var - expressão</code>	<code>var -= expressão</code>
<code>var = var * expressão</code>	<code>var *= expressão</code>
<code>var = var / expressão</code>	<code>var /= expressão</code>
<code>var = var % expressão</code>	<code>var %= expressão</code>

OPERADOR DE ATRIBUIÇÃO ARITMÉTICA

- Exemplos:

`quantidade = quantidade + 10;`

`quantidade += 10;`

`salario = salario - salario * percentual / 100;`

`salario -= salario * percentual / 100;`

CONVERSÃO DE TIPOS

- Quando misturamos vários tipos em uma expressão, o Java tenta sempre converter os tipos com valores menos significativos para tipos mais significativos para não haver perda de dados durante o processamento.
- Essa conversão se dá na seguinte ordem:

byte → short → int → long → float → double

- Exemplos:
 - Se uma expressão envolve tipos byte e int, os valores das variáveis do tipo byte serão convertidos para int antes de avaliar a expressão.
 - Se uma expressão envolve os tipos int, float e double, os valores das variáveis int e float serão convertidos para double antes da avaliação.

CONVERSÃO DE TIPOS

- Uma expressão só pode ser atribuída a uma variável se o tipo dessa expressão for igual ou menos significativo que o tipo da variável. Caso contrário será gerado um erro de compilação.
- Assim, a atribuição deve ser feita obedecendo à seguinte ordem :

byte → short → int → long → float → double

- Exemplos:
 - Um expressão do tipo int pode ser armazenada em uma variável do tipo int ou long.
 - Um expressão do tipo float pode ser armazenada em uma variável do tipo float ou double.
 - Um expressão do tipo long pode ser armazenada em uma variável do tipo long, float ou double

CASTING

- Para forçarmos a conversão de um tipo para outro usamos o operador de casting. Existem duas sintaxes:

(tipo) *variável*

converte a variável para o tipo

(tipo) (*expressão*)

converte o resultado da expressão para o tipo

- Exemplos:

```
int i = 6, j = 3, k = 4;
```

```
(float) i / j   converte i para 6.0 e o resultado é 2.0
```

```
(float)(i) / k  converte i para 6.0 e o resultado é 1.5
```

```
(float)(i / k) converte 1 para 1.0, ou seja só faz o cast depois da divisão
```

CONVERSÃO DE TIPOS E CASTING

- Qual o resultado das expressões abaixo?

- $5 * 4 / 6 + 7$

- $5 * 4 / (6 + 7)$

- $5 * 4.0 / 6 + 7$

- $5 * 4 \% 6 + 7$

- $5 * 4 / (\text{float})6 + 7$

- $(4 / 3) + (3.0 * 5)$

- $(4 / 3.0) + (3 * 5)$

- $(\text{int})(4 / 3.0) + (3 * 5)$

OPERADORES RELACIONAIS

- Os operadores relacionais do Java são:

Operador	Significado
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual a
!=	Diferente de

- A sintaxe das operações relacionais é:

expressão_aritmética_1 **op_relacional** expressão_aritmética_2

OPERADORES RELACIONAIS

- Os operadores aritméticos tem precedência sobre os operadores relacionais. Assim, se colocarmos esses dois tipos de operadores em uma mesma expressão, os operadores aritméticos serão avaliados primeiro e, em seguida, os relacionais.

$$1 + 3 >= 3 + 6$$

$$(1 + 3) >= (3 + 6)$$

$$5.0 / 3 <= 10 / (4 + 1)$$

$$(5.0 / 3) <= (10 / (4 + 1))$$

OPERADORES LÓGICOS

- Os operadores lógicos do Java são:

Operador	Significado
&&	E (AND)
	OU (OR)
!	NEGAÇÃO (NOT)

- A sintaxe das operações relacionais é:

expressão_relacional_1 **op_lógico** expressão_relacional_2

OPERADOR CONDICIONAL

- O operador ?: é usado em expressões condicionais, ou seja, uma expressão que podem ter um ou outro valor dependendo de uma condição. Sua sintaxe é:

`condição ? expressão_1 : expressão_2`

- A avaliação dessa expressão é feita da seguinte forma:

se `condição` for verdadeira então

o resultado da expressão é `expressão_1`

senão

o resultado da expressão é `expressão_2`

OPERADOR CONDICIONAL

- Exemplo:

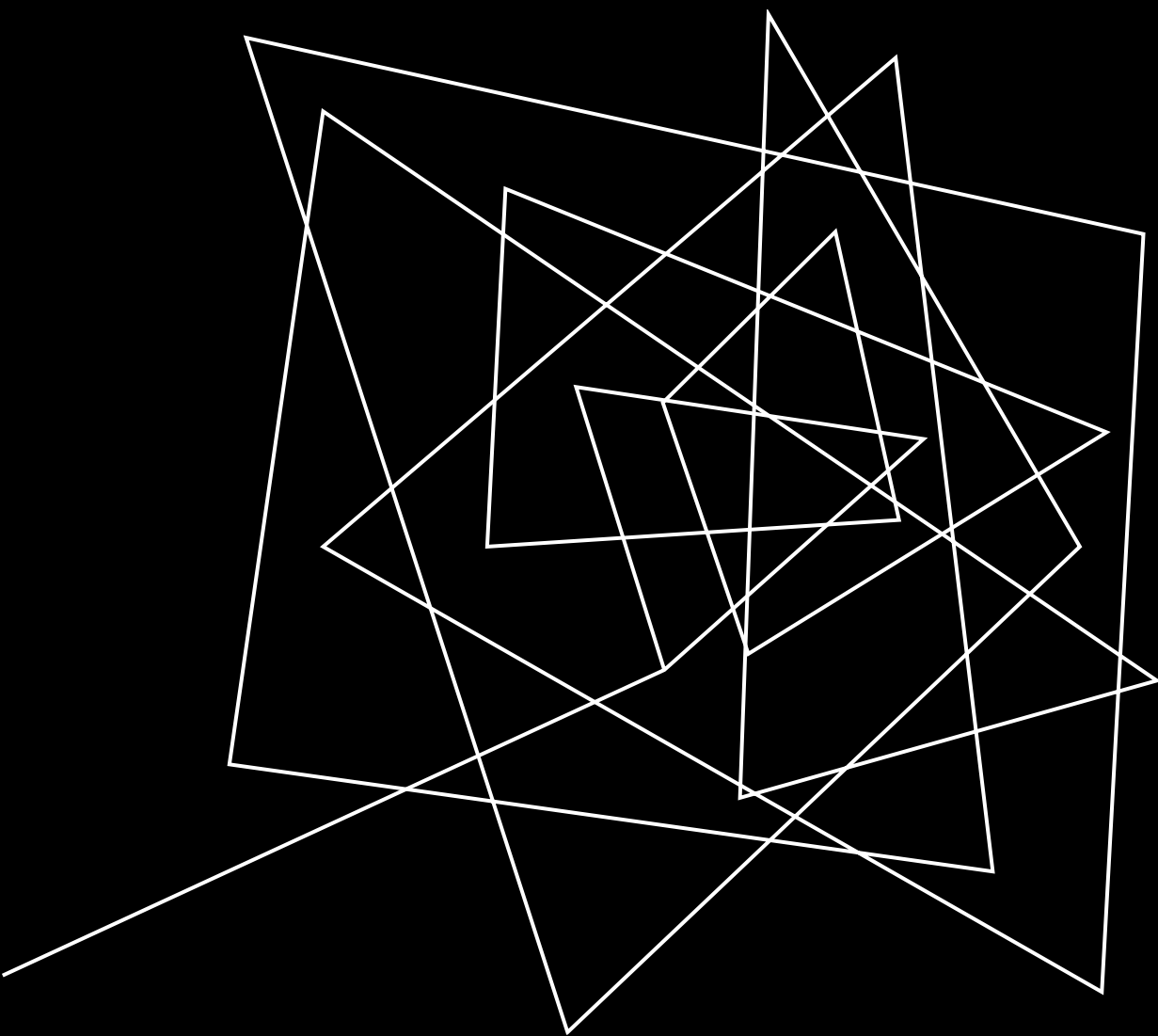
```
int menor, i, j;
```

```
if (i < j)
    menor = i;
else
    menor = j;
```

} menor = i < j ? i : j;

OPERADOR CONDICIONAL

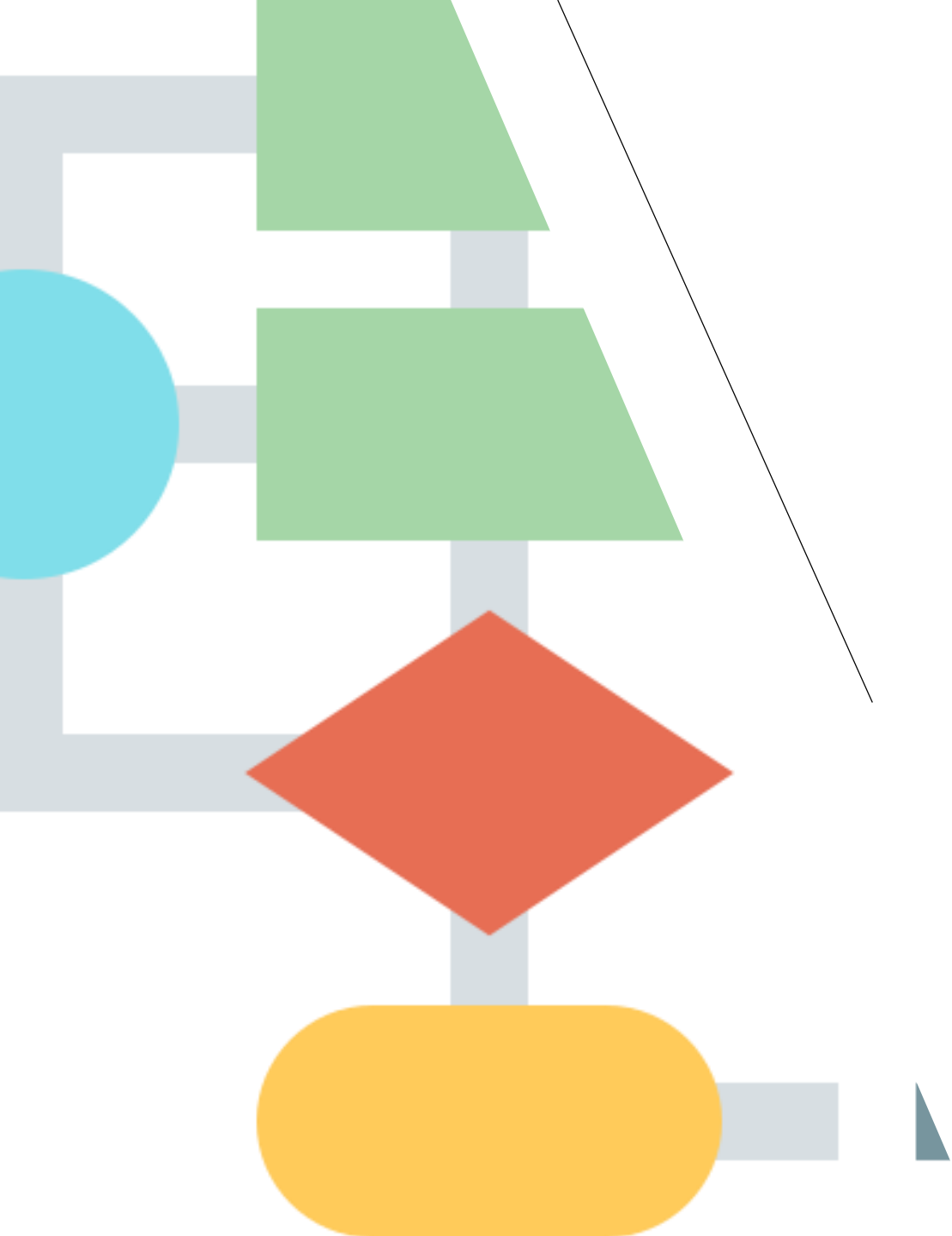
- Qual o valor de i em cada expressão?
 - `int i = 1, j = 2, k = 3;`
 - `i = i > k ? i : k;`
 - `i = i > 0 ? j : k;`
 - `i = j > i ? ++k : --k;`
 - `i = k == i && k != j ? i + j : i - j;`



PRATICANDO

PRATICANDO

5. Leia uma variável t com um tempo qualquer em segundos e imprima esse valor em hora, minuto e segundo.
6. Leia a distância percorrida por um carro, o tempo gasto e a quantidade de gasolina consumida. Em seguida, imprima a velocidade média (KM/h) e o consumo de combustível (Km/l).
7. Leia uma variável n inteira. Em seguida, imprima uma mensagem informando se o número n é par ou ímpar.
8. Leia duas variáveis com a quantidade de kilowatts consumidos em uma casa e o valor do kilowatt. Em seguida, calcule o valor a ser pago, concedendo um desconto de 10% caso o consumo seja menor que 150Kw.



ESTRUTURAS DE CONTROLE

ESTRUTURA DE CONTROLE – BLOCO

- Bloco de Comandos

- Cria um bloco que agrupa declaração de variáveis e comandos.
- Variáveis declaradas dentro de um bloco são visíveis apenas nesse bloco.
- Podemos aninhar blocos, ou seja, declarar um bloco dentro de outro.
- Se houver duas variáveis com o mesmo nome declaradas em um bloco externo e um bloco interno, o Java apresenta o mesmo erro de quando duas variáveis com mesmo nome são declaradas no mesmo bloco.
- Sintaxe:

```
{  
    /* declaracao e comandos */  
}
```

ESTRUTURA CONDICIONAL – IF...ELSE

- Comando if...else

- Comando de seleção que permite analisar uma expressão lógica e desviar o fluxo de execução.
- Sintaxes:

```
if (expressao_logica)
    comando;
```

```
if (expressao_logica)
    comando1;
else
    comando2;
```

```
if (expressao_logica1)
    comando1;
else if (expressao_logica2)
    comando2;
else
    comando3;
```

```
if (expressao_logica){
    bloco_de_comandos}
```

```
if (expressao_logica){
    bloco_de_comandos1}
else{
    bloco_de_comandos2}
```

```
if (expressao_logica){
    bloco_de_comandos1}
else if (expressao_logica){
    bloco_de_comandos2}
else{
    bloco_de_comandos3}
```

ESTRUTURA CONDICIONAL – SWITCH

- **Comando switch**
 - É um comando de seleção semelhante ao if-else, porém ele é mais recomendado quando temos muitos caminhos possíveis a partir de uma única condição.
 - A expressão do switch tem que ser, obrigatoriamente, do tipo caracter (char) ou inteiro (byte, short, int ou long).
 - O comando break é usado para terminar o switch.

ESTRUTURA CONDICIONAL – SWITCH

- Comando switch

```
switch(expressão)
{
    case valor1: comando1;
                comando2;
                break;

    case valor2: comando3;
                break;

    case valor3: comando4;
                comando5;
                break;

    default:    comando6;
                break;
}
```

valor1, valor2, valor3, etc.,
podem ser **variáveis** ou
constantes.

Quando executa um **break** o
switch termina.

Se a expressão não for
igual a nenhuma opção é
executado o **default**.

ESTRUTURA CONDICIONAL – SWITCH

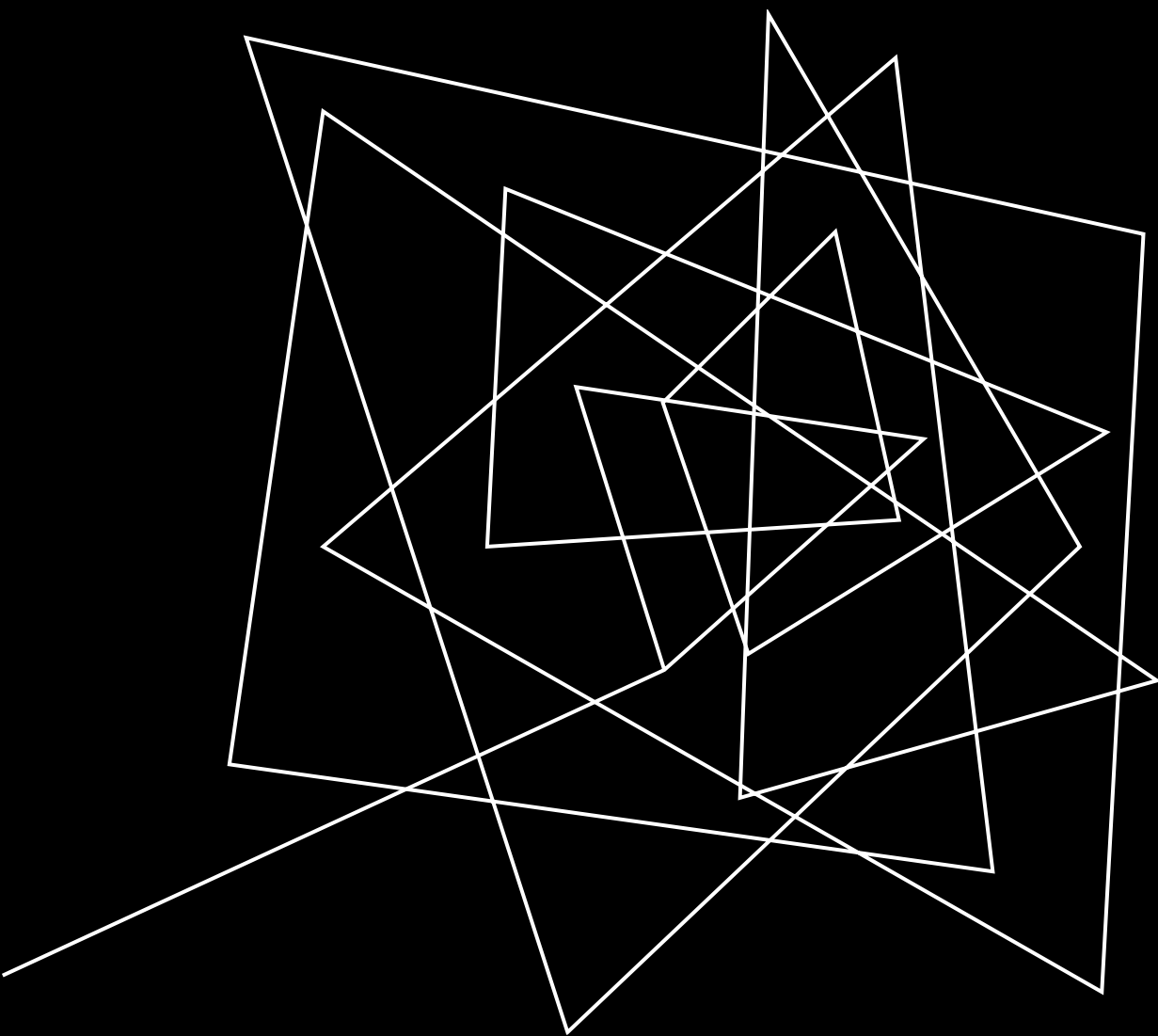
- Comando switch

- Quando o switch encontra uma opção igual ao valor da expressão, ele executa todos os comandos daí em diante até encontrar o comando break.
- O case pode ter um comando vazio.

```
switch(caracter)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':    System.out.println("É uma vogal");
                break;

    case 'x':    System.out.println("Letra X");

    default:    System.out.println("Letra inválida");
                break;
}
```

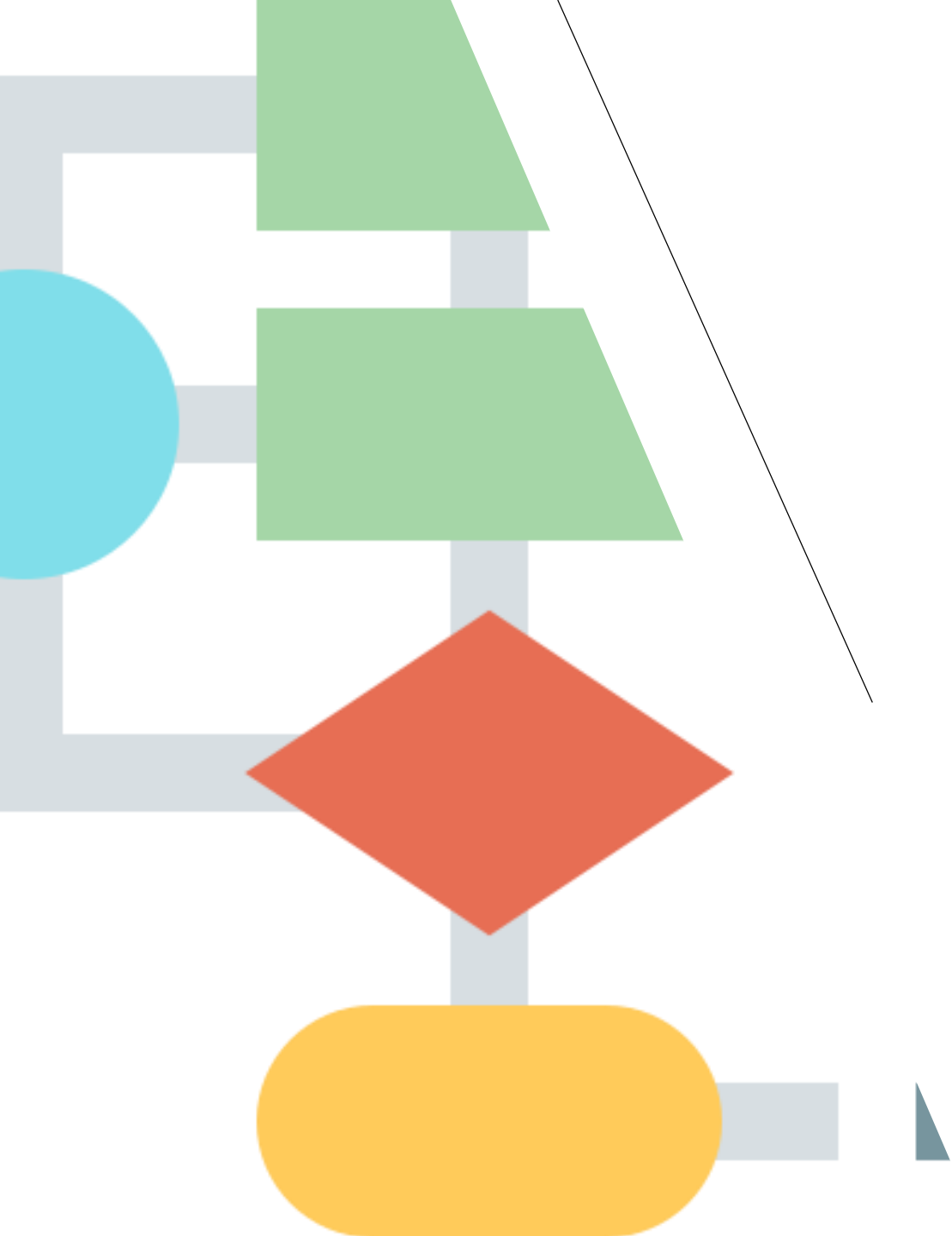


PRATICANDO

PRATICANDO

9. Ler um número real x e imprimi-lo arredondando seu valor para mais ou para menos. Se a parte decimal for menor que 0.5 arredonda para menos. Se for maior ou igual a 0.5 arredonda para mais.
10. Ler um número inteiro n e mais dois números (inferior e superior) que formam um intervalo. Ao final, imprima uma mensagem informando se n está antes, dentro ou depois do intervalo.
11. Ler um caractere op representando uma operação aritmética (+, -, *, /) e em seguida dois números reais a e b . Imprimir a expressão matemática junto com o seu resultado no formato:

$a \text{ op } b = \text{resultado}$



ESTRUTURAS DE REPETIÇÃO

ESTRUTURA DE REPETIÇÃO – WHILE

- Comando while
 - Avalia uma expressão lógica e executa um bloco de comando enquanto ela for verdadeira
 - O bloco é executado ZERO ou mais vezes.
 - Sintaxe:

```
while (expressao_logica)  
    comando;
```

```
while (expressao_logica){  
    bloco_de_comandos  
}
```

ESTRUTURA DE REPETIÇÃO – DO...WHILE

- Comando do...while
 - Avalia uma expressão lógica e executa um bloco de comando enquanto ela for verdadeira.
 - O bloco é executado UMA ou mais vezes.
 - Sintaxe:

```
do{  
    bloco_de_comandos  
} while (expressao_logica);
```

ESTRUTURA DE REPETIÇÃO – FOR

- Comando for
 - Executa um bloco de comandos enquanto uma expressão booleana for verdadeira.
 - É composto de 3 partes. NENHUMA é obrigatória.
 - Sintaxe:

```
for (expr_inicializacao; expressao_logica; expr_incremento)  
    comando;
```

```
for (expr_inicializacao; expressao_logica; expr_incremento){  
    bloco_de_comandos  
}
```

ESTRUTURA DE REPETIÇÃO – FOR

- Comando for

- A execução do for se dá da seguinte forma:

1. Executa a expressão de inicialização
2. Testa a expressão lógica. Se for FALSA termina o for
3. Executa o bloco de comandos
4. Executa a expressão de incremento
5. Volta para o passo 2

```
for (expr_inicializacao; expressao_logica; expr_incremento){  
    comando1;  
    comando2;  
}
```


ESTRUTURA DE REPETIÇÃO – FOR

- Comando for

a) `for (int i = 0; i < 10; i++)
 System.out.println(i);`

b) `for (int i = 0; i < 10;) {
 System.out.println(i);
 i += 2;
}`

c) `int i = 1;
for (; i < 50; i *= 2)
 System.out.println(i);`

d) `int i = 20;
for (; i >= 0;) {
 System.out.println(i);
 i--;
}`

e) `for (int i = 0; ; i++)
 System.out.println(i);`

f) `int i = 0;
for (; ; i++)
 System.out.println(i);`

g) `int i = 0;
for (; ;)
 System.out.println(i);`

ESTRUTURA DE REPETIÇÃO – FOR E WHILE

- Os seguintes comandos for e while são equivalentes:

```
for (expr_inicializacao; expressao_logica; expr_incremento){  
    comando1;  
    comando2;  
}
```

```
expr_inicializacao;  
while (expressao_logica){  
    comando1;  
    comando2;  
    expr_incremento;  
}
```

ESTRUTURA DE CONTROLE EM REPETIÇÃO – BREAK

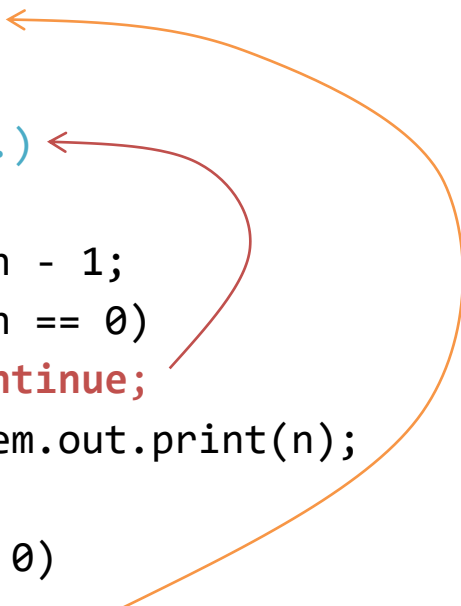
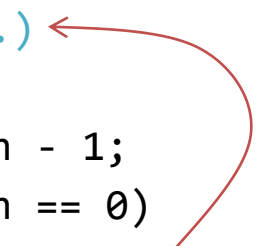
- O comando break força a saída do loop mais interno de um comando de repetição (while, do...while ou for) ou em um comando switch.
- Exemplo:

```
int n;  
  
while(...)  
{  
    for (...)  
    {  
        n = n - 1  
        if (n == 0)  
            break;  
        n++;  
    }  
    System.out.println(n);  
}
```

Sai do for, que é o comando de repetição mais interno.
Não executa o **n++**

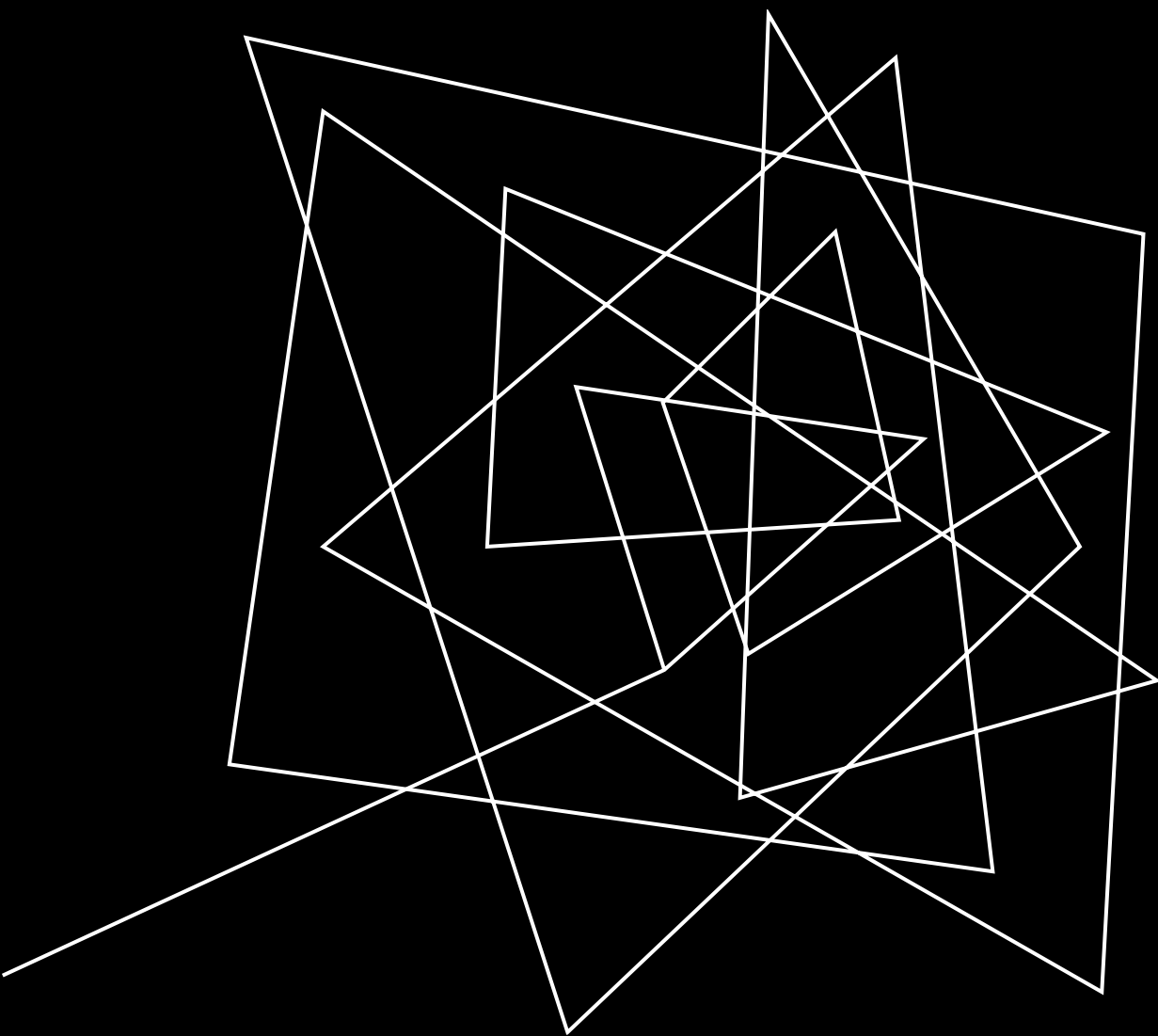
ESTRUTURA DE CONTROLE EM REPETIÇÃO – CONTINUE

- O comando continue força o início da próxima interação do loop mais interno de um comando de repetição (while, do..while ou for).
- Exemplo:

```
int n;  
while(...)   
{  
    for (...)   
    {  
        n = n - 1;  
        if (n == 0)  
            continue;  
        System.out.print(n);  
    }  
    if (n < 0)  
        continue;  
    System.out.print(n);  
}
```

Para o comando **for** reinicia o loop executando a expressão de incremento e depois testando a expressão lógica.

No **while** e **do..while** reinicia o loop testando a expressão lógica.



PRATICANDO

PRATICANDO

12. Ler dois números inteiros (a e b) e imprimir os pares no intervalo $a..b$, além da soma e da média desses números. Caso a seja maior que b , imprima os números no intervalo $b..a$.
13. Ler um número de alunos n . Em seguida, ler as notas dos n alunos e imprimir, ao final, a média da turma.
14. Ler notas de alunos até que o usuário digite -1 . Ao final imprimir a quantidade de alunos, a média da turma, a maior nota e a menor nota.
15. Ler caracteres até que o usuário digite '.' (ponto). Ao final imprimir: a quantidade de vogais, a quantidade de dígitos e a quantidade dos demais caracteres.