Linguagem de Programação

JAVA – *Sintaxe Básica*

Profa. Joyce Miranda

Variáveis Primitivas

- Tópicos
  - declaração, atribuição de valores, casting e comparação de variáveis;
  - controle de fluxo através de if e else;
  - instruções de laço for e while, controle de fluxo com break e continue.

Estrutura do código JAVA

Primeiro Programa em Java

Arquivo: PrimeiroPrograma.java

```
public class PrimeiroPrograma {
 public static void main( String[] args ) {
   System.out.println( "Meu primeiro programa em Java" );
 }
}
```

Compilando o código-fonte:

```
javac PrimeiroPrograma.java
```

Executando o programa:

```
java PrimeiroPrograma
```

Saída gerada:

```
Meu primeiro programa em Java
```

Estrutura do código JAVA


```
public class Dog{

}
```
classe

```
public class Dog {
   void bark( ) {

}
}
```
método

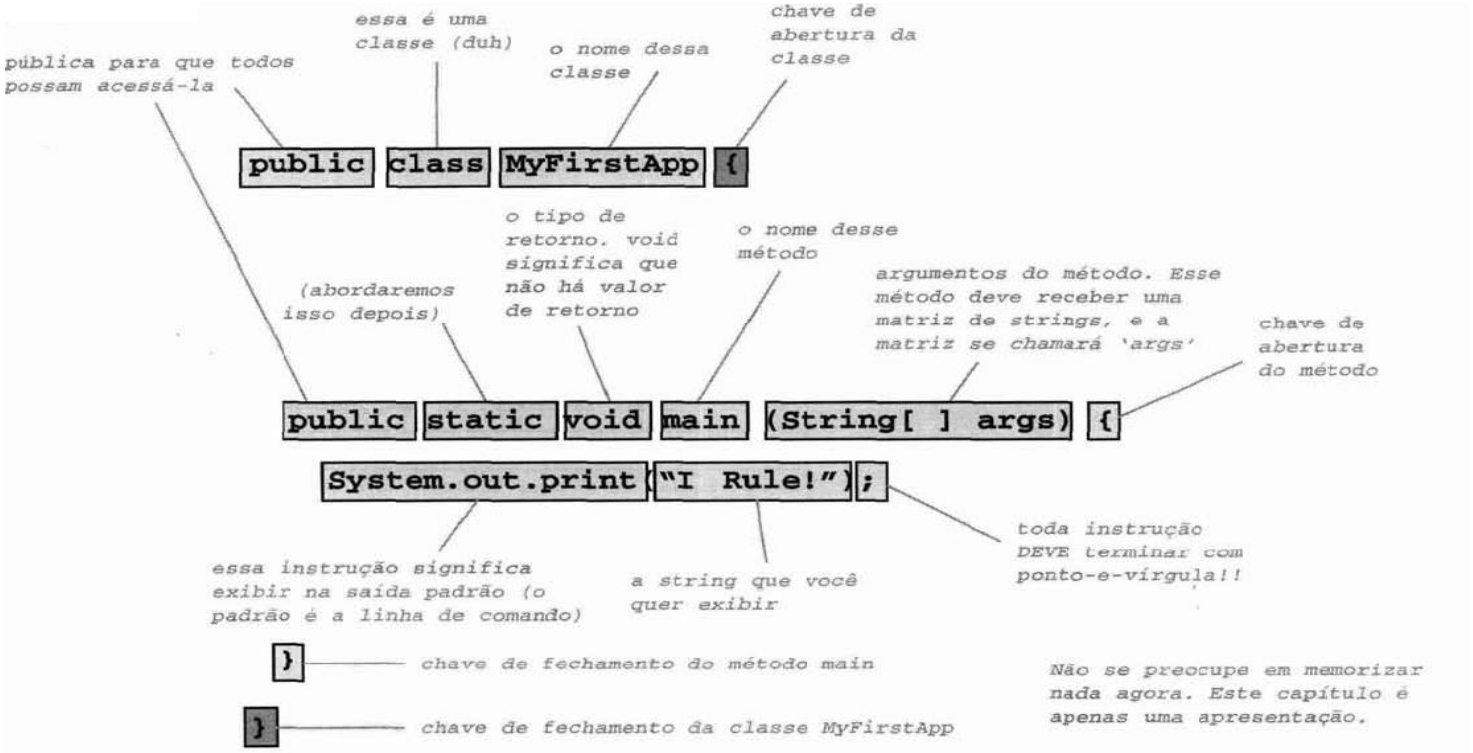
```
public class Dog {
   void bark( ) {

      instrução1;
      instrução2;

   }
}
```
instruções

Estrutura do Código

- Todo aplicativo JAVA precisa:
  - Ter pelo menos uma classe JAVA
  - Um método *main*



Variáveis Primitivas

- Declaração de Variáveis
  - **tipoDaVariavel** nomeDaVariavel;
    - Ex: **int** idade;
- Atribuição de Valores
  - idade = 32;
- Comentários em Java
  - **//** para comentar até o final da linha
  - **/* */** para comentar o que estiver entre eles

```
/* comentário daqui,
ate aqui */

//uma linha de comentário sobre a idade
int idade;
```

Variáveis Primitivas

- Utilização do valor

```
//declara a idade
int idade;
idade = 15;

// imprime a idade
System.out.println(idade);
```

- Inicialização de valor por praticidade

```
int idade = 15;
```

Tipos Primitivos e Valores

- Tipos utilizados e seu respectivo tamanho

TIPO	TAMANHO
boolean	1 bit
byte	1 byte
short	2 bytes
char	2 bytes
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes

Operadores

- Operadores Unários

Incremento e Decremento: ++ e --

```
int a = 0;
int b = a++;    // incrementado depois de atribuir
int c = ++a;    // incrementado antes de atribuir
b = a--;        // decrementado depois de atribuir
c = --a;        // decrementado antes de atribuir
```

Mais e Menos Unário: + e -

```
int x = +3;     // x recebe o positivo 3
x = -x;         // x recebe -3, neste caso
```

Inversão de Bits: ~

```
int i = ~1;     // i = -2  (os bits foram invertidos)
```

Complementar booleano: !

```
boolean falsidade = ! (true);     // inverte o valor booleano
```

Conversão de Tipos: (tipo)

```
double d = 1.99;
int i = (int) d;   // converte de double p/ int (perda de precisão)
```

Operadores

- Operadores Aritméticos

Multiplicação e Divisão: * e /

```
int um = 3 / 2;                     // divisão de inteiros gera um inteiro
float umEmeio = (float) 3 / 2;      // ocorre promoção aritmética para float
double xyz = umEmeio * um;          // ocorre promoção aritmética para float
```

Módulo: %

```
int resto = 7 % 2;                  // resto = 1
```

Adição e Subtração: + e -

```
long l = 1000 + 4000;
double d = 1.0 - 0.01;
```

Concatenação:

```
long var = 12345;
String str = "O valor de var é " + var;
```

Operadores

- Operadores de Comparação

Comparação ordinal: >, >=, < e <=

Compara tipos primitivos numéricos e o tipo char.

```
boolean b = ( 10 < 3 );
boolean w = (x <= y);
if( x >= y ) { }
```

Comparação de Igualdade: == e !=

Comparam tipos primitivos, valores literais e referências de objetos.

```
if( abc == 10 ) { }
boolean b = ( xyz != 50 );
if( refObj1 == refObj2 ) { }
```

Operadores

- Operadores de Atribuição

```
int i = 10;

int dois = 1;
dois += 1;        // dois = dois + 1;

int cinco = 7;
cinco -= 2;       // cinco = cinco - 2;

int dez = 5;
dez *= 2;         // dez = dez * 2;

int quatro = 12;
quatro /= 3;      // quatro = quatro / 3;
```

Operadores

- Operadores Lógicos
  - (e) &&
  - (ou) ||
  - (não) !

Casting e Promoção

- Incompatibilidade entre valores

```
double d = 3.1415;
int i = d; // não compila
int i = 3.14;
double d = 5; // ok, o double pode conter um número inteiro
int i = d; // não compila
```

Casting e Promoção

- Para realizar atribuições com tipos incompatíveis sem que haja o erro de compilação, é preciso ordenar que o valor que será recebido seja moldado (*casted*) com o tipo da variável que irá recebê-lo.
- Ex.

```
double d3 = 3.14;
int i = (int) d3;
```

Casting e Promoção

- Deve ser **explícito** quando for de um tipo maior para um menor (*narrowing*)
- Pode ser **implícito** ou **explícito** de um tipo menor para um maior (*widening*) Ex.

PARA: / DE:	byte	short	char	int	long	float	double
byte	----	*Impl.*	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
short	(byte)	----	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
char	(byte)	(short)	----	*Impl.*	*Impl.*	*Impl.*	*Impl.*
int	(byte)	(short)	(char)	----	*Impl.*	*Impl.*	*Impl.*
long	(byte)	(short)	(char)	(int)	----	*Impl.*	*Impl.*
float	(byte)	(short)	(char)	(int)	(long)	----	*Impl.*
double	(byte)	(short)	(char)	(int)	(long)	(float)	----

Tipos Primitivos e Valores

- Exercícios
  - Fazer um algoritmo que calcule a média aritmética de três notas.
    - N1 = 10; N2 = 6.5; N3 = 8.5
  - Fazer um algoritmo que troque os valores entre duas variáveis.
    - A = 100; B = 1000
  - Fazer um algoritmo que transforme um valor de meses em anos.
    - QtdMeses = 204;
  - Fazer um algoritmo que calcule a área de um:
    - Círculo: Pi * r^2

Controles de Fluxo do Programa

- If-Else

```
if (condicaoBooleana) {
    codigo;
}
```

  - Condição Booleana: qualquer expressão que retorne **true** ou **false;**
  - Para isso deve-se usar os operadores de comparação (>,<,==,!=);
  - A cláusula "***else***" é utilizada para indicar o comportamento que deve ser executado no caso da expressão booleana ser falsa.

Controles de Fluxo do Programa

- If-Else

```
public class ClausulaIf {
  public static void main( String[] args ) {
   int idade = 20;
   if( idade <= 12 ) {
     System.out.println( "Criança" );
   }
   if( idade > 12 && idade <= 19  ) {
     System.out.println( "Adolescente" );
   }
   if( idade > 19 && idade <= 60  ) {
     System.out.println( "Adulto" );
   }
   if( idade > 60 ){
     System.out.println( "Idoso" );
   }
  }
}
```

Controles de Fluxo do Programa

- If-Else

```
public class ClausulaIf {
  public static void main( String[] args ) {
   int idade = 20;
    if( idade <= 12 ) {
      System.out.println( "Criança" );
    }
    else if( idade <= 19  ) {
      System.out.println( "Adolescente" );
    }
    else if( idade <= 60  ) {
      System.out.println( "Adulto" );
    }
    else {
      System.out.println( "Idoso" );
    }
  }
}
```

Controles de Fluxo do Programa

- While
  - É um comando usado para fazer um laço (loop), isto é, repetir um trecho de código algumas vezes.
  - A idéia é que esse trecho de código seja repetido enquanto uma determinada condição permanecer verdadeira.

```
public class LacoWhile {
  public static void main( String[] args ) {
    int i = 0;
    //laço while() com bloco de código definido
    while( i < 10 ) {
      System.out.println( "Linha: " + i );
      i++;
    }
  }
}
```

Controles de Fluxo do Programa

- Do-While
  - É um comando usado para fazer um laço (loop), isto é, repetir um trecho de código algumas vezes.
  - O bloco é executado ao menos um vez. Após a primeira repetição é que a expressão é avaliada.

```
public class LacoWhile {
  public static void main( String[] args ) {
    int i = 0;
    //laço do / while() com bloco de código definido
    do {
      System.out.println( "Linha: " + i );
      i++;
    } while( i < 10 );
  }
}
```

Controles de Fluxo do Programa

- For
  - É um comando usado para fazer um laço (loop), isto é, repetir um trecho de código algumas vezes.
  - Isola também um espaço para inicialização de variáveis e o modificador dessas variáveis. Isso faz com que fiquem mais legíveis, as variáveis que são relacionadas ao loop.

```
for (inicializacao; condicao; incremento) {
    codigo;
}
```

```
public class LacoFor {
  public static void main( String[] args ) {
    for( int i=0; i < 10; i++ ) {
       System.out.println( "Linha: " + i );
    }
  }
}
```

Controles de Fluxo do Programa

- Break
  - Aborta a execução de um laço, quando executado.

```
public class ClausulaBreak {
  public static void main( String[] args ) {
    char letras[] = { 'A', 'B', 'C', 'D', 'E' };
    int i;
    for( i=0; i<letras.length; i++ ) {
      if( letras[i] == 'C' ) {
        break;
      }
    }
    System.out.println( "Último índice: " + i );
  }
}
```

Controles de Fluxo do Programa

- Continue
  - Ignora a execução dos comandos seguintes do bloco, no laço, quando executado.

```
public class ClausulaContinue {
  public static void main( String[] args ) {
    char letras[] = { 'B', 'X', 'R', 'A', 'S', 'I', 'L' };
    int i;
    for( i=0; i<letras.length; i++ ) {
      if( letras[i] == 'X' ) {
        continue;
      }
      System.out.print( letras[i] );
    }
  }
}
```

Controles de Fluxo do Programa

- Switch – Seleção Encadeada

```
public class ClausulaSwitch {
  public static void main( String[] args ) {
    int numero = 1;
    switch( numero ) {
      case 1 :
        System.out.println( "UM" );
        break;
      case 2 :
        System.out.println( "DOIS" );
        break;
      case 3 :
        System.out.println( "TRES" );
        break;
      default :
        System.out.println( "NENHUM" );
        break;
    }
  }
}
```

Controles de Fluxo do Programa

- Exercícios
  - Imprima o somatório de 1 até 1000.
  - Imprima todos os múltiplos de 3, entre 1 e 100.
  - Imprima os fatoriais de 1 a 5.
    - 1! 2! 3! 4! 5!
    - * O fatorial de um número n é (n * n-1 * n-2 ... até n = 1).

Vetores

- Também são conhecidos como estruturas homogêneas de tamanho fixo.
- Estruturas de dados de acesso aleatório.
- Cada posição de um vetor é identificada unicamente por um valor inteiro positivo, linear e seqüencialmente numerado.

O Java é *zero-based*

vetor[5]: posições de 0 a 4.

Declaração de Vetores

- Na declaração de vetores deverão ser fornecidas três informações:
  - o nome do vetor;
  - o número de posições do vetor (seu tamanho);
  - o tipo de dado que será armazenado no vetor.

```
int[]   v   = new int[10];
```

  - **v** é declarado com um vetor de inteiros
  - **new int[10]** aloca espaço na memória e cria efetivamente um vetor de inteiros, de tamanho 10.

Declaração de Vetores

- Um vetor também pode ser criado a partir de uma lista de valores entre { } e separados por vírgula.
  - Definido através de uma tupla

```
{ valor, valor, ... , valor }
```

```
int[] primos = { 2,3,5,7,11,13,17,19 };
char[] dd = { 'd','s','t','q','q','s','s'};
String[] meses = {"jan","fev","mar","abr" };
```

Atribuição em Vetores

- Tendo criado um vetor, o acesso aos seus elementos é feito a partir da sua posição, ou índice, no vetor.
  - Lembrando que: $0 \leq i \leq (tam\text{-}1)$

```
int x[] = new int[5];
//atribuição
x[0] = 1;
x[1] = 2;
x[2] = x[1] * 2;
x[3] = x[0];
x[4] = x[2] + x[3];
//escrita
for(int i=0; i <x.length;i++){
    System.out.println("x["+i+"] = " + x[i]);
}
```

Percorrendo Vetores

- Todo vetor em Java tem o atributo **length** que define o seu número de elementos.

```
for(int i=0; i <x.length;i++){
     System.out.println("x["+i+"] = " + x[i]);
}
```

Percorrendo Vetores

- ***Enhanced for***

```
class AlgumaClasse {
    void imprimeArray(int[] array) {
        for (int x : array) {
            System.out.println(x);
        }
    }
}
```

```
int[] array = new int[10];
for (int aux : array) {
    System.out.println(aux);
}
```

Matrizes

- São variáveis indexadas com duas dimensões.
- A declaração de uma matriz na linguagem JAVA é a seguinte:

```
tipo [][] nomeDaMatriz

int [][] matrizDeInteiros;
```

- Para se criar efetivamente uma matriz deve-se utilizar o operador **new.**

```
nomeDaMatriz = new tipo[tamanho1][tamanho2];

matrizDeInteiros = new int[10][5];
```

Matrizes - Atribuição

- Iniciando uma matriz 3x4.

```
int[][] matrizDeInteiros =
{
        { 11, 12, 13, 14 },
        { 21, 22, 23, 24 },
        { 31, 32, 33, 34 },
};
```

```
int mat[][] = new int[3][4];
mat[0][0] = 11;
mat[0][1] = 12;
mat[0][2] = 13;
mat[0][3] = 14;
mat[1][0] = 21;
mat[1][1] = 22;
mat[1][2] = 23;
mat[1][3] = 24;
```

Matrizes

- Declaração em conjunto

```
tipo [][] nomeDaMatriz = new tipo[tamanho1][tamanho2];

int[][]  matrizDeInteiros = new int[10][5];
```

- Percorrendo a matriz

```
for (int i = 0; i < 10; i++)
        for (int j = 0; j < 5; j++)
```

Prática – Vetores & Matrizes

Para Praticar