Universidade de Brasília

Faculdade UnB Gama

Orientação a Objetos

Aula 2 - Instruções condicionais

Daniel Porto

daniel.porto@unb.br

APRESENTAÇÃO

Paradigma de Programação

Programação em Java

Instruções de Controle - Condicionais

PARADIGMA DE PROGRAMAÇÃO

O paradigma de programação está relacionado à **forma de pensar** do programador e como este busca solucionar problemas computacionais.

- Existem vários paradigmas computacionais
- Mostra como o programador analisou e abstraiu sobre o problema a ser resolvido
- O paradigma proíbe ou permite o uso de algumas técnicas de programação
- Cada linguagem de programação atende a pelo menos um paradigma

PARADIGMA DE PROGRAMAÇÃO

Esta disciplina inicia o **estudo do paradigma Orientado a Objeto (O.O.)**, usando para isso a Linguagem **Java**, que possui este paradigma nativo a sua criação

"... qualquer paradigma pode ser implementado em qualquer linguagem"

E

"uma linguagem nativa em determinado paradigma não significa que ele foi usado na elaboração de seus programas..."

PARADIGMA DE PROGRAMAÇÃO

Deve ser observado que este paradigma (O.O.) não exclui o Estruturado, estando bem claro que ambos **trabalham juntos na criação de códigos** (programas), uma vez que a lógica embutida nos objetos segue o pensamento estruturado (condicionais, repetições, etc.

ESTRUTURADO

- Emprega instruções estruturadas na solução de seus problemas
- Uso da modularização em seu código

ORIENTADO OBJETO

- Compreende o problema como uma coleção de objetos interagindo por meio de mensagens
- Objetos são estruturas de dados contendo lógicas

ESTRUTURA DO PROGRAMA

No material de aula anterior foi elaborado um algoritmo, programa em C e em Java, porém algumas palavras reservadas e instruções em Java foram usadas sem esclarecimentos formais (conteúdo de aula).

```
1   /**
2     * Síntese
3     *    Objetivo: calcular a média entre 3 alturas de pessoas
4     *    Entrada:  sem entrada (só atribuições)
5     *    Saída:    média das alturas
6     */                                              // ↑ Síntese do problema
7      public class PrimeiroExercicio {                // ↓ Classe Java
8           public static void main(String[] args) {  // → Método Main
9           // Declarações
10          final int QTDE = 3;  // constante de quantidade de pessoas
11          float altura1, altura2, altura3, mediaAlturas;
12          // Instruções
13          altura1 = 1.58F;                    // F → indicador de valor float
14          altura2 = 2.07F;
15          altura3 = 0.55F;
16          mediaAlturas = (altura1 + altura2 + altura3) / QTDE;
17          System.out.println("Média das alturas = " + mediaAlturas);
18          }// termina o método main()
19     }   // encerra a descrição da classe
```

ANALISANDO O PRIMEIRO PROGRAMA

public - qualificador ou modificador de acesso a variáveis, métodos e classes.

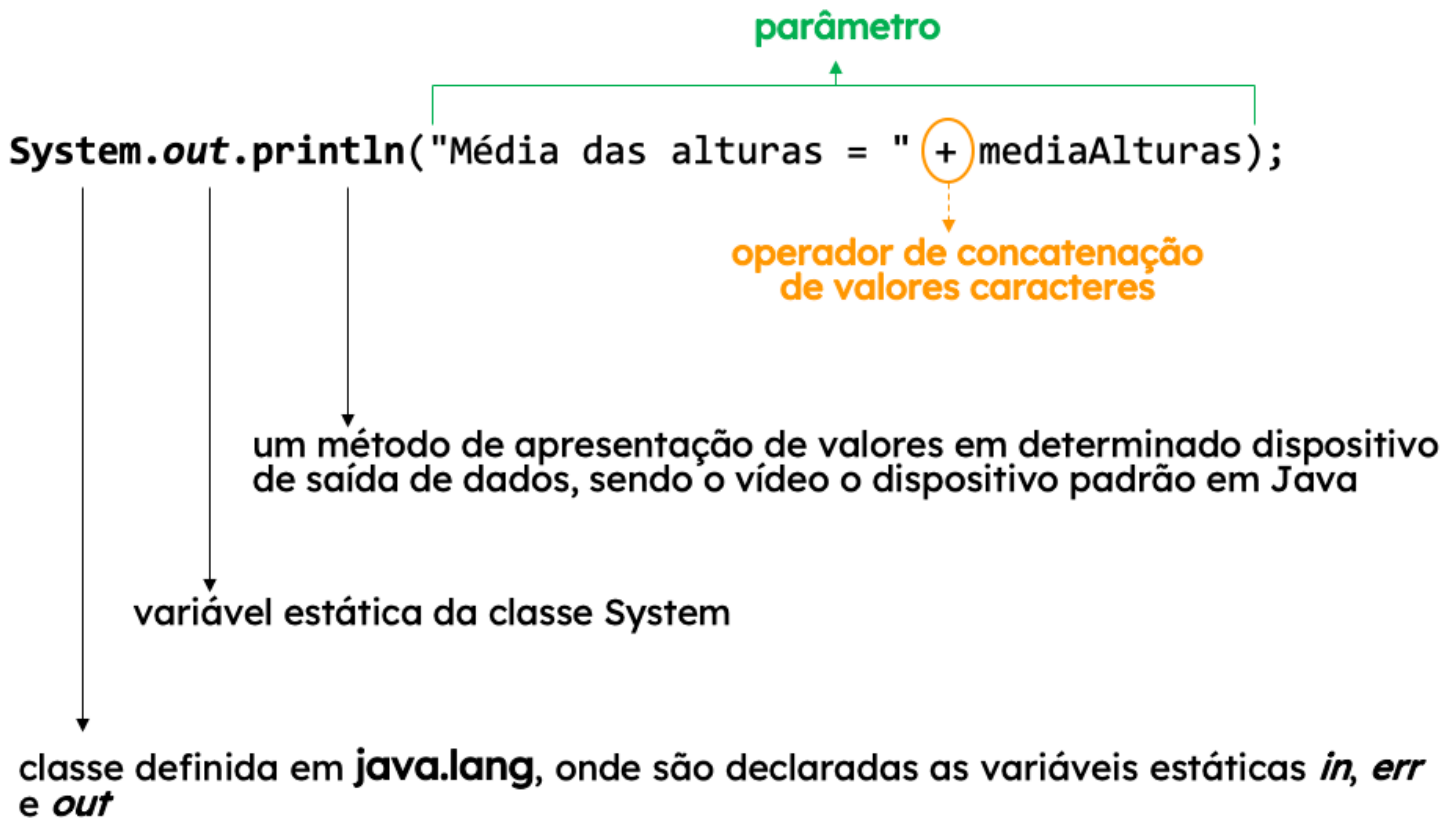
class - inicia todo programa em Java, definindo seus campos (variáveis) e métodos. Sendo uma classe executável deverá possuir o método main(), a partir de onde esta classe será executada quando acionada.

static - outro qualificador de acesso que indica o compartilhamento entre todos os objetos que são criados por uma classe.

void - expressão que indica que um método não retornará um valor (procedimento).

String[] args - único argumento do método principal main() formado por um vetor de **String** que referência todos os parâmetros inseridos na linha de comando, durante o acionamento de um programa (classe) Java.

ANALISANDO O PRIMEIRO PROGRAMA



Digitação rápida: sysout + Ctrl + barra de espaço no corpo do programa

ESTRUTURA DO PROGRAMA

É possível usar recursos e definições de outras classes Java através da diretiva import. Esta diretiva possibilita acesso a outras classes como se fossem bibliotecas (situação bem comum em várias linguagens de programação).

Em Java, essas ”bibliotecas” correspondem aos pacotes (package):

- Correspondem ao que seriam diretórios que armazenam uma ou mais classes similares ou relacionadas
- Identificadores dos pacotes são totalmente em letras minúsculas, por exemplo:
  - java.util.Date → classe Date do pacote java.util (localizada na pasta java/util)
  - java.awt.* → todas as classes do pacote java.awt

* = Todas as classes (curinga)

ESTRUTURA DO PROGRAMA

A linguagem Java possui uma infinidade de recursos para interação com seus programas, porém este conteúdo estará voltado ao ensino-aprendizagem mais convencional, usando classes existentes em seus pacotes padrões, como a saída de dados proporcionada pelos métodos tradicionais existentes na classe **System** implementada em **java.lang**.

Por se tratar de uma classe essencial ao desenvolvimento de aplicações executáveis Java, não existe a necessidade de importar a **java.lang** para utilizar a **System**.

SAÍDA DE DADOS

```
System.out.println("Média das alturas = " + mediaAlturas);
```

- System - aciona classe System do pacote java.lang
- Operador ponto (.) - acessa recurso do componente especificado antes do ponto, respeitando sua hierarquia e disponibilidades de recursos computacionais deste componente (campos ou variáveis, métodos, ...)
- out - variável estática que manipula recursos de saída de dados em Java
- println - método implementado e disponível para acionamento na classe System que apresenta como saída de dados o conteúdo especificado como parâmetro desta instrução (definição entre parênteses e antes do marcador de fim de instrução (;))
- + - operador de concatenação para apresentação dos dados (tudo como caracter, com conversão automática)

SAÍDA DE DADOS

O recurso de saída de dados padrão em Java é o vídeo (monitor), sendo apresentado os dados desejados na instrução do exemplo anterior em uma janela de execução do S.O. (Sistema Operacional) do computador que esta executando este programa.

Geralmente, esta ”janela de execução” também é chamada de **CONSOLE** e indica onde a aplicação Java está sendo executada realmente.

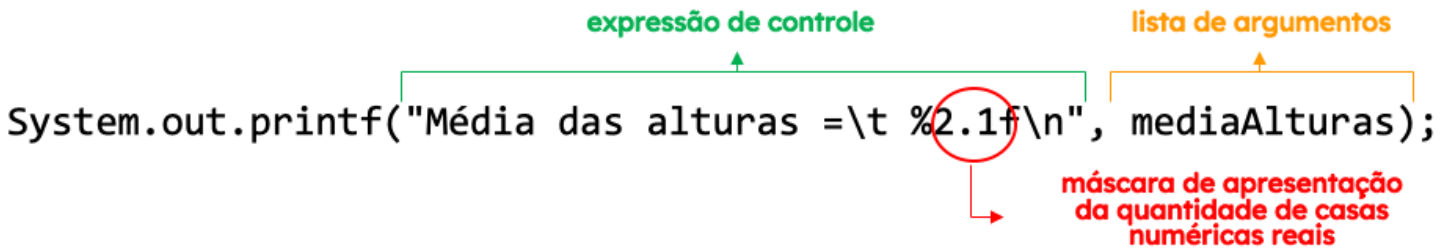
Está disponível em **System** vários métodos de apresentação de dados na console, por exemplo:

- println - apresenta todos os dados referenciados no parâmetro desta instrução e salta uma linha ao seu final
- print - somente apresenta todos os dados referenciados no parâmetro da instrução, sem saltar nenhuma linha ao seu final

SAÍDA DE DADOS

Existem outros métodos e variações, inclusive destes mesmos, implementados em classes Java, porém um pode ser destacado por possibilitar sua apresentação formata:

- printf - instrução muito similar a implementação na Linguagem C, em que são respeitados alguns de seus códigos de formatação na expressão de controle, por exemplo:



Código	Significado	Código	Significado
\t	salto de tabulação	%f	número real float
\n	salta para nova linha	%d	número inteiro
\“	mostra uma aspas	%c	um único caracter
\‘	mostra um apostrofe	%s	cadeia de caracteres (String)
\\	mostra uma barra		

ENTRADA DE DADOS

Na linguagem Java também existem componentes diferentes que podem fazer a entrada de dados, em que valores fornecidos ao computador serão armazenados em uma área de memória.

Entre as possíveis, será iniciado o estudo pela classe que pode ser associada ao conteúdo estudado na disciplina anterior (APC, CB ou ICC – estudo da Linguagem C).

Todas as entradas de dados em Java são realizadas por meio de conjunto de caracteres (String), sendo necessária a conversão para seu tipo de dado a ser armazenado.

- Scanner - uma das formas de leitura (entrada) de dados que será usada nesta disciplina, inicialmente
- java.util- esta classe (Scanner) exige a importação do pacote java.util, ou seja:
- import java.util.Scanner; ou import java.util.*;

ENTRADA DE DADOS

Abaixo estão relacionadas algumas das funções de conversão de tipos para classe Scanner.

FUNÇÃO	FUNCIONALIDADE
next()	string com uma única palavra
nextLine()	string com uma ou várias palavras (necessita limpeza de buffer)
nextByte()	entrada de um valor inteiro - byte
nextShort()	entrada de um valor inteiro - short
nextInt()	entrada de um valor inteiro - int
nextLong()	entrada de um valor inteiro - long
nextBoolean()	entrada de um valor lógico - boolean
nextFloat()	entrada de um valor real - float
nextDouble()	entrada de um valor real - double
:	:

Após criar um objeto Scanner em seu IDE, digite o nome deste novo objeto e o operador ponto. Aguarde a apresentação dos possíveis acessos deste objeto ou pressione Ctrl + Espaço para suas opções serem mostradas (API).

ENTRADA DE DADOS

Observe o programa de exemplo abaixo com instruções de entrada e saída de dados.

```java
/** Síntese
  *    Objetivo: armazenar primeiro nome de uma pessoa
  *    Entrada:  primeiro nome
  *    Saída:    confirmação com nome cadastrado
 */

import java.util.Scanner; // importa classe (biblioteca)
public class Nome {
   public static void main(String[] args) {
       // Declarações
       String primeiroNome;  // variável do tipo String
       Scanner ler = new Scanner(System.in); // NEW → cria um novo objeto
       // Instruções
       System.out.print("Digite o primeiro nome:\n");
       primeiroNome = ler.next();
       System.out.println("\n\nNome:\t" + primeiroNome); // + → Concatenação
   }   // termina o método main()
}  // encerra a descrição da classe
```

ENTRADA DE DADOS

A limpeza das mensagens na console não é algo trivial, mas é possível satisfazer a visão do usuário através da **artimanha de saltar a quantidade de linhas** referentes a área de execução do programa, ou seja:

- Por exemplo, a janela de execução padrão de um programa em C possui 25 linhas, onde um salto de 30 linhas deixaria o usuário com a impressão de que a janela foi limpa, caso não existissem funções estruturadas que a realizassem (clrscr ou cls)
- Na console de execução Java poderia ser realizada uma instrução de saída de dados com 30 saltos de linha, em tempo de execução, ou uma repetição que propiciasse esta quantidade de salto (será estudado mais a frente as instruções de repetição em Java)

```
System.out.println("\n\n\n\n\n\n\n..."); // 30 \n nesta instrução
```

FUNÇÕES MATEMÁTICAS

Algumas funcionalidades matemáticas mais comuns podem ser realizadas por meio da classe (biblioteca) denominada **Math**, disponível na **java.lang**

FUNÇÃO	FUNCIONALIDADE
abs(x)	obtém valor absoluto (módulo)
cos(x)	calcula o co-seno de x, estando x em radianos
log(x)	obtém o logaritmo natural de x
pow(x,y)	calcula a potência de x elevado a y
sin(x)	calcula o seno de x, estando x em radianos
sqrt(x)	calcula a raiz quadrada de x
tan(x)	calcula a tangente de x, estando x em radianos
floor(x)	arredonda número real para baixo
ceil(x)	arredonda número real para cima
min(x,y)	menor entre dois números (double, float, int, long misturados)
max(x,y)	maior entre dois números (double, float, int, long misturados)
toDegress(x)	converte x de radianos para graus
toRadians(x)	converte x de graus para radianos
PI	retorna o valor de π (PI)
:	:

FUNÇÕES MATEMÁTICAS

Observe o exemplo usando algumas funções matemáticas

```
1   /** Síntese
2     *    Objetivo: calcular o valor do seno
3     *    Entrada:  grau
4     *    Saída:    seno referente ao grau fornecido
5     */
6
7    import java.util.Scanner;
8    import java.text.DecimalFormat;   // classe de formatação
9    public class Calculo {
10      public static void main(String[] args) {
11          // Declarações
12          double grau,resultado;
13          Scanner ler = new Scanner(System.in);
14          DecimalFormat mascara;
15          mascara = new DecimalFormat("0.000");
16          // Instruções
17          System.out.print("Digite o grau desejado:\n");
18          grau = ler.nextDouble();  // interessante validar
19          resultado = Math.sin(Math.toRadians(grau));
20          System.out.println("\n\n\nSeno=\t" + mascara.format(resultado));
21      }
22  }
```

FUNÇÕES MATEMÁTICAS

O cálculo do seno, cosseno ou tangente exige o parâmetro em radianos para uso dos métodos (funcionalidades) disponíveis na classe **Math**, sendo primeiro convertido o grau fornecido pelo usuário em radianos através do método toRadians().

O resultado do método toRadians() é aplicado no cálculo do seno realizado pelo método sin() da classe **Math** (Math.sin()).

Não é necessária a importação da classe que contém **Math** porque esta é a java.lang, porém a identificação da classe e seu método que será acionado é obrigatória com o uso do operador ponto:

Math.sin() ou Math.toRadians()

FUNÇÕES MATEMÁTICAS

O exemplo anterior também utiliza uma classe específica para formatação ou máscara de alguns tipos de valores:

- Realização da importação de outra classe
  - import java.text.**DecimalFormat**;
- Criação de um objeto do tipo **DecimalFormat**
  - DecimalFormat mascara; → objeto mascara
- Alocação de espaço de memória para o objeto mascara
  - mascara = new DecimalFormat(”0.000”);
- Uso do objeto mascara para formatar a apresentação do resultado esperado
  - println(”\n\n\nSeno=\t” + mascara.format(resultado));

WWW.ECLIPSE.ORG

EXERCÍCIO PROPOSTO

1) Faça um programa que solicite ao usuário um valor percentual a ser calculado sobre o piso salarial da categoria. Este piso deverá estar especificado em uma constante com valor de R$1000,00 (mil reais) e será usado para o cálculo do percentual informado. O programa deverá calcular este percentual e apresentá-lo de maneira formatada com no mínimo uma casa inteira e duas fracionárias (depois da vírgula).

INSTRUÇÕES DE SELEÇÃO (CONDICIONAL)

```
if (<condição>)
        <instrução>;
```

A <instrução> será executada se a <condição> for verdadeira.

Caso seja mais que uma instrução a ser executada deverá ser criado um bloco de instruções (marcador de chaves).

```
if (<condição>) {
        <instrução_1>;
        <instrução_2>;
                :
        <instrução_n>;
}
```

INSTRUÇÕES DE SELEÇÃO (CONDICIONAL)

```
if (<condição>)
        <instrução_1>;
else
        <instrução_2>;
```

OU

```
if (<condição>) {
        <instrução_V>;
} else {
        <instrução_F>;
}
```

A <instrução_1> será executada somente se a <condição> for verdadeira, senão, quando a <condição> for falsa, será executada a <instrução_2>.

Quando a <instrução_1> ou <instrução_2> ou ambas as instruções (1 e 2) possuírem, cada uma, mais que uma instrução a ser executada será necessário a criação de bloco de instruções individuais para cada possível situação (verdadeira ou falso), com chaves para o bloco verdadeiro e outras chaves para o bloco do falso (else).

INSTRUÇÕES DE SELEÇÃO (CONDICIONAL)

Um condicional aninhado é simplesmente um if dentro de outro if externo.

O único cuidado que deve-se ter é com a identificação de qual else pertence determinado if .

O emprego de técnicas de texto estruturado facilitam a correta identificação da estrutura condicional aninhada.

```
if (<condição_1>)
{
        if (<condição_2>)
                <instrução_1>;
        else
                <instrução_2>;
}
else
{
        if (<condição_3>)
                <instrução_3>;
        else
                <instrução_4>;
}
```

INSTRUÇÕES DE SELEÇÃO (CONDICIONAL)

Condicional Ternário

Este condicional não atende a uma gama grande de casos, mas pode ser usado para simplificar expressões condicionais.

<condição> ? <instrução_1> : <instrução_2>;

Exemplo: transformação para instrução condicional:

```
if (nota < 5)
        System.out.println(“Reprovado“);
else
        System.out.println(“Aprovado“);
```

OU

```
System.out.println( (nota < 5) ? ”Reprovado” : ”Aprovado”);
```

Várias são as formas de combinação e aninhamento das instruções condicionais, inclusive envolvendo ternários, como será estudado e praticado mais adiante.

INSTRUÇÕES DE SELEÇÃO (CONDICIONAL)

Condicional de Múltipla Escolha

Instrução de tomada de decisão mais apropriada ao teste condicional de uma variável em relação a diversos valores pré-estabelecidos (somente analisa a igualdade).

- Similiar ao *if – else – if*, em que a diferença fundamental é que ele não aceita expressões, mas atributos do tipo byte, short, int ou char que serão analisados (igualdade)
- A opção default só será executada se não apresentar igualdade a nenhum dos valores constantes especificados na instrução (case)
- O tipo de dado a ser analisado pela switch mais segura deve ser:
  - byte, short, int e char

INSTRUÇÕES DE SELEÇÃO (CONDICIONAL)

Instrução break

Esta instrução causa a interrupção (parada ou quebra) imediata de alguns blocos de execução, alterando seu funcionamento sequencial, como é o caso no **switch**.

O uso do **break** no bloco do switch interrompe a continuidade das verificações condicionais sobre seus valores constantes, tornando-se um comando interessante dentro deste tipo de estrutura, pois salta para o final do bloco e continua a execução do programa.

Exemplo:

```
1  switch (valor) {
2      case 1: System.out.println("Valor = 1");
3          break;
4      case 2: System.out.println("Valor = 2");
5          break;
6      default: System.out.println("Valor diferente!");
7  }   // Não se coloca break na opção default
```

EXERCÍCIOS DE FIXAÇÃO

2) Usando o condicional ternário desenvolva um programa que solicite o sexo de uma pessoa e o mostre por extenso em maiúsculo, após três linhas abaixo da solicitação e no meio horizontal da linha na CONSOLE.

3) Elabore um programa que indique o nome da capital do centro-oeste brasileiro que possui o DDD digitado pelo usuário (61-Brasília, 62-Goiânia, 65-Cuiabá e 67-Campo Grande). Caso o DDD seja válido (maior que 10), mas não esteja entre estes, deverá ser mostrada a mensagem: ”DDD não pertence a capital do centro-oeste brasileiro”.

4) Faça um programa que leia três valores numéricos em um vetor referentes ao peso de elefantes (nunca menores que 5 quilos) e os mostre em ordem crescente. Consulte o material da Aula 5 para uso de vetores (Array) em Java.