

# ORIENTAÇÃO A OBJETOS

## AULA 2

### Controle de Fluxo

Vandor Roberto Vilardi Rissoli



# APRESENTAÇÃO

- Paradigma de Programação
- Programação em Java
- Instruções de Controle
- Introdução de String
- Referências



# 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

Na 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).

```
/**
 * Síntese
 * Objetivo: calcular a média entre 3 alturas de pessoas
 * Entrada: sem entrada (só atribuições)
 * Saída: média das alturas
 */
```

síntese do problema

método main()

```
public class PrimeiroExercicio {
    public static void main(String[] args) {
        // Declarações
        final int QTDE = 3; // constante de quantidade de pessoas
        float altura1, altura2, altura3, mediaAlturas;
        // Instruções
        altura1 = 1.58F; --> indicador de valor float
        altura2 = 2.07F;
        altura3 = 0.55F;
        mediaAlturas = (altura1 + altura2 + altura3) / QTDE;
        System.out.println("Média das alturas = " + mediaAlturas);
    } // termina o método main()
} // encerra a descrição da classe
```

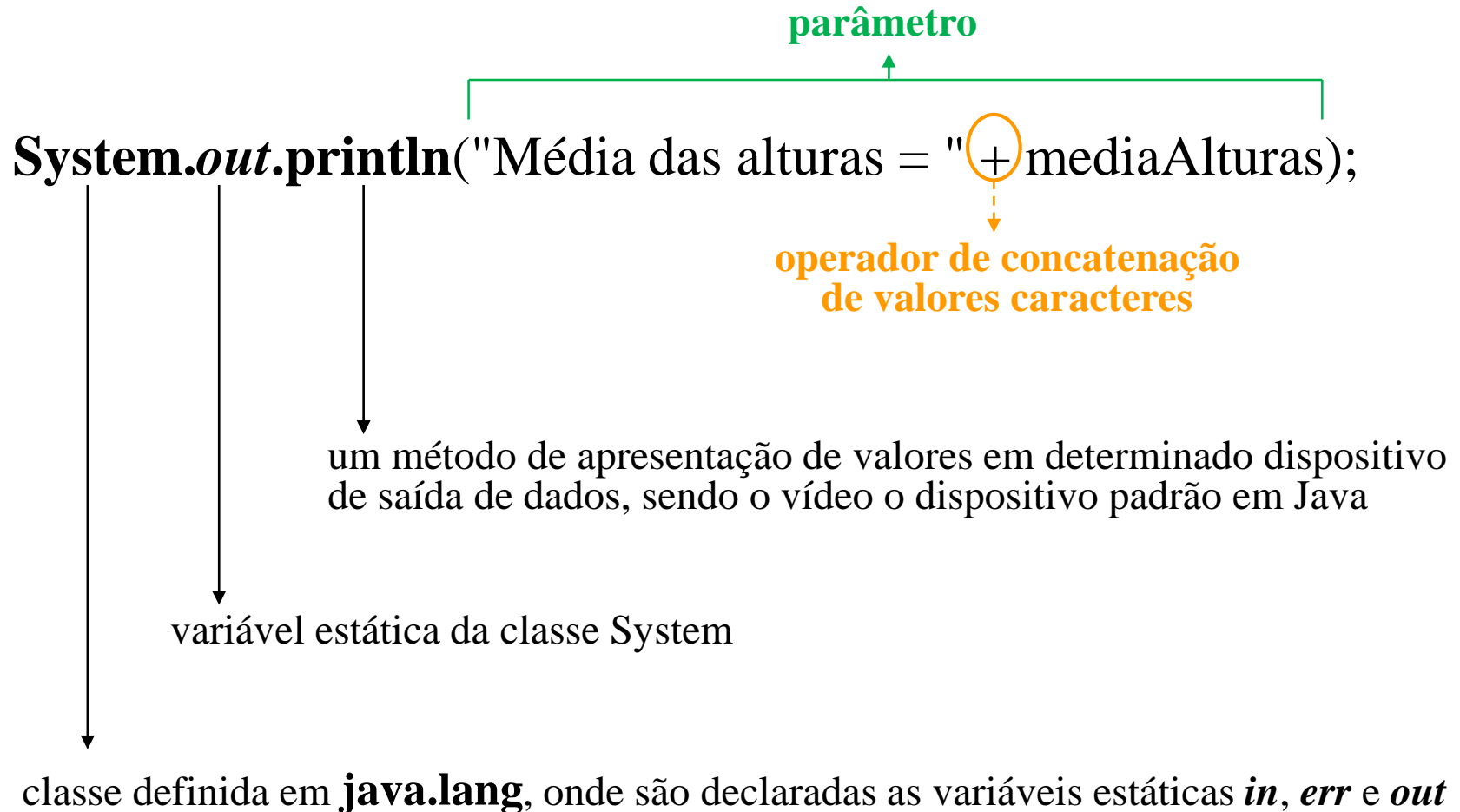
classe Java



# Analizando 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 referencia todos os parâmetros inseridos na linha de comando, durante o acionamento de um programa (classe) **Java**

# Analizando 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).

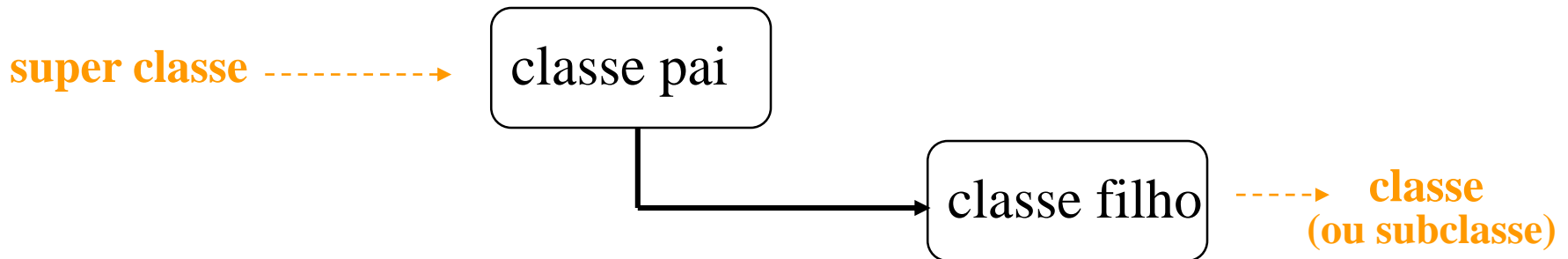
Em Java, essas "bibliotecas" correspondem aos pacotes (*package*):

- 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** (classe pai) para utilizar a **System** (classe filho).



# 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 character, 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.

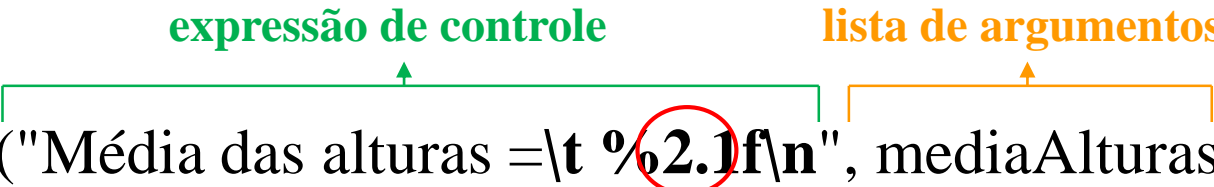
Está disponível em **System** vários métodos de apresentação de dados na console, por exemplo:

- **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:

  
System.out.**printf**("Média das alturas =\t %2.1f\n", mediaAlturas);  
máscara de apresentação da quantidade de casas numéricas reais

<u>Código</u>	<u>Significado</u>
\t	salto de tabulação
\n	salta para nova linha
\“	mostra uma aspas
\‘	mostra um apostrofe
\\	mostra uma barra

<u>Código</u>	<u>Significado</u>
%f	número real float
%d	número inteiro
%c	um único caracter
%s	cadeia de caracteres (String)

# 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**.

<b><u>FUNÇÃO</u></b>	<b><u>FUNCIONALIDADE</u></b>
next()	string com uma única palavra
nextLine()	string com uma ou várias palavras ( <b>necessita limpeza de <u>buffer</u></b> )
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.

```
/** 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);
        // Instruções
        System.out.print("Digite o primeiro nome:\n");
        primeiroNome = ler.next();
        System.out.println("\n\nNome:\t" + primeiroNome);
    } // termina o método main()
} // encerra a descrição da classe
```

 operador de concatenação







# Funções Matemáticas

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

<b>FUNÇÃO</b>	<b>FUNCIONALIDADE</b>
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)
toDegrees(x)	converte x de radianos para graus
toRadians(x)	converte x de graus para radianos
PI	retorna o valor de $\pi$ (PI)
:	:

# Funções Matemáticas

Observe o exemplo usando algumas funções matemáticas

[illegible]

# 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()`



# Formatação ou Máscara de Apresentação

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));
```



# 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)

## Condicionais Simples

```
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)

## Condicional Composta

if(<condição>)		if(<condição>) {
<instrução_1>;		<instruções_V>;
	OU	}
else		else
<instrução_2>;		{
		<instruções_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)

## Condicional Aninhada

- 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: switch (**valor**) {  
    case 1: System.out.println("Valor = 1");  
        **break**;  
    case 2: System.out.println("Valor = 2");  
        **break**;  
    default: System.out.println("Valor diferente!");  
} // 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 4 para uso de vetores (**Array**) em Java.

# Instruções de Repetição

Existem 3 instruções de repetição em Java que são usadas de acordo com a correta necessidade lógica a ser aplicada em determinadas situações, em que as características de cada uma possam ser melhor aproveitadas.

```
for(<inicialização>; <condição>; <incremento>)  
    <instrução>;
```

- ➔ <inicialização> é executada uma única vez, no momento em que o laço será iniciado, podendo conter declarações
- ➔ A sequência de instruções será repetida enquanto a <condição> for **verdadeira**. Quando ela for **falsa** os comandos após o laço serão executados, interrompendo a sua repetição e prosseguindo com execução do programa
- ➔ <incremento> define como será alterada a variável de controle da repetição, cada vez que ele for repetido, sendo executado logo após o fim do bloco de repetição

# Instruções de Repetição

Observe os exemplos:

```
for (int aux = 0; aux < 10; aux++)  
    System.out.println("Valor = " + aux);
```

**Declaração e atribuição inicial**

Similar as instruções anteriores, a necessidade de mais que uma instrução a ser executada durante a repetição *for* exige a criação do bloco de instruções com chaves.

```
for (int aux = 0, int cont = 1; aux <= 10; aux+=2,cont++) {  
    System.out.print("\t" + cont + " Par = ");  
    System.out.println("\t" + aux);  
}
```

A inicialização, condição e incremento podem ser nulos, possibilitando inclusive a elaboração de um laço infinito.

# Instruções de Repetição

Similar as instruções condicionais, também é possível o aninhamento entre as instruções de repetição, por exemplo:

```
for (int aux = 0; aux < 10; aux++)  
    for (cont = 5; cont > 1; cont--) {  
        total = aux * cont;  
        System.out.println("Total = " + total);  
    }
```

- O *for* com controle sobre *cont* não cria variável que deve ter sido declarada antes do laço, assim como total
- No *for* mais interno (controle sobre *cont*) é realizada a operação de decremento ao invés do incremento
- Existe um bloco de instrução somente no *for* mais interno, pois ele possui duas instruções, enquanto o primeiro *for* possui só uma (outro *for*) em seu corpo



# Instruções de Repetição

Uma outra instrução de repetição usa os mesmos elementos do laço *for*, mas distribuídos de forma diferente.

**while** (<condição>)

<instrução>;

- ➔ No *while* a sequência de instruções é repetida enquanto a <condição> for **verdadeira**
- ➔ O laço se repete até que a condição se torne **falsa**, encerrando o laço e continuando a execução do programa
- ➔ O laço *while* é mais apropriado para situações que a repetição possa ser **encerrada inesperadamente**, enquanto o *for* é mais indicado para quantidades de repetições conhecidas (ou definidas).
- ➔ Similar ao *for*, a instrução *while* também pode ser aninhada, ou seja, possuir um *while* dentro de outro, além de possuir um bloco de instruções com uso das **chaves**

# Instruções de Repetição

Observe o trecho de um programa com a repetição *while* como exemplo:

```
public static void main(String[] args) {  
    // Declarações  
    int qtde, aux;  
    Scanner ler = new Scanner(System.in);  
    DecimalFormat casas;  
    casas = new DecimalFormat("00");  
  
    // Instruções  
    System.out.print("Informe a quantidade:\n");  
    qtde = ler.nextInt();  
    aux = 0;  
    while (aux++ < qtde)  
        System.out.println("Valor= " + casas.format(aux));  
} // encerra o método main()
```



# Instruções de Repetição

A última estrutura de repetição cria um ciclo repetitivo até sua condição ser **falsa**. A condição neste laço é avaliada depois da repetição ser executada, ou seja, seu bloco será executado ao menos uma vez para ser verificado (testado).

```
do {  
    <instruções>;  
} while (<condição>);
```

- ➔ As chaves não são sempre necessárias, somente quando existem mais que uma instrução em seu bloco, mas elas favorecem a legibilidade do programa
- ➔ Similar as repetições anteriores (*for* e *while*), esta instrução também aceita o aninhamento de outras repetições, inclusive dela mesmo (uma dentro da outra)
- ➔ Esta instrução também é mais adequada logicamente a quantidade de repetições não definidas ou desconhecidas

# Instruções de Repetição

Observe exemplo com *do...while*:

[illegible]

# Instruções de Repetição

## Instrução *break*

No corpo de uma instrução de repetição o *break* encerra (quebra) o bloco de repetição que esta sendo executado imediatamente, passando a executar as instruções do programa que estão fora da repetição (continua o programa sem repetir mais).

`break;`

➔ Em instruções aninhadas o `break` só afeta o laço que o contém, sendo somente este interrompido.



# Instruções de Repetição

## Instrução *continue*

Este comando permite que se retorne ao início do laço imediatamente e seja verificada a condição de execução da mesma novamente.

Em alguns casos que não exista a necessidade de continuar até o final do bloco de instruções do laço, pode-se usar a instrução `continue`.

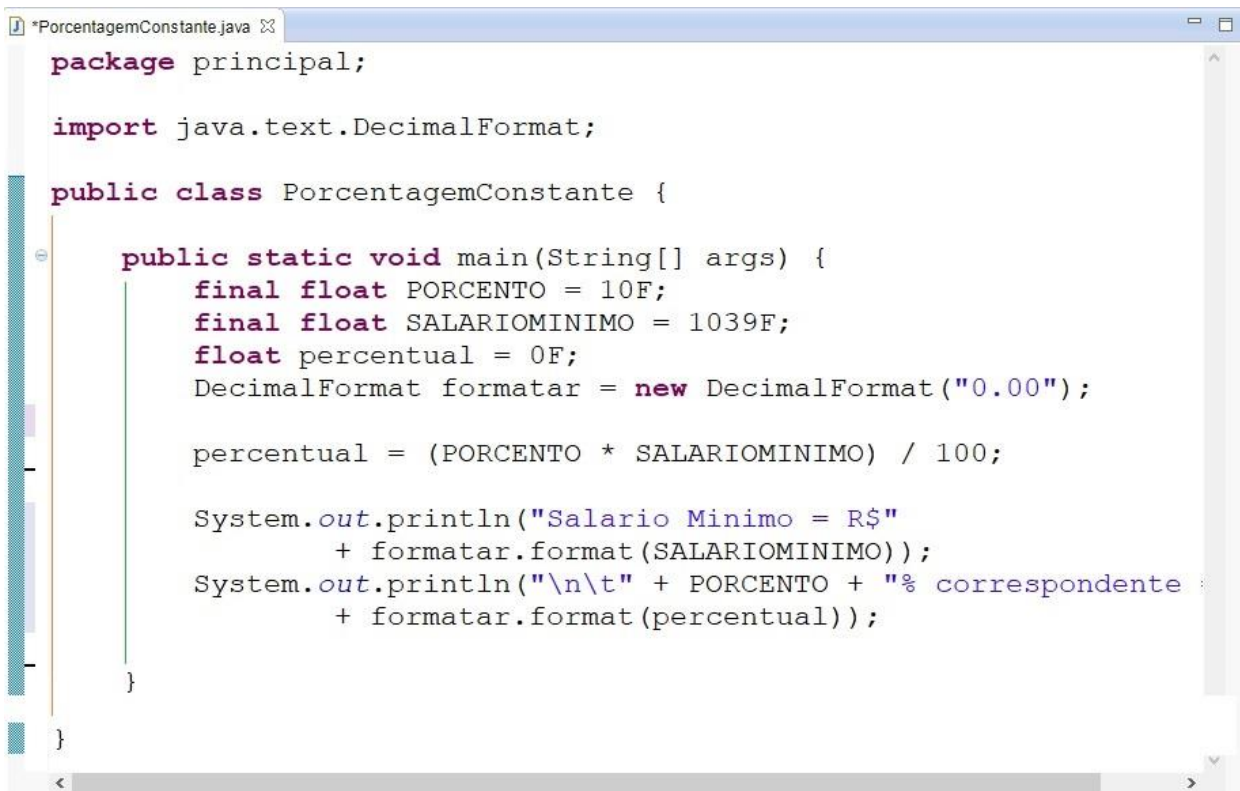
`continue;`

Analise o exemplo a seguir que mostra o funcionamento das instruções `break` e `continue`.



# Instruções de Repetição

As "boas práticas" de programação exigem um código organizado e respeitando as normas para elaboração de um código que seja fácil de entender para qualquer programador.



```
package principal;

import java.text.DecimalFormat;

public class PorcentagemConstante {

    public static void main(String[] args) {
        final float PORCENTO = 10F;
        final float SALARIOMINIMO = 1039F;
        float percentual = 0F;
        DecimalFormat formatar = new DecimalFormat("0.00");

        percentual = (PORCENTO * SALARIOMINIMO) / 100;

        System.out.println("Salario Minimo = R$"
            + formatar.format(SALARIOMINIMO));
        System.out.println("\n\t" + PORCENTO + "% correspondente
            + formatar.format(percentual));
    }
}
```

Algumas teclas de atalho agilizam a organização do código nesse IDE.

**CRTL+SHIF + F**  
arruma endentação

**CRTL +SHIF + C**  
comenta com // todas as linhas selecionadas

**CRTL + SHIF + /**  
comenta como bloco as linhas selecionadas

# Instruções de Repetição

Observe exemplo:

```
/** Síntese
 *   Objetivo: obter um número impar
 *   Entrada:  número inteiro impar
 *   Saída:    número inteiro impar
 */
import java.util.Scanner;
public class Continua {
    public static void main(String[] args) {
        // Declarações
        int numero;
        Scanner ler = new Scanner(System.in);
        // Instruções
        do {
            System.out.println("Digite valor inteiro impar: ");
            numero = ler.nextInt();
            if ((numero % 2) == 0)
                continue;
            else
                break;
            System.out.print("Valor inválido!");
        } while ((numero % 2) == 0);
        System.out.print("Valor Impar = " + numero);
    }
}
```





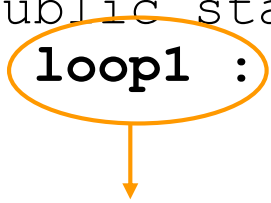
# Instruções de Repetição

## Instrução Rotulada (label)

Observe o trecho do programa Java abaixo e verifique a utilização do recurso de especificação de um rótulo (label) no processamento referenciado pela instrução continue. Este tipo de rótulo/referência também pode ser feito no break.

```
public class Label {  
    public static void main (String args[]) {  
        loop1 : for (int i=1; i < 10 ; i++)  
                for (int j= i+1; j < 6; j++)  
                    if (i == 2) {  
                        System.out.println("Continue i=" + i  
                                           + "\tj=" + j);  
                        continue loop1; // referência ao label  
                    }  
                else  
                    System.out.println("i="+i+"\tj="+j);  
    }  
}
```

**Definição do rótulo (label)**  
**no programa fonte**



# COMPARTILHAR PROJETOS

As atividades de desenvolvimento e entrega (compartilhamento) dos projetos Java, solicitados por esta disciplina, não deverão ser mais somente do arquivo (programa) fonte, mas de todos os aspectos definidos em seu desenvolvimento. Para isso o projeto elaborado será entregue de maneira compactada (formato **zip**), respeitando as seguintes diretrizes para entrega correta:

- 1) Elaborar um projeto Java que solucione o problema (exercício) proposto;
- 2) Fechar o projeto pelo IDE usado e localizar seu diretório (pasta) específico de armazenamento;
- 3) Compactar no formato zip este diretório (pasta);
- 4) Renomear o arquivo compactado para o padrão exigido para entrega;
- 5) Entregar o arquivo compactado contendo todo o projeto elaborado.

# COMPARTILHAR PROJETOS

A utilização (**execução**) dos projetos Java compactados deve seguir as orientações abaixo para ser executado:

- 1) Copiar o arquivo compactado do projeto Java no diretório de trabalho (*workspace*) de seu IDE;
- 2) Descompactar este arquivo exatamente neste diretório de trabalho (*workspace*);
- 3) No menu de barra do IDE escolher a opção de Importar;
- 4) Escolher o tipo de arquivo da opção **General** denominada ***Existing Projects into Workspace*** e pressionar botão ***Next***;
- 5) Indicar o caminho do diretório que foi descompactado para o IDE e deixar selecionado o projeto Java que será importado, pressionado o botão ***Finish***.

Confira que o projeto foi importado e está aberto em seu IDE para ser executado.

➔ A partir destes esclarecimentos todo programa entregue nesta disciplina sempre deve acontecer por seu projeto compactado.

# Exercícios de Fixação

- 5) Faça um programa que analise o conjunto de dados contendo peso e sexo (masculino, feminino) de 20 pessoas (quantidade definida em constante no programa). Este programa deverá mostrar o valor do maior e menor peso informado, a média dos pesos dos homens e o número de mulheres que participaram desta análise.
  
- 6) Solicite ao usuário a quantidade de números inteiros que ele gostaria de informar para o cálculo do fatorial e após validar este número solicite cada um deles e mostre o seu valor fatorial.



# String

Corresponde a um conjunto de caracteres (estrutura de dados homogênea do tipo **char**), sendo em Java uma classe disponível em sua biblioteca padrão (java.lang).

Cada instância desta classe guardará valores entre aspas, por exemplo:

```
String saudar = "Bom dia!";
```

↓                      ↓                      ↓

classe Java                      instância String                      conteúdo desta instância  
(identificador)

A criação de um novo objeto String também pode ser realizada por meio do acionamento de seu método construtor, por exemplo:

```
String nome = new String("Maria"); // construtor
```

# String

Vários são os métodos disponíveis nesta classe (String), sendo alguns deles abordados a seguir:

## **MÉTODO**

## **FUNCIONALIDADE**

charAt(int)	retorna o caracter da posição
indexOf(char)	retorna a posição do caracter
concat(String)	concatena String (como operador + )
equals(String)	compara se 2 Strings são idênticas
equalsIgnoreCase(String)	compara 2 Strings sem considerar Mai./min.
length()	tamanho da String instanciada
lastIndexOf(char)	retorna a última posição do caracter
replace(char, char)	troca os caracteres na String toda
substring(int inicial, int fim)	retorna um pedaço (subcadeia) da String
toLowerCase()	transforma toda String em minúsculo
toUpperCase()	transforma toda String em maiúsculo
startsWith(String)	retorna true quando String estiver no início
endsWith(String)	retorna true quando String estiver no final
trim()	retira espaços do início e final da String
valueOf(x)	converte vários tipos de dados (x) em String

:

:

# String

## Processo de Concatenação

- O sinal **+** é usado para concatenar valores em uma String, podendo esta concatenação envolver valores de diferentes tipos, como é realizado no uso da classe System, por exemplo:

```
System.out.print("Salário = " + sal + "Ano = " + ano);
```

sendo a expressão Salário concatenada com a variável sal (salário) do tipo double e a expressão Ano com a variável inteira ano

- Este tipo de concatenação com o sinal **+** converte os outros valores em String para serem mostrados pelo computador.



# String

## Comparação entre Strings

Dois métodos efetuam este tipo de comparação, sendo o `compareTo` similar a função `strcmp()` em C.

```
String nome1 = new String("Paulo");  
String nome2 = new String("Paula");
```

```
if(nome1.equals(nome2))
```

└→ Retorna true para Strings idênticas  
ou false para Strings diferentes

```
if(nome1.compareTo(nome2))
```

└→ Retorna zero (0) para Strings idênticas ou  
qualquer outro valor inteiro para diferentes

```
if(nome1 == nome2)
```

↓  
Compara a referência ou endereço de memória dos objetos



# Exercícios de Fixação

- 7) Elabore um programa que cadastre uma senha de até 5 caracteres, não podendo ser menor que 3 caracteres e nem conter espaço em branco. Seu programa deverá permitir até 9 tentativas de acerto, definidas em uma constante denominada MAXIMO, por outros usuários diferentes daquele que cadastrou a senha. Caso ninguém consiga acertar até este limite, seu programa deve informar que o computador irá se auto-destruir em 10 segundos e encerrar o programa.

Use os recursos que achar adequados para solução deste problema usando String, onde os caracteres maiúsculos são diferentes dos minúsculos na averiguação da senha. Caso o usuário acerte a senha o programa deverá parabenizá-lo com uma mensagem que será apresentada depois de 22 linhas abaixo da senha informada e no centro da linha será escrita esta mensagem.

## Exercícios de Fixação

- 8) Faça um programa que armazene uma frase e mostre a quantidade de cada vogal contida na mesma, além do total geral de vogais. Sua solução deverá considerar as vogais independente das mesmas estarem em maiúsculo ou minúsculo. Encerre sua solução apresentando a quantidade de cada vogal, o total das vogais e o tamanho da frase digitada pelo usuário. Repita este processo de análise de uma frase enquanto o usuário desejar. Frase sem dados é inválida e deve ser solicitada novamente.
- 9) No campeonato de basquete regional existem 5 equipes e cada uma possui 10 jogadores. Faça um programa que mostre a quantidade de jogadores com idade maior que 15 anos, a média destas idades por equipe e a porcentagem de jogadores com altura menor que 1,50 metros entre todas as equipes.

# Referência de Criação e Apoio ao Estudo

## Material para Consulta e Apoio ao Conteúdo

- HORSTMANN, C. S., CORNELL, G., Core Java2 , volume 1, Makron Books, 2001.
  - Capítulo 3
- FURGERI, S., Java 2: Ensino Didático: Desenvolvendo e Implementando Aplicações, São Paulo: Érica, 2002.
  - Capítulo 1, 3 e 4
- ASCENCIO, A. F. G.; CAMPOS, E. A. V., Fundamentos da programação de computadores, 2 ed., São Paulo: Pearson Prentice Hall, 2007.
  - Capítulo 2, 3, 4 e 5
- Universidade de Brasília (UnB Gama)
  - <https://cae.ucb.br/conteudo/unbfga>  
(escolha a disciplina **Orientação a Objetos** no menu superior)