



DIGITAL  
INNOVATION  
ONE

# Métodos

---

Thiago Leite e Carvalho

Engenheiro de Software, Professor, Escritor

# Objetivo do curso

Possibilitar que o aluno compreenda o que é um método, como criá-lo e utilizá-lo.

# Percurso

## Aula 1

Criação

## Aula 2

Sobrecarga

## Aula 3

Retornos

# Requisitos

- ✓ Lógica de Programação
- ✓ Java
- ✓ IntelliJ

# Aula 1: Criação

Métodos

# Objetivos

1. Entender o que é um método
2. Saber como definir e utilizar métodos
3. Aplicar boas práticas em sua criação e uso



# Conceituação

"É uma porção de código (sub-rotina) que é disponibilizada por uma classe. Este é executado quando é feita uma requisição a ele. São responsáveis por definir e realizar um determinado comportamento."

Ou seja, é método que é responsável por realmente fazer a aplicação funcionar. É nele que iremos definir os códigos que irão manipular os dados. Como dito, um método deve ser chamado para executar, pois não funciona sozinho. Esta chamada é através de uma classe ou objetos.

# Criação

Padrão de definição:

**<?visibilidade?> <?tipo?> <?modificador?> retorno  
nome (<?parâmetros?>) <?exceções?> corpo**



# Criação

onde:

V: "public", "protected" ou "private"

T: concreto ou abstrato

M: "static" ou "final"

R: tipo de dado ou "void" // nao retorna nada só

N: nome que é fornecido ao método// padrao

P: parâmetros que pode receber // se o metodo for usado deficar  
dendro do parametro ou paramenrtro vasio

E: exceções que pode lançar

C: código que possui ou vazio

# Criação

**V**: são as visibilidades. Assim como as variáveis, os métodos tb podem definir as visibilidades.

**T**: se é concreto ou abstrato. Este conceito é mais fácil de explorar em um curso de OO. Aqui vamos sempre utilizar métodos concretos.

**M**: se é estatico, não estático ou final. Este conceito é mais fácil de explorar em um curso de OO. Aqui vamos sempre utilizar métodos estaticos.

**R**: é o tipo da informação que o método pode ou não retornar. Se retornar, pode ser qualquer um dos tipos de dados já apresentados, além de objetos(que são explorados no devido curso). Se não retornar nada, usa-se a palavra reservada "void". Vazio em inglês.

**N**: nome que é fornecido ao método// padrao

**P**: são os parâmetros que o método pode receber pra manipular e gerar novos valores.

**E**: Lista de exceções que pode lançar.

**C**: códigos que pode possuir. Se não tiver código, termina com ";".

É válido ressaltar que os itens sublinhados são digamos os mais "comuns de usar" e os que exploraremos neste curso.

Existe tb considerações sobre T e M. Existem alguma combinações entre estes que não são validas. Mais uma vez, em OO conseguimos explorar isso.

# Exemplos

Abaixo temos alguns exemplos de métodos válidos e mais comuns, no que diz respeito à utilização das possibilidades apresentadas. Cada método terá sua necessidade e usará os itens de seu padrão de definição.

```
public String getNome() { ... } // retorna um Nome
public double calcularTotalNota() {...}
public int verificarDistancia(int cordenada1, int cordenada2) {...}
public abstract void executar() ; // corpo vazio do metodo
public void alterarFabricante(Fabricante fabricante) { ... }
public Relatorio gerarDadosAnaliticos(Cliente cliente, List<Compra>
compras) {...} // como passar mais de um paramentos
```

**public static R N(P) {...}** forma que vamos utilizar neste curso

R = RETORNO, N = NOME, P = PARÂMETROS

# Utilização

Passa-se uma mensagem através de uma classe ou objeto.

```
nome_da_classe.nome_do_metodo(); ou nome_da_classe.nome_do_metodo(...);  
nome_do_objeto.nome_do_metodo(); ou nome_do_objeto.nome_do_metodo(...);
```

Para chamar uma classe

```
Math.random(); ou Math.sqrt(4);
```

Para chamar uma objeto

```
usuario.getEmail(); ou usuario.alterarEndereco(  
    endereco);
```

# Particularidades

- Assinatura: é a forma de identificar unicamente o método

Ass = nome + parâmetros

**Nome()**      Obs: tb é uma assinatura, é uma lista vazia

**Método:**

```
public double calcularTotalVenda(double  
precoItem1, double precoItem2, double precoItem3)  
{...}
```

**Assinatura:**

```
calcularTotalVenda(double precoItem1,  
double precoItem2, double precoItem3)
```

# Particularidades

- **Construtor e Destrutor:** são métodos especiais usados na Orientação a Objetos.  
Os construtores criam objetos a partir de classes. Os destrutores auxiliam na destruição de objetos.
- **Mensagem:** é o ato de solicitar ao método que o mesmo execute. Esta pode ser direcionada a um objeto ou a uma classe.

É o que faz o software de fato funcionar. São as execuções dos métodos, as mensagens que são passadas para eles para que eles executem seus processamentos (códigos) internos. Nesse momento apenas passaremos mensagens a métodos através de uma classe.



# Particularidades

- Passagem de Parâmetros
  - Por valor (cópia)
  - Por referência (endereço)
    - Por valor (cópia)

```
int i = 10;  
public void fazerAlgo(int i) {
```

```
    i = i + 10;  
    System.out.println("Valor de i dentro: " + i);  
}  
System.out.println("Valor de i fora: " + i);
```

O resultado vai ser = 20

O resultado vai ser = 10

# Boas práticas

- Nomes devem ser descritivos, mas curtos
- Notação camelo

```
verificarSaldo(); executarTransferencia(...); existeDebito();
```

- Deve possuir entre 80 e 120 linhas
- Evite lista de parâmetros longas
- Visibilidades adequadas



# Boas práticas

Esse é o grande desafio! Criar nomes que transmitam a ideia do comportamento que o método define, mas sem ficar grande demais. Via de regras preposições como "de", "do", "da" são evitadas, assim como artigos. Na maioria das vezes verbos e substantivos conseguem suprir tal necessidade.

Métodos muito grandes são difíceis de entender e manter. Então evitar isto ajuda na manutenção do mesmo. Essas valores não são uma regra, mas existem estudos que aconselhem a este valor [entre 80 e 120 linhas](#). Sendo 150 a exceção o máxima. Sempre que possível a criação e reuso de métodos deve ser feita, assim evita-se também a repetição de códigos.

Lista de parâmetros muito longas geram um forte acoplamento. Listas curtas são mais fáceis de manter. Acoplamento é um conceito um pouco mais avançado, mas tenha em mente que listas longas geram forte acoplamento.

Definir a visibilidade adequada de um método é importantíssimo. Agora tudo será público(public) por facilidade de explicação. Mas na verdade a visibilidade deve ser bem pensada.

# Exercitando

Cria uma aplicação que resolva as seguintes situações:

- Calcule as 4 operações básicas: soma, subtração, multiplicação e divisão. Sempre 2 valores devem ser passados.
- A partir da hora do dia, informe a mensagem adequada: Bom dia, Boa tarde e Boa noite.
- Calcule o valor final de um empréstimo, a partir do valor solicitado. Taxas e parcelas influenciam. Defina arbitrariamente as faixas que influenciam nos valores.

# Exercitando

## Observações:

- Tente ao máximo criar métodos que trabalhem sozinhos ou em conjunto
- Pode chamar um método dentro de outro
- Pode passar como parâmetro, a chamada de um outro método

```
public class Calculadora {  
    //Método soma  
    public static void soma(double numero1, double numero2) {  
        //Visib/Modif/Retorno/Nome(Parametro1, Parametro2) Obs.: O retorno Void não tem retorno é um retorno vazio  
        // O Modificador static que possibilita chamar um método a partir de uma classe  
        double resultado = numero1 + numero2;  
  
        System.out.println("A soma de " + numero1 + " mais " + numero2 + " é " + resultado);  
    }  
  
    public static void subtracao(double numero1, double numero2) {  
  
        double resultado = numero1 - numero2;  
  
        System.out.println("A subtracao de " + numero1 + " menos " + numero2 + " é " + resultado);  
    }  
  
    public static void multiplicacao(double numero1, double numero2) {  
  
        double resultado = numero1 * numero2;  
  
        System.out.println("A multiplicação de " + numero1 + " vezes " + numero2 + " é " + resultado);  
    }  
  
    public static void divisao(double numero1, double numero2) {  
  
        double resultado = numero1 / numero2;  
  
        System.out.println("A divisão de " + numero1 + " por " + numero2 + " é " + resultado);  
    }  
}
```

```
public class Mensagem {
    //Método obterMensagem
    public static void obterMensagem(int hora) {
        //Visib/Modif/Retorno/Nome(Parametro1) Obs.: O retorno Void não tem retorno é um retorno vazio
        // O Modificador static que possibilita chamar um método a partir de uma classe.
        switch (hora) {
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
            case 10:
            case 11:
            case 12:
                mensagemBomDia();
                break;
            case 13:
            case 14:
            case 15:
            case 16:
            case 17:
                mensagemBoaTarde();
                break;
            case 18:
            case 19:
            case 20:
            case 21:
            case 22:
            case 23:
```

```
        case 23:
        case 0:
        case 1:
        case 2:
        case 3:
        case 4:
            mensagemBoaNoite();
            break;
        default:
            System.out.println("Hora inválida.");
            break;
    }
}

//Método mensagemBomDia
public static void mensagemBomDia () { //Esse método foi criado para mostrar que é possível criar um
método dentro de outro método e que isso é comum
//Visib/Modif/Retorno/Nome(Parametro1) Obs.: O retorno Void não tem retorno é um retorno vazio
    System.out.println("Bom dia!");
}

public static void mensagemBoaTarde () {
    System.out.println("Bom tarde!");
}

public static void mensagemBoaNoite () {
    System.out.println("Bom noite!");
}
```

```
public class Emprestimo {
    public static int getDuasParcelas () {
        //Visib/Modif/Retorno/Nome(parâmetro "no caso está sem parâmetro")
        return 2;
    }

    public static int getTresParcelas () {
        return 3;
    }

    public static double getTaxaDuasParcelas () {
        return 0.3;
    }

    public static double getTaxaTresParcelas () {
        return 0.45;
    }

    public static void calcular(double valor, int parcelas) { //Método principal
        if (parcelas == 2) {
            double valorFinal = valor + (valor * getTaxaDuasParcelas()); //Para mostrar que é possível
            //criar um método dentro de outro

            System.out.println("Valor final do empréstimo para 2 parcelas: R$ " + valorFinal);
        } else if (parcelas == 3) {
            double valorFinal = valor + (valor * getTaxaTresParcelas());
            System.out.println("Valor final do empréstimo para 3 parcelas: R$ " + valorFinal);
        } else {
            System.out.println("Quantidade de parcelas não aceita." );
        }
    }
}
```

```
public class Main {  
    // A classe Main é onde o programa vai ser executado  
    public static void main(String[] args) {  
        // Calculadora  
        System.out.println("Exercício calculadora");  
        Calculadora.soma(3, 6); //Para chamar o método (passar uma mensagem) a partir da classe nesse  
caso calculadora  
        //Classe.nome(parâmetro1, parâmetro2) - Precisamos passar esses parâmetros  
        Calculadora.subtracao(9, 1.8);  
        Calculadora.multiplicacao(7, 8);  
        Calculadora.divisao(5, 2.5);  
  
        // Mensagem  
        System.out.println("Exercício mensagem");  
        Mensagem.obterMensagem(9);  
        Mensagem.obterMensagem(14);  
        Mensagem.obterMensagem(1);  
  
        // Empréstimo  
        System.out.println("Exercício empréstimo");  
        Empréstimo.calcular(1000, Empréstimo.getDuasParcelas()); // Foi criado para mostrar que é  
possível passar um parâmetros para outro método  
        Empréstimo.calcular(1000, Empréstimo.getTresParcelas());  
        Empréstimo.calcular(1000, 5);  
    }  
}
```



# Aula 2: Sobrecarga

Métodos

# Objetivos

1. Entender o que é sobrecarregar um método
2. Saber como criar sobrecargas

# Conceituação

"É a capacidade de definir métodos para diferentes contextos, mas preservando seu nome. "

Definição um pouco abstrata. Mas isso quer dizer que na sobrecarga, conseguimos criar vários métodos com o mesmo nome, mas que poderão se comportar diferente(contexto) de acordo com sua lista de parâmetros. Ou seja, esse lista pode mudar.

Obs: quando os parâmetros são completamente iguais devemos alterar o tipo de dado, por exemplo tipo de dado double para um parâmetro que estão iguais deverá ser alterado para float.

# Criação

Alterar a assinatura do método:

**Ass = nome + parâmetros**

**Sobrecarga é Mudar a lista de parâmetros e manter o nome do método.**

```
converterParaInteiro(float f);  
converterParaInteiro(double d);  
converterParaInteiro(String s);  
converterParaInteiro(float f, RoundType rd);  
converterParaInteiro(double d, RoundType rd);  
converterParaInteiro(String s, RoundType rd);  
  
converterParaInteiro(RoundType rd, String s);  
converterParaInteiro();
```



# Exemplos

<https://d>

tml

void	<code>println()</code> Terminates the current line by writing the line separator string.
void	<code>println(boolean x)</code> Prints a boolean and then terminate the line.
void	<code>println(char x)</code> Prints a character and then terminate the line.
void	<code>println(char[] x)</code> Prints an array of characters and then terminate the line.
void	<code>println(double x)</code> Prints a double and then terminate the line.
void	<code>println(float x)</code> Prints a float and then terminate the line.
void	<code>println(int x)</code> Prints an integer and then terminate the line.
void	<code>println(long x)</code> Prints a long and then terminate the line.
void	<code>println(Object x)</code> Prints an Object and then terminate the line.
void	<code>println(String x)</code> Prints a String and then terminate the line.



# Exemplos

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

<code>static String</code>	<code>valueOf(boolean b)</code> Returns the string representation of the boolean argument.
<code>static String</code>	<code>valueOf(char c)</code> Returns the string representation of the char argument.
<code>static String</code>	<code>valueOf(char[] data)</code> Returns the string representation of the char array argument.
<code>static String</code>	<code>valueOf(char[] data, int offset, int count)</code> Returns the string representation of a specific subarray of the char array argument.
<code>static String</code>	<code>valueOf(double d)</code> Returns the string representation of the double argument.
<code>static String</code>	<code>valueOf(float f)</code> Returns the string representation of the float argument.
<code>static String</code>	<code>valueOf(int i)</code> Returns the string representation of the int argument.
<code>static String</code>	<code>valueOf(long l)</code> Returns the string representation of the long argument.
<code>static String</code>	<code>valueOf(Object obj)</code> Returns the string representation of the Object argument.

# Curiosidade

## Sobrecarga x Sobrescrita

Embora sejam dois conceitos relacionados á metodos, estas são completamente diferentes. O sobrecarga, como disse tem relação ao mesmo método com parametros diferentes. Já a sobrescrita, tem relação com herança, qual é um assunto relacionado a orientação a objeto. Então não confunda. Ambos tem relação com método mas tem formas diferentes de definição, uso, além de comportamentos diferentes.

# Exercitando

Cria uma aplicação que calcula a área dos 3 quadriláteros notáveis: quadrado, retângulo e trapézio.

Obs: Use sobrecarga.



```
public class Quadrilatero {
    //Assinatura = Nome + Parâmetros
    public static void area(double lado) { // Sobrecarga: Pq se mantem o nome do método, mas muda a
        lista de parâmetros.
        //O nome d método é sempre o mesmo, porém em todos os casos os parâmetros são diferentes.
        System.out.println("Área do quadrado:" + lado * lado);
    }

    public static void area(double lado1, double lado2) { //Sobrecarga: Pq se mantem o nome do método,
        mas muda a lista de parâmetros.
        // O nome do método é sempre o mesmo, porém em todos os casos os parâmetros são diferentes.
        System.out.println("Área do retângulo:" + lado1 * lado2);
    }

    public static void area(double baseMaior, double baseMenor, double altura) { //Sobrecarga: Pq se
        mantem o nome do método, mas muda a lista de parâmetros.
        // O nome do método é sempre o mesmo, porém em todos os casos os parâmetros são diferentes.
        System.out.println("Área do trapézio:" + ((baseMaior+baseMenor)*altura) / 2);
    }

    public static void area(float diagonal1, float diagonal2) { //Sobrecarga: Pq se mantem o nome do
        método, mas muda a lista de parâmetros.
        //O nome do método é sempre o mesmo, porém em todos os casos os parâmetros são diferentes.
        System.out.println("Área do losango:" + (diagonal1 * diagonal2)/ 2);
    }
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
  
        // Quadrilátero  
        System.out.println("Exercício quadrilátero");  
        Quadrilatero.area(3);  
        Quadrilatero.area(5d, 5d);  
        Quadrilatero.area(7, 8, 9);  
        Quadrilatero.area(5f, 5f);  
        //Classe.nome(parâmetro) - Isso para chamar a classe ou melhor passar  
uma mensagem.  
  
    }  
}
```

# Aula 3: Retornos

Métodos

# Objetivos

## 1. Entender como funcionam

# Relembrando

- Retorno - É uma instrução de interrupção
- Simbologia: return

O continue e o break também são instruções de interrupção, mas estão mais atrelados a laços de repetição e o retorno está atrelado a métodos.

# Funcionamento

O método executa seu retorno quando:

- Completa todas suas instruções internas
- Chega a uma declaração explícita de retorno
- Lança uma exceção

O que ocorrer primeiro deste tres casos, faz o método finalizar. Assim, a execução do programa volta para o ponto onde o método foi chamado, ou seja, foi passada uma mensagem para ele.

# Considerações

- O tipo de retorno do método é definido na sua criação e pode ser um tipo primitivo ou objeto;
- O tipo de dado do return deve ser compatível com o do método;
- Se o método for sem retorno(void), pode ou não ter um "return" para encerrar sua execução.



Naquele padrão é o R e fica logo antes do nome do método. Já foi dito que pode ser um TP ou um O. Neste caso, o retorno deve ser compatível com o definido no método. Se não for, gera um erro de compilação.

Se precisar, o método pode não retornar nada. Usa-se o void. Mas se ainda precisar, pode usar o "return puro e sem valor" para abortar no momento desejado a execução do método.

# Exemplos

```
public String getMensagem() {  
    return "Ola!";  
}
```

```
public double getJuros() {  
    return 2.36;  
}
```

```
public int getParcelas() {  
    return 1.36f;  
}
```

Vai dar erro de compilação pq um float não é compátivel com um int

```
public void setIdade() {  
    return 10;  
}
```

Vai dar erro de compilação pq void não retorna nada. Deveria ser return;

```
public void executar() {  
    ...  
    return;  
    ....  
}
```



# Exercitando

Recrie a aplicação que calcula a área dos 3 quadriláteros notáveis. Agora faça os métodos retornarem valores.

```
public class Quadrilatero {

    public static double area(double lado) {

        return lado * lado;
    }

    public static double area(double lado1, double lado2) {

        return lado1 * lado2;
    }

    public static double area(double baseMaior, double baseMenor, double altura) {

        return ((baseMaior+baseMenor)*altura) / 2;
    }

    public static void xpto() {

        System.out.println("Antes");
        return;
    }

    public static long abc() {
        return 1.6; // Esse método está retornando um double e não um log. isso dá um erro.
    }

}
```

```
public class Main {  
  
    public static void main(String[] args) {  
  
        // Retornos  
        System.out.println("Exercício retornos");  
  
        double areaQuadrado = Quadrilatero.area(3);  
        System.out.println("Área do quadrado:" + areaQuadrado);  
  
        double areaRetangulo = Quadrilatero.area(5,5);  
        System.out.println("Área do retângulo:" + areaRetangulo);  
  
        double areaTrapezio = Quadrilatero.area(7,8,9);  
        System.out.println("Área do trapézio:" + areaTrapezio);  
    }  
}
```

# Dúvidas?

- > Fórum do curso
- > Comunidade online (discord)

# Para saber mais

- <https://www.casadocodigo.com.br/products/livro-oo-conceitos>
- <https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>

# Para saber mais

- <https://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html>
- <https://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html>

# Objetivo do curso

Possibilitar que o aluno compreenda o que é um método, como criá-lo e utilizá-lo.

# Percurso

## Aula 1

Criação

## Aula 2

Sobrecarga

## Aula 3

Retornos



# Requisitos

- ✓ Lógica de Programação
- ✓ Java
- ✓ IntelliJ

# Dúvidas durante o curso?

- > Fórum do curso
- > Comunidade online (discord)

# Aula 1: Criação

Métodos

# Objetivos

1. Entender o que é um método
2. Saber como definir e utilizar métodos
3. Aplicar boas práticas em sua criação e uso



# Conceituação

"É uma porção de código (sub-rotina) que é disponibilizada por uma classe. Este é executado quando é feita uma requisição a ele. São responsáveis por definir e realizar um determinado comportamento."

# Criação

Padrão de definição:

**<?visibilidade?> <?tipo?> <?modificador?> retorno  
nome (<?parâmetros?>) <?exceções?> corpo**

# Criação

onde:

V: "public", "protected" ou "private"

T: concreto ou abstrato

M: "static" ou "final"

R: tipo de dado ou "void" // não retorna nada só

N: nome que é fornecido ao método// padrão

P: parâmetros que pode receber // se o método for usado definir dentro do parâmetro ou parâmetro vazio

E: exceções que pode lançar

C: código que possui ou vazio

# Exemplos

```
public String getNome() { ... } // retorna um Nome  
public double calcularTotalNota() {...}  
public int verificarDistancia(int cordenada1, int cordenada2) {...}  
public abstract void executar() ; // corpo vazio do metodo  
public void alterarFabricante(Fabricante fabricante) { ... }  
public Relatorio gerarDadosAnaliticos(Cliente cliente,  
List<Compra> compras) {...} // como passar mais de um  
paramentro
```

```
public static R N(P) {...}
```



# Utilização

Passa-se uma mensagem através de uma classe ou objeto.

`nome_da_classe.nome_do_metodo();` ou `nome_da_classe.nome_do_metodo(...);`  
`nome_do_objeto.nome_do_metodo();` ou `nome_do_objeto.nome_do_metodo(...);`

`Math.random();` ou `Math.sqrt(4);`

`usuario.getEmail();` ou `usuario.alterarEndereco(  
    endereco);`

# Particularidades

- Assinatura: é a forma de identificar unicamente o método  
Ass = nome + parâmetros

## Método:

```
public double calcularTotalVenda(double  
precoItem1, double precoItem2, double precoItem3)  
{...}
```

## Assinatura:

```
calcularTotalVenda(double precoItem1,  
double precoItem2, double precoItem3)
```

# Particularidades

- Construtor e Destrutor: são métodos especiais usados na Orientação a Objetos.
- Mensagem: é o ato de solicitar ao método que o mesmo execute. Esta pode ser direcionada a um objeto ou a uma classe.

# Boas práticas

- Nomes devem ser descritivos, mas curtos
- Notação camelo

```
verificarSaldo(); executarTransferencia(...); existeDebito();
```

- Deve possuir entre 80 e 120 linhas
- Evite lista de parâmetros longas
- Visibilidades adequadas

# Exercitando

Cria uma aplicação que resolva as seguintes situações:

- Calcule as 4 operações básicas: soma, subtração, multiplicação e divisão. Sempre 2 valores devem ser passados.
- A partir da hora do dia, informe a mensagem adequada: Bom dia, Boa tarde e Boa noite.
- Calcule o valor final de um empréstimo, a partir do valor solicitado. Taxas e parcelas influenciam. Defina arbitrariamente as faixas que influenciam nos valores.

# Exercitando

## Observações:

- Tente ao máximo criar métodos que trabalhem sozinhos ou em conjunto
- Pode chamar um método dentro de outro
- Pode passar como parâmetro, a chamada de um outro método

# Aula 2: Sobrecarga

Métodos

# Objetivos

1. Entender o que é sobrecarregar um método
2. Saber como criar sobrecargas



# Conceituação

"É a capacidade de definir métodos para diferentes contextos, mas preservando seu nome. "

# Criação

Alterar a assinatura do método:

Ass = nome + parâmetros

```
converterParaInteiro(float f);  
converterParaInteiro(double d);  
converterParaInteiro(String s);  
converterParaInteiro(float f, RoundType rd);  
converterParaInteiro(double d, RoundType rd);  
converterParaInteiro(String s, RoundType rd);  
  
converterParaInteiro(RoundType rd, String s);  
converterParaInteiro();
```



# Exemplos

<https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html>

void	<code>println()</code> Terminates the current line by writing the line separator string.
void	<code>println(boolean x)</code> Prints a boolean and then terminate the line.
void	<code>println(char x)</code> Prints a character and then terminate the line.
void	<code>println(char[] x)</code> Prints an array of characters and then terminate the line.
void	<code>println(double x)</code> Prints a double and then terminate the line.
void	<code>println(float x)</code> Prints a float and then terminate the line.
void	<code>println(int x)</code> Prints an integer and then terminate the line.
void	<code>println(long x)</code> Prints a long and then terminate the line.
void	<code>println(Object x)</code> Prints an Object and then terminate the line.
void	<code>println(String x)</code> Prints a String and then terminate the line.



# Exemplos

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

<code>static String</code>	<code>valueOf(boolean b)</code> Returns the string representation of the boolean argument.
<code>static String</code>	<code>valueOf(char c)</code> Returns the string representation of the char argument.
<code>static String</code>	<code>valueOf(char[] data)</code> Returns the string representation of the char array argument.
<code>static String</code>	<code>valueOf(char[] data, int offset, int count)</code> Returns the string representation of a specific subarray of the char array argument.
<code>static String</code>	<code>valueOf(double d)</code> Returns the string representation of the double argument.
<code>static String</code>	<code>valueOf(float f)</code> Returns the string representation of the float argument.
<code>static String</code>	<code>valueOf(int i)</code> Returns the string representation of the int argument.
<code>static String</code>	<code>valueOf(long l)</code> Returns the string representation of the long argument.
<code>static String</code>	<code>valueOf(Object obj)</code> Returns the string representation of the Object argument.

# Curiosidade

Sobrecarga x Sobrescrita

# Exercitando

Cria uma aplicação que calcula a área dos 3 quadriláteros notáveis: quadrado, retângulo e trapézio.

Obs: Use sobrecarga.

# Aula 3: Retornos

Métodos

# Objetivos

## 1. Entender como funcionam



# Relembrando

- É uma instrução de interrupção
- Simbologia: return

# Funcionamento

O método executa seu retorno quando:

- Completa todas suas instruções internas
- Chega a uma declaração explícita de retorno
- Lança uma exceção

# Considerações

- O tipo de retorno do método é definido na sua criação e pode ser um tipo primitivo ou objeto;
- O tipo de dado do return deve ser compatível com o do método;
- Se o método for sem retorno(void), pode ou não ter um "return" para encerrar sua execução.

# Exemplos

```
public String getMensagem() {  
    return "Ola!";  
}  
  
public double getJuros() {  
    return 2.36;  
}  
  
public int getParcelas() {  
    return 1.36f;  
}
```

```
public void setIdade(...) {  
    return 10;  
}  
  
public void executar() {  
    ...  
    return;  
    ....  
}
```

# Exercitando

Recrie a aplicação que calcula a área dos 3 quadriláteros notáveis. Agora faça os métodos retornarem valores.

# Para saber mais

- <https://www.casadocodigo.com.br/products/livro-oo-conceitos>
- <https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html>

# Para saber mais

- <https://docs.oracle.com/javase/tutorial/java/javaOO/returnvalue.html>
- <https://docs.oracle.com/javase/tutorial/java/javaOO/arguments.html>