



# Programação Orientada a Objetos

*Diego Marques de Carvalho*



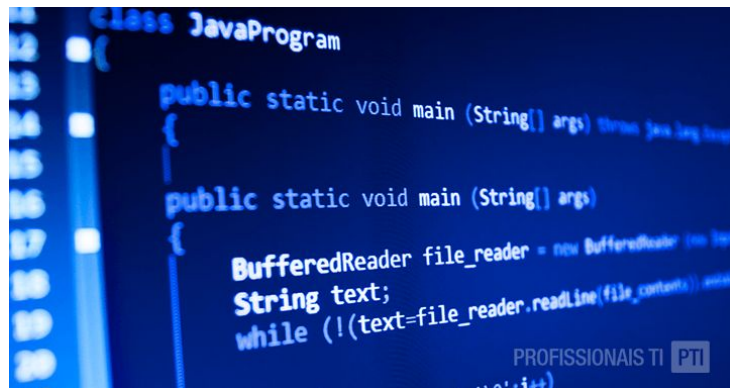
# FUNDAMENTOS

# ORIENTAÇÃO A OBJETOS

- Não se limita a uma forma de programação é também um paradigma de análise e projeto (modelagem) dos processos/tarefas que devem ser realizados.
- Orientação a Objetos é uma forma de pensar e representar mais realista as necessidades dos softwares.
- OO facilitou o processo de programação a partir do que já existia.
- Programação -> Processo de Alto Nível

PROGRAMAÇÃO  
ORIENTADA A  
OBJETOS

POO

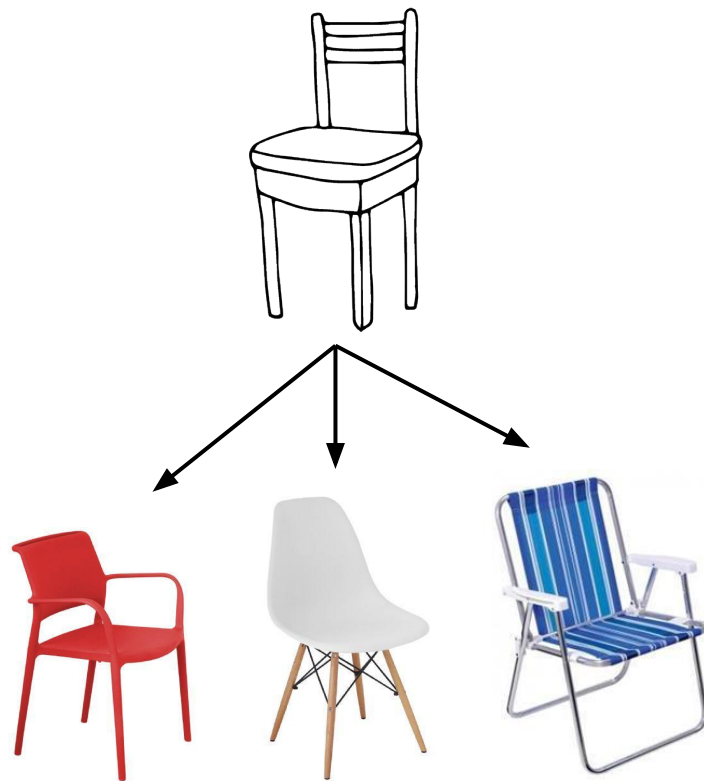


```
Class JavaProgram
{
    public static void main (String[] args) throws java.lang.Exception
    {
        public static void main (String[] args)
        {
            BufferedReader file_reader = new BufferedReader (new InputStreamReader(System.in));
            String text;
            while (!(text=file_reader.readLine(file_contents)).equals(""))
            {
                //...
            }
        }
    }
}
```

PROFISSIONAIS TI PTI

# FUNDAMENTOS

- **Abstração**
- “Processo pelo qual se isolam características de um objeto, considerando os que tenham em comum certos grupos de objetos”
  - Resumindo: Não devemos nos preocupar com características menos importantes, ou seja, acidentais.
  - Concentração nos aspectos essenciais



# FUNDAMENTOS

## Encapsulamento



Mesilato de di-hidroergotamina .....1 mg  
Dipirona .....350 mg  
Cafeína.....100 mg

Misturar bem e ingerir com água. Repetir toda vez que estiver com dor

# FUNDAMENTOS

## Reúso

- Não existe nada pior em programação do que repetição de código.
- Quanto maior o número de repetições mais difícil fica a manutenção.
  - Herança -> Criar Classes a partir de outras Classes  
(reaproveitamento de código – dados – da classe mãe)
  - Associação
    - O reaproveitamento é diferente -> Uma Classe pede ajuda para outra a fim de fazer o que ela não é capaz de fazer sozinha

# CONCEITOS ESTRUTURAIS



# A CLASSE

- Uma estrutura que abstrai um conjunto de objetos que possui características semelhantes.
- Definição do comportamento dos objetos -> Métodos
- Estados possíveis -> Atributos
- A Classe é a forma mais básica para definir de uma única vez como devem ser todos os objetos criados
  - Evita repetição
  - É fundamental para aplicação do conceito de abstração
  - Abstração de uma entidade
    - Física (Carro, Pessoa, Casa)
    - Conceitual (Venda, estoque, viagem)

# O ATRIBUTO

- Atributo é um elemento de uma classe, responsável por definir estrutura de dados.
- O conjunto de dados será responsável por representar suas características e farão parte dos objetos criados a partir da classe.
- Os atributos são definidos dentro da classe
- Detalhamento de uma entidade -> Atributo
- Atributo -> Caracterização
- Características e Informações -> Atributos
  - Armazenar e Manipular

Personagem
+nome +cor +quantidadeDeEstrelas +arquetipo

# O MÉTODO

- O método serve para manipular os atributos
- É uma porção de código (sub-rotina) que é disponibilizada pela classe.
- Este é executado quando feita uma requisição.
- Os métodos são responsáveis por realizar um determinado comportamento.
- Para definir métodos podemos pensar em verbos, isso otimiza sua identificação.

Personagem
+nome +cor +quantidadeDeEstrelas +arquetipo
+pular() +pegarEstrelas(Estrelas estrelas) +criarPersonagem() +getNome() +setNome(String nome)

# O MÉTODO

- Embora não faça parte de sua assinatura, os métodos devem possuir um retorno.
- O retorno pode ser: Tipos primitivos ou qualquer um dos conceitos (classes)
- Void
  - Quando necessitamos que um método não tenha um retorno devemos omitir o tipo.
  - Utilizar a palavra reservada Void
    - Void: Significa que o método não retorna nada

Personagem
+nome +cor +quantidadeDeEstrelas +arquetipo
+pular() +pegarEstrelas(Estrelas estrelas) +criarPersonagem() +getNome() +setNome(String nome)

# MÉTODOS ESPECIAIS

- Toda classe deve possuir dois métodos especiais.
  - **Construtor** e o **Destrutor**
  - O construtor é responsável por criar objetos. Sempre que for necessário criar objetos de uma determinada classe, seu construtor deverá ser utilizado.
  - Responsável por prover alguns valores iniciais que o objeto precisa ter no começo.
  - No Java para criar o construtor devemos fazer um método com o mesmo nome da classe e sem retorno.
  - Não precisamos definir nenhum retorno para o método construtor

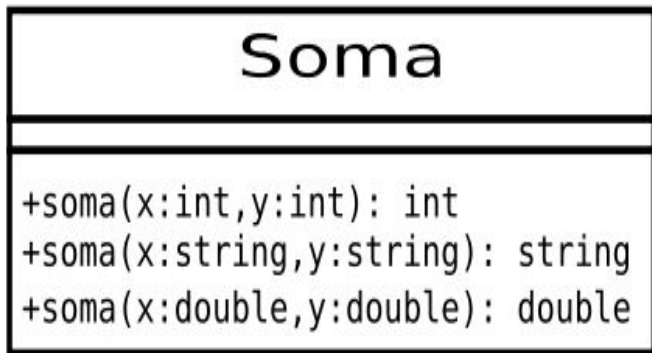
# MÉTODOS ESPECIAIS

- **Destrutor**

- Tem a função inversa do construtor
- Destruir o objeto criado a partir da classe.
- Sempre que não precisamos mais de objetos que foram criados a partir de uma determinada classe, devemos usar o seu destrutor.

# SOBRECARGA DE MÉTODO

- As vezes precisamos que um método possua entradas (parâmetros) diferentes, isso ocorre porque ele pode precisar realizar operações diferentes em determinados contextos.
- Manter o nome do método -> Alterar a lista de parâmetros
  - Adicionar ou remover parâmetros para prover um novo comportamento



```
class Quadrilatero{
    //Quadrado
    double calcularArea(double lado){
        return lado * lado;
    }
    //Retângulo
    double calcularArea(double baseMaior, double baseMenor){
        return baseMaior * baseMenor;
    }
    //Losango
    double calcularArea(double diagonalMaior, double diagonalMenor){
        return diagonalMaior * diagonalMenor;
    }
}
```



# SOBRECARGA DE MÉTODO

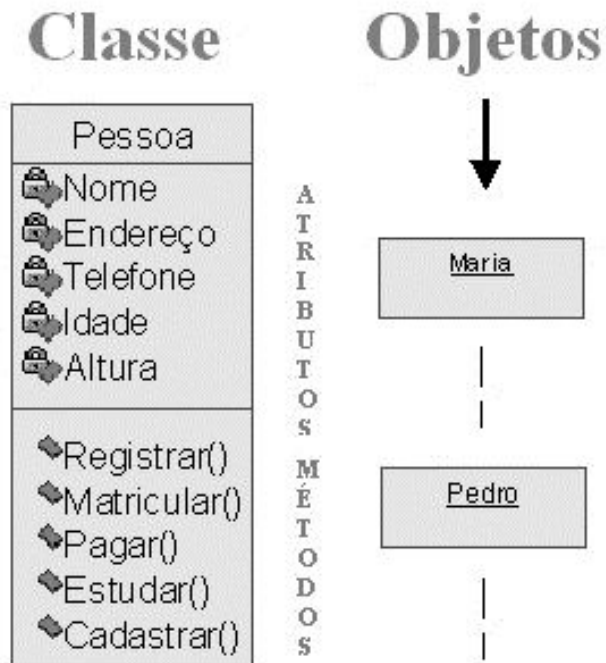
- No exemplo anterior apenas a quantidade de parâmetros foi alterada. Porém seus tipos foram os mesmos.
  - Caso exista a necessidade podemos mudar o tipo de dados

<b>Categoria</b>	<b>Tipo</b>	<b>Tamanho</b>
Inteiro	byte	8 bits
Inteiro	short	16 bits
Inteiro	int	32 bits
Inteiro	long	64 bits
Ponto Flutuante	float	32 bits
Ponto Flutuante	double	64 bits
Caracter	char	16 bits
Lógico	boolean	true / false

# O OBJETO

- Um objeto é a representação de um conceito/entidade do mundo real, que pode ser física ou conceitual e possui um significado bem definido para um determinado software.

1. Definir uma classe
2. Instanciar os objetos



# EXEMPLOS

- **Software:** Fluxo de Caixa
- **Classe:** Conta
- **Objetos representados:**

## Conta de Luz

Total a ser pago

Nome do Fornecedor

## Conta de água

Total a ser pago

Nome do Fornecedor

## Conta de Telefone

Total a ser pago

Nome do Fornecedor

Personagem
+nome +cor +quantidadeDeEstrelas +arquetipo
+pular() +pegarEstrelas(Estrelas estrelas) +criarPersonagem() +getNome() +setNome(String nome)

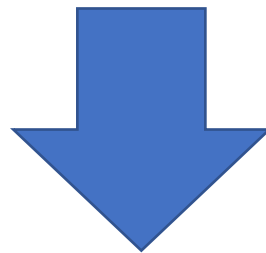
Criando um objeto

```
Personagem personagem = new Personagem
```

# GETTERS E SETTERS

- O modificador `private` faz com que ninguém consiga modificar, nem mesmo ler, o atributo.
- Para acesso aos atributos de maneira controlada
  - criar dois métodos, um que retorna o valor e outro que muda o valor.

```
private String nome;
```



## ATRIBUTO

```
class Conta {  
    private double saldo;  
  
    public double getSaldo() {  
        return this.saldo;  
    }  
  
    public void setSaldo(double saldo) {  
        this.saldo = saldo;  
    }  
}
```

# COMENTÁRIOS

DELIMITADOR	DESCRIÇÃO
//	Comentários de linha
/* */	Comentários de bloco (múltiplas linhas)
/** */	Comentários de documentação (múltiplas linhas)

# TIPOS DE DADOS PRIMITIVOS

CATEGORIA	TIPO	BYTES
Inteiro	byte	1
	short	2
	int	4
	long	8
Real	float	4
	double	8
Caractere	char	2
Lógico	boolean	1

		Valores possíveis				
Tipos	Primitivo	Menor	Maior	Valor Padrão	Tamanho	Exemplo
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;
	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d; ou double ex6 = 10.20;
Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;



# PALAVRAS RESERVADAS JAVA

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

# CARACTERES ESPECIAIS

REPRESENTAÇÃO	SIGNIFICADO
<code>\n</code>	Pula linha (newline ou linefeed)
<code>\r</code>	Retorno de carro (carriage return)
<code>\b</code>	Retrocesso (backspace)
<code>\t</code>	Tabulação (horizontal tabulation)
<code>\f</code>	Nova página (formfeed)
<code>\'</code>	Apóstrofe
<code>\"</code>	Aspas
<code>\\</code>	Barra invertida
<code>\u223d</code>	Caractere Unicode 223d
<code>\g37</code>	Octal 37
<code>\fca</code>	Hexadecimal FCA

# OPERADORES ARITMÉTICOS

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
+	Adição	Esquerda	$a + b$
-	Subtração	Esquerda	$a - b$
*	Multiplicação	Esquerda	$a * b$
/	Divisão	Esquerda	$a / b$
%	Resto da divisão inteira	Esquerda	$a \% b$
-	Sinal negativo (- unário)	Direita	$- a$
+	Sinal positivo (+ unário)	Direita	$+a$
++	Incremento unitário	Esquerda/Direita	$++a$ ou $a++$
--	Decremento unitário	Esquerda/Direita	$--a$ ou $a--$

# OPERADORES RELACIONAIS

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
==	Igual	Esquerda	a == b
!=	Diferente	Esquerda	a != b
>	Maior que	Esquerda	a > b
>=	Maior ou igual a	Esquerda	a >= b
<	Menor que	Esquerda	a < b
<=	Menor ou igual a	Esquerda	a <= b

# OPERADORES LÓGICOS

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
&&	E lógico (and)	Esquerda	a && b
	Ou lógico (or)	Esquerda	a    b
!	Negação (not)	Direita	!a

# OPERADORES DE ATRIBUIÇÃO COMPOSTA

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
<code>+=</code>	Adição e atribuição	Direita	<code>a += expr</code>
<code>- =</code>	Subtração e atribuição	Direita	<code>a - = expr</code>
<code>*=</code>	Multiplicação e atribuição	Direita	<code>a *= expr</code>
<code>/=</code>	Divisão e atribuição	Direita	<code>a /= expr</code>
<code>%=</code>	Divisão inteira e atribuição	Direita	<code>a %= expr</code>
<code>&amp;=</code>	E bitwise e atribuição	Direita	<code>a &amp;= expr</code>
<code> =</code>	Ou bitwise e atribuição	Direita	<code>a  = expr</code>
<code>^=</code>	Ou-exclusivo bitwise e atribuição	Direita	<code>a ^= expr</code>
<code>&gt;&gt;=</code>	Rotação à direita e atribuição	Direita	<code>a &gt;&gt;= expr</code>
<code>&lt;&lt;=</code>	Rotação à esquerda e atribuição	Direita	<code>a &lt;&lt;= expr</code>
<code>&gt;&gt;&gt;=</code>	Rotação à direita sem sinal e atribuição	Direita	<code>a &gt;&gt;&gt;= expr</code>

# OPERADORES BITWISE

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
&	E bit-a-bit (bitwise and)	Esquerda	a & b
	Ou bit-a-bit (bitwise or)	Esquerda	a   b
^	Ou-exclusivo bit-a-bit (bitwise xor)	Esquerda	a ^ b
~	Complemento de 2	Direita	~ a
<<	Rotação à esquerda de n bits	Esquerda	a << n
>>	Rotação à direita de n bits	Esquerda	a >> n
>>>	Rotação à direita sem sinal	Esquerda	a >>> n

Operadores bitwise são utilizados quando precisamos realizar operações em nível de bits com números inteiros, ou seja, trabalhar com sua representação binária.

# ESTRUTURAS DE CONTROLE

## Diretiva FOR

```
class EstruturaFor {  
    public static void main(String[] args) {  
        int j;  
        for (j=0; j<10; j++) {  
            System.out.println(j);  
        }  
    }  
}
```



# ESTRUTURAS DE CONTROLE

## Diretiva WHILE, DO WHILE

```
class DiretivaWhile{  
    public static void main(String[] args) {  
        int j = 10;  
        while (j >= 0){  
            System.out.println(j);  
            j--;  
        }  
    }  
}
```

# ESTRUTURAS DE CONTROLE

## Diretiva WHILE, DO WHILE

```
public class Main{  
    public static void main(String[] args) {  
        int i = 0;  
        do{  
            System.out.println(i);  
            ++i;  
        }while(i <= 10);  
    }  
}
```

# ESTRUTURAS DE CONTROLE

## Diretiva IF ELSE, SWITCH

```
class Main{
    public static void main(String[] args){
        int idade = 17;
        String sexo = "h";
        if(idade >= 18){
            System.out.println("idade menor que 18");
        }else if(sexo == "f"){
            System.out.println("sexo diferente de h");
        }else{
            System.out.println("idade > 18 e sexo h");
        }
    }
}
```

# ESTRUTURAS DE CONTROLE

## Diretiva IF ELSE, SWITCH

```
class Main{
    public static void main(String args[]) throws java.io.IOException {

        System.out.println("Concorda ou não? (s/n) ");
        char c = (char) System.in.read();
        switch(c) {
            case 's':
                System.out.println("Concorda");
                break;
            case 'n':
                System.out.println("Não concorda");
                break;
            default: System.out.println("Resposta inválida");
                break;
        }
    }
}
```

# ESTRUTURAS DE CONTROLE

## Estruturas de Controle Erros (TRY CATCH e TRY CATCH FINALLY)

```
class Main{
    public static void main(String args[]){
        String frase = null;
        String novaFrase = null;
        try{
            novaFrase = frase.toUpperCase();
        }catch(NullPointerException e){
            System.out.println("O frase inicial está nula, para solucionar tal o problema, foi lhe atribuido um valor default.");
            frase = "Frase vazia";
            novaFrase = frase.toUpperCase();
        }
        System.out.println("Frase antiga: "+frase);
        System.out.println("Frase nova: "+novaFrase);
    }
}
```

# ESTRUTURAS DE CONTROLE

## Estruturas de Controle Erros (TRY CATCH e TRY CATCH FINALLY)

```
class Main{
    public static void main(String args[]){
        int j = 5;
        try{
            j = 6;
        }catch (Exception e){
            System.out.println("Argumento invalido ou ausente. Usando default.");
        }finally {
            while (j >= 0){
                System.out.println(j);
                j--;
            }
        }
    }
}
```

# ESPECIFICADORES E MODIFICADORES ESPECIAIS

MODIFICADOR	DESCRIÇÃO	APLICAÇÃO
abstract	Indica que a classe não é concreta (não pode ser instanciada), pois contém membros não implementados, mas cuja interface já está definida.	Herança
final	Indica que a classe ou o membro particular de uma classe não pode ser modificado por meio da herança.	Herança
native	Indica ao compilador que o método é implementado em outra linguagem de programação (C, C++, assembly) de maneira específica da plataforma.	JNI
static	Define quando um membro de uma classe passa a pertencer a classe em si, não precisando ser acessado por variáveis de instância.	Geral
strictfp	Indica ao compilador que o código gerado deve aderir estritamente à especificação para cálculos numéricos, garantindo resultados idênticos mesmo em plataformas diferentes.	Geral
synchronized	Indica que o método só pode ser acessado por uma thread de cada vez, exigindo o uso de um monitor por parte da JVM.	Threads
transient	Indica que o campo referindo não deve ser serializado.	Serialização
volatile	Indica campos que podem ser modificados por várias threads simultâneas, habilitando a manutenção de cópias de trabalho em cada thread e evitando otimizações inadequadas.	Threads



**OBRIGADO!**

*Diego Marques de Carvalho*