

Laboratório de Programação

JAVA

Parte 01

Fundamentos da Programação Orientação a Objetos

Criada com o intuito de aumentar as propriedades desejáveis em relação a programação estruturada como:

- Modularização;
- Reutilização;
- Manutenibilidade.

Fundamentos da Programação Orientação a Objetos

- Na POO, implementa-se um conjunto de **classes** que definem os **objetos** presentes no *software*.
- Cada **classe** determina o **comportamento** (definidos nos **métodos**) e **estados possíveis (atributos)** de seus **objetos**, assim como o relacionamento com outros **objetos**.

Fundamentos da Programação Orientação a Objetos

Introduz um conjunto de conceitos novos:

- Classes;
- Objetos;
- Herança;
- Polimorfismo,
- Encapsulamento.

Java e C++ são exemplos de linguagens Orientadas a objeto.

Fundamentos da Programação Orientação a Objetos

Programação Orientada a Objetos	Programação Estruturada
Classes	Tipos de dados definidos pelo usuário
Métodos	Procedimentos e funções
Atributos	Variáveis
Herança	-
Polimorfismo	-
Encapsulamento	-

Linguagem Java

É uma linguagem criada por James Gosling em 1995 com o intuito de desenvolver sistemas para eletrodomésticos. Ela possui uma sintaxe semelhante a linguagem C, sendo mais simples e robusta (menos bugs!).



Linguagem Java

É uma linguagem que pode ser utilizada para desenvolvimento para desktop, web e desenvolvimento para micro dispositivos.

Para tanto, existem três plataformas distintas:

- J2SE (Standard Edition) – Para desenvolvimento Desktop;
- J2EE (Enterprise Edition) – Para desenvolvimento Web;
- J2ME (Micro Edition) – Para desenvolvimento para Micro Dispositivos.

Características da Linguagem Java

- **Sensitive case:** Diferencia maiúsculas de minúsculas;
- **Independente de plataforma:** Programas Java podem ser executados em diferentes sistemas operacionais;
- **Orientada à objetos:** Disponibiliza mecanismos como herança, polimorfismo, encapsulamento;
- **Coleta de lixo “*Garbage Collection*”:** Desaloca memória não utilizada, o mecanismo checa a memória automaticamente liberando as porções que não serão mais utilizadas;
- **API Java:** Conjunto de classes utilitárias que disponibiliza funcionalidades como interface gráfica, mecanismos de acesso a banco de dados e a arquivos;
- **JVM (Java Virtual Machine):** Para executar um programa Java precisamos de uma máquina virtual. Esta é dependente de plataforma (Sistema operacional).

Funcionamento de um Programa Java

- O código é criado em um arquivo com a extensão **.java**;
- Este arquivo passa pelo processo de compilação gerando um arquivo **.class** conhecido como Bytecode,
- O arquivo pode ser disponibilizado em qualquer sistema operacional que possua uma JVM instalada.

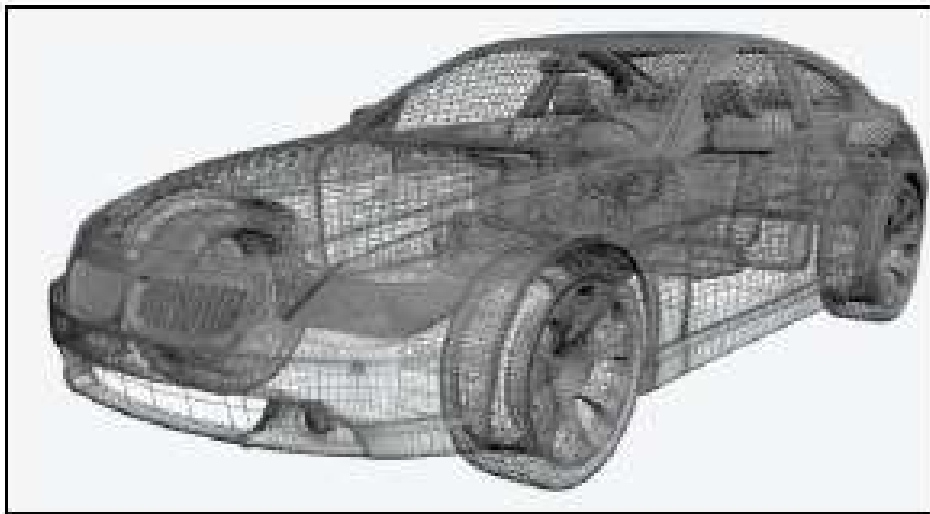
Conceito de Classe

É um dos conceitos mais utilizados em **Java**, ela é a base para criação de **objetos**.

Analogia:

- A partir de uma fôrma podemos criar tantos bolos quanto necessário.
 - Da mesma forma podemos criar vários **objetos** a partir de uma única **classe**.
- Para criarmos o bolo é preciso termos a fôrma.
 - Da mesma maneira, para criar um **objeto** precisamos ter a **classe** antes.

Conceito de Classe



Classe



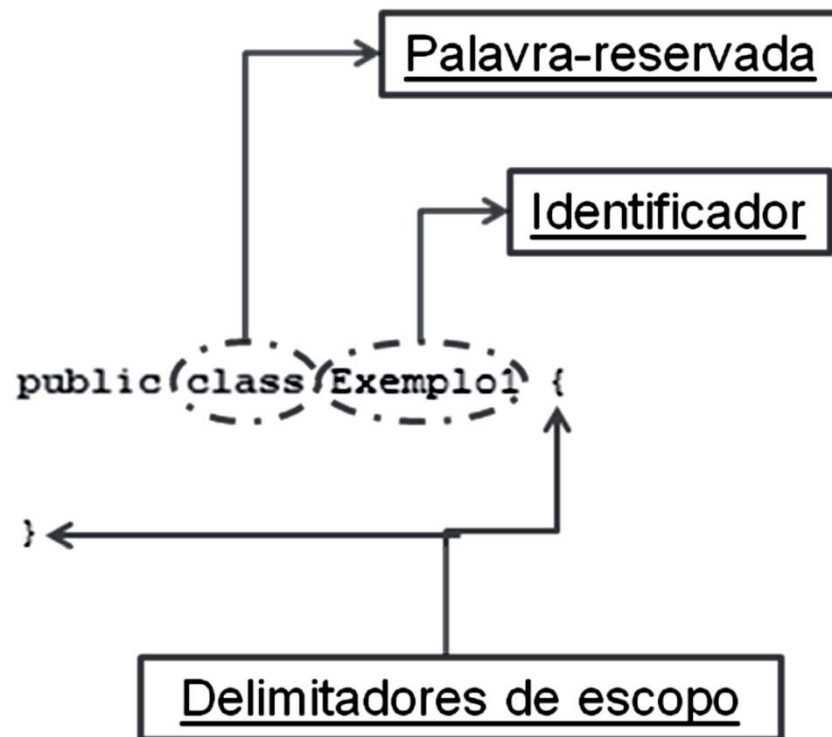
Objeto

Conceito de Classe

- Representa um conjunto de **objetos** com suas próprias características.
- Uma **classe** define o comportamento dos **objetos**, através de **métodos** e quais estados (características) ele é capaz de manter, através de **atributos**.
- Exemplo de classe:
 - Seres humanos
 - Carro

Conceito de Classe

Definição de **classes** em **Java**:

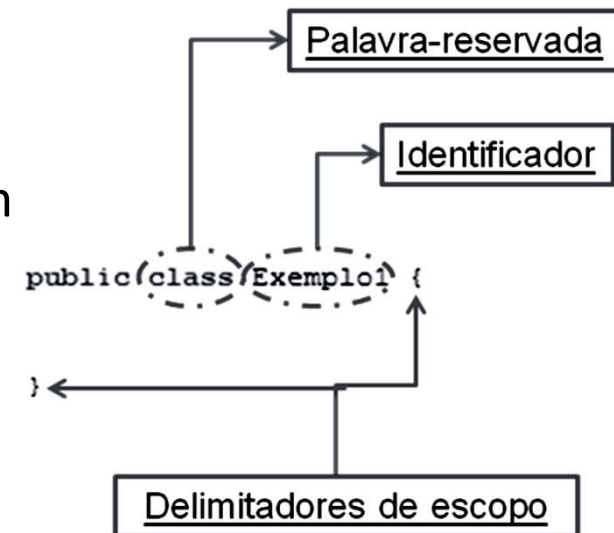


Conceito de Classe

Devem ter nomes como Casa, Funcionario, Execucao ...

Para evitar erros na escolha do **identificador de classes**, seguem algumas regras:

- O identificador não pode conter espaços;
- Deve ser iniciado com uma letra, _ (símbolo sublinhado) ou \$;
- Pode conter números a partir da 2ª posição;
- Não podemos utilizar caracteres especiais (*, -, (,), %, #, @, !, ", ', " , , , , . , + , = , < , > , : , ; , ? , | , \ , / ,) no identificador da classe;



Conceito de Classe

Palavras reservadas da Linguagem **Java**:

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

Conceito de Classe

Exemplos **inválidos** de identificadores de **classes**

```
public class default{    public class continue{    public class super{  
    }                    }                    }
```

```
public class Classe Principal{    public class Conta Corrente{  
    }                            }
```

```
public class 2010{        public class 000_{  
    }                      }
```

```
public class Sistema*De/Biblioteca{    public class Aula1(Pratica){  
    }                                    }
```

```
public class Trabalho-01{    public class @Teste{  
    }                        }
```

Conceito de Classe

Exemplos **válidos** de identificadores de **classes**

```
public class Operadores{  
}
```

```
public class Funcionario{  
}
```

```
public class ClasseTeste{  
}
```

```
public class _Hello{  
}
```

```
public class Teste2{  
}
```

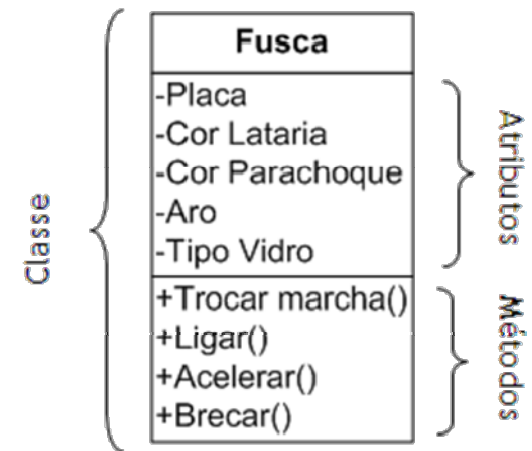
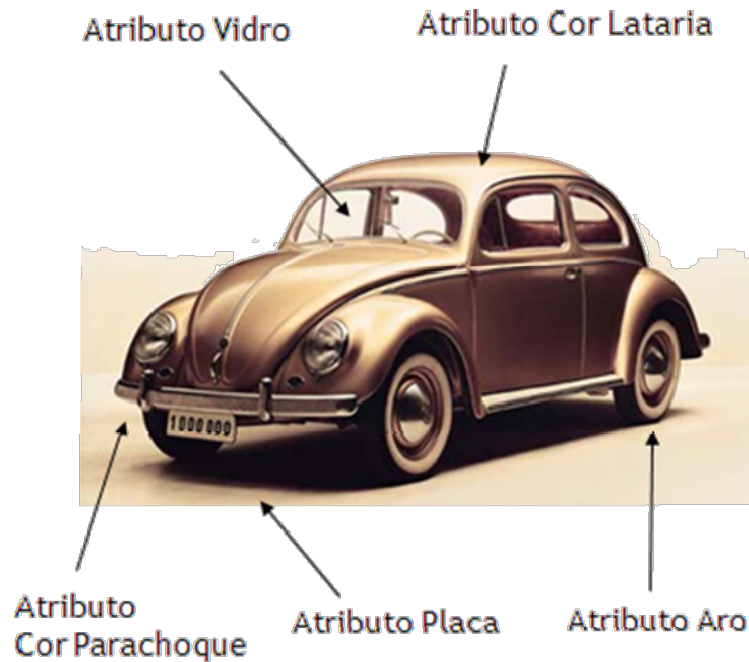
```
public class _Carro{  
}
```

```
public class $Remuneracao{  
}
```

Conceito de Classe

Uma classe pode conter:

- **Atributos;**
- **Métodos.**

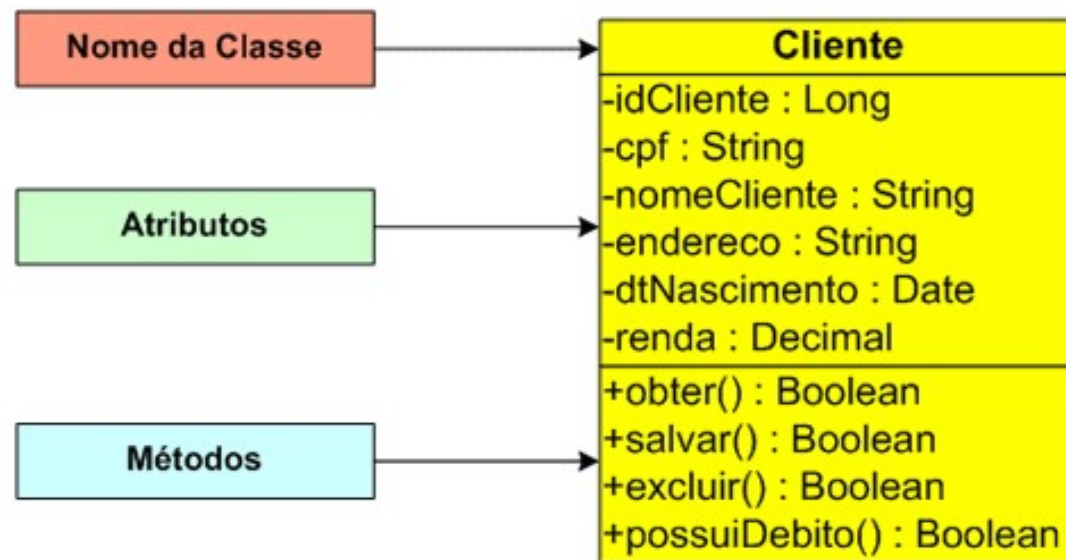


Ações do carro (Métodos)

Liga (Ignição)
Acelera
Freia (Breca)
Troca de Marcha

Atributos

- São entidades que guardam valores de um tipo de dados;
- É como são chamadas as variáveis e constantes em orientação a objetos.



Atributos

- **Atributos** possuem somente um **tipo** e um **identificador**.
- **Identificador** é o nome do **atributo** e o **tipo** define o tipo de dado que o atributo pode receber.

Os principais tipos primitivos em Java são:

Principais tipos	Valores possíveis
boolean	true; false
char	Um caractere: 'a' ... 'z'; 'A' ... 'Z'; '1' ... '9' ...
byte	Inteiros
short	Inteiros
int	Inteiros
long	Inteiros
float	Ponto Flutuante
double	Ponto Flutuante
void	-

Atributos

- Normalmente é utilizado internamente na **classe**

(A exposição das características da classe é tarefa dos **métodos**).

Exemplo:

```
public class Livro
{
    // Atributos
    private string titulo;
    private short ano;
}
```

Atributos

Para evitarmos erros na escolha do **identificador** de **atributos**, seguem algumas regras:

- A primeira palavra do identificador deve começar com letra minúscula e as demais palavras podem ser maiúsculas;
- Os atributos são definidos dentro da classe;
- Não podemos definir dois ou mais atributos na mesma classe com o mesmo identificador;
- Não podemos identificar um atributo com uma palavra reservada; (idem classes);
- O identificador não pode conter espaços; (idem classes);
- Deve ser iniciado com uma letra, _ (símbolo sublinhado) ou \$;
- Pode conter números a partir da 2ª posição; (idem classes);
- Não podemos utilizar caracteres especiais (*, -, (,), %, #, @, !, ", ', ", , , ., +, =, <, >, :, ;, ?, |, \, /,) no identificador do atributo. (idem classes);

Atributos

Exemplos **inválidos** de **identificadores de atributos**:

```
public class Exemplo2 {  
    int x;  
}
```

int z;

*Definição inválida de um atributo,
atributo definido fora da classe.*

```
int z;  
public class Exemplo2 {  
    int x;  
}
```

*Definição inválida de um atributo,
atributo definido fora da classe.*

```
public class Exemplo2 {  
    double num;  
    double num;  
}
```

*Definição inválida de atributos,
dois atributos com o mesmo
identificador na mesma classe.*

Atributos

Exemplos **válidos** de **identificadores** de **atributos**:

`public class Exemplo2 {
 int x;
}`
Definição válida de um atributo

`public class Exemplo2 {
 double num;
 double x;
}`
Definição válida de atributos

Atributos

Adicionalmente, podemos inicializar o atributo no momento da definição.

- A seguir uma ilustração da definição de um atributo juntamente com sua inicialização.

```
public class Uab{  
    int codigoCurso = 1;  
}
```

Métodos

Métodos definem o comportamento (operações possíveis), é como são chamadas as **funções** e **procedimentos** em orientação a objetos.

São criados quando queremos implementar algum comportamento, por exemplo:

- Tirar férias;
- Calcular o índice de massa corporal;
- Fazer uma soma;
- Ler dados do teclado;
- Salvar no banco de dados.

Métodos

Cada método possui um tipo de retorno, um identificador, um conjunto (zero ou mais) de argumentos (valores que podem ser recebidos) e uma implementação.

A implementação do método é delimitada por chaves.

```
public class ExemploMetodo{  
    public void teste(int num1, int num2){  
    }  
}
```

Tipo de retorno Identificador Argumentos

Métodos

Para evitarmos erros na escolha do **identificador** de **métodos**, seguem algumas regras:

- A primeira palavra do identificador deve começar com letra minúscula e as demais palavras podem iniciar com maiúsculas. **Não ocorrerá erro**, mas é uma convenção importante;
- As mesmas regras de identificadores de classes.

Métodos

- Métodos podem devolver valores.
- Quando o tipo de retorno é diferente de **void**, dentro do método devemos retornar um valor adequado ao tipo a ser retornado.
- Neste caso podemos retornar o valor literal (1 para inteiros, 'a' para char...), retornar o conteúdo de um atributo ou resultado de uma operação.

Métodos

- A palavra reservada **return** é utilizada para esta finalidade.

```
public class Operacao{  
    public int soma(int num1, int num2){  
        return num1+num2;  
    }  
}
```

Neste exemplo temos um método soma que retorna um inteiro resultante da soma de dois inteiros (num1 e num2 são atributos inteiros);

```
public class Operacao{  
    public int soma(){  
        return 3+2;  
    }  
}
```

Neste exemplo temos um método soma que retorna um inteiro resultante da soma de dois inteiros (3 e 2);

Métodos

Qualquer método possui dois momentos:

1º (Momento da definição): O comportamento que o método é definido;

2º Momento (Momento da chamada): Quando um método que **já foi definido** é acessado para processar as operações.

```
public class Operacao{  
    int numero;  
    public void atribui(int num){  
        numero = num;  
    }  
}
```

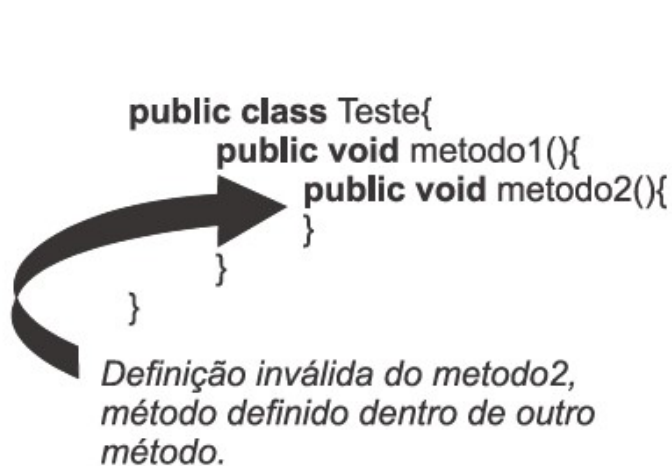
Definição do método atribui.

```
public class Operacao{  
    int numero;  
  
    public void acessa(){  
        atribui(10);  
    }  
  
    public void atribui(int num){  
        numero = num;  
    }  
}
```

Chamada do método atribui pelo método acessa.

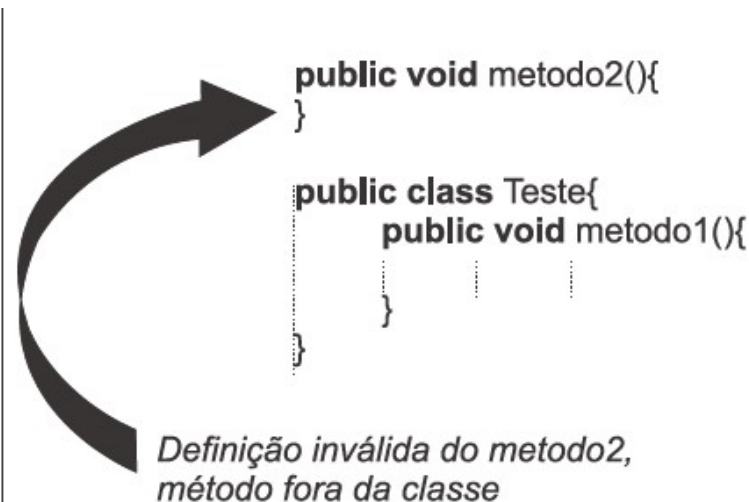
Métodos

Nunca esqueça que métodos não podem ser definidos (**1º momento**) dentro de outros métodos. Além disto, não podemos definir métodos fora da classe.



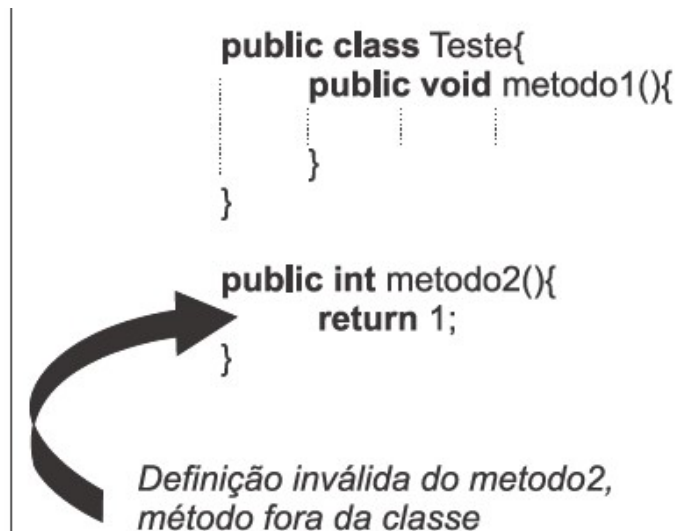
```
public class Teste{  
    public void metodo1(){  
        public void metodo2(){  
        }  
    }  
}
```

Definição inválida do metodo2, método definido dentro de outro método.



```
public void metodo2(){  
}  
  
public class Teste{  
    public void metodo1(){  
    }  
}
```

Definição inválida do metodo2, método fora da classe



```
public class Teste{  
    public void metodo1(){  
    }  
}  
  
public int metodo2(){  
    return 1;  
}
```

Definição inválida do metodo2, método fora da classe

Exemplo de Classes

- Exemplo de uma classe de execução:

```
public class HelloWorld{  
    Public static void main(String args[]){  
        System.out.println("Hello World!");  
    }  
}
```

Exemplo de Classes

- Exemplo de uma classe base simples:

```
public class Aluno {  
    String nome;  
    int idade;  
  
    public void atualizaldade(){  
    }  
}
```

Comentários em Classes

Existem três tipos possíveis de comentário:

- comentário de uma linha;
- comentário de várias linhas;
- comentário de documentação.

```
// Este comentário é comentário de uma linha.  
public class Teste{  
    /*  
        Comentário de várias linhas  
    */  
    int x;  
    /**  
        Comentário de documentação  
    */  
    public static void main(String args[]){  
    }  
}
```

Objetos

- É uma ***instância*** de uma ***classe***.
- Um objeto é capaz de armazenar ***estados*** através de seus ***atributos*** e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos.

Objetos

- Em Java os objetos são manipulados a partir de **referências** e são instanciados dentro de uma **classe** (ou dentro dos **métodos da classe**).
- A palavra-reservada **new** é utilizada para **instanciar objetos**, seguida do tipo de classe utilizado para criação do objeto e de parêntesis.

```
public class Execucao{  
    public static void main(String args[]){  
        Aluno aluno1 = new Aluno();  
    }  
}
```