Programação Orientada a Objetos

Desenvolvimento de Sistemas

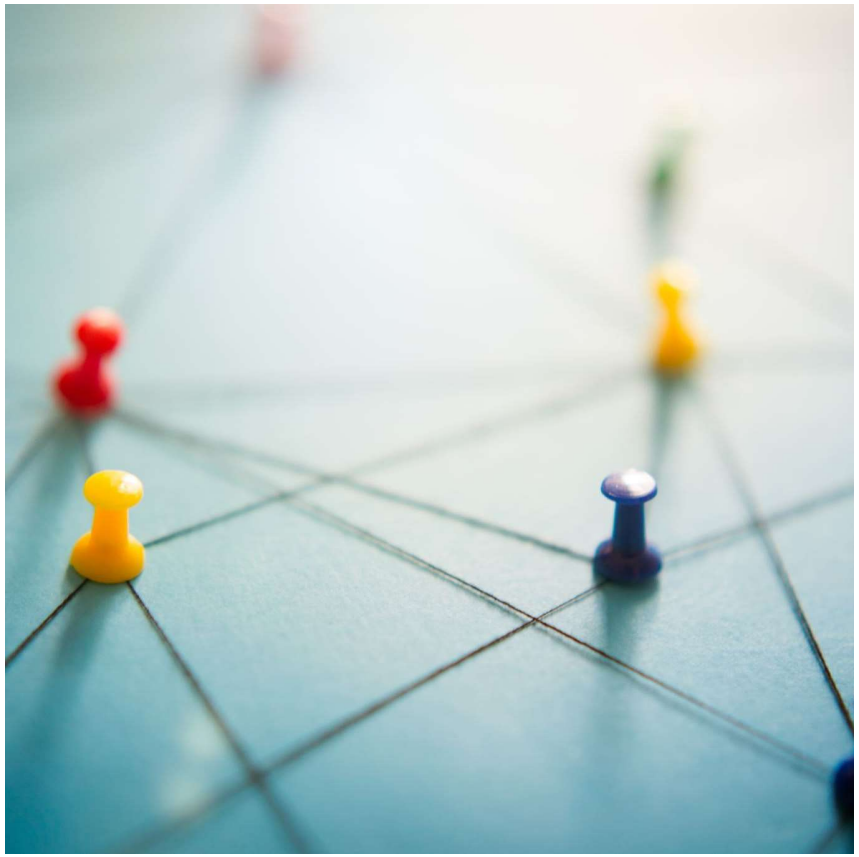
Agenda

Programação Orientada a Objeto (POO)

Conceitos sobre POO

- Pilares da POO
- Objeto
- Classe
- Construtores
- Troca de Mensagens
- Encapsulamento

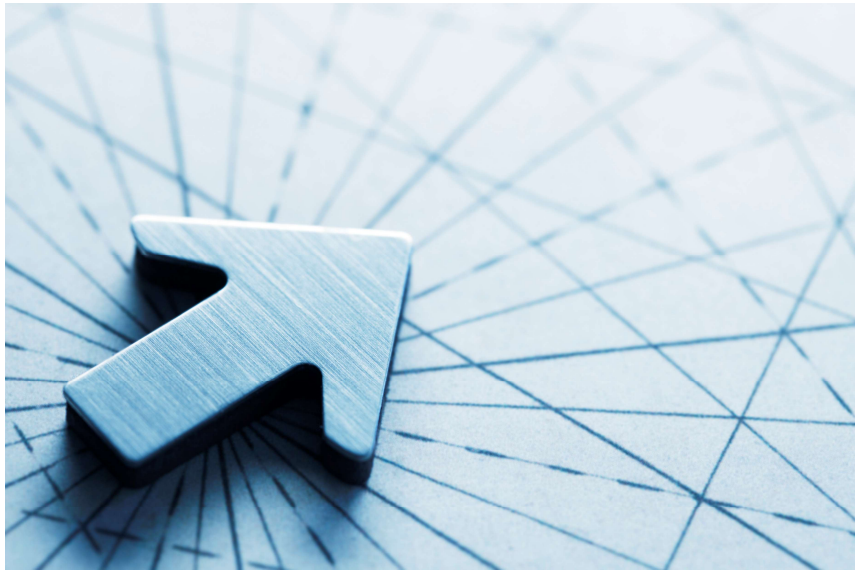
Programação Orientada a Objetos (POO)



- A Programação Orientada a Objetos (POO) é um paradigma de programação de computadores que usa os conceitos de **Objetos** e **Classes** como elementos centrais para representar e processar dados usados nos programas.

- **Você sabe o que um Paradigma?**
  - Segundo o dicionário Houaiss, paradigma significa modelo, padrão, exemplo.

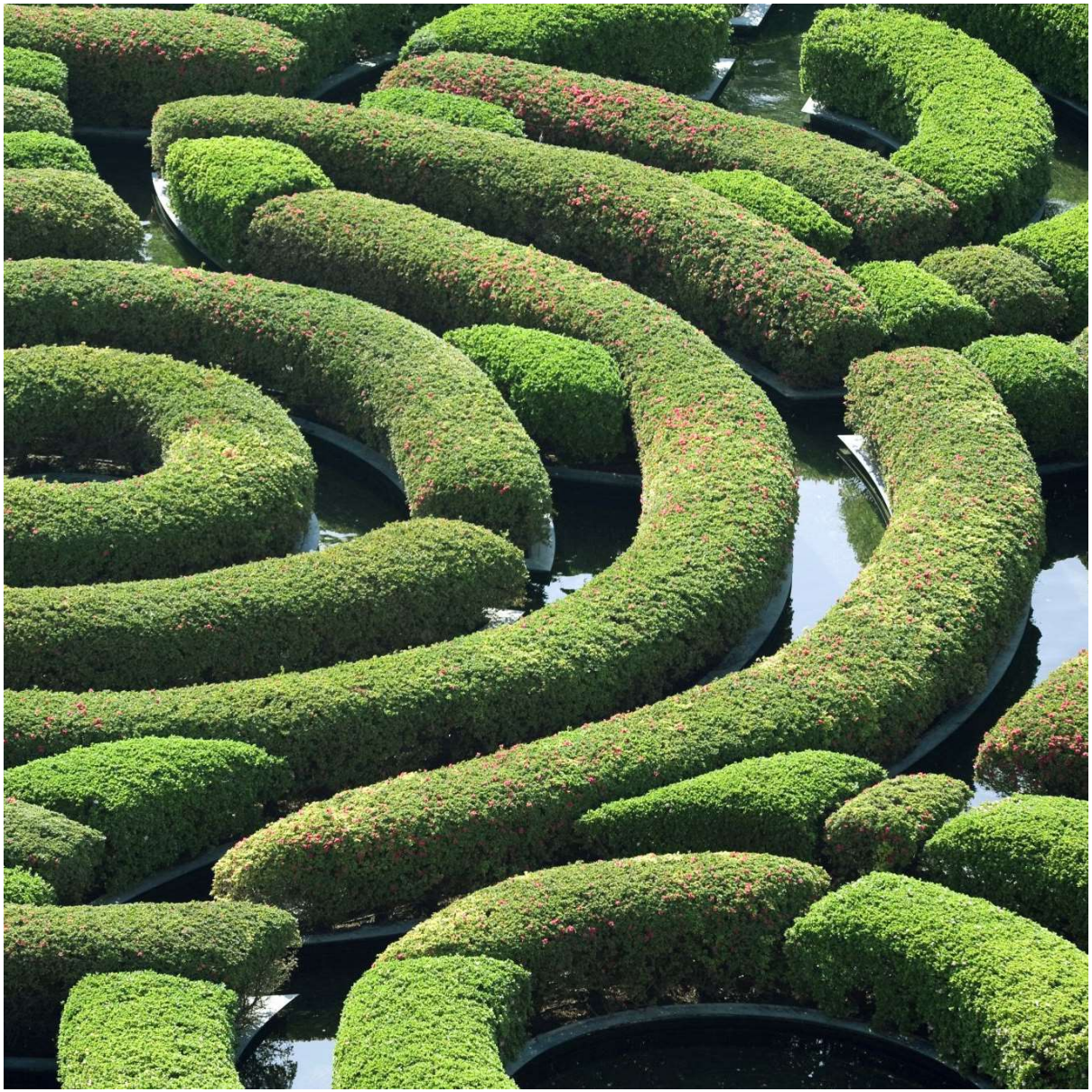
Programação Orientada a Objetos (POO)



Os conceitos da programação orientada a objetos (POO) surgiram no final da década de 1960, com a linguagem Simula-68.

Posteriormente amadurecidos e aprimorados durante a década de 1970 pela linguagem de programação Smalltalk.

A popularização da POO só se deu ao longo da década de 80 e 90, com as linguagens C++ e Java.



Programação Orientada a Objetos (POO)

Um dos principais pesquisadores que introduziu os conceitos de POO, foi o cientista Alan Kay da Xerox, um dos criadores da linguagem Smalltalk.

Durante suas pesquisas, Alan desenvolveu a ideia de que poderíamos construir um programa usando conceitos e abstrações do mundo real, como objetos, troca de mensagens.

Agenda

Programação Orientada a Objeto (POO)

Conceitos sobre POO

- Pilares da POO
- Objeto
- Classe
- Construtores
- Troca de Mensagens
- Encapsulamento

Conceitos da Programação Orientada a Objetos

- A Programação Orientada a Objetos está sedimentada sobre quatro pilares derivados do princípio da abstração, são eles:
  - **Encapsulamento,**
  - **Herança,**
  - **Composição e**
  - **Polimorfismo.**

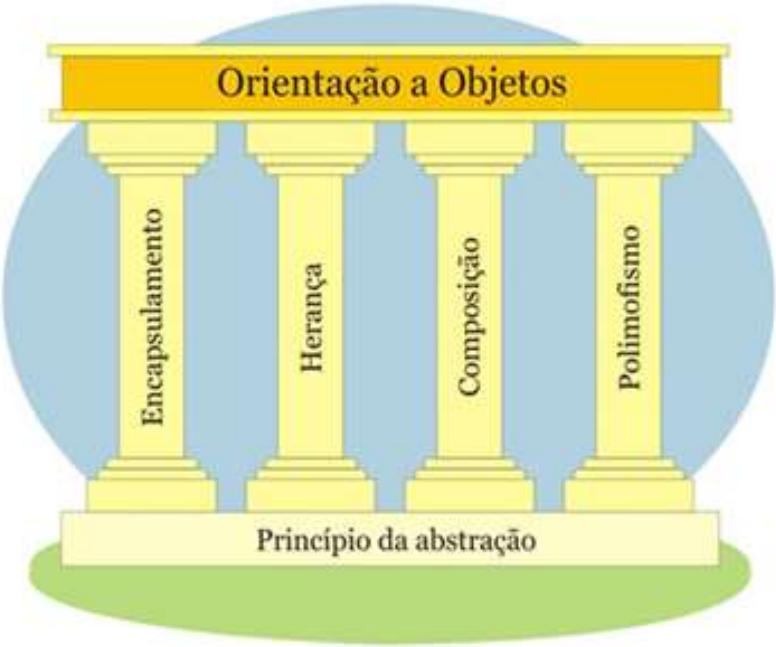


Figura 4 – Pilares da Orientação a Objetos

Conceitos da Programação Orientada a Objetos

O **princípio da abstração** é a capacidade de abstrair a complexidade de um sistema e se concentrar em apenas partes desse sistema.

Exemplo: um médico torna-se um especialista em algum órgão do corpo (exemplo, o coração). Ele abstrai sem desconsiderar as influências dos outros órgãos e foca apenas sua atenção nesse órgão.

Objetos

Na programação OO, objetos são usados para representar entidades do mundo real ou computacional.

Se observarmos ao nosso redor, veremos várias entidades ou abstrações as quais podem ser representadas como objetos no nosso programa.

As pessoas e seus carros podem ser vistas como objetos.

Objetos

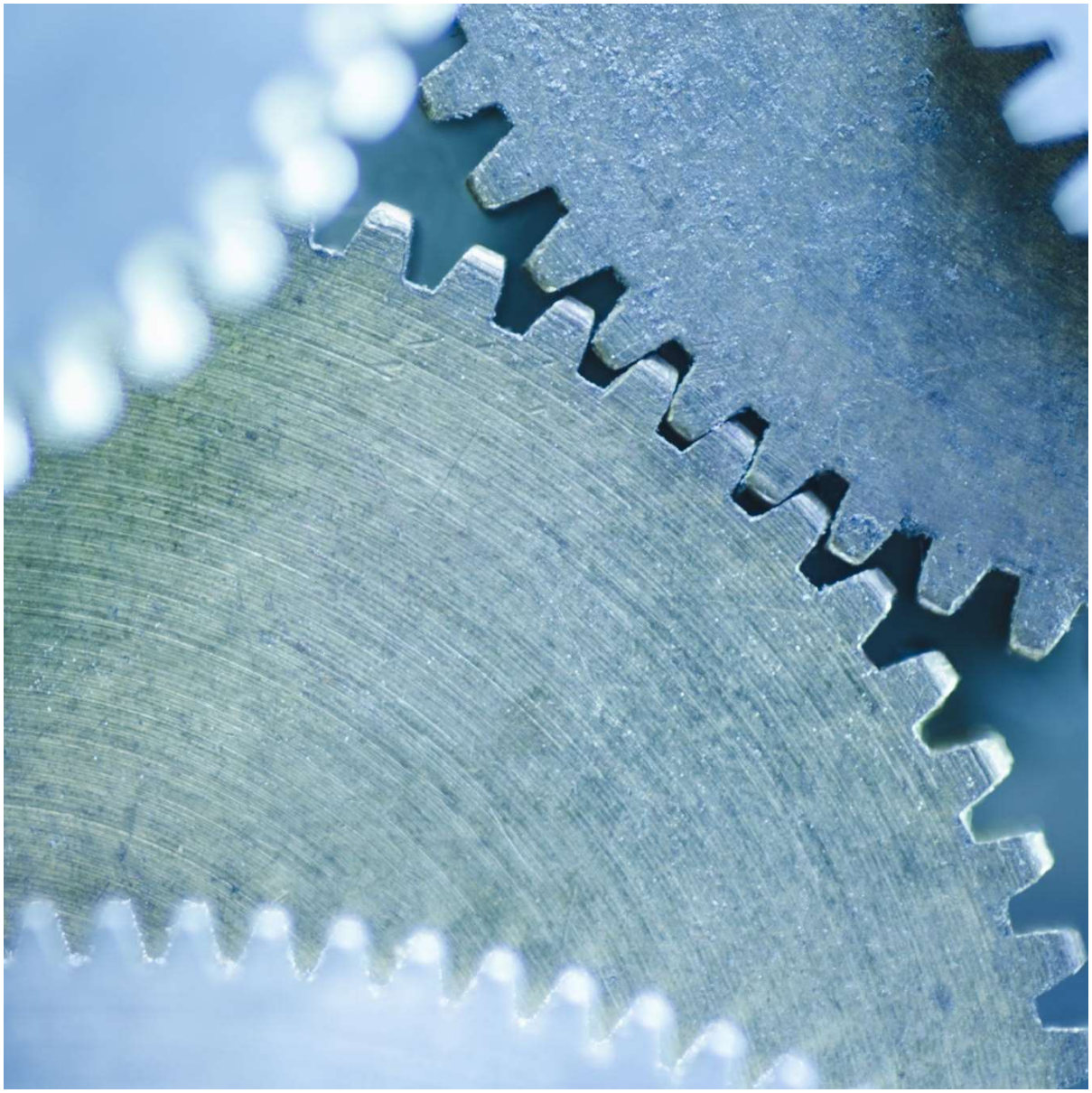
Os objetos possuem:

Características pelas quais os identificamos e Finalidades para as quais os utilizamos.

Objetos: Atributos

Características são tipicamente chamadas de atributos.

- O objeto Pessoa possui RG, nome, data de nascimento, etc.
- O objeto Carro possui tipo, cor, quantidade de portas.



Objetos: Métodos

Objetos podem também ter comportamentos associados.

- Um objeto do tipo Pessoa pode andar, correr ou dirigir carros.
- Um objeto do tipo Carro pode ligar, desligar, acelerar, frear.

Objetos

Na POO os objetos possuem características e comportamentos.

As características também podem ser chamadas de dados ou atributos.

Enquanto os comportamentos também podem ser chamados de operações ou métodos.

Objetos

Exemplos

Características: (dados, atributos)
tipo: Ferrari
placa: KZE1018
cor: vermelha
número de portas: 2

Comportamentos: (operações, métodos)
ligar
desligar
acelerar
frear

Características:
nome: Camila
cor do cabelo: negro
biotipo: magro

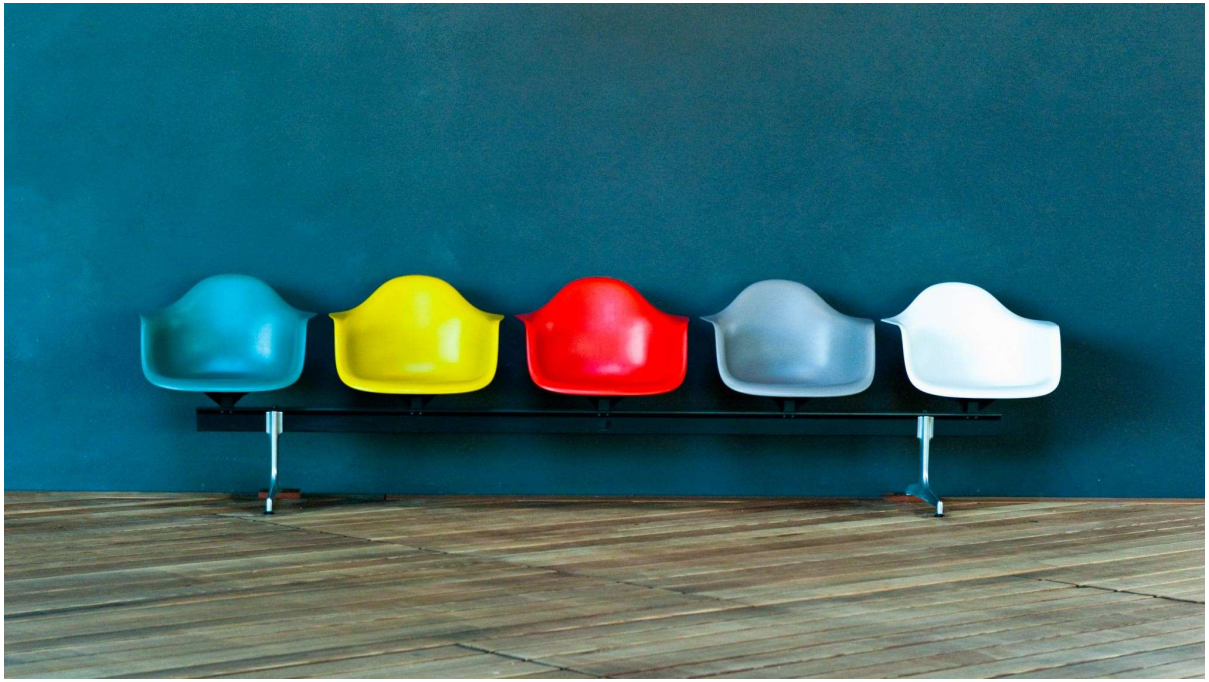
Comportamentos:
andar
correr
dirigir Carro

Figura 5 –Exemplos de objetos

Classes

Carros sempre possuirão os atributos tipo, cor e número de portas.

Pessoas sempre possuirão nome, data de nascimento e, possivelmente, um RG.



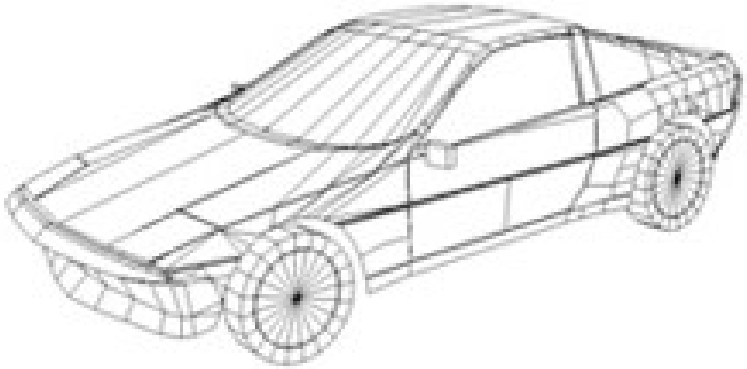
Classes

A modelagem e programação de um conjunto de objetos que possuem características (atributos) e comportamentos (métodos) comuns é feita na POO usando o conceito de **Classe**.

- Cada classe funciona no fundo como um molde para a criação de um dado objeto.
- Os objetos são vistos como representações concretas (instâncias) das classes.

Classes

- A classe define que objetos devem ter tipo, cor, placa e número de portas, mas não indica explicitamente quais são seus valores.



Tipo: ?

Cor: ?

Placa: ?

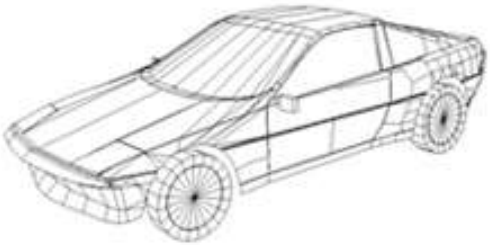
Número de Portas: ?

Figura 6 – Classe Carro

Classes

- Dois diferentes carros foram criados tomando como base a estrutura da classe.

CLASSE →



Tipo: ?

Cor: ?

Placa: ?

Número de Portas: ?

← OBJETOS →

Tipo: Porsche

Cor: Cinza

Placa: MHZ-4345

Número de Portas: 2

Tipo: Ferrari

Cor: Vermelho

Placa: JKL-0001

Número de Portas: 4

Criação de Classes

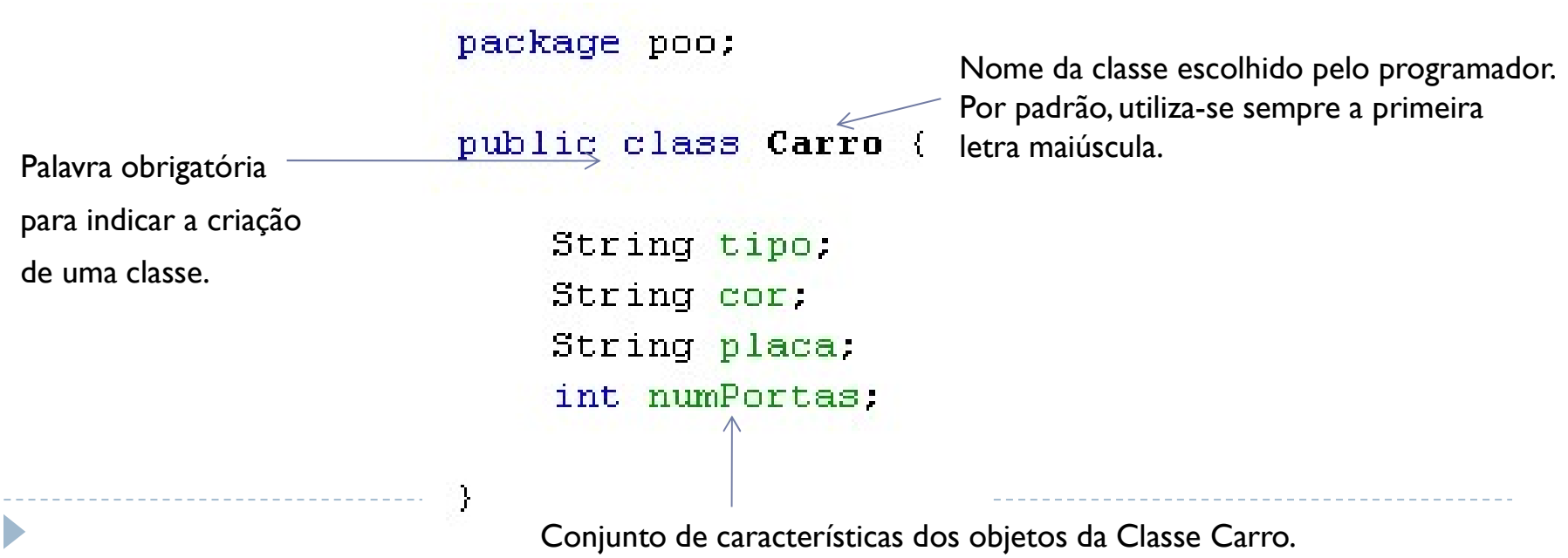
- A classe não serve para organizar no sentido de guardar os objetos, ela serve de modelo de construção.

A classe é o modelo ou molde de construção de objetos.
O modelo define as características e comportamentos que os objetos irão possuir.
A classe é abstrata (não existe concretamente).

Criação de Classes

- Vamos criar a classe Carro em Java.

1. Crie um nome Projeto chamado de POO.
2. Adicione uma nova classe java denominada Carro, clicando com o botão direito do mouse em cima do pacote poo.



Criação de Classes

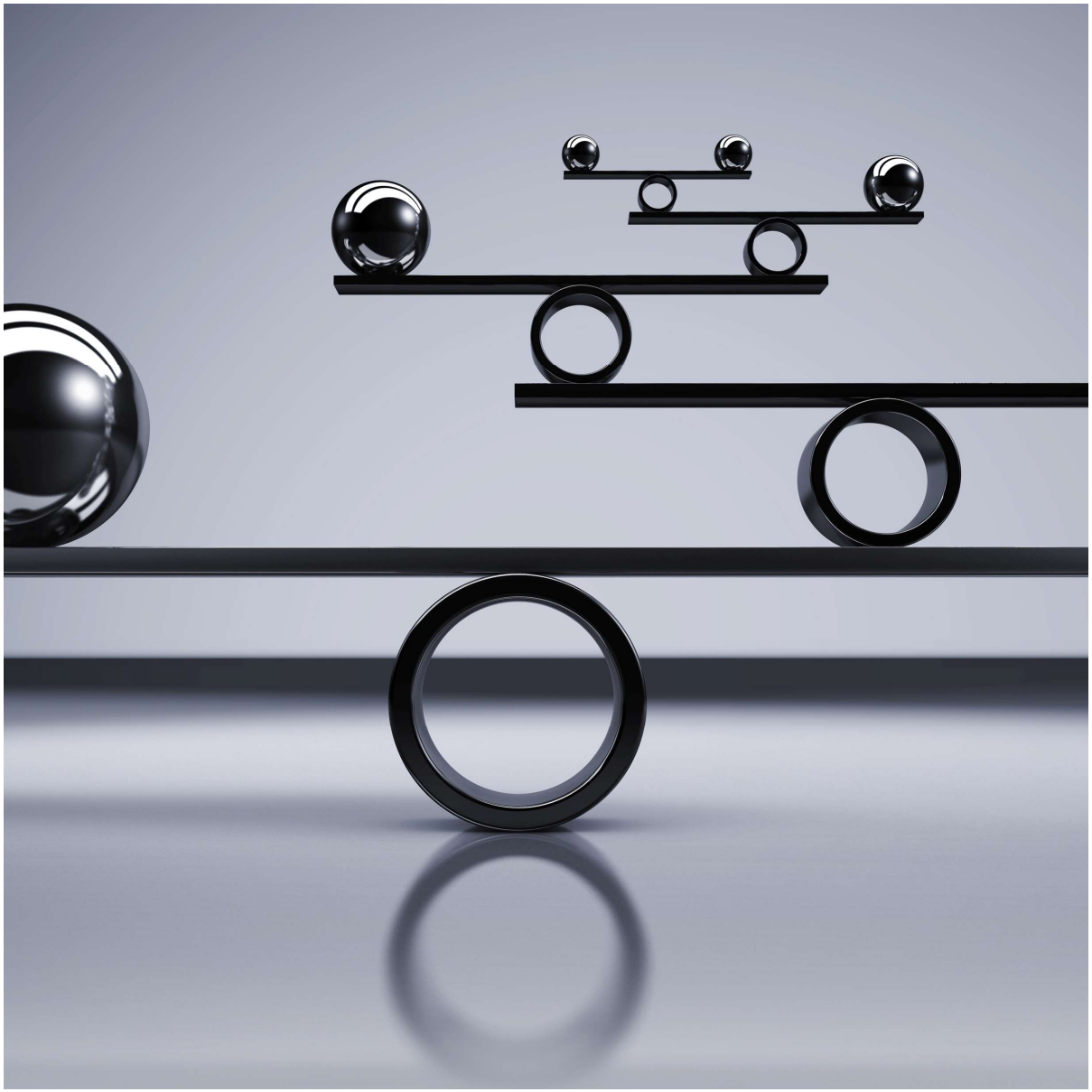
```
package poo;

public class Pessoa {
    String nome;
    String corDoCabelo;
    String biotipo;
    int idade;

}
```

- Perceba que não inserimos no código nada sobre o comportamento dos carros, como andar para frente ou para trás, virar a esquerda/direita, frear etc.
- Vamos criar a classe Pessoa em Java.

1. Adicione uma nova classe java denominada Pessoa, clicando com o botão direito do mouse em cima do pacote poo.



Adicionando Comportamento

Os objetos das classes Carro e Pessoa, precisam não só de suas características, mas também de ações possíveis de serem executadas.

Um carro precisa oferecer funções para que as pessoas os manobrem.

E pessoas, por sua vez, precisam desempenhar suas atividades, como andar, correr, estudar etc.

```java
package poo;

public class Carro {

    String tipo;
    String cor;
    String placa;
    int numPortas;

    void setCor(String cor){
        this.cor = cor;
    }

    public String getCor() {
        return cor;
    }

}
```

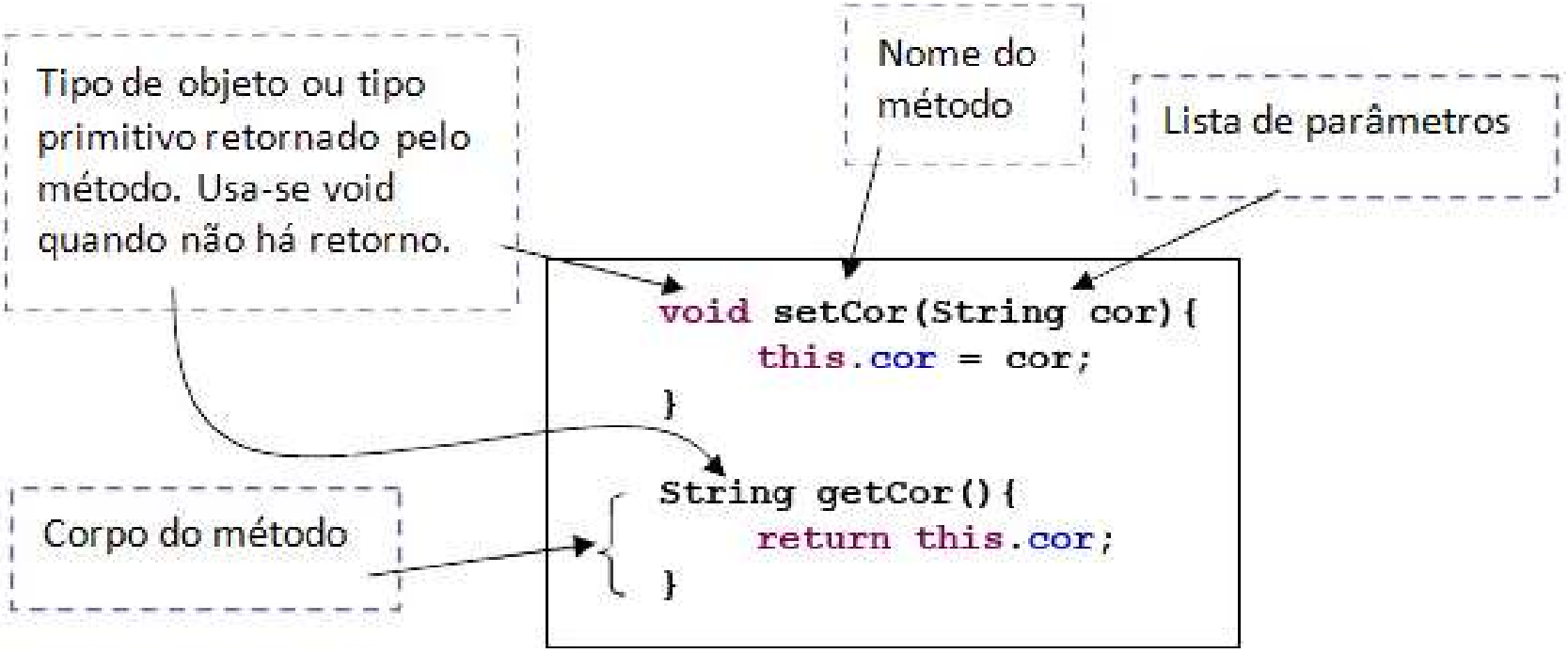
Adicionando Comportamento

Observe que adicionamos novas linhas ao código, agora com dois métodos *setCor(String cor)* e *getCor()*.

Esses métodos servem, respectivamente, para alterar o valor do atributo "cor" e retornar o valor desse atributo.

Adicionando Comportamento

A regra Java para definirmos os métodos é:



Adicionando Comportamento

```
public void setCor(String cor){
    this.cor = cor;
}
public String getCor() {
    return cor;
}
public void setTipo(String tipo) {
    this.tipo = tipo;
}
public String getTipo() {
    return tipo;
}
public void setPlaca(String placa) {
    this.placa = placa;
}
public String getPlaca() {
    return placa;
}
public void setNumPortas(int numPortas) {
    this.numPortas = numPortas;
}
public int getNumPortas() {
    return numPortas;
}
```

Atividade

- Faça o mesmo para a classe Pessoa.

Criando um Objeto

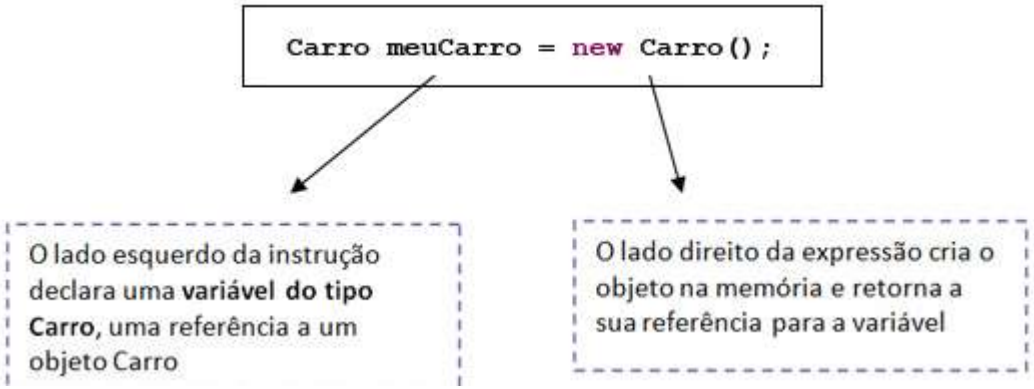
- Vá para o programa principal, onde existe o método main.

Criando um Objeto

- Vamos criar um objeto da classe Carro, no mundo da programação orientada a objetos chamamos isso de **instanciação**, ou seja, criaremos uma instância da classe Carro.

Instância = objeto

- Um objeto ou instância é criado através do operador **new**.



Criando um Objeto

- Toda a manipulação e consulta dos atributos do objeto serão feitas através de sua referência, ou seja, da variável.
- Define-se os atributos do carro utilizando os métodos da classe Carro.
- Utiliza-se o conjunto de métodos **setAtributo(valor)**.

```
meuCarro.setCor("PRETO");
meuCarro.setTipo("PASSEIO");
meuCarro.setPlaca("GGG-1111");
meuCarro.setNumPortas(4);
```

Criando um Objeto

- Exibindo cada atributo com seu respectivo método **getAtributo()**.

```
System.out.println("-----CARRO-----");
System.out.println("Cor: " + meuCarro.getCor());
System.out.println("Tipo: " + meuCarro.getTipo());
System.out.println("Placa: " + meuCarro.getPlaca());
System.out.println("Portas: " + meuCarro.getNumPortas());
```

```
run:
-----CARRO-----
Cor: PRETO
Tipo: PASSEIO
Placa: GGG-1111
Portas: 4
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Criando um Objeto - MAIN

```
package poo;

public class POO {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Carro meuCarro = new Carro();

        meuCarro.setCor("PRETO");
        meuCarro.setTipo("PASSEIO");
        meuCarro.setPlaca("GGG-1111");
        meuCarro.setNumPortas(4);

        System.out.println("-----CARRO-----");
        System.out.println("Cor: " + meuCarro.getCor());
        System.out.println("Tipo: " + meuCarro.getTipo());
        System.out.println("Placa: " + meuCarro.getPlaca());
        System.out.println("Portas: " + meuCarro.getNumPortas());
    }
}
```

Atividades

1. Instancie um objeto da classe Pessoa.
2. Atribuia aos métodos set's seus valores.
3. Imprima os valores através dos métodos get's.

Métodos Construtores

E se eu quiser, no momento da criação do objeto, passar valores para alguns de seus atributos, como placa e cor, por exemplo? É possível?

- É sim!!
- Para isso, define-se um tipo especial de método chamado **construtor**.
- E como seu próprio nome diz, ele constrói e objeto e nesse instante pode definir quaisquer atributos que desejarmos.

Métodos Construtores

- O método construtor é aquele chamado em um objeto, quando ele é criado. → new Carro();
  - Se a classe não tiver método construtor? O objeto ainda pode ser criado usando a instrução *new*.
- Definindo os construtores, pode-se estabelecer valores iniciais para os atributos do objeto.

Métodos Construtores

- Após os atributos da classe Carro, adicione o construtor abaixo.

```
public Carro(String tipo, String cor, String placa, int numPortas) {
    this.tipo = tipo;
    this.cor = cor;
    this.placa = placa;
    this.numPortas = numPortas;
}
```

- Observe que esse método possui o mesmo nome da Classe Carro e não possui tipo de retorno.

Métodos Construtores

- Existem duas maneiras de se construir um objeto derivado da classe Carro, ou do tipo Carro:
  - Com ou Sem inicialização dos valores dos atributos.

```
public Carro() {
}


public Carro(String tipo, String cor, String placa, int numPortas) {
    this.tipo = tipo;
    this.cor = cor;
    this.placa = placa;
    this.numPortas = numPortas;
}
```

Métodos Construtores

- Observe na classe Main.java as duas formas de construir objetos:

```
Carro meuCarro = new Carro();
Carro meuOutroCarro = new Carro("BRANCO", "PASSEIO", "GGG-0000", 2);
```

- As formas de acessos são as mesmas usando os métodos get's. Os métodos set's também funcionam para alterar os atributos em qualquer um dos casos.

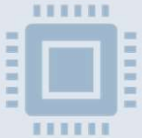
```
System.out.println("-----CARRO2-----");
System.out.println("Cor: " + meuOutroCarro.getCor());
System.out.println("Tipo: " + meuOutroCarro.getTipo());
System.out.println("Placa: " + meuOutroCarro.getPlaca());
System.out.println("Portas: " + meuOutroCarro.getNumPortas());
```

```
run:
-----CARRO1-----
Cor: PRETO
Tipo: PASSEIO
Placa: GGG-1111
Portas: 4
-----CARRO2-----
Cor: PASSEIO
Tipo: BRANCO
Placa: GGG-0000
Portas: 2
CONSTRUÍDO COM SUCESSO
```

Atividades

- Crie os construtores da classe Pessoa.
- Instancie dois objetos, cada um com um construtor diferente.
- Acesse os dois objetos, imprimindo seus valores.

Troca de Mensagens



Os objetos estão relacionados a outros, pois nenhum faz tudo sozinho, nem mesmo meu computador de última geração, pois sem a tomada na parede ele não é nada.

Os objetos precisam se "comunicar", ou seja, um objeto aciona um método de outro.

Essa comunicação é realizada através do mecanismo de **troca de mensagem**.

Troca de Mensagens

- Vejamos outro exemplo:

```
meuCarro.setCor("PRETO");
System.out.println("Cor: " + meuCarro.getCor());
```

- Esse código escrito dentro da classe Main envia uma mensagem para definir um atributo e outra para recuperar a placa do carro.
- Em Java, a troca de mensagem representa:
  - A mudança ou leitura do estado interno do objeto através da alteração de um de seus atributos; ou
  - A chamada a um dos métodos do objeto que representam seu comportamento e as tarefas que são capazes de desempenhar.

Troca de Mensagens

- Bem, quando relacionamos coisas no mundo real fazemos isso de forma natural e espontânea.
- Vejamos, quantas pessoas cabem em um automóvel?
  - Na maioria deles, a lotação é de no máximo 5 pessoas, incluindo seu condutor. Mas uma pessoa pode estar em quantos automóveis ao mesmo tempo? Até onde as leis da física permitem, só em um, de cada vez!
    - Chamamos esse relacionamento de *um-para-muitos*, um (automóvel) para muitos (pessoas).
    - Em outras situações, vemos que um marido é para uma, e somente uma, esposa, e a recíproca é verdadeira, temos então um relacionamento *um-para-um*.
    - A outra combinação possível seria *muitos-para-muitos*, no exemplo computador e impressora, o computador pode imprimir em várias impressoras e cada uma dessas podem receber documentos dos outros computadores do escritório.

Troca de Mensagens

- Vamos ligar a classe Carro à classe Pessoa, fazendo com que uma pessoa possua um carro e esse carro só possa pertencer a uma única pessoa.
  - É o que chamados de relação *um-para-um*.
- A classe Carro precisa "saber que pertence" a alguém.
  - Iremos adicionar um atributo chamado dono, que é do tipo Pessoa.

```
public class Carro {

    String tipo;
    String cor;
    String placa;
    int numPortas;
    Pessoa dono;
```

Troca de Mensagens

- E os respectivos métodos get e set:

```
public void setDono(Pessoa dono) {
    this.dono = dono;
}

public Pessoa getDono() {
    return dono;
}
```

Troca de Mensagens

- Vamos utilizar esse relacionamento no exemplo seguinte:

```
package poo;

public class POO {

    public static void main(String[] args) {
        Carro carro = new Carro();
        carro.setCor("PRETO");
        carro.setTipo("PASSEIO");
        carro.setPlaca("GGG-1111");
        carro.setNumPortas(4);

        Pessoa pessoa = new Pessoa();
        pessoa.setNome("LARA");
        pessoa.setCorDoCabelo("LOIRO");
        pessoa.setBiotipo("MAGRA");
        pessoa.setIdade(30);

        carro.setDono(pessoa);

        System.out.println("-----CARRO-----");
        System.out.println("Cor: " + carro.getCor());
        System.out.println("Tipo: " + carro.getTipo());
        System.out.println("Placa: " + carro.getPlaca());
        System.out.println("Portas: " + carro.getNumPortas());
        System.out.println("Dono: " + carro.getDono().getNome());
    }
}
```

```
run:
-----CARRO-----
Cor: PRETO
Tipo: PASSEIO
Placa: GGG-1111
Portas: 4
Dono: LARA
CONSTRUÍDO COM SUCESSO
```

Trabalho

- Faça um sistema de Biblioteca
  - Livros
    - Atributos: Título, Autores, Área, Editora, Ano, Edição, Número de Folhas.
    - Métodos: set's e get's
  - Usuários:
    - Atributos: Nome, Idade, Sexo, Telefone.
    - Métodos: set's e get's
  - Empréstimos:
    - Atributos: Data do Empréstimo, Hora do Empréstimo, Livro(Relacionamento) e Usuário (relacionamento)
  - Insira os objetos do programa principal e imprima o empréstimo.
  - Não se esquece do construtor.

Troca de Mensagens

- Os sistemas OO precisam de métodos mais complexos, que verifiquem a validade dos dados, realizem atualizações em outras entidades, pesquisem em banco de dados, escrevam em arquivos de log, atualizem objetos da interface etc.

1. Colocando um pouco de complexidade do mundo real no programa!!!

    1. Adicionar os métodos ligar(), desligar(), acelerar(), frear() à classe Carro.

Troca de Mensagens

```
void ligar(){
    System.out.println("CARRO LIGADO.");
}

void desligar(){
    System.out.println("CARRO DESLIGADO.");
}

void acelerar(){
    System.out.println("CARRO ACELERANDO.");
}

void frear(){
    System.out.println("CARRO FREANDO.");
}
```

Troca de Mensagens

2. Adicione à classe Carro, ainda, um atributo câmbio que indicará qual a "marcha" que o carro está em um dado momento, para tanto, tal atributo irá guardar um valor inteiro de 0 (zero) a 5 (cinco).
   - 0 – neutro (ponto morto)
   - 1 a 5 – marchas

```
int cambio;

public void setCambio(int marcha) {
    this.cambio = marcha;
}

public int getCambio() {
    System.out.println("MARCHA " + this.cambio);
    return this.cambio;

}
```

Troca de Mensagens

3. Colocar uma referência do automóvel dentro da classe Pessoa.

```
Carro carro;

public void setCarro(Carro carro) {
    this.carro = carro;
}

public Carro getCarro() {
    return carro;
}
```

Troca de Mensagens

4. Adicione outros métodos à classe Pessoa para interagir (trocar mensagens) com seu carro.

```
public Carro getCarro() {
    return carro;
}

void ligarCarro(){
    carro.ligar();
}

void desligarCarro(){
    carro.desligar();
}

void acelerarCarro(){
    carro.acelerar();
}

void frearCarro(){
    carro.frear();
}

void setCambioMarcha(int marcha){
    carro.setCambio(marcha);
}
```

Troca de Mensagens

5) Vamos ao Main! Nosso programa precisa:

```
pessoa.setCarro(carro);

pessoa.ligarCarro();
pessoa.setCambioMarcha(1);
pessoa.getCarro().getCambio();
pessoa.acelerarCarro();

System.out.println();

pessoa.setCambioMarcha(2);
pessoa.getCarro().getCambio();
pessoa.acelerarCarro();

System.out.println();

pessoa.setCambioMarcha(3);
pessoa.getCarro().getCambio();
pessoa.acelerarCarro();
```

Troca de Mensagens

```
System.out.println();

pessoa.setCambioMarcha(2);
pessoa.getCarro().getCambio();
pessoa.acelerarCarro();

System.out.println();

pessoa.ligarCarro();
pessoa.setCambioMarcha(1);
pessoa.getCarro().getCambio();
pessoa.acelerarCarro();

System.out.println();

pessoa.ligarCarro();
pessoa.setCambioMarcha(0);
pessoa.frearCarro();
pessoa.desligarCarro();
```

Troca de Mensagens

- O que podemos ver é que o método Main envia uma mensagem para o objeto Pessoa, que, por sua vez, manda ou repassa essa mensagem para seu objeto da classe Carro, que executa a ação final desejada, que consiste em guiá-lo.