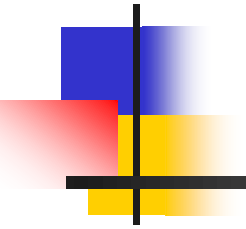


Padrões de Projeto

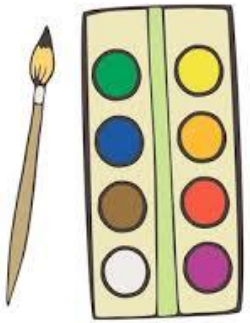
Profª. Rachel Reis

rachel@inf.ufpr.br

Motivação

- Imagine que uma pessoa tenha aprendido diversas técnicas de pintura. Nesse caso, podemos supor que ela aprendeu...

Como segurar o pincel



Como misturar as cores

Como trabalhar com diferentes tipos de tintas

Motivação

- Será que esse conhecimento é suficiente para a pessoa conseguir pintar um quadro?

Motivação

- A pessoa tem todo o conhecimento para realizar a pintura.
- No entanto, esse conhecimento só será válido se a pessoa souber como utilizá-lo.
- Logo, além do conhecimento, é necessário ter habilidade (que só se aprende com muita prática e treino).

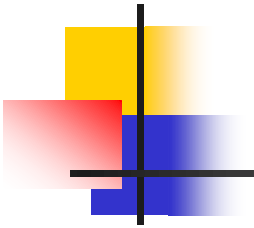
Saber as técnicas é apenas o primeiro passo...

Motivação

- Na programação acontece um fenômeno similar.

Aprender uma linguagem orientada a objetos e seus recursos

≈



Motivação

- Exemplo:
  - Saber como utilizar herança e polimorfismo não é suficiente para diferenciar em quais situações eles devem ser empregados de forma apropriada.
  - Então, o que é necessário?
    - Conhecer os **problemas** que podem aparecer durante a modelagem do sistema.
    - Saber quais **soluções** podem ser implementadas para equilibrar os requisitos.

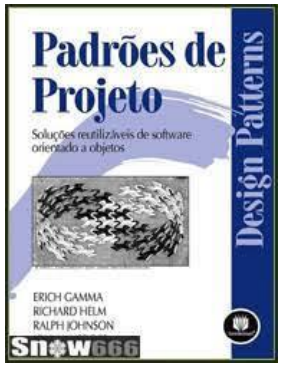
Motivação

- Quais são as alternativas?

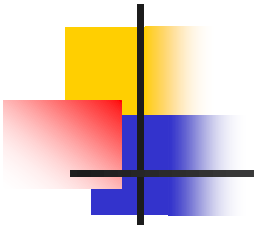
1) Errar várias vezes antes de aprender a forma correta.

2) Utilizar os padrões de projetos (*design patterns*)

Padrões de Projeto

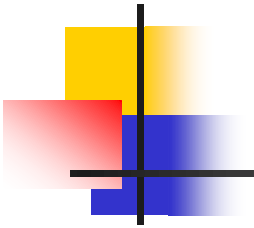


- São soluções elegantes (utilizadas e testadas) para problemas recorrentes e conhecidos no desenvolvimento de *software*.
- São apenas sugestões de código que podem ser aplicadas a diferentes linguagem de programação.
- Foram catalogados e popularizados pelo livro "*Padrões de projeto – Soluções reutilizáveis de software orientado a objetos*" (padrões da GOF de 1994/1995)



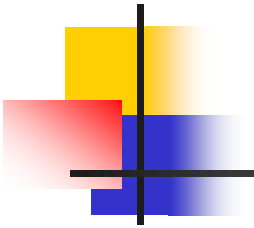
Padrões de Projeto

- Vantagens:
  - Não é preciso reinventar a roda.
  - São padrões universais que facilitam o entendimento do projeto.
  - Evita a refatoração desnecessária do código.
  - Ajuda na reutilização de código.
  - Facilitam na aplicação de testes unitários.



Padrões de Projeto

- Desvantagens:
  - Alguns padrões podem ser complexos até que você os compreenda.
  - Muito código para atingir um objetivo simples.
  - Se usados incorretamente, podem atrapalhar ao invés de ajudar.



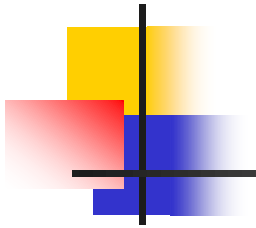
Padrões de Projeto

- Em geral, são classificados em três categorias:
  - Padrões de criação: abstraem o processo de criação de objetos a partir da instanciação de classes.
  - Padrões estruturais: tratam da forma como as classes e objetos estão organizados para a formação de estruturas maiores.
  - Padrões comportamentais: caracterizam como as classes e objetos interagem e distribuem responsabilidades na aplicação.

Padrões de Projeto - Exemplos

- Exemplos:

Criação	Estrutural	Comportamental
• Abstract factory • Builder • Factory Method • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of responsibility • Command • Interpreter • Iterator • Mediator • Memento • Observer • Etc.



Padrões de Projeto

- Principais atributos de um padrão de projeto bem descrito:
  1. Nome: referência que descreve de forma sucinta o padrão.
  2. Problema (motivação, intenção e objetivos, aplicabilidade): apresenta o contexto e quando utilizar o padrão.
  3. Solução (estrutura, participantes, exemplo de código): descreve os elementos que compõem o padrão de projeto, seus relacionamentos e colaborações.
  4. Consequências e padrões relacionados: analisa os resultados, vantagens e desvantagens obtidas com a aplicação do padrão.

Padrões de Projeto

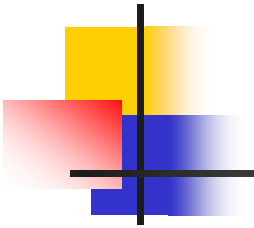
- Lista de atributos usadas pelo livro GOF para a descrição dos padrões de projeto:

• **Nome**	• Colaborações
• **Intenção**	• Consequências
• Motivação	• **Implementação**
• Aplicabilidade	• Exemplo de código
• **Estrutura**	• Usos conhecidos
• Participantes	• Padrões relacionados

Padrão – Factory Method

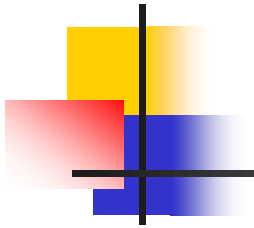
- Exemplos:

Criação	Estrutural	Comportamental
• Abstract factory • Builder • Factory Method • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of responsibility • Command • Interpreter • Iterator • Mediator • Memento • Observer • Etc.



Sobre o Factory Method

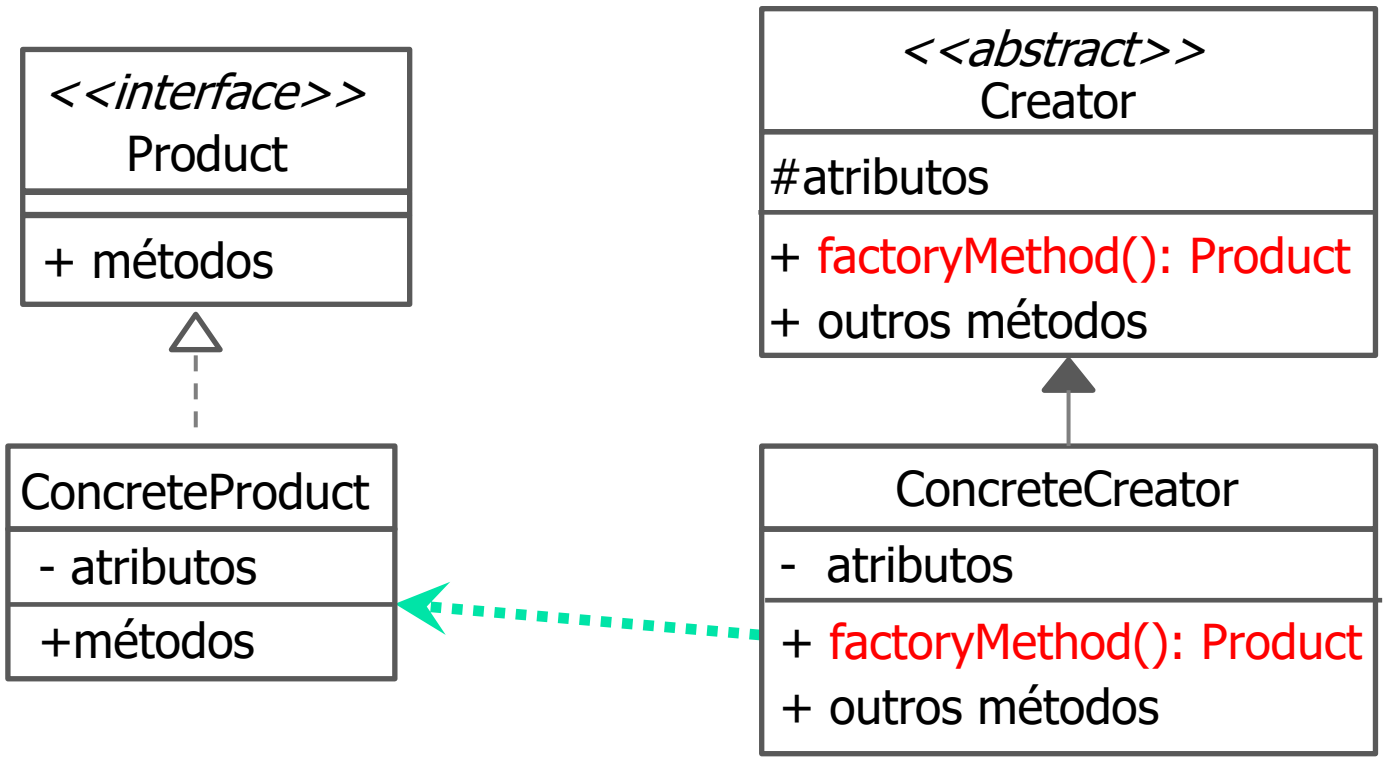
- É um padrão de projeto de criação (lida com a criação de objetos)
- Oculta a lógica de instanciação do código cliente (desacopla o código que cria o objeto do código que utiliza o objeto).
- Utiliza os conceitos de interface, classe abstrata e herança.
- Oferece flexibilidade ao código permitindo a criação de novas *factories* sem a necessidade de alterar o código já escrito.
- Pode usar parâmetros para determinar o tipo dos objetos a serem criados ou receber esses parâmetros para repassar aos objetos que estão sendo criados.



Factory Method - Intenção

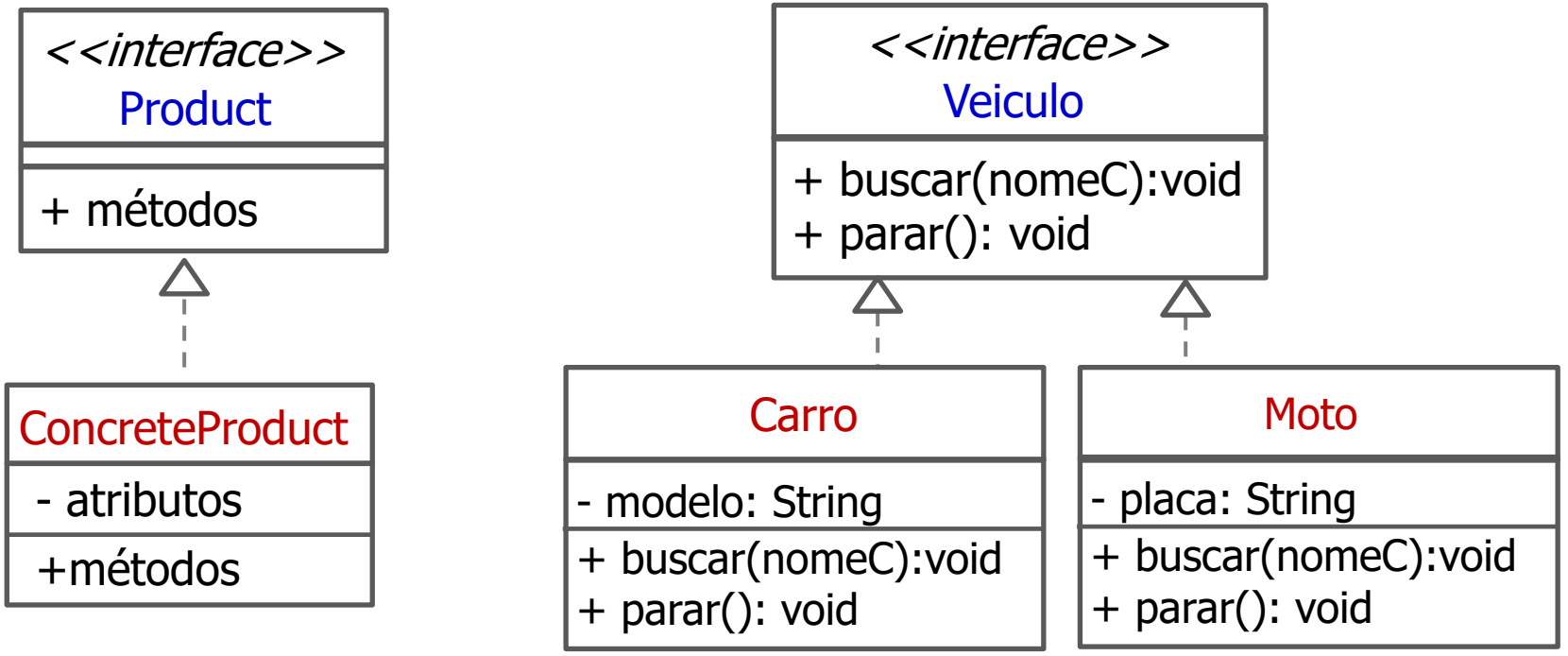
- Definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar.
- Permite a uma classe adiar a instanciação para as subclasses.

Factory Method - Estrutura

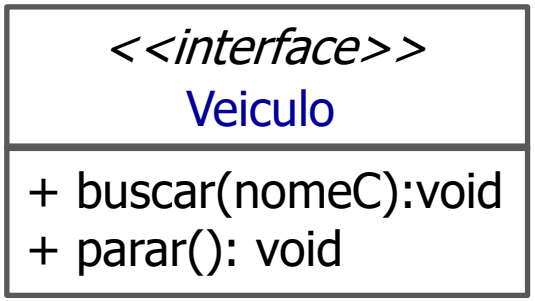


Exemplo: Aplicação Uber

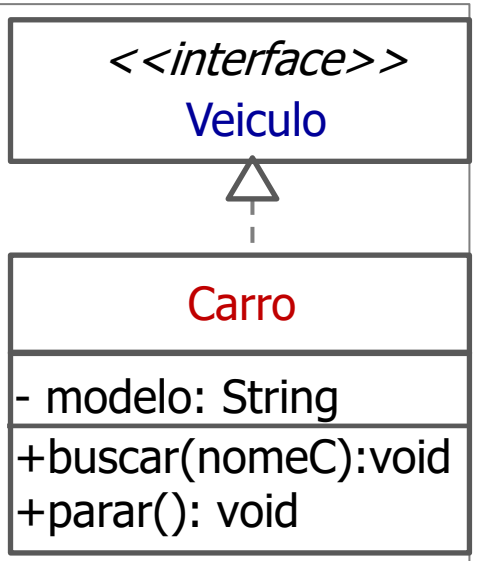
- Empresa que oferece o serviço de transporte de pessoas (similar ao taxi) em carros e motos.



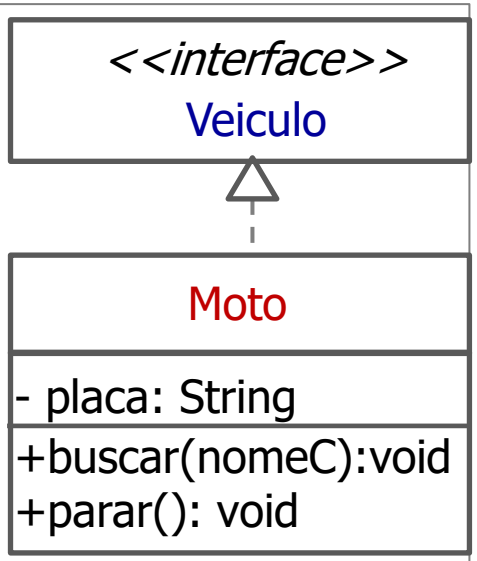
```
public interface Veiculo
{
    public abstract void buscar(String nomeC);
    public abstract void parar();
}
```



```
public class Carro implements Veiculo
{
    private String modelo;

    public Carro(String modelo){
        this.setModelo(modelo);
    }

    // Implementar métodos get/set

    public void buscar(String nomeC){
        System.out.println(this.modelo + " buscando " + nomeC);
    }
    public void parar(){
        System.out.println(this.modelo + " parado ");
    }
}
```



```
public class Moto implements Veiculo
{
    private String placa;

    public Moto(String placa){
        this.setPlaca(placa);
    }

    // Implementar métodos get/set

    public void buscar(String nomeC){
        S.o.p("Moto com a placa "+ this.placa + " buscando " + nomeC);
    }
    public void parar(){
        S.o.p("Moto com a placa "+ this.placa + " parada");
    }
}
```

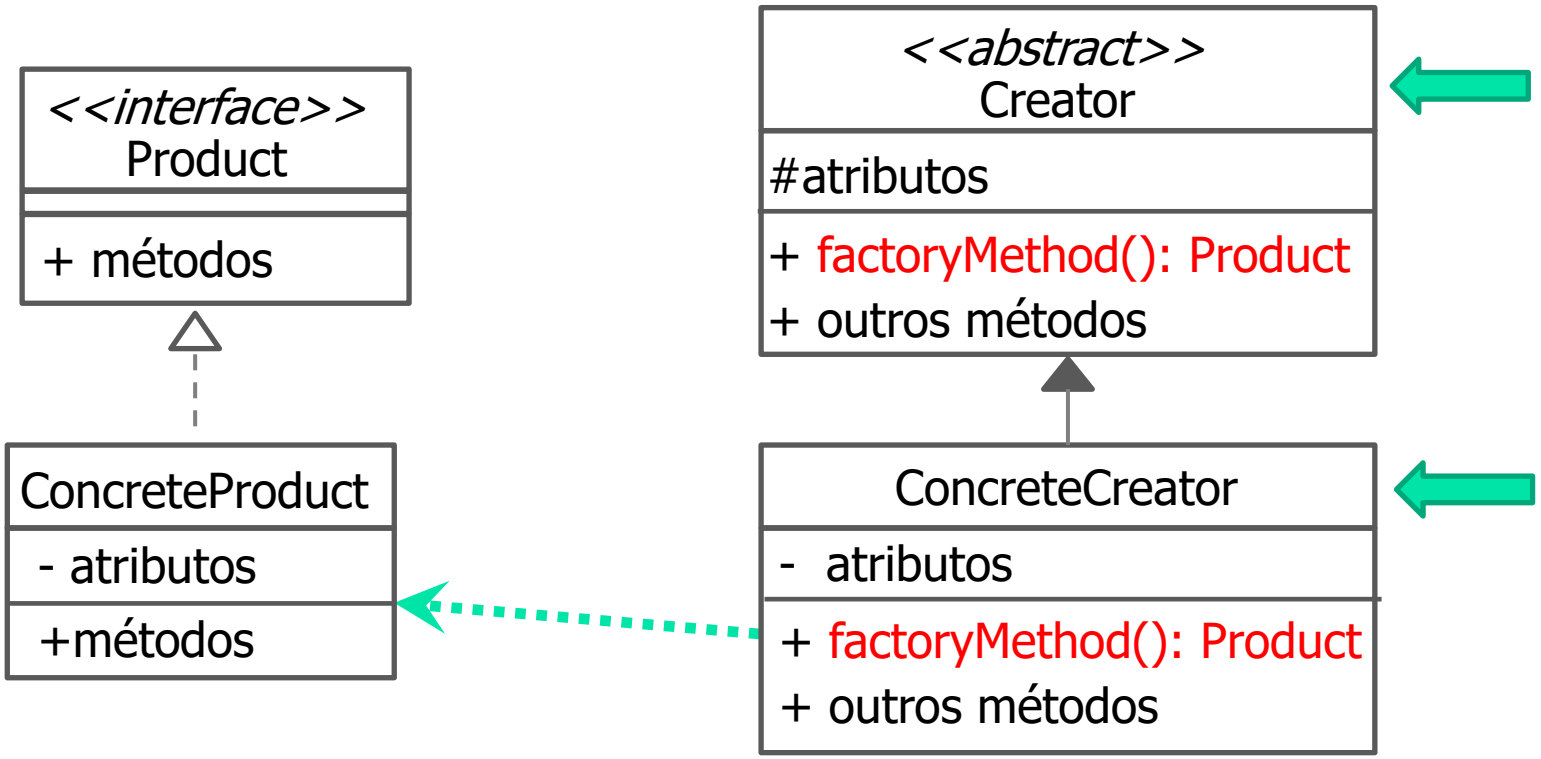


```
public class Principal
{
    public static void main(String []args)
    {
        // Código sem usar o padrão Method Factory
        Veiculo fusca = new Carro(“Fusca”);
        fusca.buscar(“Joana”);
        fusca.parar();

        Veiculo honda = new Moto(“DX23”);
        honda.buscar(“João”);
        honda.parar();

    }
}
```

O que acontece se alterarmos o nome das classes Carro e Moto?

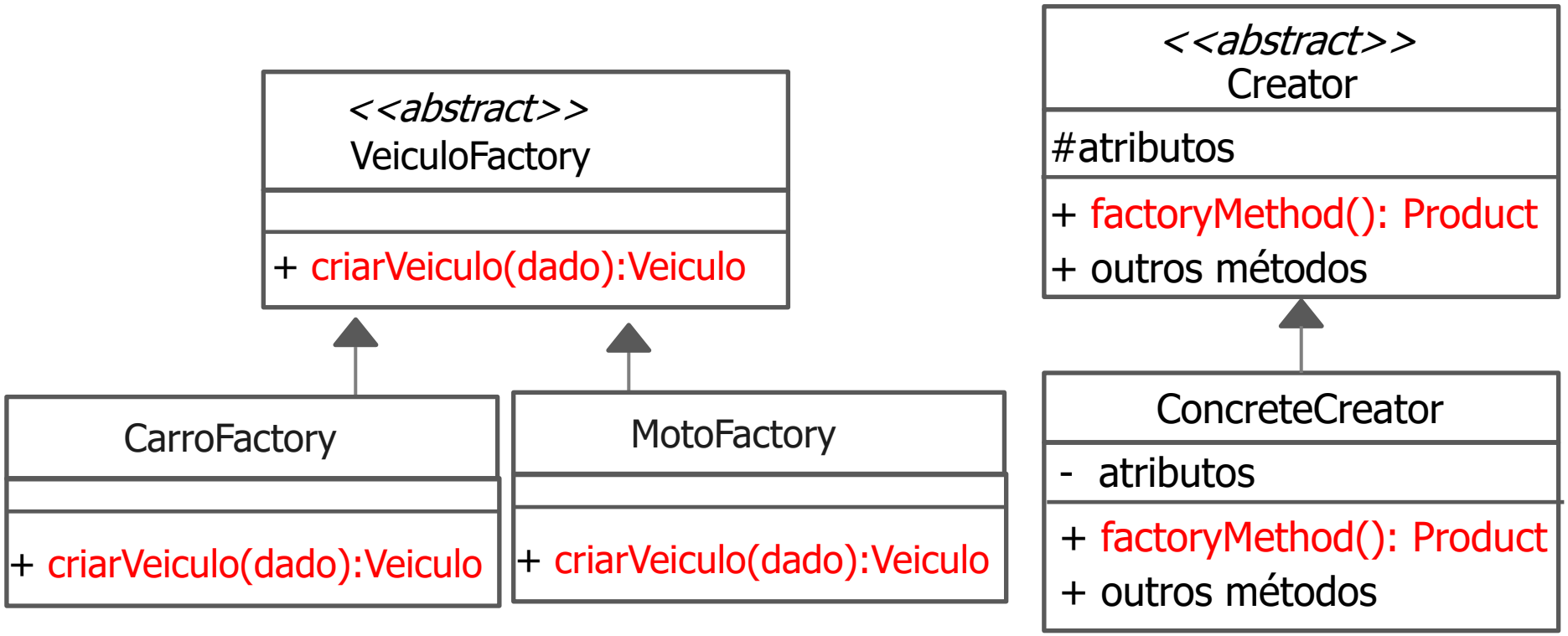
Factory Method - Estrutura

- Solução: aplicar o padrão Method Factory



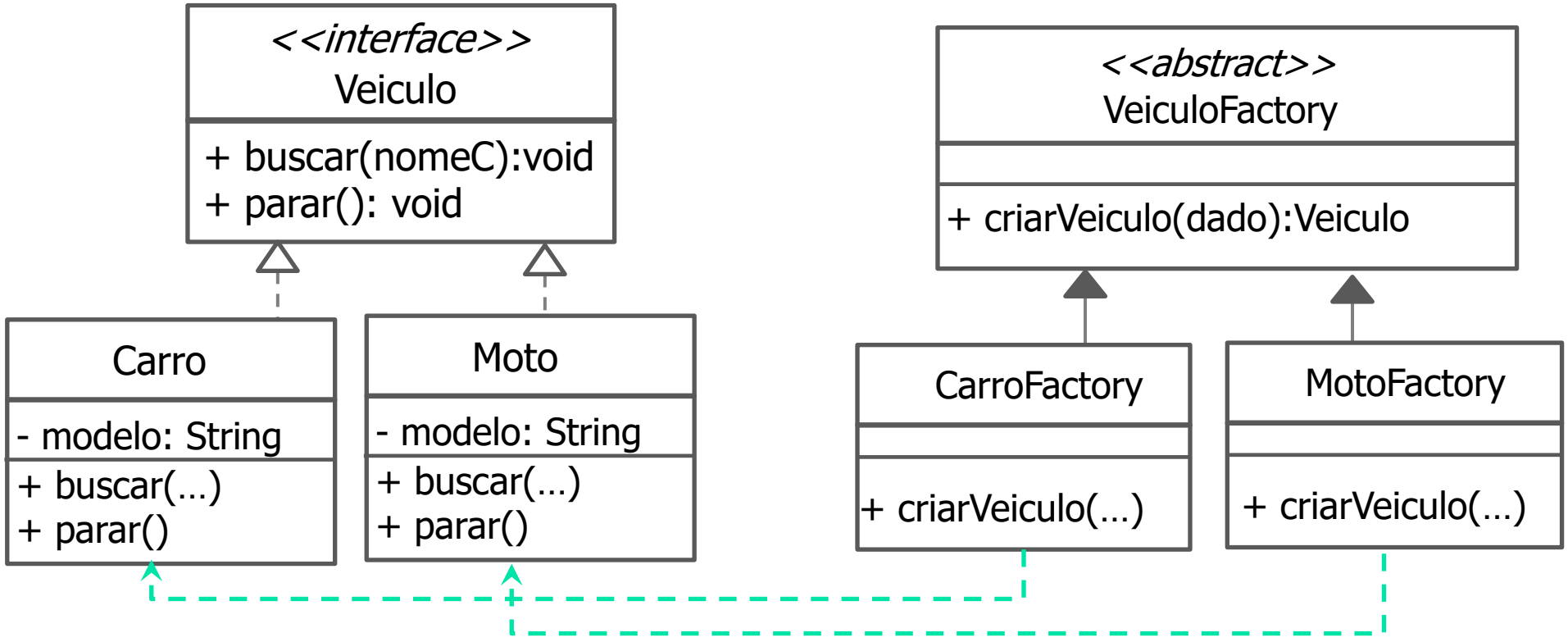
Exemplo: Aplicação - Uber

- Definição do método para criar objetos

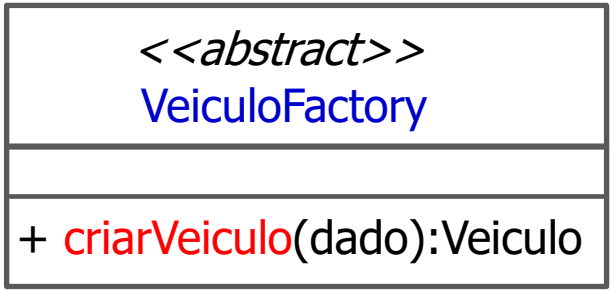


Exemplo: Aplicação - Uber

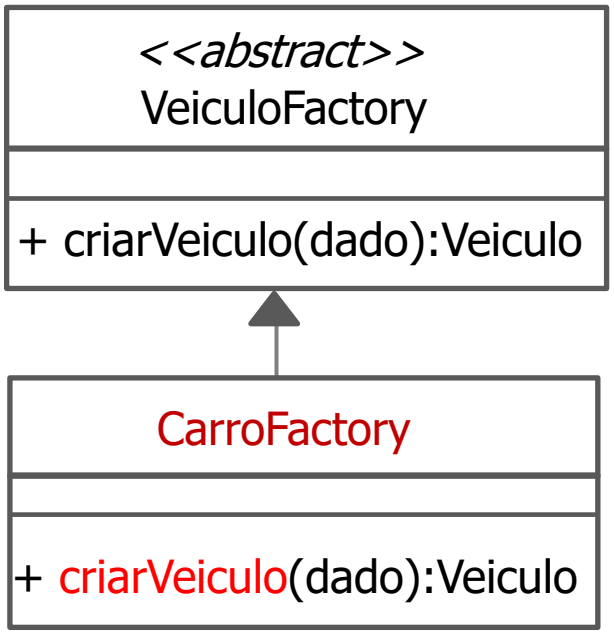
- Estrutura completa do padrão Factory Method.



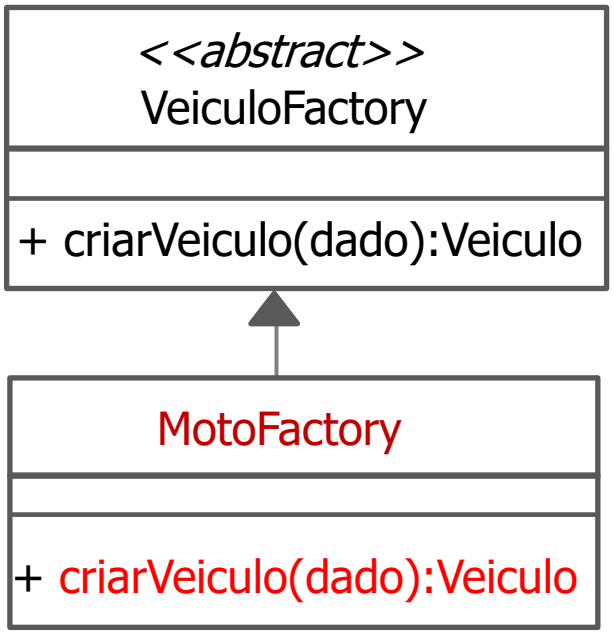
```
public abstract class VeiculoFactory
{
    public abstract Veiculo criarVeiculo (String dado);

}
```



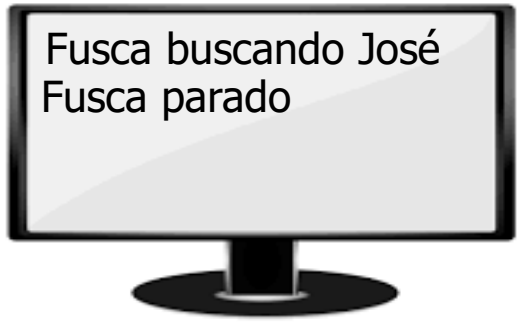
```
public class CarroFactory extends VeiculoFactory
{
    public Veiculo criarVeiculo (String dado){
            return new Carro(dado);
    }
}
```

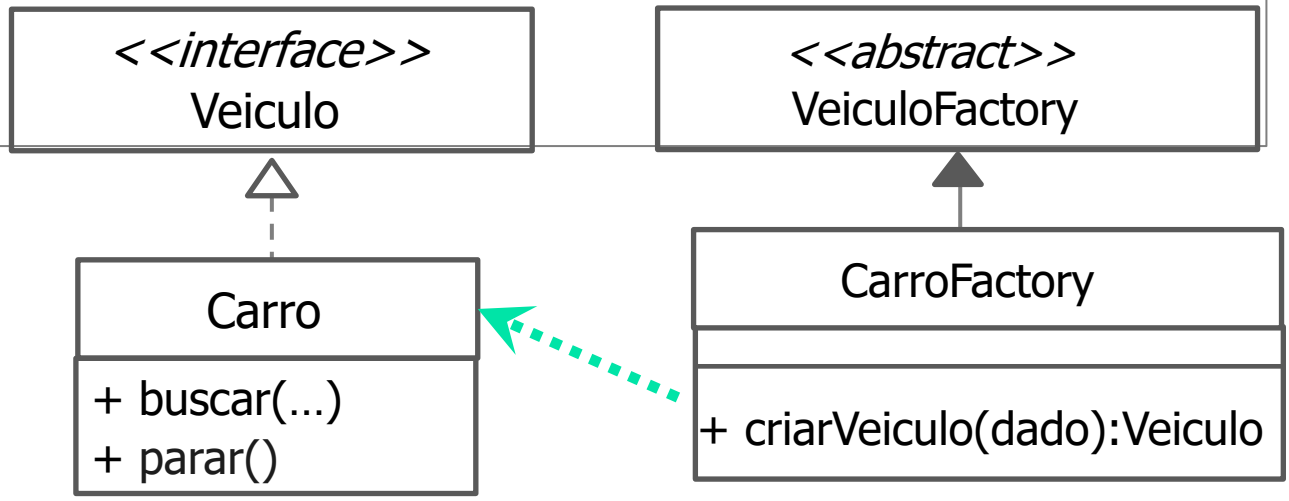


```
public class MotoFactory extends VeiculoFactory
{
    public Veiculo criarVeiculo (String dado){
            return new Moto(dado);
    }
}
```

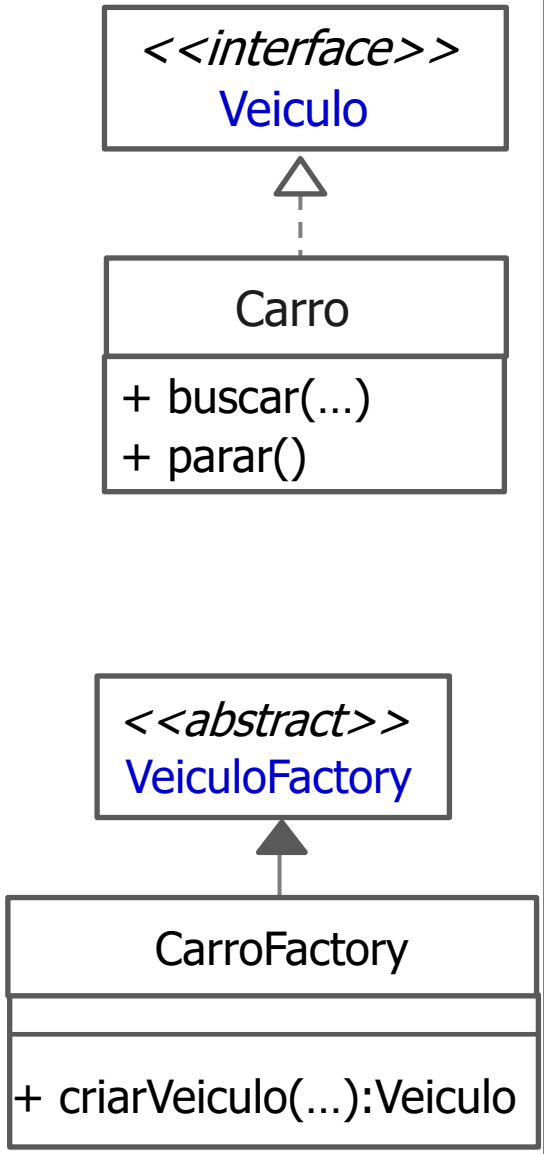


```
public class Principal
{
    public static void main(String []args)
    {
        // Código usando o padrão Method Factory
        VeiculoFactory carroF = new CarroFactory();
        Veiculo fusca = carroF.criarVeiculo("Fusca");
        fusca.buscar("José");
        fusca.parar();
    }
}
```



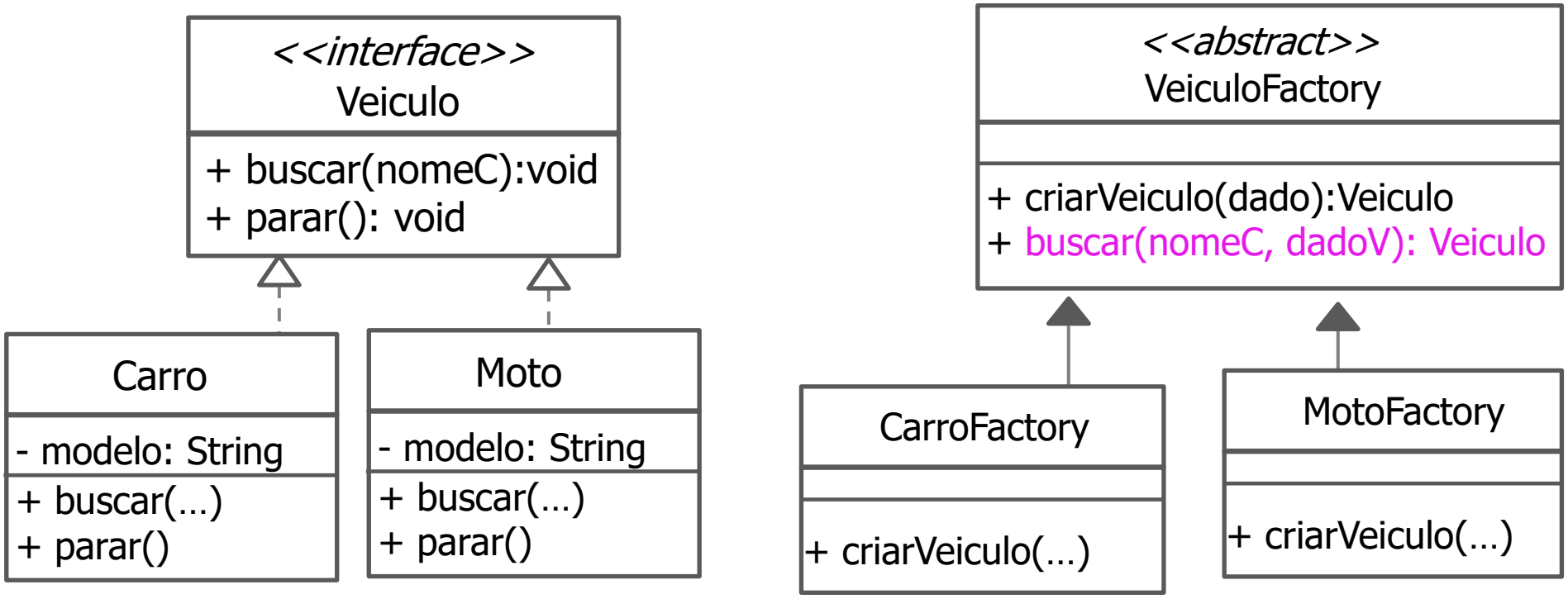


```
public class Principal{
    public static void main(String []args)
    {
       // Código sem usar o padrão Method Factory
        Veiculo fusca = new Carro(“Fusca”);
        fusca.buscar(“Joana”);
        fusca.parar();

       // Código usando o padrão Method Factory
        VeiculoFactory carroF = new CarroFactory();
        Veiculo fusca = carroF.criarVeiculo(“Fusca”);
        fusca.buscar(“José”);
        fusca.parar();

    }
}
```

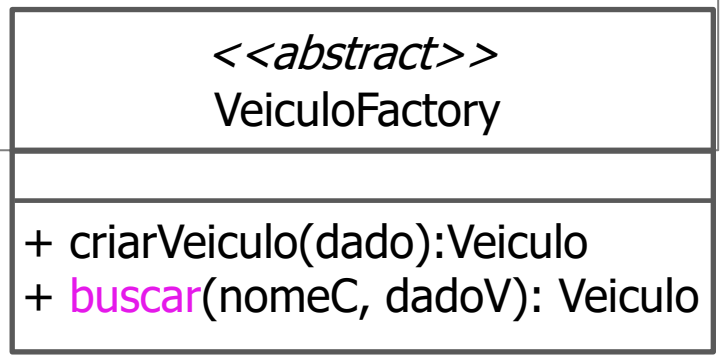


Factory Method - Estrutura

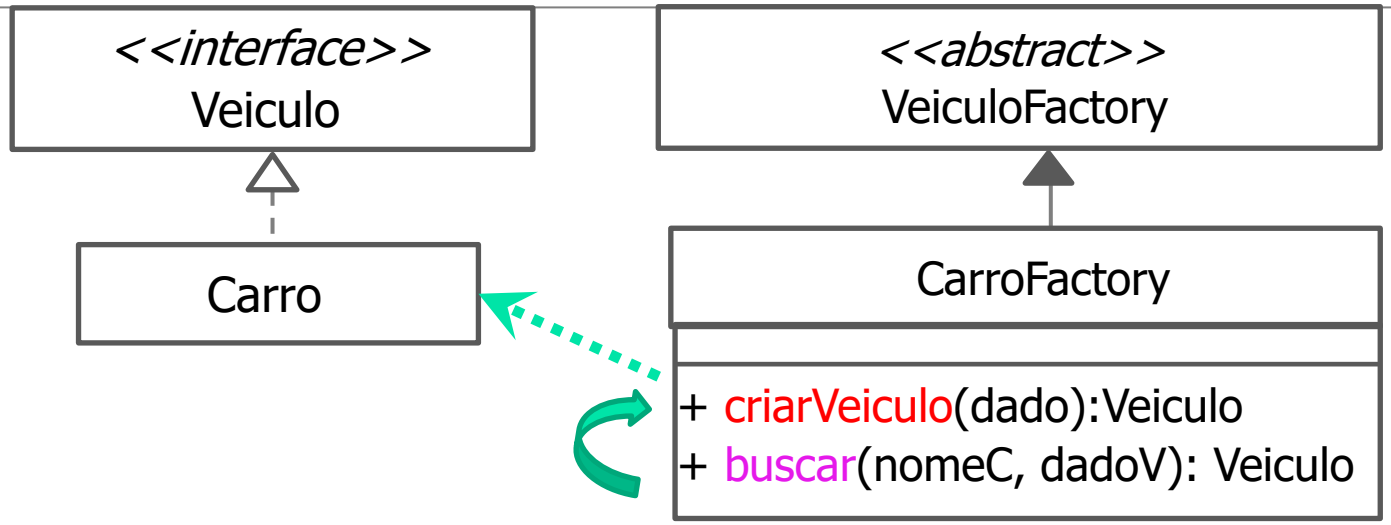
- Podemos ter outros métodos na classe VeiculoFactory



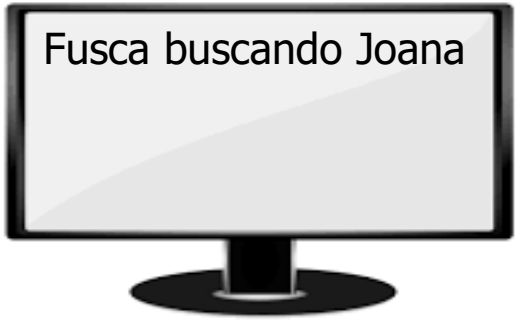
```
public abstract class VeiculoFactory
{
    public abstract Veiculo criarVeiculo (String dado);

    Veiculo buscar(String nomeC, String dadoV)
    {
            Veiculo v = this.criarVeiculo(dadoV);
             v.buscar(nomeC);
             return v;
     }
}
```



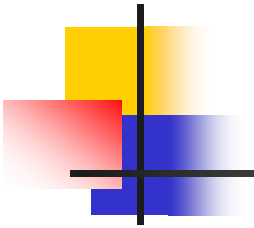
```
public class Principal{
    public static void main(String []args)
     {
         // Código usando o padrão Method Factory
         VeiculoFactory vf1 = new CarroFactory();
         Veiculo c = vf1.buscar(“Joana”, “Fusca”);
      }
}
```



```
public class Principal{
    public static void main(String []args)
    {
        // Código usando o padrão Method Factory
        VeiculoFactory vf1 = new CarroFactory();
        Veiculo c = vf1.buscar("Joana", "Fusca");
    }
}
```



Como fazer para chamar o método parar()?



Para praticar...

- Pense em uma aplicação em que o padrão Factory Method poderia ser usado. Em seguida, elabore a estrutura do padrão.