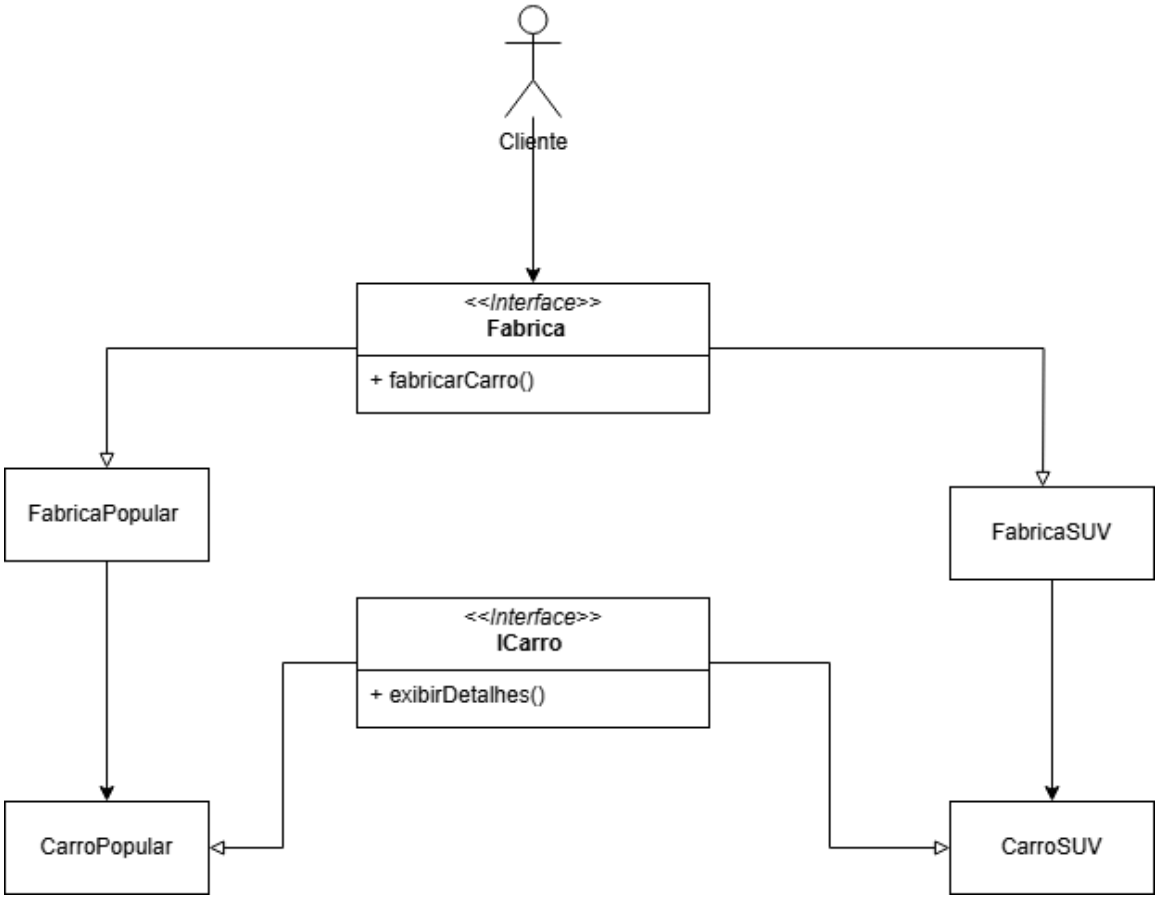
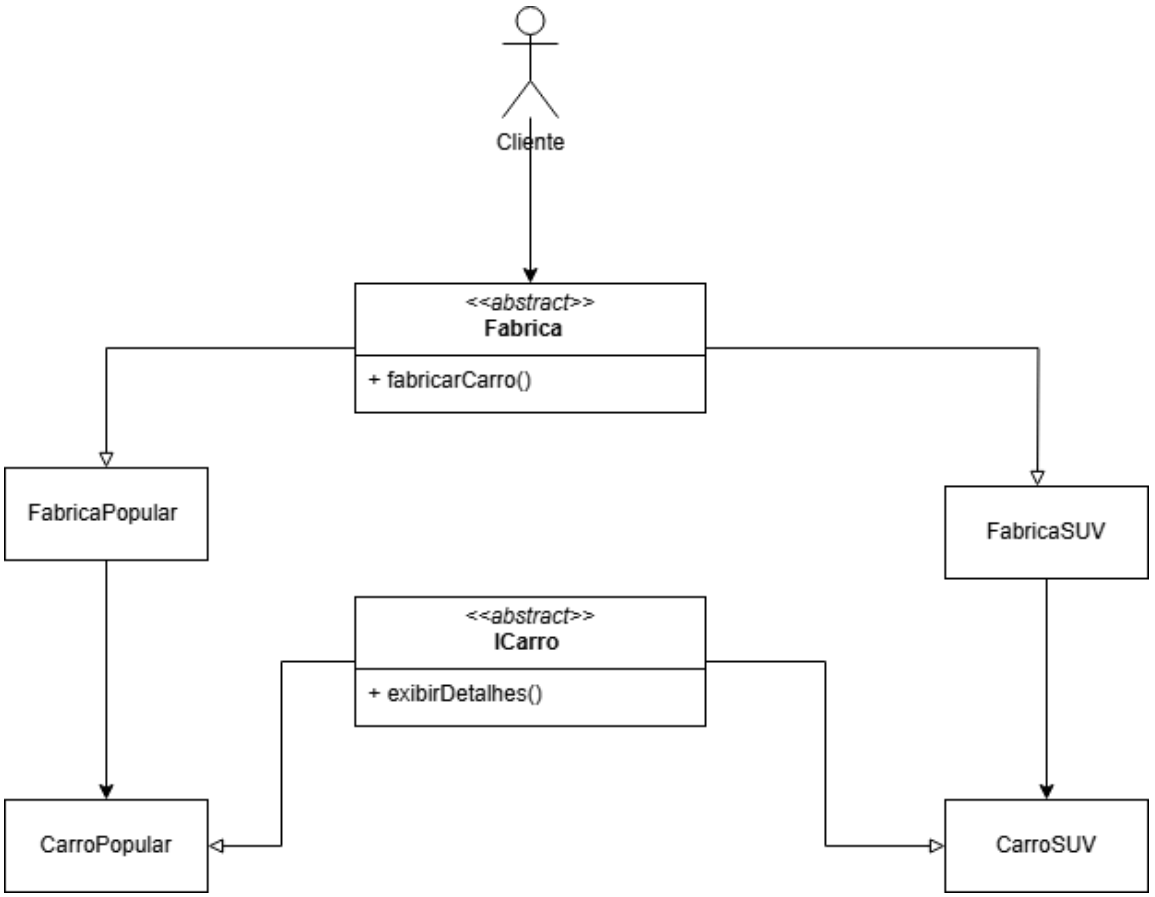
Instituto Federal Catarinense *(Campus Blumenau)*
Professor: Ricardo de La Rocha Ladeira
Matéria: Padrões de Projeto
Nomes: Gabrielli Danker
Turma: BCC 2025.1
Data de entrega: 13 de Março de 2025

Exercícios

1) Considerando as duas versões de códigos para o exemplo das montadoras de veículos, faça o que se pede:
    a) Elabore os diagramas de classes para os dois exemplos.
    INTERFACE:



ABSTRACT:



b) Quem é o *produto*?

RESPOSTA: O produto em ambos os casos são os carros (CarroPopular, CarroSUV), que são criados pelas fábricas (FabricaPopular, FabricaSUV). O produto é representado pela interface ICarro ou pela classe Carro (dependendo da versão do código), com as implementações específicas fornecidas por CarroPopular e CarroSUV.

2) Considerando que a ABC Motors deseja expandir seus segmentos de atuação criando uma fábrica de automóveis sedans, faça o que se pede:

a) Altere os dois códigos para comportar essa nova produção.

INTERFACE:

```
// Interface para os carros
public interface ICarro {
    void exibirDetalhes();
}

// Implementações dos carros
public class CarroPopular implements ICarro {
    @Override
```

```
    public void exibirDetalhes() {
           System.out.println("Carro Popular da ABC Motors:
econômico e compacto.");
    }
}

public class CarroSUV implements ICarro {
    @Override
    public void exibirDetalhes() {
          System.out.println("Carro SUV da ABC Motors: mais
espaço e conforto.");
    }
}

// Novo modelo Sedan
public class CarroSedan implements ICarro {
    @Override
    public void exibirDetalhes() {
            System.out.println("Carro Sedan da ABC Motors:
equilíbrio entre conforto e eficiência.");
    }
}

// Interface da fábrica
public interface IFabrica {
    ICarro criarCarro();
}

// Implementações das fábricas
public class FabricaPopular implements IFabrica {
    @Override
    public ICarro criarCarro() {
        return new CarroPopular();
    }
}

public class FabricaSUV implements IFabrica {
    @Override
    public ICarro criarCarro() {
        return new CarroSUV();
    }
}
```

```java
// Nova fábrica para Sedan
public class FabricaSedan implements IFabrica {
    @Override
    public ICarro criarCarro() {
        return new CarroSedan();
    }
}

// Cliente que usa as fábricas
public class Cliente {
    public static void main(String[] args) {
        IFabrica fabricaPopular = new FabricaPopular();
        ICarro carroPopular = fabricaPopular.criarCarro();
        carroPopular.exibirDetalhes();

        IFabrica fabricaSUV = new FabricaSUV();
        ICarro carroSUV = fabricaSUV.criarCarro();
        carroSUV.exibirDetalhes();

        // Novo carro Sedan
        IFabrica fabricaSedan = new FabricaSedan();
        ICarro carroSedan = fabricaSedan.criarCarro();
        carroSedan.exibirDetalhes();
    }
}
```

ABSTRACT:

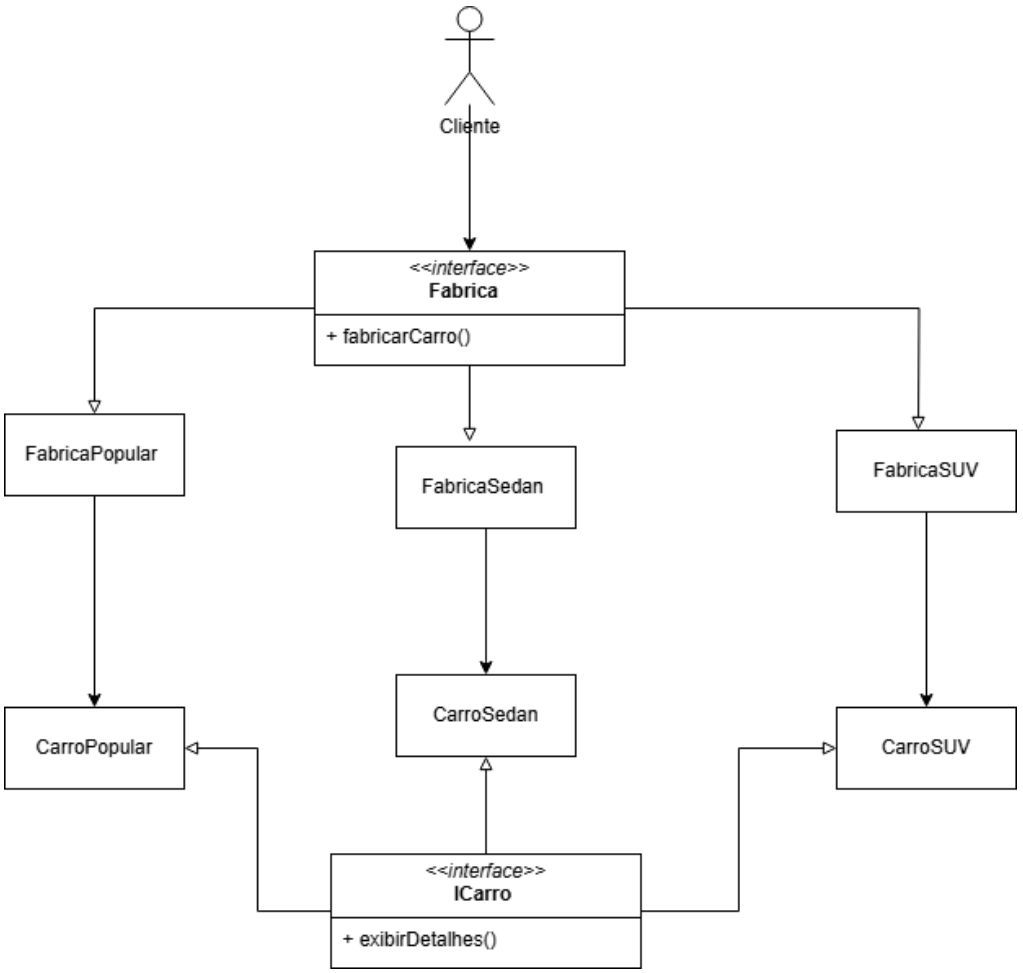
```java
// Classe abstrata para os carros
public abstract class Carro {
    protected String modelo;
    protected int potencia;

    public Carro(String modelo, int potencia) {
        this.modelo = modelo;
        this.potencia = potencia;
    }

    public void exibirDetalhes() {
            System.out.println("Modelo: " + modelo + ".\nPotência: " +
potencia + " CV.");
    }
```

```
}

// Implementações dos carros
public class CarroPopular extends Carro {
    public CarroPopular() {
        super("Carro Popular", 100);
    }
}

public class CarroSUV extends Carro {
    public CarroSUV() {
        super("Carro SUV", 200);
    }
}

// Novo modelo Sedan
public class CarroSedan extends Carro {
    public CarroSedan() {
        super("Carro Sedan", 150);
    }
}

// Classe abstrata para a fábrica
public abstract class Fabrica {
    public Carro fabricarCarro() {
        Carro carro = criarCarro();
        System.out.println("Carro fabricado com sucesso!");
        return carro;
    }

    protected abstract Carro criarCarro();
}

// Implementações das fábricas
public class FabricaPopular extends Fabrica {
    @Override
    protected Carro criarCarro() {
        return new CarroPopular();
    }
}

public class FabricaSUV extends Fabrica {
    @Override
```

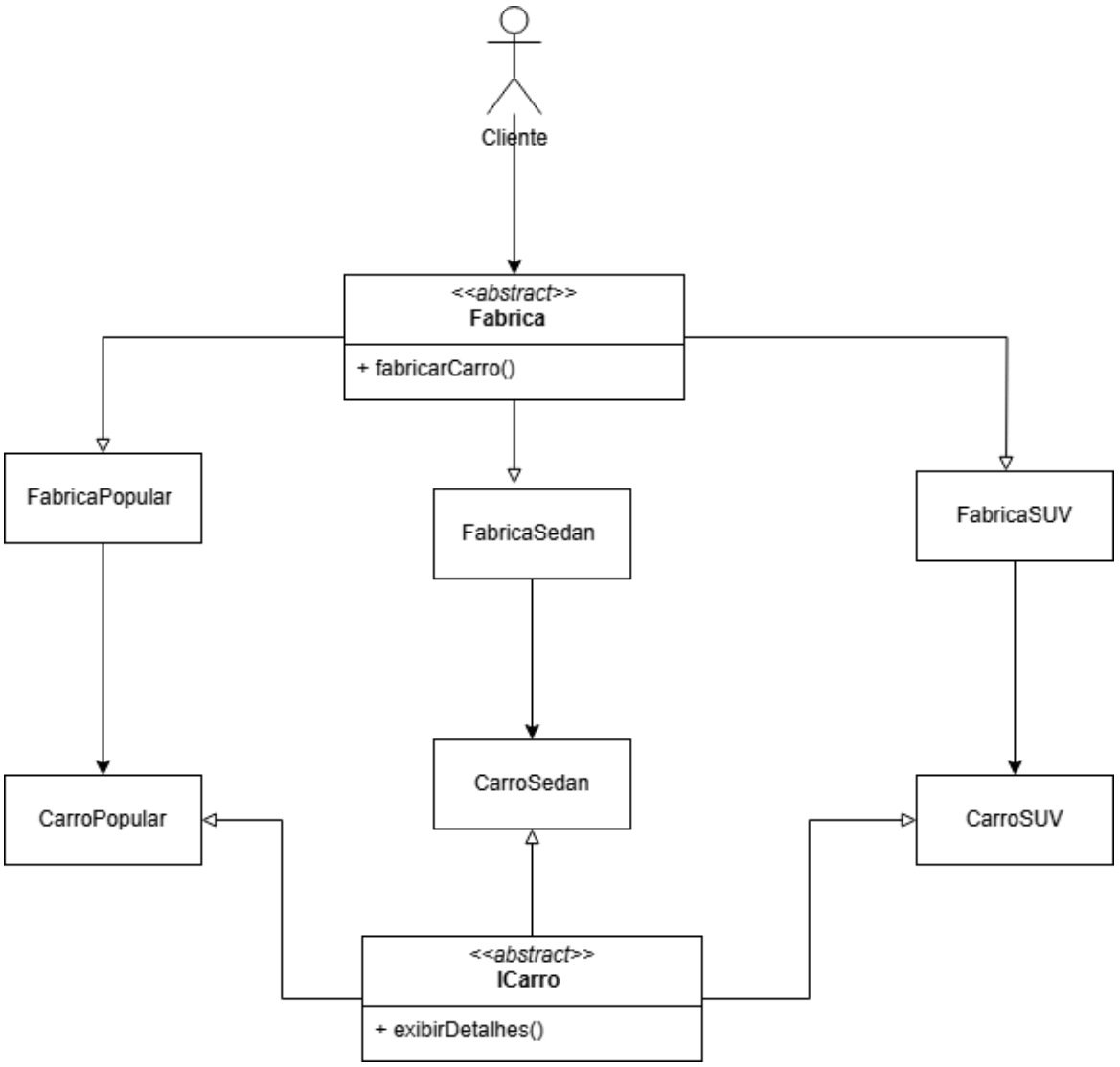
```
    protected Carro criarCarro() {
        return new CarroSUV();
    }
}

// Nova fábrica para Sedan
public class FabricaSedan extends Fabrica {
    @Override
    protected Carro criarCarro() {
        return new CarroSedan();
    }
}

// Cliente que usa as fábricas
public class Cliente {
    public static void main(String[] args) {
        Fabrica fabricaPopular = new FabricaPopular();
        Carro carroPopular = fabricaPopular.fabricarCarro();
        carroPopular.exibirDetalhes();

        Fabrica fabricaSUV = new FabricaSUV();
        Carro carroSUV = fabricaSUV.fabricarCarro();
        carroSUV.exibirDetalhes();

        // Novo carro Sedan
        Fabrica fabricaSedan = new FabricaSedan();
        Carro carroSedan = fabricaSedan.fabricarCarro();
        carroSedan.exibirDetalhes();
    }
}
```

b) Elabore os diagramas de classes para os dois novos códigos.

INTERFACE:



ABSTRACT:



3) Existe uma abordagem denominada *Simple Factory* em que somente uma fábrica é criada, e nela o tipo certo de objeto é criado com base em um parâmetro recebido. Refaça o projeto da montadora de veículos usando o padrão *Simple Factory*.

RESPOSTA:

```
public enum TipoCarro {
    POPULAR, SUV, SEDAN
}
```

```
public interface ICarro {
    void exibirDetalhes();
}
```

```
public class CarroPopular implements ICarro {
    @Override
    public void exibirDetalhes() {
                System.out.println("Carro Popular da ABC Motors:
econômico e compacto.");
    }
}

public class CarroSUV implements ICarro {
    @Override
    public void exibirDetalhes() {
          System.out.println("Carro SUV da ABC Motors: mais espaço
e conforto.");
    }
}

// Novo modelo Sedan
public class CarroSedan implements ICarro {
    @Override
    public void exibirDetalhes() {
         System.out.println("Carro Sedan da ABC Motors: equilíbrio
entre conforto e eficiência.");
    }
}
```

```
public class FabricaCarro {
    public static ICarro criarCarro(TipoCarro tipo) {
        switch (tipo) {
            case POPULAR:
                return new CarroPopular();
            case SUV:
                return new CarroSUV();
            case SEDAN:
                return new CarroSedan();
            default:
                 throw new IllegalArgumentException("Tipo de carro
não reconhecido.");
        }
    }
}
```

```
public class Cliente {
    public static void main(String[] args) {
                                        ICarro     carroPopular     =
FabricaCarro.criarCarro(TipoCarro.POPULAR);
        carroPopular.exibirDetalhes();

        ICarro carroSUV = FabricaCarro.criarCarro(TipoCarro.SUV);
        carroSUV.exibirDetalhes();

                                          ICarro     carroSedan     =
FabricaCarro.criarCarro(TipoCarro.SEDAN);
        carroSedan.exibirDetalhes();
    }
}
```

4) Considerando o projeto da montadora de veículos, qual padrão é mais indicado: *Factory Method* ou *Simple Factory*? Justifique.
   RESPOSTA: O Factory Method é a melhor escolha para a ABC Motors, pois permite a criação de novas fábricas especializadas sem precisar modificar uma única fábrica central.
5) (adaptada de CESPE – 2023 – SERPRO – Analista – Especialização: Tecnologia) No catálogo GoF, a classe *Factory Method* tem, em seu escopo, os padrões *Builder*, *Prototype*, *Composite* e *Iterator*. Certo ou errado?
   RESPOSTA: errado