+Devs2Blu

Linguagem de programação JAVA

Profª. Heloisa Moura

+Devs2Blu

Fundamentos avançados OOP

O que vamos ver:

- Recapitulando os princípios SOLID
- Introdução a Padrões de projetos do GoF (Design Patterns)
- Definição e funcionamento de alguns Padrões

Recapitulando os princípios SOLID

Cinco regras que orientam o design de software, facilitando o trabalho em equipe e evitando problemas ao longo do tempo.

- **Single Responsibility Principle (SRP)** –

Princípio da Responsabilidade Única: cada classe deve ter uma única responsabilidade ou propósito. Em Java, isso significa que uma classe deve realizar apenas uma função específica.

- **Open/Closed Principle (OCP)** –

Princípio do Aberto/Fechado: uma classe deve estar "aberta para extensão, mas fechada para modificação". Em Java, você pode usar interfaces ou herança para adicionar novas funcionalidades, evitando modificar o código existente. Isso reduz o risco de erros ao adaptar funcionalidades.

Recapitulando os princípios SOLID

- **Liskov Substitution Principle (LSP) –**

Princípio da Substituição de Liskov: se uma classe filha herda de uma classe pai, ela deve ser capaz de substituir a classe pai sem problemas. Isso garante que, em Java, subclasses possam ser usadas no lugar de suas superclasses sem alterar o comportamento esperado.

- **Interface Segregation Principle (ISP) –**

Princípio da Segregação de Interface: classes não devem ser forçadas a implementar métodos que não usam. Em Java, isso se traduz em dividir interfaces grandes em menores, para que cada classe implemente apenas o que realmente precisa.

Recapitulando os princípios SOLID

- **Dependency Inversion Principle (DIP) -** Princípio da Inversão de Dependência: módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações. Em Java, você pode usar interfaces para evitar que uma classe dependa diretamente de outra. Isso torna o código mais flexível e fácil de testar.

Padrões de projetos (Design Patterns)

Padrões de projetos (Design Patterns)

O conceito dos padrões foi consolidado pelo GoF (*Gang of four)* se refere à um grupo de quatro grandes nomes no desenvolvimento (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides) que catalogaram 23 padrões de projeto em um livro de referência.

Classificação dos Padrões de projeto

4 Categorias:

- **Padrões de Criação**: Ajudam a instanciar objetos.
- **Padrões Estruturais**: Focam em organizar classes e objetos.
- **Padrões Comportamentais**: Definem interações entre objetos.
- **Padrões Concorrentes**: Lidam com operações em ambientes multi-thread.

Classificação dos Padrões de projeto

- **Padrões Criacionais:** Trazendo formas para criação de objetos e forma independente, simples e desacoplada. Visando facilitar e até mesmo reduzir futuras manutenções. Os principais padrões de projeto: **Factory Method, Abstract Factory, Singleton, Builder e Prototype**.

- **Padrões Estruturais:** Esta categoria apresenta como é possível estruturar diversos objetos e classes de forma extensível e flexível. Os principais padrões são: **Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Mediator e Proxy.**

Classificação dos Padrões de projeto

- **Padrões Comportamentais:** Trata em como as responsabilidades são propagadas entre as classes e objetos. Os principais padrões: **Chain of Responsability, Command, Iterator, Memento, Observer, State, Strategy e Template Method.**

Padrões de projetos (Design Patterns)

Padrões Criacionais

Factory Method, Abstract Factory.

Padrões de projetos Factory Method

Problema

- Como posso escrever um código onde as classes instanciadas possam variar dentro de uma mesma interface?
- Como deixar o meu código desacoplado das classes concretas?

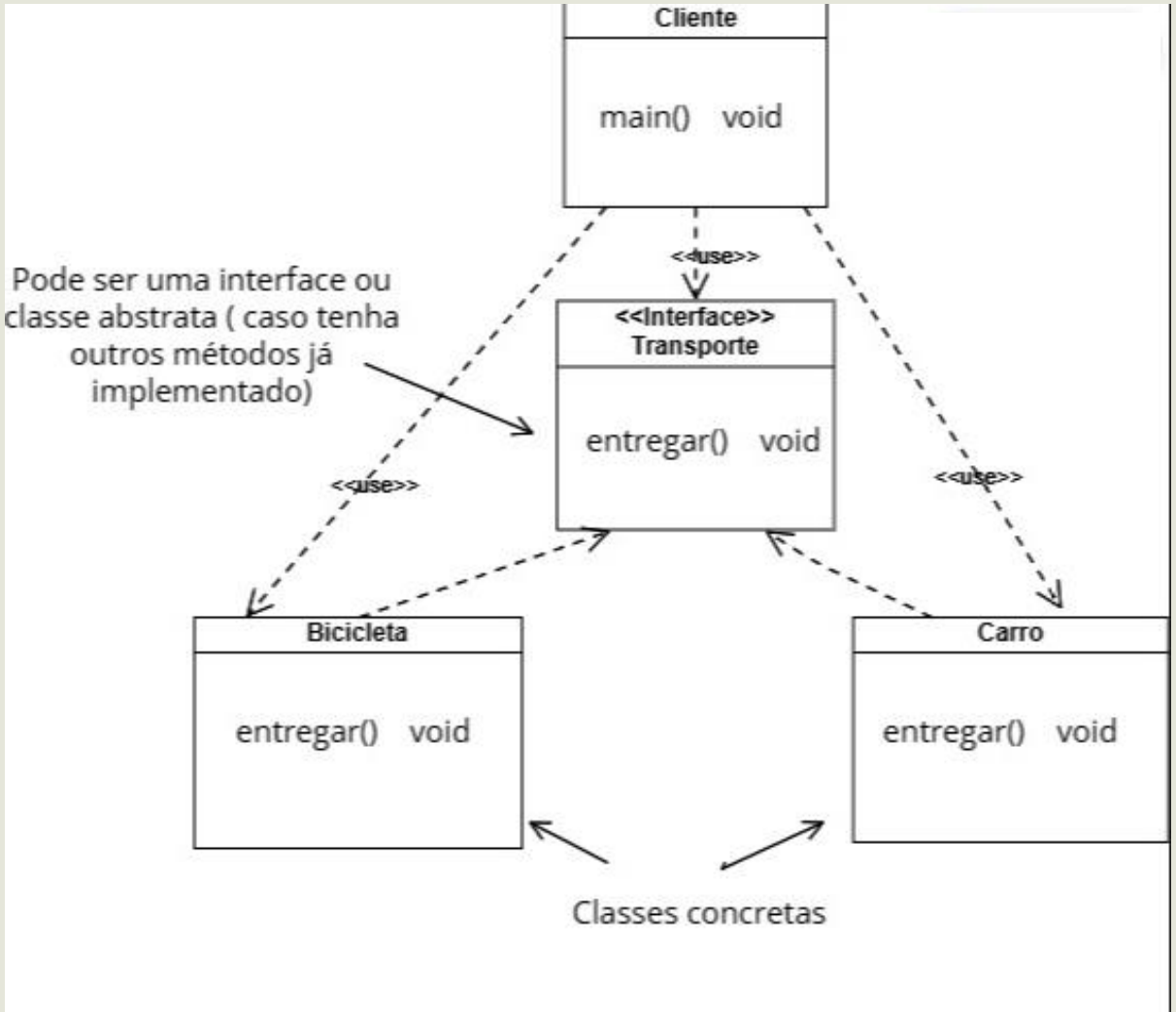
Padrões de projetos Factory Method

Exemplo de um cenário sem usar o padrão:

Um sistema de entregas que usa diferentes tipos de transporte (Bicicleta, Carro, etc).

Como mostra o diagrama de classes

Diagrama de classes



Solução Padrões de projetos: Factory Method

- Extrair a lógica de criação dos objetos para um factory method;
- Invocar o Factory method para receber uma instância qualquer que implemente uma determinada interface.

Solução Padrões de projetos: Factory Method

Definição: Define uma interface ou classe abstrata para criar objetos, mas permite que as subclasses decidam qual classe instanciar.

- É um dos padrões de criação que se concentra na criação de objetos. Em vez de instanciar objetos diretamente com o operador **new**, ele sugere a criação de um método fábrica que decide qual objeto será criado.

- Esse padrão permite a subclasses definir a classe dos objetos que serão criados.

Estrutura do Padrão Factory Method em Java

- **Product (Produto):** Define a interface para objetos que o método fábrica cria.
- **ConcreteProduct (Produto concreto):** Implementação específica da interface **Product**.
- **Creator (Criador):** Declara o método fábrica, que retorna um objeto do tipo **Product**.
- **ConcreteCreator (Criador concreto):** Implementa o método fábrica para retornar uma instância de **ConcreteProduct**

Padrões de projetos Factory Method

Exemplo Prático: Implementação de um sistema de entregas onde o Factory Method é usado para criar diferentes tipos de transporte (Bicicleta, Carro).

Produto : **Interface** Transporte
método: entregar()

Produto concreto: **Classe concreta** Bicicleta
implementa: Transporte
método: entregar()

Produto concreto: **Classe concreta** Carro
implementa: Transporte
método: entregar()

+Devs2Blu

Fundamentos avançados OOP

Padrões de projetos Factory Method

Criador: **Classe abstrata:** TransporteFactory
método abstrato: criarTransporte()
retorna Transporte

Criador concreto: **Classe concreta:** BicicletaFactory
extends: TransporteFactory
sobrescreve método: criarTransporte()
retorna: new Bicicleta()

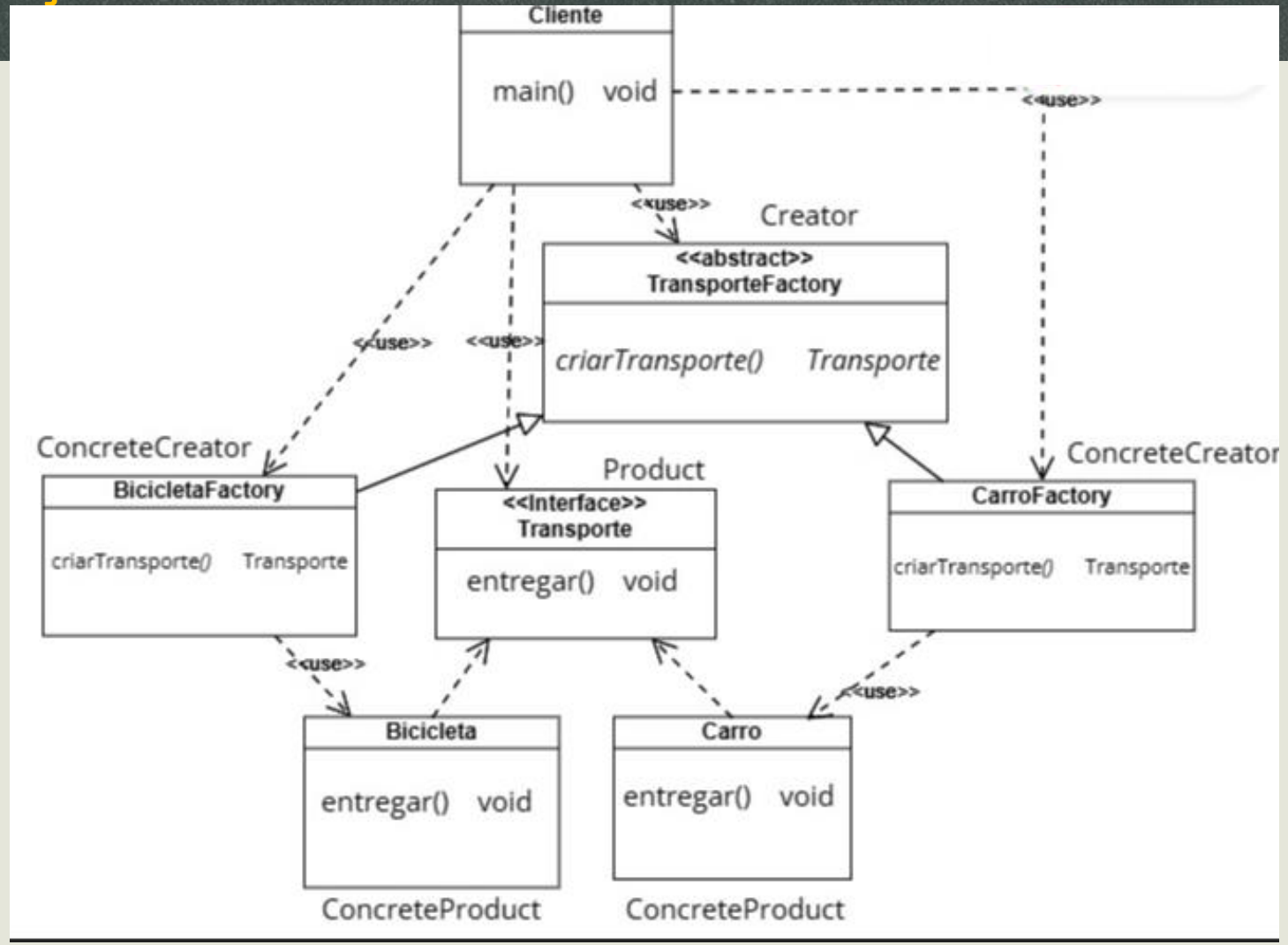
Criador concreto: **Classe concreta:** CarroFactory
extends: TransporteFactory
sobrescreve método: criarTransporte()
retorna: new Carro()

Visão do uso do Factory Method na classe Cliente.

+Devs2Blu

Fundamentos avançados OOP

Diagrama de classes



+Devs2Blu

Fundamentos avançados OOP

A implementação em Java

```java
//Produto
public interface Transporte {
 void entregar();
}
```

```java
//Produto Concreto
public class Carro implements Transporte {
 public void entregar() {
     System.out.println("Entrega feita por carro!");
 }
}
```

```java
//Produto Concreto
public class Bicicleta implements Transporte {
    public void entregar() {
     System.out.println("Entrega feita por bicicleta!");
    }
}
```

```java
//Criador
public abstract class TransporteFactory {

 public abstract Transporte criarTransporte();

}
```

```java
//Criador Concreto
public class CarroFactory extends TransporteFactory {

 public Transporte criarTransporte() {
     return new Carro();
 }
}
```

```java
//Criador Concreto
public class BicicletaFactory extends TransporteFactory {

 public Transporte criarTransporte() {
     return new Bicicleta();
 }
}
```

+Devs2Blu

Fundamentos avançados OOP

A implementação em Java

```java
//Uso do Factory Method
public class  Cliente {

 public static void main(String[] args) {
      TransporteFactory factory = new CarroFactory();
      Transporte transporte = factory.criarTransporte();
      transporte.entregar();  // Saída: "Entrega feita por carro!"

      factory = new BicicletaFactory();
      transporte = factory.criarTransporte();
      transporte.entregar();  // Saída: "Entrega feita por bicicleta!"
  }
}
```

Padrões de projetos Factory Method

Resumo do exemplo implementado:

- Transporte é a interface do produto.
- Carro e Bicicleta são implementações concretas de Transporte.
- TransporteFactory é a classe criadora que define o método criarTransporte().
- CarroFactory e BicicletaFactory são criadores concretos que retornam uma instância específica de Transporte.

Padrões de projetos Factory Method

Exercícios