AULA 8 – PADRÃO *FACTORY METHOD*

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br

UFU

Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender o funcionamento do padrão de projeto *Factory Method* (Método-Fábrica).

Factory Method

- É um padrão de projeto de propósito **de criação** com escopo de **classes**.
- Propõe uma solução para a criação de objetos da mesma família.

Intenção

- Definir uma interface para criar um objeto, mas deixar as subclasses decidirem qual classe instanciar.
- Basicamente, a lógica criacional é encapsulada dentro do *factory* e é fornecido um método que retorna um novo objeto criado.
- Enquanto o Método-template é utilizado para criar algoritmos, o Método-fábrica é utilizado para criar objetos
  - *Ou seja, a responsabilidade de criação é delegada para as subclasses*

Quando usar

- Uma subclasse não pode antecipar (ainda não foi definida) a classe de objetos que deve ser criada.
  - *A escolha de qual classe instanciar é tomada em tempo de execução.*
- Uma classe quer que suas subclasses especifiquem os objetos que elas criam.
- Deseja-se criar um objeto sem especificar qual a classe vai ser utilizada (inversão de dependência).

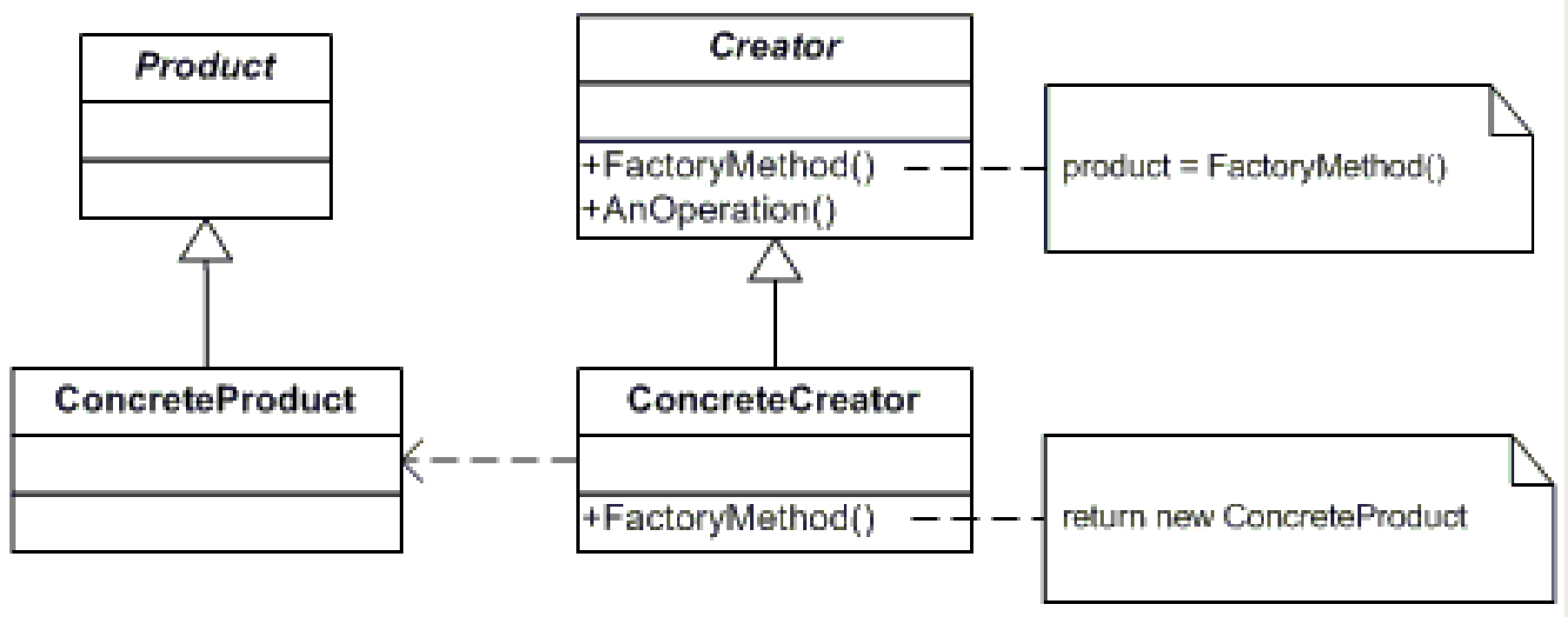
Elementos participantes

- **Product**, define a interface de objetos que o método-fábrica cria.
- **ConcreteProduct**, implementa a interface Product (abstração de um produto concreto).
- **Creator**, declara o método-fábrica, o qual retorna um objeto do tipo Product.
- **ConcreteCreator**, sobrescreve o método-fábrica para retornar uma instância de um ConcreteProduct.

Estrutura

Duas principais variações do padrão Factory Method:

- O Creator é uma classe abstrata e não fornece uma implementação para método de fabricação (factory method) que ela declara
- O Creator é uma classe concreta e fornece uma implementação por omissão (padrão) para o método de fabricação (factory method)



Benefícios

- Fornece um gancho para as subclasses (especialização). A criação de classes e subclasses se torna mais flexível.
- Encapsula o código que cria objetos.
- Classes mais desacopladas.
  - *Elimina dependência de classes específicas, ou seja, depende de abstrações e não de detalhes de implementação.*

Desvantagens

- Muitas classes são criadas.
- Necessidade de criar várias subclasses da classe Creator apenas para criar um ConcreteProduct.
- O código se torna mais difícil de se ler devido às abstrações.

Problema prático

- Imagine duas máquinas de bebidas que distribuem, respectivamente, Refrigerante e Suco.

  1) Crie um método-fábrica chamado de entregarBebida que retorna o tipo concreto de bebida de acordo com cada subclasse.
  - *Classes a serem criadas: Bebida (Product), Refrigerante (ConcreteProduct), Suco (ConcreteProduct), MaquinaDeBebidas (Creator), MaquinaDeRefri (ConcreteCreator) e MaquinaDeSuco (ConcreteCreator)*

Problema prático

2) Crie o método `public String tipoBebida()` na interface Bebida e implemente-o nas subclasses.

3) Crie uma classe Quiosque que contém o método main.

4) Crie duas máquinas diferentes para cada tipo de bebida: maquina1 e maquina2.

5) Crie uma única variável do tipo Bebida e chame o método entregarBebida() de cada uma das máquinas e imprima o tipo de bebida de cada uma.

Problema prático

6) Agora, imagine um novo tipo de bebida é distribuído: Café. O que deve ser feito no código?

Referências

- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995. Capítulo 3.