nappacademy

Napp Academy

Orlando Saraiva Júnior

Padrões de Projeto

Padrões de Projetos

nappacademy

Padrões de projeto são soluções típicas para problemas comuns em projeto de software. Eles são como plantas de obra pré fabricadas que você pode customizar para resolver um problema de projeto recorrente em seu código.

O padrão não é um pedaço de código específico, mas um **conceito geral para resolver um problema em particular**. Você pode seguir os detalhes do padrão e implementar uma solução que se adeque às realidades do seu próprio programa.

Como é um padrão ?

nappacademy

- O **Propósito** do padrão descreve brevemente o problema e a solução.

- A **Motivação** explica a fundo o problema e a solução que o padrão torna possível.

- As **Estruturas** de classes mostram cada parte do padrão e como se relacionam.

Classificação

nappacademy

- Os **padrões criacionais** fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização de código.

- Os **padrões estruturais** explicam como montar objetos e classes em estruturas maiores, enquanto ainda mantém as estruturas flexíveis e eficientes.

- Os **padrões comportamentais** cuidam da comunicação eficiente e da assinalação de responsabilidades entre objetos.

Fábricas

nappacademy

Uma fábrica (*Factory*) é uma classe para a criação de outros objetos. Normalmente, esta classe possui métodos que aceitam alguns parâmetros e retornos algum tipo de objeto dependendo dos parâmetros passados.

O padrão Factory ajuda a criar objetos de tipos diferentes em vez de os objetos serem diretamente instanciados.

Aplicabilidade

nappacademy

O framework Django usa o padrão Factory Method para criar os campos de um formulário.

O módulo de formulários do Django suporta a criação de diferentes tipos de campos (CharField, EmailField) e customizações (max_length, obrigatório, etc).

Por que usar Fábricas ?

nappacademy

As fábricas fornecem acoplamento fraco, separando a criação de objetos (instanciação) do uso da classe específica.

Uma classe que usa o objeto criado não precisa saber exatamente qual classe é criada.

Tudo que o programador precisa saber é a interface da classe criada, ou seja, qual os métodos da classe criada podem ser chamados e com quais argumentos.

simple_factory.py

Variações de Fábricas

nappacademy

Há três variantes:

Simple Factory: permite que as interfaces criem objetos sem expor a lógica de sua criação

Factory Method: permite que as interfaces criem objetos, mas adia a decisão para a classe sobre a criação do objeto.

Abstract Factory: uma fábrica abstrata é uma interface para criar objetos relacionados sem especificar/expor suas classes. O padrão fornece objetos de outra fábrica, que internamente cria outros objetos.

Variações de Fábricas

nappacademy

Para alguns, o simple Factory não é, por si, um padrão, é mais um conceito que os desenvolvedores devem conhecer antes de saberem mais sobre o Factory Method e o Abstract Factory.

Factory Method

Factory Method

nappacademy

Imagine que você está criando uma aplicação de gerenciamento de logística.

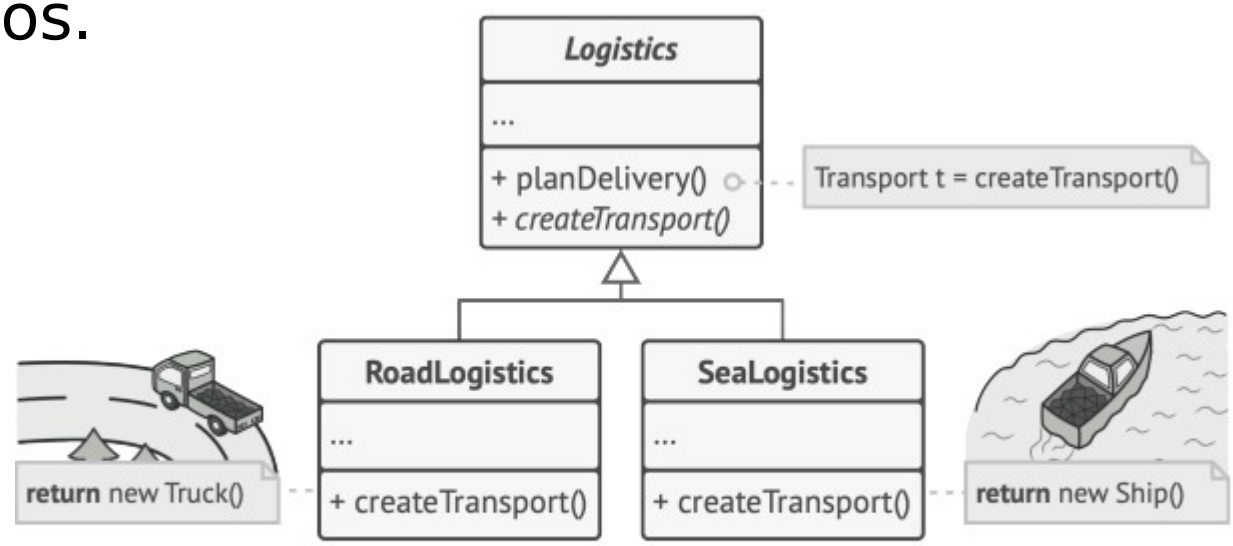
A primeira versão da sua aplicação pode lidar apenas com o transporte de caminhões, portanto a maior parte do seu código fica dentro da classe Caminhão.

Depois de um tempo, sua aplicação se torna bastante popular. Todos os dias você recebe dezenas de solicitações de empresas de transporte marítimo para incorporar a logística marítima na aplicação.

Factory Method

nappacademy

Com o Factory Method você substitui chamadas diretas de construção de objetos por chamadas para um método fábrica especial. Objetos retornados por um método fábrica geralmente são chamados de produtos.

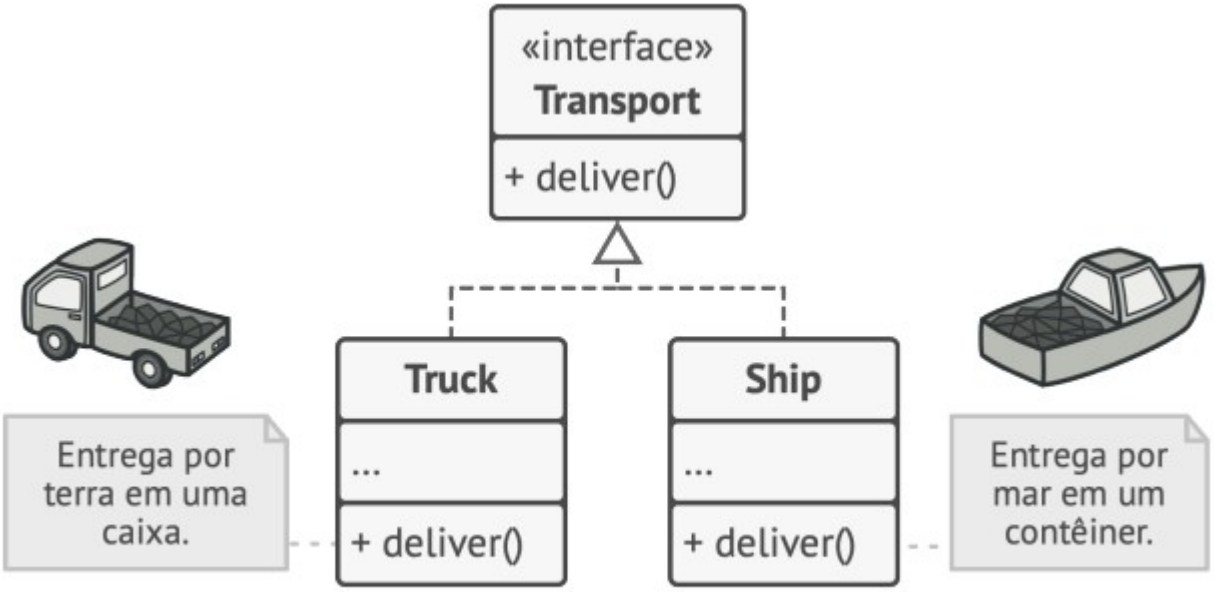


As subclasses podem alterar a classe de objetos retornados pelo método fábrica.

Factory Method

nappacademy

Porém, há uma limitação: as subclasses só podem retornar tipos diferentes de produtos se esses produtos tiverem uma classe ou interface base em comum



Todos os produtos devem seguir a mesma interface.

Factory Method

nappacademy

Por exemplo, ambas as classes ***Caminhão*** e ***Navio*** devem implementar a interface ***Transporte*** , que declara um método chamado **entregar** (deliver) .

Cada classe implementa esse método de maneira diferente: caminhões entregam carga por terra, navios entregam carga por mar.

O método fábrica na classe *LogísticaViária* retorna objetos de caminhão, enquanto o método fábrica na classe *LogísticaMarítima* retorna navios.

Factory Method

nappacademy

O código que usa o método fábrica (geralmente chamado de código cliente) não vê diferença entre os produtos reais retornados por várias subclasses.

O cliente trata todos os produtos como um Transporte abstrato.

O cliente sabe que todos os objetos de transporte devem ter o método **entregar** (deliver), mas como exatamente ele funciona não é importante para o cliente.

Factory Method

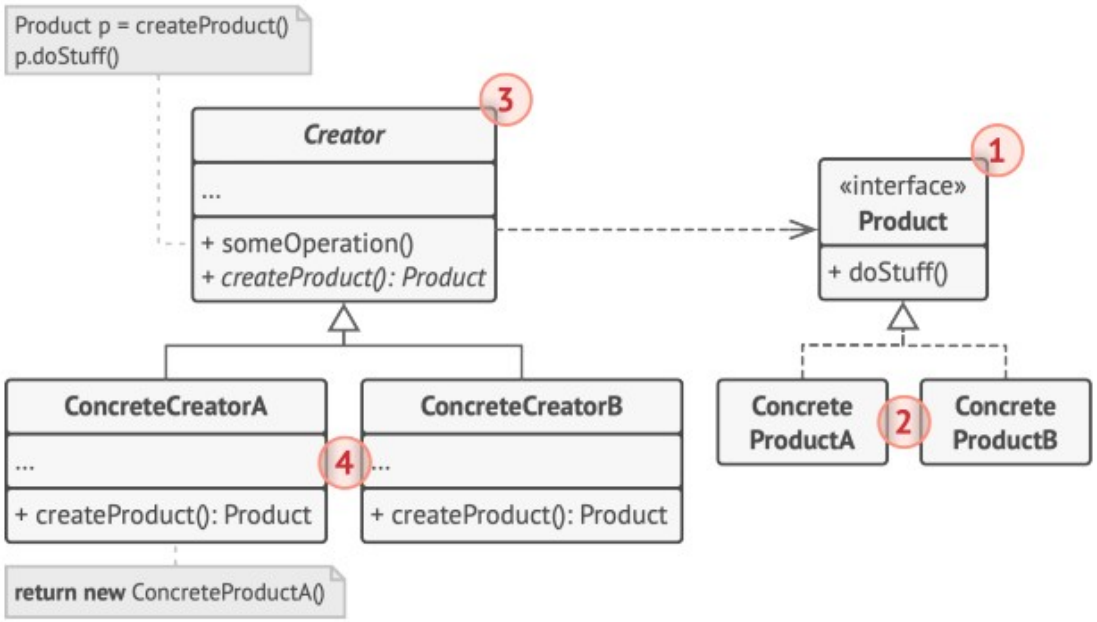
nappacademy

1. O Produto declara a interface, que é comum a todos os objetos.

2. Produtos Concretos são implementações diferentes da interface do produto

3. A classe Criador declara o método fábrica que retorna novos objetos produto.

4. Criadores Concretos sobrescrevem o método fábrica base para retornar um tipo diferente de produto



Aplicabilidade

nappacademy

Use o Factory Method quando deseja economizar recursos do sistema reutilizando objetos existentes em vez de recriá-los sempre.

Use o Factory Method quando desejar fornecer aos usuários da sua biblioteca ou framework uma maneira de estender seus componentes internos.

Use o Factory Method quando não souber de antemão os tipos e dependências exatas dos objetos com os quais seu código deve funcionar.

Abstract Factory

Abstract Factory

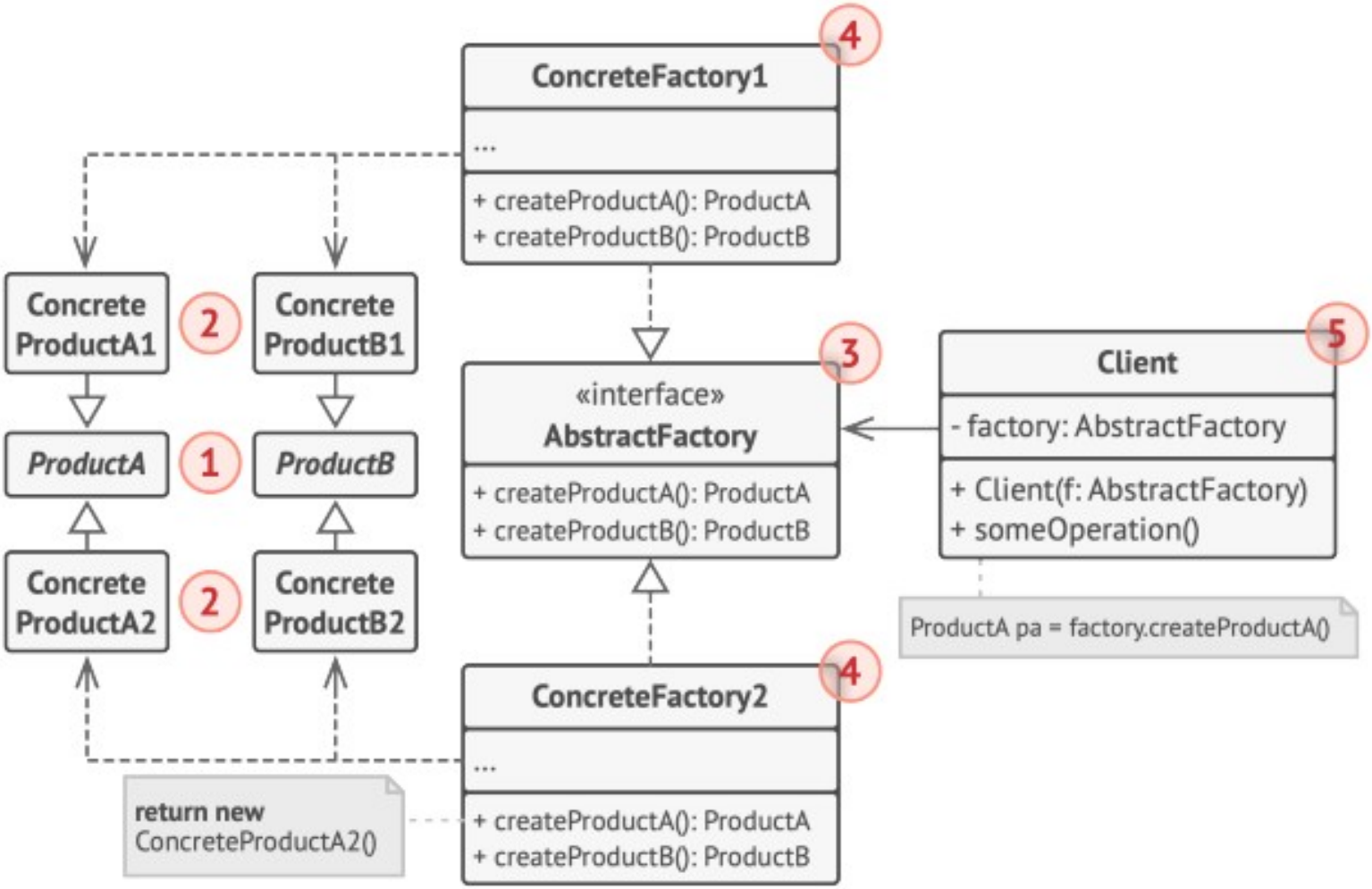
nappacademy

O Abstract Factory é um padrão de projeto criacional que permite que você produza **famílias de objetos** relacionados sem ter que especificar suas classes concretas.

abstract_factory.py

Abstract Factory

nappacademy



Abstract Factory

nappacademy

1. Produtos Abstratos declaram interfaces para um conjunto de produtos distintos mas relacionados que fazem parte de uma família de produtos.

2. Produtos Concretos são várias implementações de produtos abstratos, agrupados por variantes. Cada produto abstrato deve ser implementado em todas as variantes dadas.

3. A interface Fábrica Abstrata declara um conjunto de métodos para criação de cada um dos produtos abstratos.

Abstract Factory

nappacademy

4. Fábricas Concretas implementam métodos de criação fábrica abstratos. Cada fábrica concreta corresponde a uma variante específica de produtos e cria apenas aquelas variantes de produto.

5. Embora fábricas concretas instanciam produtos concretos, assinaturas dos seus métodos de criação devem retornar produtos abstratos correspondentes.O Cliente pode trabalhar com qualquer variante de produto/fábrica concreto, desde que ele se comunique com seus objetos via interfaces abstratas.

Aplicabilidade

nappacademy

Use o Abstract Factory quando seu código precisa trabalhar com diversas famílias de produtos relacionados, mas que você não quer depender de classes concretas daqueles produtos.

Muitos projetos começam usando o Factory Method (menos complicado e mais customizável através de subclasses) e evoluem para o Abstract Factory.

Comparação

nappacademy

Factory Method	Abstract Factory
Expõe um método ao cliente para criar os objetos.	O método Abstract Factory contém um ou mais métodos de fábrica para criar uma família de objetos relacionados.
Usa herança e subclasses para definir o objeto a ser criado.	Usa composição para delegar a responsabilidade de criar objetos de outra classe.
O Factory Method é usado para criar um produto.	O método Abstract Factory diz respeito à criar famílias de produtos relacionados.

Prós / Contras

nappacademy

- Você evita acoplamentos firmes entre o criador e os produtos concretos (*Factory Method*) ou pode ter certeza que os produtos que você obtém de uma fábrica são compatíveis entre si (*Abstract Factory*).
- Princípio de responsabilidade única. Você pode extrair o código de criação do produto para um lugar, fazendo o código ser de fácil manutenção.
- Princípio aberto/fechado. Você pode introduzir novas variantes de produtos sem quebrar o código cliente existente.

O código pode tornar-se mais complicado do que deveria ser, uma vez que muitas novas interfaces e classes são introduzidas junto com o padrão.