O **Factory Method** é um padrão de projeto que fornece uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objetos que serão criados.

**Problema:** Vamos imaginar a criação de uma ferramenta para gerenciamento de logística. A primeira versão da sua aplicação lida apenas com o transporte de caminhões, ou seja, implementamos apenas a classe Caminhao e criamos toda a nossa aplicação com base nela.

Depois de um tempo, a aplicação se torna bastante popular e você começa a receber solicitações de empresas de transporte marítimo para incorporar a logística marítima na aplicação. Eis que temos um problema: Acoplamos a maioria do nosso código na classe Caminhao e para adicionar Navio, por exemplo, teríamos que fazer alterações em toda base dela. Inclusive se tivéssemos que adicionar um outro veículo posteriormente, teríamos o mesmo problema. Isso resultaria num código sujo, repleto de condicionais que alteram o comportamento da aplicação dependendo do problema.

**Solução:** O padrão sugere que as chamadas diretas de construção de objetos sejam substituídas por um método de fábrica especial. Agora, poderemos sobrescrever o método fábrica em uma subclasse e alterar a classe que será criada pelo método.

Existe um porém: As subclasses só podem retornar tipos diferentes de produtos se estes produtos tiverem uma classe ou interface base em comum. Além disso, o método fábrica na classe base deve ter seu retorno declarado como essa classe ou interface.

Exemplo: Ambas as classes Caminhao e Navio devem implementar Transporte, que declara um método chamado Entregar. Cada classe implementa esse método de maneira diferente: Caminhões entregam a carga por terra e navios por mar. O método fábrica LogisticaViaria retorna objetos de caminhão, enquanto o método fábrica LogisticaMaritima retorna objetos de navio.

**Aplicabilidade:** Use o Factory Method quando não souber de antemão os tipos e dependências exatas dos objetos com os quais o código deve funcionar. Use também quando deseja fornecer aos usuários da sua biblioteca ou framework uma forma de estender seus componentes internos. Por último, use-o quando deseja economizar recursos do sistema, reutilizando objetos existentes em vez de recriá-los sempre.

**Prós:** Evitamos acoplamentos entre o criados e os produtos concretos (subprodutos). Seguimos os *princípios da responsabilidade única*, onde podemos mover o código de criação do produto para um único local do programa. *Princípio aberto/fechado,* podemos introduzir novos tipos de produtos sem quebrar o código inexistente.

**Contras:** O código pode se tornar mais complicado, pois é necessário introduzir muitas subclasses para alterar o comportamento padrão.