



#R4E

Software Developer

# Design Patterns

Factory Method



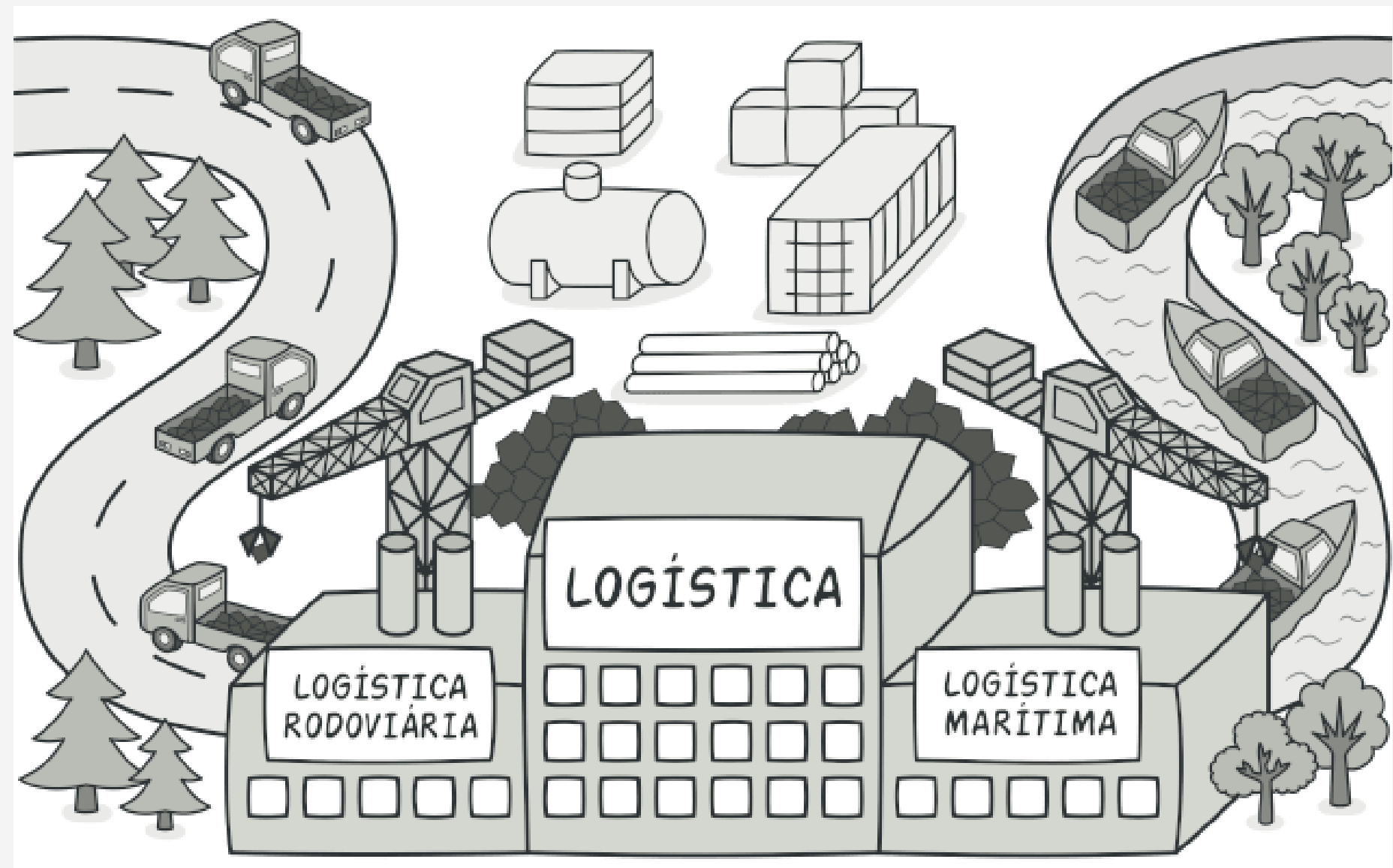
# Conteúdo



- Propósito
- Problema
- Solução
- Estrutura
- Aplicabilidade
- Como Implementar
- Prós e Contras
- Relações c/ Outros Padrões

# Propósito

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



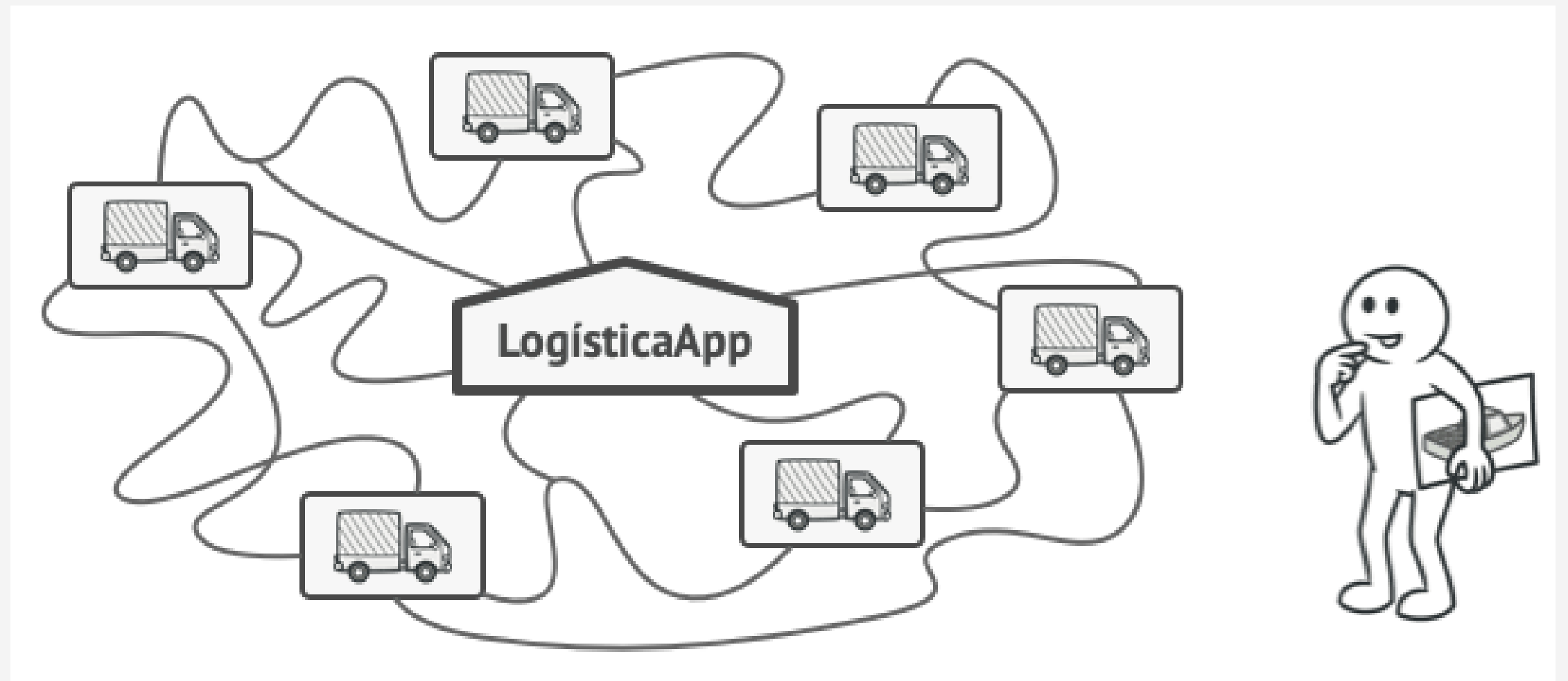
# Problema



- Imagine que está a criar uma aplicação de gestão de logística. A primeira versão da sua aplicação pode lidar apenas com o transporte de camiões, portanto a maior parte do seu código fica dentro da classe Camião.
- Depois de um tempo, a aplicação torna-se bastante popular. Todos os dias recebe dezenas de solicitações de empresas de transporte marítimo para incorporar a logística marítima na aplicação.

# Problema

- Adicionar uma nova classe ao programa não é tão simples se o restante do código já estiver acoplado às classes existentes.

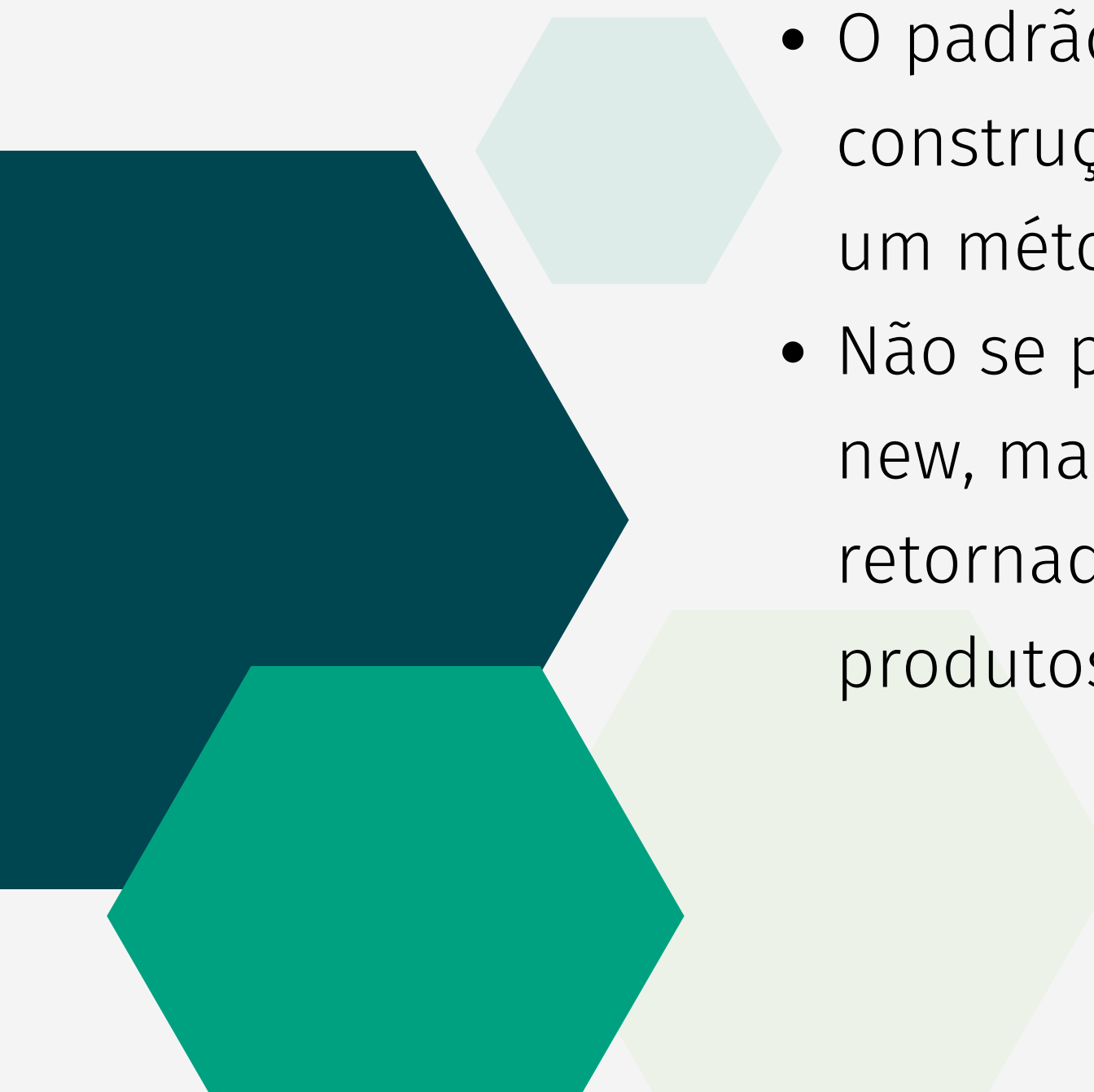


# Problema



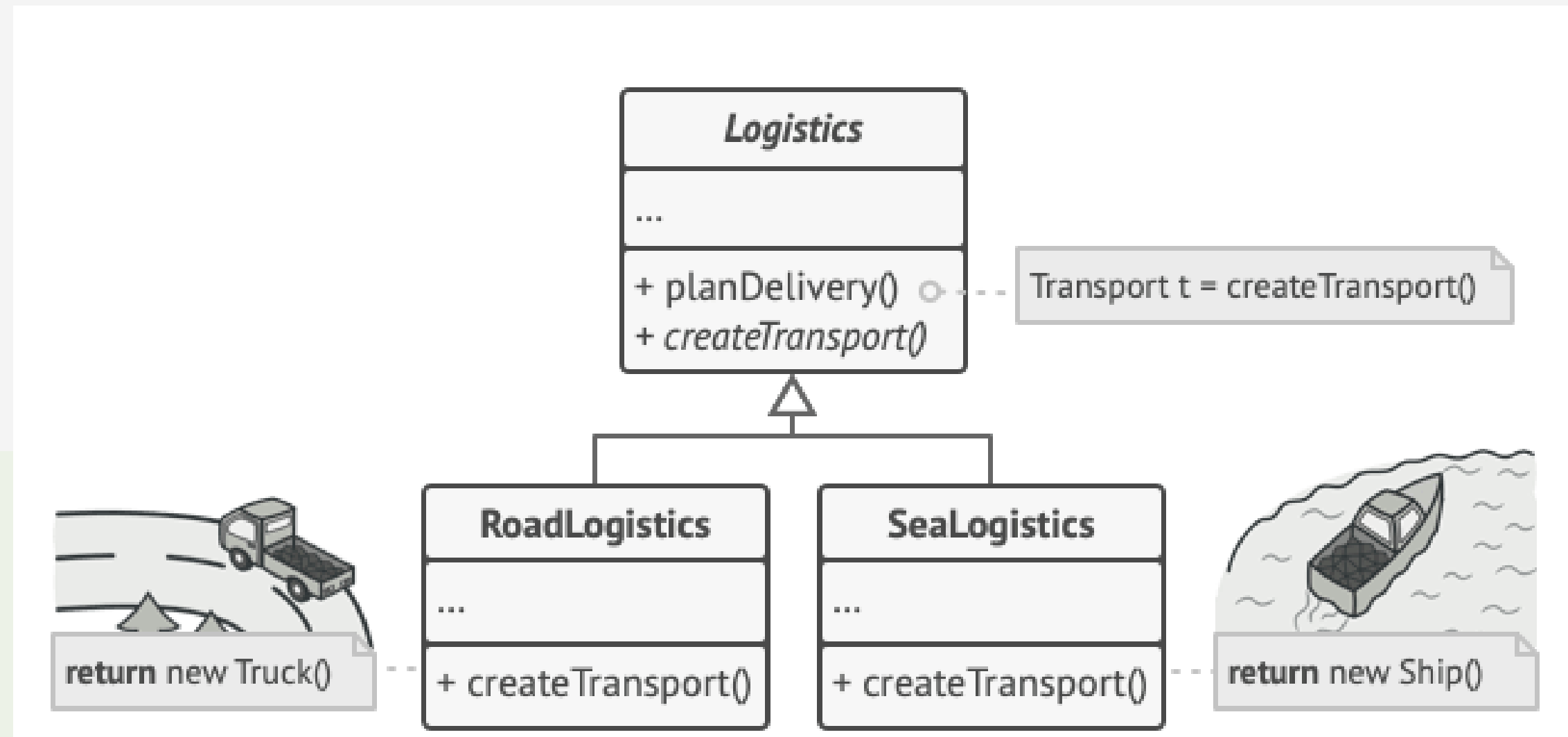
- Boa notícia, certo? Mas e o código? Atualmente, a maior parte do seu código é acoplada à classe Camião. Adicionar Navio à aplicação exigiria alterações por toda a base de código. Além disso, se mais tarde decidir adicionar outro tipo de transporte à aplicação, provavelmente vai ser necessário fazer essas alterações todas novamente.
- Como resultado, terá um código bastante "sujo", repleto de condicionais que alteram o comportamento da aplicação, dependendo da classe de objetos de transporte.

# Solução

- 
- O padrão **Factory Method** sugere que substitua chamadas diretas de construção de objetos (usando o operador new) por chamadas para um método fábrica especial.
  - Não se preocupe: os objetos ainda são criados através do operador new, mas esse será invocado dentro do método fábrica. Objetos retornados por um método fábrica geralmente são chamados de produtos.

# Solução

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





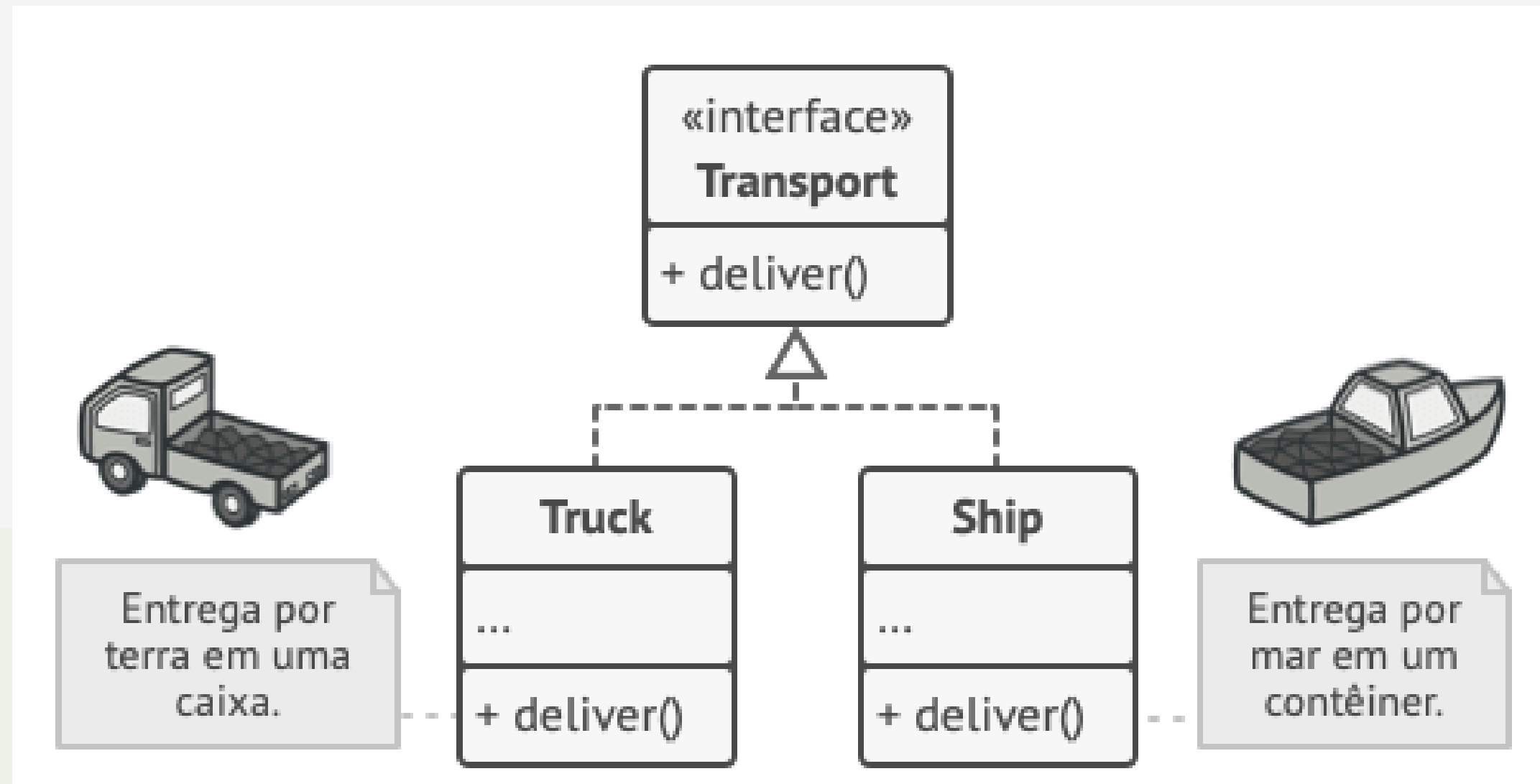
# Solução



- À primeira vista, essa mudança pode parecer sem sentido: apenas mudamos a invocação do construtor de uma parte do programa para outra. No entanto, considere o seguinte: agora é possível sobrescrever o método fábrica numa subclasse e alterar a classe de produtos que estão a ser criados pelo método.
- Porém, há uma pequena limitação: as subclasses só podem retornar tipos diferentes de produtos se esses produtos tiverem uma classe ou interface base em comum. Além disso, o método fábrica na classe base deve ter tipo de retorno declarado como essa interface.

# Solução

- Todos os produtos devem seguir a mesma interface.

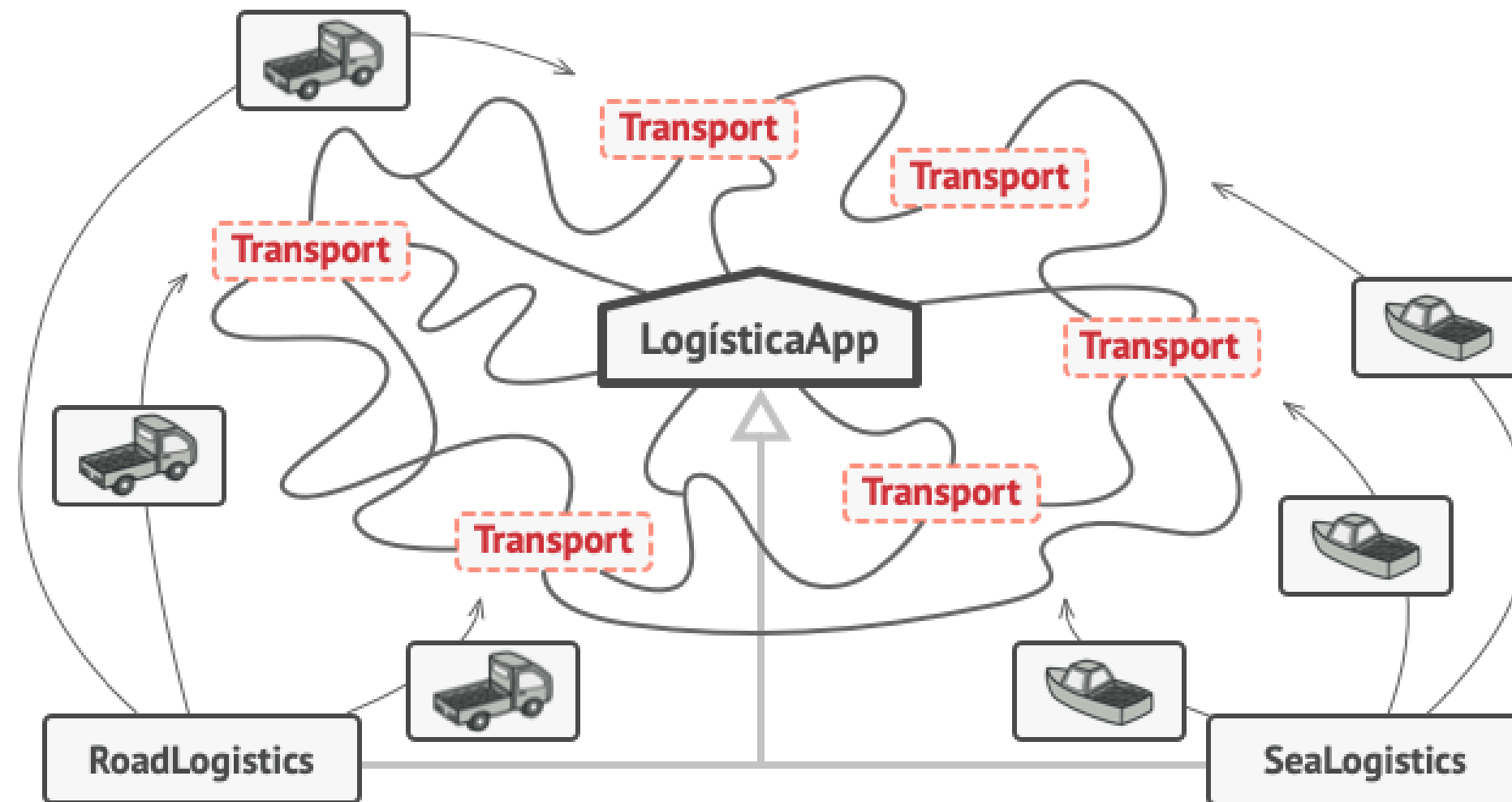


# Solução

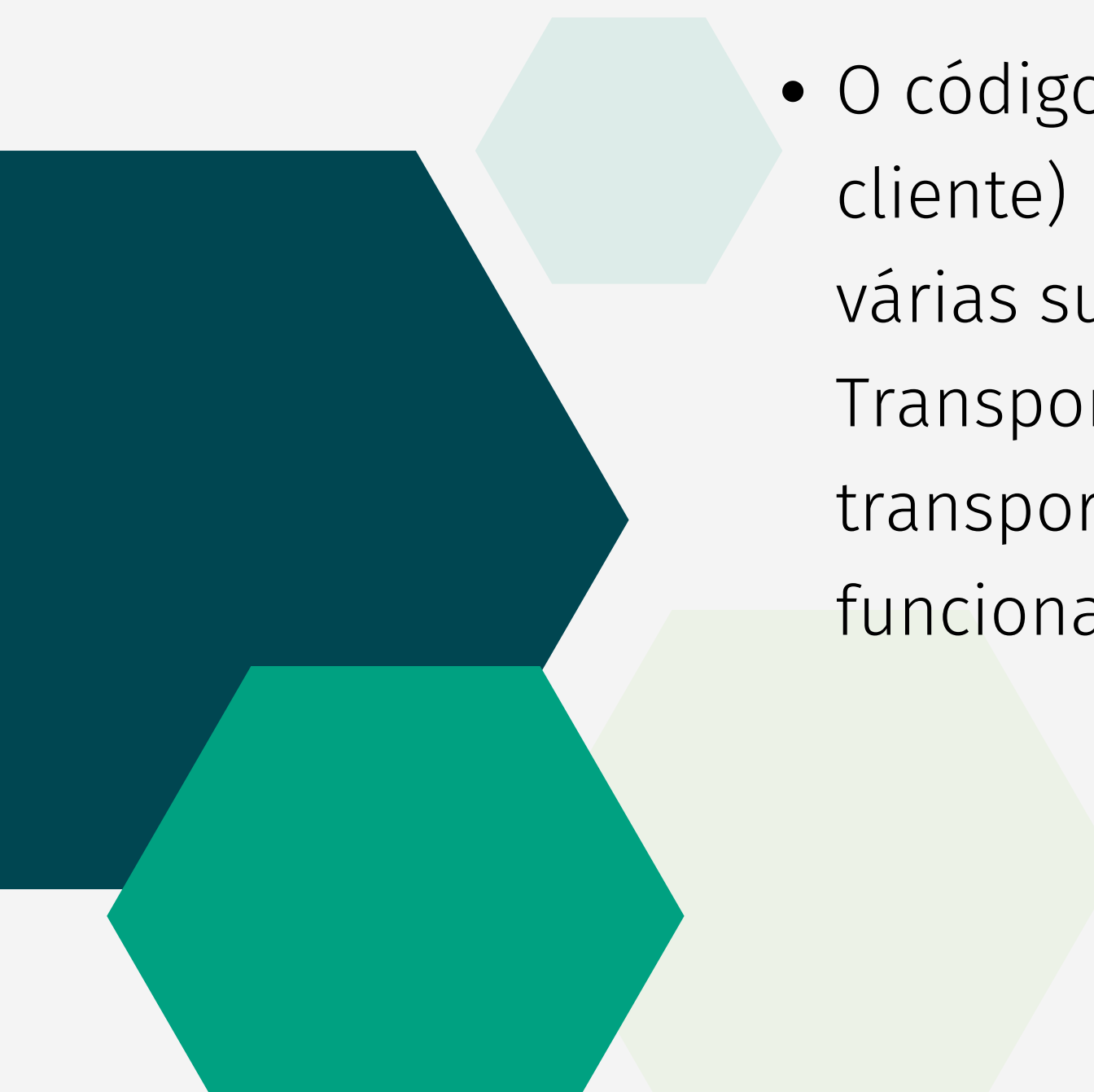
- Por exemplo, ambas as classes Camião e Navio devem implementar a interface Transporte, que declara um método chamado entregar. Cada classe implementa esse método de maneira diferente: caminhões entregam carga por terra, navios entregam carga por mar. O método fábrica na classe LogísticaVia retorna objetos de camião, enquanto o método fábrica na classe LogísticaMarítima retorna navios.

# Solução

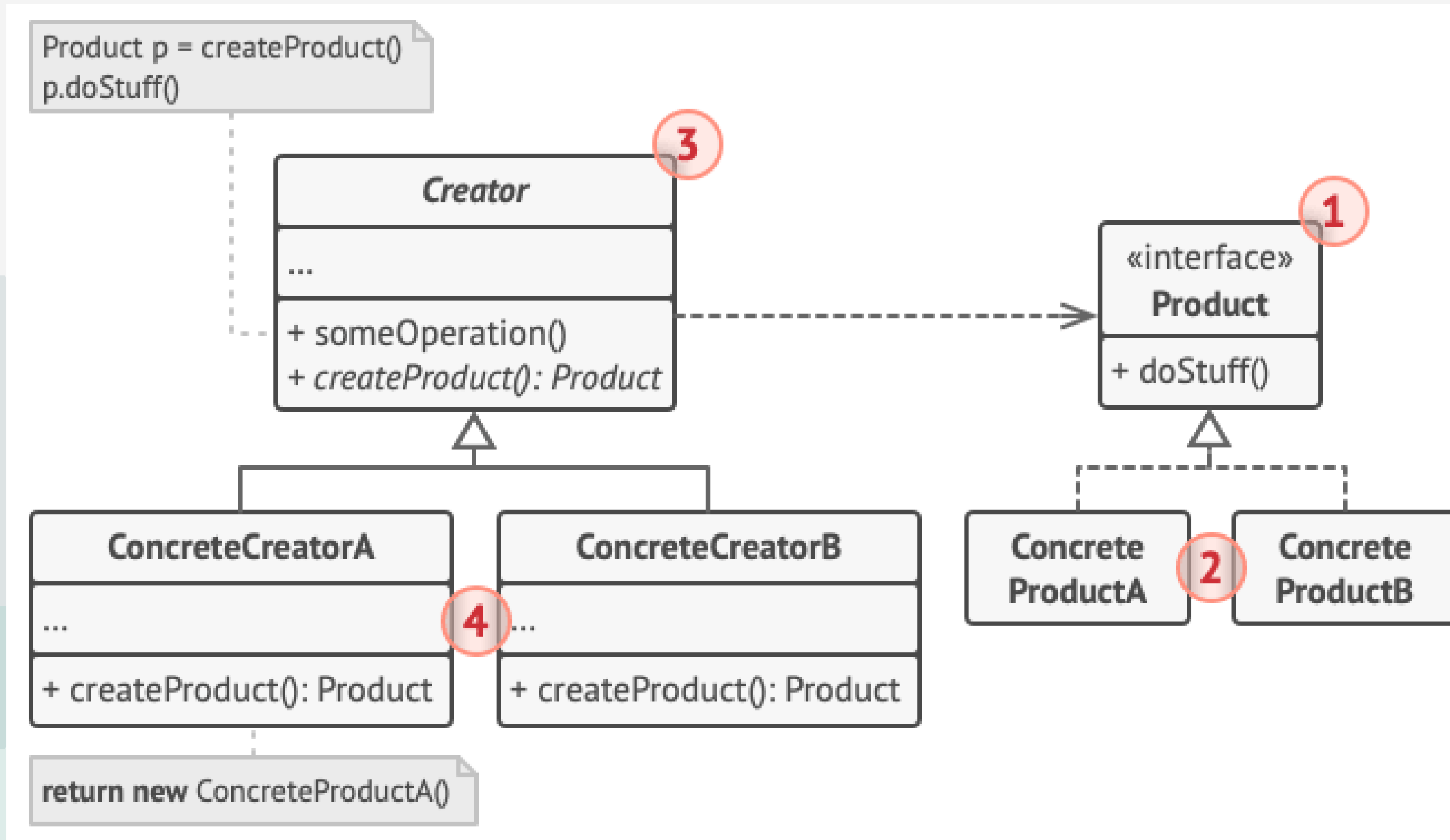
- Desde que todas as classes de produtos implementem uma interface comum, você pode passar seus objetos para o código cliente sem quebrá-lo.




# Solução

- 
- 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, mas como exatamente ele funciona não é importante para o cliente.

# Estrutura



# Estrutura

- 
1. O Produto declara a interface, que é comum a todos os objetos que podem ser produzidos pelo criador e suas subclasses.
  2. Produtos Concretos são implementações diferentes da interface do produto.
  3. A classe Criador declara o método fábrica que retorna novos objetos produto. É importante que o tipo de retorno desse método corresponda à interface do produto.

# Estrutura

- Podemos declarar o método fábrica como abstrato para forçar todas as subclasses a implementar as suas próprias versões do método. Como alternativa, o método fábrica base pode retornar algum tipo de produto padrão.
- Observe que, apesar do nome, a criação de produtos não é a principal responsabilidade do criador. Normalmente, a classe criadora já possui alguma lógica de negócio relacionada aos produtos. O método fábrica ajuda a dissociar essa lógica das classes concretas de produtos. Analogia: uma grande empresa de desenvolvimento de software pode ter um departamento de formação para programadores. No entanto, a principal função da empresa como um todo ainda é escrever código, não produzir programadores.



# Estrutura

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

- Observe que o método fábrica não precisa de criar novas instâncias constantemente. Também pode retornar objetos existentes de uma cache, um conjunto de objetos, ou outra fonte.

# Aplicabilidade

- Utilize o Factory Method quando deseja fornecer aos utilizadores da sua biblioteca, framework ou programa uma maneira de estender seus componentes internos.
  - Herança é provavelmente a maneira mais fácil de estender o comportamento padrão de uma biblioteca ou framework. Mas como o framework reconhece que a sua subclasse deve ser usada em vez de um componente padrão?
  - A solução é reduzir o código que constrói componentes no framework num único método fábrica e permitir que qualquer pessoa sobrescreva esse método, além de estender o próprio componente.

# Aplicabilidade

- Vamos ver como funciona. Imagina que escreveste uma aplicação usando um framework de UI de código aberto. A aplicação deve ter botões redondos, mas o framework fornece apenas botões quadrados. Estendes a classe padrão Botão com uma gloriosa subclasse BotãoRedondo. Mas agora é preciso informar à classe principal UIFramework para usar a nova subclasse no lugar do botão padrão.
- Para conseguir isso, criamos uma subclasse UIComBotõesRedondos a partir de uma classe base do framework e sobreescreve o método criarBotão. Enquanto este método retorna objetos Botão na classe base, faz sua subclasse retornar objetos BotãoRedondo. Agora use a classe UIComBotõesRedondos no lugar de UIFramework. E é isso!

# Aplicabilidade

- Use o Factory Method quando deseja economizar recursos do sistema reutilizando objetos existentes em vez de recriá-los sempre.
  - Será indispensável enfrentar essa necessidade ao lidar com objetos grandes e pesados, como conexões com bases de dados, sistemas de arquivos e recursos de rede.

# Aplicabilidade

- Vamos pensar no que deve ser feito para reutilizar um objeto existente:
  1. Primeiro, precisamos de criar algum armazenamento para controlar todos os objetos criados.
  2. Quando alguém solicita um objeto, o programa deve procurar um objeto livre dentro desse conjunto.
  - 3....e retorná-lo ao código cliente.
  4. Se não houver objetos livres, o programa deve criar um novo (e adicioná-lo ao conjunto de objetos).
- Isso é muito código! E tudo deve estar colocado num único local para que não polua o programa com código duplicado.

# Aplicabilidade

- Provavelmente, o lugar mais óbvio e conveniente onde esse código deve ficar é no construtor da classe cujos objetos estamos a tentar reutilizar. No entanto, um construtor deve sempre retornar novos objetos por definição. Não pode retornar instâncias existentes.
- Portanto, é preciso ter um método regular capaz de criar novos objetos e reutilizar os existentes. Isto é muito parecido com o método fábrica.

# Como Implementar

1. Faça todos os produtos implementarem a mesma interface. Essa interface deve declarar métodos que fazem sentido em todos os produtos.
2. Adicione um método fábrica vazio dentro da classe criadora. O tipo de retorno do método deve corresponder à interface comum do produto.

# Como Implementar

3. No código da classe criadora, encontre todas as referências aos construtores de produtos. Um por um, substitua-os por invocações ao método fábrica, enquanto extrai o código de criação do produto para o método fábrica.
- Pode ser necessário adicionar um parâmetro temporário ao método fábrica para controlar o tipo de produto retornado.
  - Neste ponto, o código do método fábrica pode parecer bastante feio. Pode ter um grande operador switch que escolhe qual a classe de produto instanciar. Mas vamos resolver isso brevemente.



# Como Implementar

3. No código da classe criadora, encontre todas as referências aos construtores de produtos. Um por um, substitua-os por invocações ao método fábrica, enquanto extrai o código de criação do produto para o método fábrica.
  - Pode ser necessário adicionar um parâmetro temporário ao método fábrica para controlar o tipo de produto retornado.
  - Neste ponto, o código do método fábrica pode parecer bastante feio. Pode ter um grande operador switch que escolhe qual a classe de produto instanciar. Mas vamos resolver isso brevemente.
4. Agora, crie um conjunto de subclasses criadoras para cada tipo de produto listado no método fábrica. Sobrescreva o método fábrica nas subclasses e extraia os pedaços apropriados do código de construção do método base.

# Como Implementar

5. Se houver muitos tipos de produtos e não fizer sentido criar subclasses para todos, pode reutilizar o parâmetro de controlo da classe base nas subclasses.
- Por exemplo, imagine que tem a seguinte hierarquia de classes: a classe base `Correio` com algumas subclasses: `CorreioAéreo` e `CorreioTerrestre`; as classes `Transporte` são `Avião`, `Camião` e `Comboio`. Enquanto a classe `CorreioAéreo` usa apenas objetos `Avião`, o `CorreioTerrestre` pode funcionar com os objetos `Camião` e `Comboio`. Podemos criar uma nova subclasse (por exemplo, `CorreioFerroviário`) para lidar com os dois casos, mas há outra opção. O código do cliente pode passar um argumento para o método fábrica da classe `CorreioTerrestre` para controlar qual produto deseja receber.

# Como Implementar

The slide features several overlapping hexagonal shapes in the bottom-left corner. These shapes are in various shades of teal, light green, and pale yellow, creating a modern, abstract background element.

6. Se, após todas as extrações, o método fábrica base ficar vazio, pode torná-lo abstrato. Se sobrar algo, pode tornar isso num comportamento padrão do método.

# Prós

- Evita acoplamentos firmes entre o criador e os produtos concretos.
- Princípio de responsabilidade única. Pode mover o código de criação do produto para um único local do programa, facilitando a manutenção do código.
- Princípio aberto/fechado. Pode introduzir novos tipos de produtos no programa sem comprometer o código cliente existente.

# Contras

- O código pode escalar e ficar mais complicado, pois é preciso introduzir muitas subclasses novas para implementar o padrão. O melhor cenário é quando está a introduzir o padrão numa hierarquia existente de classes criadoras.

# Relações c/ Outros Padrões

- Muitos projetos começam por usar o Factory Method (menos complicado e mais customizável através de subclasses) e evoluem para o Abstract Factory, Prototype, ou Builder (mais flexíveis, mas mais complicados).
- Classes Abstract Factory são quase sempre baseadas num conjunto de métodos fábrica, mas também se pode usar o Prototype para compor métodos dessas classes.
- Pode ser usado o Factory Method junto com o Iterator para permitir que uma coleção de subclasses retornem diferentes tipos de iteradores que são compatíveis com as coleções.

# Relações c/ Outros Padrões

- O Prototype não é baseado em heranças, então ele não tem os seus inconvenientes. Por outro lado, o Prototype precisa de uma inicialização complicada do objeto clonado. O Factory Method é baseado em herança mas não precisa de uma etapa de inicialização.
- O Factory Method é uma especialização do Template Method. Ao mesmo tempo, o Factory Method pode servir como uma etapa num Template Method grande.



#R4E

Software Developer

# Design Patterns

Factory Method

