Arquitetura de Software

Padrões de Projetos – Padrões Estruturais

Nairon Neri Silva

Sumário

Padrões de Projeto Estruturais

1. *Adapter*
2. *Bridge*
3. *Composite*
4. *Decorator*
5. *Façade*
6. *Flyweight*
7. *Proxy*

Introdução

- Os Padrões Estruturais se preocupam como classes e objetos são compostos para formar estruturas maiores

- Utilizam a herança para compor interfaces ou implementações

Adapter

Também conhecido como ***Wrapper***

Intenção

- Converter a interface de uma classe em outra interface, esperada pelos clientes

- O ***Adapter*** permite que classes com interfaces incompatíveis trabalhem em conjunto – o que, de outra forma, seria impossível

Motivação

- Algumas vezes, uma classe de um *toolkit*, projetada para ser reutilizada não é reutilizável porque sua interface não corresponde à interface específica de um domínio requerida por uma aplicação

- Exemplo: considere um editor gráfico
  - A abstração chave é um objeto gráfico (definida pela classe abstrata ***Shape***)
  - Definida uma subclasse representar linhas – ***LineShape***
  - A classe ***PolygonShape*** pode representar polígonos e etc.

Motivação

- Continuando...
  - Porém, uma subclasse ***TextShape*** (que pode exibir texto) é mais difícil de ser implementada
  - Entretanto, pode já existir um *toolkit* para a construção de interfaces de usuário, o qual já oferece uma sofisticada classe ***TextView*** para a exibição e edição de textos
  - Idealmente, gostaríamos de reutilizar ***TextView*** para implementar ***TextShape***, porém este *toolkit* não foi projetado levando essa ideia em consideração

- Como classes existentes e não-relacionadas podem funcionar em uma aplicação que espera classes com uma interface diferente e incompatível?

Aplicabilidade

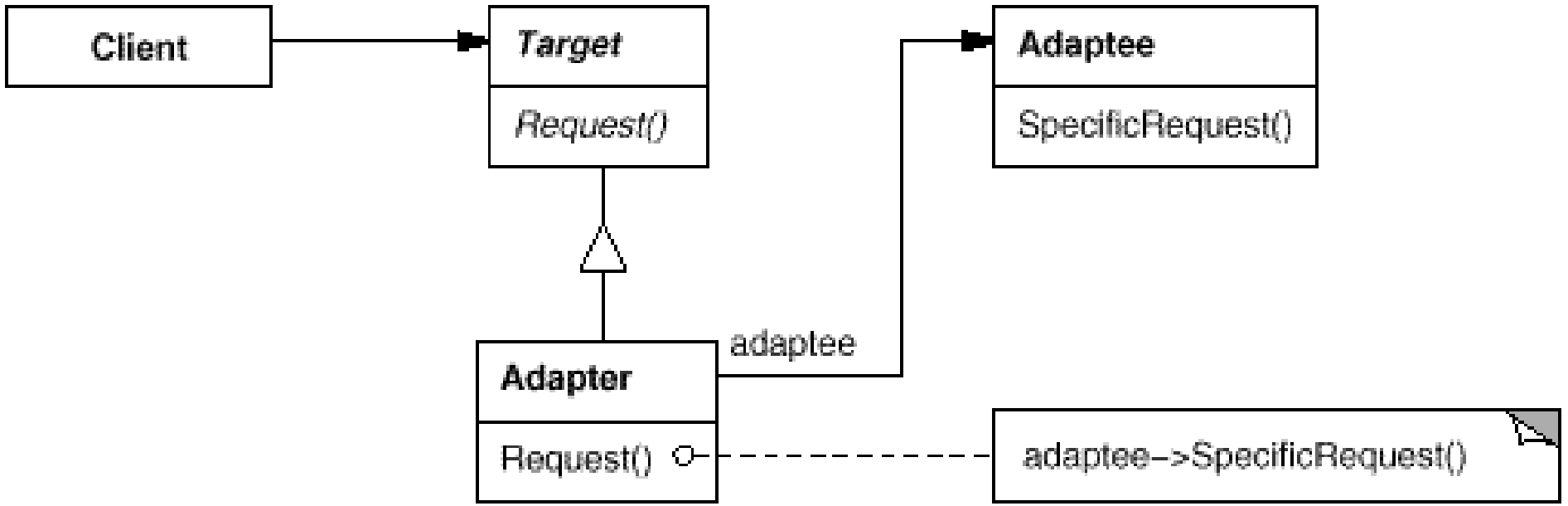
1. Queremos usar uma classe existente, mas sua interface não corresponde à interface de que necessitamos

2. Queremos criar uma classe reutilizável que coopera com classes não-relacionadas ou não-previstas

3. Necessitamos usar várias subclasses existentes, porém, é impraticável adaptar estas interfaces criando subclasses para cada uma (um adaptador pode adaptar a interface de sua classe mãe)

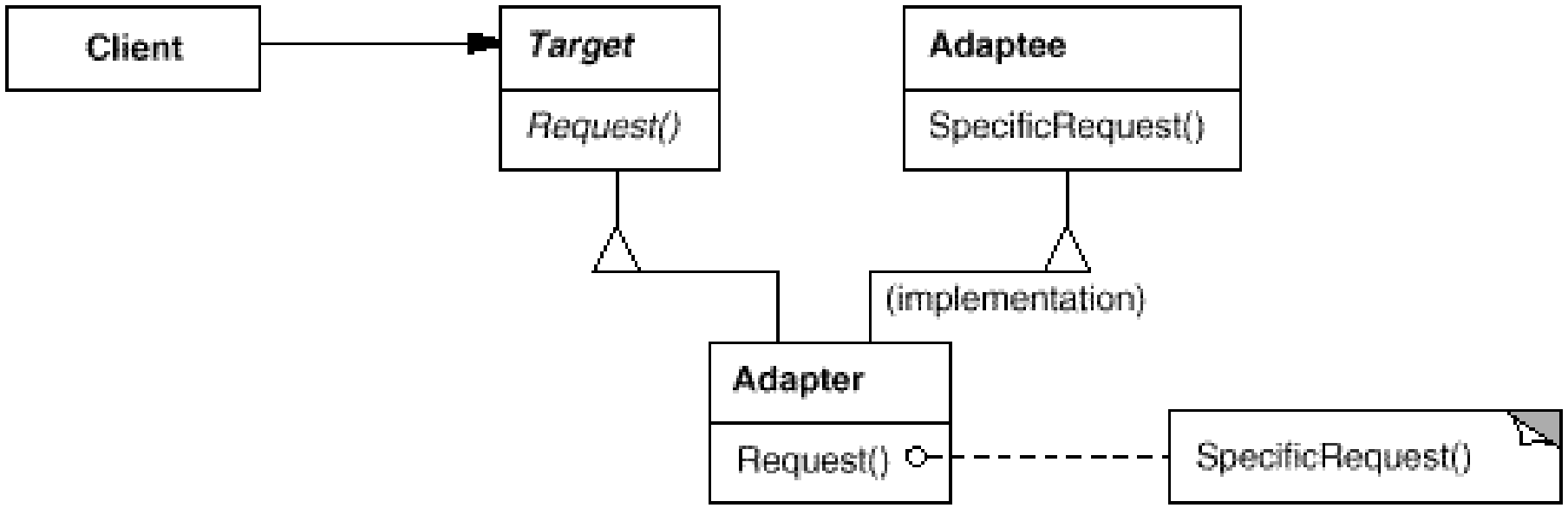
Estrutura

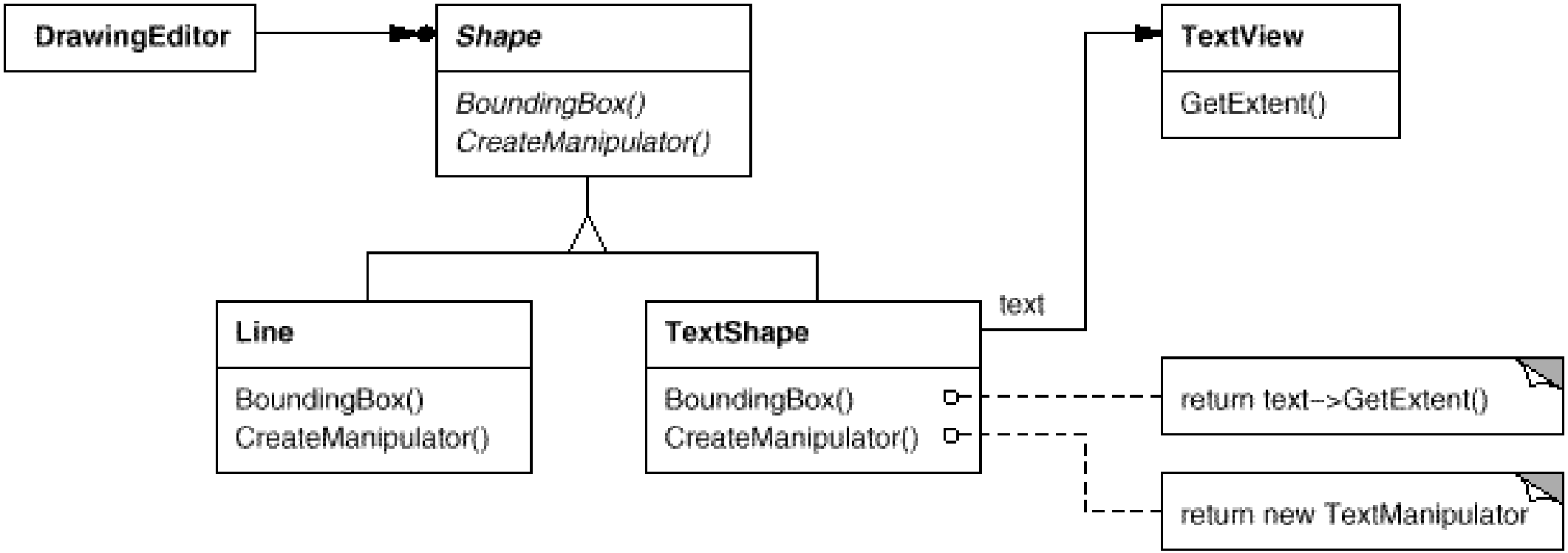
- **<u>Adaptador de objeto</u>**: dependente da composição de objetos:



Estrutura

- **<u>Adaptador de classe</u>**: usa herança múltipla para adaptar uma interface à outra:




DrawingEditor
Shape
BoundingBox()
CreateManipulator()
TextView
GetExtent()
Line
BoundingBox()
CreateManipulator()
TextShape
BoundingBox()
CreateManipulator()
text
return text->GetExtent()
return new TextManipulator

Exemplo/Participantes

1. ***Target (Shape)***
   - Define a interface específica do domínio usado por ***Client***
2. ***Client (DrawingEditor)***
   - Colabora com objetos compatíveis à interface de ***Target***
3. ***Adaptee (TextView)***
   - Define uma interface existente que precisa ser adaptada
4. ***Adapter (TextShape)***
   - Adapta a interface do ***Adaptee*** à interface de ***Target***

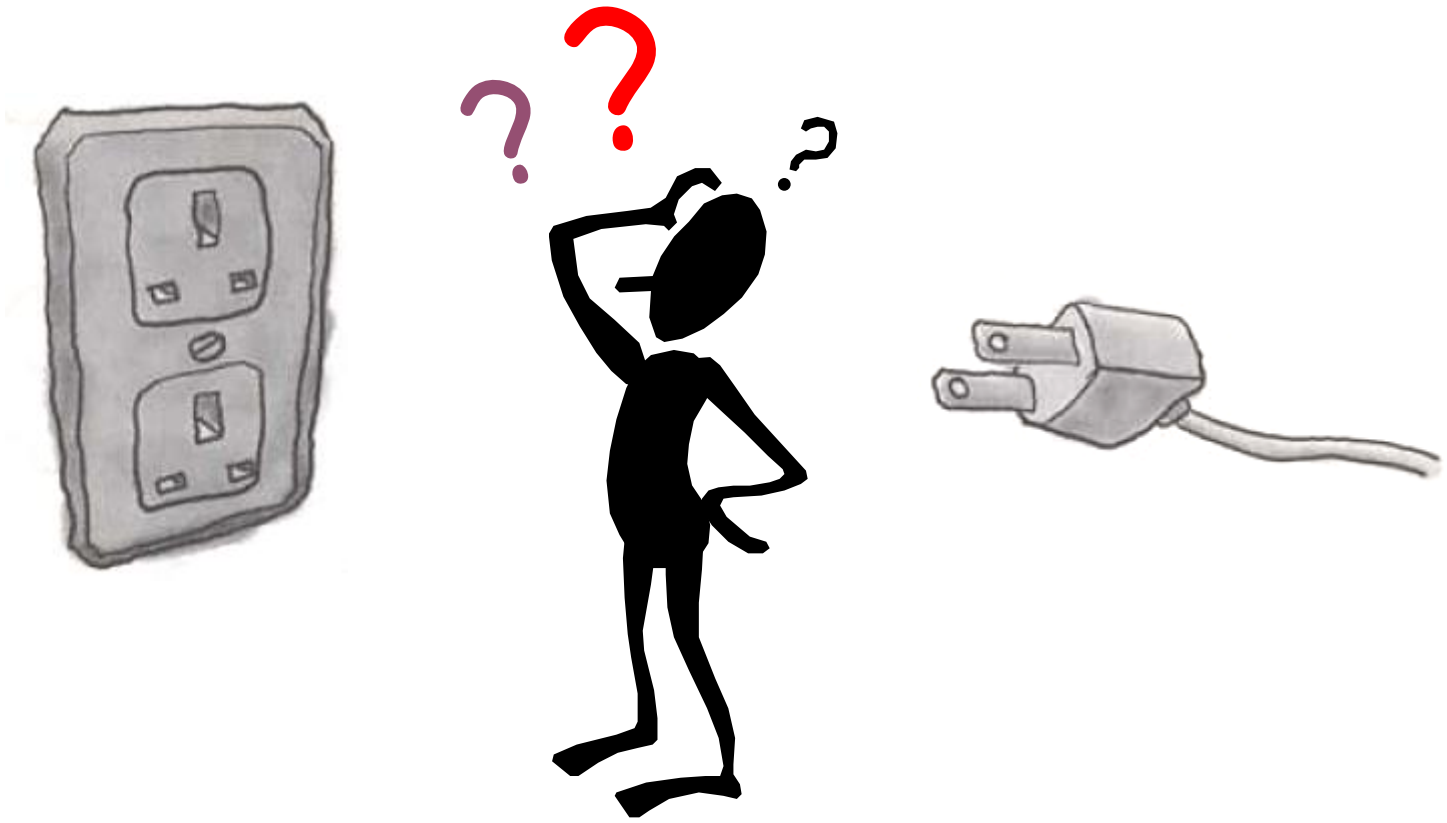
Consequências

- Boas
  - Desacopla a aplicação de aplicações de terceiros
  - Pode-se criar novos *adapters* no código existente

- Ruins
  - Aumenta a complexidade da aplicação porque você precisa introduzir um conjunto de novas interfaces e classes, porém gera um resultado positivo pois desacopla o código
  - Em determinadas situações, talvez seja mais adequado mudar a classe que fornece o serviço para ser compatível com o seu código

Exemplo: Adapter Pattern

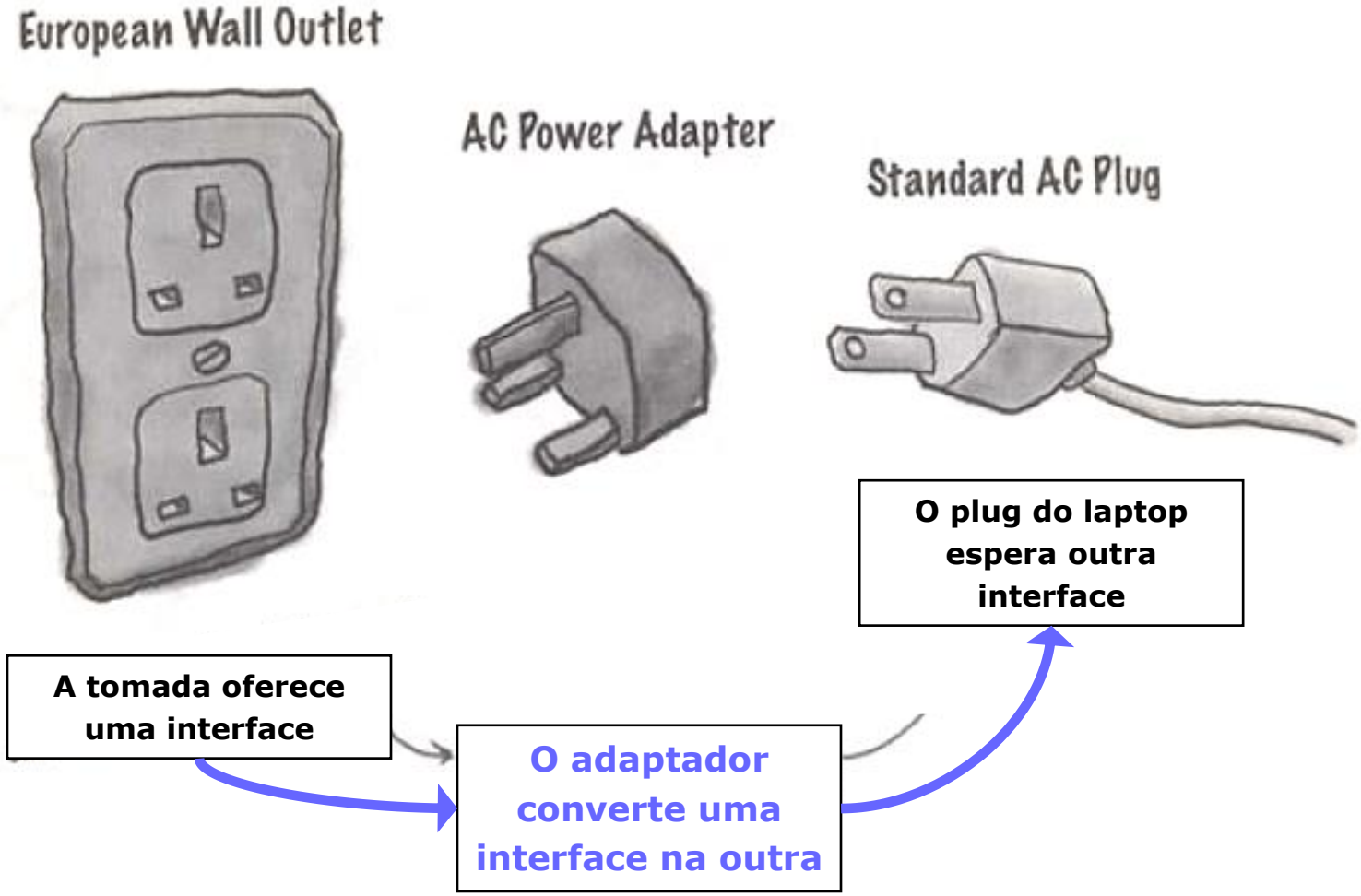
- Você já precisou ligar seu laptop na tomada num país estrangeiro?



Exemplo: Adapter Pattern

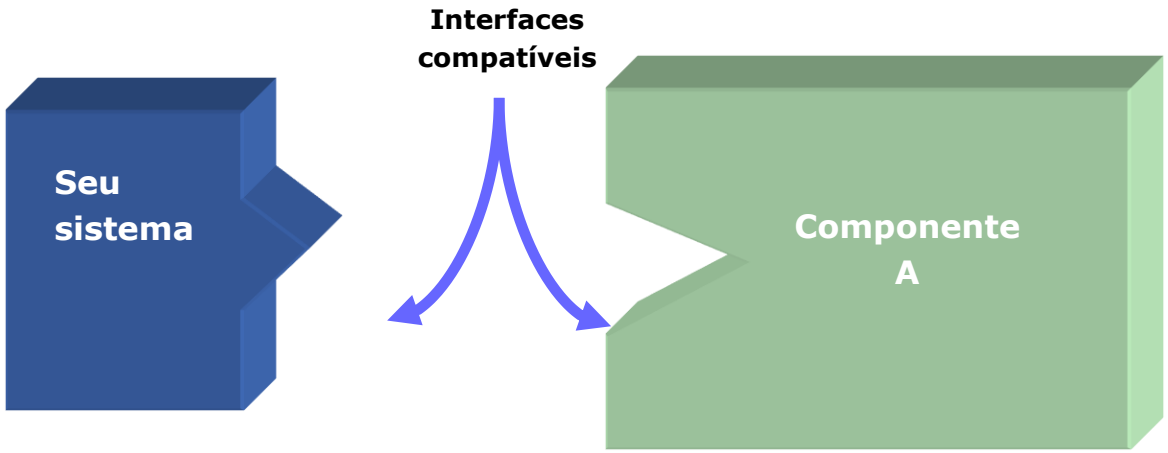
- Existem diversos adaptadores

Exemplo: Adapter Pattern



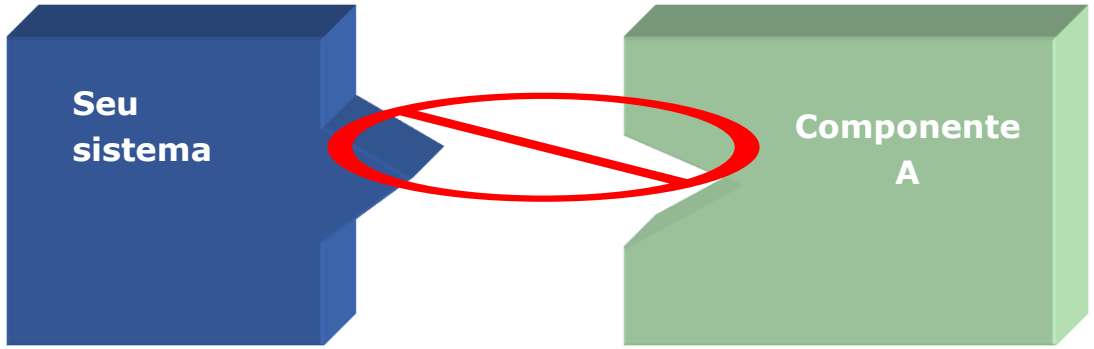
Exemplo: Adapter Pattern

- Suponha que você tem um sistema que usa o componente A



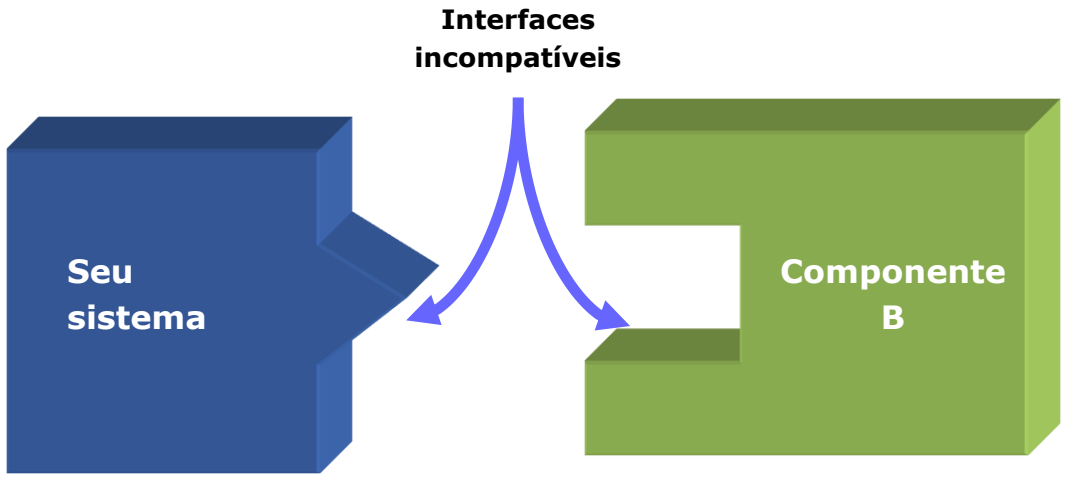
Exemplo: Adapter Pattern

- Porém o fornecedor do componente A faliu…
- Você precisa utilizar o componente de outro fornecedor



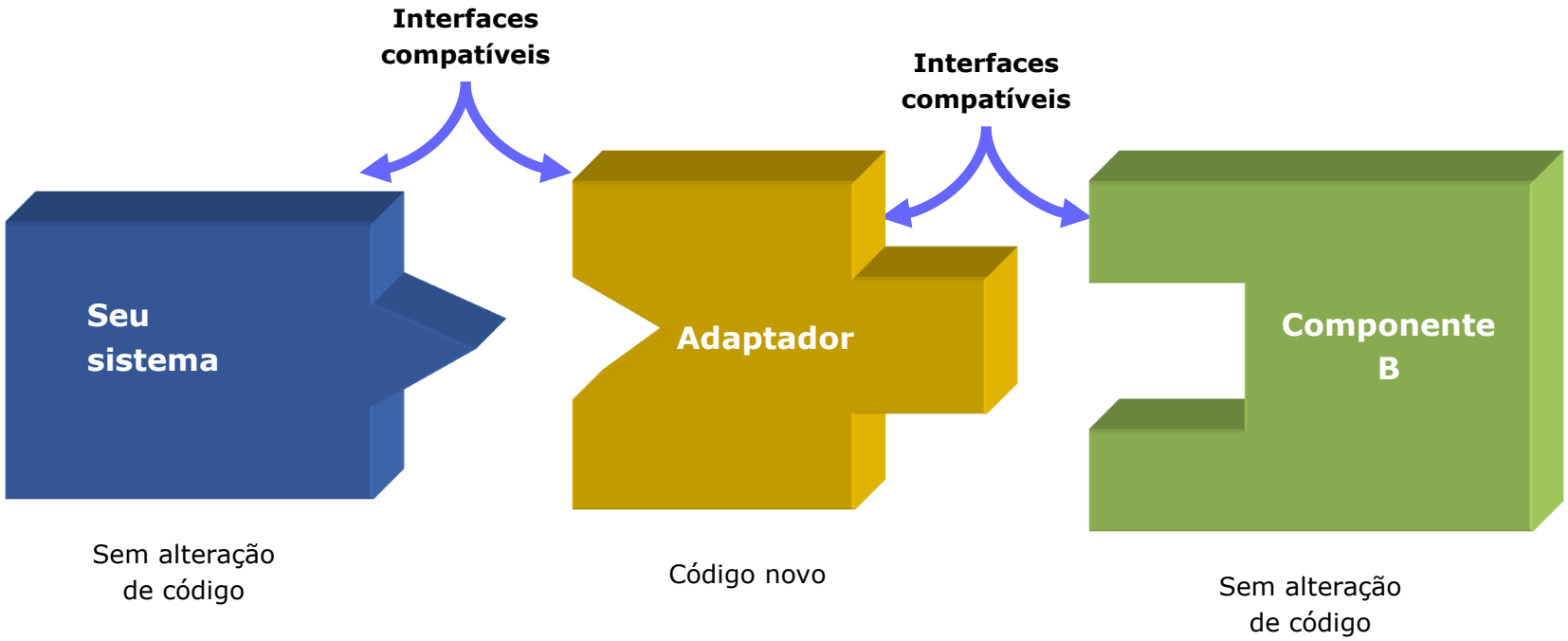
Exemplo: Adapter Pattern

- Porém, o fornecedor do componente B oferece uma interface incompatível com o seu sistema



Exemplo: Adapter Pattern

- Para não correr riscos, você cria um adaptador



Adapter Pattern

- Problema
  - Como adaptar a interface de dois componentes?
- Solução
  - Adapter Pattern (Padrão Adapter)
- “Converte a interface para outra interface que o cliente espera encontrar. O adaptador permite que componentes com interfaces incompatíveis trabalhem juntos”

Fontes de Consulta

- https://www.youtube.com/watch?v=Y69BsV9-23M
- https://refactoring.guru/pt-br/design-patterns/adapter
- https://www.baeldung.com/java-adapter-pattern

Acesse os endereços e veja mais detalhes sobre o padrão Adapter