Padrão de Projeto Strategy

Strategy

- Propósito
  - Definir uma família de algoritmos, encapsular cada um, e fazê-los intercambiáveis.
  - Permitir que algoritmos variem independentemente entre clientes que os utilizam.
  -
- Também conhecido como:
  - Policy

Strategy – Exemplo de Motivação

- Existem muitos algoritmos para quebrar em linhas um texto a ser visualizado.
- Implementar tais algoritmos dentro das classes que os usam não é desejável por diversas razões:
  - Tais classes ficarão mais complexas, maiores e mais difíceis de manter, especialmente se tiverem vários algoritmos de quebrar linhas.
  - Diferentes algoritmos são apropriados para diferentes aplicações. Não queremos todos sempre.
  - Dificulta adição de novos algoritmos e modificação dos existentes.

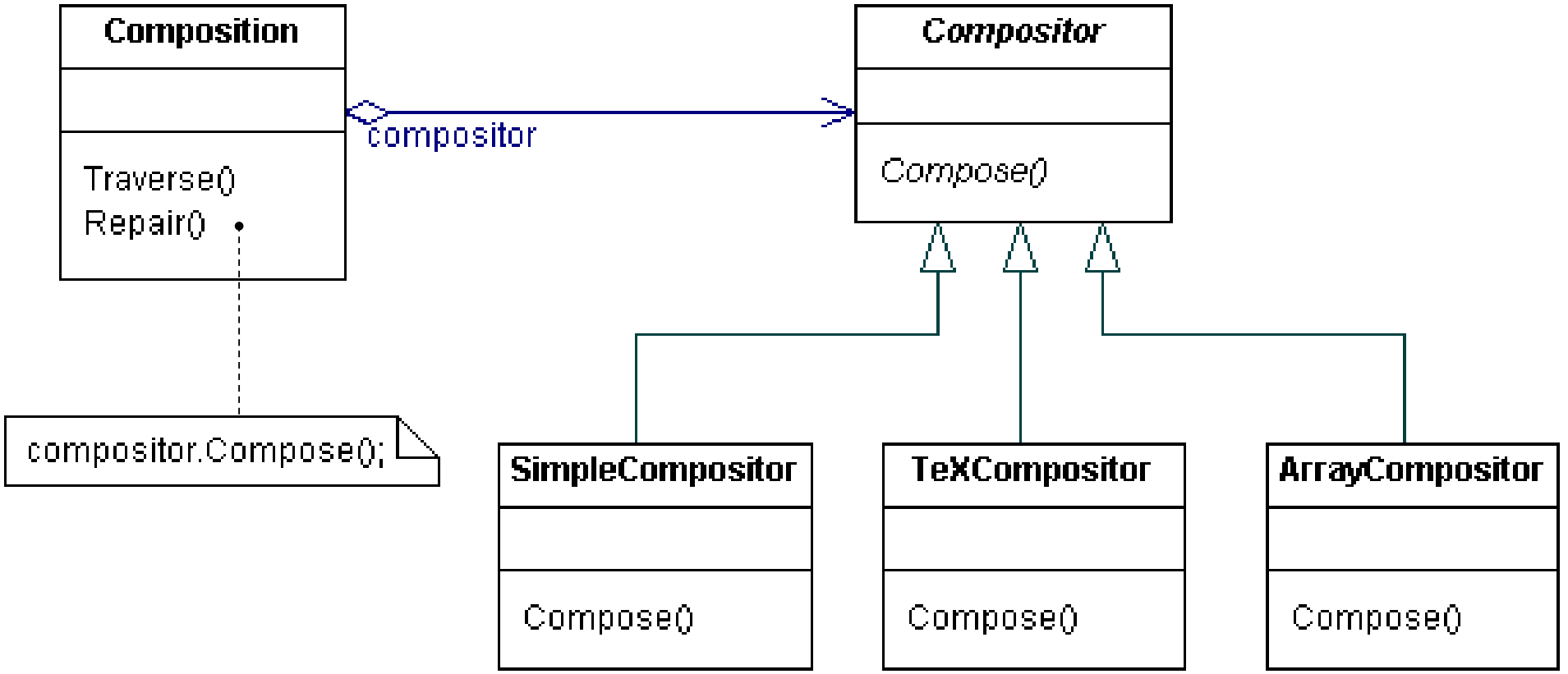
Strategy - Exemplo de Motivação

- Podemos evitar esses problemas definindo classes que encapsulam diferentes algoritmos de quebra de linha.
- Um algoritmo encapsulado dessa forma é chamado de **estratégia**.
  - Podemos ter várias **estratégias** de quebra de linha.

Strategy - Exemplo de Motivação

- Suponha uma classe Composition que é responsável por manter e atualizar as quebras de linhas de um texto apresentado em um visualizador de texto.
- Os algoritmos de quebra de linha não são implementados pela classe.
- Os algoritmos são implementados por subclasses da classe abstrata ou interface Compositor.

Strategy - Exemplo de Motivação



Strategy - Exemplo de Motivação

- Nesse caso, as subclasses de Compositor implementam as seguintes estratégias:
  - SimpleCompositor implementa um algoritmo simples que quebra uma linha por vez.
  - TeXCompositor implementa um algoritmo que tenta otimizar o processo, quebrando linhas de cada parágrafo por vez.
  - ArrayCompositor implementa um algoritmo de forma que todas as linhas tenham um número fixo de itens.

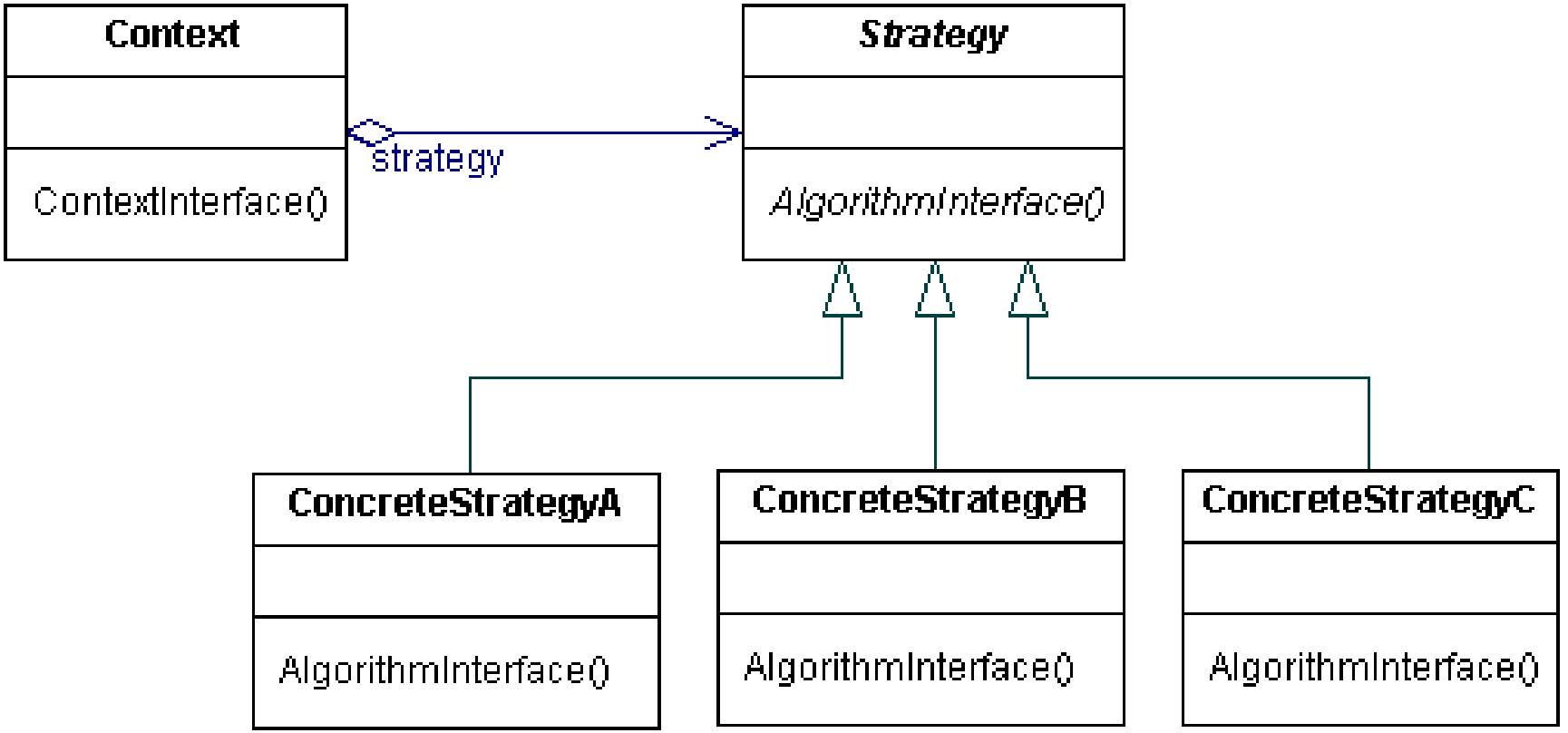
Strategy - Exemplo de Motivação

- A classe Composition mantém uma referência para um objeto da classe Compositor.
- Toda vez que Composition reformata o texto, ela delega essa responsabilidade para um objeto da classe Compositor.
- O cliente da classe Composition (quem usa tal classe) diz qual Compositor deve ser usado, configurando a classe Composition com o objeto Compositor desejado.

Strategy - Aplicabilidade

- Use o padrão Strategy quando
  - muitas classes relacionadas diferem entre si apenas por seu comportamento.
  - você precisa de diferentes variantes de um algoritmo.
  - um classe define muitos comportamentos, e eles aparecem em suas operações como múltiplas estruturas de condição (if, case).

Strategy - Estrutura



Strategy - Participantes

- Strategy (Compositor)
  - Define uma interface comum para todos os algoritmos suportados. Context usa esta interface para chamar o algoritmo definido por uma ConcreteStrategy.
- ConcreteStrategy (SimpleCompositor, TeXCompositor, ArrayCompositor)
  - Implementa o algoritmo usando a interface de Strategy.
- Context (Composition)
  - É configurado com um objeto ConcreteStrategy;
  - Mantém uma referência para um objeto Strategy;
  - Pode definir uma interface que permite a Strategy acessar seus dados.

Strategy - Colaborações

- A classe Context delega as requisições de seus clientes para seu objeto Strategy.
- Clientes geralmente criam e passam um objeto de ConcreteStrategy para Context.
- Context pode passar (como argumento) todos os dados necessários para o algoritmo quando métodos de Strategy são chamados.
- Alternativamente, o objeto Context pode se passar como argumento. Isso permite que Strategy chame métodos de Context quando necessário.

Strategy - Consequências

- Vantagens
  - Uma alternativa à herança
    - Subclasses da classe contexto
  - Eliminação de estruturas condicionais (ifs, case)
- Desvantagens
  - Clientes tem que saber da existência de várias estratégias.
  - Overhead de comunicação entre Context e Strategy.

Strategy - Exemplo