Padrões de Projeto de Software Orientado a Objetos

ACH 2006 – Engenharia de Sistemas de Informação I

Marcos L. Chaim

Padrões estruturais

- Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores.
- Padrões estruturais do GoF:
  - Adapter, Bridge, ***Composite***, Decorator, ***Façade***, Flyweight, Proxy.

Composite (160) – Intenção

- Compor objetos em estruturas (árvores) para representarem hierarquias parte-todo. *Composite* permite aos clientes tratarem de maneira uniforme objetos individuais e composição de objetos.

Composite (160) – Motivação

- Aplicações gráficas, tais como editores de desenhos, permitem aos usuários construir diagramas complexos a partir de componentes simples.
- É desejável que o código do cliente consiga manipular de maneira uniforme tanto componentes simples como componentes complexos.

Composite (160) – Motivação

- Ver figura página 160 do GoF.

- O padrão *Composite* descreve como usar a composição recursiva de maneira que os clientes não tenham que fazer esta distinção.
- A chave do padrão Composite é uma classe abstrata que representa tanto primitivas como os seus recipientes.

Composite (160) – Aplicabilidade

- Use o padrão Composite quando:
  - Você quiser representar hierarquias parte-todo de objetos;
  - Você quiser que os clientes sejam capazes de ignorar a diferença entre composição de objetos e objetos individuais. Os clientes tratarão todos os objetos na estrutura composta de maneira uniforme.

Composite (160) – Estrutura

- Ver figura situada na parte de baixo da página 161.

Composite (160) – Participantes

- Componentes (Graphic)
  - Declara a interface para os objetos na composição;
  - Implementa comportamento *default* para a interface comum a todas as classes;
  - Declara uma interface para gerenciar os seus componentes-filhos;
  - (opcional) Define uma interface para acessar o pai de um componente na estrutura recursiva e a implementa.

Composite (160) – Participantes

- Folha (Retangle, Line, Text etc.)
  - Representa objetos-folha sem filhos;
  - Define comportamento para objetos primitivos na composição.
- Composite (Picture)
  - Define comportamento para componentes que possuem filhos.
  - Armazena os componentes-filhos.
- Cliente
  - Manipula objetos na composição através da interface de *Component*.

Composite (160) – Colaborações

- Os clientes usam a interface da classe Component para interagir com objetos na estrutura composta.
- Se o receptor é uma folha (*leaf*), então a solicitação é tratada diretamente. Se o receptor é um *Composite*, então a solicitação é repassada para os filhjos.

Composite (160) – Conseqüências

- O padrão Composite:
  - Define hierarquias de classe que consistem de objetos primitivos e objetos compostos.
  - Torna o cliente mais simples.
  - Torna mais fácil acrescentar novos tipos de componentes.
  - Pode tornar o projeto excessivamente genério.

Composite (160) – Exemplo

- Arquivos e diretórios
  - Um diretorio é composto de arquivos?
  - Um diretório pode ter entre seus elementos diretório?
- Então podemos usar Composite (160)?
- Ver exemplo.

Façade (160) – Intenção

- Fornecer uma interface unificada para um conjunto de interfaces em um subsistema.
- Façade define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

Façade (160) – Motivação

- Estruturar um sistema em subsistemas ajuda a reduzir a complexidade.
- Um objetivo comum de todos os projetos é minimizar a comunicação e as dependências entre subsistemas.
- Uma maneira de atingir este objetivo é introduzir um objeto ***façade*** (fachada).

Façade (160) – Motivação

- Exemplo não relacionado com software:
  - Serviço de atendimento ao cliente:
    - A única entrada para obter serviços;
    - É composto de subsistemas como:
      - Preenchimento de pedido;
      - Envio;
      - Faturamento.
  - O serviço de atendimento é a façade!

Façade (160) – Aplicabilidade

- Use o padrão Façade quando:
  - Você desejar fornecer uma interface simples para um sistema complexo.
  - Existirem muitas dependências entre os clientes e as classes de implementação de uma abstração. Ao introduzir uma fachada para desacoplar o subsistema dos clientes e de outros subsistemas, está-se promovendo a independência e portabilidade de sistemas.

Façade (160) – Aplicabilidade

- Use o padrão Façade quando:
  - Você desejar estruturar em camadas seus subsistemas. Use uma façade para definir o ponto de entrada para cada nível de subsistema.

Façade (160) – Participantes

- Façade
  - Conhece quais classes do subsistema responsáveis pelo atendimento de uma solicitação.
  - Delega solicitações de clientes a objetos apropriados do subsistema.

Façade (160) – Participantes

- Classes de subsistema:
  - Implementam a funcionalidade do subsistema;
  - Encarregam-se do trabalho atribuído a elas pelo objeto Façade.
  - Não têm conhecimento da fachada, isto é, elas não mantém referências para ela.

Façade (160) – Colaborações

- Os clientes comunicam-se com um subsistema através do envio de solicitações para Façade que as repassa para o(s) objeto(s) apropriados do subsistema. A façade pode ter que efetuar trabalho próprio dela para traduzir a sua interface para interface dos subsistemas.
- Os clientes que usam a façade não têm que acessar os objetos do subsistema diretamente.

Façade (160) – Conseqüências

- O padrão façade oferece os seguintes benefícios:
  - Ele isola os clientes dos componentes do subsistema, desta forma reduzindo o número de objetos com que os clientes têm que lidar e tornando o sistema mais fácil de usar.
  - Promove um aclopamento fraco entre o subsistema e seus clientes.

Façade (160) – Conseqüências

- O padrão façade oferece os seguintes benefícios:
  - A façade não impede as aplicações de utilizarem as classes do subsistema caso necessitem fazê-lo.

Façade (160) – Conseqüências

- O padrão façade oferece os seguintes benefícios:
  - A façade não impede as aplicações de utilizarem as classes do subsistema caso necessitem fazê-lo.

Padrões comportamentais

- Os padrões coportamentais se preocupam com algoritmos e a atribuição de responsabilidade entre os objetos.
- Os padrões comportamentais não descrevem apenas padrões de objetos ou classes, mas também padrões de comunicação entre eles.

Padrões comportamentais

- Os padrões coportamentais se preocupam com algoritmos e a atribuição de responsabilidade entre os objetos.
- Os padrões comportamentais não descrevem apenas padrões de objetos ou classes, mas também padrões de comunicação entre eles.

Strategy (292) – Intenção

- Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis.
- Strategy permite que o algoritmo varie independemente dos clientes que o utilizam.

Strategy (292) – Conhecido como

- Também conhecido como:
  - Policy

Strategy (292) – Motivação

- Existem muitos algoritmos para quebrar um *stream* texto em linhas.
- Codificar de maneira fixa e rígida tais algoritmos nas classes que os necessitam não é desejável por várias razões:
  - Clientes que necessitam de quebras de linhas se tornam mais complexos se incluírem o código para quebra de linhas.

Strategy (292) – Motivação

- Codificar de maneira fixa e rígida tais algoritmos nas classes que os necessitam não é desejável por várias razões:
  - Diferentes algoritmos serão apropriados em diferentes situações.
  - Difícil adicionar novos algoritmos e variar os existentes quando a quebra de linha é parte integrante de um cliente.

Strategy (292) – Motivação

- Podemos evitar estes problemas definindo classes que encapsulam diferentes algoritmos de quebra de linha.
- Um algoritmo encapsulado desta maneira é chamado ***strategy***.

Strategy (292) – Aplicabilidade

- Use o padrão strategy quando:
  - Muitas classes relacionadas diferem somente no seu comportamento.
  - Voce necessita de variantes de um algoritmo.
  - Um algoritmo usa dados de que os clientes não deveriam ter conhecimento. Use o padrão Strategy para evitar a exposição das estruturas de dados complexas, específicas do algoritmo

Strategy (292) – Estrutura

- Ver figura da página 294.

Strategy (292) – Participantes

- Strategy
  - Define uma interface comum para todos os algoritmos suportados. Context usa esta interface para chamar o algoritmo definido por uma ConcreteStrategy.
- ConcreteStrategy
  - Implementa o algoritmo usando a interface de Strategy.

Strategy (292) – Participantes

- Context
  - É configurado com um objeto ConcreteStrategy;
  - Mantém uma referência para um objeto Strategy.
  - Pode definir uma itnerface que permite a Strategy acessar os seus dados.

Strategy (292) – Colaborações

- Strategy e Context interagem para implementar o algoritmo escolhido.
- Um Context repassa solicitações dos seus clientes para sua estratégia.

Strategy (292) – Conseqüências

- O padrão Strategy tem os seguintes benefícios e vantagens:
  - Famílias de algoritmos relacionados.
  - Uma alternativa ao uso de subclasses.
  - Estratégias eliminam comandos condicionais da linguagem de programação.
  - Possibilidade de escolha da implementação.

Strategy (292) – Conseqüências

- O padrão Strategy tem as seguintes desvantagens:
  - Clientes devem conhecer diferentes estratégias.
  - Custo de comunicação.
  - Aumento no número de objetos.

Strategy (292) – Exemplo

- Sistema de cobrança precisa imprimir datas.
  - Mas as datas variam dependendo do país: EUA, Brasil.
  - Dependendo da extensão: só número, mês escrito, etc.
  - Exemplos: October 2, 2005 2 Oct 2005 10/2/05 02/10/05 20051002 2005275
- Dá para utilizar Strategy?

Resumo

- Padrões estruturais:
  - Composite e façade.
- Padrões comportamentais:
  - Strategy e Visitor.

Referências

- Duel, Michael -- “Non-Software Examples of Software Design Patterns”, http://wwwswt.informatik.uni-rostock.de/deutsch/Lehre/Uebung/Beispiele/PatternExamples/patexamples.htm.
- Houston, Vince -- “Design Patterns”, http://www.vincehuston.org/dp/.
- E. Gamma, R. Helm, R. Johnson e J. Vlissides. *Padrões de projetos: soluções reusáveis de software orientado a objetos*. Porto Alegre: Bookman, 2000.