Padrão Composite:

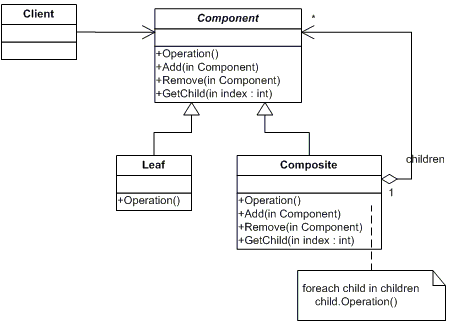
O padrão Composite é frequentemente usado com o Iterator para percorrer a estrutura semelhante a uma árvore oferecida pelo primeiro. Vamos imaginar que existe uma classe abstrata Component com vários métodos. Um deles é usado para imprimir toda a estrutura. Este método print() é herdado pela planilha e pelo Composite.

Objetivo: Um padrão é normalmente usado para representar listas de elementos recorrentes ou recursivas. Usado para representar um objeto formado pela composição de objetos semelhantes. Este conjunto de objetos pressupõe a mesma hierarquia de classes a que pertence. Esse padrão geralmente é usado para mostrar uma lista de objetos.

A intenção do padrão Composite é compor objetos em estruturas de árvore para representar hierarquia partes-todo.

Aplicabilidade: O padrão Composite pode ser utilizado quando queremos trabalhar com uma hierarquia de objetos que representam a mesma superclasse. Ele é usado em combinação com o padrão Iterator para criar um sistema de gerenciamento de arquivos. Através do padrão Composite, todos terão uma forma comum de serem reconhecidos e trabalhados.

Estrutura: A estrutura UML do padrão, descrita no livro GoF, é a seguinte. Folha representa o objeto folha na composição. A folha não possui nenhum componente filho, ela define o comportamento de objetos primitivos. O Cliente opera os objetos de composição através da interface do Componente.



Consequências: objetos complexos podem ser compostos de objetos mais simples recursivamente. O cliente pode tratar objetos simples ou compostos uniformemente desde que uma interface comum (Componente) seja definida para ambos. Também facilita a adição de novos componentes. Mas por outro lado, seu projeto de aplicativo acaba sendo muito ‘generalizado’.

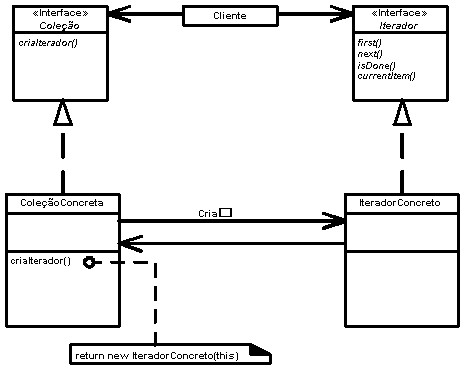
Padrão Iterator:

Objetivo: Prover uma forma de sequencialmente acessar os elementos de uma coleção sem expor sua representação interna.

Queremos isolar o uso de uma estrutura de dados de sua representação interna de forma a poder mudar a estrutura sem afetar quem a usa Para determinadas estruturas, pode haver formas diferentes de caminhamento ("traversal") e queremos encapsular a forma exata de caminhamento

Exemplo: árvore pode ser varrida "em ordem", em "pós-ordem", em "pré-ordem" Exemplo: podemos ter um "iterador com filtro" que só retorna certos elementos da coleção

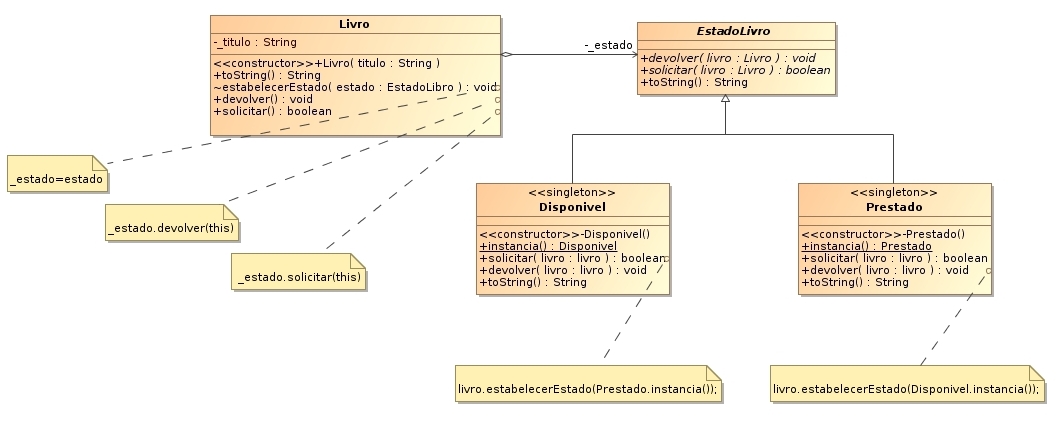
Exemplo: num editor de documentos, poderíamos ter os elementos do documento organizados em árvores (documento consiste de páginas que consistem de parágrafos, ...) e ter um iterador especial para elementos não gráficos (que podem ter sua grafia verificada, por exemplo) A idéia do iterador é de retirar da coleção a responsabilidade de acessar e caminhar na estrutura e colocar a responsabilidade num novo objeto separado chamado um iterador



Padrão State

Objetivo: Permite que um objeto altere seu comportamento de acordo com o estado interno que se encontra em um momento dado.

Motivação: É motivado por aqueles objetos que, em seu estado atual, variam seu comportamento devido às diferentes mensagens que podem receber. Como exemplo, pegamos uma classe Livro, um objeto desta classe terá diferentes respostas, dependendo do seu estado (Disponível ou Emprestado). Por exemplo, invocando o método book, seu comportamento será diferente, se o Book estiver no estado Disponível ou no estado Empréstimo



Vantagens e desvantagens:

* É fácil de localizar as responsabilidades de estados específicos, devido a que os encontram nas classes que correspondem a cada estado. Isto proporciona uma maior clareza no desenvolvimento e na manutenção subsequente. Esta facilidade é fornecida pelo fato de que diferentes estados são representados por um único atributo (estado) e não envolvidos em diferentes variáveis ​​e grandes condicionais.
* Faz as mudanças de estado explícitas, posto que em outro tipo de implantação os estados são alterados, modificando os valores em variáveis, enquanto aqui fazer-se representar cada estado.
* Os objetos Estado podem ser compartilhados se eles não contêm variáveis ​​de instância, isto pode ser alcançado se o estado está totalmente codificado representando seu tipo. Quando isso é feito, os estados são flyweights sem estado intrínseco.
* Facilita a expansão de estados.
* Permite a um objeto alterar de classe em tempo de execução dado que ao modificar suas responsabilidades pela de outro objeto de outra classe, a herança e responsabilidades do primeiro mudaram pelas do segundo.

Desvantagem:

* Aumenta o número de subclasses.

Padão Proxy

Um proxy é um objeto ou agente wrapper que está sendo chamado pelo cliente para acessar o objeto de serviço real, nos bastidores. O uso do proxy pode simplesmente ser encaminhado para o objeto real ou pode fornecer lógica adicional. No proxy, funcionalidade extra pode ser fornecida, por exemplo, armazenamento em cache quando as operações no objeto real consomem muitos recursos.

* Prover um substituto (ou *placeholder*) para um outro objeto controlar seu acesso;
* Usar um nível extra de indireção para fornecer acesso distribuído, controlado ou inteligente;
* Adicionar um agregador e delegador para proteger o componente real de complexidade indevida.

Discussão:

Desenvolver um substituto, ou proxy, objeto que: instancia o objeto real na primeira vez que o cliente faz uma solicitação de proxy, lembra a identidade desse objeto real, e então redireciona a requisição para o objeto real. Então todas as requisições subsequentes são simplesmente redirecionadas diretamente para o objeto real encapsulado.

Essas são as quatro situações mais comuns em que o padrão Proxy é aplicável.

Um proxy virtual é um espaço reservado para objetos cuja criação é cara. O objeto real só é criado quando um cliente solicita ou acessa o objeto. Um proxy remoto fornece acesso a um objeto que reside em um espaço de endereço diferente. Um proxy protetor controla o acesso a um objeto de destino.

