**Padrão de Projeto Estrutural - O padrão Decorator**

Os padrões estruturais preocupam-se com a forma como classes e objetos são compostos para formar estruturas maiores. O Decorator é usado quando se necessita adicionar novas funcionalidades a um objeto. Em Java, por exemplo, utiliza-se muito o padrão Decorator nos objetos de entrada e saída de dados (E/S). O Exemplo contém trechos de um código Java que realizada leitura de um arquivo no qual a classe Reader, responsável pela leitura caracter a caracter do arquivo, recebe a funcionalidade de ler o arquivo linha a linha pela classe LineNumberReader.

public static void main(String[ ] args) {

try {

File file = new File("arquivoTexto.txt");

Reader in = new FileReader(file);

LineNumberReader reader = new LineNumberReader(in);

while (reader.ready()) {

System.out.print("Linha " + reader.getLineNumber() + " --> ");

System.out.println(reader.readLine());

}

reader.close();

in.close();

} catch (FileNotFoundException e) {

System.out.println("Arquivo não encontrado. Causa: " + e.getMessage());

} catch (IOException e) {

System.out.println("Ocorreu um erro de E/S de dados. Causa: " + e.getMessage());

Exemplo –Padrão Decorator

Fonte TERRA, 2008

***Objetivo***

* Adicionar responsabilidades dinamicamente a um objeto
* Decoradores provêm uma alternativa flexível à herança para estender funcionalidade
* Permitem adicionar responsabilidades a um objeto e não a uma classe inteira

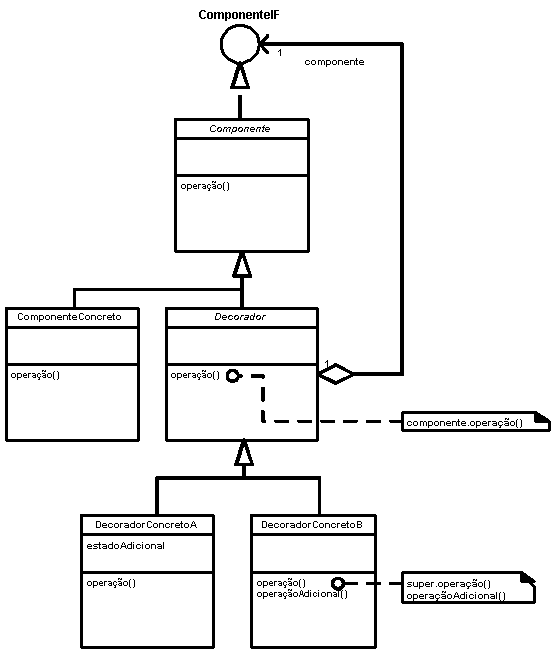
***Também chamado de***

* Wrapper

***Quando usar o padrão Decorator?***

* Para adicionar responsabilidades dinamicamente a objetos individuais e transparentemente (sem afetar outros objetos)
* Quando há responsabilidades que podem ser retiradas
* Quando a herança geraria uma explosão de subclasses
* Quando a herança seria uma boa alternativa, mas a definição da classe está escondida ou não disponível para herança.

***Estrutura genérica***



***Participantes***

* ComponenteIF (ex. ElementoDeDocumentoIF)

Define a interface de objetos que podem ter responsabilidades adicionadas dinamicamente

* Componente (ex. ElementoDeDocumento)

Possível classe abstrata para fatorar o código comum dos ComponenteConcreto

* ComponenteConcreto (ex. ElementosAFormatar)

Define um objeto ao qual se pode adicionar responsabilidades adicionais

* Decorador (ex. MonoElementoDeDocumento)

Mantém uma referência a um objeto ComponenteIF e define uma interface igual à do Componente

* DecoradorConcreto (ex. Borda, Rolador)

Adiciona responsabilidades ao Componente

***Colaborações entre objetos***

* O Decorador encaminha pedidos ao Componente
* Pode opcionalmente adicionar operações antes ou depois deste encaminhamento

***Conseqüências do uso do padrão Decorator***

* Mais flexível do que herança estática

Pode adicionar responsabilidades dinamicamente

Pode até adicionar responsabilidades duas vezes! (ex. borda dupla)

* Evita classes com features demais no topo da hierarquia

Em vez de tentar prever todos os features numa classe complexa e customizável, permite definir classes simples e adicionar funcionalidade de forma incremental com objetos decoradores

Desta forma, as aplicações não têm que pagar pelos features que não usam

* Ponto negativo: o Decorador e o objeto incluso não têm a mesma identidade

Portanto, cuidado com a identidade de objetos: um Decorador adicionado a um objeto vai mudar o objeto com o qual o cliente lida (em termos de identidade)

* Ponto negativo: muitos pequenos objetos

Fácil de customizar por programadores que entendem o projeto mas pode ser difícil entender e depurar

***Considerações de implementação***

* A interface do decorador deve ser igual à interface do objeto sendo decorado
* Se tiver que adicionar apenas uma única responsabilidade, pode-se remover o decorador abstrato

Neste caso, o decorador concreto pode tratar do forwarding para o Componente incluso

* Mantendo Componentes enxutos

A classe Componente deve ser mantida enxuta e os dados serem definidos nos Componentes concretos

Caso contrário, os decoradores (que herdam de Componente) ficam "pesados".