Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Modelagem e Especificação de Software
Professor: Jair Cavalcanti Leite

Software compactador Press4All: Reengenharia e inserção de padrões de projeto.

Alunos: Bráulio Bezerra da Silva
Clarissa Araújo Azevedo
Tyago de Medeiros Silva

Natal, 18 de outubro de 2007.

[Introdução]

O software foco deste projeto consiste da implementação de um compactador de dados. A compactação de dados é realizada por um subsistema que se baseia no clássico algoritmo de Huffman. Os dados são comprimidos pelos mais diversos motivos. Entre os mais conhecidos estão economizar espaço em dispositivos de armazenamento, como discos rígidos, ou ganhar desempenho (diminuir tempo) em transmissões.

O desenvolvimento de software é um trabalho complexo, mas agregar valores, como reusabilidade, torna o processo mais difícil ainda. Para tornar esse processo menos árduo, podemos nos fundamentar na experiência coletiva de desenvolvedores de software para encontrar soluções para problemas comuns. Esta solução expressa em termos de objetos e interfaces é chamada padrão.

Um padrão de projeto nomeia, abstrai e identifica os aspectos-chave de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável. O padrão de projeto identifica as classes e instâncias participantes, seus papéis, colaborações e a distribuição de responsabilidades. Cada padrão de projeto focaliza um problema ou tópico particular de projeto orientado a objetos. Ele descreve em que situação pode ser aplicado, se ele pode ser aplicado em função de outras restrições de projeto e as conseqüências, custos e benefícios de sua utilização.

O padrão *Singleton* objetiva garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma, tornando a própria classe responsável pelo controle de sua única instância. O padrão *Façade* fornece uma interface unificada para um conjunto de interfaces em um subsistema, definindo uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

[Inserção do padrão Singleton]

O padrão singleton foi adotado na re-implementação das classes 'compactador' e 'descompactador' para garantir a existência de somente uma instância das mesmas. Na versão anterior, cada vez que se queria compactar ou descompactar um arquivo, era necessária a instância de um novo objeto para realizar tal operação. O software tinha assim trabalho extra pois era necessário instanciar um novo objeto a cada ação e este era perdido após o término da operação. Com a adoção do padrão singleton ao projeto, existe somente uma única instância de cada uma dessas classes e existe um único ponto de acesso a cada uma.

Classes 'compactador' e 'descompactador':

As mudanças nas duas classes são semelhantes. Em ambos os casos, podia ocorrer a instanciação de vários objetos da mesma classe, o que realmente ocorria. O padrão singleton vêm garantir a existência de somente uma instância de cada classe e para isso encarrega as próprias classes dessa tarefa.

Novo comportamento das classes, baseado no padrão singleton:

```
public class compactador
{
        ...

 private static compactador instancia = null;
 public static compactador getInstancia( )
 {
        if(instancia==null)
                instancia = new compactador( );
        return instancia;
 }
 protected compactador( ){ ... }

        ...

}
```

```
public class descompactador
{
        ...

 private static descompactador instancia = null;
 public static descompactador getInstancia()
 {
        if(instancia==null)
                instancia = new descompactador();
        return instancia;
 }
 protected descompactador( ){ ... }

        ...

}
```

[Inserção do padrão Façade]

O objetivo do software é possibilitar compactar e descompactar qualquer arquivo digital, indiferentemente do algoritmo que seja utilizado. Para separar o sistema de classes clientes (GUI) do sub-sistema que implementa a compactação através do algoritmo de huffman, foi implementada uma fachada para que o cliente acessasse o sub-sistema através de uma interface única. A adoção desse padrão agrega valor no momento em que minimiza a comunicação e a dependência entre os sub-sistemas distintos, agregando também maior reusabilidade a ambos.

Classes 'codecFacade':
Ao invés do subsistema cliente (GUI) precisar acessar as classes de compactação, descompactação e o método de comparação de arquivos, foi implementada a classe 'codecFacade' para servir de fachada e diminuir a dependência e servir de ponto único de comunicação entre os sub-sistemas.

Classe implementada para servir de fachada de acesso à implementação do sub-sistema de compactação/descompactação:

```
import huffman.compactador;
import huffman.descompactador;
        ...

public class codecFacade
{
 protected codecFacade( ){ }

 public static void compactar(File fIn,File fOut,JTextArea aTextArea){ ... }

 public static void descompactar(File fIn,File fOut,JTextArea aTextArea){ ... }

 public static boolean equals(File f1, File f2){ ... }
}
```

Diagrama de representação da fachada:

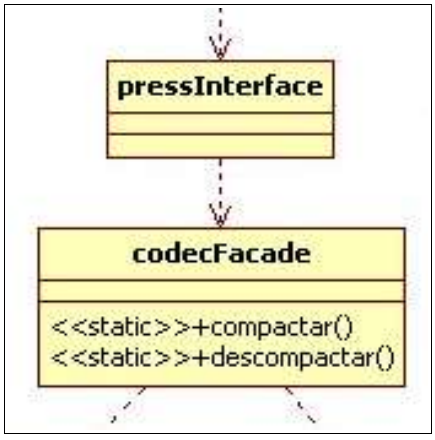
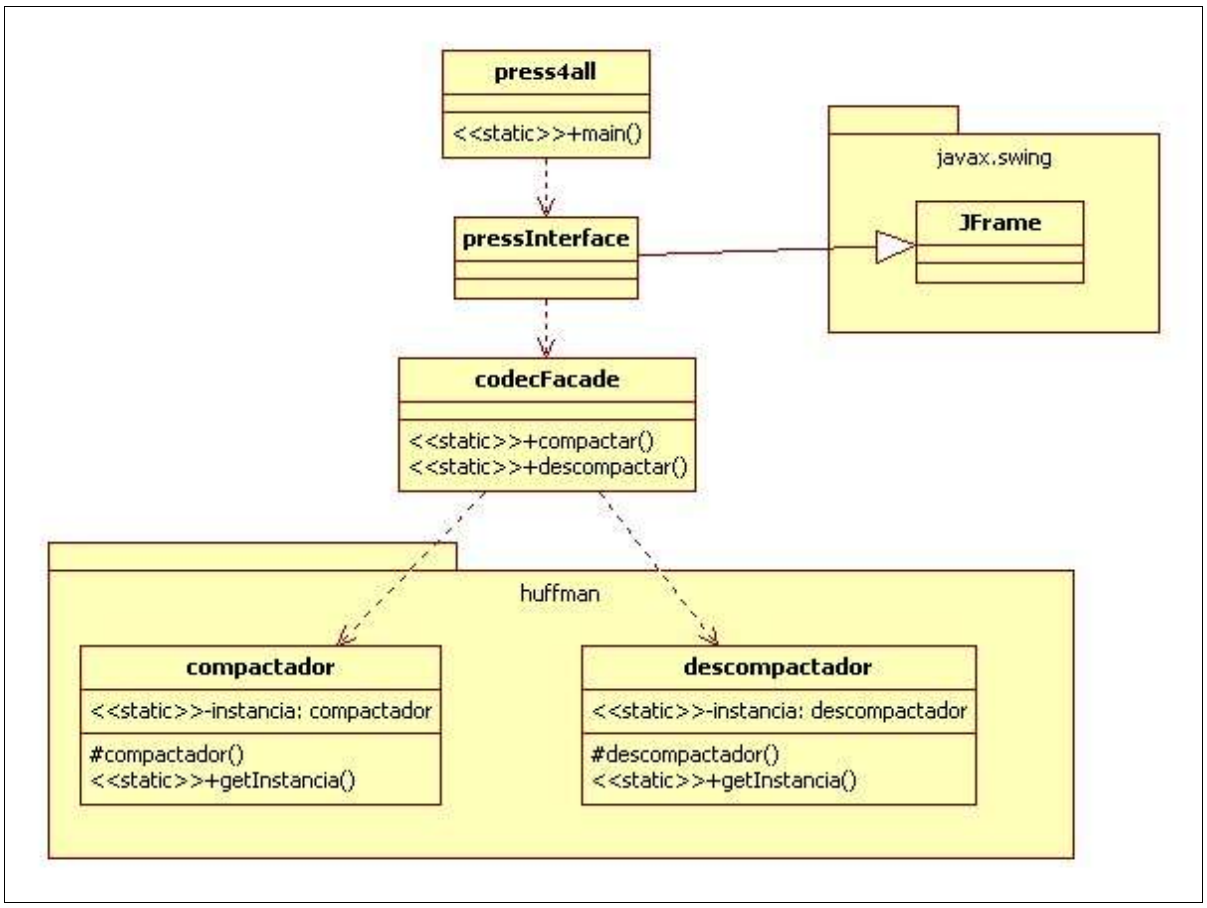


Diagrama de classes na nova versão implementada:



[Conclusão]

Com a introdução dos padrões de projeto, é possível solucionar problemas que inicialmente podem ser imperceptíveis, mas que mais tarde podem ficar maiores. Tornando também o código muito mais organizado. Além disso, foi notável a melhora no desempenho e reusabilidade do software.

[Observações]

O código das versões com e sem inserção de projetos, além de documentação adicional, está disponível no endereço: *http://consiste.dimap.ufrn.br/~tyago/Compactador/*