Universidade de São Paulo
Instituto de Matemática e Estatística
MAC 413 – Tópicos de Programação Orientada a Objetos

Padrão

4-ação

(lê-se *quatração*)

Danilo M. Rubio Conde - 3116832
Emilio de C. Francesquini - 3114889

Outubro / 2002

Intenção

Fornecer um método de desenvolvimento de software onde o acoplamento entre os componentes do sistema seja suficientemente baixo de forma que seja possível a troca de componentes com pouquíssimo esforço e trabalho para adaptação do sistema.

Também conhecido como

Separação de especificação, implementação e configuração, que é da onde se origina o nome 4-ação.

Motivação

Imagine que o fabricante de placas-mães MãeDigital desenvolveu um novo e sofisticado modelo de placa-mãe, o EndoesqueletoDigital. Durante seu desenvolvimento, foram utilizadas memórias apenas da marca ElefanteDigital. Como as memórias ElefanteDigital eram de ótima qualidade e possuíam instruções especiais que aumentavam o desempenho do sistema, os pesquisadores da MãeDigital basearam a implementação da BIOS da nova placa-mãe nessas instruções especiais. Pela sofisticação e inovações deste projeto, seu custo foi muito elevado, e se não as novas placas não fossem um sucesso, isso provavelmente seria o fim da MãeDigital.

Com a prolongada seca que vem atingindo a África nesta última década, uma crise sem precedentes ocorreu. A ElefanteDigital, cuja sede ficava em Serra Leoa, acabou falindo devido a crise. A mãe MãeDigital, que não era sediada na África, estava planejando o lançamento do EndoesqueletoDigital quando recebeu a notícia da extinção da ElefanteDigital. Foi um imenso choque na empresa.

Em uma reunião de emergência, diretor da MãeDigital chamou os engenheiros que haviam desenvolvido o EndoesqueletoDigital e disse a eles que a placa deveria ser

lançada em duas semanas com os módulos de memória de um novo fornecedor, o ChingLingMing, que, obviamente não possuiam os mesmos instruções especiais das famosas memórias ElefanteDigital. Os engenheiros, responderam friamente: “num dá!”. Os diretor quis saber por quê e os engenheiros lhe explicaram que haveriam de reescrever boa parte da BIOS da placa-mãe, pois precisavam retirar do código as intruções das memórias antigas e utilizar instruções da ChingLingMing. O diretor, então, disse a eles para não utilizarem instruções específicas das placas ChingLingMing, mas para desenvolverem apenas usando as interfaces genéricas, comuns a todos os módulos de memória existentes no mercado, que seguem uma determinada norma.

Após algumas madrugadas de trabalho incessante e muitas horas extras pagas pela MãeDigital, os engenheiros terminaram as adaptações do software para as instruções genéricas a tempo para o lançamento, que foi um grande sucesso.

Alguns meses depois, a ChingLingMing teve o mesmo fim que a ElefanteDigital, devido aos aviões que se chocaram contra sua sede. Dessa vez, o diretor da MãeDigital chamou mais uma vez seus engenheiros e disse a eles que agora usariam memórias DaIskina. Os engenheiros, então, disseram: “belê”, pois sabiam que dessa vez não teriam que reescrever código algum, bastava apenas encaixar as novas memórias na EndoesqueletoDigital que ela funcionaria, pois agora sua BIOS utilizava a interface comum a todas as memórias.

Durante a implementação de um software, é muito comum deixar a implementação de algum módulo diretamente dependente da implementação de outro módulo, ao invés de sua interface, como é sugerido em [1]. Isso conduz a um alto acoplamento entre os módulos, o que, a priori, deve ser evitado.

Programar para uma implementação leva a problemas similares aos que a MãeDigital enfrentou, tais como: aumento do custo do projeto, caso alguma haja necessidade de trocar algum componente, falta de agilidade na adaptação do sistema, entre outros.

Quando os engenheiros resolveram reescrever a BIOS baseando-se apenas nas interfaces comuns a todas as memórias, reduzindo assim o acoplamento, a troca das memórias por outra marca não representou um grande problema, como havia representado anteriormente.

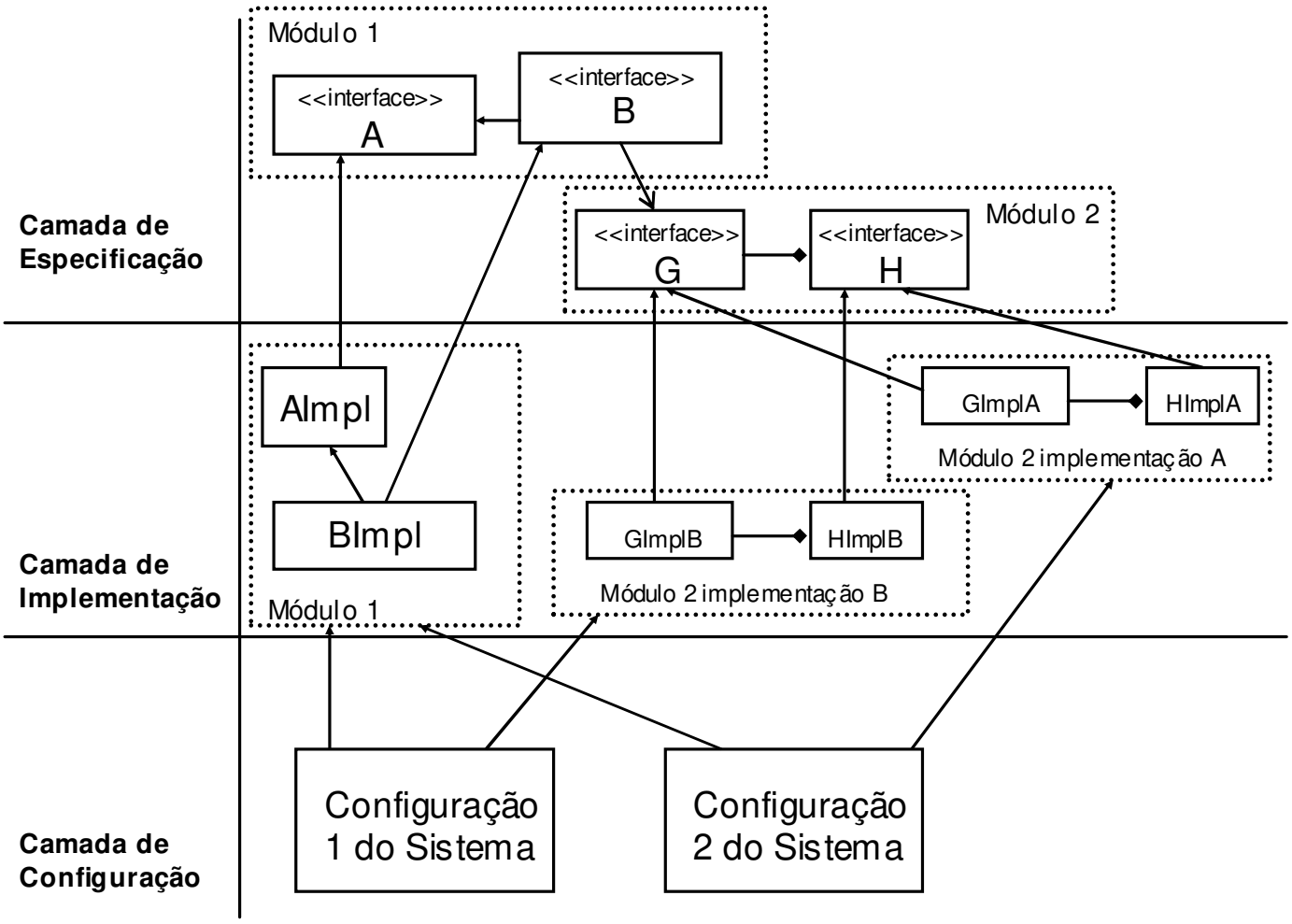
Aplicabilidade

Use este padrão:

- para que os componentes do sistema possam ser facilmente trocados por outros que tenham a mesma interface
- para que exista agilidade em tal troca
- para que haja legibilidade do código final, com módulos com interfaces e funções muito bem definidas

Estrutura

A estrutura de um sistema que foi construido utilizando este padrão deve ser como o diagrama de exemplo abaixo:



Existem 3 camadas: especificação, implementação e configuração.

A camada de especificação é onde devem ser criadas todas as interfaces que o sistema utilizará e como essas interfaces interagem umas com as outras. Na camada de implementação devem ser criadas as classes concretas que implementam as interfaces da camada de especificação. São permitidas apenas relacionamentos apenas entre classes de um mesmo módulo. Relacionamentos entre classes de diferentes módulos são estritamente proibidas, para evitar o acoplamento entre as implementações, deixando apenas existir acoplamento entre as interfaces. A camada de configuração é onde são definidos quais módulos serão usados no sistema para implementar as interfaces definidas na camada de especificação.

No exemplo acima, temos definidas as interfaces A e B no módulo 1, e G e H no módulo 2. Para o módulo 2, há duas implementações diferentes disponíveis. Na camada de configuração há também duas configurações possíveis, uma para cada implementação do módulo 2.

Voltando à história da empresa MãeDigital, fazemos uma analogia do ato de encaixar uma nova memória na placa-mãe à fazer uma troca de configuração. As interfaces que estariam definidas na camada de especificação são as instruções genéricas que todos os módulos de memória implementam, segundo uma determinada norma. Cada marca distinta de memória representa uma implementação diferente na camada de implementação que, por respeitar as interfaces especificadas na camada de especificação, são intercambiáveis, bastando apenas realizar uma troca de configuração para trocá-las.

Participantes

- Camada de especificação
  - especifica todas as interfaces e módulos do sistema
  - especifica como os módulos e interfaces devem interagir
  - especifica o comportamento do sistema
  - define **o que** vai ser feito [2] e os contratos entre as classes
- Camada de implementação
  - contém as classes que implementam as interfaces e módulos do sistema
  - define **como** vai ser feito [2]
- Camada de configuração
  - Define quais implementações contidas na camada de implementação serão usadas

Colaborações

Durante a inicialização do sistema a configuração deve instanciar algumas implementações, que não dependem de nenhuma outra, para passá-las como parâmetro em suas dependentes.

No exemplo do diagrama acima ambas as configurações devem antes instanciar uma implementação do módulo 2 para passá-la como parâmetro para a instanciação do módulo 1, já que a especificação do módulo 1 depende da especificação do módulo 2.

Conseqüências

Vantagens:

- **baixo** acoplamento entre módulos
- **fácil** reutilização de módulos entre sistemas diferentes, já que os módulos tem sua interface e funcionalidade muito bem definidas na camada de especificação
- troca entre diferentes implementações de módulos do sistema é realizada de forma **fácil**, bastando para isso uma única alteração na camada de configuração (tipicamente uma única classe)
- o entendimento do sistema se dá de uma maneira mais **tranqüila**, em particular, o sistema como um todo pode ser compreendido apenas olhando para a camada de especificação
- avaliações de desemprenho de diferentes implementações de uma dada especificação são feitas de maneira mais **rápida**

- **facilidade** de desenvolvimento distribuído, em diferentes equipes, uma em cada módulo da aplicação por exemplo, após a especificação das interfaces.

Desvantagens:

- o número de classes do sistema mais que dobra. Para qualquer classe de implementação é necessária uma interface para representá-la, assim como uma outra interface para representar sua AbstractFactory e a implementação da mesma
- qualquer mudança simples de assinatura de um método acarreta em mudanças em no mínimo 3 lugares diferentes, sendo elas a interface que a define, sua implementação e seus clientes
- aumento no tempo de análise e desenho do software

Implementação

Seguem abaixo alguns tópicos que devem ser considerados durante a implementação de um sistema que segue este padrão.

O processo de desenvolvimento de tal sistema segue 3 passos bem definidos que são a criação da camada de especificação, a criação da camada de implementação e por fim a criação de uma configuração.

Durante a criação da camada de especificação devem ser considerados:

- análise muito cuidadosa do comportamento que o sistema deve possuir e seu particionamento em módulos com uma função e comportamento muito bem definidos
- após o particionamento definir as interfaces que os módulos componentes do sistema utilizarão para interagir. Note que uma definição não muito boa nesta fase pode trazer sérias conseqüencias durante a implementação das interfaces
- em suma, definir precisamente **o que** o sistema faz

Fase de criação da camada de implementação

- uma classe não pode de forma alguma depender diretamente de classes de que constam nas implementações de outros módulos. Para corrigir isto deve-se apenas utilizar dependencias para as interfaces de tais módulos já definidas na camada de especificação
- aqui deve-se focar na maneira que o sistema irá resolver um dado problema, ou seja, o aspecto algoritmico da solução
- talvez durante o desenvolvimento seja necessário, para testes, o desenvolvimento de *proxies* já que não se pode fazer referência alguma para implementações de outros módulos
- em suma, nesta fase é definido um conjjunto de maneiras, implementações, de **como** o sistema faz suas tarefas

Fase de configuração

- nesta fase são definidas que implementações serão utilizadas pelo sistema. Determinadas implementações são escolhidas baseadas em como efetuam o serviço definido pela camada de especificação. Isso tudo com a garantia de

que o serviço está em conformidade com os contratos estipulados na camada de especificação.

- em suma, é onde é feita a **"ligação"** entre as implementações dos módulos

Exemplo de código

O exemplo de código ilustra a aplicação do padrão para o desenvolvimento de uma aplicação bastante simples, que apenas cria uma lista e a ordena, podendo ser a ordenação por inserção ou usando o quick sort. A escolha é feito trocando-se a configuração, que é escolhida por um parâmetro passado na linha de comando.

Interfaces do sistema (Camada de especificação):

```
package ordena.specs;

public interface Elemento {

    //devolve -1 se este for menor que e, 0 se for igual e 1 caso
contrário
    public int compara (Elemento e);

    //pega o valor real deste elemento
    public Object pegaValor ();

}
```

```
package ordena.specs;

public interface ElementoFactory {
    // instancia e devolve um elemento, usando o parametro n como
valor
    public Elemento criaElemento (int n);
}
```

```
package ordena.specs;


public interface Lista {

    //devolve o núemro de elementos
    public int tamanho ();

    //devolve o elemento da lista na posicao pos. pos deve
     //ser tal que 0 <= pos < tamanho()
    public Elemento elementoEm (int pos);
```

```
    //troca o elemento na posicao pos1 com o elemento na posição
pos2
    // 0 <= pos1 < tamanho() e 0 <= pos2 < tamanho()
    public void troca (int pos1, int pos2);

    //coloca o elemento ele na posicao pos
    public void coloca (Elemento ele, int pos);

    // insere o elemento e no final da lista
    public void adiciona (Elemento e);
}
```

```
package ordena.specs;

public interface ListaFactory {

    // instancia e devolve uma lista
    public Lista criaLista();

}
```

```
package ordena.specs;

public interface Ordenador {

    //ordena a lista lista
    void ordena (Lista lista);

}
```

Implementação das interfaces (Camada de Implementação):

```
package ordena.impl;

import ordena.specs.*;

public class ElementoNumeroInteiroImpl implements Elemento {
    Integer valor;

    public ElementoNumeroInteiroImpl (int n) {
        valor = new Integer (n);
    }

    public int compara (Elemento e) {
        return valor.compareTo(e.pegaValor());
    };

    public Object pegaValor () {
        return valor;
```

```
    };

}
```

```
package ordena.impl;

import ordena.specs.*;

public class ElementoNumeroInteiroImplFactory implements
ElementoFactory {
    public Elemento criaElemento (int n) {
        return new ElementoNumeroInteiroImpl (n);
    }
}
```

```
package ordena.impl;

import ordena.specs.*;
import java.util.Vector;

public class ListaImpl implements Lista {

   Vector elems = new Vector();

    public int tamanho () {
        return elems.size();
    };

    public Elemento elementoEm (int pos) {
        return (Elemento)elems.get(pos);
    };

    public void troca (int pos1, int pos2) {
        Elemento tmp = elementoEm(pos1);
        coloca (elementoEm(pos2), pos1);
        coloca (tmp, pos2);
    };

    public void coloca (Elemento ele, int pos) {
        elems.set(pos, ele);
    }

    public void adiciona (Elemento e) {
        elems.add(e);
    }

};
```

```
package ordena.impl;
```

```
import ordena.specs.*;

public class ListaImplFactory implements ListaFactory{

    public Lista criaLista () {
        return new ListaImpl();
    }
}
```

```
package ordena.impl;

import ordena.specs.*;

public class OrdenadorPorInsercaoImpl implements Ordenador {

    public void ordena (Lista lista) {
        int n = lista.tamanho();
        int j, i;
        Elemento x;
        for (j = 1; j < n; j++) {
            x = lista.elementoEm(j);
            for    (i    =    j    -    1;    i    >=    0    &&
lista.elementoEm(i).compara(x) > 0; --i)
                lista.coloca(lista.elementoEm(i), i + 1);
            lista.coloca(x, i + 1);
       }
    }

}
```

```
package ordena.impl;

import ordena.specs.*;

public class OrdenadorPorQuickSortImpl implements Ordenador {

    public void ordena (Lista lista) {
        quicksort (lista, 0, lista.tamanho() - 1);
    }

    private void quicksort (Lista v, int p, int r)  {
       int j;
       if (p < r) {
          j = separa (v, p, r);
          quicksort(v, p, j-1);
          quicksort(v, j+1, r);
       }
    }
```

```
    private int separa (Lista v, int p, int r)
    {
       int i = p+1, j = r;
       Elemento c = v.elementoEm(p);
       while (i <= j) {
          if (v.elementoEm(i).compara(c) <= 0) ++i;
          else if (c.compara(v.elementoEm(j)) < 1) --j;
          else {
             v.troca (i, j);
             ++i; --j;
          }
       }
       v.troca (p, j);
       return j;
    }

}
```

Camada de Configuração:

```
package ordena.conf;

import ordena.specs.*;
import java.util.HashMap;

public abstract class Conf {

    protected ElementoFactory ef;
    protected ListaFactory lf;
    protected Ordenador o;
    protected static final HashMap configuracoes;

    static {
        configuracoes = new HashMap();
        InicializadorDeConfiguracao.inicializa();
    }

    public static Conf pegaConf (String nome) {
        return (Conf)configuracoes.get(nome);
    }

    public Ordenador pegaOrdenador () {
        return o;
    }

    public ListaFactory pegaListaFactory () {
        return lf;
    }

    public ElementoFactory pegaElementoFactory() {
        return ef;
```

```
    }
}
```

```
package ordena.conf;

import ordena.impl.*;

public class ConfInsercao extends Conf{

    protected ConfInsercao() {
        this.ef = new ElementoNumeroInteiroImplFactory();
        this.lf = new ListaImplFactory();
        this.o = new OrdenadorPorInsercaoImpl();
    }

}
```

```
package ordena.conf;

import ordena.impl.*;

public class ConfQuicksort extends Conf{

    protected ConfQuicksort() {
        this.ef = new ElementoNumeroInteiroImplFactory();
        this.lf = new ListaImplFactory();
        this.o = new OrdenadorPorQuickSortImpl ();
    }

}
```

```
package ordena.conf;


public abstract class InicializadorDeConfiguracao {

   public static void inicializa () {
        Conf.configuracoes.put("insercao", new ConfInsercao ());
        Conf.configuracoes.put("quicksort", new ConfQuicksort ());
   }

}
```

O programa executável:

```
package ordena;

import ordena.conf.*;
import ordena.specs.*;
```

```
import java.util.Random;

public class Teste {

    public static void main (String args[]) {
        if (args.length != 1) {
            System.out.println("Numero incorreto de parametros.");
            System.exit(1);
        }
        Conf conf = Conf.pegaConf(args[0]);
        if (conf == null) {
            System.out.println("Configuracao nao encontrada");
            System.exit(1);
        }
        ListaFactory lf = conf.pegaListaFactory();
        ElementoFactory ef = conf.pegaElementoFactory();
        Ordenador ordenador = conf.pegaOrdenador();

        Random rand = new Random();
        Lista lista = lf.criaLista();
        for (int i = 0; i < 50; i++) {
            Elemento e = ef.criaElemento(rand.nextInt());
            lista.adiciona (e);
        }
        ordenador.ordena (lista);
    }
}
```

Usos conhecidos

Prevayler – http://www.prevayler.org
O Prevayler é um framework para a persistencia de objetos em java.

Transferidor de arquivos [3]
Tal sistema foi implementado pelos autores como projeto final na disciplina MAC0440 – Sistemas de objetos distribuídos durante o ano de 2002.

Transcritor Fonético – http://www.linux.ime.usp.br/~emilio/mac499/
Projeto de formatura do aluno Emilio de C. Francesquini

Padrões relacionados

Veja AbtractFacory, Builder, FactoryMethod, Prototype, Singleton, Adapter , Proxy em [1].

Referências

[1] Gamma, et al., Padrões de Projeto (*Design Patterns*), 1995
[2] Feofiloff, http://www.ime.usp.br/~pf/algoritmos/aulas/docu.htm
[3] Conde D., Francesquini E. Transferidor de arquivos