Padrão Depurador

André Gustavo Andrade
Rodrigo Moreira Barbosa

October 27, 2002
Versão 0.1

Abstract

Em todas as aplicações que desenvolvemos, sempre necessitamos, em um dado momento, depurar nosso código. Este padrão visa a tornar essa tarefa um pouco mais simples e organizada, sem contudo perder flexibilidade e configurabilidade. Além disso, também é muito útil para ser utilizado em aplicações que necessitem de *log*.

1 Exemplo

Imagine uma certa classe de uma aplicação recentemente desenvolvida por você. A partir de um certo momento, depois de depurar esta classe razoavelmente bem, você se convence de que não há erros, e portanto, remove aquele monte de *System.outo.println*'s da classe. Depois de seis meses de uso (e conseqüentemente seis meses de desgaste da sua memória), você infelizmente descobre que existe uma entrada para a qual sua classe não funciona. Pior: você não sabe **exatamente onde** no meio de todo aquele código de seis meses atrás está o defeito.

Se esse fenômeno já pode ser um desastre restrito a uma classe isolada, imagine o que pode acontecer se você não puder determinar com certeza em qual classe o erro se deu? Esse caso é por si só desesperador, porque não apenas exige uma nova etapa de depuração, como a reinserção dos códigos apagados, e portanto, duplicação de trabalho. Por outro lado, deixar as mensagens de depuração para sempre dentro do código implica uma péssima estética para o usuário, que ocasionalmente vai ver passar à sua frente um monte de mensagens que não fazem o menor sentido para ele (e ocasionalmente, fazendo-o ignorar mensagens realmente importantes as quais ele deveria ver).

Este é um exemplo típico de quando devemos usar o padrão *Depurador*. Quando estamos desenvolvendo aplicações grandes e complexas que necessitem de manutenção ou expansão constantes. Pela similaridade entre mensagens de erro e *log* de operações, é fácil ver que este padrão também serve para ser utilizado em sistemas que necessitem de log.

2 Contexto

Desenvolvimento de aplicações que necessitem de constante depuração ou aplicações que necessitem de um sistema de registro (*log*).

3 Problema

Aplicações quando são inicialmente programadas necessitam de depuração. Muitas pessoas tem um pouco de aversão a usar depuradores, e isso por alguns motivos:

- Muitas vezes é difícil analisar as mensagens mostradas pelo depurador, bem como seguir a pilha inteira para saber onde está a origem do erro.
- Estabelecer *break points* pode ser cansativo, e executar o programa até onde um *break point* esteja inserido, por exemplo dentro de um laço, pode ser extenuante.
- Apesar dos avanços feito na área de depuração, com a criação de interface gráfica para os depuradores mais conhecidos, ainda assim, lidar com esses softwares exige o aprendizado de uma nova ferramenta

Por esses e outros motivos, o que se faz em geral é imprimir algum tipo de mensagem de depuração no meio do código, muitas vezes exibindo valores de variáveis, e outras informações importantes. Esse método tem a vantagem de ser **persistente**, enquanto a depuração com um depurador é sempre nova a cada seção. Por outro lado, algumas vertentes veêm alguns problemas com essa técnica, a saber:

- Há poluição do código com comandos de impressão de mensagens.
- Quando o software estiver pronto, as mensagens de depuração devem ser removidas, o que acarreta em um novo trabalho. Quando não removidas, devem ser ao menos comentadas.

4 Solução

A proposta apresentada pelo padrão Depurador é simples, e se baseia no seguinte paradigma:

- Manter as mensagens de depuração, podendo facilmente habilitar e desabilitar as mesmas.
- Permitir depuração tanto por classe quanto por característica funcional.
- Permitir vários tipos de saída, bem como a especificação e a implementação de novos tipos de saída, além de configurações.

Dessa forma, o usuário quando desejar fazer um log ou uma depuração de um dado trecho de código, tipo de erro etc., chama um método da classe Depurador, enviando uma *string* de identificação do trecho, tipo de erro etc. São instanciados objetos de configuração e saída para a depurador. A partir daí, todo mensagem de depuração/log de um dado tipo são exibidas utilizando-se essa *string* para se obter o depurador correto. Assim, podemos ter vários depuradores dentro do código trabalhando de forma consistente e configurável.

5 Estrutura

A estrutura básica do padrão pode ser observada na Figura 1.

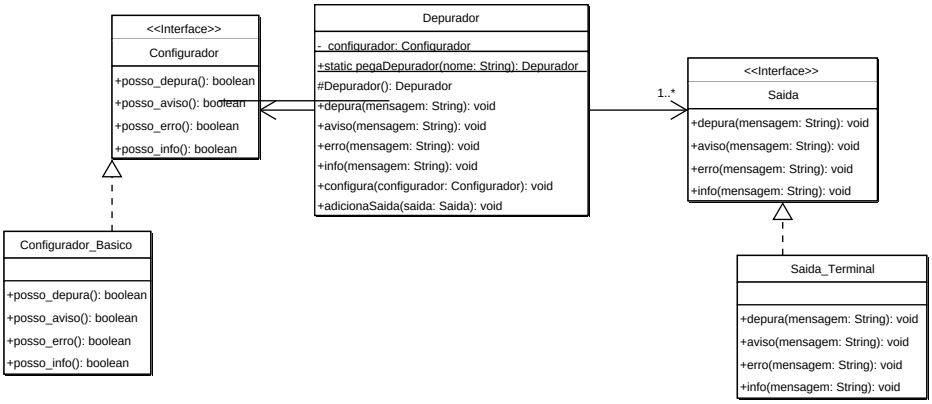

Figure 1: Diagrama de classes

Assim, o padrão é constituído de duas interfaces e três classes:

5.1 Configurador

Esta é a interface responsável por definir os Configuradores. Configuradores são objetos que são capazes de determinar quais dos diferentes tipos de mensagens existentes podem ser impressas.

Deixá-lo apenas como uma interface possibilita ao usuário implementar diferentes tipos de configuradores, podendo inclusive serem alterados dinâmicamente ou até mesmo remotamente, através de chamadas CORBA, por exemplo.

5.2 Saída

Esta interface descreve para onde as mensagens de cada tipo - depuração, erro, informação, aviso - devem ir. Em geral, todas vão para o mesmo dispositivo, sendo que estes podem ser livremente programados pelo usuário, como arquivos, terminais, redes, etc. Mas além disso, permite que se programe a saída para mandar as mensagens de cada tipo para dispositivos diferentes.

5.3 Configurador_Basico

Responsabilidades: É um configurador padrão, para o caso do usuário não puder (souber) desenvolver um novo configurador. Tem como responsabilidade configurar o sistema para imprimir todas as mensagens.

5.4 Saida_Terminal

Responsabilidades: Saída padrão. Tem como responsabilidade imprimir no terminal todas as mensagens.

5.5 Depurador

Responsabilidades: Devolver a instância certa de Depurador, controlar o destino de cada mensagem enviada.

A devolução do depurador correto é feita mediante o envio de uma *string* de id. Toda vez que um novo depurador é solicitado mediante o uso de uma dada *string* a mesma instância é devolvida. Isso garante a utilização do depurador correto e livra o programador da responsabilidade de controlar as instâncias de Depurador.

Colaboradores: Configurador, Saida.

6 Dinâmica

No método a ser depurado, uma chamada `Depurador pegaDepurador (id)` é realizada. *id* é um identificador único para cada depurador. Se duas chamadas com o mesmo id forem feitas a `pegaDepurador`, a mesma instância de Depurador é devolvida (algo semelhante ao padrão **singleton**). De posse do objeto concreto da clase Depurador, instancia-se uma classe que implemente a interface Configurador. Esse objeto pode ser da classe Configurador_Basico, por exemplo. Esse objeto é associado ao objeto Depurador através do método `configura`. O objeto do tipo Configurador determina se uma mensagem de um dado tipo (depuração, erro,

informação, aviso) deverá ser considerada ou não. Em seguida, instancia-se um objeto que implementa Saida, que pode ser o Saida_Terminal. Esse objeto implementa os métodos de exibição de mensagens de depuração, aviso, erro, e informações gerais. Esse objeto Saida, é associado a Depurador através do método `adicionaSaida`. Vale notar, que vários objetos Saida podem ser associados, o que implica várias formas diferentes de se tratar um dado tipo de mensagem. Feitas essas configurações, toda vez que o programador quiser exibir uma mensagem qualquer para um dado método, basta chamar o método `pegaDepurador` com o id correto e de posse do objeto Depurador usar os métodos `depura, aviso, erro, info` para exibição de mensagens de depuração. Para desabilitar saídas basta implementar um configurador que devolva false nos métodos `possoDepurar (), possoAviso ()` etc. de acordo com o tipo de mensagem que não interessa mais.

7 Implementação

```
public interface Configurador {
    public boolean posso_depura();
    public boolean posso_aviso();
    public boolean posso_erro();
    public boolean posso_info();
}
public class Configurador_Basico implements Configurador{

    public Configurador_Basico()                                    Configurador_Basico
    {                                                               10
    }

    public boolean posso_depura()                                   posso_depura
    {
        return true;
    }

    public boolean posso_aviso()                                    posso_aviso
    {
        return true;                                                20
    }

    public boolean posso_erro()                                     posso_erro
    {
        return true;
    }

    public boolean posso_info()                                     posso_info
    {
        return true;                                                30
    }
}
```

```
public class Configurador_Dinamico implements Configurador{

    public boolean depura;
    public boolean aviso;
    public boolean erro;
    public boolean info;

    public Configurador_Dinamico()                                          40 Configurador_Din
    {
        depura=true;
        aviso=true;
        erro=true;
        info=true;
    }

    public Configurador_Dinamico(boolean dep, boolean avis, boolean er,     Configurador_Dinam
                                 boolean inf)
    {                                                                       50
        depura=dep;
        aviso=avis;
        erro=er;
        info=inf;
    }

    public boolean posso_depura()                                           posso_depura
    {
        return depura;
    }                                                                       60

    public boolean posso_aviso()                                            posso_aviso
    {
        return aviso;
    }

    public boolean posso_erro()                                             posso_erro
    {
        return erro;
    }                                                                       70

    public boolean posso_info()                                             posso_info
    {
        return info;
    }

    public void seta_depura(boolean dep)                                    seta_depura
    {
        depura=dep;
```

```
    }                                                                    80

    public void seta_aviso(boolean avis)                                 seta_aviso
    {
        aviso=avis;
    }

    public void seta_erro(boolean er)                                    seta_erro
    {
        erro=er;
    }                                                                    90

    public void seta_info(boolean inf)                                   seta_info
    {
        info=inf;
    }
}

public class Configurador_Inicializavel implements Configurador {
    private boolean depura;
    private boolean aviso;                                               100
    private boolean erro;
    private boolean info;

    public Configurador_Inicializavel()                                  Configurador_Iniciali
    {
        depura=true;
        aviso=true;
        erro=true;
        info=true;
    }                                                                    110

    public Configurador_Inicializavel(boolean dep, boolean avis, boolean er,  Configurador_Iniciali
                                      boolean inf)
    {
        depura=dep;
        aviso=avis;
        erro=er;
        info=inf;
    }

                                                                         120
    public boolean posso_depura()                                        posso_depura
    {
        return depura;
    }

    public boolean posso_aviso()                                         posso_aviso
```

```
    {
        return aviso;
    }
                                                                    130
    public boolean posso_erro()                                     posso_erro
    {
        return erro;
    }

    public boolean posso_info()                                     posso_info
    {
        return info;
    }
}                                                                   140

import java.util.*;
public class Depurador {

    private static Hashtable nomes=new Hashtable();
    private Vector saidas;
    private Configurador _configurador;

    private Depurador()                                             Depurador
    {                                                               150
        saidas=new Vector();
    }

    static Depurador pegaDepurador(String nome)                     pegaDepurador
    {
        if (Depurador.nomes.containsKey(nome))
            return (Depurador)(Depurador.nomes.get(nome));
        Depurador dep=new Depurador();
        Depurador.nomes.put(nome,dep);
        return dep;                                                 160
    }


    void depura(String mensagem)                                    depura
    {
        if (_configurador.posso_depura())
            {
                for (Enumeration e=saidas.elements();e.hasMoreElements();)
                    {
                        ((Saida)(e.nextElement())).depura(mensagem);    170
                    }
            }
    }
```

```
    void aviso(String mensagem)                                          aviso
    {
        if (_configurador.posso_aviso())
            {
                for (Enumeration e=saidas.elements();e.hasMoreElements();)
                    {                                                    180
                        ((Saida)(e.nextElement())).aviso(mensagem);
                    }
            }
    }

    void erro(String mensagem)                                           erro
    {
        if (_configurador.posso_erro())
            {
                for (Enumeration e=saidas.elements();e.hasMoreElements();)  190
                    {
                        ((Saida)(e.nextElement())).erro(mensagem);
                    }
            }
    }

    void info(String mensagem)                                           info
    {
        if (_configurador.posso_info())
            {                                                            200
                for (Enumeration e=saidas.elements();e.hasMoreElements();)
                    {
                        ((Saida)(e.nextElement())).info(mensagem);
                    }
            }
    }

    void configura(Configurador configurador)                            configura
    {
        _configurador=configurador;                                      210
    }

    void adiciona_Saida(Saida s)                                         adiciona_Saida
    {
        saidas.add(s);
    }
}

public interface Saida{
    public void depura(String mensagem);                                 220
```

```
    public void aviso(String mensagem);
    public void erro(String mensagem);
    public void info(String mensagem);
}
public class Saida_Diferenciada implements Saida {

    public void depura(String mensagem)                                    depura
    {
        System.out.println("DEPURANDO: "+mensagem);
    }                                                                      230

    public void aviso(String mensagem)                                     aviso
    {
        System.out.println("! AVISO: "+mensagem+" !");
    }

    public void erro(String mensagem)                                      erro
    {
        System.out.println("!!! ERRO: "+mensagem+" !!!");
    }                                                                      240

    public void info(String mensagem)                                      info
    {
        System.out.println("-->INFO: "+mensagem);
    }
}
public class Saida_Terminal implements Saida {

    public void depura(String mensagem)                                    depura
    {                                                                      250
        System.out.println(mensagem);
    }

    public void aviso(String mensagem)                                     aviso
    {
        System.out.println(mensagem);
    }

    public void erro(String mensagem)                                      erro
    {                                                                      260
        System.out.println(mensagem);
    }

    public void info(String mensagem)                                      info
    {
        System.out.println(mensagem);
    }
```

```
}
public class Teste1 {
    public static void main(String args[])                                    270 main
    {
        Configurador_Dinamico conf=new Configurador_Dinamico(false,false,
                                                              true,true);
        Depurador dep=Depurador.pegaDepurador("Saida 1");
        dep.configura(conf);
        Saida_Terminal s1=new Saida_Terminal();
        dep.adiciona_Saida(s1);

        System.out.println("Isto ocorre em uma parte qualquer do programa. . .\n");
                                                                              280
        dep.depura("Estou fazendo um debug. . .");
        dep.info("Estou informando algo. . .");
        dep.aviso("Estou avisando algo. . .");
        dep.erro("Xi. . . Deu erro");

        System.out.println("\nAgora eu ativo as mensagens de debug\n");

        conf.seta_depura(true);

        dep.depura("Estou fazendo um debug. . .");                            290
        dep.info("Estou informando algo. . .");
        dep.aviso("Estou avisando algo. . .");
        dep.erro("Xi. . . Deu erro");

        System.out.println("\nOutro teste\n");

        Configurador_Dinamico conf2=new Configurador_Dinamico(true,true,
                                                              false,false);
        Saida_Diferenciada s2=new Saida_Diferenciada();
        Depurador dep2=Depurador.pegaDepurador("Saida 2");                    300
        dep2.configura(conf2);
        dep2.adiciona_Saida(s2);

        dep2.depura("Estou fazendo um debug. . .");
        dep2.info("Estou informando algo. . .");
        dep2.aviso("Estou avisando algo. . .");
        dep2.erro("Xi. . . Deu erro");

        dep2.adiciona_Saida(s1);
                                                                              310
        System.out.println("\nMais de uma saida\n");

        dep2.depura("Estou fazendo um debug. . .");
        dep2.info("Estou informando algo. . .");
```

```
        dep2.aviso("Estou avisando algo. . .");
        dep2.erro("Xi. . . Deu erro");
    }
}
```

320

8 Usos conhecidos

- Uma implementação de *log* semelhante à apresentada neste padrão foi utilizada no framework *log* do projeto Jakarta do Apache.
- A API de log do J2SE implementa mecanismos bastante semelhantes ao padrão Depurador.

9 Conseqüências

9.1 Vantagens

As vantagens do padrão são claras: há uma maior padronização da depuração, além de uma maior versatilidade na exibição de mensagens, com a eliminação fácil de mensagens idesejáveis. Além disso, com a utilização desse padrão, a inserção de um log na aplicação torna-se trivial.

9.2 Desvantagens

Entre algumas desvantagens, podemos citar, obviamente, a necessidade de seu aprendizado. Além disso, sua complexidade é maior do que simplesmente inserir-se `system.out.println`'s . Em aplicações muito simples, tal padrão acaba sendo inadequado.

10 Veja também

Seria interessante a revisão de padrões como Singleton(130) e Strategy(292) do GoF. A classe Depurador utiliza uma variação do padrão Singleton e as interfaces Configurador e Saída foram criadas para a implementação de um Strategy.