Padrão de projeto de software

Paulo Venancio Lopes e Daniel Sguillaro

Nome

Roupa Suja Se Lava Em Casa.

Intenção

Dar maior capacidade e flexibilidade ao conceito de entidade (no contexto de persitência de objetos).

Motivação

Pensemos na definição de entidade: é um objeto representativo de uma entidade (no sentido do modelo E/R) persistente, e que suas informações tipicamente residem em um banco de dados.

Considere agora uma entidade que, ao invés de ter somente seus atributos com seus estados persistidos em um banco de dados, consiga também verificar, por exemplo, em qual mídia ela deve ser persistida. E que ainda possa prover funcionalidades extras como retornar dados formatados, lista dados com formatos especiais, etc.

Tomemos como exemplo um sistema de controle de uma locadora de DVDs. Um cliente da locadora será representado no sistema por uma entidade **Cliente.** A entidade deve ser capaz de saber qual a idade, endereço, etc, do cliente que representa. Também pode buscar dados de outros clientes, como por exemplo, retornar uma lista de clientes. Isto tudo por que a entidade Cliente deve conhecer todas as suas propriedades e ainda ser capaz de resolver problemas simples sobre si mesmo.

Uma entidade delegaria a um **DAO**[1] a função de fazer a manutenção dos dados de tal forma que cada DAO seria específico a uma mídia de persistência diferente, podendo assim a entidade ter vários DAOs de tal forma que ela escolheria a melhor opção.

Funcionalidades extras poderiam ser adicionadas a entidade através de **Decorator**[2], tais como mensagens formatadas, conversão de dados, buscas especiais, e etc. Por isso a entidade se comportaria como o velho dito popular, onde se diz que roupa suja se lava em casa, não deixando que os problemas sejam passados para outros lugares e sim para aqueles que sabem exatamente quais são e como resolve-los.

[1] DAO (Data Access Object): padrão usado para persistência de dados (http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html).

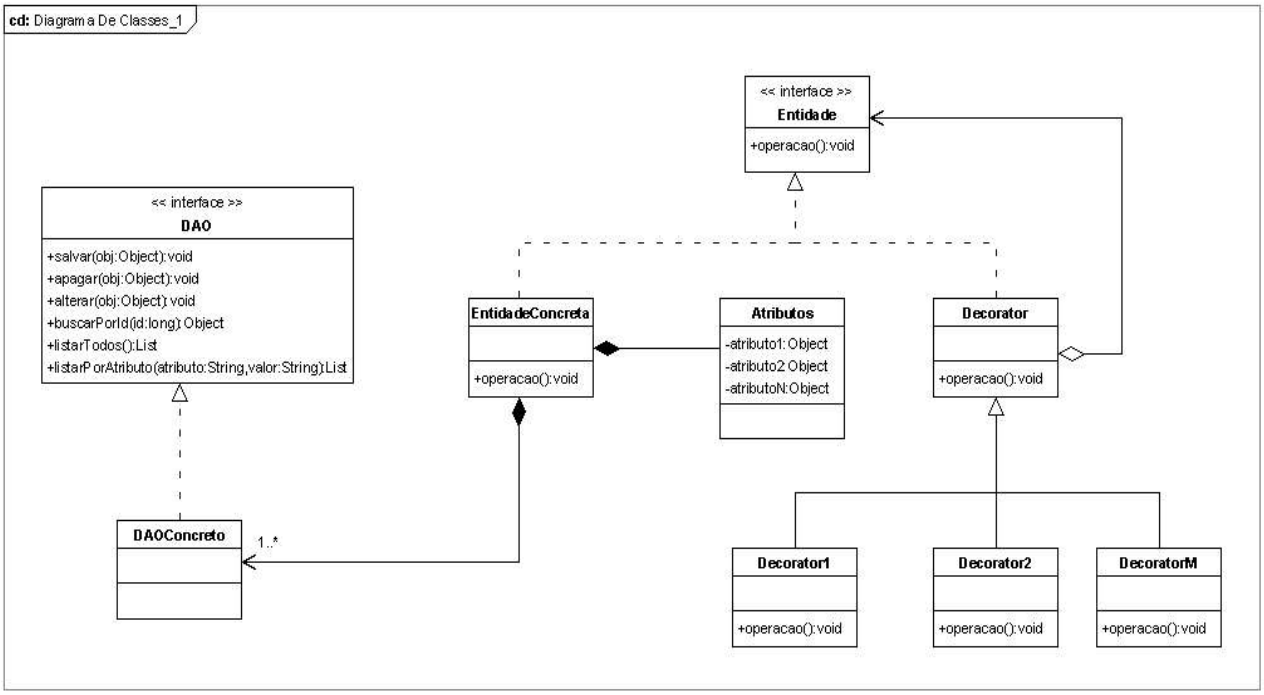
[2] Decorator: padrão que acrescenta funcionalidades extras a um objeto (http://en.wikipedia.org/wiki/Decorator_pattern).

Aplicabilidade

Este padrão será aplicado quando o objeto possuir uma ou mais das caracteristicas abaixo:

1- for complexo, com muitas funcionalidades;
2- for persistente;
3- tiver manutenção de seus dados;
4- precisar ser visto por diferentes aplicações;
5- precisar de funcionalidades especiais.

Estrutura



Participantes

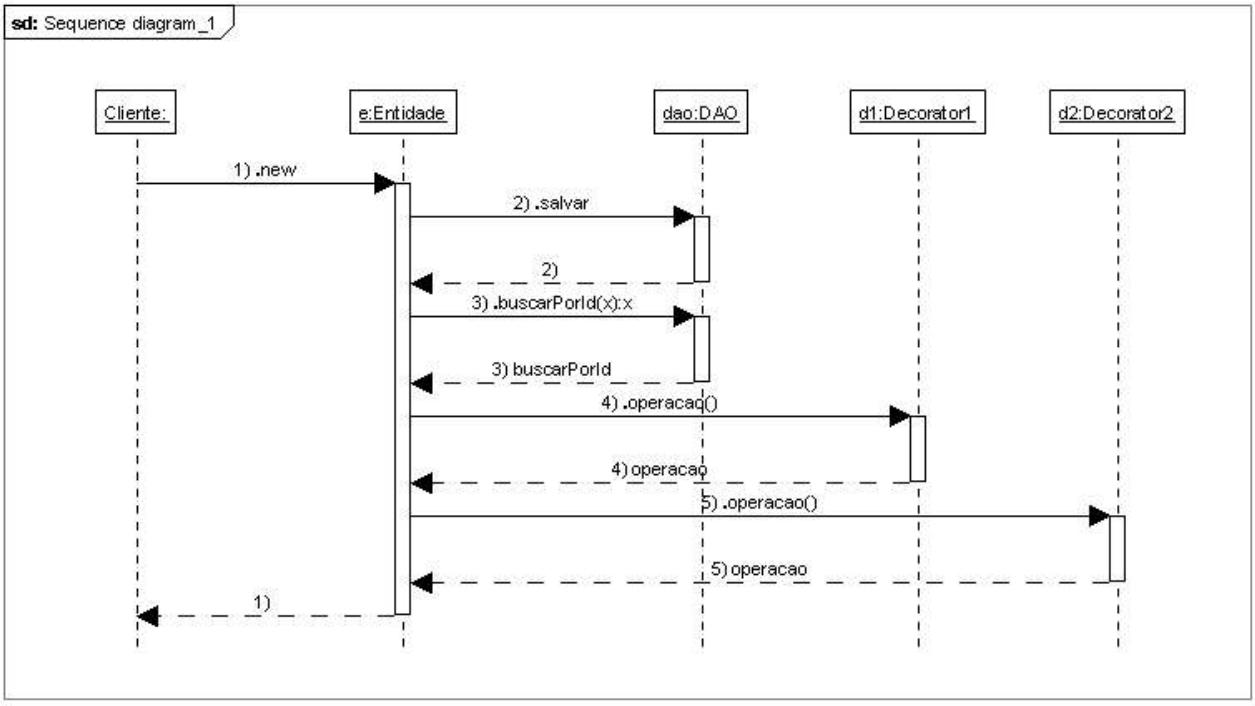
DAO

Faz a manutenção dos dados (inserir, apagar, trocar), prover métodos de busca de dados persistidos, e listar dados de várias formas.

Decorator

Adiciona funcionalidades especiais com baixo acoplamento, dentre elas podemos ter formatar dados, cálculos, e outras.

Colaborações



Consequências:

O objeto criado saberá se deve ser persistido e qual meio será usada para este processo. Se for persistido, terá um mínimo de métodos para a sua manutenção no banco de dados. O objeto poderá ter funcionalidades extras para garantir eficiência para a camada de modelo e ainda poderá ser visto por outros clientes.

As vantagens de padrão são:

1- inteligência, pois descobre o melhor meio de persistir seus dados.
2- tem funcionalidades mínimas para a manutenção de seus dados.
3- pode adicionar funcionalidades extras sem mexer nas suas características.
4- Maior flexibilidade para o padrão MVC, pois a camada de modelo apenas cria a entidade e chama os métodos de manutenção ou as função extras sem se preocupar com seu funcionamento.
5- Divisão de responsabilidade, pois projeta-se as entidades separadamente das outras fases.

As desvantagens são:

1- Objeto entidade fica mais complexo, não podendo ser usado em qualquer caso.

2- Antecipação de problemas, o que segundo o método XP não é um bom método de desenvolvimento

Implementação

Para a implementação devemos levar em conta os seguintes critérios:

1- Criar um javabean para os atributos de sua entidades. Este objeto java será usado para manutenção dos dados na tabela do banco de dados, compartilhar dados com os Decorators, e etc.

2- Criar uma interface de DAO genérica que indica os métodos mínimos que uma implementação de manutenção de dados no banco de dados deve fazer.

3- Cria as classe(s) que implementara(ão) a interface DAO genérica. No mínimo deveremos ter uma classe, mas podemos ter quantas quisermos. Pense num caso em que vamos salvar os dado em uma banco de dados, mas este esta em um servidor remoto, com problemas de conexão. Neste caso podemos criar um DAO que persiste os dados em um arquivo no nosso desktop e quando a conexão estiver aberta ele salva os dados no banco de dados. A entidade deverá verificar estas possibilidades não deixando transparecer para outras camadas.

4- Criar uma interface Entidade com pelo menos dois métodos, o getAtributos que serve para buscar os atributos encapsulados no objeto java, e o método operação (ou qualquer nome) que faz a decoração da entidade.

5- Criar classe abstrata Decorador que implementa a interface Entidade e coloca uma referência para o objeto java com os atributos da entidade. Nesta classe será implementado os métodos getAtributo para buscar os atributos da entidade e o método operação que faz uma implementação default da decoração.

6- Criar as classe(s) de decoração que são subclasses da classe Decorador, sobre-escrevendo o método operação, daí criando seu próprio decorador.

Código Exemplo

```
public class Atributo {

        private tipo campo1;
        ...
        private tipo campoN;

        public tipo getCampo1() {return campo1;}
        public void setCampo1(tipo campo1) {this.campo1=campo1;}
        ...
        public tipo getCampoN() {return campoN;}
```

```
        public void setCampoN(tipo campoN) {this.campoN=campoN;}

}

interface  DaoGenerico {

        void salvar(Object ob);
        void apagar(Object ob);
        void alterar(Object ob);
        Object buscarPorId(Long id);
        List ListaTodos();
        List listarPorAtributo(String campo, String valor);
        prontoPraSalvar();

}

public class DaoConcreto1 implements DaoGenerico {

        void salvar(Object ob) {
            ...
        }
        ...

}

public class DaoConcreto2 implements DaoGenerico {

        void salvar(Object ob) {
            ...
        }
        ...

}

interface Entidade {

        void operacao();
        Object getAtributos();

}

public class EntidadeConcreta implements Entidade {

        private DaoGenerico g1 =  new DaoConcreto1();
        private DaoGenerico g2 =  new DaoConcreto2();
        private Atributos at = new Atributos();
```

```
        Object getAtributos() { return at;}
        void operacao() { //Operacao default}

        // Tomando decisão de onde salvar
        void salvar() {
            if (g1.prontoParaSalvar())
                g1.salvar(at);
            else
                g2.salvar(at);
        }
        ....
        //Outros métodos
        ...
}

public abstract class Decorador implements Entidade {

        // instância concreta de Entidade
        private Entidade ent;
        public Decorator(Entidade ent) {this.ent = ent;}
        public getAtributos() {return ent.getAtributos();}
        //Caso não tenhamos implemtações esta é a operação default
        public void operacao() {ent.operacao();}

}

public class Decorador1 extends Decorador {

        public Decorator(Entidade ent) {
            super(ent);
            //Implementa inicializacao
            ...
        }

        void operacao() {//Implementa decaração 1}

}

public class Decorador2 extends Decorador {

        public Decorator(Entidade ent) {
            super(ent);
            //Implementa inicializacao
            ...
        }
        void operacao() {//Implementa decaração 1}
```

}

Padrões Relacioinados

DAO (Data Access Object)

Padrão usado para persistência de dados

Decorator

Padrão que acrescenta funcionalidades extras a um objeto