UNIVERSIDADE
DO ESTADO DE MINAS GERAIS

UEMG

SEMINÁRIO- OPTATIVA
PROFº JOÃO PAULO
8º PERÍODO

PADRÕES DE PROJETO

PROTOTYPE

DOUGLAS MAGALHÃES BORGES
LAYON MARTINS FONSECA
MIRELA ÁGATHA AMARAL

PASSOS 2015

RESUMO

Somente conhecer e utilizar os princípios básicos da orientação a objetos ou de uma linguagem de programação, não garante o desenvolvimento de softwares flexíveis, reutilizáveis e de fácil manutenção. Os padrões de projeto representam soluções que tentam suprir tais necessidades. No entanto, a utilização de alguns padrões, apesar de ser benéfica na maioria dos casos, torna o código-fonte maior e mais complexo. Isto nos faz refletir sobre a possibilidade de estarmos desnecessariamente aumentando a complexidade do design. Portanto, é necessário não somente conhecer os padrões de projeto, mas sim, realmente entendê-los para identificar quando utilizá-los e usufruir positivamente.

O presente trabalho tem por finalidade apresentar a definição, objetivo e importância dos padrões de projeto, bem como apresentar o padrão Prototype e sua utilização.

Padrão de Projeto

Design Pattern

Introdução

A necessidade da utilização dos Padrões de Projeto tornasse cada vez mais necessária para desenvolver projetos que satisfaz e possibilita fazer teste, para corrigir eventuais erros, também ajuda caso o projeto tenha que ser uma arquitetura flexível para acomodar futuros problemas e requisitos sem a necessidade de realizar um novo projeto.

Cada tipo de padrão descreve problemas que ocorre diversas vezes, descrevendo um projeto de solução para o problema, e dessa forma podendo reutilizar está solução diversas vezes.

O padrão é formado por quatro elementos essenciais, sendo o nome, descrição do problema a qual o padrão se aplica, descrição da solução genérica proposta, consequências da aplicação do padrão (custo e benefício).

Com os padrões de projetos pode-se aprender com a experiência de outras pessoas, ajudando a resolver os principais problemas dos iniciantes, permite que faça bons projetos mais rapidamente, os softwares são de melhor qualidade com melhores práticas em orientação a objeto com soluções bem testadas e documentadas.

Tipos de Padrões de Projeto

Criação

- ✓ **Factory Method** (Método Fábrica): Criando de objetos mas deixa a implementação para as subclasses.
- ✓ **Abstract Factory** (Fábrica Abstrata)**:** Provê uma interface para criação de famílias de objetos relacionados sem especificar suas classes concretas.
- ✓ **Builder** (Construtor)**:** Separa o processo de construção de objetos complexos, desacoplando a criação de instâncias destes objetos.
- ✓ **Prototype (Protótipo):** Define um protótipo dos objetos a serem usados e cria-os clonando este protótipo.
- ✓ **Singleton** (Objeto Único)**:** Garante que uma classe possui somente uma instância e provê um ponto de acesso global a ela.

Estrutura

- ✓ **Composite**(Composto)**:** Agrupa objetos de mesma interface em estrutura de árvore para tratar todos eles como se fossem uma única entidade.
- ✓ **Adapter** (Adaptador)**:** Converte uma interface de uma classe em outra que objetos clientes esperam utilizar.
- ✓ **Bridge** (Ponte)**:** Desacopla uma abstração de sua implementação para que ambas possam variar independentemente.
- ✓ **Decorator** (Decorador)**:** Estende a funcionalidade de um objeto dinamicamente anexando objetos que irão executar antes ou depois do objeto "decorado".
- ✓ **Façade** (Fachada)**:** Provê uma interface única (ponto central) para diversas outras interfaces do sistema.
- ✓ **Proxy** (Procurador)**:** Provê um intermediário (procurador) que representa o objeto mas se comporta de forma diferente.
- ✓ **Flyweight** (Peso Mosca)**:** Forma um pool de objetos imutáveis para serem utilizados em diversas partes do sistema.

Comportamento

- ✓ **Strategy (Estratégia):** Define uma família de algoritmos, encapsula-os em objetos e permite que sejam intercambiados.
- ✓ **Iterator (Iterador):** Acessar diversos elementos de um conjunto em sequência sem expor sua representação interna.
- ✓ **Template Method** (Método Modelo): Define o esqueleto do algoritmo da operação na superclasse, delegando partes do mesmo para as subclasses.
- ✓ **Observer** (Observador): Define uma dependência um-para-muitos entre objetos tal que quando um objeto muda de estado, todos são notificados.
- ✓ **Mediator** (Mediador): Delega a um objeto a responsabilidade de fazer outros objetos se comunicarem, tirando destes últimos o acoplamento entre si.
- ✓ **Command** (Comando): Encapsula requisições como objetos, permitindo parametrização, *log* ou *undo* de funções.
- ✓ **Memento** (Recordação): Sem violar o encapsulamento, armazena o estado atual do objeto para que possa ser restaurado posteriormente.
- ✓ **Chain of Responsibility** (Cadeia de Reponsabilidade): Monta uma corrente de objetos que serve a uma requisição, dando a cada oportunidade de respondê-la e/ou passá-la adiante.

- ✓ **Interpreter** (Interpretador): Criar uma linguagem de representação de operações e construir um interpretador para essa linguagem.
- ✓ **State** (Estado): Permite que um objeto altere seu comportamento quando mudar seu estado.
- ✓ **Visitor** (Visitante): Representa operações em objetos como outros objetos com uma interface comum, permitindo que sejam criadas novas operações se alterar o objeto.

Padrão PROTOTYPE

Objetivo

Especificar os tipos de objetos a serem criados usando uma instância-protótipo e criar novos objetos pela cópia desse protótipo.

Utilização

Um dos diversos cenários que podemos aplicar o padrão Prototype é quando necessitamos guardar o estado de um objeto, em determinados pontos de execução de um processamento, por exemplo. Na maioria das vezes, é mais interessante criarmos "clones", ou cópias, de um objeto em determinado momento, do que criar a instância manualmente.

Elementos do Prototype:

- ➔ **Prototype:** Abstração dos objetos que possuem a capacidade de se auto copiar;
- ➔ **ConcretePrototype** (Evento): Classe que define um tipo particular de objeto que pode ser clonado;
- ➔ **Client:** Classe que cria novos objetos a partir da interface definida por Prototype;

Quando usar Prototype?

Use o padrão Prototype quando um sistema tiver que ser independente de como os seus produtos são criados, compostos e representados; e:

Quando as instâncias de uma classe puderem ter uma dentre poucas combinações diferentes de estados. Pode ser mais conveniente instalar um

número correspondente de protótipos e cloná-los, ao invés de instanciar a classe manualmente, cada vez com um estado apropriado.

Diagrama de classe padrão

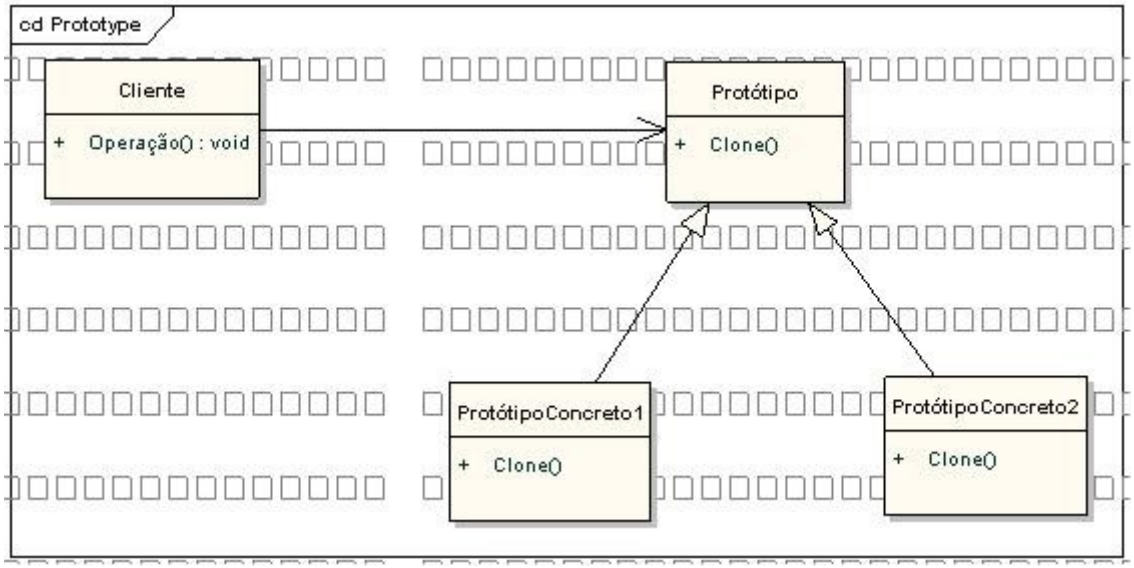

Fig. 1 – Diagrama de classe XML Padrão Prototype

Para explicar melhor vamos ao um exemplo pratico:

Situação:

A UEMG tem um departamento responsável pelos eventos de computação como, por exemplo, maratona de programação, semana acadêmica de TI, concurso de software onde recebe a mesma verba para a realização de todos; o cadastro é realizado no mesmo site, entre outros, fazendo com que sejam praticamente iguais. Para criar novos eventos ao longo do ano são modificadas poucas informações de um evento para o outro.

Problema:

O programador precisará criar várias classes para criar vários eventos.

Como seria utlilizando o padrão prototype?

Diagrama de classe

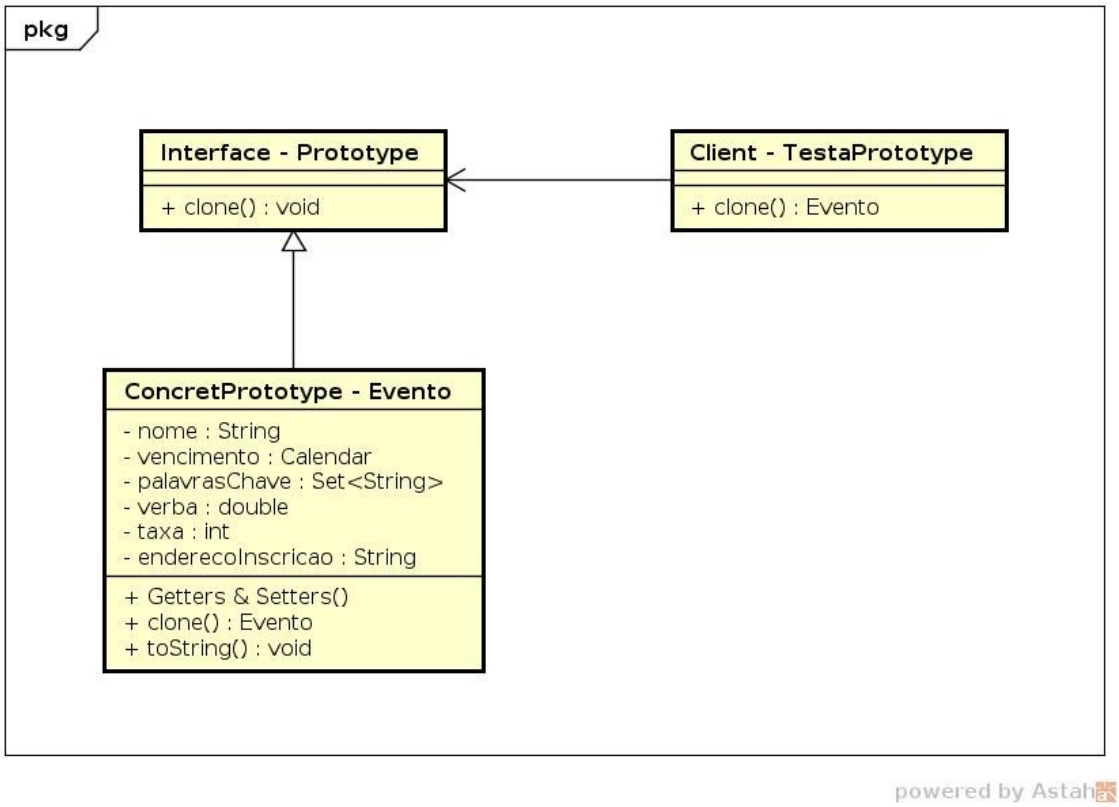

Fig. 2 – Diagrama de classe xml Evento

O codigo fonte da implementação deste projeto se econtra na pasta raiz do trabalho!

Feito em linguagem Java 8, é a IDE eclipse.