Padrões de Projeto de Software

Definindo Padrões de Projeto

Definir o que é um padrão de projeto de maneira clara e objetiva, tem sido o objetivo da comunidade de software, desde a década de 80.O primeiro a apresentar uma definição do que seria um padrão, foi o arquiteto de professor Christopher Alexandre, no seu livro “A Times Way of Building” (Oxford University Press, 1979), que é: “Cada padrão é uma regra de três partes, que expressa uma relação entre um certo contexto, um problema e uma solução”. Sendo assim para entender a necessidade, existência, de um padrão é necessário estudar suas partes: o problema, a solução e o contexto sobre o qual ele é aplicável.

Dessa forma, resumidamente pode-se entender como padrão de projeto, como a solução recorrente para um problema em um contexto, mesmo que em projetos e áreas distintas. Observe que os termos chaves dessa definição são: contexto, problema e solução, o que torna obrigatório à compreensão inequívoca de cada um. Um contexto diz respeito ao ambiente, e as circunstâncias dentro do qual algo existe. O problema é a questão indefinida, algo que precisa ser investigado e solucionado. Normalmente, está atrelado ao contexto em que ocorre. Finalmente, a solução refere à resposta do problema que ajuda a soluciona-lo.

Entretanto, se tivermos uma solução para um problema em um certo contexto, ela não necessariamente pode constituir um padrão, pois é necessário que ela tenha como característica a regularidade, isto é, ela se constituirá como um padrão se puder ser utilizada repetidamente.

Segundo Christopher Alexander, “cada padrão descreve um problema no nosso ambiente e o núcleo da sua solução, de tal forma que você possa usar esta solução mais de um milhão de vezes, sem nunca faze-lo da mesma maneira”.

O grupo dos quatro amigos classificou os padrões de projeto por dois critérios. O primeiro critério é a finalidade - reflete o que um padrão faz. Os padrões podem ter finalidades de criação, comportamento e estrutural. Os padrões de criação descrevem as técnicas para instanciar objetos (ou grupos de objetos), e possibilitam organizar classes e objetos em estrutura maiores, os de comportamento se caracterizam pela maneira pelas quais classes ou objetos interagem e distribuem responsabilidades e os estruturais lidam com a composição de classes ou objetos. O segundo critério é o escopo - especifica se o padrão é aplicado à classe ou objeto.

Características de Um Padrão de Projeto

Embora um padrão seja a descrição de um problema, de uma solução genérica e sua justificativa, isso não significa que qualquer solução conhecida para um problema possa constituir um padrão, pois existem características obrigatórias que devem ser atendidas pelos padrões:

1. Devem possuir um nome, que descreva o problema, as soluções e consequências. Um nome permiti definir o vocabulário a ser utilizado pelos projetistas e desenvolvedores em um nível mais alto de abstração.

2. Todo padrão deve relatar de maneira clara a qual (is)problema(s) ele deve ser aplicado, ou seja, quais são os problemas que quando inserido em um determinado contexto o padrão conseguirá resolve-lo. Alguns podendo exigir pré-condições.

3. Solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Um padrão deve ser uma solução concreta, ele deve ser exprimido em forma de gabarito (algoritmo) que, no entanto pode ser aplicado de maneiras diferentes.

4. Todo padrão deve relatar quais são as suas consequências para que possa ser analisada a solução alternativa de projetos e para a compreensão dos benefícios da aplicação do projeto.

Não pode ser considerado um padrão de projeto trecho de códigos específicos, mesmo que para o seu criador ele reflita um padrão, que soluciona um determinado problema, porque os padrões devem estar a um nível maior de abstração e não limitado a recursos de programação.

Um padrão de projeto nomeia, abstrai e identifica os aspectos chaves de uma estrutura de projeto comum para torna-la útil para a criação de um projeto orientado a objetos reutilizável.

A Importância dos Padrões de Projeto

O mais importante sobre os padrões é que eles são soluções aprovadas. Cada catálogo inclui apenas padrões que foram considerados úteis por diversos desenvolvedores em vários projetos. Os padrões catalogados também são bem definidos; os autores descrevem cada padrão com muito cuidado e em seu próprio contexto, portanto será fácil aplicar o padrão em suas próprias circunstâncias. Eles também formam um vocabulário comum entre os desenvolvedores.

Quando os Padrões Não o Ajudarão

Os padrões são um mapa, não uma estratégica. Os catálogos geralmente apresentarão algum código-fonte como uma estratégica de exemplo, portanto eles não devem ser considerados como definitivos. Os padrões não ajudarão a determinar qual aplicação você deve estar escrevendo apenas como implementar melhor a aplicação assim que o conjunto de recursos e outras exigências forem determinados. Os padrões ajudam com o que e como, mas não com por que ou quando.

O conceito de utilizar os padrões de forma indiscriminada é conhecida como antipadrões (anti patterns). De acordo com Andrew Koenig, se um padrão representa a “melhor prática”, então um antipadrão representa uma “lição aprendida”.

Existem duas noções de antipadrões:

1. Aqueles que descrevem uma solução ruim para um problema que resultou em uma situação ruim;

2. Aqueles que descrevem como se livrar de uma situação ruim e como proceder dessa situação para uma situação boa.

Em suma um antipadrão constitui ao uso indevido dos padrões de projeto, ou o seu uso exagerado, o que pode ser constatado pela utilização de padrões impróprios para um determinado contexto, ou uso inadequado. A utilização dos padrões proporciona um aumento na flexibilidade do sistema, entretanto pode deixa-lo mais complexo ou degradar a performance. Algumas perdas são toleráveis, mas subestimar os efeitos colaterais da adoção dos patterns, é um erro comum, principalmente daqueles que tomam o uso como um diferencial e não pela real necessidade.

Como Padrões de Projeto Solucionam Problemas de Projeto

O alvo principal do uso dos padrões de projeto no desenvolvimento de software é o da orientação a objetos. Como os objetos são os elementos chaves em projetos OO, a parte mais difícil do projeto é a decomposição de um sistema em objetos. A tarefa é difícil porque muitos fatores entram em jogo: encapsulamento, granularidade, dependência, flexibilidade, desempenho, evolução, reutilização e assim por diante. Todos influenciam a decomposição, frequentemente de formas conflitantes.

Muito dos objetos participantes provêm do método de análise. Porém, projetos orientados a objetos acabam sendo compostos por objetos que não possui uma contrapartida no mundo real.

As abstrações que surgem durante um projeto são as chaves para torna-lo flexível. Os padrões de projeto ajudam a identificar abstrações menos óbvias bem como os objetos que podem captura-las. Por exemplo, objetos que representam processo ou algoritmo não ocorrem na natureza, no entanto, eles são uma parte crucial de projetos flexíveis. Esses objetos são raramente encontrados durante a análise ou mesmo durante os estágios iniciais de um projeto; eles são descobertos mais tarde, durante o processo de tornar um projeto mais flexível e reutilizável.

Como Selecionar Um Padrão de Projeto

Escolher dentre os padrões existentes aquele que melhor soluciona um problema do projeto, sem cometer o erro de escolher de forma errônea e torná-lo inviável, é uma das tarefas mais difíceis. Em suma, a escolha de um padrão de projeto a ser utilizado, pode ser baseada nos seguintes critérios:

1. Considerar como os padrões de projeto solucionam problemas de projeto.

2. Examinarqual a intenção do padrão, ou seja, o que faz de fato o padrão de projeto, quais seus princípios e que tópico ou problema particular de projeto ele trata (soluciona).

3. Estudar como os padrões se relacionam.

4. Estudar as semelhanças existentes entre os padrões.

5. Examinar uma causa de reformulação de projeto.

6. Considerar o que deveria ser variável no seu projeto, ou seja, ao invés de considerar o que pode forçar uma mudança em um projeto, considerar o que você quer ser capaz de mudar sem reprojetá-lo.

Como Usar Um Padrão de Projeto

Depois de ter sido feita a escolha do(s) padrão (ões) a ser (em) utilizado(s) no projeto é necessária conhecer como utilizá-lo(s). Uma abordagem recomendada pela gangue dos quatros amigos para aplicar um padrão a um projeto é:

1. Ler o padrão por completo uma vez, para obter sua visão geral. Conhecer o padrão principalmente a sua aplicabilidade e consequências são importantes para que ele realmente solucione o seu problema;

2. Estudar seções Estrutura, Participantes e Colaborações. Assegurando-se de que compreendeu as classes e objetos no padrão e como se relacionam entre si;

3. Escolher os nomes para os participantes do padrão que tenham sentido no contexto da aplicação;

4. Definir as classes. Declarar as interfaces, estabelecer os seus relacionamentos de herança e definir as variáveis de instância que representam dados e referências a objetos. Identifique as classes existentes na aplicação que serão afetadas pelo padrão e modifique-as;

5. Defina os nomes específicos da aplicação para as operações no padrão. Os nomes em geral dependem da aplicação. Use as responsabilidades e colaborações associadas com cada operação como guia;

6. Implemente as operações para suportar as responsabilidades e colaborações presentes do padrão. A seção de Implementação oferece sugestões para guiá-lo na implementação.

Essas são apenas diretrizes que podem ser utilizadas até que seja obtido experiência e conhecimento necessário para desenvolver uma maneira de trabalho particular com os padrões de projeto. Os padrões de projeto não devem ser aplicados indiscriminadamente. Frequentemente eles obtêm flexibilidade e variabilidade pela introdução de níveis adicionais de endereçamento indireto, e isso pode complicar um projeto e/ou custar algo em termos de desempenho. Um padrão de projeto deverá apenas ser aplicado quando a flexibilidade que ele oferece for realmente necessária.

Principais Padrões de Projeto

Uma rápida leitura dos códigos da VCL (Visual Component Library) do Delphiconstata-se que ela fora construído fazendo uso intensivo dos padrões de projetos. O que é muito bom, pois constata o nível de excelência da ferramenta. Isto devido ao Delphiimplementar completamente as boas práticas da orientação a objetos – OOP, que auxiliam na implementação de projetos reutilizáveis.

Padrões de Criação

Os padrões de criação são aqueles que abstraem e ou adiam o processo criação dos objetos. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objeto delegará a instanciação para outro objeto.

Os padrões de criação tornam-se importantes à medida que os sistemas evoluem no sentido de dependerem mais da composição de objetos do que a herança de classes. O desenvolvimento baseado na composição de objetos possibilita que os objetos sejam compostos sem a necessidade de expor o

seu interior como acontece na herança de classe, o que possibilita a definição do comportamento dinamicamente e a ênfase desloca-se da codificação de maneira rígida de um conjunto fixo de comportamentos, para a definição de um conjunto menor de comportamentos que podem ser compostos em qualquer número para definir comportamentos mais complexos.

Há dois temas recorrentes nesses padrões. Primeiro todos encapsulam conhecimento sobre quais classes concretas são usadas pelo sistema. Segundo ocultam o modo como essas classes são criadas e montadas. Tudo que o sistema sabe no geral sobre os objetos é que suas classes são definidas por classes abstratas. Consequentemente, os padrões de criação dão muita flexibilidade no que é criado, quem cria, como e quando é criado. Eles permitem configurar um sistema com objetos "produto" que variam amplamente em estrutura e funcionalidade. A configuração pode ser estática (isto é, especificada em tempo de compilação) ou dinâmica (em tempo de execução).

Abstract Factory

A intenção deste é fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Também é conhecido como Kit.

Este padrão deve ser aplicado quando se deseja isolar a aplicação da implementação da classe concreta, que poderia ser um componente e ou framework específico no qual a aplicação conheceria apenas uma interface e a implementação concreta seria conhecida apenas em tempo de execução ou compilação.

Imagine que em uma aplicação houvesse a necessidade de que ela fosse implementada para oferecer suporte a plataformas e características distintas. Por exemplo: Uma visão desktop e uma móvel (celular Pocket PC). A maneira de constituí-la, seria definindo uma família de componentes para cada plataforma e uma fábrica que os instancia de acordo com a plataforma alvo na qual a aplicação estará sendo executada.

De acordo com o exposto pelos quatro amigos, o uso do padrão Abstract Factory deve estar restrito as seguintes situações:

- Um sistema deve ser independente de como seus produtos são criados, compostos ou representados;

- Um sistema deve ser configurado como um produto de uma família de múltiplos produtos;

- Uma família de objetos for projetada para ser usada em conjunto, e você necessita garantir esta restrição;

- Você quer fornecer uma biblioteca de classes de produtos e quer revelar somente suas interfaces, não suas implementações.

A estrutura arquitetural do padrão definido segundo GoF é de acordo com o apresentado na Figura 1.

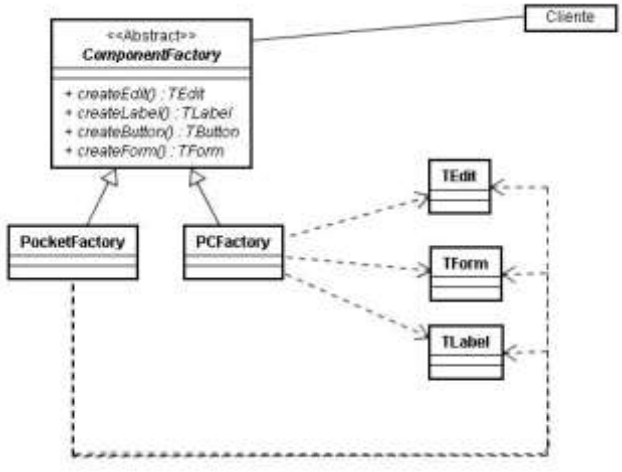

Figura 1. Estrutura arquitetural do padrão

A estrutura de um exemplo mais de acordo com a realidade do desenvolvedor é apresentada na Figura 2. A ideia básica apresentada pela figura é a de oferecer ao usuário (desenvolvedor) a possibilidade de executar uma aplicação sobre diferentes plataformas.

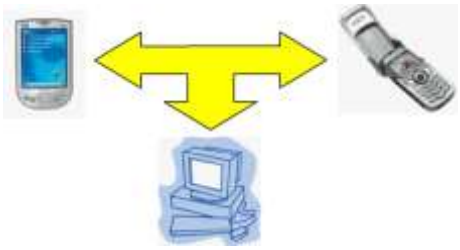

Figura 2. Diversas plataformas

Os participantes são:

• ComponentFactory-declara uma interface para operações que criam objetos dos componentes utilizados na aplicação;

• PocketFactory– classe concreta que implementa as operações que criam os objetos no formato do dispositivo cliente;

• PCFactory -classeconcreta que implementa as operações responsáveis por criar os objetos no formato do PC.

O padrão Abstract Factory possui os seguintes benefícios e desvantagens:

• Ele isola as classes concretas.

• Ele torna fácil a troca de famílias de produtos.

• Ela promove a harmonia entre produtos.

• É difícil de suportar novos tipos de produtos.

Factory Method

Definir uma interface para criar objetos, mas deixar que as subclasses decidem que classe instanciar. O Factory Method, também conhecido como construtor virtual, possibilita adiar a criação do objeto a subclasses.

Esse padrão é comumente utilizado pelos projetistas de software quando existe a necessidade de encapsular a criação de uma classe se isolando do conhecimento da classe concreta da aplicação cliente através de uma interface. Essa necessidade é comumente desejada por aqueles que trabalham no desenvolvimento de frameworks, que utilizam classes abstratas para definir e manter relacionamentos entre os objetos. Dessa forma os clientes implementam as funcionalidades esperadas pelo framework adicionando a lógica de negócio específica da aplicação, sem que o framework tenha o conhecimento de como e qual a lógica implementada pela aplicação para complementa-lo.

Um exemplo de utilização do padrão pode ser na construção de aplicações que tenha que dar suporte a diferentes implementações de persistência com o mínimo de re-trabalho.

A utilização do padrão Factory Method pode estar condicionada quando:

1. Uma classe não pode antecipar a classe/tipo de objetos que devem criar;

2. Uma classe especifique que suas subclasses tenham o conhecimento dos objetos que criam;

3. Classes que delegam responsabilidade para uma dentre várias subclasses auxiliares, e você quer obter o conhecimento de qual subclasse auxiliar que é a delegada.

Os participantes são:

• Product - define a interface de objetos que deverá ser criado pelo método fábrica;

• ConcreteProduct – implementação da interfaceProduct;

• Creator – declara o método fábrica o qual retorna um objeto do tipoProduct;

• ConcreteCreator – redefine(override) o método fábrica que retorna uma instância concreta da interfaceProduct.

O padrão Factory Method elimina a necessidade de anexar classes específicas das aplicações no código. O código lida somente com a interface deProduct; portanto ele pode trabalhar com qualquer implementação da classe que implementaProduct,definida pelo usuário.

Singleton

Garantir que um objeto terá apenas uma única instância, isto é, que uma classe irá gerar apenas um objeto e que este estará disponível de forma única para todo o escopo de uma aplicação.

Algumas aplicações têm a necessidade de controlar o número de instâncias criadas de algumas classes, seja pela necessidade da própria lógica ou por motivos de performance e economia de recursos.

Imagine o momento em que uma aplicação que exista simultaneamente num dispositivo móvel (Pocket PC, Celular, Palm) e num ambiente corporativo, necessite de um processo de sincronização entre as informações processadas no dispositivo móvel e na base corporativa. Ambas aplicações deveriam se comunicar com um objeto que deveria ser único para processar este sincronismo, a fim de evitar a possibilidade de criar dados na base.

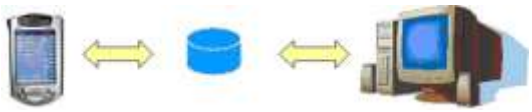

Figura 3. Elemento único de comunicação

O uso do padrão Singleton está condicionadoa:

• Quando for necessário manter num sistema, seja ele distribuído ou não, apenas uma instância de objeto e que o ponto de acesso para este seja bem conhecido (Ex. objeto responsável por um pool de impressão numa rede, gerenciador de janelas);

• Quando a única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usarem uma instância estendida sem alterar o seu código (visões polimórficas).

O desenvolvedores Delphi já utilizam o comportamento do padrão Singleton em suas aplicações, quando declaram variáveis globais na área de inicialização do projeto e depois reutilizam a instância dos objetos.TApplication, TCiipBoardsão exemplos de objetos que não teriam sentido existir mais de uma instância na aplicação e por isso assumem o comportamento do padrão Singleton.

A estrutura do padrão é apresentada na Figura 4.

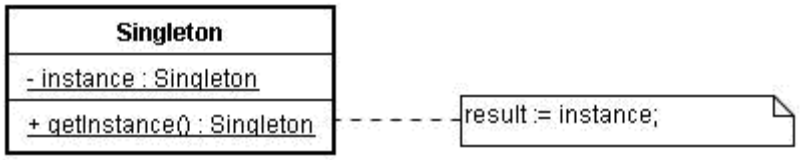

Figura 4. Estrutura do Sigleton

Os participantes são:

1. Singleton –define um método estático que permite aos clientes obterem o objeto único. Ele também será responsável pelo processo de criação do objeto.

Os benefícios que o padrão apresenta são:

• Acesso controlado a instância única;

• Espaço de nomes reduzido (diga não a proliferação de variáveis globais);

• Permite um refinamento de operações e representações;

• Permite um número variável de instâncias - O padrão possibilita que seja adotada a estratégica de criar mais de uma instância da classe, de forma controlada;

• Mais flexível do que as operações de classe;

Padrões Estruturais

Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os de classes utilizam a herança para compor interfaces ou implementações, eos de objeto ao invés de compor interfaces ou implementações, eles descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução o que não é possível com a composição estática (herança de classes).

Adapter

Converter a interface de uma classe por outra esperada pelos clientes. O que possibilita que classes com interfaces incompatíveis trabalhem em conjunto – ou que, de outra forma, seria impossível.Também conhecido comoWrapper(adaptador).

Algumas vezes, uma classe de um toolkit, projetada para ser reutilizada não condiz com a interface específica de um domínio requerida por uma aplicação.

O uso do padrão está condicionado a:

1. Usar uma classe existente, mas sua interface não corresponde à interface requerida;

2. Criar classes reutilizáveis que cooperam com classes não-relacionadas ou não previstas, ou seja, classes com interface inicialmente incompatível.

Os participantes são:

1. Target (Alvo)–define a interface específica do domínio do cliente;

2. Client (cliente) –colaboracom objetos compatíveis comTarget;

3. Adaptee (Adaptação) –interface existente de necessita de adaptação;

4. Adapter (Adaptador) –adapta a interfaceAdapteeà interfaceTarget.

Para adaptações de classes:

• Um adaptador de classe não funcionará quando quisermos adaptar uma classe e todas as suas subclasses;

• Permite aAdaptersubstituir algum comportamento deAdaptee, já queAdapteré uma subclasse.

Para adaptações de objetos:

• Permite a um únicoAdapteradaptar umAdapteee suas subclasses;

• Torna mais difícil redefinir o comportamento de umAdaptee. Conseguido através de uma subclasse deAdapteeque é referenciada porAdapter.

Padrões de Comportamento

Os padrões de comportamento se concentram nos algoritmos e atribuições de responsabilidades entre os objetos. Eles não descrevem apenas padrões de objetos ou de classes, mas também os padrões de comunicação entre os objetos.

Os padrões comportamentais de classes utilizam a herança para distribuir o comportamento entre classes, e os padrões de comportamento de objeto utilizam a composição de objetos em contrapartida a herança. Alguns descrevem como grupos de objetos cooperam para a execução de uma tarefa que não poderia ser executada por um objeto sozinho.

Template Method

Definir o esqueleto de um algoritmo em uma operação, postergando(deferring) alguns passos para subclasses. Template Method (gabarito de método) permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura do mesmo.

Agora imagine que você tenha de construir uma aplicação que possui uma determinada função da qual só é de conhecimento o algoritmo de execução, e o trabalho de codificação para realização da operação possa ser postergado. Observe a Figura 5.

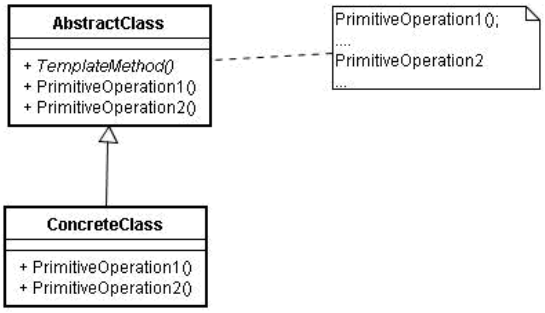

Figura 5. Estrutura do padrão

Sobre a aplicabilidade desse padrão temos:

• Para implementar as partes invariantes de um algoritmo e deixar para subclasses a implementação da parte variante;

• Para fatorar o comportamento semelhante entre subclasses numa superclasse evitando-se assim a duplicação de código;

• Para controlar extensões de classes com métodos "gancho";

Seus participantes são:

• AbstractClass –define as operações primitivas e abstratas que representam os passos de um algoritmo e implementa um método que invoca estas operações primitivas;

• ConcreteClass –implementa as operações específicas definidas na superclasse com o código específico.

Os métodos template são uma técnica fundamental para a reutilização de código. Eles conduzem a uma estrutura de inversão de controle “em inglêsInversion of Control IoC”ou princípio da dependência inversa,comumente conhecida como “o princípio de Hollywood”, ou seja: “não nos chame, nós chamaremos você”. Isto se refere a como uma classe-mãe chama as operações de uma subclasse, e não o contrário.