Skip to navigation

Marcos Brizeno

Desenvolvimento de Software #showmethecode

Prototype

dezembro 5, 2011

Mão na massa: Prototype

Para encerrar a série de posts sobre os padrões de projeto, o padrão Prototype!

Problema:

O padrão Prototype é mais um dos padrões de criação. Assim seu intuito principal é criar objetos. Este intuito é muito parecido com todos os outros padrões criacionais, tornando todos eles bem semelhantes.

Para explicitar ainda mais esta semelhança vamos analisar o mesmo problema apresentado na discussão sobre o padrão Factory Method.

O problema consiste em uma lista de carros que o cliente precisa utilizar, mas que só serão conhecidos em tempo de execução. Vamos analisar então como o problema pode ser selecionado utilizando o padrão Prototype.

Prototype

A intenção do padrão:

"Especificar tipos de objetos a serem criados usando uma instância protótipo e criar novos objetos pela cópia desse protótipo." [1]

Pela intenção podemos perceber como o padrão vai resolver o problema. Precisamos criar novos objetos a partir de uma instância protótipo, que vai realizar uma cópia de si mesmo e retornar para o novo objeto.

A estrutura do padrão inicia então com definição dos objetos protótipos. Para garantir a flexibilidade do sistema, vamos criar a classe base de todos os protótipos:

```
1  public abstract class CarroPrototype {
2      protected double valorCompra;
3
4      public abstract String exibirInfo();
5
6      public abstract CarroPrototype clonar();
7
8      public double getValorCompra() {
9          return valorCompra;
10     }
11
12     public void setValorCompra(double valorCompra) {
13         this.valorCompra = valorCompra;
14     }
15 }
```

Definimos a partir dela que todos os carros terão um valor de compra, que será manipulado por um conjunto de getters e setters. Também garantimos que todos eles possuem os métodos para exibir informações e para realizar a cópia do objeto.

Para exemplificar uma classe protótipo concreta, vejamos a seguinte classe:

```
1  public class FiestaPrototype extends CarroPrototype {
2
3      protected FiestaPrototype(FiestaPrototype fiestaPrototype) {
4          this.valorCompra = fiestaPrototype.getValorCompra();
5      }
6
7      public FiestaPrototype() {
8          valorCompra = 0.0;
9      }
10
11     @Override
12     public String exibirInfo() {
13         return "Modelo: Fiesta\nMontadora: Ford\n" + "Valor: R$"
14                 + getValorCompra();
15     }
16
17     @Override
18     public CarroPrototype clonar() {
19         return new FiestaPrototype(this);
20     }
21
22 }
```

Note que no início são definidos dois construtores, um protegido e outro público. O construtor protegido recebe como parâmetro um objeto da própria classe protótipo. Este é o chamado construtor por cópia, que recebe um outro objeto da mesma classe e cria um novo objeto com os mesmos valores nos atributos. A necessidade deste contrutor será vista no método de cópia.

O método exibirInfo() exibe as informações referentes ao carro, retornando uma string com as informações. Ao final o método de clonagem retorna um novo objeto da classe protótipo concreta. Para garantir que será retornado um novo objeto, vamos utilizar o construtor por cópia definido anteriormente.

Agora, sempre que for preciso criar um novo objeto FiestaPrototype vamos utilizar um único protótipo. Pense nesta operação como sendo um método fábrica, para evitar que o cliente fique responsável pela criação dos objetos e apenas utilize os objetos.

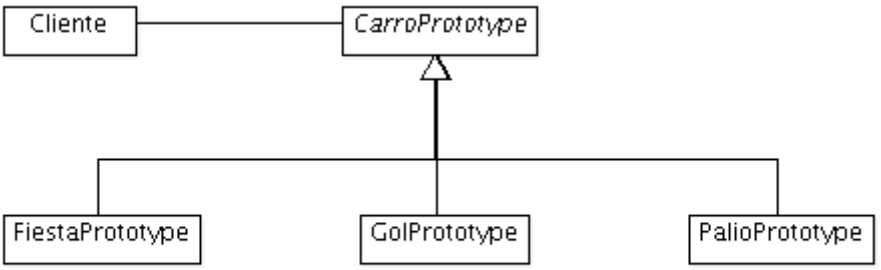
Para verificar esta propriedade, vamos analisar o código cliente a seguir, que faz uso do padrão prototype.

```
1  public static void main(String[] args) {
2      PalioPrototype prototipoPalio = new PalioPrototype();
3
4      CarroPrototype palioNovo = prototipoPalio.clonar();
5      palioNovo.setValorCompra(27900.0);
6      CarroPrototype palioUsado = prototipoPalio.clonar();
7      palioUsado.setValorCompra(21000.0);
8
9      System.out.println(palioNovo.exibirInfo());
10     System.out.println(palioUsado.exibirInfo());
11 }
```

Observe com atenção o cliente. Criamos dois protótipos de carros, cada um deles é criado utilizando o prótipo PalioPrototype instanciado anteriormente. Ou seja, a partir de uma instância de um protótipo é possível criar vários objetos a partir da cópia deste protótipo.

Outro detalhe é que, se a operação de clonagem não fosse feita utilizando o construtor de cópia, quando a chamada ao setValorCompra fosse feita, ela mudaria as duas instâncias, pois elas referenciariam ao mesmo objeto.

O diagrama UML que representa a estrutura do padrão Prototype utilizada nesta solução é o seguinte:



Um pouco de teoria

Inicialmente é fácil ver que o padrão Prototype oferece as mesmas vantagens que outros padrões de criação: esconde os produtos do cliente, reduz o acoplamento e oferece maior flexibilidade para alterações nas classes produtos.

A diferença básica deste padrão é a flexibilidade. Por exemplo: o cliente instancia vários protótipos, quando um deles não é mais necessário, basta removê-lo. Se é preciso adicionar novos protótipos, basta incluir a instanciação no cliente. Essa flexibilidade pode ocorrer inclusive em tempo de execução.

O padrão Prototype também poderia fazer uso do Abstract Factory. Imagine que uma classe instância todos os protótipos e oferece métodos para copiar estes protótipos, ela seria uma fábrica de famílias de produtos.

Os produtos do Prototype podem ser alterados livremente apenas mudando os atributos, como no exemplo onde criamos um palio novo e um palio usado. No entanto é preciso garantir que o método de cópia esteja implementado corretamente, para evitar que a alteração nos valores mude todas as instâncias.

O padrão Prototype leva grande vantagem quando o processo de criação de seus produtos é muito caro, ou mais caro do que uma clonagem. No lugar de criar um Proxy para cada produto, basta definir, no objeto protótipo, como será essa inicialização, ou parte dela.

Um detalhe que torna o Prototype único em relação aos outros padrões de criação é que ele utiliza objetos para criar os produtos, enquanto os outros utilizam classes. Dependendo da arquitetura ou linguagem/plataforma do problema, é possível tirar vantagem deste comportamento.

Código fonte completo

O código completo pode ser baixado no seguinte repositório Git: https://github.com/MarcosX/Padr-es-de-Projeto.

Os arquivos estão como um projeto do eclipse, então basta colocar no seu Workspace e fazer o import.

Se gostou do post compartilhe com seus amigos e colegas, senão, comente o que pode ser melhorado. Encontrou algum erro no código? Comente também. Possui alguma outra opinião ou alguma informação adicional? Comenta ai! 😀

Referências:

[1] GAMMA, Erich et al. Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos.

 Prototype Java, Padrões, Projeto 7 Comentários

__PRESENT