Prototype

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Divisão de Computação
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande

Agenda

Agenda

Introdução

- O padrão Prototype é um padrão de criação
  - Seu intuito principal é criar objetos.

Introdução

A intenção do padrão:

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

Agenda

Exemplo

Descrição

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 resolvido pelo o padrão de projetos Prototype.

Exemplo

Precisamos criar novos objetos a partir de uma instância protótipo, que vai realizar uma cópia de si mesmo e retornar para o novo objeto.

Exemplo

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 }
```

Exemplo

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.

Exemplo

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

```java
public class FiestaPrototype extends CarroPrototype {

    protected FiestaPrototype(FiestaPrototype fiestaPrototype) {
        this.valorCompra = fiestaPrototype.getValorCompra();
    }

    public FiestaPrototype() {
        valorCompra = 0.0;
    }

    @Override
    public String exibirInfo() {
        return "Modelo: Fiesta\nMontadora: Ford\n" + "Valor: R$"
                + getValorCompra();
    }

    @Override
    public CarroPrototype clonar() {
        return new FiestaPrototype(this);
    }

}
```

Exemplo

- **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 construtor será vista no método de cópia.

Exemplo

- 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

Exemplo

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 }
```

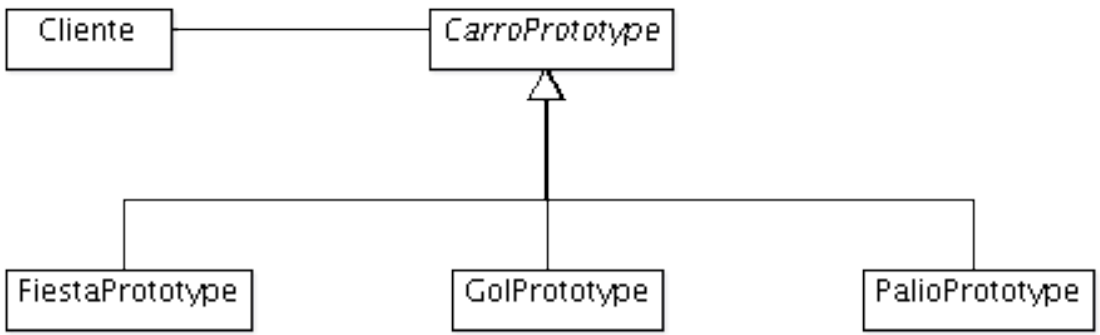
Exemplo

Observe com atenção o cliente

- Criamos dois protótipos de carros, cada um deles é criado utilizando o protótipo **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.

Exemplo

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



Agenda

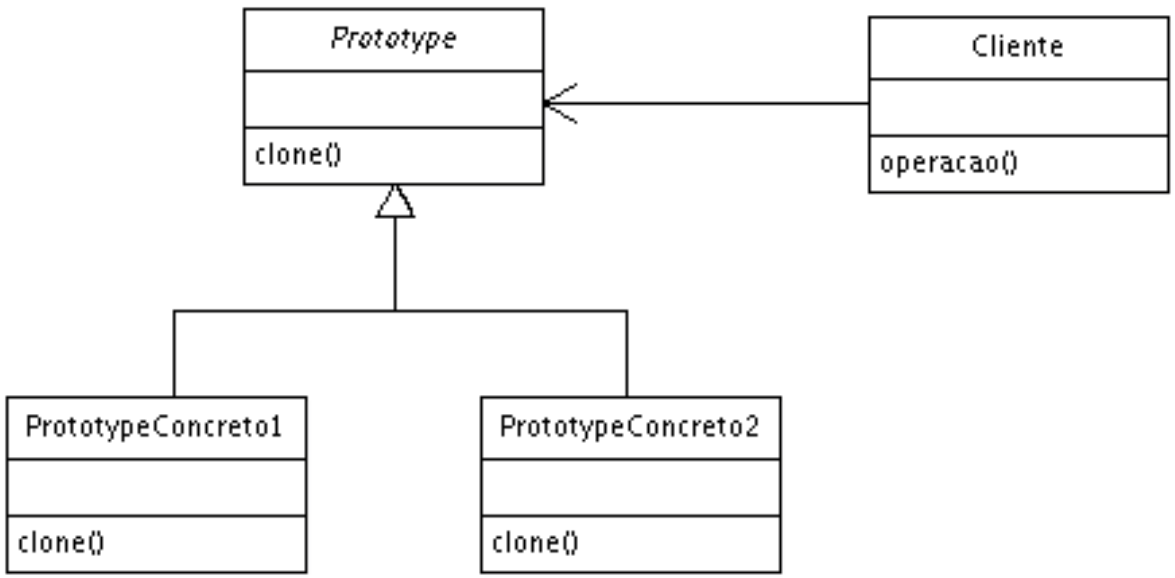
Aplicabilidade

Este padrão pode ser utilizado para:

- Evitar criar um novo objeto utilizando a palavra *new*, o que diminui o custo de memória.
- Em vez de o cliente implementar um código que utiliza o operador *new*, este utiliza o método *clone()* presente no protótipo que fica encarregado de clonar o novo objeto.

Agenda

Estrutura


Prototype
clone()
Cliente
operacao()
PrototypeConcreto1
clone()
PrototypeConcreto2
clone()

Estrutura

O padrão Prototype contém os seguintes elementos:

- **prototype** - uma classe que declara uma interface para objetos capazes de clonar a si mesmo.
- **prototype concreto** - implementação de um prototype;
- **cliente** - cria um novo objeto através de um prototype que é capaz de clonar a si mesmo.

Agenda

Características

- O padrão Prototype é utilizado frequentemente em linguagens estaticamente tipadas como C++ e Java, e menos frequentemente utilizadas em linguagens dinamicamente tipadas como Smalltalk.
- O padrão Prototype exige a implementação de uma operação de clonagem em cada uma das classes concretas do protótipo.

Características

- Esta tarefa pode ser inconveniente, no caso do reaproveitamento de classes preexistentes que não possuem tal operação, ou mesmo complexa, se for considerada a possibilidade de existirem referências circulares nos atributos de um objeto
  - um objeto possui um atributo que referencia um objeto que, por sua vez, referencia o objeto original.

Características

- O padrão Prototype é aplicado quando existe a necessidade de clonar, literalmente, um objeto.
- Quando a aplicação precisa criar cópias exatas de algum objeto em tempo de execução este padrão é altamente recomendado.

Características

O padrão Prototype pode ser útil em sistemas com as seguintes características:

- sistemas que utilizam classes definidas em tempo de execução;
- sistemas que possuem componentes cujo estado inicial possui poucas variações e onde é conveniente disponibilizar um conjunto preestabelecido de protótipos que dão origem aos objetos que compõem o sistema.

Características

- Uma das principais vantagens de sua utilização é quando a inicialização de um objeto pode se tornar custosa, e você quer fazer algumas pequenas variações ao inicializar.
  - Nesse contexto, o Prototype pode então evitar a **criação do zero** de novos objetos.

Agenda

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.

Um pouco de teoria

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.

Um pouco de teoria

- 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.

Um pouco de teoria

- 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

Agenda

Trabalho

Tema livre

Encontre alguma aplicação que seja resolvida com Prototype.

Prototype

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Divisão de Computação
Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande