Skip to navigation

Marcos Brizeno

Desenvolvimento de Software #showmethecode

setembro 24, 2011

Mão na massa: Singleton

Neste post apresentaremos o tão famoso padrão Singleton!

Problema

Como já vimos antes no padrões Abstract Factory (http://wp.me/p1Mek8-1h) e Factory Method (http://wp.me/p1Mek8-1c) é possível criar um objeto que fique responsável por criar outros objetos. Desta maneira nós centralizamos a criação destes objetos e podemos ter mais controle sobre eles.

Imagine o exemplo da fábrica de carros do padrão Factory Method (http://wp.me/p1Mek8-1c). A classe fábrica centraliza a criação de objetos carro. Por exemplo, se fosse necessário armazenar quantos carros foram criados, para elaborar um relatório de quais foram os carros mais vendidos, seria bem simples não? Bastaria adicionar um contador para cada tipo de carro e, ao executar o método que cria um carro, incrementar o contador referente a ele.

Vamos então ver como ficaria a nossa classe fábrica, para simplificar, apenas retornamos uma String para dizer que o carro foi criado:

```
public class FabricaDeCarro {
    protected int totalCarrosFiat;
    protected int totalCarrosFord;
    protected int totalCarrosVolks;

    public String criarCarroVolks() {
        return new String("Carro Volks #" + totalCarrosVolks++ + " criado
    }

    public String criarCarroFord() {
        return new String("Carro Ford #" + totalCarrosFord++ + " criado."
    }

    public String criarCarroFiat() {
        return new String("Carro Fiat #" + totalCarrosFiat++ + " criado."
    }

    public String gerarRelatorio() {
        return new String("Total de carros Fiat vendidos: " + totalCarros
                + "\nTotal de carros Ford vendidos: " + totalCarrosFord
                + "\nTotal de carros Volks vendidos: " + totalCarrosVolks
    }

}
```

A cada carro criado nós incrementamos o contador e exibimos a informação que o carro foi criado. No final adicionamos um método que gera um relatório e mostra todas as vendas.

O código cliente seria algo do tipo então:

```
public static void main(String[] args) {
    FabricaDeCarro fabrica = new FabricaDeCarro();
    System.out.println(fabrica.criarCarroFiat());
    System.out.println(fabrica.criarCarroFord());
    System.out.println(fabrica.criarCarroVolks());

    System.out.println(fabrica.gerarRelatorio());
}
```

Uma excelente solução não? Agora imagine que, em algum outro lugar do código acontece isso:

```
fabrica = new FabricaDeCarro();
System.out.println(fabrica.gerarRelatorio());
```

Todos os dados até o momento foram APAGADOS! Todas as informações estão ligadas a uma instância, quando alteramos a instância, perdemos todas as informações. Qual o centro do problema então?

Temos que proteger que o objeto seja instanciado. Como fazer isso?

Singleton

A intenção do padrão é esta:

"Garantir que uma classe tenha somente uma instância e fornece um ponto global de acesso para a mesma." [1]

Pronto, achamos a solução! Com o uso do padrão garantimos que só teremos uma instância da classe fábrica.

O padrão é extremamente simples. Para utilizá-lo precisamos apenas de:

Uma referência para um objeto fábrica, dentro da própria fábrica:

```
1  public class FabricaDeCarro{
2      public static FabricaDeCarro instancia;
3  }
```

Não deixar o construtor com acesso público:

```
1  public class FabricaDeCarro {
2
3      public static FabricaDeCarro instancia;
4
5      protected FabricaDeCarro() {
6      }
7  }
```

E por fim um método que retorna a referência para a fábrica:

```
 1  public class FabricaDeCarro {
 2
 3      public static FabricaDeCarro instancia;
 4
 5      protected FabricaDeCarro() {
 6      }
 7
 8      public static FabricaDeCarro getInstancia() {
 9          if (instancia == null)
10              instancia = new FabricaDeCarro();
11          return instancia;
12      }
13  }
```

Pronto, esta agora é uma classe Singleton. Ao proteger o construtor nós evitamos que esta classe possa ser instanciada em qualquer outro lugar do código que não seja a própria classe.

O código cliente ficaria da seguinte forma:

```
1  public static void main(String[] args) {
2      FabricaDeCarro fabrica = FabricaDeCarro.getInstancia();
3      System.out.println(fabrica.criarCarroFiat());
4      System.out.println(fabrica.criarCarroFord());
5      System.out.println(fabrica.criarCarroVolks());
6      System.out.println(fabrica.gerarRelatorio());
7  }
```

Perceba que, mesmo que em outra parte do código apareça algo do tipo:

```
1  fabrica = FabricaDeCarro.getInstancia();
2  System.out.println(fabrica.gerarRelatorio());
```

Não será perdido nenhuma informação, pois não houve uma nova instanciação. O método getInstancia() verifica se a referência é válida e, se preciso, instância ela e depois retorna para o código cliente.

Seria possível, no código cliente, nem mesmo utilizar uma referência para uma fábrica, veja o código a seguir:

```
1 System.out.println(FabricaDeCarro.getInstancia().gerarRelatorio());
```

Um pouco de teoria

Já mostramos em código que a maior vantagem do Singleton é unificar o acesso das instâncias. Além disso msotramos também que não é preciso nem mesmo criar referências para classes Singleton.

No C++ nós temos as tão temidas variáveis globais. Temidas não pela sua natureza, mas pelo uso que se faz de variáveis globais. As classes Singleton são uma excelente saída quando é necessário "globalizar" certos aspectos do programa.

Em concorrência é muito comum utilizar recursos compartilhados que precisam de um acesso unificado, geralmente utilizando monitores. Encapsular o recurso compartilhado em uma classe Singleton torna muito mais fácil sincronizar o acesso.

Outro fator bem positivo do padrão é que, ao mesmo tempo que nós unificamos o acesso aos recursos, nós compartilhamos todos eles! Lembre que não é necessário criar referências, ou seja, qualquer objeto/classe/método em qualquer lugar do seu programa tem acesso aos recursos.

Isto também pode ser visto como uma grande desvantagem, pois não é possível inibir o acesso a classe Singleton. Qualquer parte do código por chamar o método getInstance(), pois ele é estático, e ter acesso aos dados da classe.

Código fonte completo

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

 Padrões de Projeto, Singleton Java, Padrões, Projeto, Singleton 10 Comentários

10 comentários sobre “Mão na massa: Singleton”

1. outubro 28, 2012 às 1:24 PM
   Glaubert
   Muito bom seu site. está me ajudando muito com seus exemplos, muito claro e objetivo. Parabéns!

   Responder
2. outubro 3, 2013 às 10:36 AM
   Leinylson Fontinele
   Muito boa suas explicações. Só duas observações, o singleton não está no pacote e o incremento dos contadores deve vir antes como informado abaixo, ou na declaração inicializar com 1(um):

   public String criarCarroVolks() {
   return new String(“Carro Volks #” + ++totalCarrosVolks + ” criado.”);
   }

   public String criarCarroFord() {
   return new String(“Carro Ford #” + ++totalCarrosFord + ” criado.”);
   }

   public String criarCarroFiat() {
   return new String(“Carro Fiat #” + ++totalCarrosFiat + ” criado.”);
   }

   Responder
3. agosto 9, 2015 às 2:47 PM
   Mão na massa: Singleton | Pensamentos de Programador
   […] Mão na massa: Singleton. […]

   Responder
4. novembro 21, 2015 às 8:06 PM
   anderson
   Amigo o codigo do singleton n ta disponivel lá nos padrões do git :s

   Responder
5. dezembro 29, 2015 às 8:32 AM
   Roberto Licciardo
   Parabéns, muito bons seus artigos. Uma observação: sendo o construtor “protected” qualquer classe derivada poderia instanciar os objetos.

   Responder
6. fevereiro 17, 2018 às 6:05 PM
   Antonio Lira
   Muito Bom o Post, parabéns,…
   sou novo e programação e gostaria de tirar uma duvida, tenho um projeto com dez classes preciso implementar o singleton em cada uma delas, ou crio uma classe singleton e crio instancio as dez na classe singleton ???

   Responder
   fevereiro 17, 2018 às 9:11 PM

Marcos Brizeno
Oi Antonio, valeu pelo comentário!
Usar um ou vários singletons depende do que você quer alcançar. Se cada classe precisar de uma instância única, é melhor criar um singleton pra cada, já se a ideia é fornecer um ponto único de acesso a todas elas, daí criar um singleton só pode ser uma ideia melhor.
Mas na verdade acho que vale repensar o design pois utilizar uma instância global geralmente não é uma boa ideia devido ao alto acoplamento que isso cria. Se você criar 10 delas imagino que vai ficar bem estrutural e você não vai conseguir se beneficiar da Orientação a Objetos pra ter flexibilidade no seu design.

Responder

7. julho 16, 2018 às 10:02 PM

Anônimo
Marcos, boa noite.

No padrão Singleton, a classe de criação deve ser PRIVATE e não PROTECTED. Senão, é possível instaciar várias com … = new …

```
private FabricaDeCarro() {
}
```

Responder

8. julho 16, 2018 às 10:05 PM

Rinaldo Maria Faria
Marcos, boa noite.
No padrão SINGLETON, a classe de criação deve ser PRIVATE e não PROTECTED senão, é possível instaciar vários objetos com uso: classe var = new classe();

```
private FabricaDeCarro() {
}
```

Responder

9. novembro 14, 2018 às 10:50 PM

Leonardo
Muito obrigado pelos ensinamentos. Você está ensinando coisas que alguns autores usam dezenas de página para explicar e no fim não conseguem.

Responder

__PRESENT