Skip to navigation

Marcos Brizeno

Desenvolvimento de Software #showmethecode

Proxy

outubro 1, 2011

Mão na massa: Proxy

Problema

Suponha que um determinado programa faz uma conexão com o banco de dados para pegar algumas informações relativas aos usuários do sistema. Para simplificar o exemplo, considere a seguinte classe:

```java
public class BancoUsuarios{
    private int quantidadeDeUsuarios;
    private int usuariosConectados;

    public BancoUsuarios() {
        quantidadeDeUsuarios = (int) (Math.random() * 100);
        usuariosConectados = (int) (Math.random() * 10);
    }

    public String getNumeroDeUsuarios() {
        return new String("Total de usuários: " + quantidadeDeUsuarios);
    }

    public String getUsuariosConectados() {
        return new String("Usuários conectados: " + usuariosConectados);
    }
}
```

Nesta classe nós consultamos o número de usuários que estão conectados e o total de usuários do sistema. Poderiam existir diversas outras operações de consulta ao banco de dados, apenas simplificamos o exemplo.

O que queremos é implementar uma nova funcionalidade que permite verificar se um usuário possui ou não permissão para visualizar as informações do banco. A classe nos fornece uma maneira de acessar o banco de dados do sistema, no entanto ela não possui nenhuma proteção sobre quem está tentando acessá-la. Como podemos implementar um mecanismo de proteção?

Uma primeira solução seria adicionar os campos de usuário e senha e verificar se quem está tentando acessá-la possui as devidas permissões. O problema com essa abordagem é que, como precisaríamos alterar a própria classe, todo o resto do programa que usa essa classe teria que ser alterado também.

Uma solução melhor seria utilizar outra classe para verificar se o usuário possui permissão de acesso e só então exibir as informações do banco. Vamos mostrar esta solução utilizando o padrão Proxy.

Proxy

Vejamos a intenção do padrão Proxy:

“Fornecer um substituto ou marcador da localização de outro objeto para controlar o acesso a esse objeto.” [1]

Extamente a solução falada antes. Vamos utilizar uma classe substituta à classe BancoUsuarios para controlar o acesso.

Vamos então criar a classe Proxy. Para garantir que ela possa realmente substituir a classe original precisamos fazer com que a classe proxy extenda o comportamento da classe original:

```
1  public class BancoProxy extends BancoUsuarios {
2
3      protected String usuario, senha;
4
5      public BancoProxy(String usuario, String senha) {
6          this.usuario = usuario;
7          this.senha = senha;
8      }
9  }
```

Pronto, como a classe BancoProxy extende o comportamento de BancoUsuarios nós podemos utilizar um BancoProxy em qualquer lugar onde um BancoUsuarios era esperado. Além disso adicionamos aqui os campos de usuário e senha, que serão verificados nas operações.

Vamos então sobreescrever os métodos que buscam informação no banco. Agora, como temos o usuário e a senha, podemos realizar a verificação. Caso o usuário tenha permissão de acesso nós realizamos a chamado ao método da classe BancoUsuarios, caso contrário, retornamos nulo:

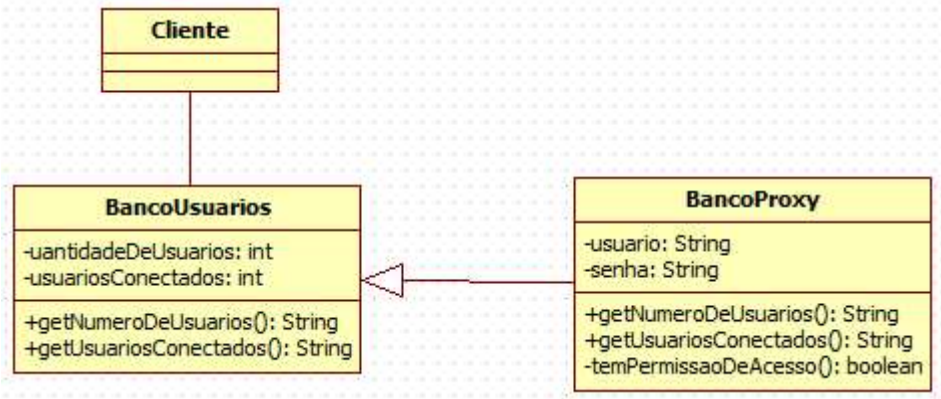
```java
public class BancoProxy extends BancoUsuarios {

    protected String usuario, senha;

    public BancoProxy(String usuario, String senha) {
        this.usuario = usuario;
        this.senha = senha;
    }

    @Override
    public String getNumeroDeUsuarios() {
        if (temPermissaoDeAcesso()) {
            return super.getNumeroDeUsuarios();
        }
        return null;
    }

    @Override
    public String getUsuariosConectados() {
        if (temPermissaoDeAcesso()) {
            return super.getUsuariosConectados();
        }
        return null;
    }

    private boolean temPermissaoDeAcesso() {
        return usuario == "admin" && senha == "admin";
    }
}
```

Pronto, com esta classe nós implementamos a segurança necessária sem precisar alterar o programa. Quando for necessário implementar um acesso seguro, utilizamos a classe Proxy, quando não, podemos utilizar a classe original sem nenhum problema. Vejamos por exemplo o seguinte código cliente:

```
1  public static void main(String[] args) {
2  System.out.println("Hacker acessando");
3  BancoUsuarios banco = new BancoProxy("Hacker", "1234");
4  System.out.println(banco.getNumeroDeUsuarios());
5  System.out.println(banco.getUsuariosConectados());
6
7  System.out.println("\nAdministrador acessando");
8  banco = new BancoProxy("admin", "admin");
9  System.out.println(banco.getNumeroDeUsuarios());
10 System.out.println(banco.getUsuariosConectados());
11 }
```

Veja que utilizamos uma referência a um objeto do tipo BancoUsuarios, mas instanciamos um objeto do tipo BancoProxy. Isto mostra o que falamos antes sobre utilizar um proxy em qualquer lugar que um objeto original seria esperado. Caso houvesse um método que tivesse como entrada um objeto BancoUsuarios poderíamos sem nenhum problema utilizar um objeto BancoProxy para manter a segurança.

O diagrama UML para este caso de uso do Proxy seria bem simples:



Um pouco de teoria

Antes de falar sobre o padrão proxy é preciso comentar um pouco sobre os tipos de proxy que podem ser utilizados:

- Protection Proxy: esse é o tipo de proxy que utilizamos no exemplo. Eles controlam o acesso aos objetos, por exemplo, verificando se quem chama possui a devida permissão.
- Virtual Proxy: mantem informações sobre o objeto real, adiando o acesso/criação do objeto em si. Como exemplo podemos citar o caso mostrado em [1], onde é utilizado um Proxy que guarda algumas informações sobre uma imagem, não necessitando criar a imagem em si para acessar parte de suas informações.
- Remote Proxy: fornece um representante local para um objeto em outro espaço de endereçamento. Por exemplo, considere que precisamos codificar todas as solicitações enviadas ao banco do exemplo anterior, utilizaríamos um Remote Proxy que codificaria a solicitação e só então faria o envio.
- Smart Reference: este proxy é apenas um susbtituto simples para executar ações adicionais quando o objeto é acessado, por exemplo para implementar mecanismos de sincronização de acesso ao objeto original.

Cada proxy implicaria em um design diferente. No exemplo que citamos, consideramos que a classe original não poderia ser modificada, por isso foi necessário extender seu comportamento. No entanto, caso essa não fosse uma restrição poderíamos criar uma interface comum de acesso ao banco e utilizar ela em nosso programa, assim a implementação do programa ficaria independente da implementação do banco em si (segura ou não).

A principal vantagem de utilizar o Proxy é que, ao utilizar um substituto, podemos fazer desde operações otimizadas até proteção do acesso ao objeto. No entanto isto também pode ser visto como um problema, pois, como a responsabilidade de um proxy não é bem definida é necessário conhecer bem seu comportamento para decidir quando utilizá-lo ou não.

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.

Padrões de Projeto, Proxy Java, Padrões, Projeto, Proxy 2 Comentários

__PRESENT