Universidade do Minho

Mestrado em Engenharia Informática

Arquiteturas Aplicacionais

Ano lectivo 2014/2015

Trabalho Prático

Design Patterns

Autores:
José Morgado (pg27759)
Luís Miguel Pinto (pg27756)
Pedro Carneiro (pg25324)

Braga, 23 de Março de 2015

Resumo

Conteúdo

1 Introdução **3**

2 Patterns de Criação **4**
2.1 Singleton 4

3 Patterns Estruturais **5**
3.1 Facade 5

4 Patterns de Comportamento **7**

5 Conclusão **8**

6 Referências **9**

1 Introdução

2 Patterns de Criação

2.1 Singleton

Este *pattern* é utilizado quando se pretende a existência de apenas uma instância de uma classe. Para uma classe ser *singleton* deve garantir-se que há apenas uma instância e um ponto de acesso à mesma.

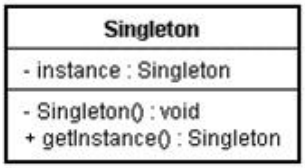

Figura 1: Diagrama de classe

No diagrama de classe apresentado anteriormente existe um atributo *singleton* que é do tipo da própria classe. Nos métodos da classe podemos verificar a presença do construtor de classe, *Singleton()*, que é privado. Ora, para que a classe seja instanciada é necessário utilizar o método estático *getInstance()*, assim pode ser acedido por qualquer outra classe.

Exemplo 1: Solução de implementação

```
public class MyClass {
  private static MyClass instance = null;

  private MyClass() {
  }

  public static MyClass getInstance() {
    if(instance==null) {
      instance = new MyClass();
    }
    return MyClass.instance;
  }
}
```

Com a implementação anterior assegurasse que ao tentar-se criar duas instâncias da mesma classe isso não é possível, pois o atributo *instance* já não está a *null*.

Vantagens É um *Pattern* simples e é utilizado quando se pretende ter um acesso único e global para organizar os recursos.

Desvantagens A utilização exagerada deste *pattern* pode provocar depedências muito fortes entre os objetos e dificultar alterações de configurações futuras.

3 Patterns Estruturais

3.1 Facade

O *pattern* Facade esconde toda a complexidade de uma ou mais classes do sistema e fornece uma interface, que o cliente pode aceder para utilizar o sistema.

Ao se utilizar uma Facade, que implementa uma interface, pretende-se reduzir o nível de complexidade no subsistema, aumentando a facilidade de utilização.

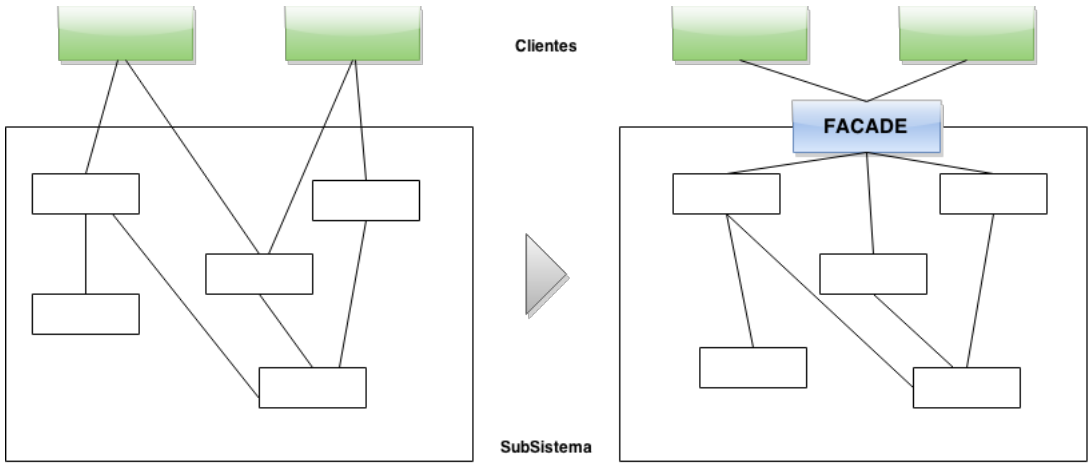

Figura 2: Antes e depois da utilização do Facade

Como vemos na figura anterior, este *pattern* é definido através de uma única classe que fornece os métodos necessários ao cliente e envia os pedidos deste às classes do subsistema.

Aplicação
Utilizar Facade quando:

- Se deseja fornecer uma interface simples para um sistema complexo.
- Existe muitas dependências entre clientes e as classes de implementação de uma abstração.
- Se pretende ter uma estrutura em camadas no sistema.

Colaboração
Os clientes comunicam com o subsistema através do envio de pedidos ao Facade, que redireciona os mesmos para as classes apropriadas do subsistema. Ainda que, os clientes que utilizam este *pattern* não tem que aceder aos objetos do seu subsistema diretamente.

Vantagens

- A utilização deste *pattern* permite diminuir as ligações entre os clientes e o sistema.

- Para adicionar novas funcionalidades ao sistema seria necessário alterar apenas o Facade, ao invés de alterar os vários objetos do sistema, sem afetar o cliente.

Implementação

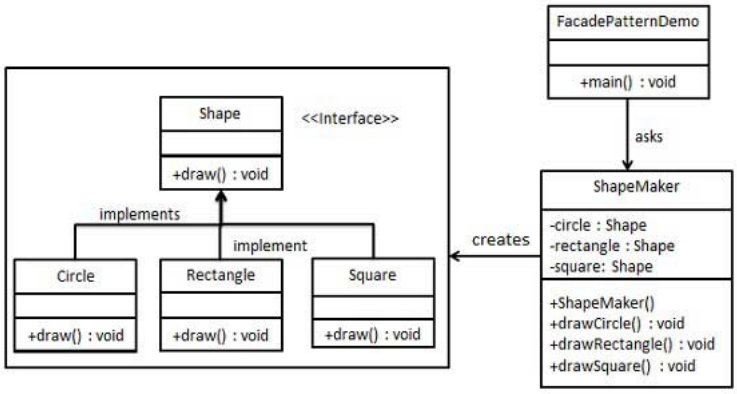

Figura 3: Antes e depois da utilização do Facade

Nós vamos criar uma interface Forma e classes concretas implementando a interface Shape. A ShapeMaker classe de fachada é definido como um próximo passo.

4 Patterns de Comportamento

5 Conclusão

6 Referências

Referências

[1] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra. Head First Design Patterns. O'Reilly Media, 2004.

[2] http://wiki.portugal-a-programar.pt/dev_geral:java:padrao_singleton

[3] http://www.devmedia.com.br/padrao-de-projeto-singleton-em-java/26392

[4] http://www.tutorialspoint.com/design_pattern/facade_pattern.htm