Padrões de Projeto

Padrões Criacionais

Builder

Objetivo

- Separar a construção de um objeto complexo da sua representação para que o mesmo processo de construção possa criar diferentes representações.

- Construir um objeto complexo passo a passo.

Exemplo

Se quisermos construir uma casa, devemos levar em consideração os atributos básicos comuns à toda casa como:

- 4 paredes
- Teto
- Piso
- Porta

Mas também devemos considerar atributos mais específicos:

Se uma casa pode ter:

- Jardim, Garagem, Piscina, Estátuas, Automação, Mais de um andar, etc.
- E mais uma infinidade de combinações dessas configurações do objeto casa.

Dessa forma, cada configuração englobaria seus respectivos métodos.

No entanto, criar uma subclasse para cada combinação possível deixaria o programa muito complexo e poluído.

E nem todo objeto necessita de todas as configurações existentes.

Solução

O Padrão Builder propõe mover o código de construção do objeto para um novo objeto “builder” e separar a construção em etapas.

Assim, podemos criar um construtor único chamando apenas as etapas necessárias para a produção de uma configuração específica de um objeto.

Exemplo Prático

O padrão Builder é muito usado em bibliotecas do Java, um exemplo é a classe StringBuilder que permite:

- Criar e armazenar dados de strings dinamicamente, criando variáveis de Strings modificáveis.
- Concatenar strings, alocando o método append.

```
public class TestaStringBuilder {
         public static void main(String[] args) {

  StringBuilder nomes = new StringBuilder();
    nomes.append("Carlos>").append("Maria>").append("José>").append("Renata");

  System.out.println(nomes.toString());

        }
}
```

Singleton

Objetivo

- Garantir que uma classe tenha apenas uma instância.
- Prover um ponto de acesso global para essa instância.

Problemas

- As vezes faz-se necessário controlar as instâncias de uma classe, por exemplo, para controlar o acesso a algum recurso compartilhado.
- Fornecer acesso global para uma instância é perigoso já que, assim como com as variáveis globais, ela corre o risco de ser sobrescrita.

Solução

- A classe Singleton declara um método estático que retorna a mesma instância da sua própria classe.
- Implementar o construtor privado, para prevenir que outros objetos usem o operador *new* com a classe Singleton.
- A única forma de obter o objeto Singleton deve ser através do construtor Singleton.

Prós vs Contras

- Garante-se que a classe só terá uma instância
- Tem um acesso global para a instância, sem o risco de ser sobrescrito.
- O objeto Singleton só é inicializado quando pedido a primeira vez.

- Viola o princípio da responsabilidade única, já que resolve dois problemas de uma vez.
- Dificulta a realização de testes unitários. Já que o construtor Singleton é privado, não é possível simular o objeto Singleton.

Exemplos práticos

- Classes Logger

Essas classes são normalmente implementadas com padrão Singleton já que fornece um acesso logging global em toda a aplicação sem que seja necessário criar um objeto cada vez que uma operação logging for realizada.

- Banco de dados

A lógica do Singleton garante que a conexão com o banco de dados seja aberta apenas uma vez, uma vez o objeto criado, a conexão é estabelecida e dali em diante será retornado apenas o objeto da conexão. Então ficará na memória enquanto o *script* estiver rodando.