



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 (construtor) 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.



```
1 import logging
2
3 class MyLogger:
4     _logger = None
5
6     def __new__(cls):
7         if cls._logger is None:
8
9             print("Logger new")
10            cls._logger = logging.getLogger("logger_name")
11            cls._logger.setLevel(logging.DEBUG)
12            formatter = logging.Formatter(
13                '%(asctime)s \t [%(levelname)s | %(filename)s:%(lineno)s] > %(message)s')
14
15            streamHandler = logging.StreamHandler()
16
17            streamHandler.setFormatter(formatter)
18
19            cls._logger.addHandler(streamHandler)
20
21            return cls._logger
22
23 if __name__ == "__main__":
24     logger1 = MyLogger()
25     logger1.info("Hello, Logger")
26     logger2 = MyLogger()
27     logger2.warning("bug occurred")
```

- 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.

- Janela de jogo

A janela de um jogo é um Singleton já que é esperado que, uma vez o objeto janela já exista, se o usuário tentar abrir uma nova janela do mesmo jogo seja retornado a janela pré existente.