Design Patterns

REVIEW

UNIVAS

DEFINIÇÃO

Efetivamente, portanto, padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular.

UNIVAS

INTRODUÇÃO

Quais os benefícios principais conferidos pelos padrões de projeto?

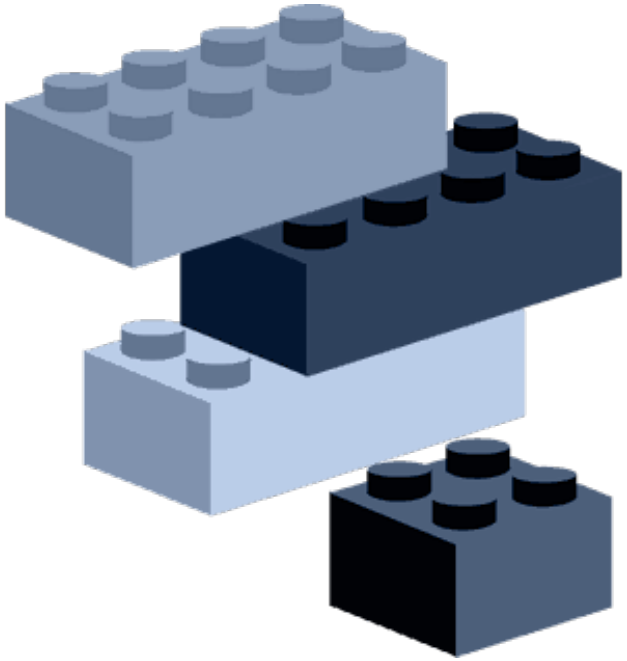
1. Não precisa reinventar a roda;
2. São universais;
3. Evita refatoração desnecessária;
4. Reutilização de código;
5. Abstrai e nomeia partes particulares do código;
6. Facilitam a criação de testes unitários.

Em outras palavras, ajudam o projetista a obter mais rapidamente um projeto adequado.

		Propósito		
		De criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method (112)	Adapter (class) (140)	Interpreter (231) Template Method (301)
	Objeto	Abstract Factory (95) Builder (104) Prototype (121) Singleton (130)	Adapter (object) (140) Bridge (151) Composite (160) Decorator (170) Façade (179) Flyweight (187) Proxy (198)	Chain of Responsibility (212) Command (222) Iterator (244) Mediator (257) Memento (266) Observer (274) State (284) Strategy (292) Visitor (305)

UNIVAS

ExClasse
+ nome : string - cpf : inteiro # email: string
+ getNome() : string + getCpf() : inteiro + setNome(nome : string) + setCpf(cpf : inteiro)



SINGLETON

APRESENTAÇÃO E PRÁTICA DO PP

UNIVAS

INTENÇÃO

Garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma.

UNIVAS

MOTIVAÇÃO

Em alguns casos, é importante ter uma, e somente uma, instância para não fazer concorrência incorreta.

Ex.: spooler de impressoras, sistema de arquivos, gerenciador de janelas, logger.

UNIVAS

MOTIVAÇÃO

Uma variável global torna um objeto acessível, mas não impede você de instanciar múltiplos objetos.

O padrão Singleton sugere que a classe possa garantir que nenhuma outra instância seja criada, bem como fornecer o meio para acessar sua única instância.

UNIVAS

QUANDO USAR?

- Quando for preciso haver apenas uma instância de uma classe, e essa instância tiver que dar acesso aos clientes através de um ponto bem conhecido;

- Quando a única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usar uma instância estendida sem alterar o seu código.

UNIVAS

ESTRUTURA

SingleObject
- Instance : singleton anyData
- <<constructor>> SingleObject() + get() : Singleton anyOperation()

CONSEQUÊNCIAS

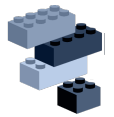
1. Acesso controlado à instância única. Como a classe Singleton encapsula a sua única instância, possui controle total sobre como e quando os clientes a acessam.

2. Espaço de nomes reduzido. O padrão Singleton representa uma melhoria em relação ao uso de variáveis globais. Ele evita a poluição do espaço de nomes com variáveis globais que armazenam instâncias únicas.

CONSEQUÊNCIAS

3. Permite um refinamento de operações e da representação. A classe Singleton pode ter subclasses. Você pode configurar a aplicação com uma instância da classe de que necessita em tempo de execução.

4. Permite um número variável de instâncias. O padrão torna fácil mudar de idéia, permitindo mais de uma instância da classe Singleton. Além disso, você pode usar a mesma abordagem para controlar o número de instâncias que a aplicação utiliza. Somente a operação que permite acesso à instância de Singleton necessita ser mudada.



CONSEQUÊNCIAS

5. Mais flexível do que operações de classe. Uma outra maneira de empacotar a funcionalidade de um singleton é usando operações de classe (ou seja, funções-membro estáticas em C++ ou métodos de classe em Smalltalk). Porém, as técnicas de ambas as linguagens tornam difícil mudar um projeto para permitir mais que uma instância de uma classe. Além disso, as funções- membro estáticas em C++ nunca são virtuais, o que significa que as subclasses não podem redefini-las polimorficamente.

UNIVAS

EXEMPLIFICANDO: STEP 1 – CRIAR A CLASSE SINGLETON

```java
package pp_singleton_ex1;

public class SingleObject {
  //criar o objeto do SingleObject
    private static SingleObject instance = null;

  //tornar o construtor privado para que a classe não possa
  //ser instanciada
    private SingleObject() {}

  //torne o objeto disponível
    public static SingleObject getInstance() {
        if(SingleObject.instance == null) {
            SingleObject.instance = new SingleObject();
        }
        return instance;
    }

    public void showMessage() {
        System.out.println("Hello Singleton!");
    }
}

```

EXEMPLIFICANDO: STEP 2 – OBTER O ÚNICO OBJETO DA CLASSE SINGLETON

UNIVAS

```
1  package pp_singleton_ex1;
2
3  public class SingletonDemo {
4    public static void main(String[] args) {
5      //pegue o único objeto disponível
6        SingleObject object = SingleObject.getInstance();
7
8      //mostre a mensagem
9        object.showMessage();
10
11   }
12 }
13
```

UNIVAS

SÓ COISA BOA?
PONTOS NEGATIVOS

1. Mais difícil de testar;
2. Viola o princípio da responsabilidade única;
3. Requer tratamento especial em caso de concorrência;
4. Erich Gamma afirmou que removeria se fosse escrever o livro novamente (bit.ly/nosingleton).

UNIVAS

SÓ COISA BOA?
PONTOS NEGATIVOS

A abordagem do Singleton fere o SOLID (princípios da programação OO e design de código), por isso não é tão bem vista atualmente.

Como recurso a ela, existe uma abordagem sugerida chamada Monostate.

Leitura complementar: bit.ly/mais_sobre_o_singleton

UNIVAS

Obrigado!

Professor Gustavo Dias
luizdias@univas.edu.br