Command Design Pattern

em JavaScript

Design Patterns

- Por onde começou
- A presença no nosso dia a dia
- O uso na programação
- Categorias de DP
- Por que usar?

Christopher Alexander.

A Pattern Language

Towns · Buildings · Construction

Christopher Alexander

Sara Ishikawa · Murray Silverstein

WITH

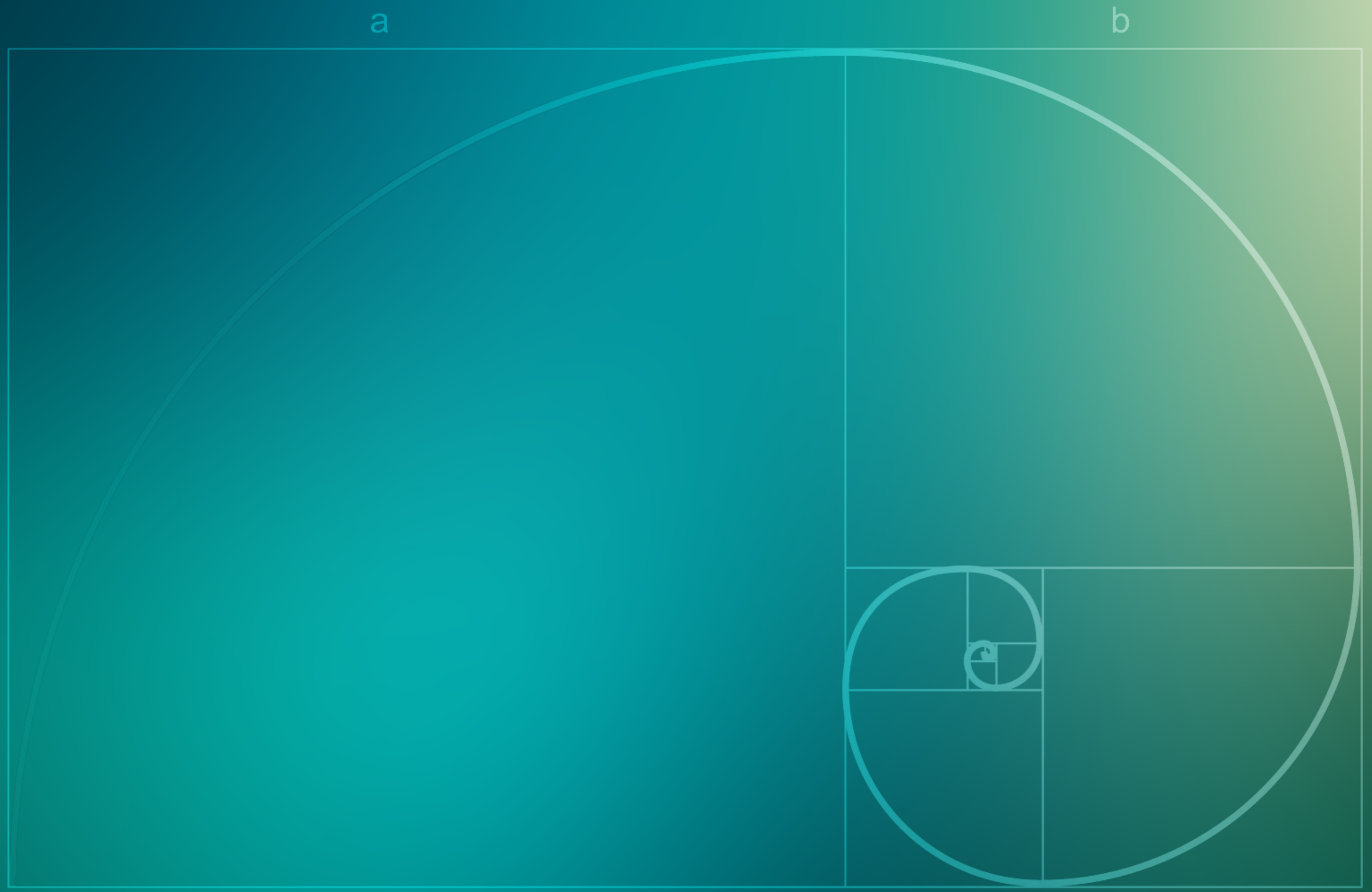
Max Jacobson · Ingrid Fiksdahl-King

Shlomo Angel

Soluções de reutilização de
código para problemas comuns
que ocorrem no desenvolvimento
de um projeto

Patterns estão presentes *em tudo*

1 1 2 3 5 8 13 21 34 55 89 144



$$\frac{a+b}{a} = \frac{a}{b} = \varphi \approx 1{,}61803$$

Na programação

São melhores praticas formalizadas, desenvolvidas e testadas por vários programadores com a finalidade de resolver os diferentes problemas

Como elas se *categorizam?*

Creational Design Patterns

Lidam com mecanismos de criação de objeto

Abstract

Constructor

Prototype

Factory

Structural patterns

Se preocupam com a composição do objeto

Adapter

Facade

Proxy

Decorator

Behavioral patterns

Se concentram na melhoria e simplificação da comunicação entre diferentes objetos

Command

Mediator

Iterator

Observer

Por que *utilizar?*

- Tempo
- Qualidade de código
- Melhores práticas

Command *Pattern*

1. Conceitos
2. Exemplo de uso
3. Diagrama
4. Implementação

1. Implementação
2. Diagrama
3. Exemplo de uso
4. Conceitos

Implementação

```
(function(){

    var Cusco = {

        // latir
        latir: function( nome ){
            return '0 cusco ' + nome + ' latiu';
        },

        // andar
        andar: function( nome ){
            return '0 cusco ' + nome + ' andou!';
        },

        // correr
        correr: function( nome ){
            return '0 cusco ' + nome + ' correu!';
        },

        // sentar
        sentar: function( nome ){
            return '0 cusco ' + nome + ' sentou!';
        }

    };

})();
```

Implementação

```
Cusco.execute = function ( metodo ) {
    return Cusco[metodo] && Cusco[metodo].apply( Cusco, [].slice.call( arguments, 1 ) );
};
```

Implementação

```
Cusco.execute( 'latir', 'Tob' );
Cusco.execute( 'andar', 'Tob' );
Cusco.execute( 'correr', 'Tob' );
Cusco.execute( 'Sentar', 'Tob' );
```

Implementação

```
(function(){

    var Cusco = {
        // latir
        latir: function( nome ){
            return 'O cusco ' + nome + ' latiu';
        },

        // andar
        andar: function( nome ){
            return 'O cusco ' + nome + ' andou!';
        },

        // correr
        correr: function( nome ){
            return 'O cusco ' + nome + ' correu!';
        },

        // sentar
        sentar: function( nome ){
            return 'O cusco ' + nome + ' sentou!';
        }
    };

    Cusco.execute = function ( metodo ) {
        return Cusco[metodo] && Cusco[metodo].apply( Cusco, [].slice.call( arguments, 1 ) );
    };

    Cusco.execute( 'latir', 'Tob' );
    Cusco.execute( 'andar', 'Tob' );
    Cusco.execute( 'correr', 'Tob' );
    Cusco.execute( 'Sentar', 'Tob' );

})();
```

Diagrama

Client: São as requisições;

Receiver: O nosso objeto Cusco

Command: O método "*execute()*"

Invoker: Métodos do objeto

Exemplo de uso

Imagine uma aplicação que suporta as ações de recortar, copiar e colar. Estas ações podem ser disparadas de diferentes maneiras em nosso app:

- Por um menu do sistema
- Clicando com botão direito do mouse e copiando
- Atalho do teclado.

Conceitos

- Um objeto é na real a representação de um verbo

- Nasceu da necessidade de se emitir requisições para objetos sem necessariamente ter conhecimento sobre a operação que esta sendo requisitada ou o do objeto que recebe essa requisição.

- Pode receber a solicitação de diferentes requisições por parâmetro
- Pode enfileirar comandos bem como a possibilidade de desfazer as operações.
- Separar as responsabilidades do objeto que emite uma requisição dos objetos que realmente executam e processam essa solicitação

Conclusão

Me agradou muito a maneira como ele abstrai a execução de métodos na aplicação, possibilitando uma maior versatilidade e até mesmo uma ferramenta poderosa em aplicações de grande escala, principalmente pelo poder de controlar a chamada de métodos.

Referências

1. Learning JavaScript Design Patterns - Addy Osmani https://addyosmani.com/resources/essentialjsdesignpatterns/book/

2. JavaScript Design Patterns - dofactory http://www.dofactory.com/javascript/design- patterns

3. Design Patterns - SourceMaking https://sourcemaking.com/design_patterns

4. JavaScript Design Patterns: Command - Joe Zimmerman https://www.joezimjs.com/javascript/javascript-design-patterns-command/

5. A Idiomatic JavaScript Command Design Pattern - Derick Bailey https://jsfiddle.net/derickbailey/HsDNG/