

Design Patterns

aplicados ao desenvolvimento Android



Prof. Pedro Henrique

Introdução

O que é um *Design Pattern*?

- Uma idéia de solução para um problema que ocorre com frequência;
- Uma descrição, um modelo de como resolver um problema;
- Pode ser usado em diversas situações diferentes;
- São melhores práticas documentadas para resolver problemas;
- Não é algo que pode ser diretamente transformado em código. Uma análise é exigida antes da aplicação

Design Pattern

Características de um Padrão de Projeto

- Um padrão de projeto, normalmente, possui 4 características:
 - Nome
 - Problema
 - Solução
 - Consequências

Design Pattern

Tipos de Padrão de Projeto

- Existem três tipos principais de Design Pattern:
 - Estruturais
 - Comportamentais
 - De criação

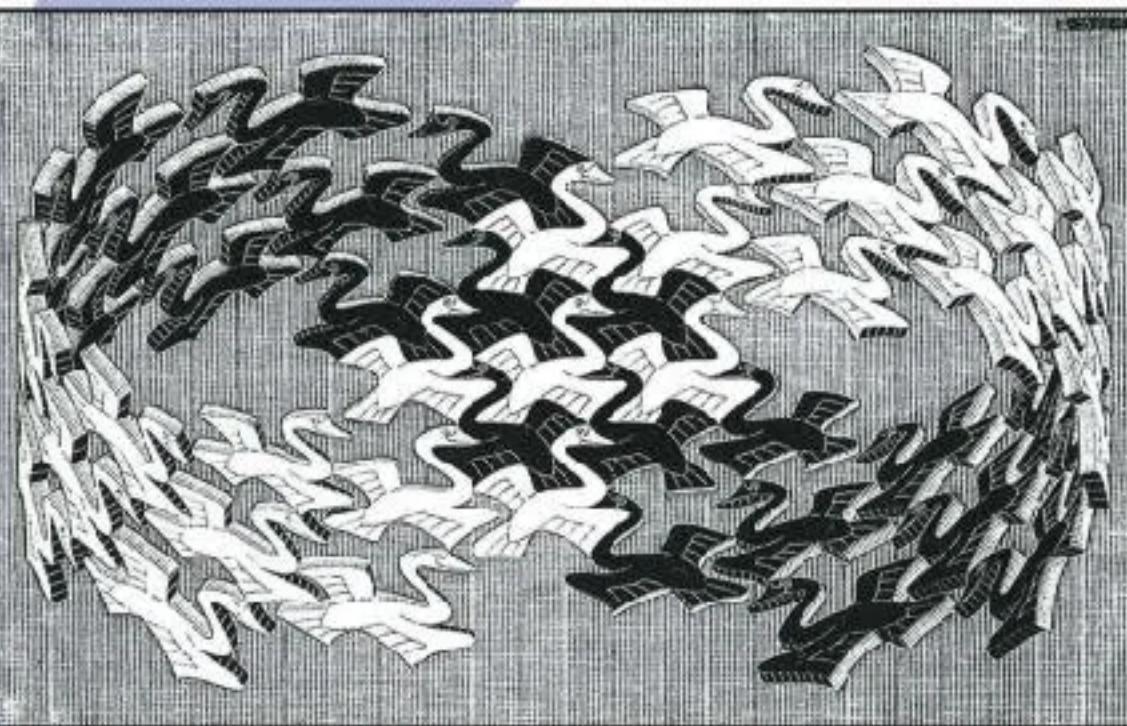
Design Pattern

De onde eles vêm?

- A história começa no final da década de 1970, quando Christopher Alexander definiu, em sua obra, quais características um padrão de projeto deve ter e como ele deve ser descrito;
- Só no final da década de 1980 foram propostos os primeiros padrões de projeto, à época aplicados à linguagem Smalltalk (musa inspiradora do Objective-C)
- A popularidade só veio em 1995, com a publicação do livro **Padrões de Projetos: Soluções Reutilizáveis de Software Orientados a Objeto** pelos autores que ficaram conhecidos como a gangue dos quatro (*Gang of Four - GoF*)

Padrões de Projeto

Soluções reutilizáveis de software orientado a objetos



© 1994 M.C. Escher / Cordon Art - Baam - Holland. Todos os direitos reservados.

ERICH GAMMA
RICHARD HELM
RALPH JOHNSON
JOHN VLISSIDES

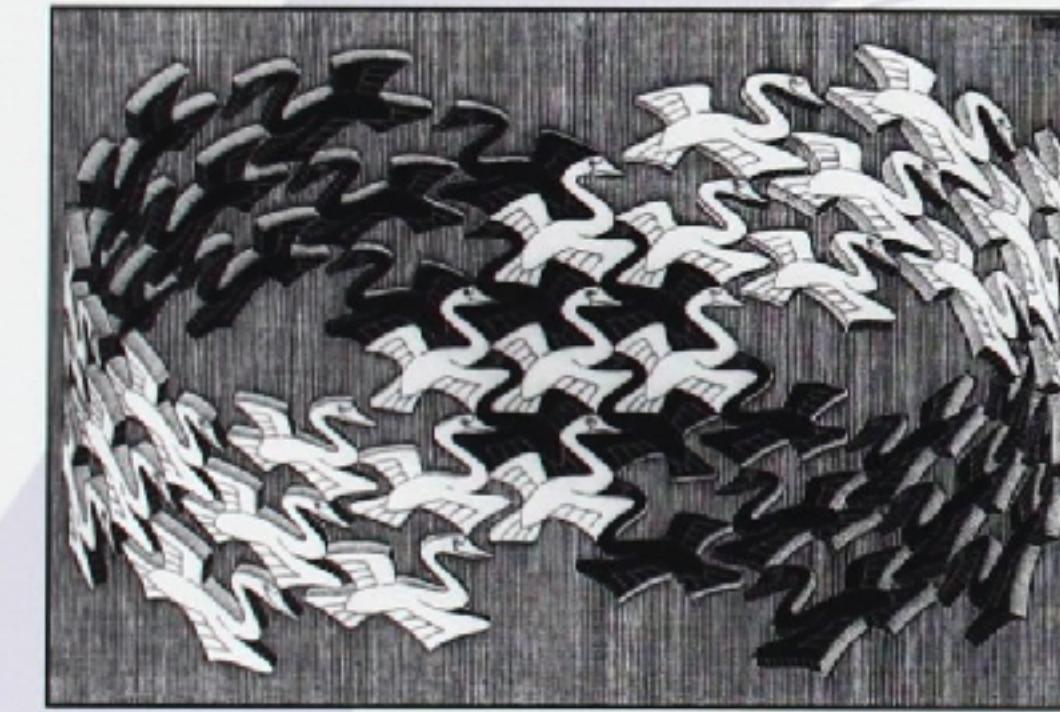


Design Patterns

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baam - Holland. All rights reserved.

Foreword by Grady Booch



* ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Aplicando *Design Patterns*

ao mundo Android

Builder

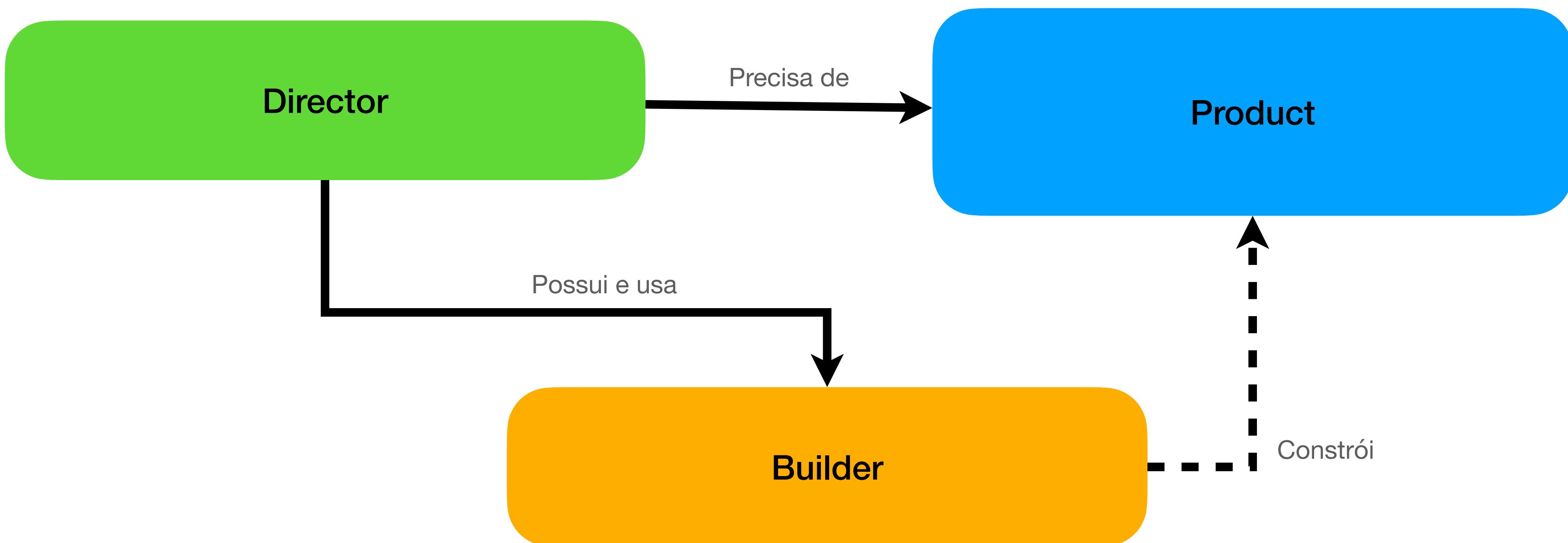
Um padrão de Criação



**Quando construir for difícil,
chame o pedreiro!**

Builder

Como funciona?



Builder

Quando usar?

- O principal objetivo é usar quando se precisa criar um objeto complexo, que necessita de várias etapas, de forma que usar um construtor seria contra-productivo;
- O builder, portanto, abstrai essa complexidade, permitindo ao Director executar as etapas na ordem que bem entender ;
- Vamos ao exemplo...

Singleton

Um padrão de criação



There can be only one

Singleton

O padrão *High Lander*

- Restringe a classe a uma e apenas uma instância;
- Desta forma, todas as referências apontam sempre para uma mesma instância;
- Este é um *Design Pattern* extremamente comum no iOS, especialmente na sua variação “*Shared Singleton*”;
- Este é, talvez, o padrão mais suscetível ao abuso e por isso é considerado por muitos autores um *anti-pattern*;

Singleton

Quando usar?

- Quando ter mais de uma instância pode ser problemático (ex: concorrência);
- Quando simplesmente não for lógico ter mais de uma instância (é aqui que mora o problema do abuso);
- Quando uma única instância fizer sentido na maior parte do tempo (não todo o tempo), o *Shared Singleton* pode ser empregado;
 - No *Shared Singleton*, uma instância padrão oferece aquilo que é útil na maior parte do tempo;
 - A diferença é que, com o *Shared Singleton*, a criação de outras instâncias é permitida para cobrir outros cenários;

Singleton

Who wants to live forever?

- Como já mencionado, o Singleton é muito suscetível ao abuso;
- Então, antes de adotar o Singleton como solução, analise cuidadosamente outras alternativas;
- Se, após uma boa análise, você determinar que precisa de um Singleton, considere a adoção do Shared Singleton;
- Uma das principais fontes de problema do Singleton é a testabilidade, já que com uma única instância, não se pode variar estado sem comprometer outros cenários;

Strategy

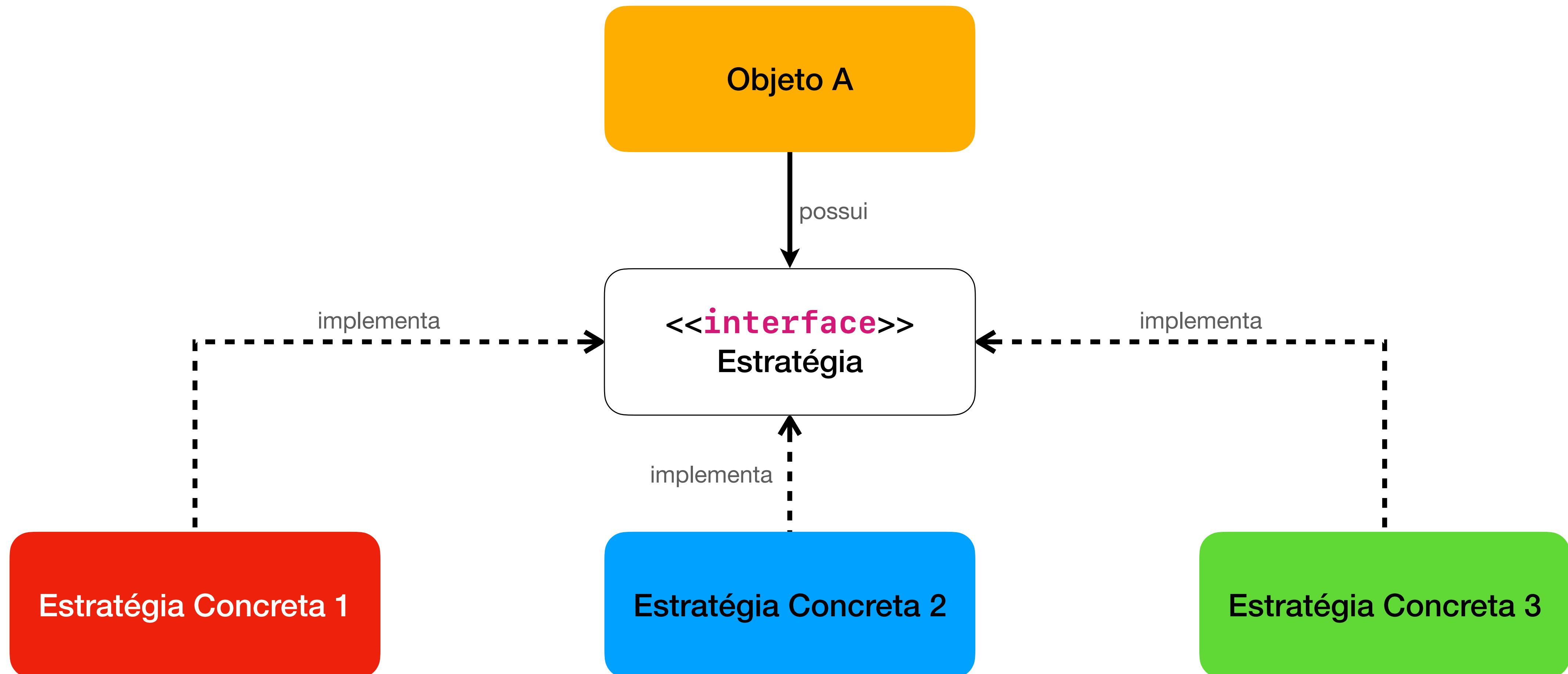
Um padrão comportamental



Definir uma família de
comportamentos e torná-los
intercambiáveis entre si

Strategy Pattern

Também conhecido como *Policy*



Strategy Pattern

Quando usar? Quando não usar?

- Quando, em um dado cenário, você se deparar com dois ou mais comportamentos que são intercambiáveis entre si;
- Apesar de ter o uso de **interfaces** em comum com o *Business Delegate*, o *Strategy* fomenta que as diferentes estratégias devem ser facilmente trocadas em tempo de execução;
- O abuso deste Design Pattern ocorre quando ele é usado com um comportamento que não será alterado. Neste caso, não use;
- Vamos ao exemplo...

K-BÔ!



<https://github.com/intellitour/TesteDescomplica>