Especialização em Desenvolvimento Java

Padrões de Projeto

Prof. Vinícius de Paula

https://github.com/viniciusdepaula/aulas-uml-padroes

Unitri
CENTRO UNIVERSITÁRIO DO TRIÂNGULO

Sobre os Padrões de Projeto

Sobre os Padrões de Projeto

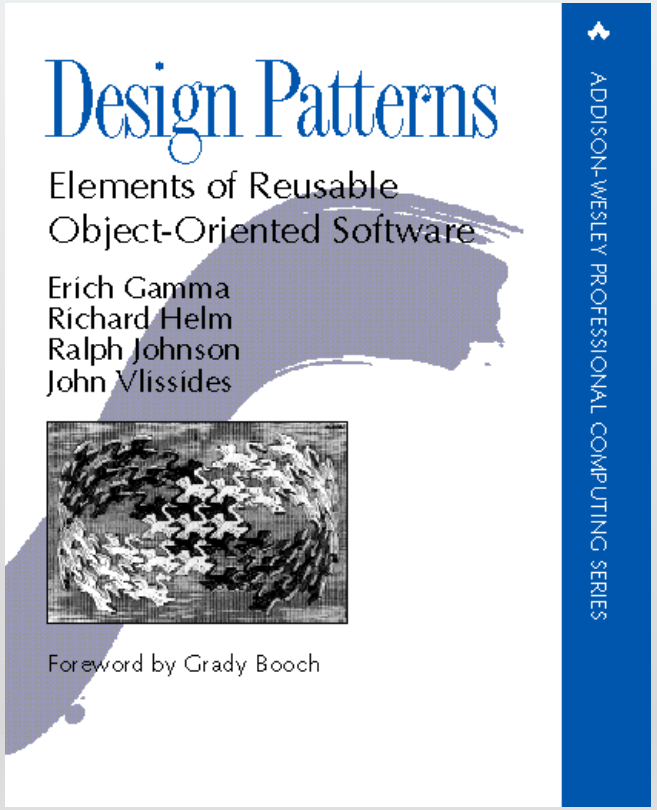
- Apesar de específicos, os sistemas corporativos possuem diversas características semelhantes.
- Consequentemente, muitos problemas se repetem em contextos distintos.
- Para não perder tempo e dinheiro elaborando soluções diferentes para o mesmo problema, poderíamos escolher uma solução como padrão e adotá-la toda vez que o problema correspondente ocorrer.
- Além de evitar o retrabalho, facilitaríamos a comunicação dos desenvolvedores e o entendimento técnico do sistema.

Sobre os Padrões de Projeto

- Dado o exposto, surge o conceito de padrão de projeto ou design pattern.
- Um padrão de projeto é uma solução consolidada para um problema recorrente no desenvolvimento e manutenção de software orientado a objetos.

Sobre os Padrões de Projeto

- A referência mais importante relacionada a padrões de projeto é o livro *Design Patterns: Elements of Reusable Object-Oriented Software* (editora Addison-Wesley, 1995) dos autores Erich Gamma, Ri- chard Helm, Ralph Johnson e John Vlissides.
- Esses quatro autores são conhecidos como “Gang of Four”(GoF).



Padrões GoF

- Os padrões definidos no livro *Design Patterns: Elements of Reusable Object-Oriented Software* são denominados padrões GoF.
- Eles são classificados em três categorias: padrões de **criação**, **estruturais** e **comportamentais**.

Padrões de Criação

- Em um sistema orientado a objetos, a criação de certos objetos pode ser uma tarefa extremamente complexa. Podemos destacar dois problemas relacionados a criação de objetos:
  - Definir qual classe concreta deve ser utilizada para criar o objeto;
  - Definir como os objetos devem ser criados e como eles se relacionam com outros objetos do sistema.
- Seguindo o princípio do encapsulamento, essa complexidade deve ser isolada.
- Veremos os padrões de projetos que podem ser adotados para encapsular a criação de objetos em diversas situações distintas.

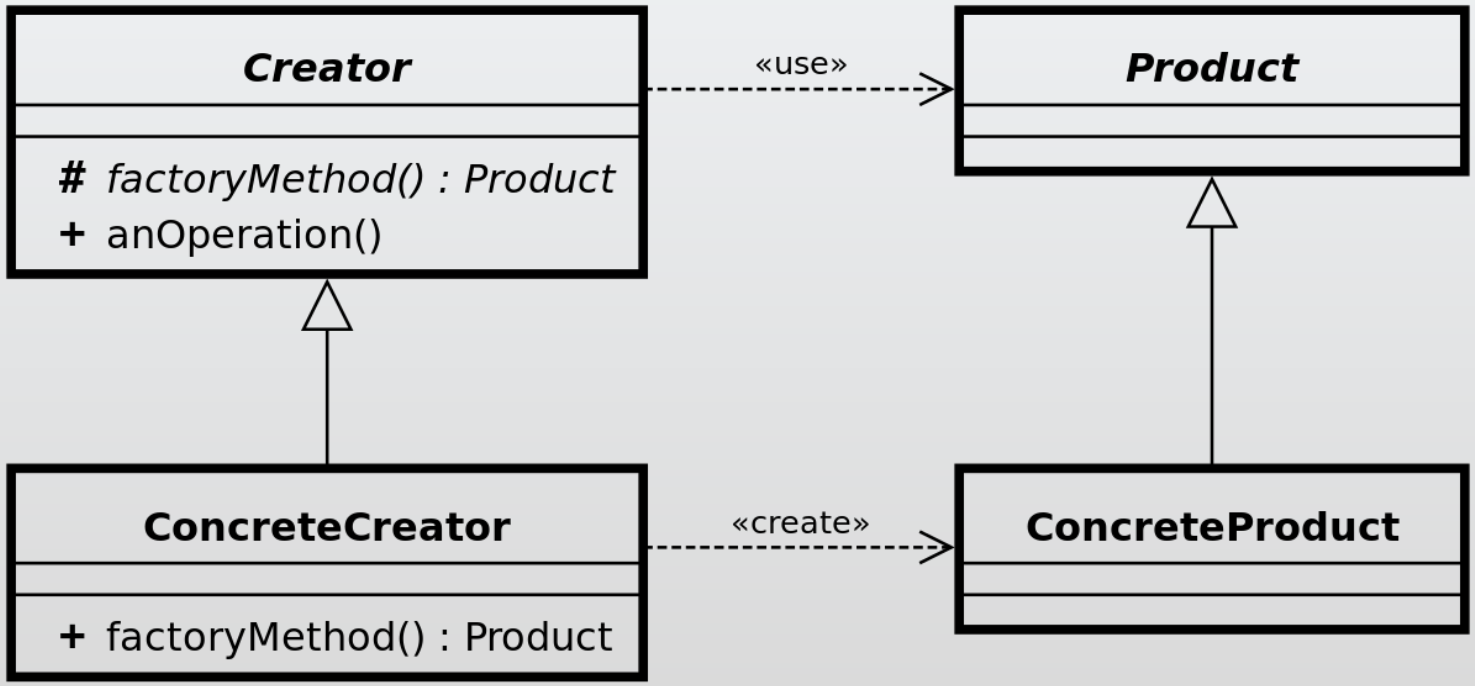
Factory Method

Factory Method

Tem como objetivo:

- Encapsular a escolha da classe concreta a ser utilizada na criação de objetos de um determinado tipo.
- Definir uma interface para criar objetos de forma a deixar subclasses decidirem qual classe instanciar.

Factory Method



Factory Method

Exemplo prático:

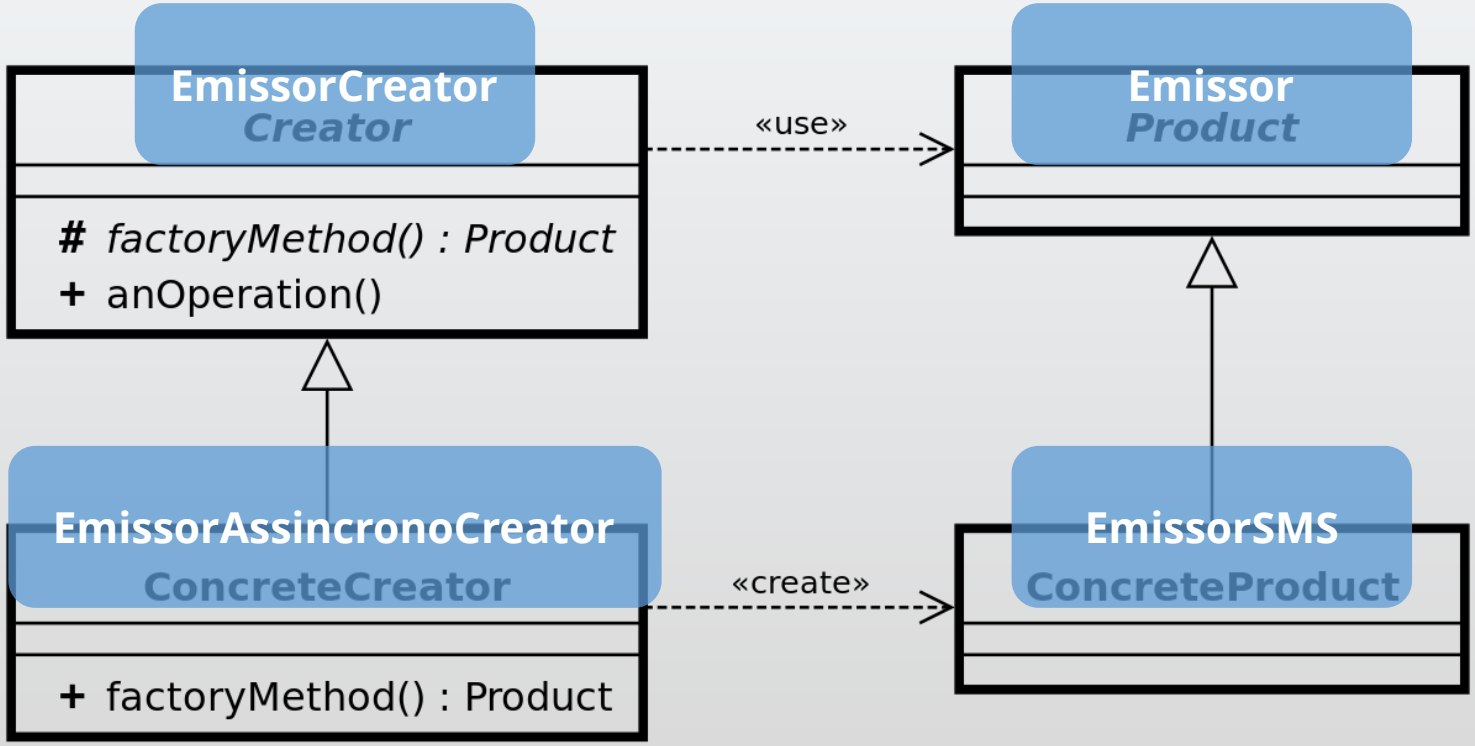
- Considere um sistema bancário que precisa enviar mensagens aos seus clientes. Por exemplo, após a realização de uma compra com cartão de crédito, uma mensagem contendo informações sobre a compra pode ser enviada ao cliente.
- Se esse cliente for uma pessoa física, poderá optar pelo recebimento da mensagem através de email ou SMS. Por outro lado, se for uma pessoa jurídica, poderá também receber a mensagem através de JMS(Java Message Service).

Factory Method

Exercício:

- Talvez, o sistema tenha que trabalhar com dois tipos diferentes de envio de mensagens: síncrono e assíncrono.
- Especialize o criador de emissores, definindo subclasses, de modo que seja possível enviar mensagens por Email, SMS e JMS de forma síncrona ou assíncrona.

Factory Method



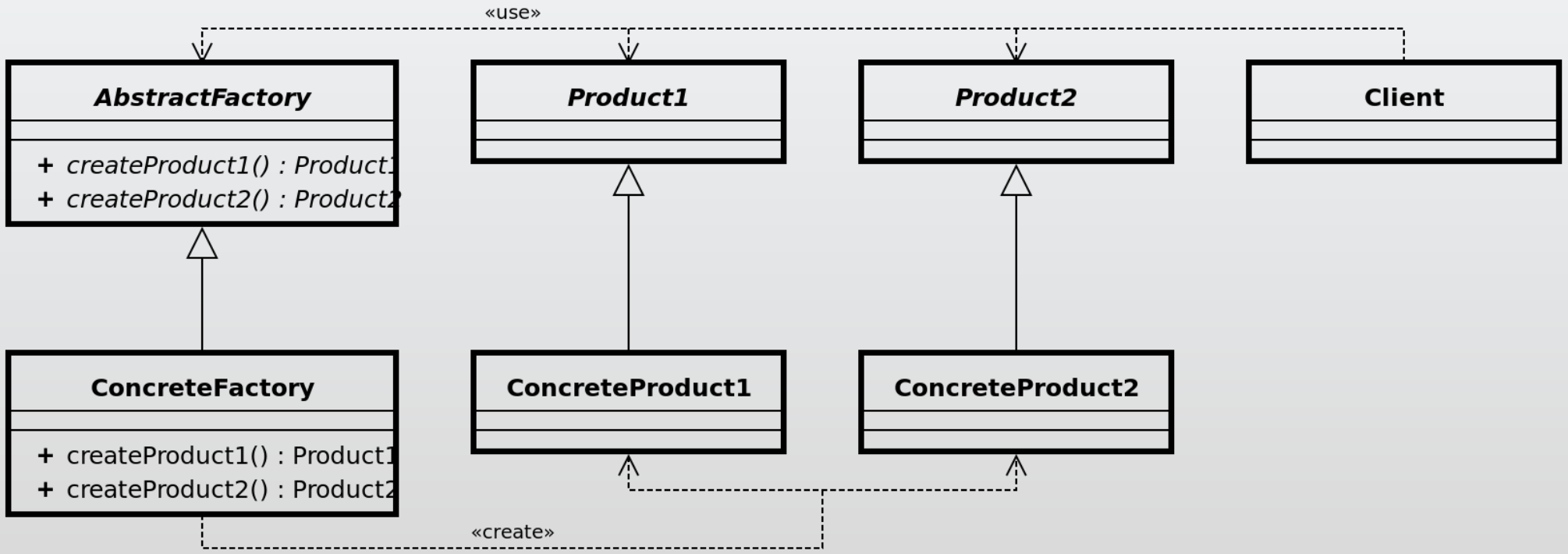
Abstract Method

Abstract Method

Tem como objetivo:

- Encapsular a escolha das classes concretas a serem utilizadas na criação dos objetos de diversas família.

Abstract Method



Abstract Method

Exemplo prático:

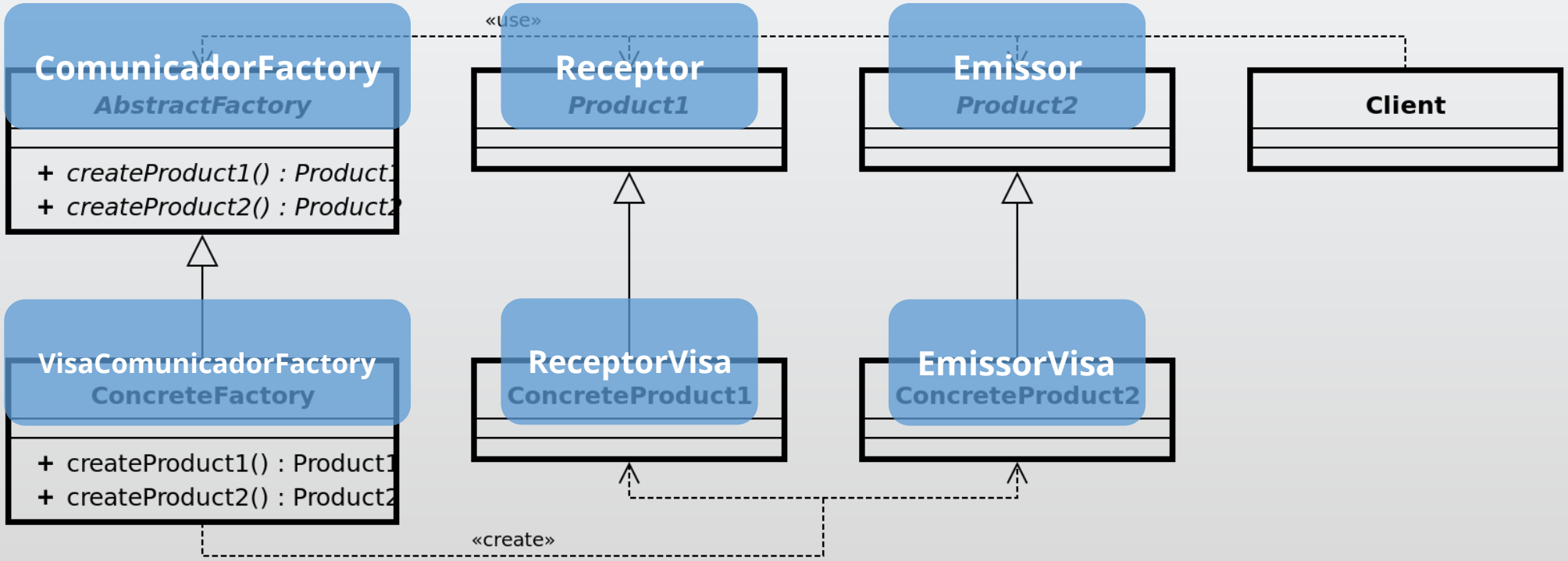
- Estabelecimentos comerciais normalmente oferecem aos clientes diversas opções de pagamento. Por exemplo, clientes podem efetuar pagamentos com dinheiro, cheque, cartões de crédito ou débito, entre outros.
- Se esse cliente for uma pessoa física, poderá optar pelo recebimento da mensagem através de email ou SMS. Por outro lado, se for uma pessoa jurídica, poderá também receber a mensagem através de JMS.

Abstract Method

Exemplo prático:

- Pagamentos com cartões são realizados por meio de uma máquina de cartão, oferecida e instalada por empresas como Cielo e Redecard. Geralmente, essa máquina é capaz de lidar com cartões de diferentes bandeiras (como Visa e Mastercard).
- Nosso objetivo é programar essas máquinas, isto é, desenvolver uma aplicação capaz de se comunicar com as diferentes bandeiras e registrar pagamentos.
- No momento do pagamento, a máquina de cartão deve enviar as informações relativas a transação (como valor e senha) para a bandeira correspondente ao cartão utilizado. Além disso, a máquina deve aguardar uma resposta de confirmação ou recusa do pagamento.

Abstract Method




Builder

Builder

Tem como objetivo:

- Separar o processo de construção de um objeto de sua representação e permitir a sua criação passo-a-passo.
- Diferentes tipos de objetos podem ser criados com implementações distintas de cada passo.

Builder

Exemplo prático:

Estamos desenvolvendo um sistema para gerar boletos bancários. No Brasil, a FEBRABAN (Federação Brasileira de Bancos) define regras gerais para os boletos. Contudo, cada instituição bancária define também as suas próprias regras específicas para o formato dos dados dos boletos.

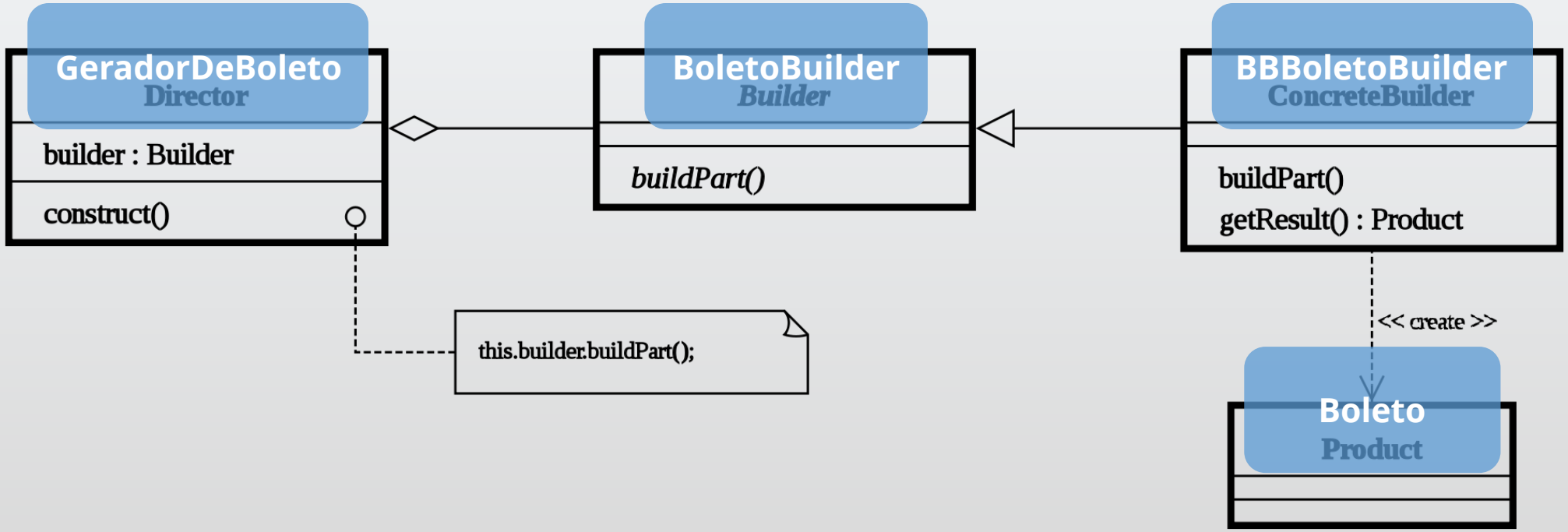
Builder

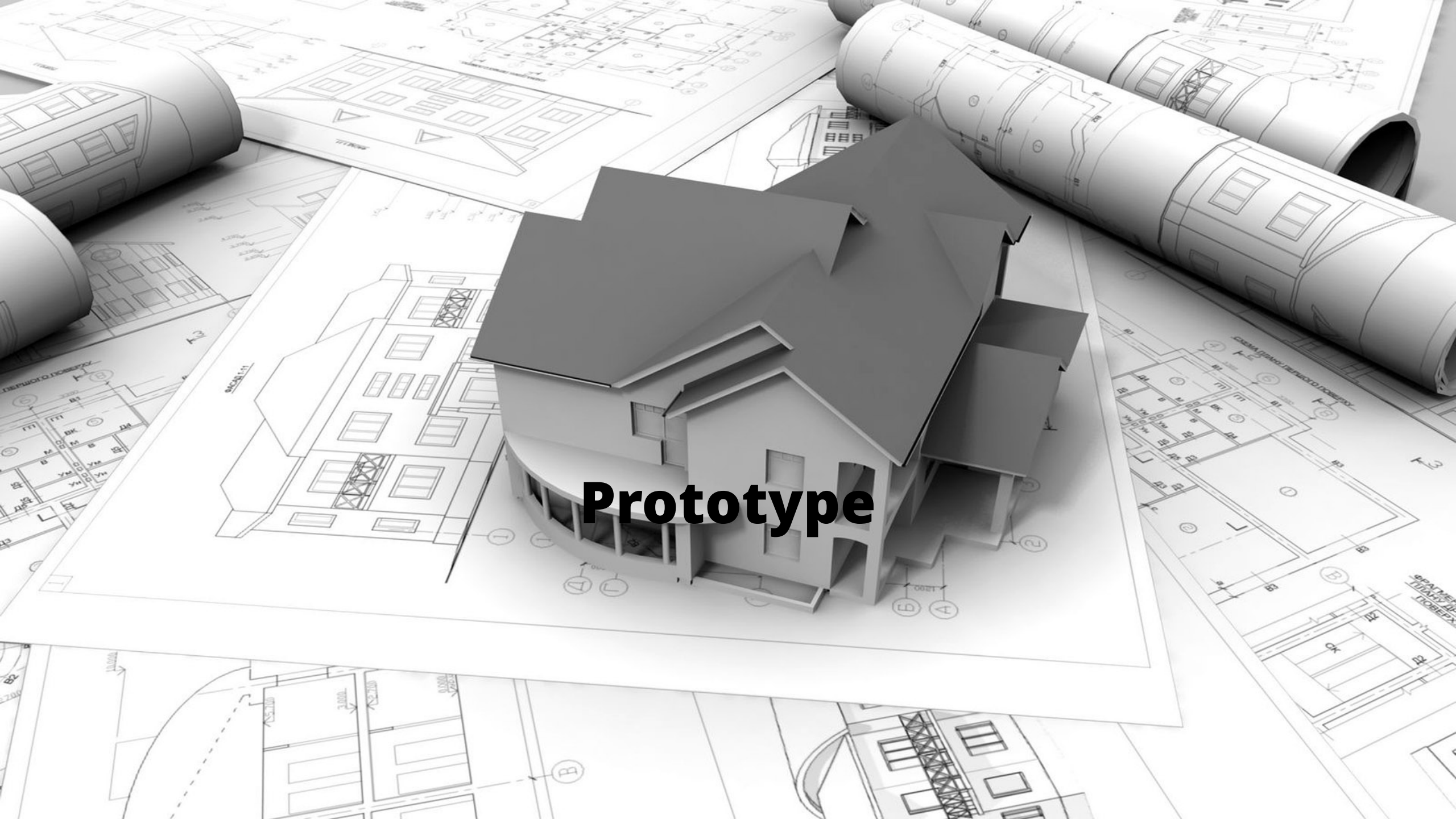
Exemplo prático:

Segundo a FEBRABAN, os elementos principais relacionados a um boleto são:

- **Sacado:** Pessoa ou empresa responsável pelo pagamento do boleto.
- **Cedente:** Pessoa ou empresa que receberá o pagamento do boleto.
- **Banco:** Instituição que receberá o pagamento do sacado e creditará o valor na conta do cedente.
- **Código de Barras:** Representação gráfica das informações do boleto.
- **Linha Digitável:** Representação numérica das informações do boleto.
- **Nosso Número:** Código de identificação do boleto utilizado pela instituição bancária e pelo cedente para controle dos pagamentos.

Builder


GeradorDeBoleto
Director
builder : Builder
construct()
BoletoBuilder
Builder
buildPart()
BBBoletoBuilder
ConcreteBuilder
buildPart()
getResult() : Product
this.builder.buildPart();
<< create >>
Boleto
Product

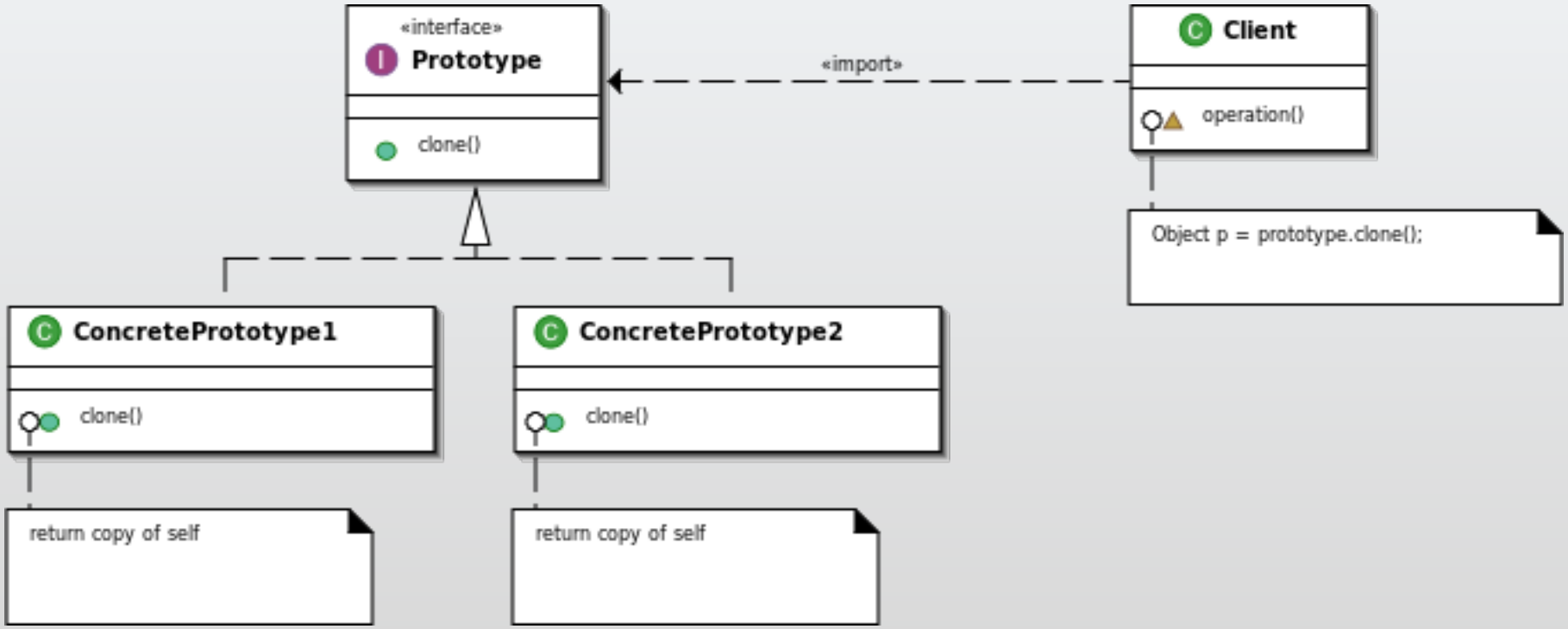

Prototype

Prototype

Tem como objetivo:

- Possibilitar a criação de novos objetos a partir da cópia de objetos existentes.

Prototype


«interface»
Prototype
clone()
«import»
Client
operation()
Object p = prototype.clone();
ConcretePrototype1
clone()
return copy of self
ConcretePrototype2
clone()
return copy of self

Prototype

Exemplo prático:

Estamos desenvolvendo um sistema de anúncios semelhante ao do Google Adwords. Nesse sistema, os usuários poderão criar campanhas e configurá-las de acordo com as suas necessidades. Uma campanha é composta por diversas informações, entre elas:

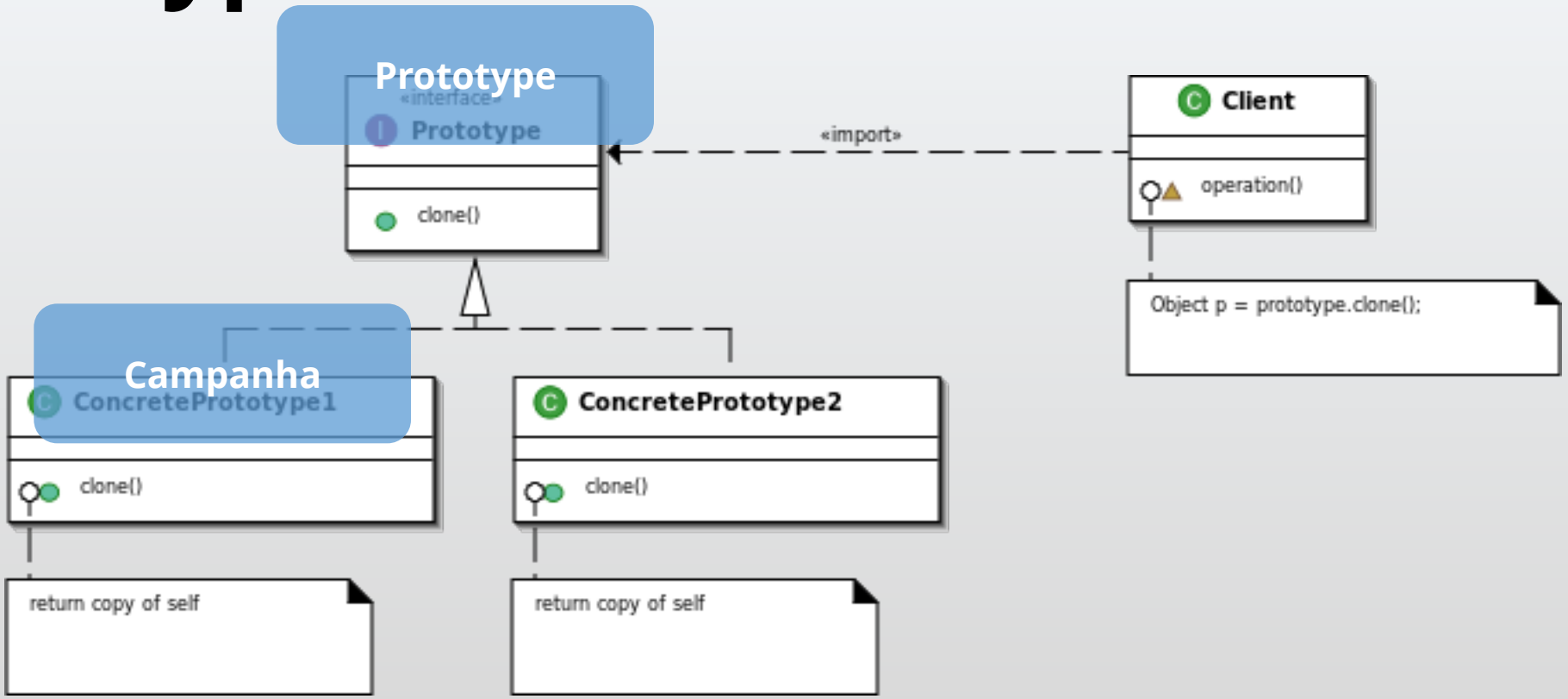
- Uma lista de anúncios;
- O valor diário máximo que deve ser gasto pela campanha;
- O valor máximo por exibição de anúncio;
- As datas de início e término.

Prototype

Exemplo prático:

Nesse tipo de sistema, os usuários geralmente criam campanhas com configurações extremamente parecidas. Dessa forma, seria interessante que o sistema tivesse a capacidade de criar uma campanha a partir de uma outra campanha já criada anteriormente, para que as configurações pudessem ser aproveitadas.

Prototype


Prototype
«interface»
Prototype
clone()
«import»
Client
operation()
Object p = prototype.clone();
Campanha
ConcretePrototype1
clone()
return copy of self
ConcretePrototype2
clone()
return copy of self

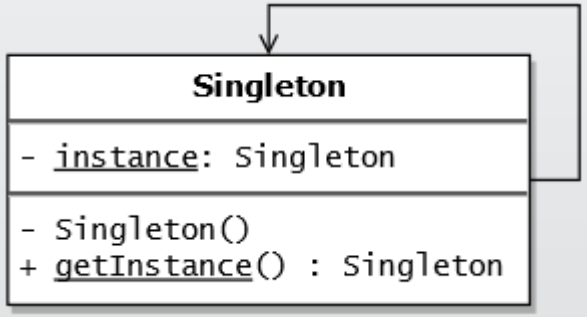
Singleton

Singleton

Tem como objetivo:

- Permitir a criação de uma única instância de uma classe e fornecer um modo para recuperá-la.

Singleton


Singleton
- instance: Singleton
- Singleton()
+ getInstance() : Singleton

Singleton

Exemplo prático:

Suponha que estamos desenvolvendo um sistema que possui configurações globais obtidas a partir de um arquivo de propriedades. Essas configurações podem ser armazenadas em um objeto.

Não desejamos que existam mais do que um objeto dessa classe ao mesmo tempo no sistema.

Bibliografia

- Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson e Vlissides, Addison-Wesley, 1995.(Padrões de Projeto - Soluções Reutilizáveis de Software Orientado a Objeto - Gamma, Helm, Johnson e Vlissides, Bookman, 2000.
- Patterns of Enterprise Application Architecture, Fowler, Addison Wesley, 2003.