**JAVASCRIPT**

**Modulo II - ECMAScipt**

**Aluno: Pierre Leon Castanha de Lima**

**Atividade proposta:**

**Pesquise sobre padrões de projeto e escolha um para apresentar e descrever o seu funcionamento. Além disso, explique quais as vantagens e desvantagens comparados a outros e mostre suas referências.**

**PADRÃO DE PROJETO ”PROTOTYPE”**

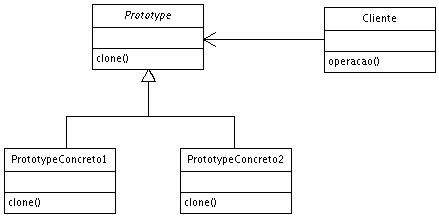
Prototype, na [ciência da computação](https://pt.wikipedia.org/wiki/Ci%C3%AAncia_da_computa%C3%A7%C3%A3o), é um [padrão de projeto de software](https://pt.wikipedia.org/wiki/Padr%C3%B5es_de_projeto_de_software) (*design pattern, em inglês*). Criacional que permite a criação de novos objetos a partir de um modelo original ou [protótipo](https://pt.wikipedia.org/wiki/Prot%C3%B3tipo) que é clonado.

Este padrão pode ser utilizado para:

1. *Evitar que as subclasses que criam objetos funcionem como o padrão abstract factory;*
2. *Evitar criar um novo objeto utilizando a palavra new, o que diminui o custo de memória.*

Basicamente, ao em vez de o cliente implementar um código que utiliza o ***operador new***, este utiliza o ***método clone()*** presente no protótipo e o método de uma fábrica***(Factory Method ou Abstratct Factory)*** que fica encarregada de clonar o novo objeto.

**Estrutura**

[](https://pt.wikipedia.org/wiki/Ficheiro:Prototype_UML_(padr%C3%A3o).png)

**Diagrama**[**UML**](https://pt.wikipedia.org/wiki/UML)**da estrutura do padrão Prototype**

O padrão Prototype, da forma como foi descrito no livro [***Design Patterns: Elements of Reusable Object-Oriented Software***](https://pt.wikipedia.org/w/index.php?title=Design_Patterns:_Elements_of_Reusable_Object-Oriented_Software&action=edit&redlink=1)***,*** contém os seguintes elementos:

***prototype***— uma classe que declara uma interface para objetos capazes de clonar a si mesmo.

***prototype***[***concreto***](https://pt.wikipedia.org/wiki/Classe_(programa%C3%A7%C3%A3o)#Classes_abstratas_e_concretas) — implementação de um prototype;

***cliente***— cria um novo objeto através de um prototype que é capaz de clonar a si mesmo.

Efetivamente, cada objeto é, ele próprio, um *[factory](https://pt.wikipedia.org/wiki/Abstract_Factory" \o "Abstract Factory)* especializado em construir objetos iguais a si mesmo. O padrão Prototype é utilizado frequentemente em linguagens estaticamente tipadas como [C++](https://pt.wikipedia.org/wiki/C%2B%2B) e [Java](https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)), e menos frequentemente utilizadas em linguagens dinamicamente tipadas como [Smalltalk](https://pt.wikipedia.org/wiki/Smalltalk" \o "Smalltalk).

O padrão Prototype exige a implementação de uma operação de clonagem em cada uma das classes concretas do protótipo. Esta tarefa pode ser inconveniente, no caso do reaproveitamento de classes preexistentes que não possuem tal operação, ou mesmo complexa, se for considerada a possibilidade de existirem referências circulares nos atributos de um objeto (um objeto possui um atributo que referencia um objeto que, por sua vez, referência o objeto original).

**Utilização**

O padrão Prototype é aplicado quando existe a necessidade de clonar, literalmente, um objeto. Ou seja, quando a aplicação precisa criar cópias exatas de algum objeto em tempo de execução este padrão é altamente recomendado. Este padrão pode ser utilizado em sistemas que precisam ser independentes da forma como os seus componentes são criados, compostos e representados. O padrão Prototype pode ser útil em sistemas com as seguintes características:

**sistemas que utilizam classes definidas em**[**tempo de execução**](https://pt.wikipedia.org/wiki/Tempo_de_execu%C3%A7%C3%A3o)**;**

sistemas que utilizam o padrão [Abstract Factory](https://pt.wikipedia.org/wiki/Abstract_Factory) para criação de objetos. Neste caso, a hierarquia de classes pode se tornar muito complexa e o padrão Prototype pode ser uma alternativa mais simples, por realizar a mesma tarefa com um número reduzido de classes;

sistemas que possuem componentes cujo estado inicial possui poucas variações e onde é conveniente disponibilizar um conjunto preestabelecido de protótipos que dão origem aos objetos que compõem o sistema.

Algumas vezes, padrões criacionais podem ser "competidores": Existem casos em que o Prototype ou Abstract Factory podem ser utilizados apropriadamente. Em outros, eles são complementares: Um Abstract Factory pode conter um set de Prototypes no qual clona e retorna o produto de objetos.

Muitos Designers começam utilizando o Factory Method, pois seu uso é menos complicado, mas então evoluem para Abstract Factory, Prototype ou Builder, que são mais flexíveis porém mais complexos. E é neste ponto em que o Factory Method e Prototype divergem: O primeiro requer subclasses, mas não precisa ser inicializado. O segundo, não necessita de subclasses porém precisa ser inicializado.

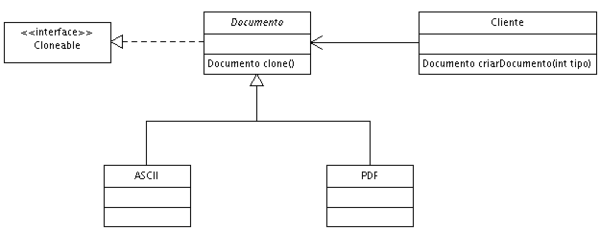
Quando utiliza o [framework](https://pt.wikipedia.org/wiki/Framework) [Spring](https://pt.wikipedia.org/wiki/Spring_Framework), por exemplo, um desenvolvedor pode configurar um [JavaBean](https://pt.wikipedia.org/w/index.php?title=Javabeans&action=edit&redlink=1" \o "Javabeans (página não existe)) como "prototype". Esta configuração faz com que cada uma das referências a um JavaBean aponte para uma instância diferente. O comportamento padrão, ou [singleton](https://pt.wikipedia.org/wiki/Singleton" \o "Singleton), define que todas as referências a um JavaBean apontem para a mesma instância de uma classe.[[1]](https://pt.wikipedia.org/wiki/Prototype#cite_note-1)

Uma das principais vantagens de sua utilização é quando a inicialização de um objeto pode se tornar custosa, e você quer fazer algumas pequenas variações ao inicializar. Nesse contexto, o Prototype pode então evitar a criação "do zero" de novos objetos.

**Exemplo**

Neste exemplo é mostrado uma hierarquia de classes representando documentos de formato [ASCII](https://pt.wikipedia.org/wiki/ASCII) e [PDF](https://pt.wikipedia.org/wiki/PDF) que são criados através da classe Cliente. A partir de duas instâncias prototípicas, ascii e pdf, o método criarDocumento cria clones de documentos de acordo com o tipo desejado. A tarefa de realizar a criação da instância é implementada na classe Documento e herdada por suas classes filhas, ASCII e PDF.

**Diagrama**

[](https://pt.wikipedia.org/wiki/Ficheiro:Prototype_sample.png)

Exemplo de Diagrama em [UML](https://pt.wikipedia.org/wiki/UML) para o Padrão Prototype.

**Código**

Este código, escrito na [linguagem Java](https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)), mostra a implementação do diagrama mostrado acima.

***public interface ICloneable {***

***public Object clone();***

***}***

***abstract class Documento implements ICloneable {***

***protected Documento clone() {***

***Object clone = null;***

***try {***

***clone = super.clone();***

***} catch (CloneNotSupportedException ex) {***

***ex.printStackTrace();***

***}***

***return (Documento) clone;***

***}***

***}***

***class ASCII extends Documento { }***

***class PDF extends Documento { }***

***class Cliente {***

***static final int DOCUMENTO\_TIPO\_ASCII = 0;***

***static final int DOCUMENTO\_TIPO\_PDF = 1;***

***private Documento ascii = new ASCII();***

***private Documento pdf = new PDF();***

***public Documento criarDocumento(int tipo) {***

***if (tipo == Cliente.DOCUMENTO\_TIPO\_ASCII) {***

***return ascii.clone();***

***} else {***

***return pdf.clone();***

***}***

***}***

***}***

Observação: Na linguagem Java, a interface***Cloneable*** não possui métodos e é utilizada apenas para indicar que o método ***Object.clone()*** pode realizar uma cópia, atributo por atributo, das instâncias de uma classe.[[2]](https://pt.wikipedia.org/wiki/Prototype#cite_note-2)

Neste exemplo, a implementação da cópia da instância é delegada ao método clone da super classe ***Object*** (A classe raiz da hierarquia de classes da linguagem Java). Se fosse necessário, as classes Documento, ASCII ou PDF implementariam esse método para refletir algum comportamento específico.

**Referências**

[↑](https://pt.wikipedia.org/wiki/Prototype#cite_ref-1) *Rod Johnson, Juergen Hoeller, Alef Arendsen, Colin Sampaleanu, Rob Harrop, Thomas Risberg, Darren Davison, Dmitriy Kopylenko, Mark Pollack, Thierry Templier, Erwin Vervaet, Portia Tung, Ben Hale, Adrian Colyer, John Lewis, Costin Leau, Rick Evans.*[«The Spring Framework - Reference Documentation»](https://web.archive.org/web/20070616073820/http:/www.springframework.org/docs/reference/beans.html#beans-factory-scopes-prototype)*(em inglês)*. Consultado em 13 de junho de 2007*. Arquivado do*[original](http://www.springframework.org/docs/reference/beans.html#beans-factory-scopes-prototype)*em 16 de junho de 2007*

[↑](https://pt.wikipedia.org/wiki/Prototype#cite_ref-2) [Sun Microsystems](https://pt.wikipedia.org/wiki/Sun_Microsystems)*.*[«JavaTM 2 Platform Std. Ed.»](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Cloneable.html)*(em inglês)*. Consultado em 13 de junho de 2007

**Bibliografia**

[Erich Gamma](https://pt.wikipedia.org/wiki/Erich_Gamma)*,*[Richard Helm](https://pt.wikipedia.org/wiki/Richard_Helm)*,*[Ralph Johnson](https://pt.wikipedia.org/wiki/Ralph_Johnson)*,*[John Vlissides](https://pt.wikipedia.org/wiki/John_Vlissides)*(1995). Design Patterns. Elements of Reusable Object-Oriented Software 1 ed.*[Estados Unidos da América](https://pt.wikipedia.org/wiki/Estados_Unidos_da_Am%C3%A9rica)*: Addison-Wesley.*[ISBN](https://pt.wikipedia.org/wiki/International_Standard_Book_Number)[0-201-63361-2](https://pt.wikipedia.org/wiki/Especial:Fontes_de_livros/0-201-63361-2)