Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

Prototype

Padrões Criacionais

O padrão *Prototype* especifica tipos de objetos a serem criados usando como base uma instância de outro objeto que serve como protótipo. Este padrão permite a clonagem de objetos existentes sem provocar dependência de suas classes.

Motivação (Por que utilizar?)

Considere que está trabalhando com programação orientada a objetos e precisa criar um objeto idêntico a outro já existente, para isso:

- Primeiro é necessário criar um objeto da mesma classe que o objeto a ser copiado.
- Depois copiar todos os atributos (variáveis de instância) do objeto existente para o novo objeto.

Nestes três passos podemos identificar três problemas:

- Alguns atributos do objeto a ser copiado podem ser inacessíveis de fora dele, por serem privados ou protegidos.
- Para copiar cada um dos atributos, o código responsável por fazer tal cópia precisa conhecer detalhes de implementação do objeto a ser copiado, isso causa dependência entre eles.
- Dependendo do contexto do código, pode-se conhecer apenas a interface do objeto a ser copiado, pois mais de um objeto de mesmo supertipo podem ser aceitos pelo contexto.

Para contornar tais problemas o padrão *Prototype* propõe que a responsabilidade de cópia do objeto passe a ser dele mesmo, ou seja, o objeto se autocopia e retorna uma cópia pronta de si mesmo.

- Mesmo atributos privados ou protegidos são acessíveis dentro do próprio objeto.
- Como o objeto retorna uma cópia de si próprio, o código que solicita tal cópia não precisa conhecer os detalhes deste processo, precisa saber apenas que ele é capaz de se auto copiar.

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

- Apenas objetos concretos podem se auto copiar, então mesmo que seja representado por uma interface ele será capaz de retornar uma cópia de si próprio.

O termo mais adequado a esse processo de se auto copiar é **clonagem**, um objeto que suporta clonagem é chamado de **protótipo**. O código solicitante de uma clonagem precisa saber que o objeto, para o qual a solicitação foi destinada, é capaz de se clonar. Por este motivo ele precisa implementar uma interface que garanta tal funcionalidade. De forma geral essa interface possui apenas um método `clonar()` que é muito parecido em todas as classes onde é implementado. Ele basicamente cria um objeto de sua própria classe e carrega todos os atributos (variáveis de instância) do objeto clonado para o objeto clone.

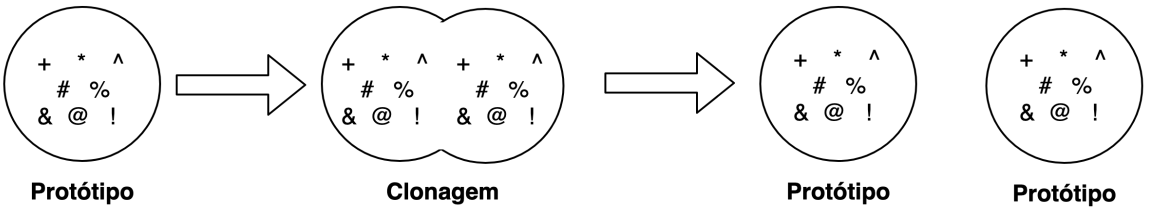
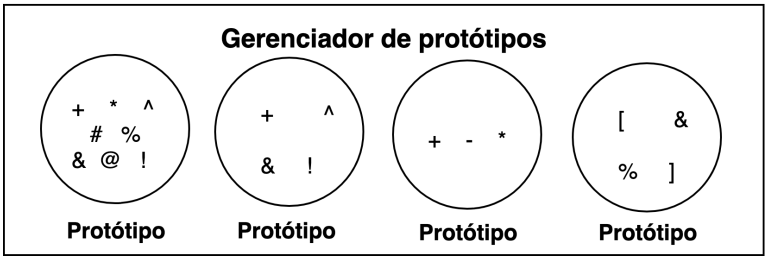


Ilustração do processo de clonagem de um objeto

A clonagem pode ser muito útil quando existem diversos objetos com configurações complexas e diferentes. Ao invés de gerenciar tais configurações no construtor da classe, pode-se criar um conjunto de objetos clonáveis, configurados de diversas formas diferentes. Quando existir a necessidade de um objeto parecido com o um dos pré-configurados, basta clonar um protótipo ao invés de construir um novo objeto do zero.

Quando o número de protótipos em um sistema não for fixo (ou seja, eles podem ser criados e destruídos dinamicamente), é preciso manter um sistema de registro dos protótipos disponíveis. Os clientes não gerenciam os protótipos, mas os recuperam e armazenam a partir do sistema de registro. Um cliente solicitará um protótipo ao sistema de registro antes de cloná-lo. Tal sistema de registro é chamado de **gerenciador de protótipos**.



Gerenciador de protótipos

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

Vamos entender esse padrão por meio de um exemplo. Considere que estamos dando manutenção em um aplicativo que gerencia o acervo de bibliotecas virtuais. Tais acervos são compostos por livros, revistas e trabalhos acadêmicos (Dissertações e Teses) todos digitais. Nos foi dito que muitas instâncias de **Livros**, **Revistas** e **Trabalhos** são requisitadas com muita frequência. Sabemos também que criar essas instâncias do zero é um processo muito trabalhoso (assumiremos isso para viabilizar o exemplo), nosso desafio é facilitar a criação destes objetos.

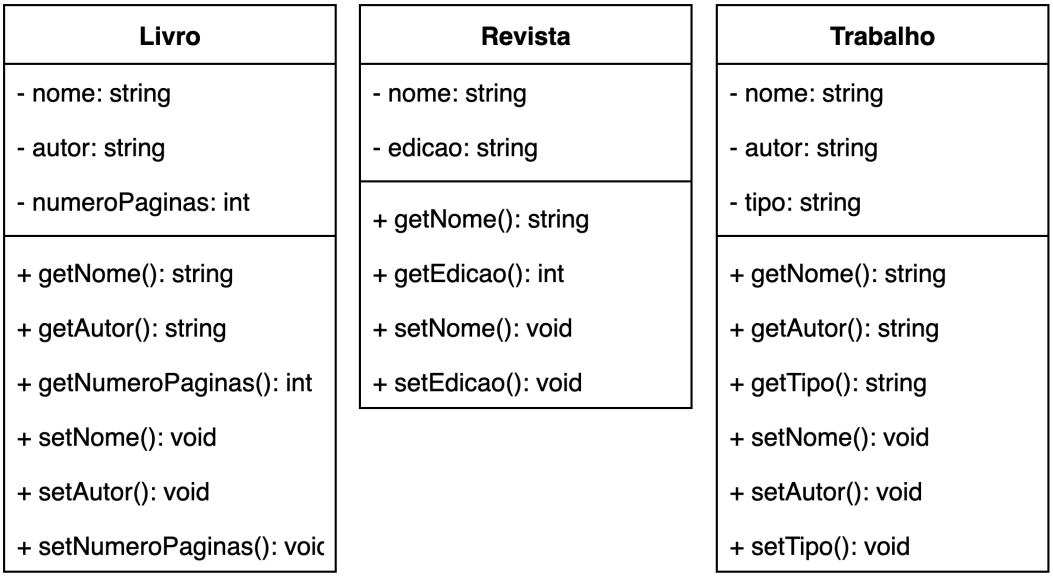


Diagrama das classes que compõem o acervo digital das bibliotecas

Podemos utilizar o padrão *Prototype* já que instâncias de **Livros**, **Revistas** e **Trabalhos** são requisitadas com muita frequência e sua criação é trabalhosa. Deste modo, criaremos as instâncias dos protótipos e sempre que precisarmos de uma nova instância faremos a clonagem de um deles.

Para gerenciar os protótipos utilizaremos a classe **GerenciadorDePrototipos** seu papel é armazenar os objetos que aceitam clonagem, ou seja, os protótipos. Cada protótipo terá um identificador que permite que o cliente solicite a instância de um protótipo específico. O método **getInstance()** será o responsável por receber tal identificador e retornar uma cópia do protótipo apropriado.

Em nosso exemplo temos um número fixo de protótipos. Caso a quantidade de protótipos fosse dinâmica, a classe **GerenciadorDePrototipos** precisaria de mais alguns métodos que permitissem adicionar e remover protótipos dinamicamente a sua lista de protótipos disponíveis.

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

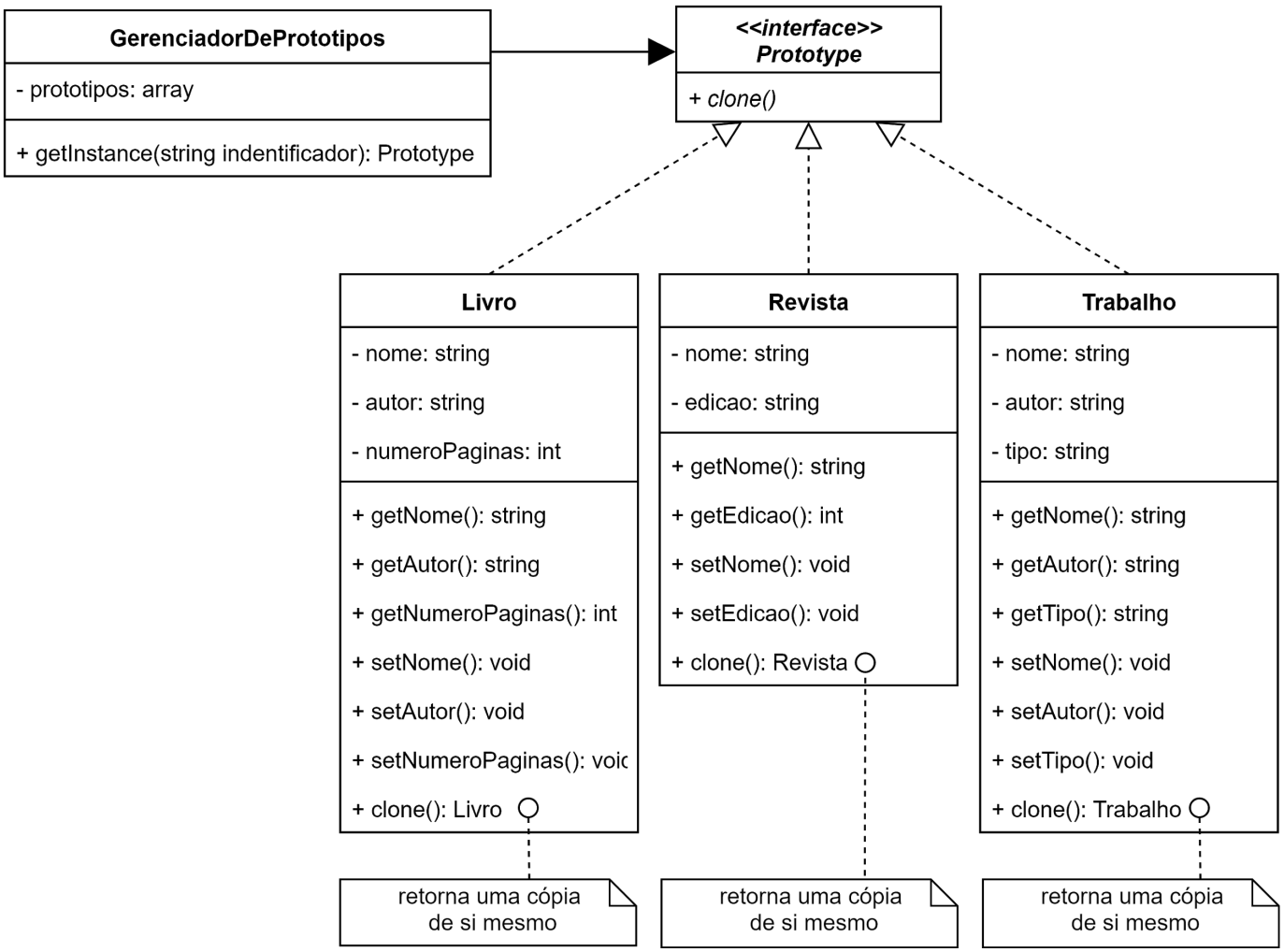


Diagrama de classes de protótipos que compõem o acervo digital das bibliotecas

Vamos começar implementando a interface `Prototype`.

```
interface Prototype
{
    public function clone(): Prototype;
}
```

A parte mais difícil do padrão *Prototype* é implementar a operação `clone()` corretamente. É particularmente complicado quando as estruturas de objetos contêm referências circulares. A maioria das linguagens fornecem suporte para clonagem de objetos.

Existem duas maneiras de clonar objetos:

- **Clonagem rasa** (*Shallow cloning*): Clona o objeto, porém qualquer propriedade que seja referência a outra variável ou objeto, permanecerá como referência. Isso implica que assim que uma dessas referências mudar ambos os objetos, clonado e clone, serão afetados.

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

- **Clonagem profunda** (*Deep cloning*): Além de clonar o próprio objeto clona também os objetos referenciados por ele, ou seja uma nova árvore de objetos é criada de modo que, objeto clonado e clone sejam completamente independentes.

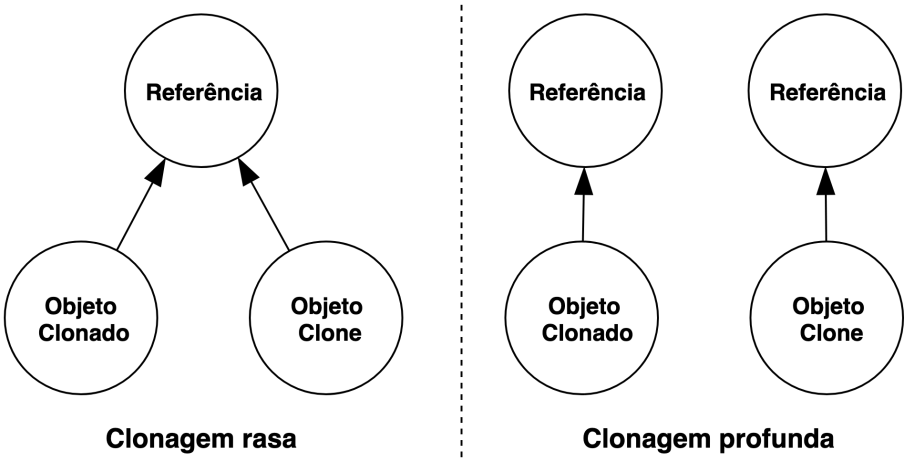
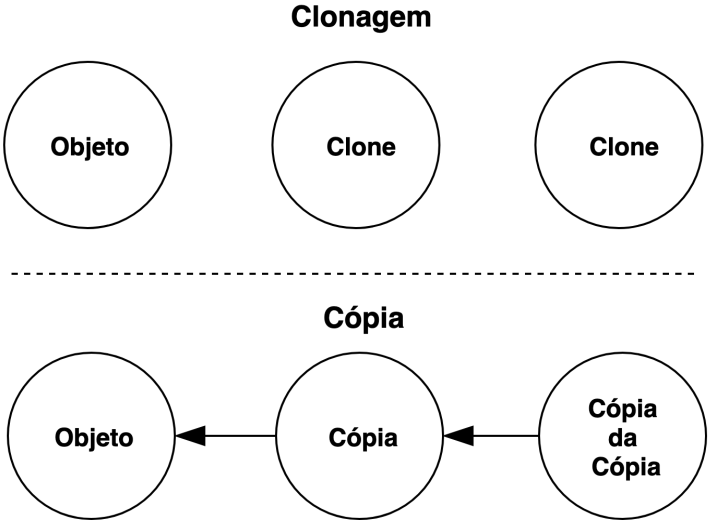


Ilustração de clonagem rasa versus clonagem profunda

Nota: O PHP possui um método nativo chamado **clone()**. Ele faz a clonagem rasa de um objeto. Existe também um método mágico chamado **__clone()**, ele não pode ser chamado de forma direta, porém, ele sempre é executado depois que a clonagem feita pelo método **clone()** termina, deste modo, qualquer propriedade do objeto resultante da clonagem pode ser alterada. Isso possibilita que dentro do método **__clone()** seja implementado um processo de clonagem profunda do objeto.

Vale ressaltar que **clonagem é diferente de cópia**. A clonagem resulta em dois objetos iguais porém independentes, o grau de independência pode variar conforme o tipo de clonagem adotada. Já a cópia de um objeto apenas cria uma referência a ele de modo que a edição de um objeto é propagada para todas as suas cópias.



Diferença entre clonagem é cópia

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

Vamos então à implementação das classes **Livro**, **Revista** e **Trabalho**.

```
class Livro implements Prototype
{
    private string $nome;
    private string $autor;
    private int $numeroPaginas;

    public function getNome(): string
    {
        return $this->nome;
    }

    public function setNome(string $nome): void
    {
        $this->nome = $nome;
    }

    public function getAutor(): string
    {
        return $this->autor;
    }

    public function setAutor(string $autor): void
    {
        $this->autor = $autor;
    }

    public function getNumeroPaginas(): int
    {
        return $this->numeroPaginas;
    }

    public function setNumeroPaginas(int $numeroPaginas): void
    {
        $this->numeroPaginas = $numeroPaginas;
    }

    //Método responsável por clonar o objeto Livro
    public function clone(): Prototype
    {
        //Criação de uma nova instância de Livro
        $clone = new Livro();
        //Cópia dos atributos do objeto atual para o novo objeto
        $clone->setNome($this->getNome());
        $clone->setAutor($this->getAutor());
        $clone->setNumeroPaginas($this->getNumeroPaginas());

        //O novo objeto idêntico ao objeto atual é retornado
        return $clone;
    }

    //Formata o objeto quando impresso.
    public function __toString()
    {
        $saida['nome'] = $this->getNome();
        $saida['autor'] = $this->getAutor();
        $saida['numeroPaginas'] = $this->getNumeroPaginas();

        return json_encode($saida);
    }
}
```

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

```php
class Trabalho implements Prototype
{
    private string $nome;
    private string $autor;
    private string $tipo;

    public function getNome(): string
    {
        return $this->nome;
    }

    public function setNome(string $nome): void
    {
        $this->nome = $nome;
    }

    public function getAutor(): string
    {
        return $this->autor;
    }

    public function setAutor(string $autor): void
    {
        $this->autor = $autor;
    }

    public function getTipo(): string
    {
        return $this->tipo;
    }

    public function setTipo(string $tipo): void
    {
        $this->tipo = $tipo;
    }

    //Método responsável por clonar o objeto Trabalho
    public function clone(): Prototype
    {
        //Criação de uma nova instância de Livro
        $clone = new Trabalho();
        //Cópia dos atributos do objeto atual para o novo objeto
        $clone->setNome($this->getNome());
        $clone->setAutor($this->getAutor());
        $clone->setTipo($this->getTipo());

        //O novo objeto idêntico ao objeto atual é retornado
        return $clone;
    }

   //Formata o objeto quando impresso.
    public function __toString()
    {
        $saida['nome'] = $this->getNome();
        $saida['autor'] = $this->getAutor();
        $saida['tipo'] = $this->getTipo();

        return json_encode($saida);
    }
}
```

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

```php
class Revista implements Prototype
{
    private string $nome;
    private int $edicao;

    public function getNome(): string
    {
        return $this->nome;
    }

    public function setNome(string $nome): void
    {
        $this->nome = $nome;
    }

    public function getEdicao(): int
    {
        return $this->edicao;
    }

    public function setEdicao(int $edicao): void
    {
        $this->edicao = $edicao;
    }

    //Método responsável por clonar o objeto Revista
    public function clone(): Prototype
    {
        //Criação de uma nova instância de Livro
        $clone = new Revista();
        //Cópia dos atributos do objeto atual para o novo objeto
        $clone->setNome($this->getNome());
        $clone->setEdicao($this->getEdicao());

        //O novo objeto idêntico ao objeto atual é retornado
        return $clone;
    }

    //Formata o objeto quando impresso.
    public function __toString()
    {
        $saida['nome'] = $this->getNome();
        $saida['edicao'] = $this->getEdicao();

        return json_encode($saida);
    }
}
```

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

Já temos os três protótipos que precisamos, agora vamos criar a classe responsável por gerenciar tais protótipos.

```php
class GerenciadorDePrototipos
{
    private array $prototipos;

    public function __construct()
    {
        //Criação de um objeto Livro
        $livro = new Livro();
        //Definição de seus valores iniciais
        $livro->setNome('Desconhecido');
        $livro->setAutor('Desconhecido');
        $livro->setNumeroPaginas(0);
        //Armazenamento do objeto como um protótipo que ficará
        //disponível no gerenciador a partir do identificador 'livro'
        $this->prototipos['livro'] = $livro;

        //Criação de um objeto Revista
        $revista = new Revista();
        //Definição de seus valores iniciais
        $revista->setNome('Desconhecido');
        $revista->setEdicao(0);
        //Armazenamento do objeto como um protótipo que ficará
        //disponível no gerenciador a partir do identificador 'revista'
        $this->prototipos['revista'] = $revista;

        //Criação de um objeto Trabalho
        $trabalho = new Trabalho();
        //Definição de seus valores iniciais
        $trabalho->setNome('Desconhecido');
        $trabalho->setAutor('Desconhecido');
        $trabalho->setTipo('Desconhecido');
        //Armazenamento do objeto como um protótipo que ficará
        //disponível no gerenciador a partir do identificador 'trabalho'
        $this->prototipos['trabalho'] = $trabalho;
    }

    public function getInstance(string $identificador): Prototype
    {
        return $this->prototipos[$identificador];
    }
}
```

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

Vamos ao teste:

```
echo '<pre>'; //Apenas para a saída ficar mais legível

//Criação do gerenciador de protótipos
$gerenciadorPrototipos = new GerenciadorDePrototipos();

//Solicitação dos protótipo de Livro ao gerenciador
$livroPrototipo = $gerenciadorPrototipos->getInstance('livro');

//Clonagem do protótipo
$livro1 = $livroPrototipo->clone();

//Impressão de livro 1 com valor padrão
echo '==== Livro 1 com valores padrão. ====<br>';
echo $livro1;

//Edição da informações de livro 1
$livro1->setNome('Livro 1');
$livro1->setAutor('Lucas da Silva');
$livro1->setNumeroPaginas(325);

//Impressão de livro 1 com valores editados
echo '<br><br>==== Livro 1 com valores editados. ====<br>';
echo $livro1;

//Clonagem de mais um livro
$livro2 = $livroPrototipo->clone();

//Impressão de livro 2 com valores padrão
echo '<br><br>==== Livro 2 com valores padrão. ====<br>';
echo $livro2;

//Edição da informações de livro 2
$livro2->setNome('Livro 2');
$livro2->setAutor('Marlene dos Santos');
$livro2->setNumeroPaginas(420);

//Impressão de livro 1 e 2 com valores editados
echo '<br><br>==== Livro 1 e 2 com valores editados. ====<br>';
echo $livro1;
echo '<br>';
echo $livro2;

echo '</pre>'; //Apenas para a saída ficar mais legível;
```

Repare que o único **new** utilizado no teste foi para criar o gerenciador de protótipos. Todos os dois objetos **Livro** foram criados com base na clonagem de seu protótipo.

O mesmo procedimento de teste seria válido para os objetos **Revista** e **Trabalho**.

Saída:

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

```
==== Livro 1 com valores padrão. ====
{"nome":"Desconhecido","autor":"Desconhecido","numeroPaginas":0}

==== Livro 1 com valores editados. ====
{"nome":"Livro 1","autor":"Lucas da Silva","numeroPaginas":325}

==== Livro 2 com valores padrão. ====
{"nome":"Desconhecido","autor":"Desconhecido","numeroPaginas":0}

==== Livro 1 e 2 com valores editados. ====
{"nome":"Livro 1","autor":"Lucas da Silva","numeroPaginas":325}
{"nome":"Livro 2","autor":"Marlene dos Santos","numeroPaginas":420}
```

Os objetos **livro1** e **livro2** são completamente independentes, mudanças no **livro1** um não causa mudanças no **livro2** e vice-versa.

Aplicabilidade (Quando utilizar?)

O padrão *Prototype* pode ser utilizado quando um sistema deve ser independente de como seus produtos são criados, compostos e representados. este padrão também pode ser utilizado:

- Quando as classes a serem instanciadas são especificadas em tempo de execução, por exemplo, por carregamento dinâmico.
- Para evitar a construção de uma hierarquia de classes de fábricas que seja paralela à hierarquia de classes de produtos.
- Quando instâncias de uma classe podem ter apenas algumas poucas combinações diferentes de estado. Pode ser mais conveniente criar um número correspondente de protótipos e cloná-los, em vez de instanciar as classes manualmente, cada vez com o estado apropriado.

Componentes

- **Prototype:** Declara uma interface para clonar a si próprio.
- **PrototypeConcreto:** Implementa uma operação para clonar a si próprio.
- **Cliente:** Cria um novo objeto solicitando a um protótipo que clone a si próprio.

Licensed to Bianca Maciel Brasilino - bianca_maciel.c@hotmail.com

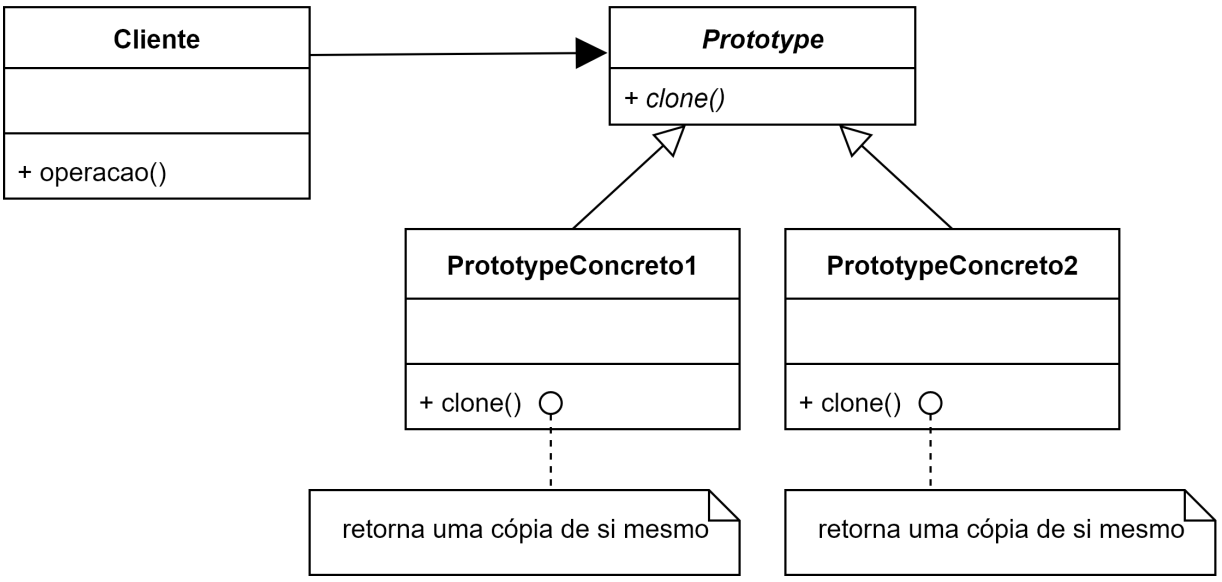


Diagrama de Classes

Consequências

O padrão *Prototype*, assim como os padrões *Abstract Factory* e o *Builder* oculta as classes de produtos concretos do cliente, reduzindo assim o número de classes que os clientes conhecem. Além disso, esses padrões permitem que um cliente trabalhe com classes específicas de uma aplicação sem a necessidade de modificação.

O *Prototype* também:

- Faz com que novos tipos de protótipos possam ser registrados pelos clientes em tempo de execução, aumentando a quantidade de tipos de objetos que podem ser clonados.
- Permite que objetos sejam clonados sem acoplá-los a suas classes concretas.
- Possibilita que códigos de inicialização repetidos sejam substituídos por processos de clonagem de objetos protótipos pré-construídos.
- Torna a produção de objetos complexos mais simples.
- É uma alternativa à herança quando é preciso lidar com configurações pré determinadas para objetos complexos. Isso reduz o número de subclasses.
- A clonagem de objetos que possuem muitas referências a outros objetos ou referências circulares pode ser muito complexa.