Memento

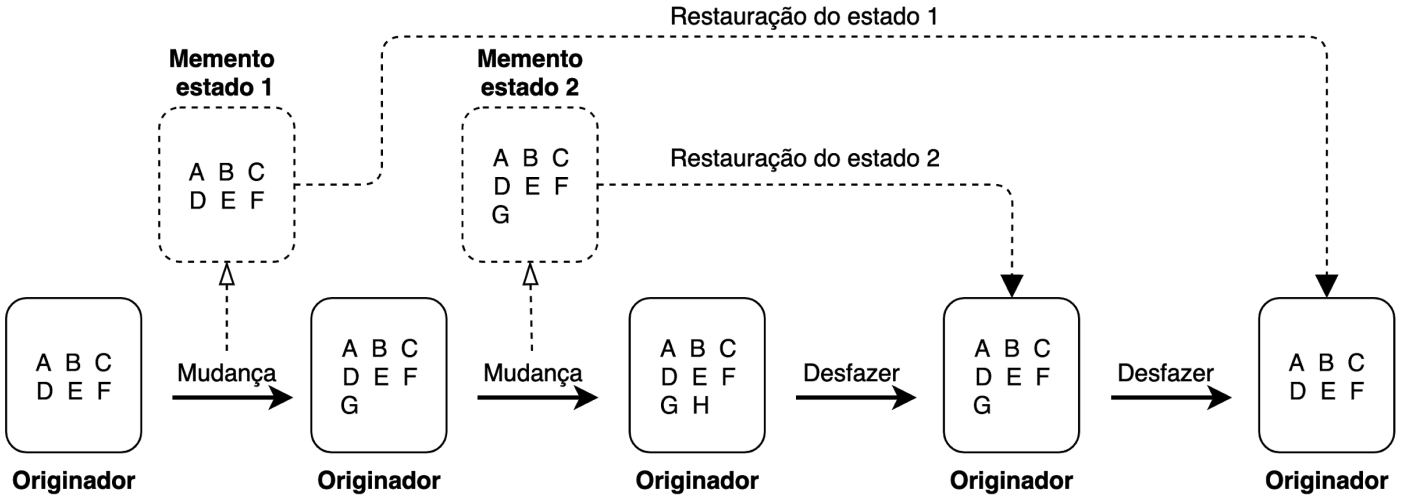
Padrões Comportamentais

O padrão *Memento* permite capturar e externalizar um estado interno de um objeto sem violar o encapsulamento, deste modo, o objeto pode ser restaurado no futuro para este estado capturado.

Motivação (Por que utilizar?)

Às vezes é necessário registrar o estado interno de um objeto. Isso é necessário ao implementar pontos de verificação (*checkpoints*) que possibilitam operações de retroceder (conhecidas pelo atalho de teclado Ctrl + Z) e operações para recuperação de erros ocorridos após a criação do *checkpoint*. Tais informações devem ser armazenadas em em algum lugar, de modo que seja possível restaurar objetos aos seus estados prévios. Em programação orientada a objetos é muito comum a criação de atributos privados, que são inacessíveis de fora do objeto, isso impossibilita que tais atributos sejam salvos externamente, e os expor violaria o encapsulamento, uma vez que tal exposição pode comprometer a confiabilidade e a extensibilidade da classe em questão.

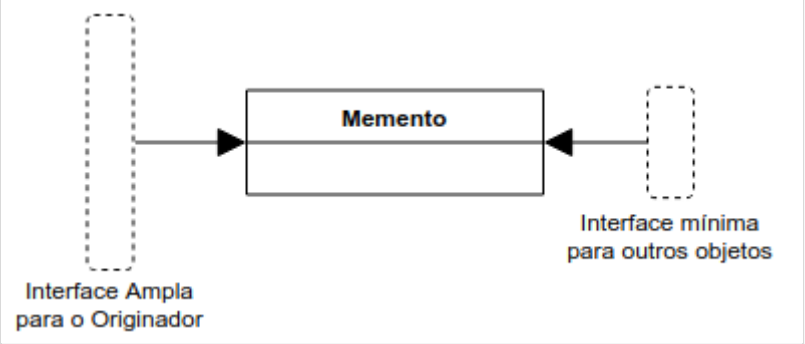
A situação descrita acima poderia ser resolvida com o padrão *Memento*. Um ***Memento*** é um objeto que **armazena um *checkpoint* do estado de um objeto**, este objeto é chamado de **Originador**. Sempre que for necessário restaurar o estado interno de um Originador um *Memento* será utilizado para isso. Podemos dizer que um *Memento* é uma recordação do estado de um objeto originador no passado.



***Mementos* de um objeto originador**

Na imagem podemos identificar que cada mudança no objeto Originador gera um *Memento* que poderá ser utilizado no futuro para restaurar o estado do Originador.

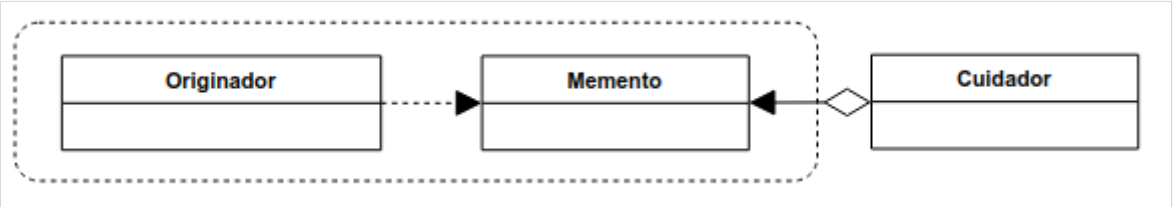
A implementação de um *Memento* exige duas interfaces: Uma ampla, para os Originadores, e uma mínima, para outros objetos.



Interfaces de um *Memento*

Existem duas maneiras de implementar o padrão *Memento*:

Implementação baseada em classes aninhadas: A implementação clássica do padrão requer a utilização de classes aninhadas, ou seja, classes dentro de outras classes. Tal recurso está disponível em linguagens de programação populares no mercado, tais como Java, C++ ou C#.



Representação do padrão *Memento* implementado utilizando classes aninhadas

Nessa implementação, a classe ***Memento*** está aninhada dentro da classe **Originador**.

```
public class Originador {
    //Restante do código

    private class Memento {
        //Restante do código
    }

    //Restante do código
}
```

Exemplo de classes aninhadas em Java

Isso permite que o **Originador** acesse os campos e métodos da classe **Memento**, mesmo que eles tenham sido declarados privados, ou seja, possui uma interface ampla de acesso ao **Memento**. Já a classe **Cuidador**, que aqui representa qualquer classe externa, tem um acesso muito limitado ao **Memento** (Interface mínima). Deste modo a classe **Cuidador** pode armazenar os **Mementos**, mas não pode alterar seus estados.

Iremos escrever os códigos para duas formas diferentes de implementação do padrão *Memento* que serão descritas a seguir. Considere que precisamos manter salvas as modificações feitas em uma caixa de texto. Tal caixa de texto possui os seguintes atributos:

- `x`: Posição no eixo vertical de um plano bidimensional;
- `y`: Posição no eixo horizontal de um plano bidimensional;
- `text`: Texto da caixa de texto;
- `fontFamily`: Fonte utilizada no texto;
- `fontSize`: Tamanho da fonte do texto;
- `textAlign`: Alinhamento do texto;
- `fontWeight`: Indica se o texto deve estar em negrito ou não.

Nota: Seguiremos as propriedades do CSS, caso não tenha conhecimentos a respeito de CSS não se preocupe, cada propriedade estará explicada nos testes que iremos realizar.

1 - Implementação baseada em uma interface intermediária: Algumas linguagens de programação também muito populares como PHP ou Javascript não possuem suporte a classes aninhadas. Na ausência de tal recurso, pode se estabelecer uma convenção para que `Cuidadores` ou qualquer outra classe externa utilizem uma interface intermediária que limita o acesso ao `Memento`.

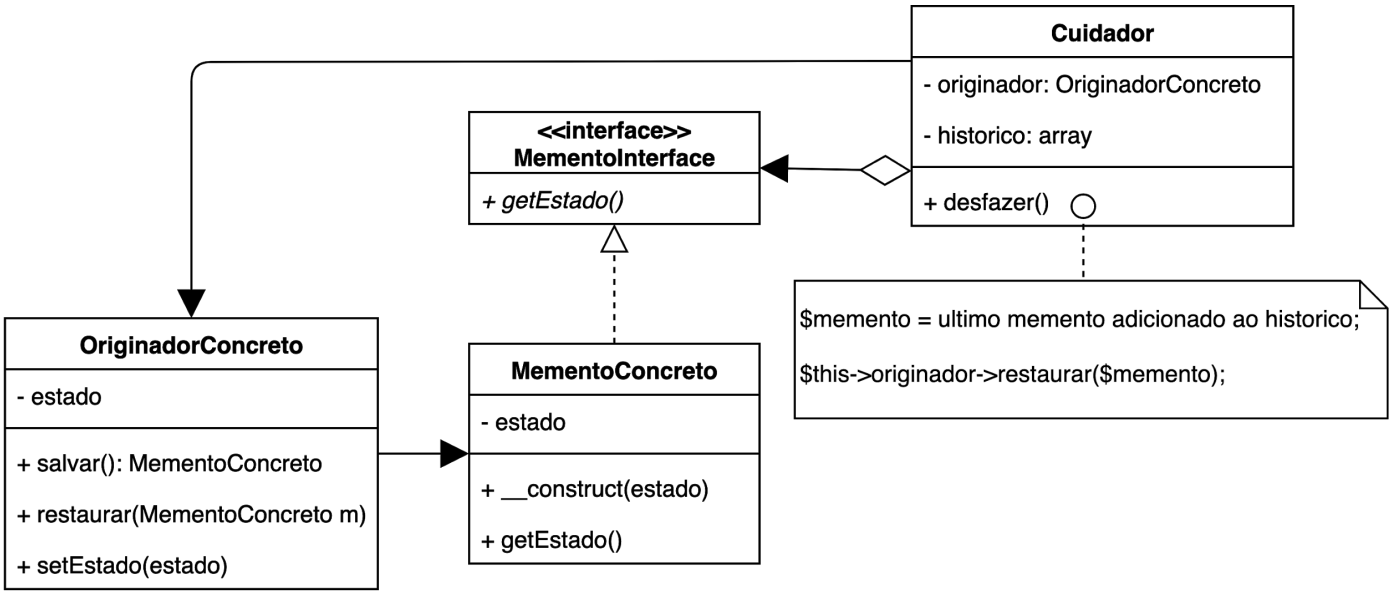


Diagrama de classes do *Memento* implementado utilizando uma interface intermediária

Nesta implementação não existem classes aninhadas. A classe `OriginadorConcreto` consome uma interface ampla fornecida pela classe `MementoConcreto` enquanto a classe `Cuidador`, que aqui representa qualquer classe externa, tem um acesso muito limitado ao `MementoConcreto` (Interface mínima) que se dá pela interface `MementoInterface`. Repare que a classe `OriginadorConcreto` possui o método `restaurar()` que restaura o estado do objeto originador a partir de um objeto do tipo `MementoInterface`, por esse motivo, a classe `Cuidador` precisa de uma referência ao objeto de `OriginadorConcreto` para ser capaz de restaurá-lo.

Trazendo para o nosso exemplo o diagrama fica da seguinte forma:

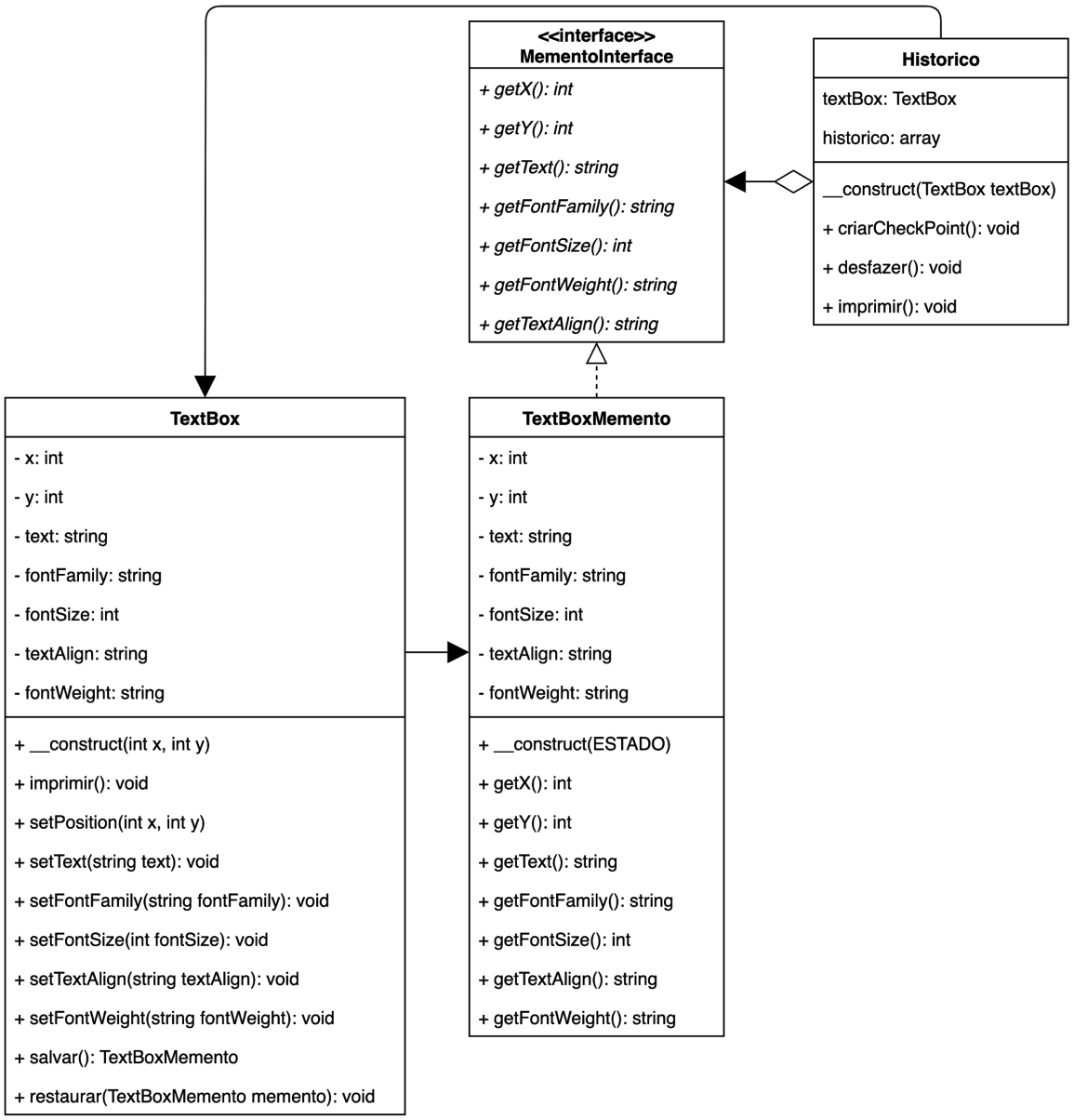


Diagrama de classes do exemplo utilizando o *Memento* com uma interface intermediária

Vamos começar implementando a interface `MementoInterface`. Ela deverá garantir o acesso a todos os métodos necessários para recuperar os valores salvos (estado) no `MementoConcreto`.

```php
interface MementoInterface
{
    public function getX(): int;

    public function getY(): int;

    public function getText(): string;

    public function getFontFamily(): string;

    public function getFontSize(): int;

    public function getTextAlign(): string;

    public function getFontWeight(): string ;
}
```

Agora o MementoConcreto.

```php
class TextBoxMemento implements MementoInterface
{
    private int $x;
    private int $y;
    private string $text;
    private string $fontFamily;
    private int $fontSize;
    private string $textAlign;
    private string $fontWeight;

    //Recebe todos os seus dados em seu construtor. Uma vez criado ele não muda mais.
    public function __construct(
        int $x,
        int $y,
        string $text,
        string $fontFamily,
        string $fontSize,
        string $textAlign,
        string $fontWeight
    ) {
        $this->x = $x;
        $this->y = $y;
        $this->text = $text;
        $this->fontFamily = $fontFamily;
        $this->fontSize = $fontSize;
        $this->textAlign = $textAlign;
        $this->fontWeight = $fontWeight;
    }

    public function getX(): int
    {
        return $this->x;
    }

    public function getY(): int
    {
        return $this->y;
    }

    public function getText(): string
    {
        return $this->text;
    }

    public function getFontFamily(): string
    {
        return $this->fontFamily;
    }

    public function getFontSize(): int
    {
        return $this->fontSize;
    }

    public function getTextAlign(): string
    {
        return $this->textAlign;
    }

    public function getFontWeight(): string
    {
        return $this->fontWeight;
    }
}
```

Já temos a interface MementoInterface e a classe concreta MementoConcreto. Vamos ver como será nossa classe originadora, que no nosso caso é a classe TextBox.

```
class TextBox
{
    private int $x;
    private int $y;
    private string $text;
    private string $fontFamily;
    private int $fontSize;
    private string $textAlign;
    private string $fontWeight;

    //Recebe as posições X e Y em seu construtores. Os Demais recebem valores padrão.
    public function __construct(int $x, int $y)
    {
        $this->x = $x;
        $this->y = $y;
        $this->text = '';
        $this->fontFamily = 'Arial';
        $this->fontSize = 14;
        $this->textAlign = 'left';
        $this->fontWeight = 'normal';
    }


    //Este método imprime o textBox no navegador.
    //Serve apenas para que seja possível visualizar o resultado das mudanças.
    public function imprimir(): void
    {
        echo "<div style='margin-left: {$this->x}px; margin-top: {$this->x}px;'>
              <span style='font-size: {$this->fontSize}px;
                           font-family: $this->fontFamily;
                           font-weight: $this->fontWeight'>
                $this->text
              </span>
              </div>";
    }

    //Os setters abaixo servem para modificar as propriedades do textBox

    public function setPosition(int $x, int $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    public function setText(string $text): void
    {
        $this->text = $text;
    }

    public function setFontFamily(string $fontFamily): void
    {
        $this->fontFamily = $fontFamily;
    }
```

```
    public function setFontSize(int $fontSize): void
    {
        $this->fontSize = $fontSize;
    }

    public function setTextAlign(string $textAlign): void
    {
        $this->textAlign = $textAlign;
    }

    public function setFontWeight(string $fontWeight): void
    {
        $this->fontWeight = $fontWeight;
    }

    //Este método é o responsável por salvar o atual estado do textBox em um Memento
    public function salvar(): MementoInterface
    {
        return new TextBoxMemento(
            $this->x,
            $this->y,
            $this->text,
            $this->fontFamily,
            $this->fontSize,
            $this->textAlign,
            $this->fontWeight
        );
    }


    //Este método restaura um antigo estado do textBox a partir de um Memento
    public function restaurar(MementoInterface $memento)
    {
        $this->x = $memento->getX();
        $this->y = $memento->getY();
        $this->text = $memento->getText();
        $this->fontFamily = $memento->getFontFamily();
        $this->fontSize = $memento->getFontSize();
        $this->textAlign = $memento->getTextAlign();
        $this->fontWeight = $memento->getFontWeight();
    }
}
```

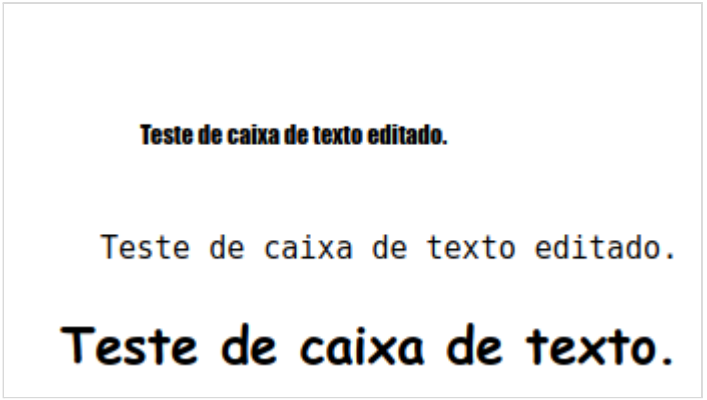
Agora é a vez do **Historico**, ele é nosso **Cuidador**.

```php
class Historico
{
    private TextBox $textBox; //Referência a TextBox (Originador)
    private array $historico = []; //Array que mantem todos os Mementos criados.

    //Recebe um TextBox como parâmetro e guarda referência a ele.
    public function __construct(TextBox $textBox)
    {
        $this->textBox = $textBox;
    }

    //Chama o método salvar de TextBox e empilha o Memento resultante no array historico.
    //Este método salva o estado atual do textBox recebido no construtor desta classe.
    public function criarCheckPoint(): void
    {
        $this->historico[] = $this->textBox->salvar();
    }

    //restaura um estado antigo do textBox recebido no construtor desta classe.
    public function desfazer(): void
    {
        //Se o historico não estiver vazio.
        if (!count($this->historico)) {
            return;
        }

        //Desempilha o último Memento inserido no array historico.
        $memento = array_pop($this->historico);
        //Chama o método restaurar de TextBox passando o Memento como parâmetro.
        $this->textBox->restaurar($memento);
    }

    //Imprime o textBox no navegador.
    public function imprimir(): void
    {
        $this->textBox->imprimir();
    }
}
```

Hora de testar:

```
//Criação de um TextBox com posição x=20 e y=100.
$textBox = new TextBox(20, 100);

//Criação de um histórico passando o $textBox como parâmetro.
$historico = new Historico($textBox);

//Edição de alguns valores de textBox.
$textBox->setText('Teste de caixa de texto.'); //Inserção de um texto.
$textBox->setFontWeight('bold'); //bold indica que o texto deve estar em negrito.
$textBox->setFontFamily('Cursive'); //Cursive é o nome da fonte a ser utilizada.
$textBox->setFontSize(25); //A fonte deve ter um tamanho de 25px.

//Salvamento do estado atual de $textBox.
$historico->criarCheckPoint(); //Um memento foi empilhado no historico.

//Edição de alguns valores de textBox.
$textBox->setText('Teste de caixa de texto editado.'); //Mudança no texto.
$textBox->setFontWeight('normal'); //Normal indica que o texto não deve estar em negrito.
$textBox->setFontFamily('monospace'); //monospace é o nome da fonte a ser utilizada.
$textBox->setFontSize(15); //A fonte deve ter um tamanho de 15px.
$textBox->setPosition(40, 110); //A posição x deve ser 40 e y deve ser 110;

//Salvamento do estado atual de $textBox.
$historico->criarCheckPoint(); //Mais um memento foi empilhado no historico.

//Edição de alguns valores de textBox.
$textBox->setFontFamily('fantasy'); //fantasy é o nome da fonte a ser utilizada.
$textBox->setFontSize(12); //A fonte deve ter um tamanho de 12px.
$textBox->setPosition(60, 120); //A posição x deve ser 60 e y deve ser 120;

$historico->imprimir(); //Imprime o estado atual do textBox.

$historico->desfazer(); //Restaura o textBox ao ultimo estado salvo.

$historico->imprimir(); //Imprime o estado atual do textBox.

$historico->desfazer(); //Restaura o textBox ao penúltimo estado salvo.

$historico->imprimir(); //Imprime o estado atual do textBox.
```

Saída:



A utilização de uma interface intermediária é uma convenção, portanto, não há garantias que uma classe externa não possa acessar o MementoConcreto (TextBoxMemento) diretamente ignorando a interface intermediária.

Repare que nessa forma de implementação o Cuidador (Historico) depende de um OriginadorConcreto (TextBox), deste modo um mesmo Cuidador não é capaz de gerenciar os estados de mais de um OriginadorConcreto.

Também é importante citar que dependendo da forma que um MementoConcreto for implementado, pode ser que o Cuidador seja capaz de acessar ou até mesmo modificar a classe OriginadorConcreto através dos métodos definidos pela interface MementoInterface. Para fechar essa brecha podemos restringir ainda mais o encapsulamento criando mais uma interface, agora para a classe Originador.

2 - Implementação com encapsulamento mais restritivo: Nesse tipo de implementação o Cuidador deixa de ser dependente do OriginadorConcreto. Antes tal dependendência era necessária, pois o método restaurar() de OriginadorConcreto era utilizado pelo método desfazer() do Cuidador.

Veja no diagrama a seguir que agora cada MementoConcreto mantém referência ao objeto OriginadorConcreto que o gerou, assim, o método restaurar() sai do OriginadorConcreto e vai para o MementoConcreto (TextBoxMemento). As informações necessárias (estados) já estarão disponíveis dentro do MementoConcreto e ele sabe qual objeto deve ser restaurado, isso dispensa a necessidade dos métodos de recuperação de estados na interface MementoInterface, antes representados por getEstado().

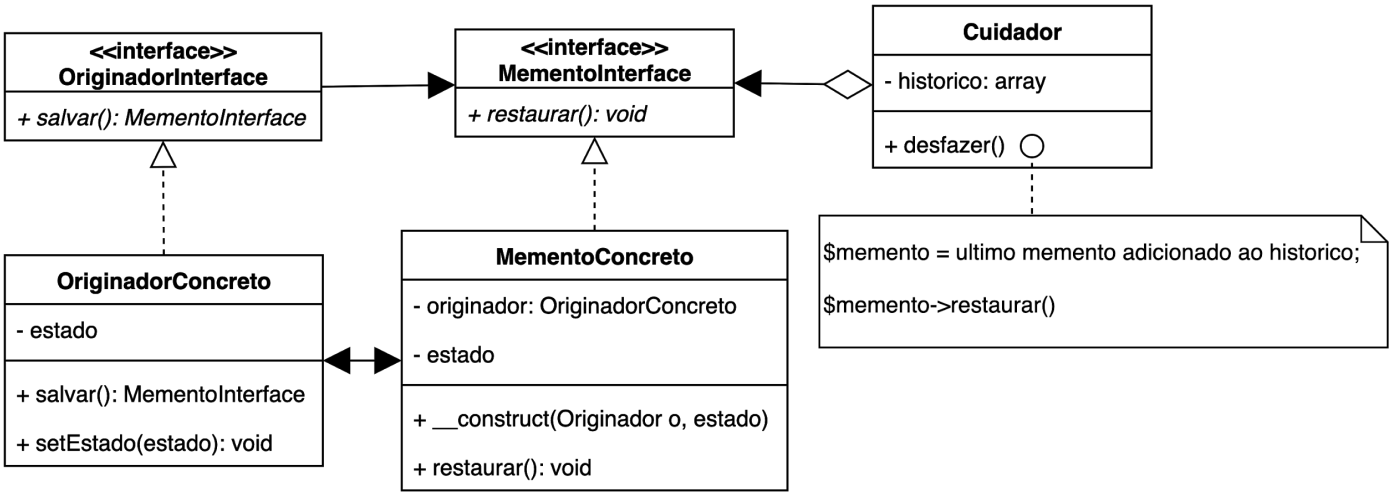


Diagrama de classes do padrão *Memento* com encapsulamento mais restritivo

De forma resumida, cada objeto MementoConcreto tem dentro dele o estado salvo do OriginadorConcreto, e é capaz de restaurar tal estado no objeto OriginadorConcreto pois ele o conhece.

Trazendo para nosso exemplo temos o seguinte:

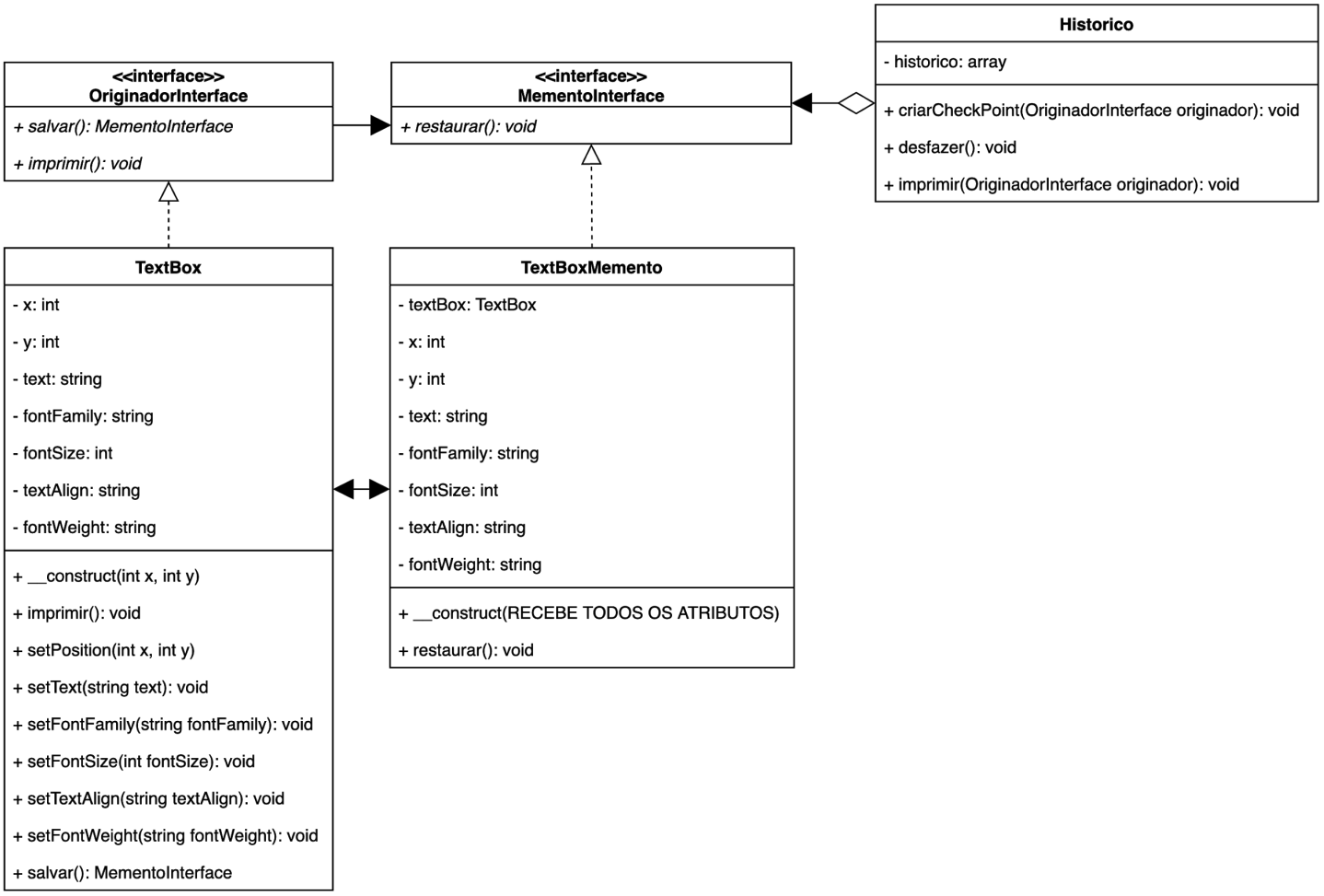


Diagrama de classes do exemplo utilizando o *Memento* com encapsulamento mais restritivo

Vamos a implementação:

```php
interface MementoInterface
{
    public function restaurar(): void;
}
```

```php
class TextBoxMemento implements MementoInterface
{
    private TextBox $textBox; //Mantém uma referência ao seu Originador.
    private int $x;
    private int $y;
    private string $text;
    private string $fontFamily;
    private int $fontSize;
    private string $textAlign;
    private string $fontWeight;

    public function __construct(
        TextBox $textBox,
        int $x,
        int $y,
        string $text,
        string $fontFamily,
        string $fontSize,
        string $textAlign,
        string $fontWeight
    ) {
        $this->textBox = $textBox;
        $this->x = $x;
        $this->y = $y;
        $this->text = $text;
        $this->fontFamily = $fontFamily;
        $this->fontSize = $fontSize;
        $this->textAlign = $textAlign;
        $this->fontWeight = $fontWeight;
    }

    //Utiliza-se os métodos setters do Originador para restaurá-lo.
    public function restaurar(): void
    {
        $this->textBox->setPosition($this->x, $this->y);
        $this->textBox->setText($this->text);
        $this->textBox->setFontFamily($this->fontFamily);
        $this->textBox->setFontSize($this->fontSize);
        $this->textBox->setTextAlign($this->textAlign);
        $this->textBox->setFontWeight($this->fontWeight);
    }
}
```

Mesmo que **TextBox** (Originador) seja recebido por parâmetro no construtor de **TextBoxMemento** (**MementoConcreto**), ainda é necessário passar os demais atributos para o construtor. Todos eles são privados na classe **TextBox** e ela não fornece os *getters* para que tais atributos sejam acessados de fora dela, portanto, a classe **TextBoxMemento** não seria capaz de extrair tais informações do **TextBox** recebido em seu construtor.

Vamos agora aos códigos referentes ao Originador.

```php
interface OriginadorInterface
{
    public function salvar(): MementoInterface;

    public function imprimir(): void; //Este método não faz parte do padrão.
}
```

```php
class TextBox implements OriginadorInterface
{
    private int $x;
    private int $y;
    private string $text;
    private string $fontFamily;
    private int $fontSize;
    private string $textAlign;
    private string $fontWeight;

    public function __construct(int $x, int $y)
    {
        $this->x = $x;
        $this->y = $y;
        $this->text = '';
        $this->fontFamily = 'Arial';
        $this->fontSize = 14;
        $this->textAlign = 'left';
        $this->fontWeight = 'normal';
    }

    public function imprimir(): void
    {
        echo "<div style='margin-left: {$this->x}px; margin-top: {$this->x}px;'>
              <span style='font-size: {$this->fontSize}px;
                           font-family: $this->fontFamily;
                           font-weight: $this->fontWeight'>
                $this->text
              </span>
              </div>";
    }

    public function setPosition(int $x, int $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    public function setText(string $text): void
    {
        $this->text = $text;
    }

    public function setFontFamily(string $fontFamily): void
    {
        $this->fontFamily = $fontFamily;
    }
```

```
    public function setFontSize(int $fontSize): void
    {
        $this->fontSize = $fontSize;
    }

    public function setTextAlign(string $textAlign): void
    {
        $this->textAlign = $textAlign;
    }

    public function setFontWeight(string $fontWeight): void
    {
        $this->fontWeight = $fontWeight;
    }

    public function salvar(): MementoInterface
    {
        return new TextBoxMemento(
            $this, //Agora a classe também se passa como parâmetro para o MementoConcreto.
            $this->x,
            $this->y,
            $this->text,
            $this->fontFamily,
            $this->fontSize,
            $this->textAlign,
            $this->fontWeight
        );
    }
}
```

Como explicado anteriormente o método **restaurar()** está na interface **MementoInterface**, deste modo, cada **MementoConcreto** sabe como restaurar o objeto que o criou, pois mantém um referência a ele. Sendo assim, o **Cuidador** pode agora gerenciar **MementosConcretos** distintos vindos de **Originadores** distintos de forma totalmente independente, baseando-se apenas nas interfaces **MementoInterface** e **OriginadorInterface** que possuem os métodos **restaurar()** e **Salvar()** respectivamente.

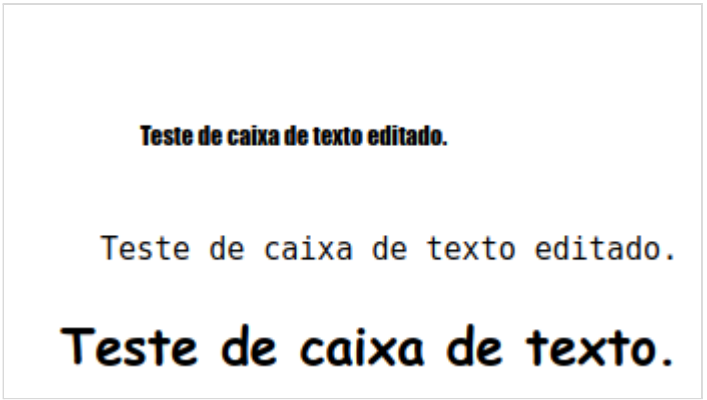
Vejamos como fica o **Historico** (**Cuidador**) nesta implementação:

```php
class Historico
{
    private array $historico = [];

    /*Agora o $originador a ser salvo é recebido por parâmetro.
      A classe Historico não precisa mais de uma referência permanente a ele.
      Qualquer Orginador é aceito, desde que implemente a interface OriginadorInterface*/
    public function criarCheckPoint(OriginadorInterface $originador): void
    {
        $this->historico[] = $originador->salvar();
    }

    //Restaura o Originador a parte do memento no topo da pilha (array) historico.
    public function desfazer(): void
    {
        //Se o array historico não estiver vazio.
        if (!count($this->historico)) {
            return;
        }

        //Remova o memento do topo da pilha historico e o restaure.
        $memento = array_pop($this->historico);
        /*Anteriormente a restauração era feita pelo TextBox (originador)
          Agora ela é feita pelo Memento*/
        $memento->restaurar();
    }

    //Apenas imprime o TextBox no navegador.
    public function imprimir(OriginadorInterface $originador): void    {
        $originador->imprimir();
    }
}
```

Vamos ao teste:

```
//Criação de um TextBox com posição x=20 e y=100.
$textBox = new TextBox(20, 100);

//Criação de um histórico, agora sem passar o $textBox como parâmetro.
$historico = new Historico();

//Edição de alguns valores de textBox.
$textBox->setText('Teste de caixa de texto.'); //Inserção de um texto.
$textBox->setFontWeight('bold'); //bold indica que o texto deve estar em negrito.
$textBox->setFontFamily('Cursive'); //Cursive é o nome da fonte a ser utilizada.
$textBox->setFontSize(25); //A fonte deve ter um tamanho de 25px.

//Salvamento do estado atual de $textBox. Agora recebe o $textBox por parâmetro.
$historico->criarCheckPoint($textBox); //Um memento foi empilhado no historico.

//Edição de alguns valores de textBox.
$textBox->setText('Teste de caixa de texto editado.'); //Mudança no texto.
$textBox->setFontWeight('normal'); //Normal indica que o texto não deve estar em negrito.
$textBox->setFontFamily('monospace'); //monospace é o nome da fonte a ser utilizada.
$textBox->setFontSize(15); //A fonte deve ter um tamanho de 15px.
$textBox->setPosition(40, 110); //A posição x deve ser 40 e y deve ser 110;

//Salvamento do estado atual de $textBox. Recebendo o $textBox por parâmetro.
$historico->criarCheckPoint($textBox); //Mais um memento foi empilhado no historico.

//Edição de alguns valores de textBox.
$textBox->setFontFamily('fantasy'); //fantasy é o nome da fonte a ser utilizada.
$textBox->setFontSize(12); //A fonte deve ter um tamanho de 12px.
$textBox->setPosition(60, 120); //A posição x deve ser 60 e y deve ser 120;

$historico->imprimir(); //Imprime o estado atual do textBox.

$historico->desfazer(); //Restaura o textBox ao ultimo estado salvo.

$historico->imprimir(); //Imprime o estado atual do textBox.

$historico->desfazer(); //Restaura o textBox ao penúltimo estado salvo.

$historico->imprimir(); //Imprime o estado atual do textBox.
```

Saída:



Chegamos no mesmo resultado, mas agora com um código mais seguro e flexível.

Aplicabilidade (Quando utilizar?)

- Quando uma captura instantânea (*screenshot*), total ou parcial, do estado de um objeto deve ser salva para que no futuro tal objeto possa ser restaurado para este estado salvo.

- Quando se deseja evitar uma interface direta para obtenção do estado atual do objeto, de modo que ela exponha os detalhes de sua implementação e quebre o encapsulamento.

Componentes

- **Memento:**

    - O ***Memento*** armazena o estado interno do objeto **Originador**. Ele pode armazenar muito ou pouco do estado interno do **Originador**, isso varia conforme as necessidades e critérios do **Originador**.

    - Protege seu estado contra acessos feitos por objetos que não sejam o objeto **Originador**. O ***Memento*** têm efetivamente duas interfaces. O **Cuidador** vê uma interface mínima do ***Memento***. Ele só pode passar o ***Memento*** para outros objetos. O **Originador**, por outro lado, vê uma interface ampla, que permite acessar todos os dados necessários para restaurar seu estado anterior. Idealmente, somente o **Originador** que produziu o ***Memento*** teria permissão para acessar o estado interno dele.

    - É uma prática comum fazer o ***Memento*** imutável passando todos os dados por meio do construtor.

- **Originador:** Cria um ***Memento*** que contém um screenshot de seu estado interno atual. Utiliza o ***Memento*** para restaurar seu estado interno.

- **Cuidador:** É responsável pela custódia do ***Memento***, ele nunca consulta ou manipula o conteúdo interno de um ***Memento***. O **Cuidador** pode também manter registros do histórico do **Originador** armazenando os ***Mementos*** em um pilha e os recuperando de maneira apropriada para restaurar o **Originador**.

Observação: Os diagramas de classes das implementações alternativas foram devidamente apresentados e explicados na seção de motivação.

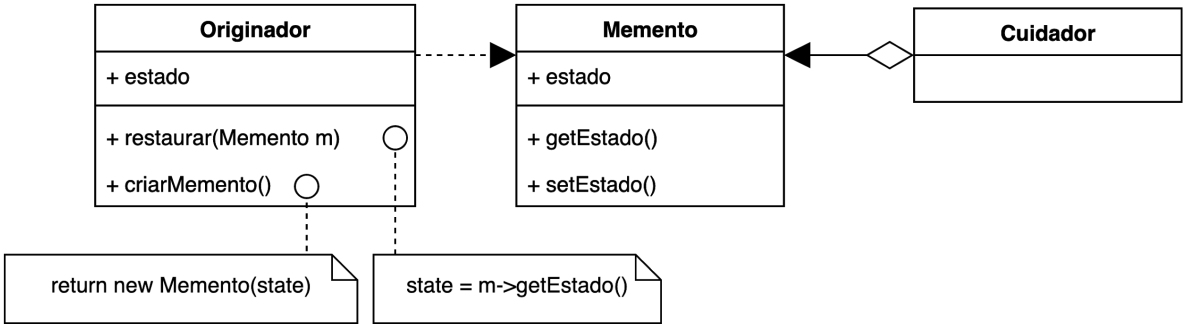


Diagrama de Classes (básico - aninhamento de classes)

Consequências

- O *Memento* preserva o encapsulamento evitando a exposição de informações que somente um originador deveria gerenciar, mas que, contudo, devem ser armazenadas nele.

- Em outras metodologias de preservação de encapsulamento, o objeto Originator mantém as versões do estado interno que os clientes solicitaram. Isso coloca toda a carga de gerenciamento de armazenamento no Originator. O fato de os clientes gerenciarem o estado que solicitam simplifica o Originador e impede que os clientes notifiquem os originadores quando terminarem.

- Usar *Mementos* pode ser custoso. *Mementos* podem sofrer uma sobrecarga considerável se o Originador precisar copiar grandes quantidades de informações para armazenar no *Memento* ou se os clientes criarem e retornarem *Mementos* ao originador com muita frequência. A menos que encapsular e restaurar o estado do Originador seja barato, o padrão pode não ser apropriado.

- Definindo interfaces estreitas e amplas. Em algumas liguagens, pode ser difícil garantir que apenas o originador possa acessar o estado do *memento*.

- Um Cuidador é responsável por excluir os *Mementos* os quais ele tem custódia. No entanto, o cuidador não tem idéia do volume ocupado pelo estado do *memento*. Portanto, um Cuidador de peso leve pode se tornar custoso ao armazenar muitos *mementos*.