Builder

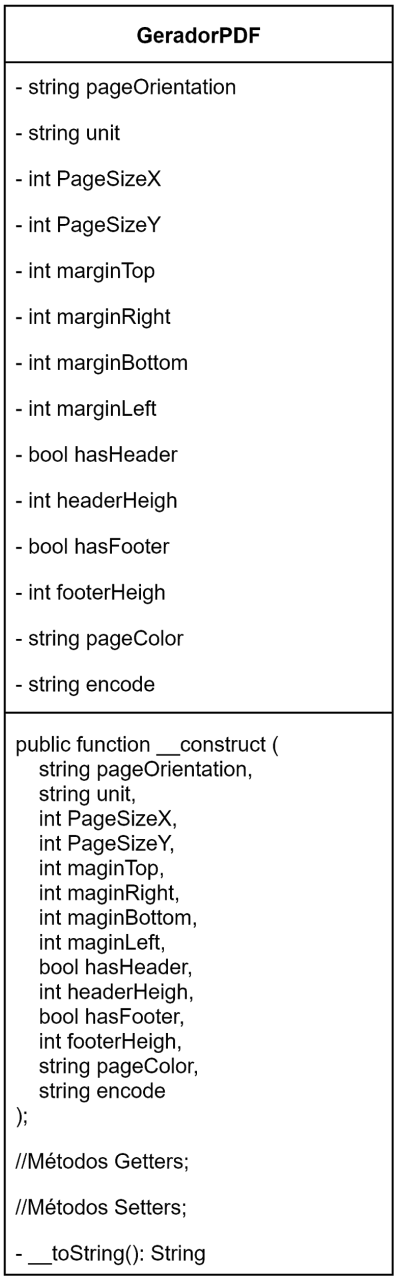
Padrões Criacionais

O padrão *Builder* separa a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção de um objeto possa criar diferentes representações.

Motivação (Por que utilizar?)

Durante o desenvolvimento de um software podemos nos deparar com objetos muito complexos de se construir, ou então um objeto que pode ser construído de muitas formas diferentes.

Vamos direto a um exemplo onde poderá ver claramente um objeto que possui muitos parâmetros em seu construtor.



Classe GeradorPDF

Na classe ao lado é possível notar a grande quantidade de atributos existentes. Existem infinitas combinações de valores que podem ser assumidos por eles, o que acarretaria na criação de objetos diferentes.

Não se trata de uma classe complexa, a classe GeradorPDF possui 14 atributos, todos são inicializados com algum valor e são redefinidos no construtor da classe, isso implica que tal construtor precisa de 14 parâmetros para criar um objeto da classe GeradorPDF. A classe pode não ser complexa, porém, a instanciação de seus objetos é trabalhosa e muito extensa.

Além dos atributos e método construtor, a classe GeradorPDF também possui um método *Getter* e outro método *Setter* para cada um de seus atributos, também possui um método __toString() que formata o objeto antes de sua impressão. Deste modo, existem nessa classe 28 métodos (*Getters* e *Setters*) mais o construtor.

Veja a seguir o código completo da classe GeradorPDF. Como citado anteriormente é uma classe extensa, porém simples, então não se assuste ao ver o tamanho dela.

```php
class GeradorPDF
{
    //Cada atributo é inicializado com um valor padrão.
    private string $pageOrientation = 'portrait';
    private string $unit = 'mm';
    private int $PageSizeX = 210;
    private int $PageSizeY = 297;
    private int $marginTop = 30;
    private int $marginRight = 20;
    private int $marginBottom = 30;
    private int $marginLeft = 20;
    private bool $hasHeader = false;
    private int $headerHeigh = 0;
    private bool $hasFooter = false;
    private int $footerHeigh = 0;
    private string $pageColor = '#ffffff';
    private string $encode = 'UTF-8';

    //Todos os atributos são exigidos no construtor.
    public function __construct (
        string $pageOrientation,
        string $unit,
        int $PageSizeX,
        int $PageSizeY,
        int $maginTop,
        int $maginRight,
        int $maginBottom,
        int $maginLeft,
        bool $hasHeader,
        int $headerHeigh,
        bool $hasFooter,
        int $footerHeigh,
        string $pageColor,
        string $encode
    ) {
        $this->pageOrientation = $pageOrientation;
        $this->unit = $unit;
        $this->PageSizeX = $PageSizeX;
        $this->PageSizeY = $PageSizeY;
        $this->marginTop = $maginTop;
        $this->marginRight = $maginRight;
        $this->marginBottom = $maginBottom;
        $this->marginLeft = $maginLeft;
        $this->hasHeader = $hasHeader;
        $this->headerHeigh = $headerHeigh;
        $this->hasFooter = $hasFooter;
        $this->footerHeigh = $footerHeigh;
        $this->pageColor = $pageColor;
        $this->encode = $encode;
    }

    public function getPageOrientation(): string
    {
        return $this->pageOrientation;
    }

    public function setPageOrientation(string $pageOrientation): void
    {
        $this->pageOrientation = $pageOrientation;
    }

    public function getUnit(): string
    {
        return $this->unit;
    }

    public function setUnit(string $unit): void
    {
        $this->unit = $unit;
    }

    public function getPageSizeX(): int
    {
        return $this->PageSizeX;
    }
```

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

    public function getPageSizeY(): int
    {
        return $this->PageSizeY;
    }

    public function setPageSizeY(int $PageSizeY): void
    {
        $this->PageSizeY = $PageSizeY;
    }

    public function getMarginTop(): int
    {
        return $this->marginTop;
    }

    public function setMarginTop(int $marginTop): void
    {
        $this->marginTop = $marginTop;
    }

    public function getMarginRight(): int
    {
        return $this->marginRight;
    }

    public function setMarginRight(int $marginRight): void
    {
        $this->marginRight = $marginRight;
    }

    public function getMarginBottom(): int
    {
        return $this->marginBottom;
    }

    public function setMarginBottom(int $marginBottom): void
    {
        $this->marginBottom = $marginBottom;
    }

    public function getMarginLeft(): int
    {
        return $this->marginLeft;
    }

    public function setMarginLeft(int $marginLeft): void
    {
        $this->marginLeft = $marginLeft;
    }

    public function getHasHeader(): bool
    {
        return $this->hasHeader;
    }

    public function setHasHeader(bool $hasHeader): void
    {
        $this->hasHeader = $hasHeader;
    }

    public function getHeaderHeigh(): int
    {
        return $this->headerHeigh;
    }

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

```
    public function getHasFooter(): bool
    {
        return $this->hasFooter;
    }

    public function setHasFooter(bool $hasFooter): void
    {
        $this->hasFooter = $hasFooter;
    }

    public function getFooterHeigh(): int
    {
        return $this->footerHeigh;
    }

    public function setFooterHeigh(int $footerHeigh): void
    {
        $this->footerHeigh = $footerHeigh;
    }

    public function getPageColor(): string
    {
        return $this->pageColor;
    }

    public function setPageColor(string $pageColor): void
    {
        $this->pageColor = $pageColor;
    }

    public function getEncode(): string
    {
        return $this->encode;
    }

    public function setEncode(string $encode): void
    {
        $this->encode = $encode;
    }

    public function __toString(): string
    {
        $saida = '';
        $saida .= 'pageOrientation: ' . $this->getPageOrientation() . '<br>';
        $saida .= 'unit: ' . $this->getUnit() . '<br>';
        $saida .= 'PageSizeX: ' . $this->getPageSizeX() . 'mm<br>';
        $saida .= 'PageSizeY: ' . $this->getPageSizey() . 'mm<br>';
        $saida .= 'maginTop: ' . $this->getMarginTop() . 'mm<br>';
        $saida .= 'maginRight: ' . $this->getMarginRight() . 'mm<br>';
        $saida .= 'maginBottom: ' . $this->getMarginBottom() . 'mm<br>';
        $saida .= 'maginLeft: ' . $this->getMarginLeft() . 'mm<br>';
        $saida .= $this->getHasHeader() ? 'hasHeader: Sim<br>' : 'hasHeader: Não<br>';
        $saida .= 'headerHeigh: ' . $this->getHeaderHeigh() . 'mm<br>';
        $saida .= $this->getHasFooter() ? 'hasFooter: Sim<br>' : 'hasFooter: Não<br>';
        $saida .= 'footerHeigh: ' . $this->getFooterHeigh() . 'mm<br>';
        $saida .= 'pageColor: ' . $this->getPageColor() . '<br>';
        $saida .= 'encode: ' . $this->getEncode() . '<br>';

        return $saida;
    }
}
```

O problema em questão não está na classe **GeradorPDF** mas na instanciação de seus objetos. Repare em quantos parâmetros são necessários para criar um objeto com a formatação de uma folha A4 (PDF A4) e outra A3 (PDF A3).

Atributos	A4	A3
pageOrientation	portrait	portrait
unit	mm	mm
PageSizeX	210	297
PageSizeY	297	420
marginTop	30	60
marginRight	20	40
marginBottom	30	60
marginLeft	20	40
hasHeader	true	false
headerHeigh	15	0
hasFooter	true	false
footerHeigh	15	0
pageColor	#ffffff	#ffffff
encode	UTF-8	UTF-8

Especificações do PDF A4 e A3

```php
echo '## Criação do objeto A4 ##<br>';

$pdf = new GeradorPDF(
    'portrait', //pageOrientation
    'mm', //unit
    210, //PageSizeX
    297, //PageSizeY
    30, //marginTop
    20, //marginRight
    30, //marginBottom
    20, //marginLeft
    true, //hasHeader
    15, //headerHeigh
    true, //hasFooter
    15, //footerHeigh
    '#ffffff', //pageColor
    'UTF-8' //encode
);

//Impressão do Objeto A4
echo $pdf;

echo '<br>## Criação do objeto A3 ##<br>';

$pdf = new GeradorPDF(
    'portrait', //pageOrientation
    'mm', //unit
    297, //PageSizeX
    420, //PageSizeY
    60, //marginTop
    40, //marginRight
    60, //marginBottom
    40, //marginLeft
    false, //hasHeader
    0, //headerHeigh
    false, //hasFooter
    0, //footerHeigh
    '#ffffff', //pageColor
    'UTF-8' //encode
);

//Impressão do Objeto A3
echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Sim
headerHeigh: 15mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8

## Criação do objeto A3 ##
pageOrientation: portrait
unit: mm
PageSizeX: 297mm
PageSizeY: 420mm
maginTop: 60mm
maginRight: 40mm
maginBottom: 60mm
maginLeft: 40mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Não
footerHeigh: 0mm
pageColor: #ffffff
encode: UTF-8
```

Veja só quanta informação foi necessária para a criação de 2 objetos da classe **GeradorPDF.** O **Cliente** (quem cria objetos da classe **GeradorPDF**) é o responsável por conhecer os valores necessários para configurar a criação dos objetos, além disso, o **Cliente** ainda pode ter mais responsabilidades o que violaria o princípio da responsabilidade única.

Como alternativa podemos remover o método construtor da classe **GeradorPDF,** assim, depois de criar o objeto poderíamos utilizar os métodos *Setters* para configurá-lo.

```php
class GeradorPDF
{
    private string $pageOrientation = 'portrait';
    private string $unit = 'mm';
    private int $PageSizeX = 210;
    private int $PageSizeY = 297;
    private int $marginTop = 30;
    private int $marginRight = 20;
    private int $marginBottom = 30;
    private int $marginLeft = 20;
    private bool $hasHeader = false;
    private int $headerHeigh = 0;
    private bool $hasFooter = false;
    private int $footerHeigh = 0;
    private string $pageColor = '#ffffff';
    private string $encode = 'UTF-8';

    //### Aqui ficava o construtor - Foi Removido ###;

    //Nada muda nos demais métodos Getters, Setters e __toString.
}
```

Agora a criação dos mesmos dois objetos ficaria da seguinte forma:

```php
echo '## Criação do objeto A4 ##<br>';
$pdf = new GeradorPDF();

$pdf->setPageOrientation('portrait');
$pdf->setUnit('mm');
$pdf->setPageSizeX(210);
$pdf->setPageSizeY(297);
$pdf->setMarginTop(30);
$pdf->setMarginRight(20);
$pdf->setMarginBottom(30);
$pdf->setMarginLeft(20);
$pdf->setHasHeader(true);
$pdf->setHeaderHeigh(15);
$pdf->setHasFooter(true);
$pdf->setFooterHeigh(15);
$pdf->setPageColor('#ffffff');
$pdf->setEncode('UTF-8');

echo $pdf;

echo '<br>## Criação do objeto A3 ##<br>';
$pdf = new GeradorPDF();

$pdf->setPageOrientation('portrait');
$pdf->setUnit('mm');
$pdf->setPageSizeX(297);
$pdf->setPageSizeY(420);
$pdf->setMarginTop(60);
$pdf->setMarginRight(40);
$pdf->setMarginBottom(60);
$pdf->setMarginLeft(40);
$pdf->setHasHeader(false);
$pdf->setHeaderHeigh(0);
$pdf->setHasFooter(false);
$pdf->setFooterHeigh(0);
$pdf->setPageColor('#ffffff');
$pdf->setEncode('UTF-8');

echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Sim
headerHeigh: 15mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8

## Criação do objeto A3 ##
pageOrientation: portrait
unit: mm
PageSizeX: 297mm
PageSizeY: 420mm
maginTop: 60mm
maginRight: 40mm
maginBottom: 60mm
maginLeft: 40mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Não
footerHeigh: 0mm
pageColor: #ffffff
encode: UTF-8
```

Estamos criando dois objetos diferentes a partir da mesma classe. Dois objetos que sempre serão configurados da mesma forma. A definição diz que *"O padrão Builder separa a construção de um objeto complexo de sua representação"*, então vamos criar duas classes que têm como responsabilidade apenas a construção de objetos da classe **GeradorPDF.** Continuando com o método construtor fora da classe **GeradorPDF** iremos criar as classes: **BuilderA4** e **BuilderA3** que irão criar objetos a partir da especificação de A4 e A3 respectivamente.

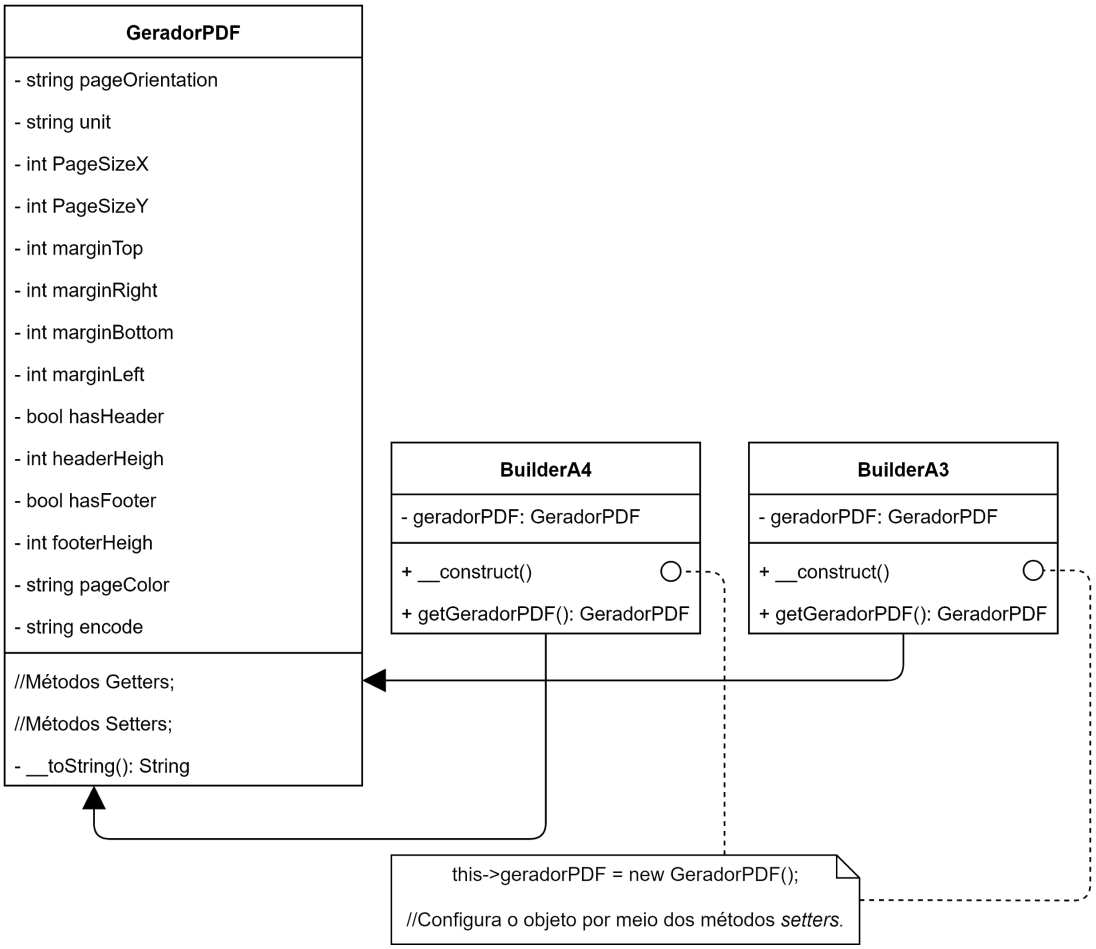


Diagrama de classes utilizando *Builders*

```php
class BuilderA4
{
    private GeradorPDF $geradorPDF;

    //Cria o objeto GeradorPDF e o configura conforme as especificações de A4.
    public function __construct()
    {
        $this->geradorPDF = new GeradorPDF();
        $this->geradorPDF->setPageOrientation('portrait');
        $this->geradorPDF->setUnit('mm');
        $this->geradorPDF->setPageSizeX(210);
        $this->geradorPDF->setPageSizeY(297);
        $this->geradorPDF->setMarginTop(30);
        $this->geradorPDF->setMarginRight(20);
        $this->geradorPDF->setMarginBottom(30);
        $this->geradorPDF->setMarginLeft(20);
        $this->geradorPDF->setHasHeader(true);
        $this->geradorPDF->setHeaderHeigh(15);
        $this->geradorPDF->setHasFooter(true);
        $this->geradorPDF->setFooterHeigh(15);
        $this->geradorPDF->setPageColor('#ffffff');
        $this->geradorPDF->setEncode('UTF-8');
    }

    //Retorna o objeto GeradorPDF já configurado conforme as especificações de A4;
    public function getGeradorPDF(): GeradorPDF
    {
        return $this->geradorPDF;
    }
}
```

```
class BuilderA3
{
    private GeradorPDF $geradorPDF;

    //Cria o objeto GeradorPDF e o configura conforme as especificações de A3.
    public function __construct()
    {
        $this->geradorPDF = new GeradorPDF();
        $this->geradorPDF->setPageOrientation('portrait');
        $this->geradorPDF->setUnit('mm');
        $this->geradorPDF->setPageSizeX(297);
        $this->geradorPDF->setPageSizeY(420);
        $this->geradorPDF->setMarginTop(60);
        $this->geradorPDF->setMarginRight(40);
        $this->geradorPDF->setMarginBottom(60);
        $this->geradorPDF->setMarginLeft(40);
        $this->geradorPDF->setHasHeader(false);
        $this->geradorPDF->setHeaderHeigh(0);
        $this->geradorPDF->setHasFooter(false);
        $this->geradorPDF->setFooterHeigh(0);
        $this->geradorPDF->setPageColor('#ffffff');
        $this->geradorPDF->setEncode('UTF-8');
    }

    //Retorna o objeto GeradorPDF já configurado conforme as especificações de A3;
    public function getGeradorPDF(): GeradorPDF
    {
        return $this->geradorPDF;
    }
}
```

Graças aos *Builders* criados a responsabilidade de criação e configuração dos objetos saí do **Cliente**.

```
echo '## Criação do objeto A4 ##<br>';

//Criação do Builder de A4;
$builder = new BuilderA4();

//Recuperação do objeto pronto, já configurado;
$pdf = $builder->getGeradorPDF();

//Impressão do objeto;
echo $pdf;

echo '<br>## Criação do objeto A3 ##<br>';

//Criação do Builder de A3;
$builder = new BuilderA3();

//Recuperação do objeto pronto, já configurado;
$pdf = $builder->getGeradorPDF();

//Impressão do objeto;
echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Sim
headerHeigh: 15mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8

## Criação do objeto A3 ##
pageOrientation: portrait
unit: mm
PageSizeX: 297mm
PageSizeY: 420mm
maginTop: 60mm
maginRight: 40mm
maginBottom: 60mm
maginLeft: 40mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Não
footerHeigh: 0mm
pageColor: #ffffff
encode: UTF-8
```

Devido aos *Builders* o código do **Cliente** ficou mais simples. As classes **BuilderA4** e **BuilderA3** controlam o processo de criação de determinados objetos da classe de acordo com suas especificações.

Nosso código já está melhor, porém ele não é flexível. Como poderíamos criar um objeto com as configurações de A4 mas sem o cabeçalho? Pois é, ainda não é possível fazer isso utilizando os *Builder* da forma que foram implementados, já que uma vez criados fazem toda a configuração do objeto **GeradorPDF** em seu construtor.

Outra parte da definição do padrão *Builder* diz que "*um mesmo processo de construção de um objeto pode criar diferentes representações dele*", em outras palavras, isso nos diz que um mesmo processo de criação pode criar objetos diferentes. Vamos explorar isso.

Podemos dividir o processo de criação dos objetos em partes, elas podem ou não ter uma ordem definida. Algumas partes podem ser facultativas e outras obrigatórias, isso varia conforme a necessidade do contexto do seu código. No nosso caso podemos quebrar as classe **BuilderA4** e **BuilderA3** nos seguintes métodos:

- **getGeradorPDF()**: Retorna o objeto já configurado.
- **setPageConfiguration()**: Define as configurações básicas das páginas do PDF.
- **setMargin()**: Configura as margens das páginas do PDF.
- **setHeader()**: Configura o cabeçalho das páginas do PDF.
- **setFooter()**: Configura o rodapé das páginas do PDF.

Para garantir que todos os *Builders* tenham tais métodos, vamos criar a interface **BuilderInterfaceGeradorPDF** a qual todos eles deverão implementar. Essa interface dita o processo de criação que todos os *Builders* devem seguir. Cada *Builder* poderá criar objetos diferentes, porém, seguindo processo de criação que é ditado pela interface.

```php
interface BuilderInterfaceGeradorPDF
{
    public function getGeradorPDF(): GeradorPDF;
    public function setPageConfiguration(): void;
    public function setMargim(): void;
    public function setHeader(): void;
    public function setFooter(): void;
}
```

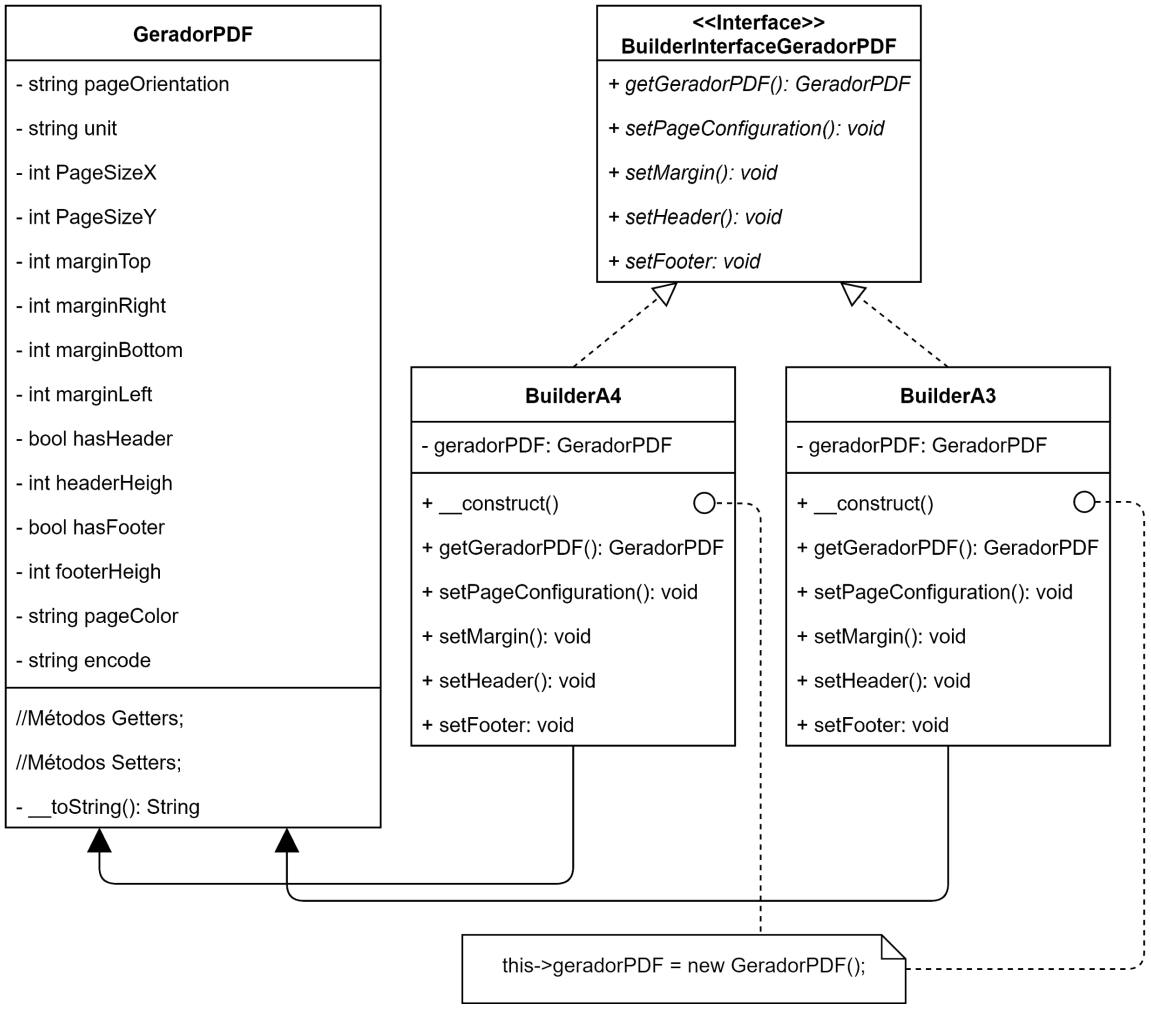


Diagrama de classes utilizando *Builders* divididos em passos de criação

Hora de refatorar os Builder:

```
class BuilderA4 implements BuilderInterfaceGeradorPDF
{
    private GeradorPDF $geradorPDF;

    //Agora o construtor cria o objeto mas não o configura.
    public function __construct()
    {
        $this->geradorPDF = new GeradorPDF();
    }

    //Retorna o Objeto.
    public function getGeradorPDF(): GeradorPDF
    {
        return $this->geradorPDF;
    }

    //Define as configurações das páginas conforme A4;
    public function setPageConfiguration(): void
    {
        $this->geradorPDF->setPageOrientation('portrait');
        $this->geradorPDF->setUnit('mm');
        $this->geradorPDF->setPageSizeX(210);
        $this->geradorPDF->setPageSizeY(297);
        $this->geradorPDF->setPageColor('#ffffff');
        $this->geradorPDF->setEncode('UTF-8');
    }

    //Configura as margens das páginas conforme A4;
    public function setMargim(): void
    {
        $this->geradorPDF->setMarginTop(30);
        $this->geradorPDF->setMarginRight(20);
        $this->geradorPDF->setMarginBottom(30);
        $this->geradorPDF->setMarginLeft(20);
    }

    //Configura o cabecalho das páginas conforme A4;
    public function setHeader(): void
    {
        $this->geradorPDF->setHasHeader(true);
        $this->geradorPDF->setHeaderHeigh(15);
    }

    //Configura o rodapé das páginas conforme A4;
    public function setFooter(): void
    {
        $this->geradorPDF->setHasFooter(true);
        $this->geradorPDF->setFooterHeigh(15);
    }
}
```

```php
class BuilderA3 implements BuilderInterfaceGeradorPDF
{
    private GeradorPDF $geradorPDF;

    //Agora o construtor cria o objeto mas não o configura.
    public function __construct()
    {
        $this->geradorPDF = new GeradorPDF();
    }

    //Retorna o Objeto.
    public function getGeradorPDF(): GeradorPDF
    {
        return $this->geradorPDF;
    }

    //Define as configurações das páginas conforme A3;
    public function setPageConfiguration(): void
    {
        $this->geradorPDF->setPageOrientation('portrait');
        $this->geradorPDF->setUnit('mm');
        $this->geradorPDF->setPageSizeX(297);
        $this->geradorPDF->setPageSizeY(420);
        $this->geradorPDF->setPageColor('#ffffff');
        $this->geradorPDF->setEncode('UTF-8');
    }

    //Configura as margens das páginas conforme A3;
    public function setMargim(): void
    {
        $this->geradorPDF->setMarginTop(60);
        $this->geradorPDF->setMarginRight(40);
        $this->geradorPDF->setMarginBottom(60);
        $this->geradorPDF->setMarginLeft(40);
    }

    //Configura o cabecalho das páginas conforme A3;
    public function setHeader(): void
    {
        $this->geradorPDF->setHasHeader(false);
        $this->geradorPDF->setHeaderHeigh(0);
    }

    //Configura o rodapé das páginas conforme A3;
    public function setFooter(): void
    {
        $this->geradorPDF->setHasFooter(false);
        $this->geradorPDF->setFooterHeigh(0);
    }
}
```

Vamos ver como fica o código do **Cliente** utilizando a nova versão dos *Builders*.

```
echo '## Criação do objeto A4 ##<br>';
$builder = new BuilderA4();

//Os métodos de configuração são chamados.
$builder->setPageConfiguration();
$builder->setMargim();
$builder->setHeader();
$builder->setFooter();
$pdf = $builder->getGeradorPDF();

echo $pdf;

echo '<br>## Criação do objeto A3 ##<br>';
$builder = new BuilderA3();

//Os métodos de configuração são chamados.
$builder->setPageConfiguration();
$builder->setMargim();
$builder->setHeader();
$builder->setFooter();
$pdf = $builder->getGeradorPDF();

echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Sim
headerHeigh: 15mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8

## Criação do objeto A3 ##
pageOrientation: portrait
unit: mm
PageSizeX: 297mm
PageSizeY: 420mm
maginTop: 60mm
maginRight: 40mm
maginBottom: 60mm
maginLeft: 40mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Não
footerHeigh: 0mm
pageColor: #ffffff
encode: UTF-8
```

Agora os *Builder* são mais flexíveis, e podemos criar um PDF A4 sem cabeçalho apenas não chamando o método **setHeader()**.

```
echo '## Criação do objeto A4 ##<br>';
$builder = new BuilderA4();

//Os métodos de configuração são chamados.
$builder->setPageConfiguration();
$builder->setMargim();
$builder->setFooter();
$pdf = $builder->getGeradorPDF();

echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8
```

Veja que na saída o valor *hasHeader* é "Não", ele assumiu o valor padrão definido na declaração do atributo **hasHeader** na classe **GeradorPDF**. Como afirma a definição, nós conseguimos criar outra representação de um objeto de **GeradorPDF**, agora uma representação sem o cabeçalho, utilizando o mesmo *Builder*, ou seja, o mesmo processo de construção.

É importante dizer que *Builders* distintos não necessariamente precisam criar objetos de uma mesma classe mudando apenas seus atributos. Eles podem criar objetos de classes diferentes e tais classes nem precisam pertencer a mesma hierarquia de classes. O único requisito é que o processo de criação seja o mesmo.

Embora nosso código esteja mais flexível, o **Cliente** voltou a ter responsabilidades a respeito da criação dos objetos, com nossa versão atual do código o **Cliente** precisa chamar os métodos dos *Builders* para que os objetos sejam criados corretamente.

Para resolver esse problema o padrão *Builder* sugere a criação de um diretor. Trata-se de uma classe que controla quais métodos dos *Builders* serão chamados em cada contexto, e também em qual ordem tais métodos serão chamados. Vamos criar a classe **GeradorPDFDirector**.

Na classe **GeradorPDFDirector** podemos deixar pré-configuradas diversas representações de um objeto que será criado por um *Builder*. Na classe acima deixamos pré-configuradas:

- A criação de um objeto totalmente configurado.
- A criação de um objeto sem a configuração cabeçalho.
- A criação de um objeto sem as configurações de cabeçalho e de rodapé.

```
class GeradorPDFDirector
{
    private BuilderInterfaceGeradorPDF $BuilderGeradorPDF;

    //Guarda referência a um objeto do tipo BuilderInterfaceGeradorPDF.
    public function __construct(BuilderInterfaceGeradorPDF $BuilderGeradorPDF)
    {
        $this->BuilderGeradorPDF = $BuilderGeradorPDF;
    }

    //Muda o Builder em tempo de execução.
    public function setBuilderGeradorPDF(BuilderInterfaceGeradorPDF $BuilderGeradorPDF): void
    {
        $this->BuilderGeradorPDF = $BuilderGeradorPDF;
    }

    //Configura completamente o objeto criado pelo Builder recebido no construtor.
    public function criarGeradorPDF()
    {
        $this->BuilderGeradorPDF->setPageConfiguration();
        $this->BuilderGeradorPDF->setMargim();
        $this->BuilderGeradorPDF->setHeader();
        $this->BuilderGeradorPDF->setFooter();
    }


    //Configura o objeto criado pelo Builde recebido no construtor sem cabeçalho.
    public function criarGeradorPDFNoHeader(): void
    {
        $this->BuilderGeradorPDF->setPageConfiguration();
        $this->BuilderGeradorPDF->setMargim();
        $this->BuilderGeradorPDF->setFooter();
    }


    //Configura o objeto criado pelo Builde recebido no construtor sem cabeçalho e rodapé.
    public function criarGeradorPDFNoHeaderNoFooter(): void
    {
        $this->BuilderGeradorPDF->setPageConfiguration();
        $this->BuilderGeradorPDF->setMargim();
    }
}
```

Vejamos como fica o **Cliente:**

```
echo '## Criação do objeto A4 ##<br>';

//Criação do Builder de A4.
$builder = new BuilderA4();

//Criação do Diretor passando para ele o Builder de A4.
$director = new GeradorPDFDirector($builder);

//Criação de um objeto totalmente configurado.
$director->criarGeradorPDF();

//Recuperação do Objeto.
$pdf = $builder->getGeradorPDF();

//Impressão do Objeto.
echo $pdf;

echo '<br>## Criação do objeto A3 ##<br>';

//Criação do Builder de A3.
$builder = new BuilderA3();

//ATUALIZAÇÃO do diretor passando para ele o Builder de A3. O diretor não foi criado novamente.
$director->setBuilderGeradorPDF($builder);

//Criação de um objeto totalmente configurado.
$director->criarGeradorPDF();

//Recuperação do Objeto.
$pdf = $builder->getGeradorPDF();

//Impressão do Objeto.
echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Sim
headerHeigh: 15mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8
```

```
## Criação do objeto A3 ##
pageOrientation: portrait
unit: mm
PageSizeX: 297mm
PageSizeY: 420mm
maginTop: 60mm
maginRight: 40mm
maginBottom: 60mm
maginLeft: 40mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Não
footerHeigh: 0mm
pageColor: #ffff
```

O **Cliente** delega a chamada dos métodos do Builder para o diretor (**GeradorPDFDirector**). Seguindo a mesma lógica também podemos criar um objeto configurado conforme as especificações de A4, porém sem cabeçalho, para isso basta trocar a chamada do método **getGeradorPDF()** pelo método **criarGeradorPDFNoHeader()**.

```php
echo '## Criação do objeto A4 ##<br>';

$builder = new BuilderA4();
$director = new GeradorPDFDirector($builder);

//Criação de um objeto sem o Header.
$director->criarGeradorPDFNoHeader();

$pdf = $builder->getGeradorPDF();
echo $pdf;
```

Saída:

```
## Criação do objeto A4 ##
pageOrientation: portrait
unit: mm
PageSizeX: 210mm
PageSizeY: 297mm
maginTop: 30mm
maginRight: 20mm
maginBottom: 30mm
maginLeft: 20mm
hasHeader: Não
headerHeigh: 0mm
hasFooter: Sim
footerHeigh: 15mm
pageColor: #ffffff
encode: UTF-8
```

O mesmo processo poderia ser feito para o *Builder* de A3 (**BuilderA3**). Porém não causaria diferenças na saída, uma vez que na classe **BuilderA3** o cabeçalho é definido como **false** e com altura de 0mm, da mesma forma que é definido como o valor padrão nas classe **GeradorPDF**.

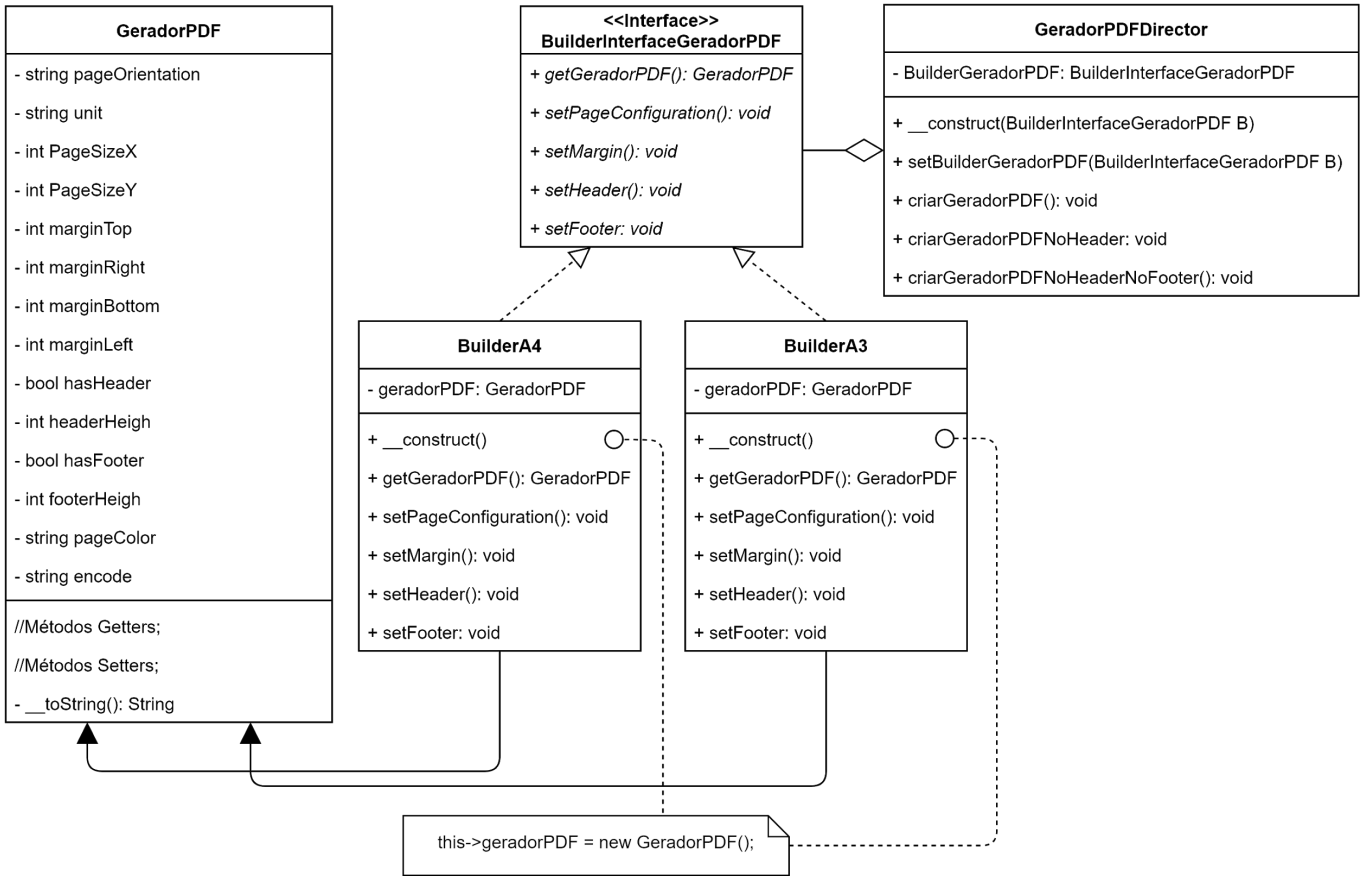


Diagrama de classes utilizando o padrão *Builder* completo

Aplicabilidade (Quando utilizar?)

- Quando o algoritmo que cria um objeto complexo deve ser independente das partes que o compõem e de como tais partes são montadas.
- Quando o processo de construção deve permitir representações diferentes para o objeto que é construído.

Componentes

- **Builder:** Especifica uma interface abstrata para criação de partes de um objeto (produto), tal interface deverá ser seguida por todos os BuildersConcretos.
- **BuilderConcreto:**
  - Constrói e monta partes do Produto utilizando a implementação concreta da interface Builder a qual implementa.
  - Define e mantém a representação (Produto) que cria.
  - Fornece uma interface (método) para recuperação do produto criado.

- **Diretor:** Constrói um objeto conforme suas necessidades. Para isso ele utiliza a interface de Builder.

- **Produto:**

    - Representa o objeto complexo em construção. BuilderConcreto constrói a representação interna do produto e define o processo pelo qual ele é criado.

    - Produtos são os objetos resultantes. Produtos construídos por diferentes BuildersConcretos não precisam pertencer a mesma interface ou hierarquia de classes.

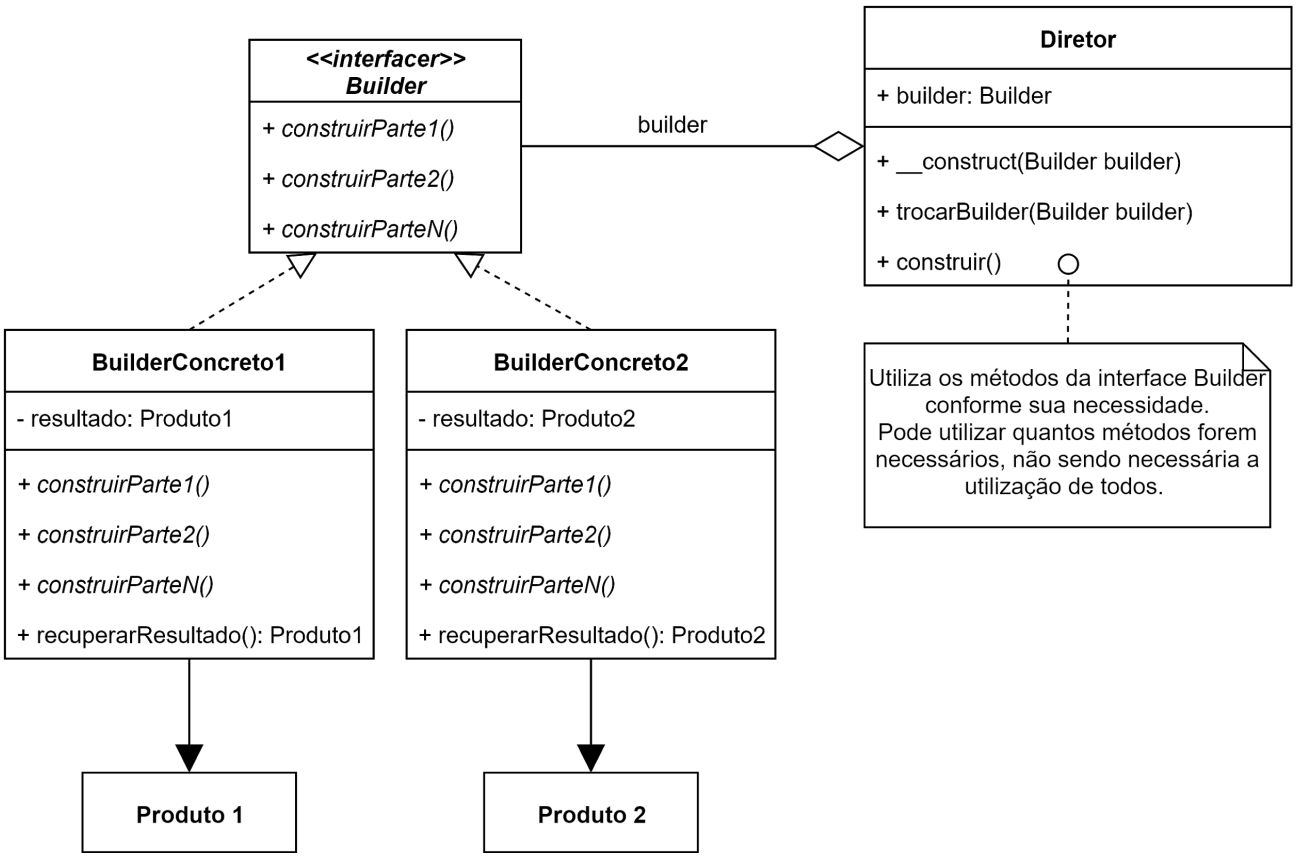


Diagrama de Classes

Consequências

- Permite variar a representação interna de um produto. O objeto *Builder* fornece ao Diretor uma interface abstrata para construir o produto. Tal interface permite que o *Builder* oculte a representação e a estrutura interna do produto, e também oculta como o produto é montado. O produto é construído por meio de uma interface abstrata, então, para alterar a representação interna do produto basta trocar o Builder do Diretor.

- Isola o código de construção e representação. O padrão *Builder* aprimora a modularidade, encapsulando a maneira a qual um objeto complexo é construído e representado. Os clientes não precisam ter conhecimento sobre como as classes definem a estrutura interna do produto. As classes de produto ficam encapsuladas nos *Builders* concretos e não aparecem na interface *Builder*. O código de criação dos produtos são escritos apenas uma vez, assim diferentes diretores podem compartilhar o mesmo código de criação.

- Oferece um controle mais fino sobre o processo de construção. Enquanto padrões criacionais constroem produtos de uma só vez, o padrão *Builder* constrói o Produto passo a passo, sob o controle do Diretor. Somente quando o Produto é concluído, o Diretor o recupera do BuilderConcreto. Portanto, a interface de *Builder* reflete o processo de construção do produto mais do que outros padrões criacionais. Isso permite um controle mais preciso sobre o processo de criação e, consequentemente, a estrutura interna do produto resultante.