1) Não é padrão de projeto! É basicamente uma técnica de programação para introduzirmos os padrões Factory´s, ele encapsula a criação de objetos, permite interfaces para criar objetos sem expor a criação lógica ao cliente.

2) Ele encapsula a criação de objetos, mas neste padrão a criação de objetos e as subclasses que decidem quais objetos criarem, em resumo, ele definem uma inteface para criação de objetos e as classes decidem quais as suas instâncias.

3) Quando queremos encapsular o código que cria objetos, e quando queremos resolver o embargo de dos casos de classes que instanciam classes concretas, e podemos encapsular esse comportamento de instanciação, quando basicamente buscamos ter nosso código de criação em um único objeto, um único local para manutenção e em suma, ele é muito utilizado em frameworks.

4) Exemplo de terminolgia (UML) e a sua implementação:

Implementação do simples Factory, um exemplo de implementação na linguagem PHP, uma fábrica de carros e suas devidas :

```
<?php

class carFactory {

public function __construct() {
    // ... //
    }

    public static function build ($type = '') {

        if($type == '') {
            throw new Exception('Invalid Car Type.');
        } else {

            $className = 'car_'.ucfirst($type);

            // Assuming Class files are already loaded using
autoload concept
            if(class_exists($className)) {
                return new $className();
            } else {
                throw new Exception('Car type not found.');
            }
        }
    }
}
```

Temos na classe de cima, um método estático que é o responsável pela criação do objeto do tipo que passamos.

```
<?php

class car_Sedan {

    public function __construct() {
        echo "Creating Sedan";
    }
```

```
}

class car_Suv {

    public function __construct() {
        echo "Creating SUV";
    }

}
```

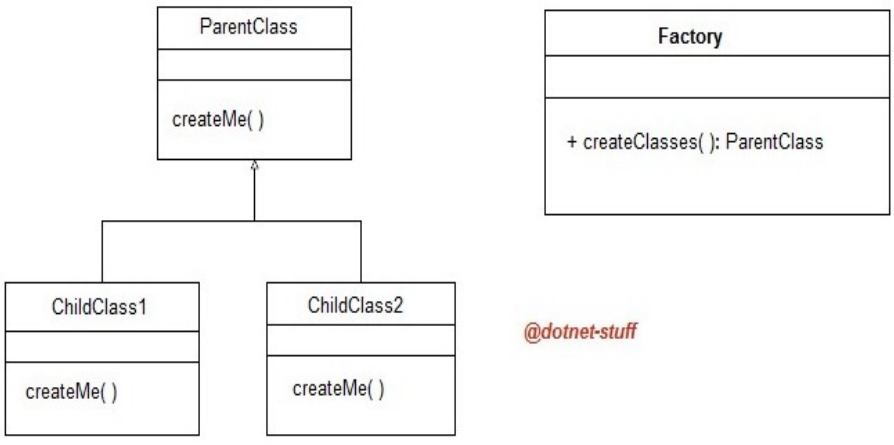
Neste momento, nós temos as nossa classes Factory(fábrica) e concrete(concreta) prontas para ser usadas, vamos praticar para criar os tipos de carros necessários.

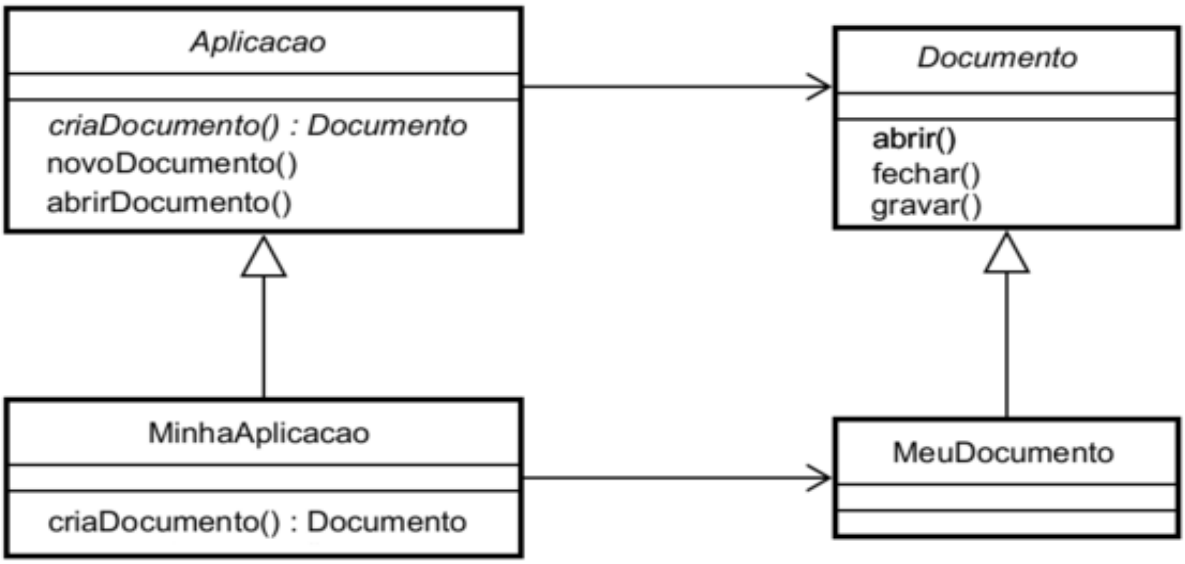
```
// Creating new Sedan
$sedan = carFactory::build('sedan');

// Creating new SUV
$suv = carFactory::build('suv');
```

A adição de uma classe é bem simples, só criar a classe concreta e está pronto, um exemplo abaixo:

```
<?php
class car_Luxury {

public function __construct() {
        echo "Creating Luxury";
    }

}

// Creating new Luxury
$luxury = carFactory::build('luxury');
```

Um exemplo de terminologia (UML):






```
public abstract class Aplicacao
{
    private Documento doc;

    Documento criaDocumento();

    void novoDocumento()
    {
        this.doc = this.criaDocumento();
    }


    void abrirDocumento()
    {
        this.doc.abrir();
    }

}

public abstract class Documento
{
    void abrir()
    {
        Console.WriteLine("Documento:Abrir documento!");
    }

 void fechar()
    {
        Console.WriteLine("Documento:Fechar documento!");
    }

    void gravar()
    {
        Console.WriteLine("Documento:Gravar documento!");
    }
}

public class MinhaAplicacao : Aplicacao
{
    public Documento criaDocumento()
```

```
    {
        return new MeuDocumento();
    }
}

public class MeuDocumento : Documento
{

}
```

Acima, vimos um exemplo de implementação do Facthory Method, onde fizemos uma aplicação de de criação de documentos

Exemplo de terminologia (UML) do padrão Facthory Method: