https://sourcemaking.com/design_patterns/abstract_factory

Patrón de diseño abstracto de fábrica

Intención

- Proporcione una interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases concretas.
- Una jerarquía que encapsula: muchas "plataformas" posibles y la construcción de un conjunto de "productos".
- El `new` operador considerado perjudicial.

Problema

Si una aplicación debe ser portátil, debe encapsular las dependencias de la plataforma. Estas "plataformas" pueden incluir: sistema de ventanas, sistema operativo, base de datos, etc. Con demasiada frecuencia, esta encapsulación no está diseñada de antemano, y muchas declaraciones de casos `#ifdef` con opciones para todas las plataformas compatibles actualmente comienzan a procrear como conejos en todo el código .

Discusión

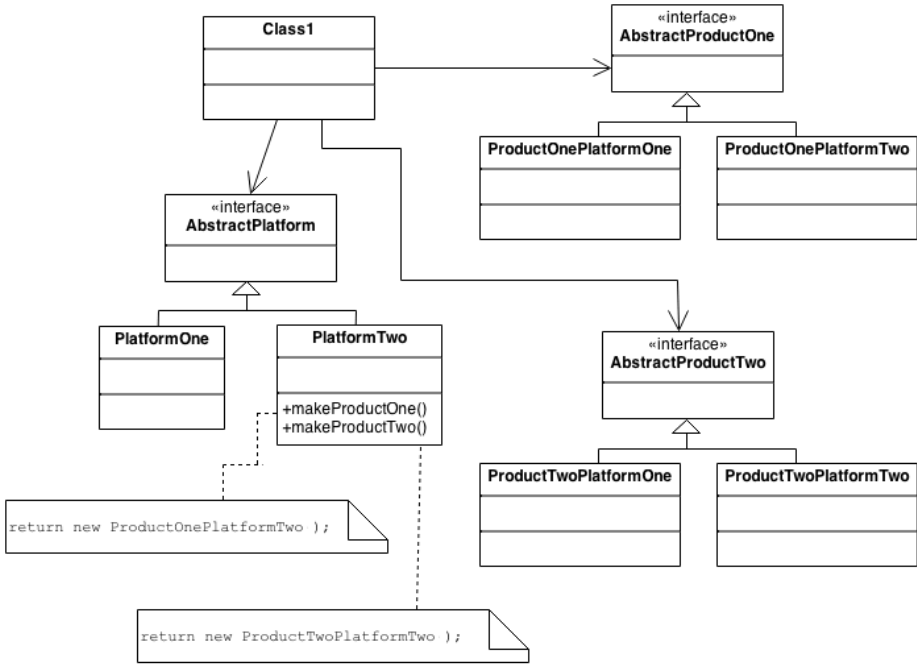
Proporcione un nivel de indirección que abstraiga la creación de familias de objetos relacionados o dependientes sin especificar directamente sus clases concretas. El objeto "fábrica" tiene la responsabilidad de proporcionar servicios de creación para toda la familia de plataformas. Los clientes nunca crean objetos de plataforma directamente, le piden a la fábrica que lo haga por ellos.

Este mecanismo facilita el intercambio de familias de productos porque la clase específica del objeto de fábrica aparece solo una vez en la aplicación, donde se instancia. La aplicación puede reemplazar al por mayor a toda la familia de productos simplemente instanciando una instancia concreta diferente de la fábrica abstracta.

Debido a que el servicio proporcionado por el objeto de fábrica es tan generalizado, se implementa rutinariamente como Singleton.

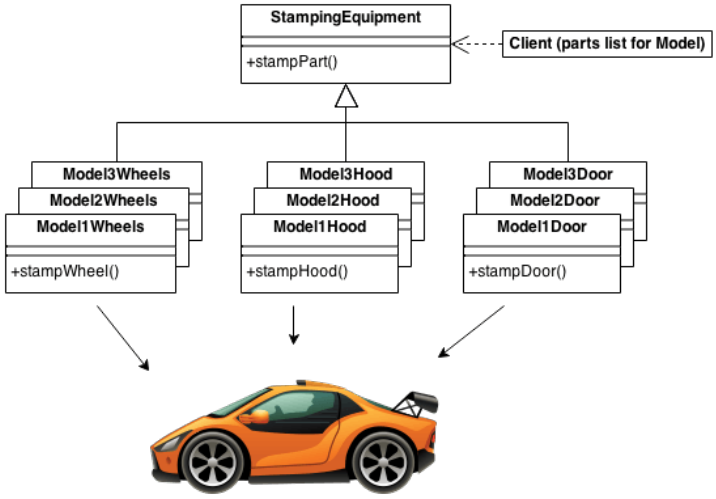
Estructura

Abstract Factory define un método de fábrica por producto. Cada Método de Fábrica encapsula al `new` operador y las clases concretas de productos específicos de la plataforma. Cada "plataforma" se modela con una clase derivada de Factory.



Ejemplo

El propósito de Abstract Factory es proporcionar una interfaz para crear familias de objetos relacionados, sin especificar clases concretas. Este patrón se encuentra en el equipo de estampado de chapa utilizado en la fabricación de automóviles japoneses. El equipo de estampado es una Fábrica abstracta que crea piezas de carrocería. La misma maquinaria se utiliza para estampar puertas a la derecha, puertas a la izquierda, guardabarros delanteros derechos, guardabarros delanteros izquierdos, capós, etc. para diferentes modelos de automóviles. Mediante el uso de rodillos para cambiar los troqueles de estampado, las clases de concreto producidas por la maquinaria se pueden cambiar en tres minutos.



Lista de Verificación

1. Decida si la "independencia de la plataforma" y los servicios de creación son la fuente actual de dolor.
2. Planifique una matriz de "plataformas" versus "productos".
3. Defina una interfaz de fábrica que consista en un método de fábrica por producto.
4. Defina una clase derivada de fábrica para cada plataforma que encapsule todas las referencias al `new` operador.
5. El cliente debe retirar todas las referencias a `new` y utilizar los métodos de fábrica para crear los objetos del producto.

Reglas de juego

- A veces, los patrones de creación son competidores: hay casos en los que Prototype o Abstract Factory podrían usarse de manera rentable. En otros momentos son complementarios: Abstract Factory puede almacenar un conjunto de prototipos para clonar y devolver objetos de producto, Builder puede usar uno de los otros patrones para implementar qué componentes se construyen. Abstract Factory, Builder y Prototype pueden usar Singleton en su implementación.
- Abstract Factory, Builder y Prototype definen un objeto de fábrica responsable de conocer y crear la clase de objetos de producto, y lo convierten en un parámetro del sistema. Abstract Factory tiene el objeto de fábrica que produce objetos de varias clases. Builder hace que el objeto de fábrica cree un producto complejo de forma incremental utilizando un protocolo correspondientemente complejo. Prototipo tiene el objeto de fábrica (también conocido como prototipo) que construye un producto copiando un objeto prototipo.
- Las clases de Fábrica abstracta a menudo se implementan con Métodos de fábrica, pero también se pueden implementar usando Prototype.
- Abstract Factory se puede utilizar como una alternativa a Facade para ocultar clases específicas de la plataforma.
- Builder se enfoca en construir un objeto complejo paso a paso. Abstract Factory enfatiza una familia de objetos de producto (ya sea simple o complejo). Builder devuelve el producto como un paso final, pero en lo que respecta a Abstract Factory, el producto se devuelve de inmediato.
- A menudo, los diseños comienzan usando el Método Factory (menos complicado, más personalizable, proliferan las subclases) y evolucionan hacia Abstract Factory, Prototype o Builder (más flexible, más complejo) a medida que el diseñador descubre dónde se necesita más flexibilidad.

https://sourcemaking.com/design_patterns/abstract_factory/php/2

in *PHP Patrón de diseño abstracto de fábrica en PHP*

En el Patrón de Fábrica Abstracta, una fábrica abstracta define qué objetos necesitará la fábrica no abstracta o concreta para poder crear.

La fábrica de concreto debe crear los objetos correctos para su contexto, asegurando que todos los objetos creados por la fábrica de concreto hayan sido elegidos para poder funcionar correctamente para una circunstancia dada.

En este ejemplo, tenemos una fábrica abstracta, `AbstractBookFactory` , que especifica dos clases, `AbstractPHPBook` y `AbstractMySQLBook` , que deberán ser creadas por la fábrica concreta.

La clase concreta `OReillyBookfactory` extiende `AbstractBookFactory` y puede crear las clases `OReillyMySQLBook` y `OReillyPHPBook` , que son las clases correctas para el contexto de `OReilly` .

```
<?php

/**
 * BookFactory classes
 */
abstract class AbstractBookFactory {
    abstract function makePHPBook();
    abstract function makeMySQLBook();
}

class OReillyBookFactory extends AbstractBookFactory {
    private $context = "OReilly";
    function makePHPBook() {
        return new OReillyPHPBook;
    }
    function makeMySQLBook() {
        return new OReillyMySQLBook;
    }
}

class SamsBookFactory extends AbstractBookFactory {
    private $context = "Sams";
    function makePHPBook() {
        return new SamsPHPBook;
    }
    function makeMySQLBook() {
        return new SamsMySQLBook;
    }
}

/**
 * Book classes
 */
abstract class AbstractBook {
    abstract function getAuthor();
    abstract function getTitle();
```

```
}

abstract class AbstractMySQLBook extends AbstractBook {
    protected $subject = "MySQL";
}

class OReillyMySQLBook extends AbstractMySQLBook {
    private $author;
    private $title;
    function __construct() {
        $this->author = 'George Reese, Randy Jay Yarger, and Tim King';
        $this->title = 'Managing and Using MySQL';
    }
    function getAuthor() {
        return $this->author;
    }
    function getTitle() {
        return $this->title;
    }
}

class SamsMySQLBook extends AbstractMySQLBook {
    private $author;
    private $title;
    function __construct() {
        $this->author = 'Paul Dubois';
        $this->title = 'MySQL, 3rd Edition';
    }
    function getAuthor() {
        return $this->author;
    }
    function getTitle() {
        return $this->title;
    }
}

abstract class AbstractPHPBook extends AbstractBook {
    protected $subject = "PHP";
}

class OReillyPHPBook extends AbstractPHPBook {
    private $author;
    private $title;
    private static $oddOrEven = 'odd';
    function __construct()
    {
        //alternate between 2 books
        if ('odd' == self::$oddOrEven) {
            $this->author = 'Rasmus Lerdorf and Kevin Tatroe';
            $this->title = 'Programming PHP';
            self::$oddOrEven = 'even';
        }
        else {
            $this->author = 'David Sklar and Adam Trachtenberg';
            $this->title = 'PHP Cookbook';
            self::$oddOrEven = 'odd';
        }
    }
    function getAuthor() {
        return $this->author;
    }
    function getTitle() {
        return $this->title;
    }
```

```
}

class SamsPHPBook extends AbstractPHPBook {
    private $author;
    private $title;
    function __construct() {
        //alternate randomly between 2 books
        mt_srand((double)microtime() * 10000000);
        $rand_num = mt_rand(0, 1);

        if (1 > $rand_num) {
            $this->author = 'George Schlossnagle';
            $this->title = 'Advanced PHP Programming';
        }
        else {
            $this->author = 'Christian Wenz';
            $this->title = 'PHP Phrasebook';
        }
    }
    function getAuthor() {
        return $this->author;
    }
    function getTitle() {
        return $this->title;
    }
}

/**
 * Initialization
 */

  writeln('BEGIN TESTING ABSTRACT FACTORY PATTERN');
  writeln('');

  writeln('testing OReillyBookFactory');
  $bookFactoryInstance = new OReillyBookFactory;
  testConcreteFactory($bookFactoryInstance);
  writeln('');

  writeln('testing SamsBookFactory');
  $bookFactoryInstance = new SamsBookFactory;
  testConcreteFactory($bookFactoryInstance);

  writeln("END TESTING ABSTRACT FACTORY PATTERN");
  writeln('');

  function testConcreteFactory($bookFactoryInstance)
  {
      $phpBookOne = $bookFactoryInstance->makePHPBook();
      writeln('first php Author: '.$phpBookOne->getAuthor());
      writeln('first php Title: '.$phpBookOne->getTitle());

      $phpBookTwo = $bookFactoryInstance->makePHPBook();
      writeln('second php Author: '.$phpBookTwo->getAuthor());
      writeln('second php Title: '.$phpBookTwo->getTitle());

      $mySqlBook = $bookFactoryInstance->makeMySQLBook();
      writeln('MySQL Author: '.$mySqlBook->getAuthor());
      writeln('MySQL Title: '.$mySqlBook->getTitle());
  }

  function writeln($line_in) {
    echo $line_in."<br/>";
  }
```

```
/*
BEGIN TESTING ABSTRACT FACTORY PATTERN


testing OReillyBookFactory
first php Author: Rasmus Lerdorf and Kevin Tatroe
first php Title: Programming PHP
second php Author: David Sklar and Adam Trachtenberg
second php Title: PHP Cookbook
MySQL Author: George Reese, Randy Jay Yarger, and Tim King
MySQL Title: Managing and Using MySQL


testing SamsBookFactory
first php Author: Christian Wenz
first php Title: PHP Phrasebook
second php Author: George Schlossnagle
second php Title: Advanced PHP Programming
MySQL Author: Paul Dubois
MySQL Title: MySQL, 3rd Edition


END TESTING ABSTRACT FACTORY PATTERN
*/
```