

Capítulo 4

Patrones y Patrones de Diseño (i)

Diseño y Programación Orientado a Objetos
Ingeniería Informática
Ingeniería Técnica de Informática de Sistemas y Gestión
Optativa (6 créditos)

<http://www.info-ab.uclm.es/assignaturas/42579>



Objetivos

- El presente capítulo pretende proporcionar herramientas para conseguir reutilización. Para ello introduce el concepto de patrón y hace un recorrido por los patrones de diseño propuestos por la banda de los cuatro (Gamma et al., 1994)



Bibliografía

- Design Patterns. Elements of Reusable Object-Oriented Software
Gamma, e., Helm, R., Johnson, R., Vlissides, J.
Addison Wesley, 1994 (Edición de 2003 en castellano)
- UML y Patrones. Introducción al análisis y diseño orientado a objetos
Larman, G.
Prentice Hall. 1999
- <http://webs.teleprogramadores.com/patrones/>
- <http://www.dofactory.com/patterns/patterns.aspx>
- <http://www.jacana.org.uk/pattern/>



Introducción

- El desarrollo del software es una tarea complicada que depende en gran medida de la **experiencia** de las personas involucradas
- La **comprensión del software** es uno de los problemas más complicados en la tarea de **mantenimiento** y **evolución**
- La sociedad requiere sistemas más complejos y más grandes. Los recursos desarrollados cada vez son más escasos. Debe existir un mecanismo de **reutilización**
- El paradigma orientado a objetos es el más utilizado en los últimos años. Se ha comprobado que este paradigma ofrece muchas ventajas



Introducción

- Con la reutilización se consigue:
 - Reducción de tiempos
 - Disminución del esfuerzo de mantenimiento
 - Eficiencia
 - Consistencia
 - Fiabilidad
 - Protección de la inversión en desarrollos
- ¿Cómo reutilizar? Usando mecanismos como: **Componentes**, **Frameworks**, **objetos distribuidos** o **Patrones de diseño**



*“Se debe ser consumidor de reutilización
antes de ser productor de reutilización”*

Los patrones. Origen

- Tienen su origen en un campo distinto al de la informática: La Arquitectura
- Christopher Alexander recopila 253 patrones (véase <http://www.jacana.org.uk/pattern/>)
- El objetivo era crear entornos que redundaran en una **quality without name** que al final se traduce en calidad de vida de los residentes en un país, ciudad o edificio
- Para alcanzar la calidad, hay que atravesar **la puerta** (lenguaje común de patrones)
- A la puerta se llega a través de **un camino**



Los patrones. Evolución

- Se utiliza inicialmente en el campo de la **Arquitectura** por Christopher Alexander. Mitad de los 70.
 - 1977. “A pattern language: Towns/Builder/Construction”
 - 1979, “The Timeless Way of Building”
- La idea se extrapola al terreno de la **informática**
 - 1989, Cunningham y Beck → Smalltalk
 - 1991, Coplien → idioms (patrones en C++)
 - 1994, Gang of Four (GoF) → Design patterns
 - 1997, Brad Appleton
 - ...



Los patrones. Definición

“Cada patrón describe un **problema** que ocurre una y otra vez en nuestro **entorno**, para describir después el núcleo de la **solución** a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma”

Christopher Alexander

Patrón = <problema, contexto, solución>
+<recurrencia, enseña, nombre>



Los patrones. Características

- **Solucionar un problema:** Los patrones capturan soluciones, no sólo principios o estrategias abstractas
- **Ser un concepto probado:** capturan soluciones demostradas, no teorías o especulaciones
- **La solución no es obvia:** los mejores patrones generan una solución a un problema de forma indirecta
- **Describe participantes y relaciones entre ellos:** describen módulos, estructuras del sistema y mecanismos complejos
- **El patrón tiene un componente humano significativa:** todo software proporciona a los seres humanos confort y calidad de vida (estética y utilidad)



Los patrones. Clases

- **Patrones de arquitectura:** se expresa una organización o esquema estructural fundamental para sistemas software. Proporciona un conjunto de subsistemas predefinidos y sus responsabilidades
- **Patrones de diseño:** proporciona esquemas para refinar subsistemas o componentes de un sistema
- **Patrones de programación:** describe la implementación de aspectos de componentes
- **Patrones de análisis:** prácticas que aseguran la consecución de un buen modelo de un problema y su solución
- **Patrones organizacionales:** describen la estructura y prácticas de las organizaciones humanas



Los patrones. Facilidades

- Facilitan la comunicación interna
- Ahorran tiempo y experimentos inútiles
- Mejoran la calidad del diseño y la implementación
- Son como “normas de productividad”
- Facilitan el aprendizaje de los paquetes de Java
- ...



Los patrones de diseño

- Los diseñadores expertos no resuelven cada problema desde el principio
- Se pueden encontrar patrones de clases y comunicaciones entre objetos en muchos sistemas orientados a objetos
- Los patrones de diseño ayudan a los diseñadores a reutilizar con éxito diseños para obtener nuevos diseños
- El objetivo de los patrones es guardar la **experiencia** en diseños de programas orientados a objetos
- Los patrones de diseño hacen más fácil reutilizar con éxito los diseños y arquitecturas, ayudan a elegir entre diseños alternativos, hacen a un sistema reutilizable y evitan alternativas que comprometen la **reutilización**



Elementos característicos

- Un patrón de diseño tiene cuatro elementos característicos:
 - El **nombre** del patrón, describe el problema de diseño, su solución, y consecuencias en una o dos palabras. Tener un vocabulario de patrones nos permite hablar sobre ellos
 - El **problema** describe cuando aplicar el patrón. Se explica el problema y su contexto. Puede describir estructuras de clases u objetos que son sintomáticas de un diseño inflexible. Se incluye una lista de condiciones
 - La **solución** describe los elementos que forma el diseño, sus relaciones, responsabilidades y colaboraciones. No se describe un diseño particular. Un patrón es una plantilla
 - Las **consecuencias** son los resultados de aplicar el patrón



Descripción de un patrón

- Nombre
- Objetivo (Problema)
- Contexto
- Aplicabilidad (Fuerzas)
- Solución
- Consecuencias
- Implementación
- Usos en el API de Java
- Código del ejemplo
- Patrones relacionados



Cualidades de un patrón de diseño

- **Encapsulamiento y abstracción.** Cada patrón encapsula un problema bien definido y su solución en un dominio particular
- **Extensión y variabilidad.** Cada patrón debería ser abierto por extensión o parametrización por otros patrones, de tal forma que pueden aplicarse juntos para solucionar un gran problema
- **Generatividad y composición.** Cada patrón una vez aplicado genera un contexto resultante, el que concuerda con el contexto inicial de uno o más de uno de los patrones del catálogo
- **Equilibrio.** Cada patrón debe realizar algún tipo de balance entre sus efectos y restricciones



*Un patrón describe un todo que es
más grande que la suma de sus partes*

Tipos de patrones de diseño

- Patrones de **creación**. Muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades
- Patrones **estructurales**. Describen la forma en que diferentes tipos de objetos pueden ser organizados para trabajar unos con otros
- Patrones de **comportamiento**. Se utilizan para organizar, manejar y combinar comportamientos



Patrones de diseño (GoF)

Creación:

Abstract Factory
Builder
Factory Method
Prototype
Singleton

Estructurales:

Adapter
Bridge
Composite
Decorator
Façade
Flyweight
Proxy

Comportamiento:

Chain of Responsibility
Command
Interpreter
Iterator
Mediator
Memento
Observer
State
Strategy
Template Method
Visitor



Patrones de creación. Descripción

- **Abstract Factory:** Proporciona una interfaz para **crear familias de objetos** relacionados o dependientes sin especificar su clase concreta
- **Builder:** Permite a un objeto **construir un objeto complejo** especificando sólo su tipo y contenido
- **Factory Method:** Define una interfaz para **crear un objeto** dejando a las subclases decidir el tipo específico al que pertenecen
- **Prototype:** Permite a un objeto **crear objetos personalizados** sin conocer su clase exacta a los detalles de cómo crearlos
- **Singleton:** Garantiza que solamente se **crea una instancia** de la clase y provee un punto de acceso global a él



Patrones estructurales. Descripción

- **Adapter:** convierte la interfaz que ofrece una clase en otra esperada por los clientes
- **Bridge:** desacopla una abstracción de su implementación y les permite variar independientemente
- **Composite:** permite gestionar objetos complejos e individuales de forma uniforme
- **Decorator:** extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes
- **Façade:** simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto de comunicación
- **Flyweight:** usa la compartición para dar soporte a un gran número de objetos de grano fino de forma eficiente
- **Proxy:** proporciona un objeto con el que controlamos el acceso a otro objeto



Patrones de comportamiento. Descripción

- **Chain of Responsibility:** evita el acoplamiento entre quien envía una petición y el receptor de la misma
- **Command:** encapsula una petición de un comando como un objeto
- **Interpreter:** dado un lenguaje define una representación para su gramática y permite interpretar sus sentencias
- **Iterator:** acceso secuencial a los elementos de una colección
- **Mediator:** define una comunicación simplificada entre clases
- **Memento:** captura y restaura un estado interno de un objeto



Patrones de comportamiento. Descripción

- **Observer**: una forma de **notificar cambios** a diferentes clases dependientes
- **State**: **modifica el comportamiento** de un objeto cuando su estado interno cambia
- **Strategy**: define una **familia de algoritmos**, encapsula cada uno y los hace intercambiables
- **Template Method**: define un **esqueleto de algoritmo** y delega partes concretas de un algoritmo a las subclases
- **Visitor**: representa una **operación** que será realizada sobre los elementos de una estructura de objetos, permitiendo definir nuevas operaciones sin cambiar las clases de los elementos sobre los que opera



Patrones de diseño. Creación

- Abstract factory
- Builder
- Factory Method
- Prototype
- Singleton



Abstract Factory

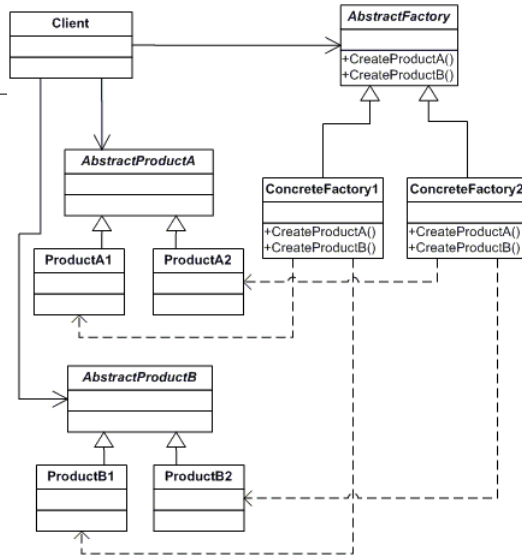
AbstractFactory: Declara una interfaz de operaciones que crean productos abstractos

ConcreteFactory: Implementa las operaciones que crean los objetos producto

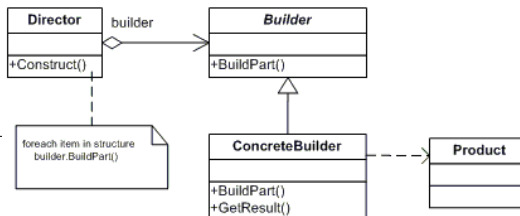
AbstractProduct: Declara una interfaz para un tipo de objeto producto

Product: Define un objeto producto que será creado por el correspondiente ConcreteFactory

Client: Usa las interfaces



Builder



Builder: Define una interfaz para la creación de partes de un objeto complejo

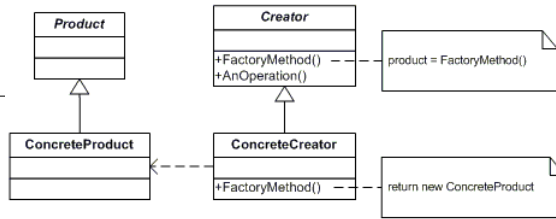
ConcreteBuilder: Implementa la interfaz de *Builder*, mantiene referencia del producto creado y proporciona una interfaz que retorna el producto. Ensambla las piezas constituyentes

Director: Construye un objeto usando la interfaz proporcionada por Builder

Product: Define las partes constituyentes del producto y representa un objeto complejo bajo construcción



Factory Method



Creator: Declara el método FactoryMethod(). Se puede definir una implementación por defecto para él

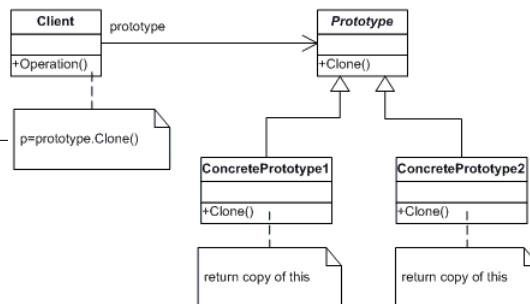
ConcreteCreator: Sobrescribe el método FactoryMethod(), devolviendo una instancia de ConcreteProduct

Product: Define la interfaz de objetos que crea el método FactoryMethod

ConcreteProduct: Implementa la interfaz del producto



Prototype



Prototype: Declara una interfaz definiendo el método clone()

ConcretePrototype: implementa el método clone()

return (Prototype) this;

Client: Crea un nuevo objeto pidiendo a un prototipo que se clone



Singleton

Singleton
-instance : Singleton
-Singleton()
+Instance() : Singleton

Singleton: Define un método Instance() que permite al cliente el acceso a su única instancia. El método Instance() es estático
Esta misma clase es la responsable de la creación y mantenimiento de su propia instancia

```
class Singleton{  
    private static Singleton instance;  
    protected Singleton(){ }  
    public static Singleton Instance{  
        if (instance == null) instance = new Singleton();  
        return instance;  
    }  
}
```



Curso 2003/04

Diseño y Programación
Orientada a Objetos

27

Capítulo 4 Patrones y Patrones de Diseño (i)

Diseño y Programación Orientado a Objetos
Ingeniería Informática
Ingeniería Técnica de Informática de Sistemas y Gestión
Optativa (6 créditos)

<http://www.info-ab.uclm.es/asignaturas/42579>

