POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño
4. Aplicación

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Selección

1. **Considerar cómo los patrones de diseño resuelven problemas.** Valorar cómo los patrones de diseño ayudan a encontrar objetos, determinar la granularidad de los objetos, especificar las interfaces de los objetos y otras más. Haciendo referencia a este aspecto puede ayudar a encontrar el patrón correcto

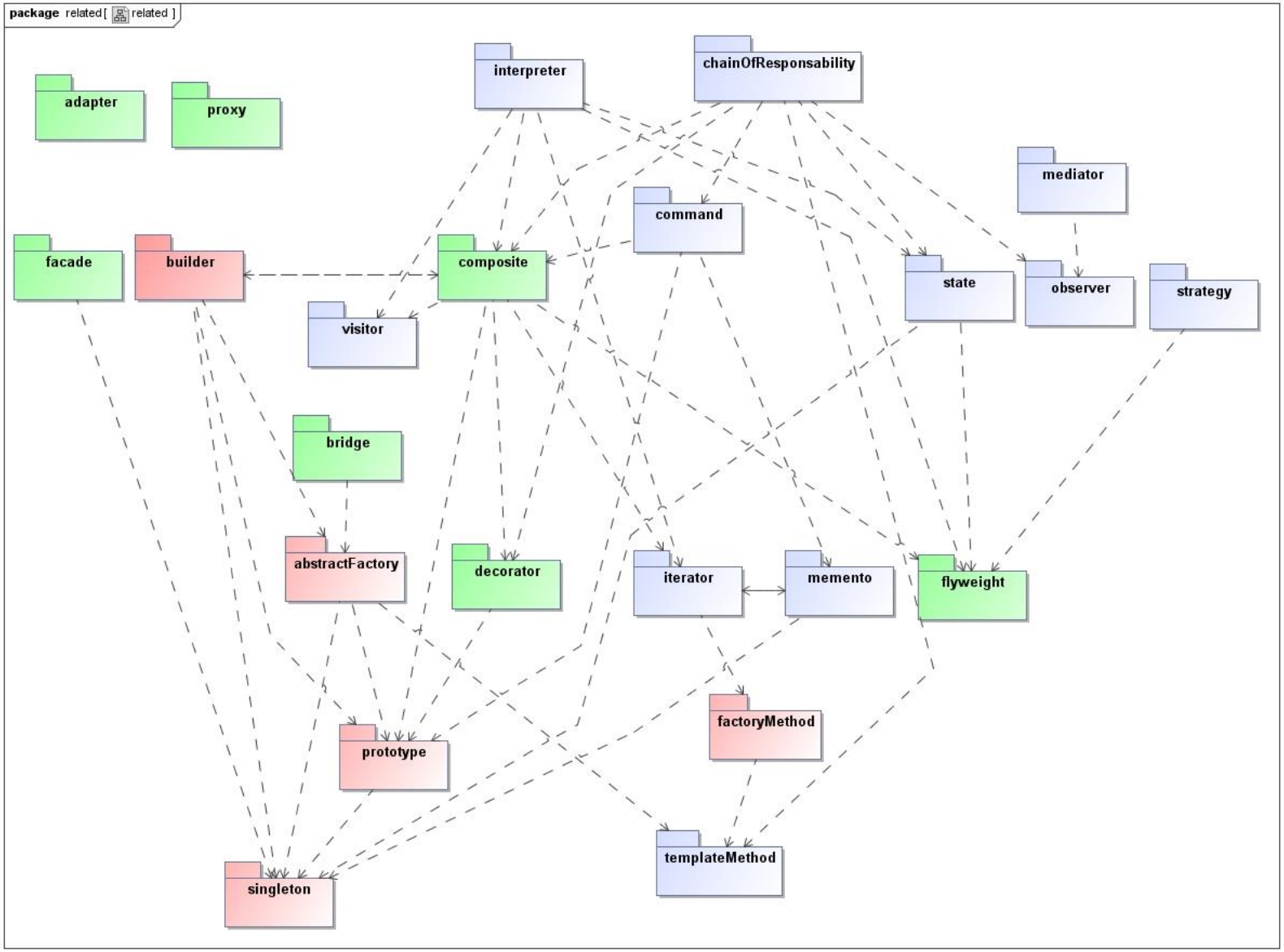
1. Selección

		Encontrando objetos adecuados	Determinando la granularidad de los objetos	Especificando la interfaz de los objetos	Especificando la implementación de objetos		Poninendo Mecanismos de Reusabilidad a trabajar			Relacionando estructuras de tiempo de compilación y ejecución
					Herencia de clases vs interfaces	Programando para la interfaz, no para la implementación	Herencia vs Composición	Delegación	Herencia vs Tipos Parametrizados	
Creacionales	Singleton					***				
	Factory Method					***				
	Abstract Factory		***			***				
	Prototype					***				
	Builder		***			***				
Estructurales	Facade		***							
	Composite	***			***					***
	Decorator			***						***
	Adapter									
	Proxy			***						
	Bridge							***		
	Flyweight		***							
De Comportamiento	Template Method								***	
	Iterator									
	State	***			***			***		
	Strategy	***			***			***		
	Command		***		***					
	Memento			***						
	Mediator							***		
	Observer				***					***
	Visitor		***	***				***		
	Interpreter									
	Chain of Responsibility				***			***		***

1. Selección

2. **Analizar las secciones de intención.** Leer la intención de cada patrón para encontrar uno o más que parecen relevantes para el problema
3. **Estudiar cómo los patrones se interrelacionan.** Estas relaciones pueden ayudar directamente a seleccionar el patrón o conjunto de patrones correcto

1. Selección


package related[related]
adapter
proxy
interpreter
chainOfResponsability
mediator
command
facade
builder
composite
state
observer
strategy
visitor
bridge
abstractFactory
decorator
iterator
memento
flyweight
factoryMethod
prototype
templateMethod
singleton

1. Selección

4. **Estudiar el propósito de los patrones.** Profundiza en las similaridades y diferencias entre los propósitos de los patrones.
5. **Examina las causas de rediseño.** Empezar observando si el problema involucra uno o más causas. Entonces, buscar los patrones que ayuden a evitar las causas de re-diseño.

1. Selección

		Relacionando estructuras de tiempo de compilación y ejecución	Creando un objeto especificando una clase explícitamente	Dependencia de operaciones específicas	Dependencia de la plataforma hardware/software	Dependencia de las representaciones o implementaciones de objetos	Dependencias algorítmicas	Fuerte acoplamiento	Extendiendo funcionalidad por subclasificación	Inabilidad para alterar clases convenientemente
Creacionales	Singleton									
	Factory Method		***							
	Abstract Factory		***		***	***		***		
	Prototype		***							
	Builder						***			
Estructurales	Facade							***		
	Composite	***							***	
	Decorator	***							***	***
	Adapter									***
	Proxy					***				
	Bridge				***	***		***	***	
	Flyweight									
De Comportamiento	Template Method						***			
	Iterator						***			
	State									
	Strategy						***		***	
	Command			***				***		
	Memento					***				
	Mediator							***		
	Observer	***						***	***	
	Visitor						***			***
	Interpreter									
	Chain of Responsibility	***		***				***	***	

1. Selección

6. **Considera que sería variable en el diseño**. Este enfoque es el opuesto al enfoque de las causas de re-diseño. En vez de considerar lo que podría forzar un cambio de diseño, considera qué se quiere ser capaz de cambiar sin re-diseño. Este enfoque se apoya en la encapsulación de lo que varía, un aspecto de muchos patrones.

1. Selección

	Patrón de Diseño	Aspecto que varía
Creacionales	Abstract Factory	familias de objetos "producto"
	Builder	cómo se crea un compuesto
	Factory Method	subclase de objeto que es instanciado
	Prototype	clase de objeto que es instanciado
	Singleton	la única instancia de una clase
Estructurales	Adapter	la interfaz de un objeto
	Bridge	la implementación de un objeto
	Composite	la estructura y composicion de un objeto
	Decorator	las responsabilidades de un objeto sin subclasificación
	Facade	la interfaz de un subsistema
	Flyweight	costes de almacenamiento de los objetos
	Proxy	cómo se accede a un objeto, su localización
De Comportamiento	Chain of Responsibility	el objeto que puede completar una petición
	Command	cuándo y cómo una petción es completada
	Interpreter	la gramática e interpretación de un lenguaje
	Iterator	cómo se accede, recorre ... los elementos de un agregado
	Memento	qué información privada es almacenada fuera de un objeto y cuándo
	Mediator	cómo y qué objetos interactúan con otros
	Observer	número de objetos que dependen de otro objeto, cómo estar actualizado
	State	estados de un objeto
	Strategy	un algoritmo
	Template Method	pasos de un algoritmo
	Visitor	operaciones que pueden ser aplicadas a objetos sin cambiar sus clases

2. Uso

1. **Leer el patrón como una visión general.** Poner atención particular a la aplicabilidad y consecuencias para asegurar que el patrón es el correcto para el problema
2. **Estudiar las secciones de Estructura, Participantes y Colaboradores.** Asegurar que se comprenden las clases y objetos del patrón y cómo se relacionan entre ellos
3. **Leer la sección de Código de Ejemplo para ver un ejemplo concreto del patrón en código.** Estudiar el código ayuda a aprender a cómo implementar el patrón

2. Uso

4. **Elegir nombre para los participantes del patrón que sean significativos en el contexto de la aplicación.** Los nombre de los participantes en los patrones de diseño son generalmente demasiado abstractos para aparecer directamente en una aplicación. En cambio, es útil incorporar el nombre de los participantes en el nombre que aparece en la solicitud. Eso ayuda a que el patrón sea más explícito en la aplicación. Por ejemplo, si se utiliza el patrón de *Strategy* para un algoritmo de composición de textos, entonces es posible tener clases *SimpleLayoutStrategy* o *TeXLayoutStrategy*.

2. Uso

5. **Definir las clases.** Declarar su interfaz, establecer sus relaciones de herencia y definir las variables de instancia que representan los datos y referencias a objetos. Identificar clases existentes en la aplicación que el patrón afecta y modificarlas acordemente.
6. **Definir nombres específicos de la aplicación para las operaciones en el patrón.** Otra vez, los nombres generalmente dependen de la aplicación. Usar las responsabilidades y colaboraciones asociadas con cada operación como guía. También, ser consistentes en las convenciones de nombrado. Por ejemplo, se podría usar el prefijo "créate" consistentemente para denotar un método factoría.
7. **Implementar las operaciones que lleven a cabo las responsabilidades y colaboraciones en el patrón.** La sección de Implementación ofrece pistas para guiar la implementación. La sección de Código de Ejemplo también puede ayudar.

3. Resumen

1. Selección

1. Considerar cómo los patrones de diseño resuelven **problemas**
2. Analizar las secciones de **Intención**.
3. Estudiar cómo los patrones se **interrelacionan**
4. Estudiar el **Propósito** de los patrones.
5. Examina las **causas de re-diseño**
6. Considera que sería **variable en el diseño**

2. Uso

1. Leer el patrón como una visión general.
2. Estudiar las secciones de **Estructura**, **Participantes** y **Colaboradores**.
3. Leer la sección de **Código de Ejemplo** para ver un ejemplo concreto del patrón en código
4. Elegir nombre para los participantes del patrón que sean significativos en el contexto de la aplicación
5. Definir las clases
6. Definir nombres específicos de la aplicación para las operaciones en el patrón
7. Implementar las operaciones que lleven a cabo las responsabilidades y colaboraciones en el patrón