UNIVERSIDAD NACIONAL de MAR DEL PLATA

Facultad de Ingeniería

Análisis y Diseño de Sistemas II

(6B4)

Carrera: Ingeniería en Informática

Equipo Docente

Adjunto: Ing. Pedro Fernando Soriano

JTP: Ing. Estanislao Srecko Mileta

Ayudante: Lic. Galia Carolina Serruya

Unidad IV. El diseño orientado a objetos

Objetivo: Implementar el diseño con OO

Temario:

- Diseño de aplicaciones OO. Convertir requerimientos en clases
- Asignación de responsabilidades. Patrones GRASP
- Patrones de diseño
- Diseño de frameworks
- Refactoring

Diseño de aplicaciones OO

La OO hoy representa el mejor framework metodológico para la ingeniería de software gracias al pragmatismo del paradigma y la sistematización de procesos que permite.

- Homogeneidad a través del análisis, diseño e implementación.
- Énfasis en el estado, comportamiento e interacción de objetos.
- Maduración y patrones de prácticas.
- Variedad de técnicas, métodos, procesos, estándares, modelos, notaciones, herramientas, componentes, lenguajes, ambientes, ejemplos, comunidad, práctica y experiencia.
- Testing y métricas

(Larman, 2003)

El Proceso Unificado (UP)

Marco de desarrollo

Disciplina	*Artefacto* *Iteración →*	*Inicio* *I1*	*Elab.* *E1…En*	*Const.* *C1…Cn*	*Trans.* *T1…T2*
Modelado del Negocio	Modelo del Dominio		c		
Requisitos	Modelo de Casos de Uso Visión Especificación Complementaria Glosario	c c c c	r r r r		
Diseño	Modelo de Diseño Documento de Arquitectura SW Modelo de Datos		c c c	r r	
Implementación	Modelo de Implementación		c	r	r
Gestión del Proyecto	Plan de Desarrollo SW	c	r	r	r
Pruebas	Modelo de Pruebas		c	r	
Entorno	Marco de Desarrollo	c	r		

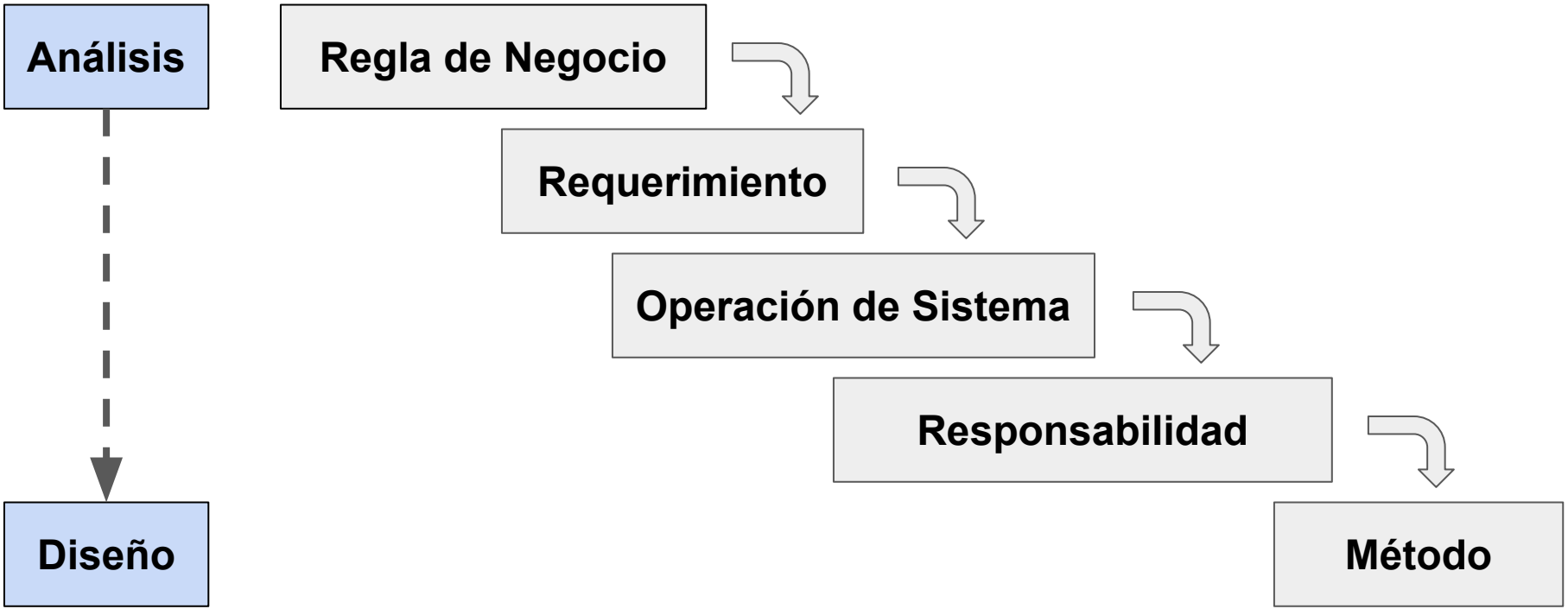
(Larman, 2003)

Diseño de aplicaciones OO

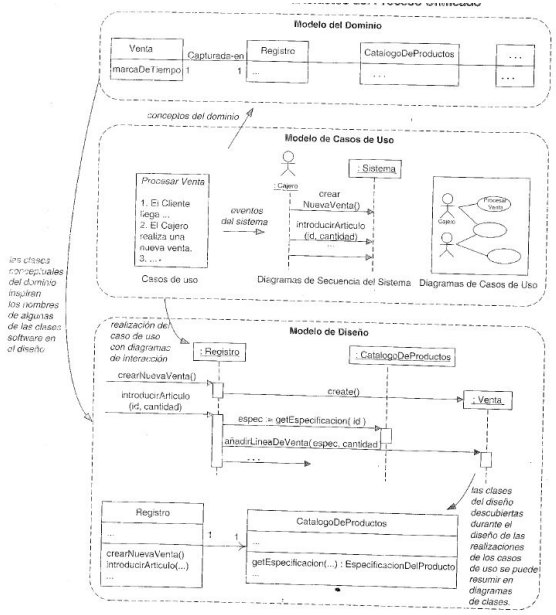
- Artefactos más relevantes
  - Dinámicos: diagramas de interacción de UML
  - Estructural: diagrama de diseño de clases UML
- Competencias más relevantes
  - Principios para la asignación de responsabilidades
  - Principios y patrones de diseño
- Si bien el UML se usa, no es lo más relevante

(Larman, 2003)

Diseño de aplicaciones OO



Diseño de aplicaciones OO - Proceso Unificado



Ampliar imagen...

(Larman, 2003)

Principios de Diseño

- The Open Close Principle (OCP) - polimorfismo
  - *“A module should be open for extension but closed for modification.”*
- The Liskov Substitution Principle (LSP) - contratos
  - *“Subclasses should be substitutable for their base classes.”*
- The Dependency Inversion Principle (DIP) - abstractas
  - *“Depend upon Abstractions. Do not depend upon concretions.”*
- The Interface Segregation Principle (ISP) - interf. simples
  - *“Many client specific interfaces are better than one general purpose interface.”*

(Pressman, 2010)

TP SOLID

Diseño de aplicaciones OO

Modelo de Dominio

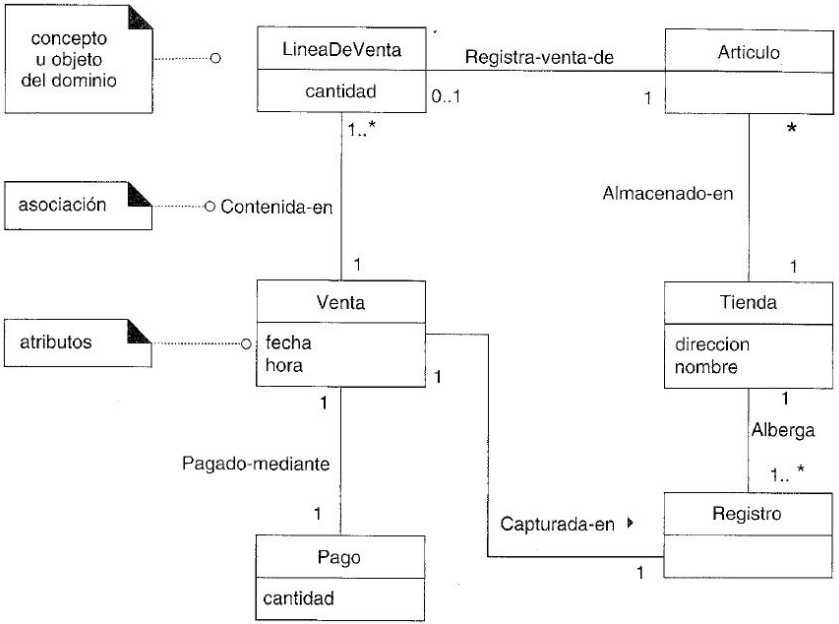
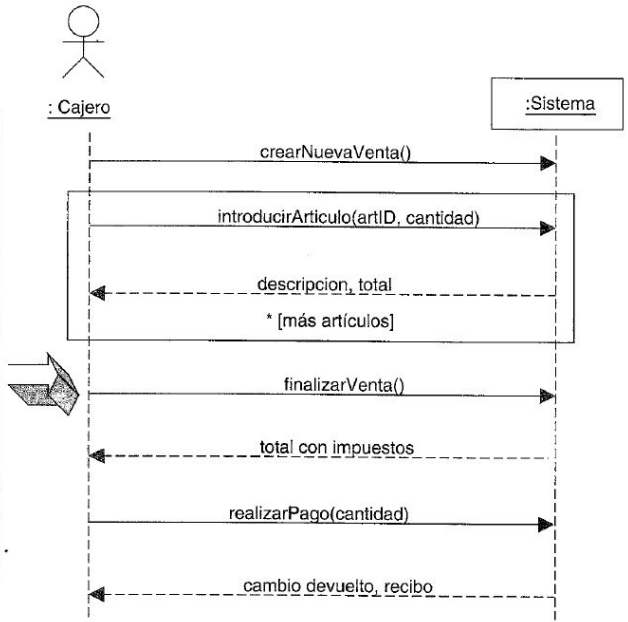


Diagrama de Secuencia de Sistema

Escenario simple de *ProcesarVenta* para el pago en efectivo

1. El Cliente llega al terminal PDV.
2. El Cajero inicia una nueva venta.
3. El Cajero inserta el identificador del artículo.
4. El Sistema registra la línea de venta y presenta la descripción del artículo, precio y suma parcial.
El Cajero repite los pasos 3 y 4 hasta que se indique.
5. El Sistema muestra el total con los impuestos calculados.
6. El Cajero le dice al Cliente el total, y pide que le pague.
7. El Cliente paga y el Sistema gestiona el pago.
...



Patrones GRASP

- Controlador
- Experto en información
- Creador
- Alta cohesión
- Bajo acoplamiento
- Polimorfismo
- Fabricación pura
- Indirección
- Variaciones protegidas

(Larman, 2003)

Patrones GRASP - Controlador

Problema

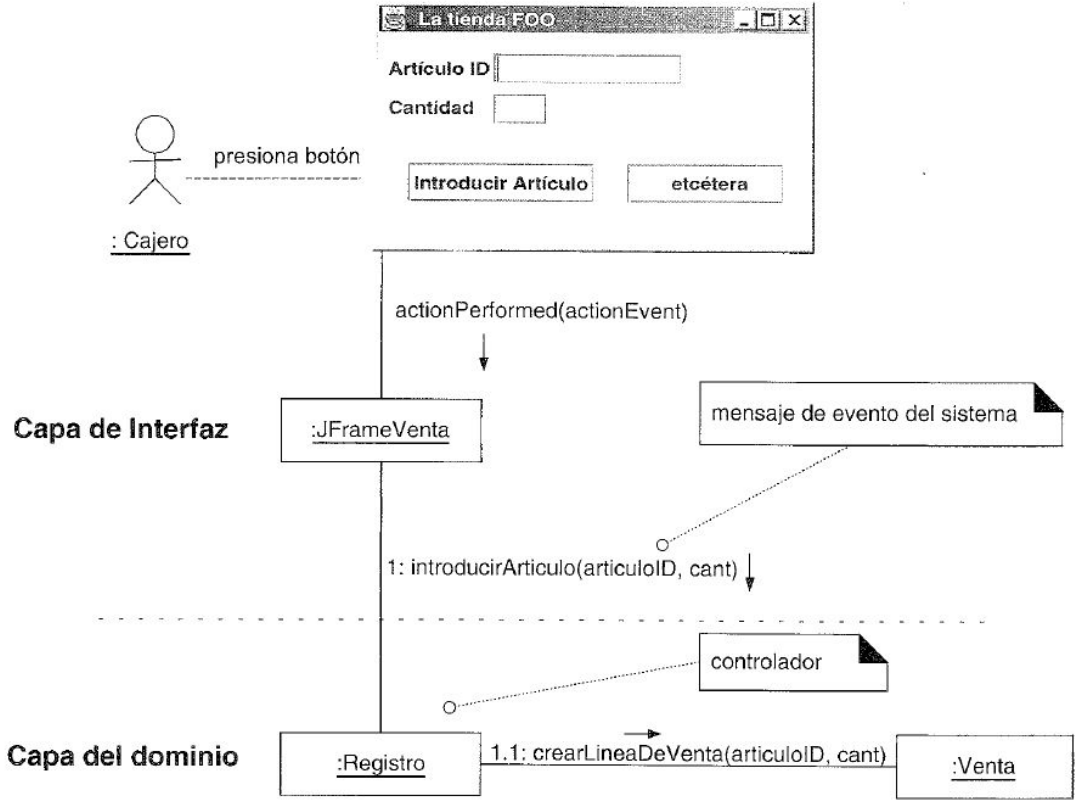
Quién debe ser el responsable de gestionar un evento de entrada del sistema (operación de sistema)?

Solución

Asignar la responsabilidad de manejar un mensaje de evento de entrada del sistema a una clase controladora que representa una parte del sistema

(Larman, 2003)

Patrones GRASP - Controlador



(Larman, 2003)

Patrones GRASP - Controlador

Controlador

Objeto que no pertenece a la interfaz de usuario, responsable de recibir o manejar un evento del sistema, es decir, define un método para las operaciones de sistema.

(Larman, 2003)

Patrones GRASP - Experto en Información

Problema

Cuál es el principio general para asignar responsabilidades a los objetos?

Solución

Asignarlas a la clase que tiene toda la información necesaria para realizar una responsabilidad, el experto de información.

(Larman, 2003)

Patrones GRASP - Creador

Problema

Quién debería ser el responsable de la creación de una nueva instancia de alguna clase?

Solución

B es un Creador de A, si se asigna a la clase B la responsabilidad de crear una instancia de la clase A, y se cumple uno o más de estos casos….

(Larman, 2003)

Patrones GRASP - Creador

- B agrega objetos de A
- B contiene objetos de A
- B registra/persiste instancias de objetos de A
- B utiliza más estrechamente objetos de A
- B tiene los datos de inicialización necesarios al momento de crear A

(Larman, 2003)

Patrones GRASP

- Controlador
- Experto en información
- Creador
- Alta cohesión
- Bajo acoplamiento
- Polimorfismo
- Fabricación pura
- Indirección
- Variaciones protegidas

(Larman, 2003)

Patrones de diseño (GoF)

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular.

(Gamma, 1995)

Patrones de diseño (GoF)

Cómo usar un patrón de diseño?

1. Conocer en gran parte la solución genérica que brinda cada patrón y el tipo de problemática que resuelve
2. Evaluar la posibilidad de aplicar el o los patrones en el problema correspondiente
3. Definir la implementación de los mismos en nuestro modelo
4. Implementar los métodos correspondientes

(Gamma, 1995)

Patrones de diseño (GoF)

By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	• Factory Method	• Adapter (class)	• Interpreter • Template Method
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter (object) • Bridge • Composite • Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento • Observer • State • Strategy • Visitor

(Gamma, 1995)

Abstract Factory (GoF)

Propósito

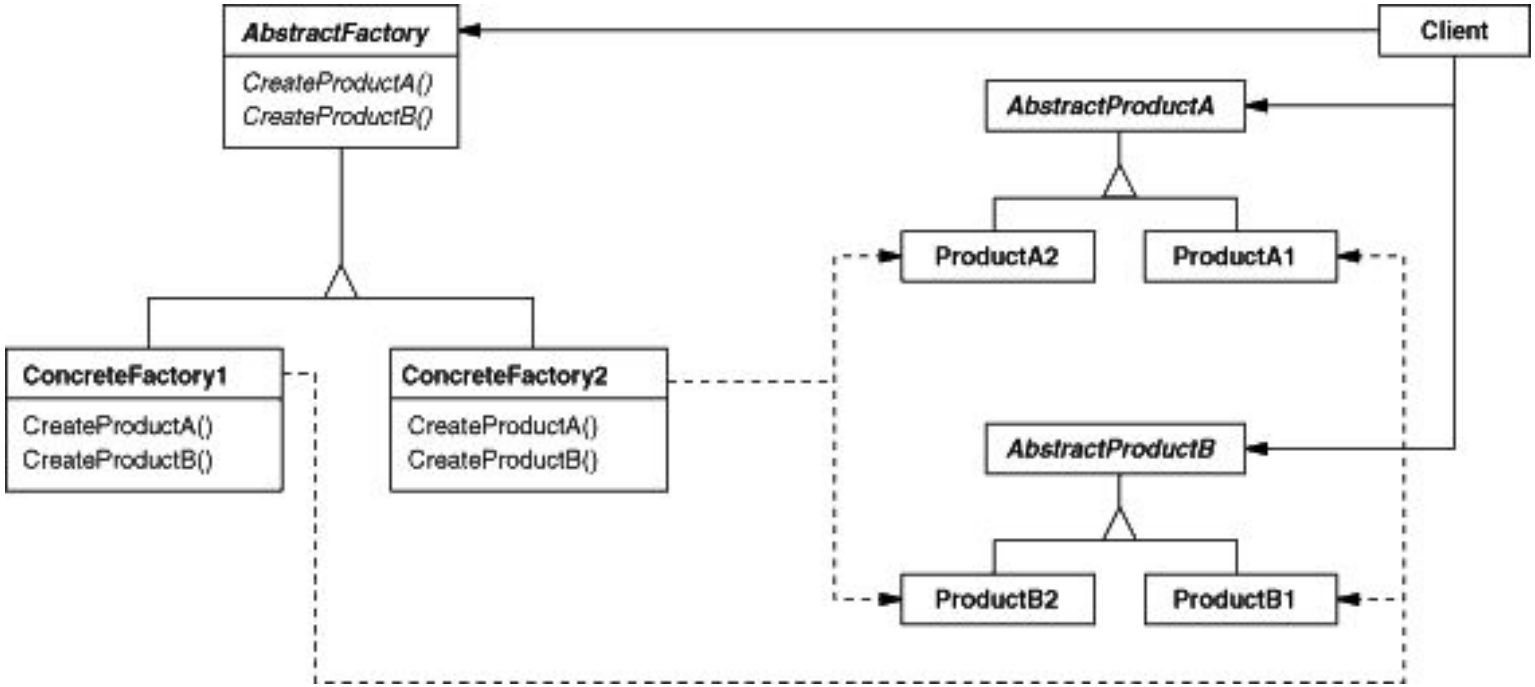
Proporciona una interfaz para crear familias de objetos relacionados a que dependen entre sí, sin especificar sus clases concretas.

Motivación

(Gamma, 1995)

- Un sistema podría ser independiente de cómo se crean, componen y representan sus productos
- Un sistema podría ser configurado para usar una familia de productos de entre varias
- Una familia de objetos es cohesiva y se usan todos en conjunto
- Se puede ofrecer una biblioteca de clase revelando solo sus interfaces

Abstract Factory (GoF)



(Gamma, 1995)

Composite (GoF)

Propósito

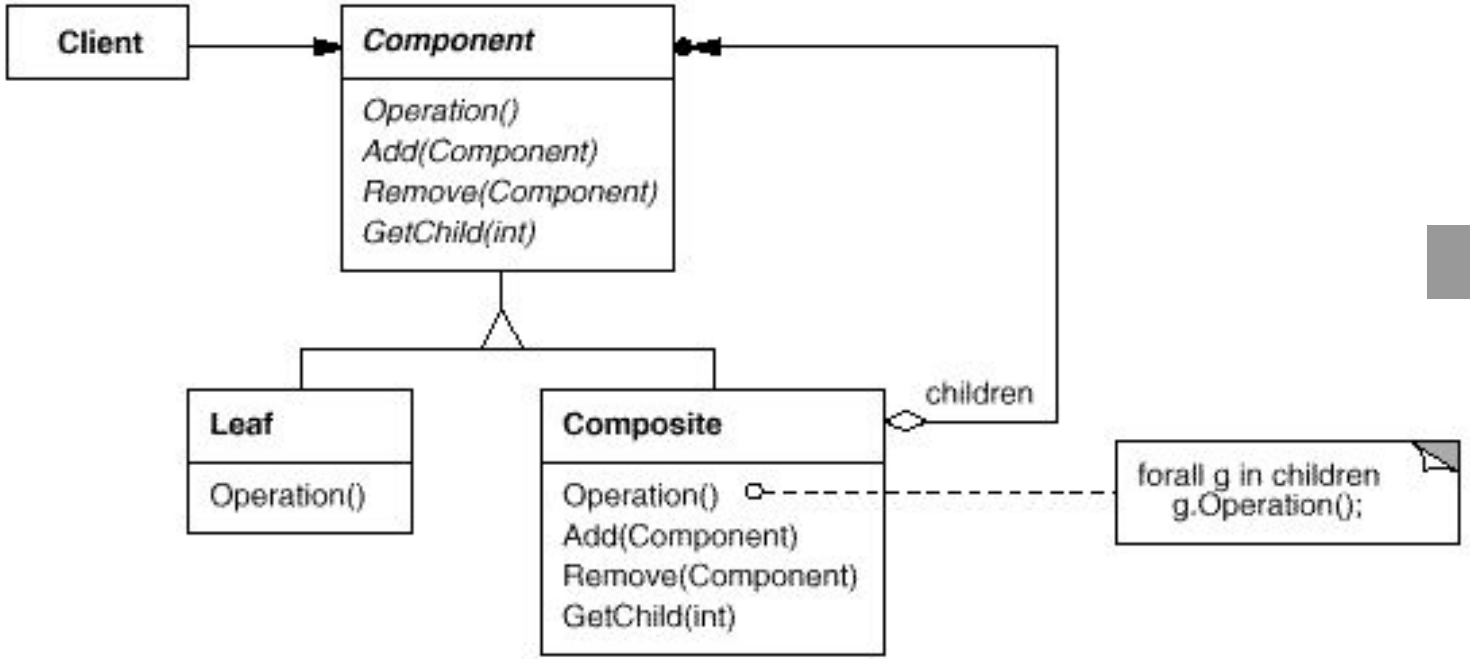
Componer objetos de complejidad mayor mediante otros más sencillos de forma recursiva.

(Gamma, 1995)

Motivación

- Cuando se necesite manejar de igual forma objetos sencillos o agrupaciones de éstos.
- Se necesite representar jerarquías de objetos (árbol)

Composite (GoF)



(Gamma, 1995)

State (GoF)

Propósito

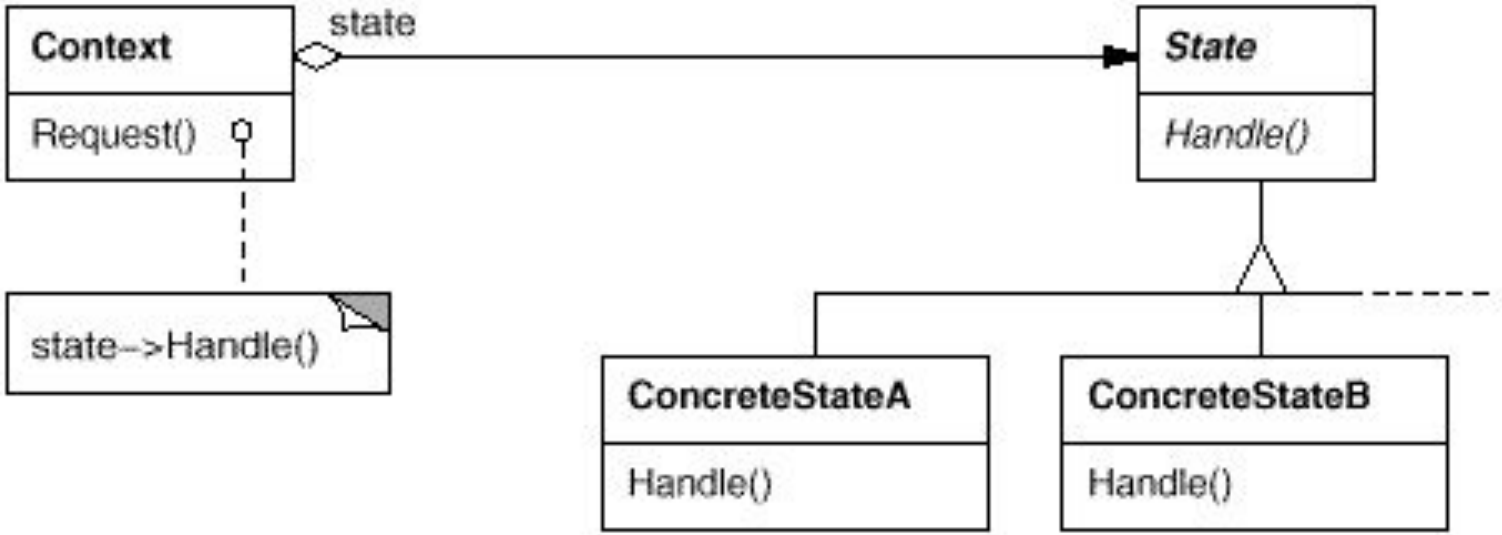
Modelo para que un objeto cambie su comportamiento dependiendo de un estado.

(Gamma, 1995)

Motivación

- Cuando se necesita implementar comportamiento basado en estados o una máquina de estados (State diagram, UML)

State (GoF)



(Gamma, 1995)

Frameworks

Un framework es una aplicación reusable, y “semi completa”, que puede ser especializada para integrarse a otra aplicación.

Es una colección de clases que en conjunto constituyen una solución genérica, a una familia de requerimientos de un dominio específico.

Frameworks

- Objetivo: Reusabilidad
  - Componer aplicaciones componiendo piezas de SW reusables.
  - + Productividad, + Calidad, -Mantenimiento
- Usos
  - FMK por herencia (white box). Template Method (GoF)
  - FMK por composición (black box)
- Inversión de control

Template Method (GoF)

Propósito

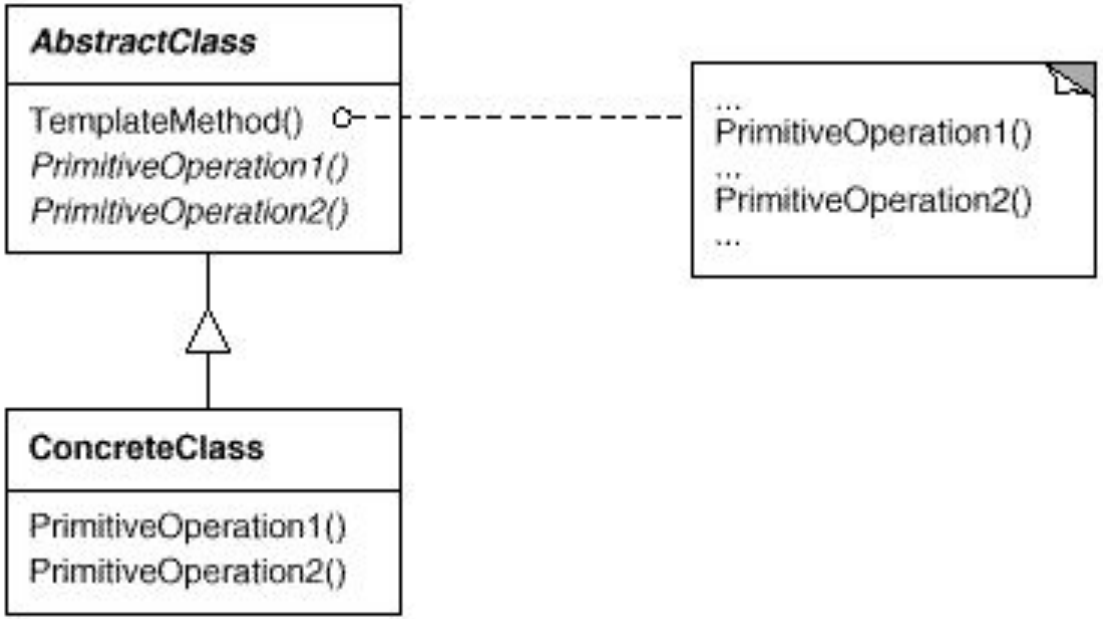
- Definir el esqueleto del algoritmo de un método, delegando tareas a las subclases para que redefinan o completen la funcionalidad pendiente.

Motivación

- Se puede implementar partes invariables de un algoritmo, dejando a las subclases que definan el comportamiento que puede variar.

(Gamma, 1995)

Template Method (GoF)



(Gamma, 1995)

Frameworks

Ventajas

- Reutilización de código y diseño
- Mejora la calidad
- Puede simplificar la programación de un sistema

Desventajas

- Depende del lenguaje
- Debug
- Curva de aprendizaje e integración de múltiples frameworks

Refactoring

...to restructure software by applying a series of refactorings without changing its observable behavior.

(Fowler, 2004)

Unidad IV. Bibliografía y referencias

Bibliografía:

"Ingeniería del software". I. Sommerville. 9na Ed. Pearson. 2011
"Ingeniería del software. Un enfoque práctico". R. Pressman. 7ma Ed. Mc Graw Hill. 2010
"UML y Patrones. Introducción al análisis y diseño orientado a objetos". Larman - Prentice Hall. 2003.
"Análisis y diseño de sistemas". K. Kendall y J. Kendall. 8va Ed. Pearson. 2005
“Patterns of Enterprise Application Architecture”. M. Fowler. AW. 2003

Enlaces:

https://martinfowler.com/bliki/DefinitionOfRefactoring.html M. Fowler. 2004