CC61J / CC5404 - Taller de UML
Apuntes de Clase

Prof. Andrés Muñoz Ordenes

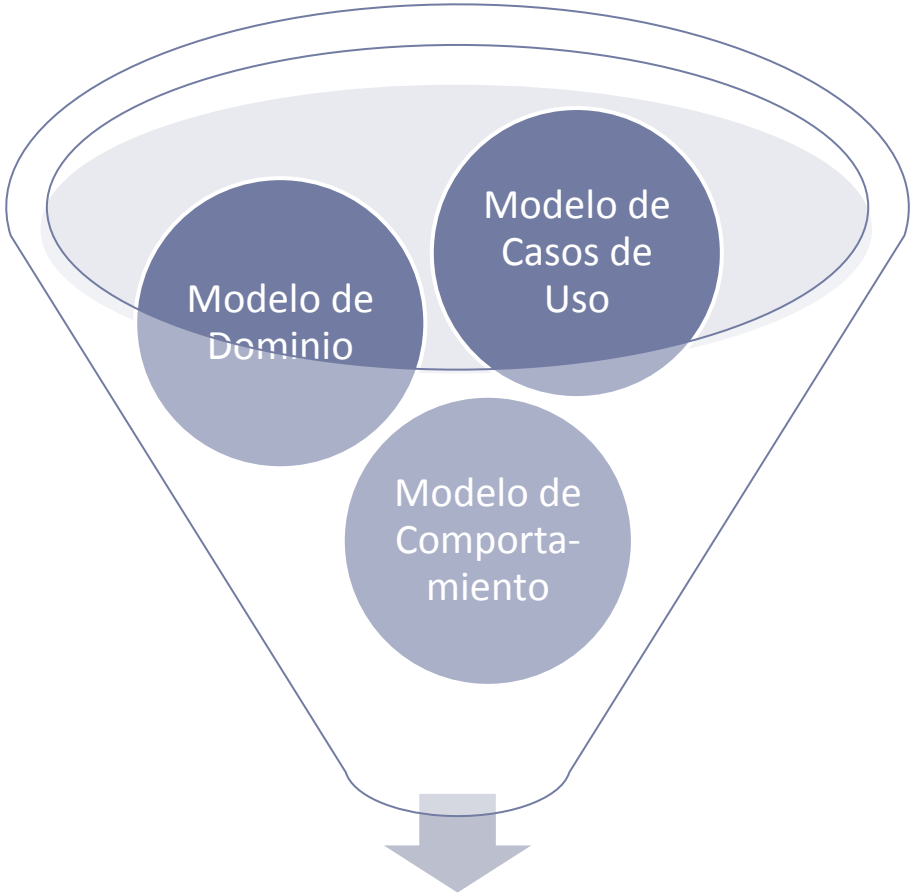
08/06/2011

Agenda

- Motivación
- Diseño del Problema
  - Diagrama de Colaboración
    - Conceptos
    - Notación y Sintaxis
    - Usos
    - Ejemplo: Caja de Supermercado
  - Patrones de Diseño
    - GRASP
    - GoF

Motivación

Motivación



Modelo de Diseño

Diagrama de Clases Conceptuales

Conceptos

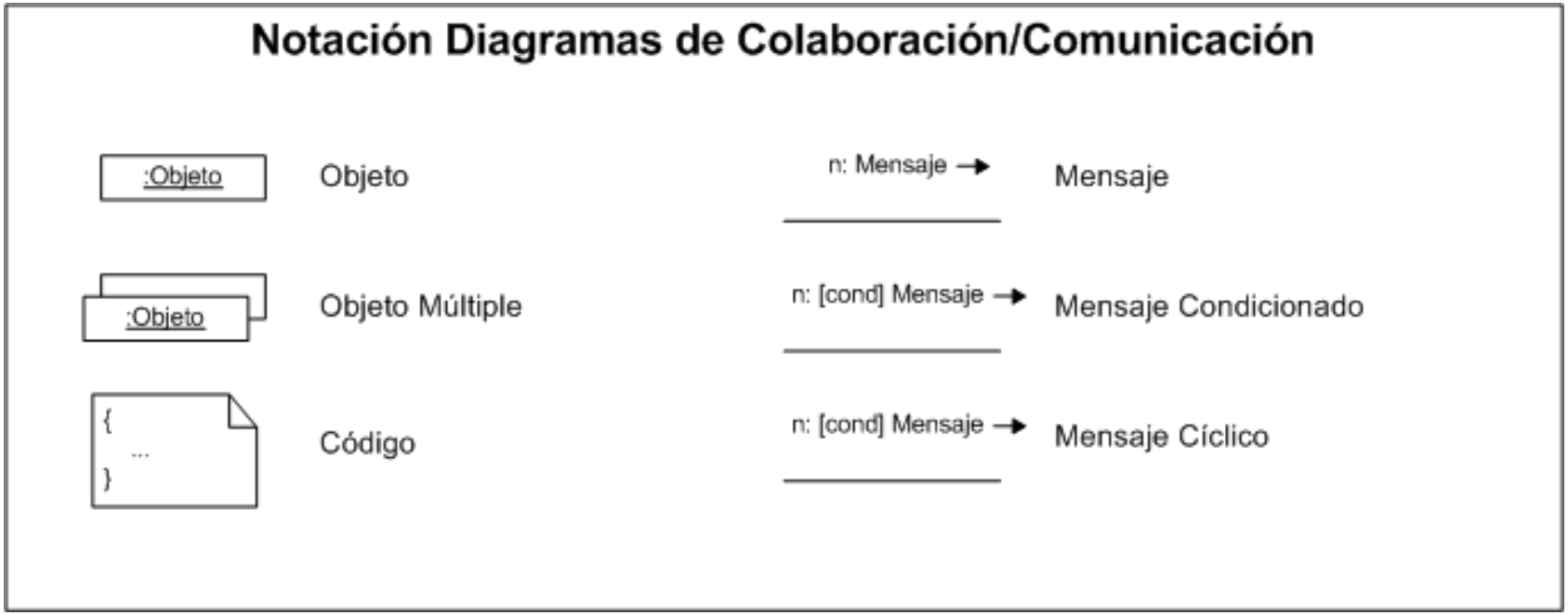
- Operaciones del sistema (Actor – Sistema):
  - Definen cuáles deben ser las acciones en un escenario.
  - Tienen definiciones particulares:
    - Precondicones: Indica los objetos de las clases conceptuales que deben existir y los atributos o condiciones que se deben cumplir.
    - Postcondiciones: Indica los cambios que la operación realiza sobre los objetos del sistema (creación, asociación y cambio de atributo).

- Responsabilidad: Contrato u obligación de un objeto en cuanto a su comportamiento.

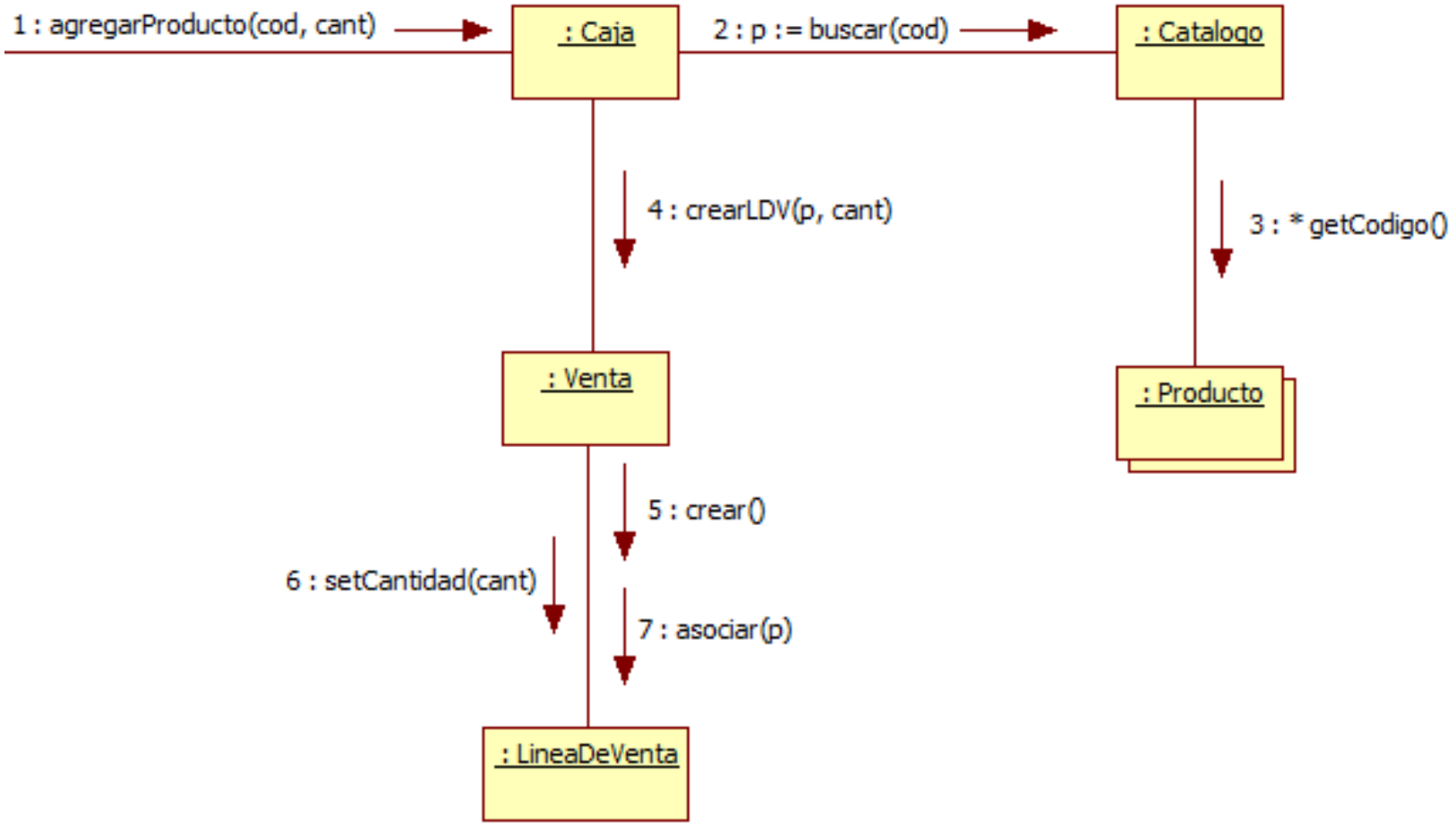
Conceptos

- Los diagramas de colaboración:
  - Se centran en la colaboración de los objetos internos del sistema.
  - Permiten mostrar la temporalidad y profundidad de la ejecución.
  - Su desarrollo permite encontrar partes del diseño en forma sencilla.
  - Son equivalentes a utilizar diagramas de secuencia con el nivel de detalle del diseño.
  - Su uso es a través del uso de **Patrones de Diseño**.
  - Se hacen a partir de las operaciones principales del sistema.

Notación


Notación Diagramas de Colaboración/Comunicación
:Objeto
Objeto
n: Mensaje
Mensaje
:Objeto
Objeto Múltiple
n: [cond] Mensaje
Mensaje Condicionado
{
...
}
Código
n: [cond] Mensaje
Mensaje Cíclico

Sintaxis


1 : agregarProducto(cod, cant)
: Caja
2 : p := buscar(cod)
: Catalogo
4 : crearLDV(p, cant)
3 : * getCodigo()
: Venta
: Producto
5 : crear()
6 : setCantidad(cant)
7 : asociar(p)
: LineaDeVenta

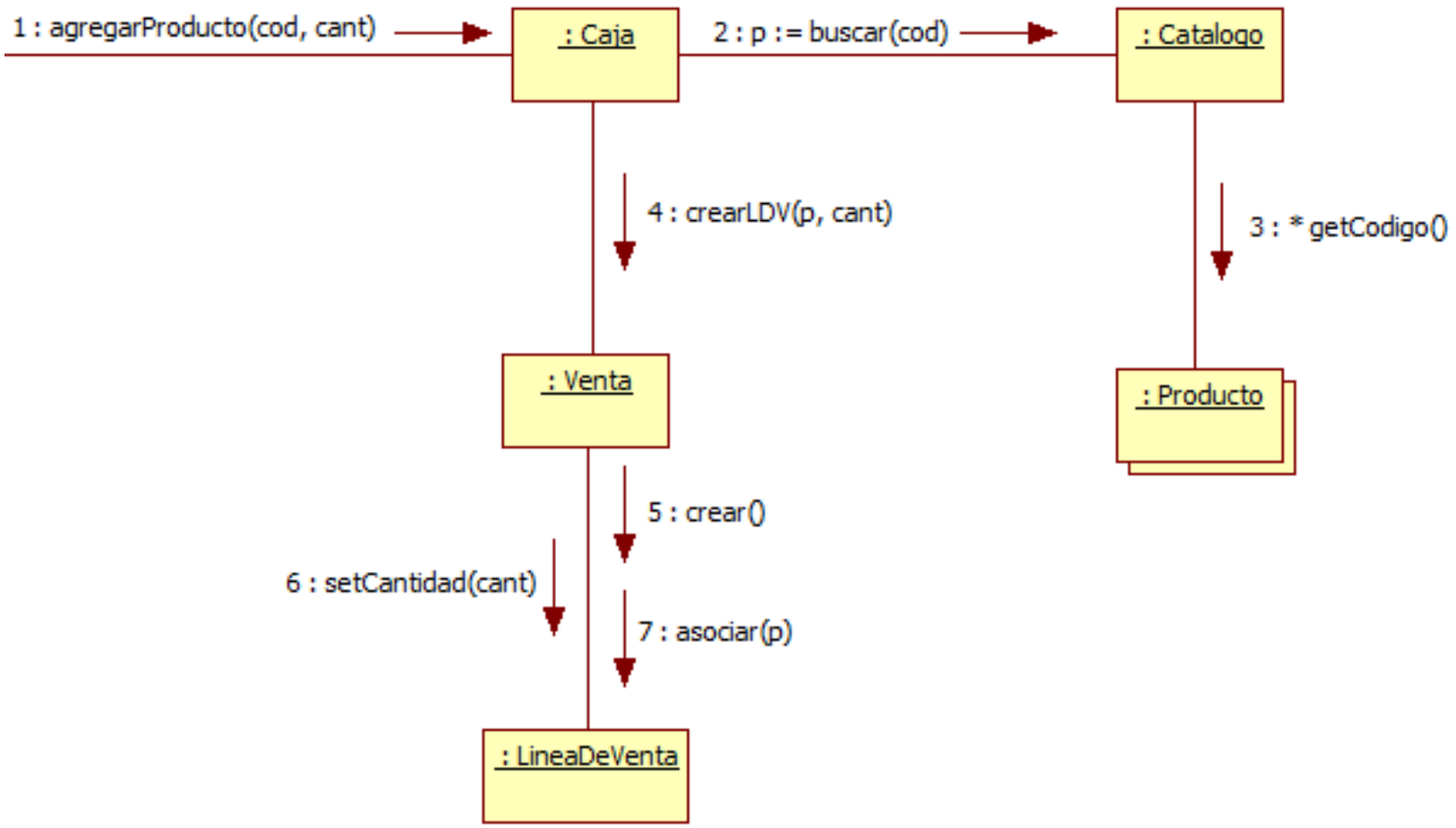
Usos

- Permiten identificar
  - Comunicación que existe entre los objetos de diseño.
  - Asignación de responsabilidades para el cumplimiento de las postcondiciones en las operaciones del sistema.
    - Métodos de las clases.
  - Algoritmo de colaboración para cada operación de sistema.

Técnica

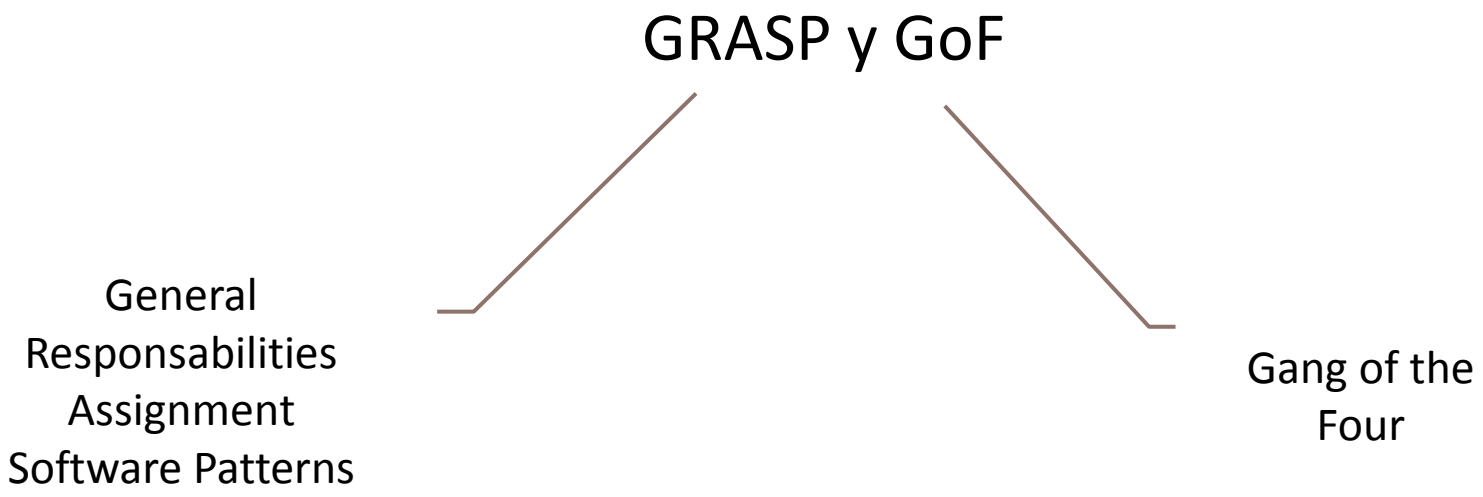
- Identificar las operaciones del sistema.
- Identificar pre-condiciones de cada operación (contrato).
  - Corresponden a los objetos que deben existir en la operación.
- Identificar post-condiciones de cada operación (contrato).
  - Asignación de responsabilidades a los objetos.
  - Aplicación de patrones de diseño.

Ejemplo: TPDV



Patrones de Diseño

- Par Problema-Solución.
- Es aplicable a varios casos generales.
- Consejos para la aplicación de soluciones típicas.
- Mejoran la calidad del diseño.



Patrones GRASP

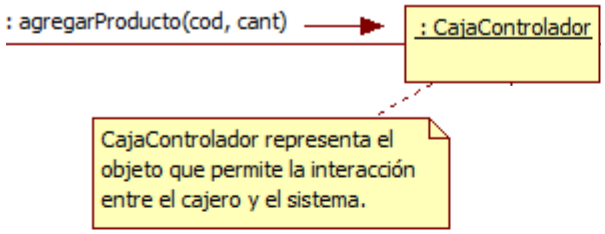
- Patrón de Controlador.
  - Solución: Asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que representa una de las siguientes opciones:
    - Representa el sistema global, dispositivo o subsistema (controlador de fachada).
    - Representa un escenario de caso de uso en el que tiene lugar el evento del sistema (controlador de sesión o de caso de uso).

    Este es nombrado como:

    <Nombre>Controlador, <Nombre>Manejador o <Nombre>Coordinador
  - Problema: ¿Quién debe ser responsable de gestionar un evento de entrada al sistema?

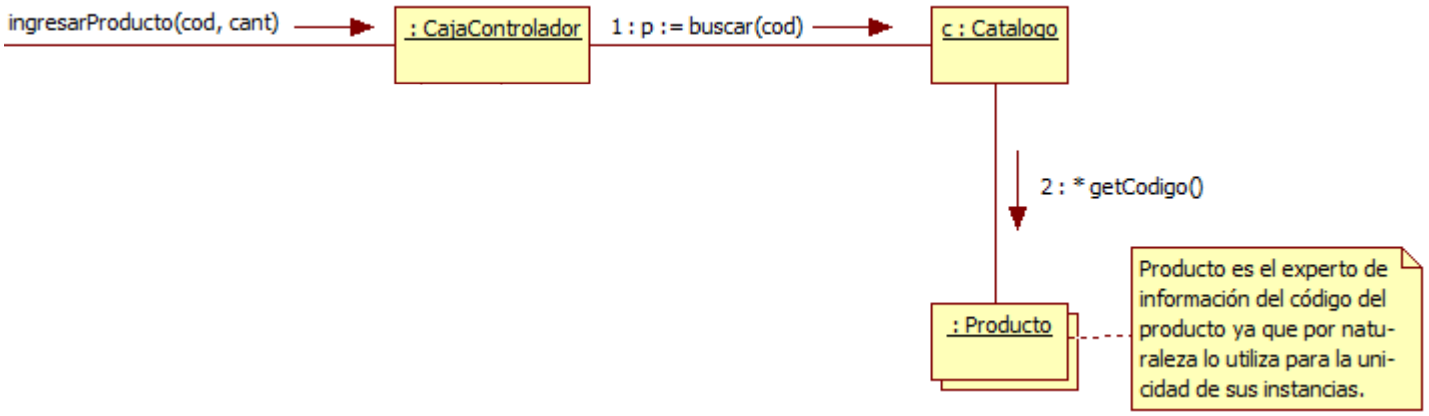
Patrones GRASP

- Patrón de Controlador.



Patrones GRASP

- Patrón Experto de Información.
  - Solución: Asignar una responsabilidad al experto en información – la clase que tiene la información necesaria para realizar la responsabilidad.
  - Problema: ¿Quién debería ser el responsable de conocer una información particular?

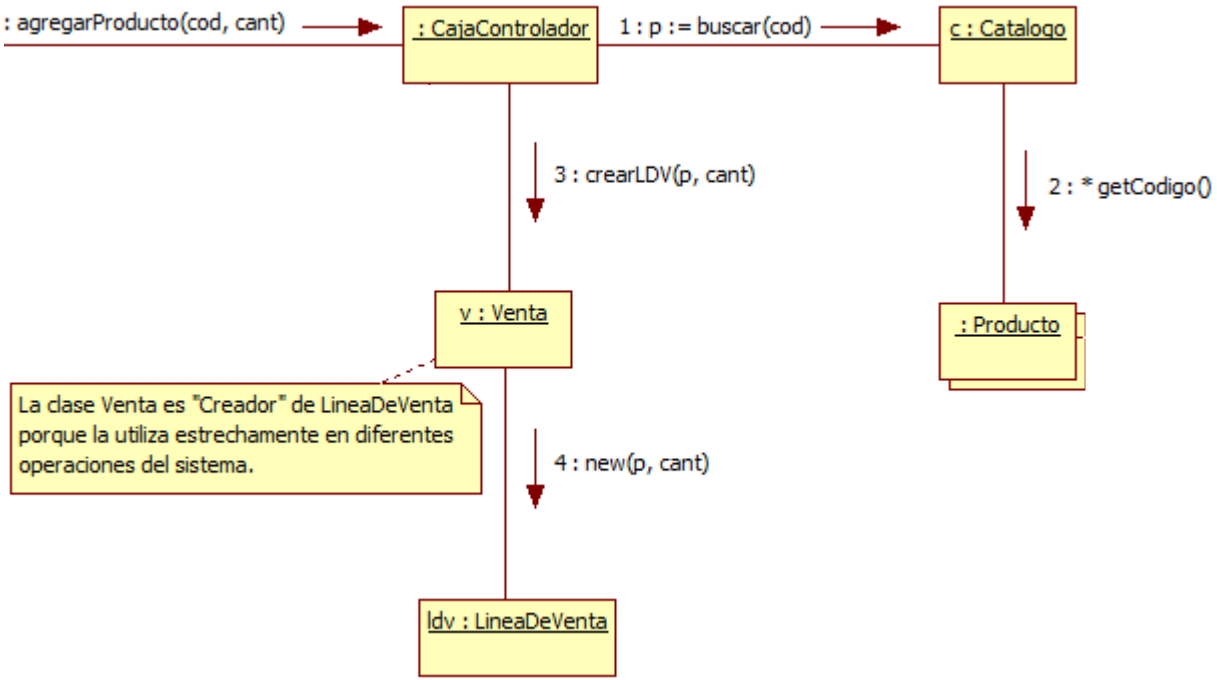


Patrones GRASP

- Patrón de Creador.
  - Solución: Asignar la responsabilidad a la clase B de crear instancias de la clase A si cumple uno o más de los siguientes casos:
    - B agrega objetos de A
    - B contiene objetos de A
    - B registra instancias de objetos de A
    - B utiliza más estrechamente objetos de A
    - B tiene los datos de inicialización que se pasará a un objeto de A cuando sea creado
  - Problema: ¿Quién debería ser el responsable de la creación de una nueva instancia de alguna clase?

Patrones GRASP

- Patrón de Creador.



Patrones GRASP

- Patrón de Alta Cohesión.
  - Solución: Asignar una responsabilidad de manera que la cohesión permanezca alta.
  - Problema: ¿Cómo mantener la complejidad manejable?
  - Cohesión:
    - Información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la ella.
    - Especialización de una clase.

Patrones GRASP

- Patrón de Bajo Acoplamiento.
  - Solución: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.
  - Problema: ¿Cómo soportar bajas dependencias, bajo impacto al cambio e incremento de la reutilización?
  - Acoplamiento:
    - Tener las clases lo menos ligadas entre sí que se pueda.
    - Beneficiar la reutilización y mantenibilidad de las clases.

Patrones GRASP

- Otros Patrones:
  - Patrón de Polimorfismo
    - Solución: Cuando una responsabilidad posee varias alternativas para comportamientos similares, se asignarán a los tipos en que el comportamiento presenta las variantes.
    - Problema: ¿Cómo manejar las alternativas basadas en el tipo?
  - Patrón de Fabricación Pura
    - Solución: Asignar un conjunto altamente cohesionado de responsabilidades a una clase artificial que no pertenece al dominio.
    - Problema: ¿Quién es el responsable de mantener la alta cohesión y el bajo acoplamiento cuando ya no tenemos más alternativas en nuestro modelo?

Patrones GRASP

- Otros Patrones:
  - Patrón de Indirección
    - Solución: Se asigna la responsabilidad a un objeto intermedio, de manera tal que medie entre componentes o servicios para que no terminen directamente acoplados.
    - Problema: ¿De qué manera podemos desacoplar los objetos para así mantener una potencial reutilización y un bajo acoplamiento? ¿Quién debe tener la responsabilidad de mediar para mantener la indirección entre las componentes?
  - Patrón “No Hables con Extraños”.
    - Solución: Se asigna la responsabilidad a un objeto directo del cliente para que colabore con un objeto indirecto, de modo que el cliente no necesite saber nada del objeto indirecto.
    - Problema: ¿Quién debe ser el responsable de evitar conocer la estructura de los objetos indirectos?

Patrones GoF

- Patrones Creacionales.
  - Se aplican en la creación de instancias de objetos en general.
    - Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
    - Builder (Constructor virtual): Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
    - Factory Method (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
    - Prototype (Prototipo): Crea nuevos objetos clonándolos de una instancia ya existente.
    - Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Patrones GoF

- Patrones Estructurales.
  - Define objetos como estructuras de control que sirven en diferentes formas.
    - Adapter (Adaptador): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
    - Bridge (Puente): Desacopla una abstracción de su implementación.
    - Composite (Objeto compuesto): Permite tratar objetos compuestos como si de uno simple se tratase.
    - Decorator (Envoltorio): Añade funcionalidad a una clase dinámicamente.
    - Facade (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
    - Flyweight (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
    - Proxy: Mantiene un representante de un objeto.

Patrones GoF

- Patrones de Comportamiento.
  - Definen algunos tipos de comportamientos de los objetos y entregan información de cómo realizar dichos comportamientos .
    - Chain of Responsibility (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
    - Command (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
    - Interpreter (Intérprete): Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
    - Iterator (Iterador): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
    - Mediator (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
    - Memento (Recuerdo): Permite volver a estados anteriores del sistema.

Patrones GoF

- Patrones de Comportamiento (cont).
  - Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
  - State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
  - Strategy (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
  - Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
  - Visitor (Visitante): Permite definir

Patrones GoF

- Patrones de Sistema.
  - Definen algunas características que dependen de la arquitectura y que deben ser reflejadas en las realizaciones.
    - MVC (Modelo Vista Controlador): Divide un componente o un subsistema en tres partes lógicas: modelo, vista y controlador, facilitando la modificación o personalización de cada parte.
    - Session (Sesión): Ofrece una forma de que los servidores de sistemas distribuidos sean capaces de distinguir los clientes, permitiendo que las aplicaciones asocien el estado con la comunicación entre el cliente y el servidor.
    - Worker Thread (Thread trabajador): Mejora la productividad y minimiza la latencia media.

Patrones GoF

- Patrones de Sistema (cont).
  - Callback (Retrollamada): Permite que un cliente se registre en un servidor para ciertas operaciones. De esta forma, el servidor puede notificar al cliente cuando la operación ha finalizado.
  - Succesive Update (Actualización Sucesiva): Ofrece a los clientes una forma de recibir actualizaciones continuas.
  - Router (Encaminador): Desacopla múltiples fuentes de información de los objetos de esa información.
  - Transaction (Transacción): Agrupa una colección de métodos de forma que todos ellos finalicen correctamente o fallen de forma colectiva.

Departamento de Ciencias de la Computación
UNIVERSIDAD DE CHILE

UNIFIED MODELING LANGUAGE™

Muchas Gracias. Hasta la Próxima.