PROCESO DE DISEÑO DEL SISTEMA

Para crear los entregables del modelo (es decir, los artefactos de éste), se necesita proceder a través del uso de técnicas o recetas. Veamos cada una de ellas en detalle.

DIAGRAMAS DE COLABORACIÓN

UML nos dice que:

> EL DIAGRAMA DE COLABORACIÓN DEFINE GRÁFICAMENTE PARA UN ESCENARIO ESPECÍFICO DE UN CASO DE USO LAS RESPONSABILIDADES GENERADAS ENTRE LOS OBJETOS DE DISEÑO DEL SISTEMA A PARTIR DE LAS OPERACIONES DEL SISTEMA.

Como podemos ver en la definición encontramos cosas parecidas a los diagramas de secuencia, y esto tiene que ver porque vienen de un mismo tipo de diagramas (en UML se conocen como diagramas de interacción). Sin embargo, el hincapié de estos diagramas o su motivación tiene que ver en diseñar las responsabilidades delegadas a partir de la operación de sistema (que ya conocemos) en los objetos internos del sistema. A este proceso de inferencia de responsabilidades se le conoce como aplicación de responsabilidades.

Para hacer estos diagramas, se necesita entender algo de la anatomía de ellos. Es por eso que definimos la notación UML de los diagramas a continuación:

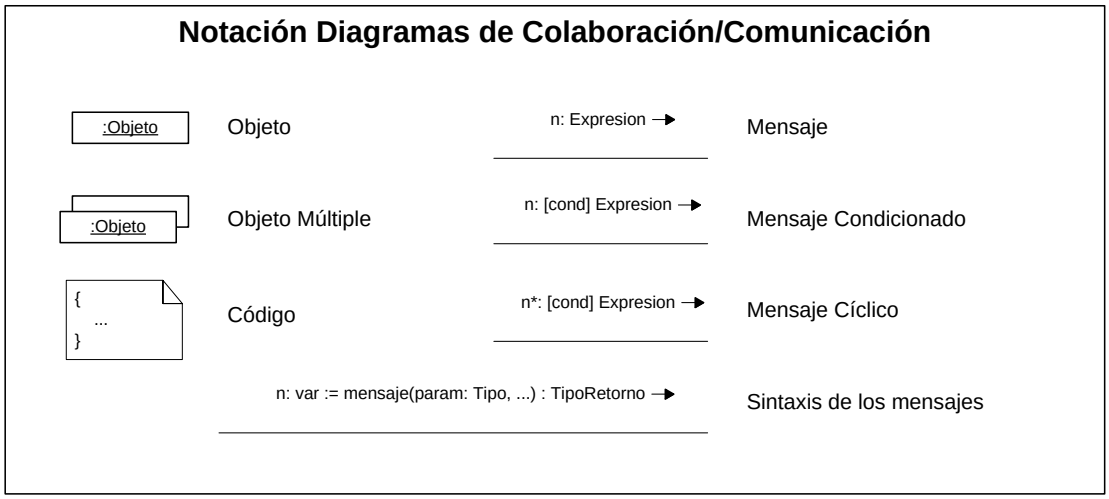


ILUSTRACIÓN 1. NOTACIÓN DE LOS DIAGRAMAS DE COLABORACIÓN

Debemos destacar que los diagramas de colaboración los utilizaremos para describir las operaciones del sistema identificadas en los diagramas de secuencia con el detalle de operación y diseño interno tal como ya lo hemos dicho. De esta manera, tendremos diagramas por cada una de las operaciones.

Un ejemplo de diagrama podemos verlo en el caso del TPDV (caja de supermercado). Si elegimos la operación de agregarProducto(cod, cant) en donde se agrega a la venta en curso una cantidad determinada de cierto código de producto podemos obtener el siguiente diagrama:

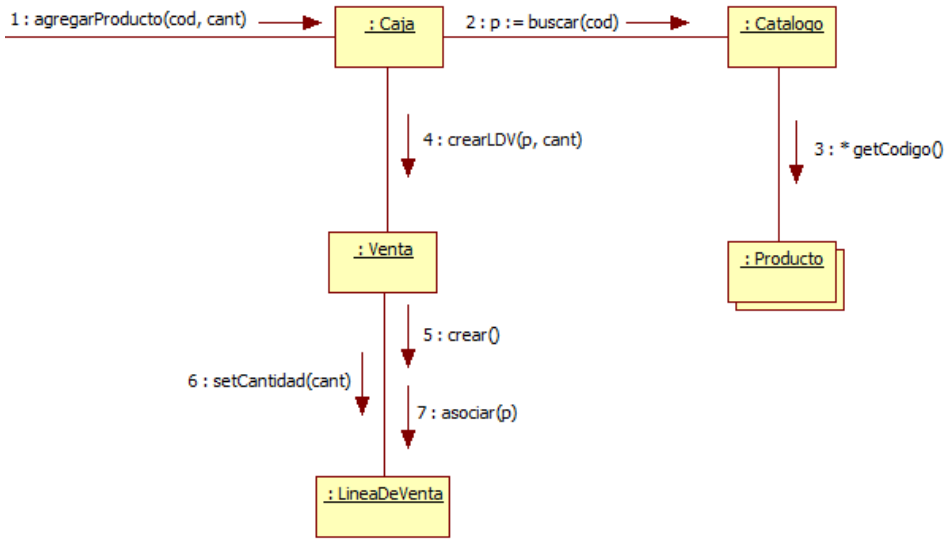


ILUSTRACIÓN 2. DIAGRAMA DE COLABORACIÓN DE INGRESAR PRODUCTO PARA TPDV[1]

Este diagrama es el reflejo de la lista de postcondiciones que se definen en el contrato de la operación indicada. Estas postcondiciones son:

- Que se haya encontrado una instancia p de tipo Producto en c (de tipo Catalogo) cuyo atributo codigo tenga el mismo valor que el parámetro cod (responsabilidades 2 y 3).
- Que se haya creado una instancia ldv de tipo LineaDeVenta (responsabilidad 5).
- Que se haya asociado p a ldv (responsabilidad 7).
- Que se haya cambiado el atributo cantidad de ldv por el valor del parámetro cant (responsabilidad 6).
- Que se haya asociado ldv a v (responsabilidad 4, solo por entregar el control de la creación a Venta).

La intensión de mostrar el diagrama no es dar la fórmula de cómo se realizan sino más bien ilustrar la forma cómo se utiliza la notación anteriormente mostrada, ya que debemos seguir un proceso con el cual obtendremos este diagrama a partir del uso de patrones de diseño.

[1] El programa StarUML no permite poner mensajes sin secuencia, por lo que el diagrama quedó con la operación del sistema indicada por la responsabilidad número 1. Sin embargo, según lo visto en clases, no es relevante.

DIAGRAMA DE CLASES DE DISEÑO

El resultado del diseño (y que por supuesto es de todo nuestro trabajo en este curso) por excelencia es el Diagrama de Clases de Diseño (DCD). Por eso, comenzaremos diciendo que:

> LOS DIAGRAMAS DE CLASES DE DISEÑO SON FIGURAS QUE MUESTRAN LA ESPECIFICACIÓN DE LAS CLASES E INTERFACES DE SOFTWARE QUE PARTICIPAN EN LA SOLUCIÓN.

A partir de esta definición podemos entender, por lo tanto, que el DCD contiene realmente el software como debe ser programado. En ellos encontramos toda la especificación necesaria para la implementación, ya que en ellos se describen los atributos, tipos relevantes y métodos que responden a las responsabilidades que cada clase posee.

Así, lo que encontraremos en este detalle es:

- ✓ Clases, asociaciones y atributos
- ✓ Interfaces, con sus operaciones y constantes
- ✓ Métodos
- ✓ Tipos de datos
- ✓ Navegabilidad
- ✓ Dependencias

Por lo tanto, el nivel de detalle es alto porque a partir de este diagrama se deben escribir las estructuras de clases que compondrán el sistema a implementar, y también gran parte de la lógica interna de cada uno de los métodos. De esta manera, la construcción del sistema está muy completa, dejando poco trabajo para la fase de implementación real.

La notación que se utiliza en estos diagramas es muy parecida a la que usamos en el diagrama de clases conceptuales. Sin embargo, es importante diferenciarlos, ya que en los DCD estamos especificando las clases de software que serán implementadas y no solo conceptos que representan elementos del negocio. Esta notación UML es la siguiente:

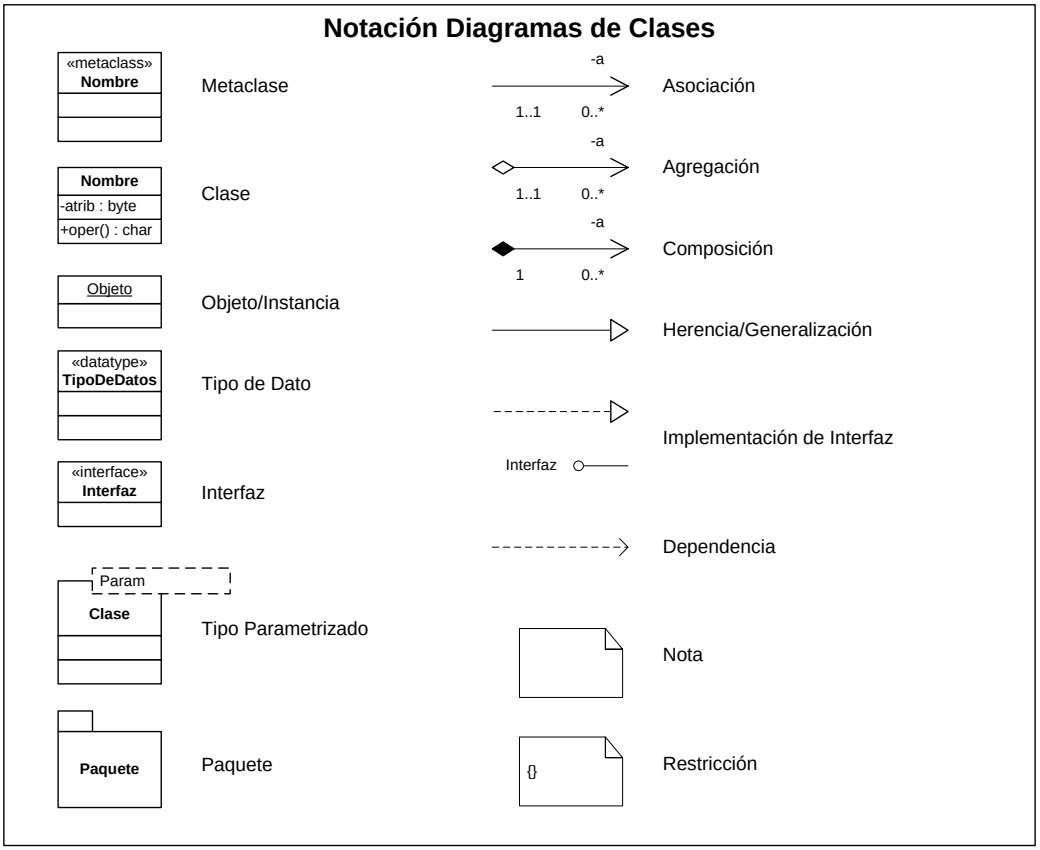


ILUSTRACIÓN 3. NOTACIÓN PARA LOS DIAGRAMAS DE CLASES DE DISEÑO

Veamos un ejemplo en el cual se nota la diferencia entre el modelo conceptual y las clases de diseño. Tomemos la asociación que hay entre Caja y Venta en el problema del supermercado. Para el diagrama de clases conceptuales, tenemos lo siguiente:

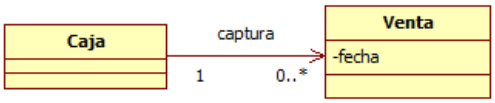


ILUSTRACIÓN 4. ASOCIACIÓN A NIVEL DE CLASES CONCEPTUALES

Después de las realizaciones, este diagrama se convierte en algo más parecido a:

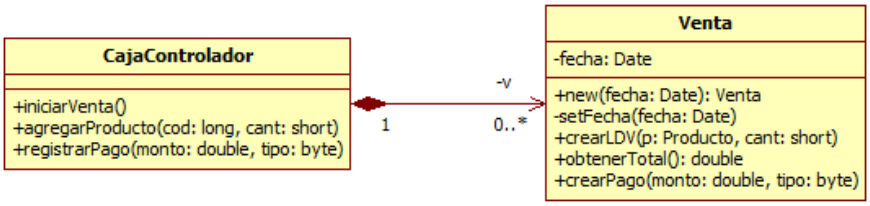


ILUSTRACIÓN 5. ASOCIACIÓN A NIVEL DE CLASES DE DISEÑO

Más adelante descubriremos notación adicional para otro tipo de situaciones específicas y que se relacionan con nuevos patrones de asignación de responsabilidades. Sólo es importante recordar:

- ✓ Considerar cuál es la audiencia del DCD nos dará el nivel de detalle.
- ✓ El DCD es un apoyo y una base para la implementación, por lo es necesario evitar el detalle exhaustivo cuando éste pueda provocar confusión.

TÉCNICA DE CONSTRUCCIÓN DE LOS DCD

En el proceso normal de generación del DCD, todo comienza por identificar las clases de diseño a partir de los diagramas de interacción. Aún cuando esto ya lo hemos hecho de manera parcial, es bueno que reunamos toda la información en un solo diagrama.

PASO 1: IDENTIFICACIÓN DE LAS CLASES DE DISEÑO Y SUS ATRIBUTOS

Lo primero que hacemos es dibujar las clases que se encuentran en los diagramas de interacción y les ponemos los atributos identificados en el Modelo de Dominio:

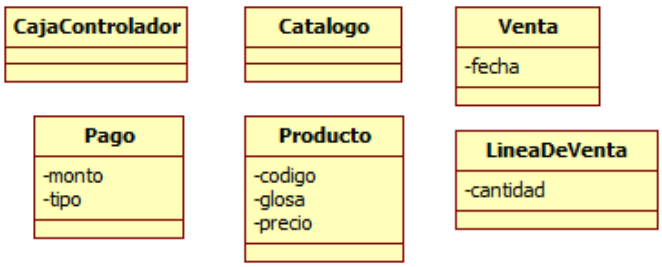


ILUSTRACIÓN 6. IDENTIFICACIÓN DE LAS CLASES DE DISEÑO

IDENTIFICANDO LOS NOMBRES DE LOS MÉTODOS DE LAS CLASES DE DISEÑO

Una vez que eso se encuentra dibujado, comenzamos añadiendo los nombres de los métodos de manera análoga a las responsabilidades identificadas en las realizaciones. De esta forma, los mensajes que encontramos en los diagramas de interacción se convierten en métodos físicos de la clase:

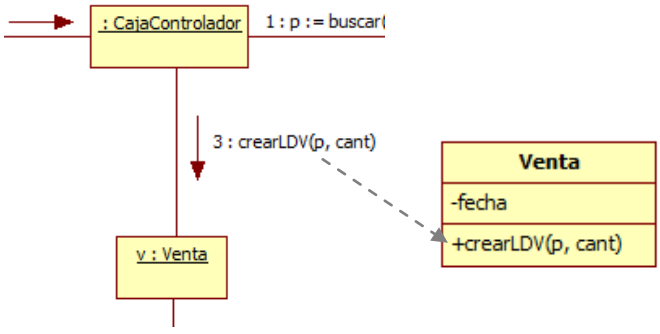


ILUSTRACIÓN 7. IDENTIFICACIÓN DE UN MÉTODO A PARTIR DE UNA COLABORACIÓN

De esta forma, nuestro DCD va creciendo y se convierte en:

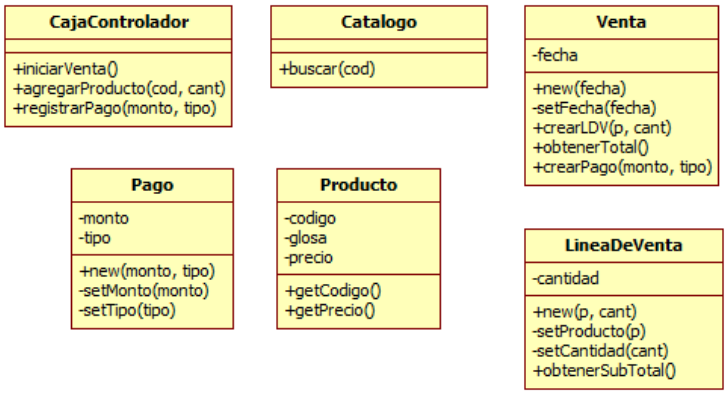


ILUSTRACIÓN 8. IDENTIFICACIÓN DE MÉTODOS DE LAS CLASES

Algo muy importante al bautizar los métodos es tener en cuenta algunas cuestiones que definen un buen diseño:

- ✓ La instanciación o inicialización de objetos no es una cuestión estándar para todos los lenguajes orientados al objeto, por lo que generalmente se omiten en los DCD (en Java, nos referimos a los constructores).

- ✓ Cuando se hace referencia a la obtención de valores de un objeto (método de obtención o de acceso) o al cambio de valores de un objeto (método de mutación o de cambio) se utiliza generalmente el nombre como "get"/"set" más el atributo que acceden. Por ejemplo, existe el método **getCodigo** en la clase *Producto* que entrega el total de la venta. Sin embargo, no se requiere especificar en el DCD todos los métodos accesadores o mutadores, ya que implicaría tener una lista muy grande de métodos que son irrelevantes para el diseño, es por eso que solo aparecen aquellos que son "relevantes" o que generan algún valor para el diseño.

- ✓ La sintaxis que se utilice en los DCD debería corresponder a la sintaxis básica de UML y no a alguna en particular del lenguaje de implementación. Idealmente la traducción se debe realizar cuando se implemente el código que representa el diseño deseado. Sin embargo, en ninguna parte UML niega el uso de otra sintaxis en sus diagramas, por lo que queda a criterio del diseñador finalmente.

AGREGANDO INFORMACIÓN DE LOS TIPOS DE DATOS

La información de tipos de datos en los DCD es relevante, ya que agrega información de bajo nivel para los desarrolladores o para las herramientas que generan el código en forma automática. Sin embargo, la decisión de agregar los tipos de datos y los parámetros a los métodos pasa por una definición del diseñador considerando que:

- ✓ Si se está utilizando una herramienta CASE que genera código en forma automática, el detalle de los tipos debe ser exhaustivo y concordante con el lenguaje que se utilizará en la implementación.

- ✓ Si el DCD es solo para que lo vean los desarrolladores, agregar muchos detalles de tipos y parámetro podría complicar el modelo más que aclararlo porque se convierte en mucho ruido para ellos. En este caso la recomendación es utilizar tipos de datos lógicos que representen la información a guardar (por ejemplo, Texto, Número, CódigoDeBarras, etc).

Sin embargo, tomar la decisión en si agregar o no el detalle debe ser considerado la audiencia que deberá utilizar este modelo, ya que si para la audiencia son obvios algunos de los tipos y parámetros del DCD, es posible omitirlos.

ASOCIACIONES Y NAVEGABILIDAD EN LOS DCD

Como es de esperar, el DCD no está completo si no asociamos las clases entre ellas. Estas asociaciones son similares y contrastadas con las que generamos en el Modelo Conceptual, ya que responden al mismo principio con las que se generaron esas. De esta forma, antes de poner una asociación, es necesario preguntarse "*¿qué asociaciones se requieren para satisfacer la visibilidad y las necesidades de memoria actuales que se indican en los diagramas de interacción?*". Esta visibilidad requerida es necesaria en los siguientes casos comunes:

- ✓ Cuando A envía un mensaje a B.
- ✓ Cuando A crea una instancia de B.
- ✓ Cuando A necesita mantener una conexión a B.

Una vez que se ha encontrado la asociación, es necesario además darle el concepto de navegabilidad. Con esto, se le da un nivel de dependencia a la asociación, lo que se traduce en visibilidad de atributo:

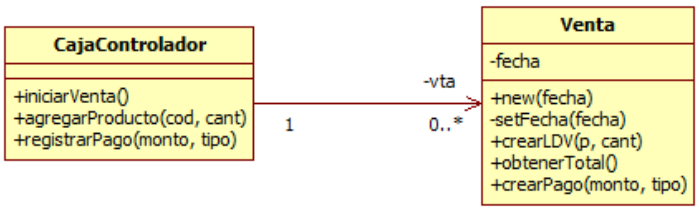


ILUSTRACIÓN 9. AGREGANDO ASOCIACIONES CON NAVEGABILIDAD

De esta forma, *CajaControlador* tiene una visibilidad de atributo de *Venta*, lo que se traduce que la clase *CajaControlador* posea un atributo de tipo *Venta*. Pero ¿dónde queda ese atributo en el DCD?. Es posible que opcionalmente se ponga ese atributo sobre la flecha hacia el lado del tipo de destino (*Venta*), pero no es absolutamente necesario ya que se puede nombrar posteriormente en la implementación.

En nuestro ejemplo de la caja del supermercado, el DCD se convierte en:

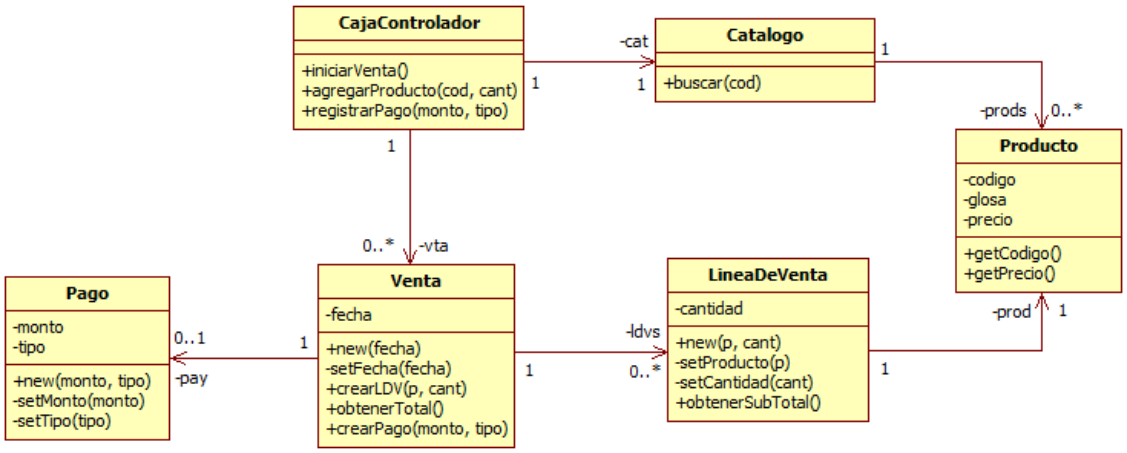


ILUSTRACIÓN 10. DIAGRAMA DE CLASES DE DISEÑO PARCIAL PARA EL SPDV[2]

Como ven en el ejemplo, el DCD es bastante simple de ver, sin embargo se ha construido a partir de los diagramas de interacción de las realizaciones. No se incluyeron los tipos de datos ni parámetros de métodos, porque la audiencia es académica y no nos entrega un valor agregado más que confundirnos en el modelo de diseño.

Nótese además que el DCD que es mucho menos extenso que el modelo conceptual, porque se basa netamente en el diseño del sistema que responde a los eventos del mismo.

RELACIONES DE DEPENDENCIA ADICIONALES

Así mismo como la navegabilidad representa la visibilidad de atributo, debemos de alguna forma representar la visibilidad de parámetro, local o global. Esto se hace con flechas punteadas que indican cuáles son las clases que requieren esa relación. De esta forma, nuestro diagrama ejemplo pasa a conformarse de la siguiente forma:

[2] Se han omitido los tipos de datos en el diagrama de manera totalmente intencional, para la visión completa de éste en el documento.

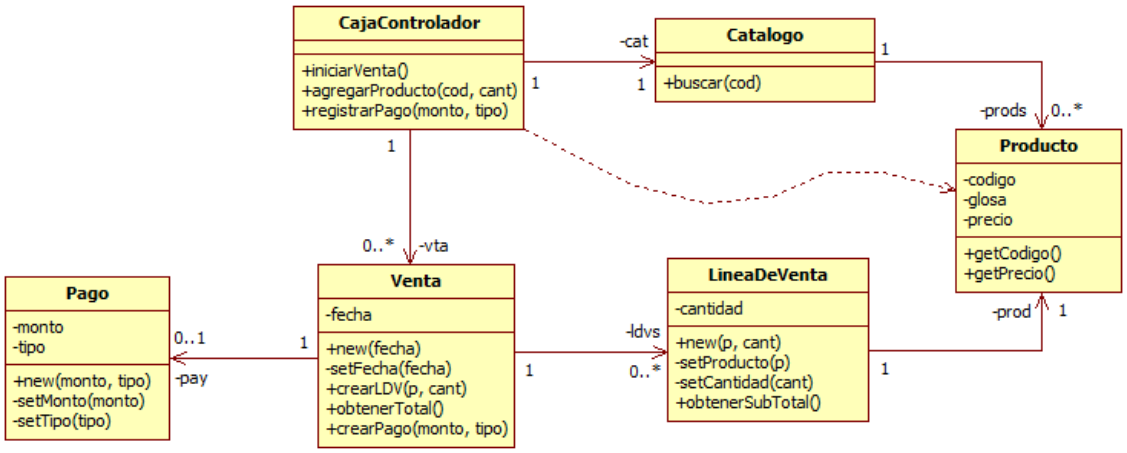


ILUSTRACIÓN 11. DIAGRAMA DE CLASES DE DISEÑO REFINADO PARA EL SPDV

En el ejemplo, aparece una visibilidad de parámetro entre *Venta* y *Producto*, ya que el método **crearLineaDeVenta** requiere una instancia de *Producto* para ser asociada en *LineaDeVenta* (en donde si está la visibilidad de atributo explícita en el DCD).

VISIBILIDAD DE ATRIBUTOS Y MÉTODOS

Hasta ahora todos los diagramas que hemos visto muestran unos pequeños signos en su definición. Estos signos no son adornos ni tampoco es casualidad que ahí estén, sino que representan la visibilidad que tienen los atributos y la visibilidad que tienen los métodos en el modelo.

UML no obliga a utilizar esto modificadores, pero nos daremos cuenta que algunos CASE los ponen por defecto como "-" a los atributos y "+" a los métodos. ¿Por qué es esto?

- ✓ El modificador "-" indica que el atributo o método es privado, esto quiere decir que el alcance que tiene es solo dentro de la clase en la cual está definido.
- ✓ El modificador "+" indica que el atributo o método es público, lo que significa que el alcance que tiene es dentro y fuera de la clase.

Por defecto en UML, si estos modificadores no se especifican en el DCD, se supondrá que los atributos son privados y los métodos son públicos.

CUERPO DE LOS MÉTODOS

Por último, y opcionalmente, es posible utilizar cuerpos de código en los métodos que especifiquen cómo se implementan con pseudocódigo o algún lenguaje en particular. Sin embargo, debemos recordar que todo el detalle que pongamos en el DCD es para apoyar la etapa de implementación y no debe significar mayor complejidad para los desarrolladores.

Desde el punto de vista de notación, el cuerpo de los métodos va sin la firma del método (encabezado) y solo se describe el contenido, de la siguiente forma:

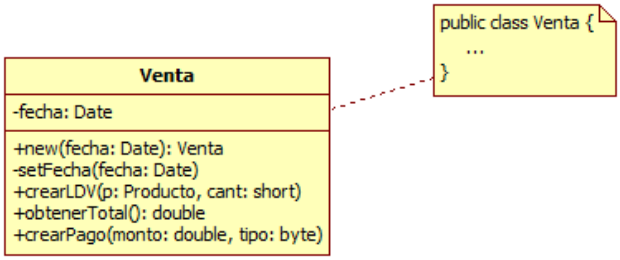


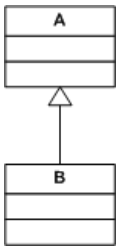
ILUSTRACIÓN 12. REPRESENTACIÓN EN LA CODIFICACIÓN DE LA CLASE

TEMAS COMPLEMENTARIOS

GENERALIZACIÓN

La generalización se refiere a la definición de subtipos a raíz de una definición similar respecto a diferentes objetos. Esta definición se puede utilizar a partir del modelo conceptual, sin embargo tampoco es tarde utilizarlo en el diseño.

Para la notación se agrega la utilización de un nuevo tipo de asociación.



La clase B es un subtipo de la clase A cuando ésta última posee características comunes, pero la primera posee características y/u operaciones que le permiten realizar acciones particulares para dicha clase. En ese sentido, la definición de la clase B contiene el 100% de las características y operaciones de A, pero no al revés.

Por ejemplo, si definimos los tipos de pago en el punto de venta, podríamos decir que existe una "jerarquía de tipos" respecto a las clases de pagos que se pueden utilizar (PagoEnEfectivo, PagoConTarjeta, PagoConCheque). De esta forma el modelo quedaría definido como:

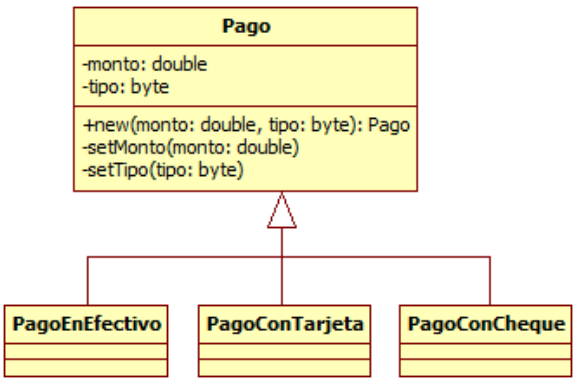


ILUSTRACIÓN 13. GENERALIZACIÓN DE LOS TIPOS DE PAGO EN EL SPDV

Existe una regla simple para definir si existe una razón para definir un subtipo.

- ✓ El subtipo tiene algún atributo particular de interés.
- ✓ El subtipo tiene alguna operación particular de interés.
- ✓ El subtipo se opera con un concepto diferente para el subtipo, en relación a lo relevante de éste.
- ✓ El subtipo representa alguna entidad física que se comporta en manera diferente al supertipo.

AGREGACIÓN

La agregación se utiliza cuando existe una dependencia directa entre dos clases, de manera de que la clase dependiente no puede existir sin su generadora.

Para la notación se utiliza un rombo pintado indicando la clase generadora:

La clase A agrega uno o más objetos de B.

Es importante definir la agregación cuando ésta corresponda. Por ejemplo en el caso del Punto de Venta podemos encontrar agregación en algunas clases del DCD como Venta y LineaDeVenta, ya que no pueden existir referencias a una LineaDeVenta si no existe la Venta primero. Es por eso que la relación queda así:

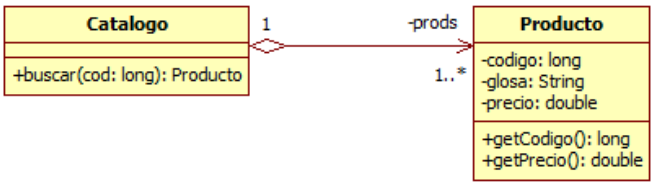


ILUSTRACIÓN 14. AGREGACIÓN DE PRODUCTOS EN EL SPDV

COMPOSICIÓN

La composición se utiliza cuando no existe una dependencia directa entre dos clases, pero si se requiere tener una referencia de la otra.

Para la notación se utiliza un rombo en blanco indicando la clase que necesita tener relación:

La clase A compone uno o más objetos de B.

Al igual como lo hicimos con la agregación, la definición se debe realizar correctamente solo cuando una de ellas requiere tener una referencia de la otra en su definición (atributos). Por ejemplo en el caso del Punto de Venta podemos encontrar composición en el DCD entre Catalogo y Producto, ya que los productos de la tienda siempre deben existir por si solos, pero el catálogo es solamente el medio para que puedan acceder a l. Es por eso que la relación queda así:

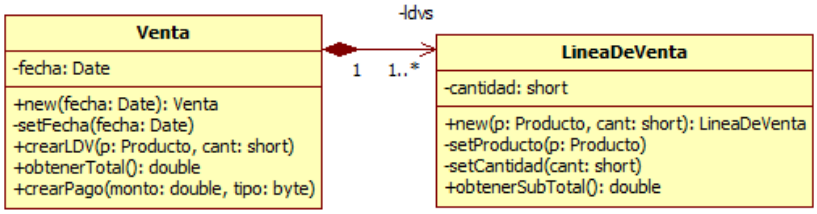


ILUSTRACIÓN 15. COMPOSICIÓN DE LÍNEAS DE VENTA EN EL SPDV