EL MODELO DE DISEÑO

Extraído de:
UML y Patrones. 2ª Edición.
Craig Larman.
Prentice Hall. 2003

1. Introducción

En este documento se estudia el diseño de objetos. El lenguaje utilizado para ilustrar los diseños es, principalmente, los diagramas de interacción. UML incluye los diagramas de interacción para ilustrar el modo en el que los objetos interaccionan por medio de mensajes. Este documento introduce fundamentalmente la notación, y comenzará a mostrar su uso básico. Sin embargo, para crear buenos diseños, también se deben entender los principios de diseño. Esto es algo que se verá más adelante en la asignatura.

Nótese que es en este momento cuando se requiere la aplicación de las técnicas de diseño. La creación de los casos de uso, modelos del dominio, y otros artefactos, es más sencilla que la asignación de responsabilidades a objetos y la creación de diagramas de interacción bien diseñados. Esto es debido a que existen un gran número de principios de diseño sutiles y "grados de libertad" que subyacen a un diagrama de interacción bien diseñado, que a la mayoría de los otros artefactos orientados a objetos.

2. Diagramas de Interacción

EL término diagrama de interacción es una generalización de dos tipos de diagramas UML más especializados; ambos pueden utilizarse para representar de forma similar interacciones de mensajes:

- *Diagramas de colaboración*. Ilustran las interacciones entre objetos en un formato de grafo o red, en el cual los objetos se pueden colocar en cualquier lugar del diagrama, como refleja la Figura 1.

- *Diagramas de secuencia*. Ilustran las interacciones en un tipo de formato con aspecto de una valla, en el que cada objeto nuevo se añade a la derecha, como se puede observar en la Figura 2.

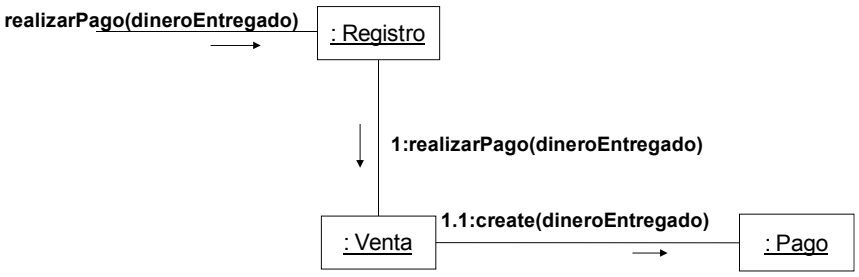


Figura 1. Ejemplo de diagrama de colaboración.

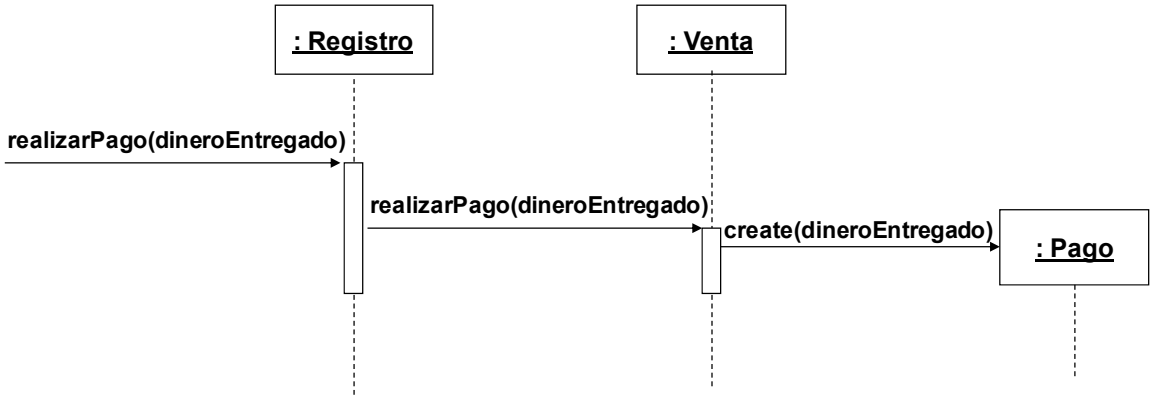


Figura 2. Ejemplo de diagrama de secuencia.

Tabla 1. Puntos fuertes y débiles de cada representación.

Tipo	*Puntos fuertes*	*Puntos débifes*
secuencia	• muestra claramente la secuencia y ordenación en el tiempo de los mensajes • notación simple	• fuerza a extender por la derecha cuando se añaden nuevos objetos • consume espacio horizontal

Tipo	*Puntos fuertes*	*Puntos débifes*
colaboración	• economiza espacio, flexibilidad al añadir nuevos objetos en dos dimensiones • es mejor para ilustrar bifurcaciones complejas, iteraciones y comportamiento concurrente	• difícil ver la secuencia de mensajes • notación más compleja

2.1. Notación General de los Diagramas de Interacción

2.1.1. Representación de Clases e Instancias

UML ha adoptado un enfoque simple y consistente para representar las instancias frente a los clasificadores (ver Figura 3). Para cualquier tipo de elemento UML (clase, actor,.. .), una instancia utiliza el mismo símbolo gráfico que el tipo, pero con la cadena de texto que lo designa subrayada.

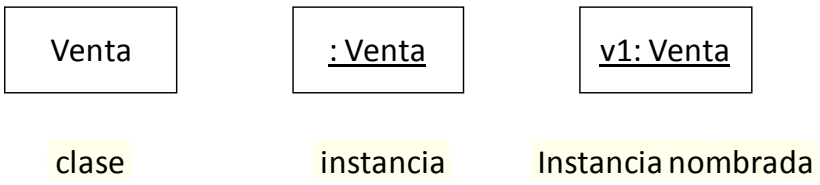


Figura 3. Clases e instancias.

En consecuencia, para mostrar una instancia de una clase en un diagrama de interacción, se utiliza el símbolo gráfico para una clase, esto es el rectángulo, pero con el nombre subrayado. Se puede utilizar un nombre para identificar de manera única la instancia. Si no se utiliza, obsérvese que los ":" preceden al nombre de la clase.

2.1.2. Sintaxis de la Expresión de Mensaje Básica

UML cuenta con una sintaxis básica para las expresiones de los mensajes:

```
return := mensaje(parametro: tipaParametro): tipoRetorno
```

Podría excluirse la información del tipo si es obvia o no es importante. Por ejemplo:

```
espec := getEspecProducto(id)

espec : = getEspecProducto (id :ArticuloID)

espec:= getEspecProducto(id:ArticuloID): EspecificacionDelProducto
```

2.2. Notación Básica de los Diagramas de Secuencia

2.2.1. Mensajes

Cada mensaje entre objetos se representa con una expresión de mensaje sobre una línea con punta de flecha entre los objetos (ver Figura 4). El orden en el tiempo se organiza de arriba a abajo.

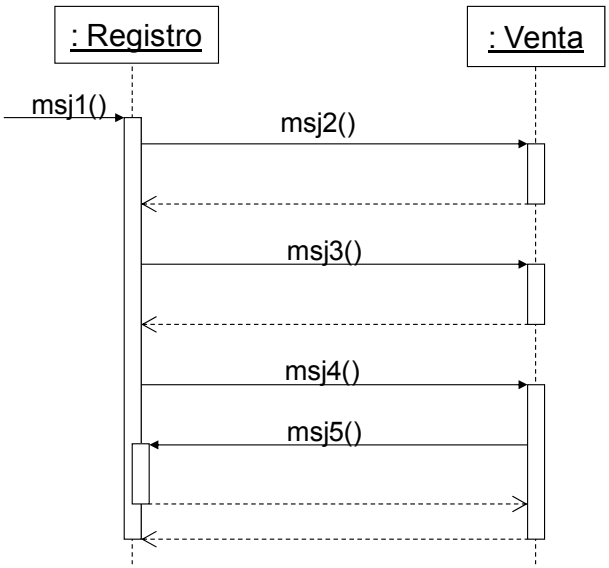


Figura 4. Mensajes y focos de control con cajas de activación.

2.2.2. Focos de Control y Cajas de Activación

Como se ilustra en la Figura 4, los diagramas de secuencia pueden también mostrar los focos de control (esto es, en una llamada de rutina ordinaria, la operación se encuentra en la pila de llamadas) utilizando una caja de activación. La caja es opcional, pero se suele utilizar habitualmente.

2.2.3. Representación de Retornos

Un diagrama de secuencia podría, opcionalmente, mostrar el retorno de un mensaje mediante una línea punteada con la punta de flecha abierta, al final de una caja de activación (ver Figura 4). La línea de retorno se suele anotar para describir lo que se está devolviendo (si es el caso) a partir del mensaje.

2.2.4. Mensajes a "self" o "this"

Se puede representar un mensaje que se envía de un objeto a él mismo utilizando una caja de activación anidada (ver Figura 5).

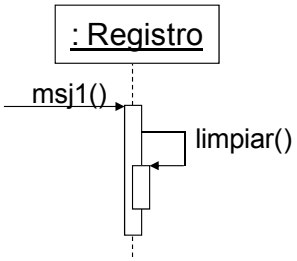


Figura 5. Mensajes a "this".

2.2.5. Creación de Instancias

La notación para la creación de instancias se muestra en la Figura 6.

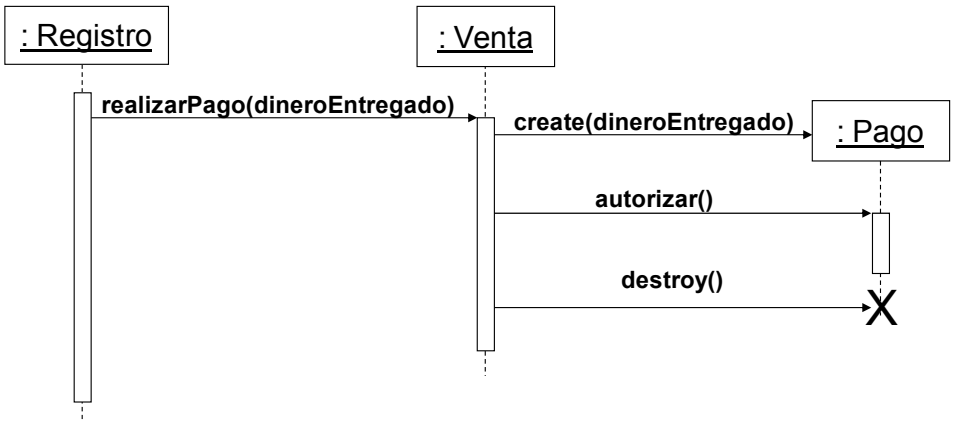


Figura 6. Creación de instancias, línea de vida y destrucción de los objetos.

2.2.6. Línea de vida del Objeto y Destrucción de Objetos

La Figura 6 también ilustra las líneas de vida de los objetos (las líneas punteadas verticales bajo los objetos). Éstas indican la duración de la vida de los objetos en el diagrama.

En algunas circunstancias es deseable mostrar la destrucción explícita de un objeto (como en C+ +, que no tiene recogida de basura); la notación UML para las líneas de vida proporcionan una forma para expresar esta destrucción (ver Figura 6). El mensaje estereotipado <<destroy>>, con la gran X y la línea de vida cortada, indican la destrucción explícita del objeto.

2.2.7. Mensajes Condicionales y Condicionales Mutuamente Exclusivos

El primer mensaje que envía el objeto A al objeto B en la Figura 7 muestra un mensaje condicional.

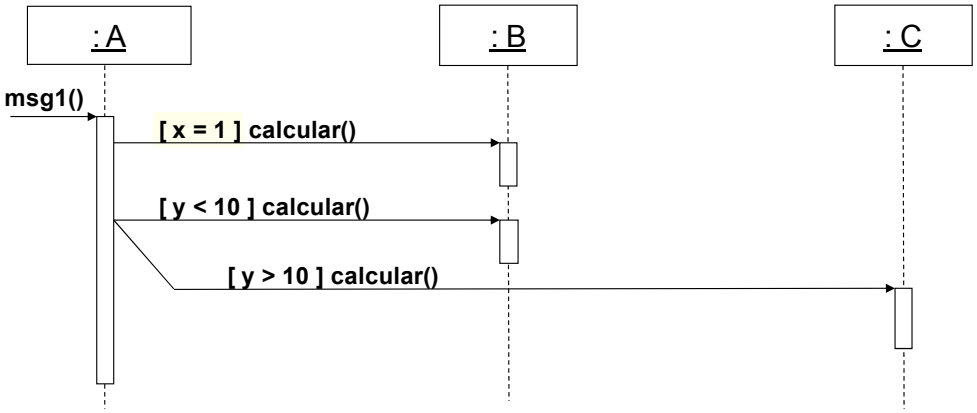


Figura 7. Mensaje condicional.

La notación para los mensajes condicionales mutuamente exclusivos es un tipo de línea de mensaje con forma de ángulo que nace desde un mismo punto, como también ilustra la Figura 7.

2.2.8. Iteración para un Único Mensaje

La notación para la iteración de un mensaje se muestra en la Figura 8.

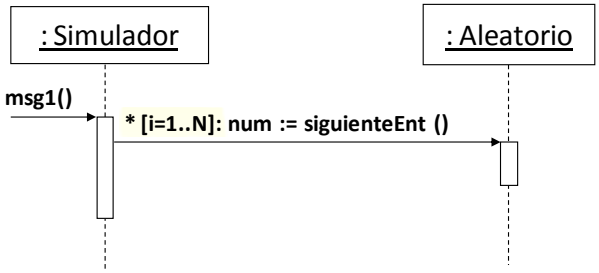


Figura 8. Iteración para un mensaje.

2.2.9. Iteración de Una Serie de Mensajes

La Figura 9 muestra la notación para indicar la iteración alrededor de una serie mensajes.

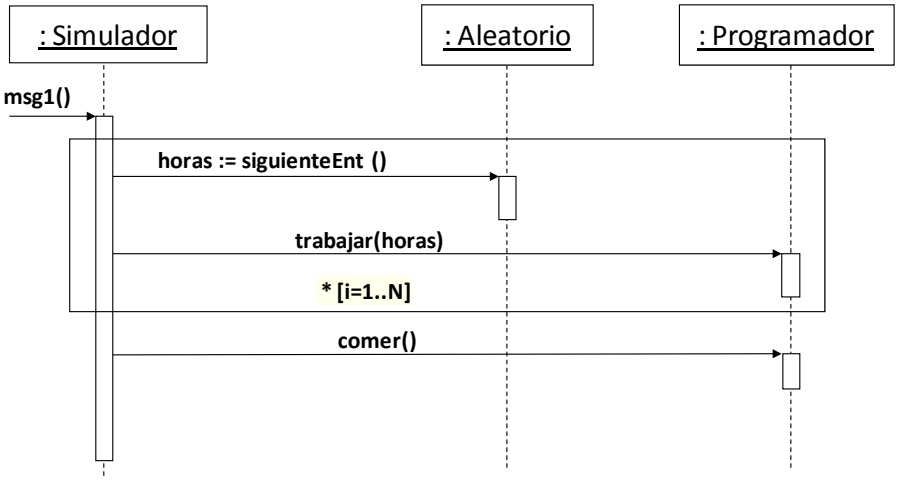


Figura 9. Iteración sobre una secuencia de mensajes.

2.2.10. Iteración sobre Una Colección (Multiobjeto)

La iteración sobre una colección en un diagrama de secuencia se muestra en la Figura 10.

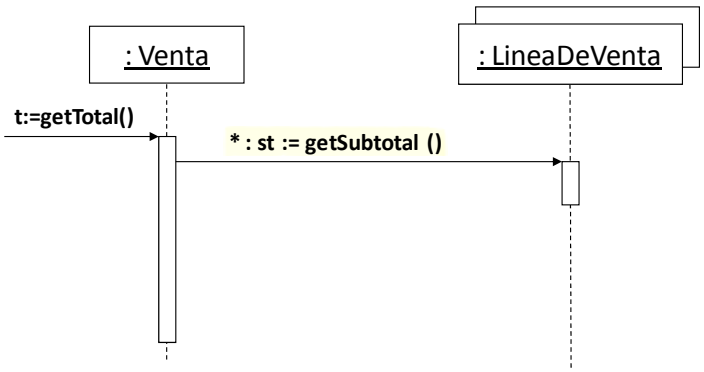


Figura 10. Iteración sobre un multiobjeto.

Con los diagramas de colaboración de UML, se especifica cómo indicar el envío de un mensaje a cada elemento, en lugar de repetidamente a la propia colección. Sin embargo, no lo hace con los diagramas de secuencia.

2.2.11. Mensajes a Objetos de Clase

Las llamadas a los métodos de clase o estáticos se representan no subrayando el nombre del clasificador, lo que significa que se trata de una clase en lugar de una instancia (ver Figura 11).

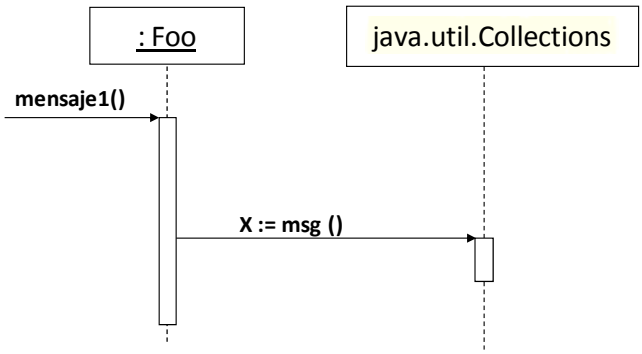


Figura 11. Invocación a un método de clase o estático.

3. Diagramas de Clases de Diseño

Una vez terminados los diagramas de interacción para las realizaciones de los casos de uso para una iteración en una aplicación, es posible identificar la especificación de las clases software (e interfaces) que participan en la solución software, y añadirles detalles de diseño, como los métodos. UML proporciona la notación para representar los detalles de diseño en los diagramas de clase. En esta sección se estudiará dicha notación.

Un diagrama de clases de diseño (DCD) representa las especificaciones de las especificaciones de las clases e interfaces software en una aplicación. Entre la información general encontramos:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Información acerca del tipo de los atributos. .
- Navegabilidad.
- Dependencias.

A diferencia de las clases conceptuales del Modelo del Dominio, las clases de diseño de los DCD muestran las definiciones de 1as clases software en lugar de los conceptos del mundo real.

El DCD de la Figura 12 muestra un ejemplo de DCD.

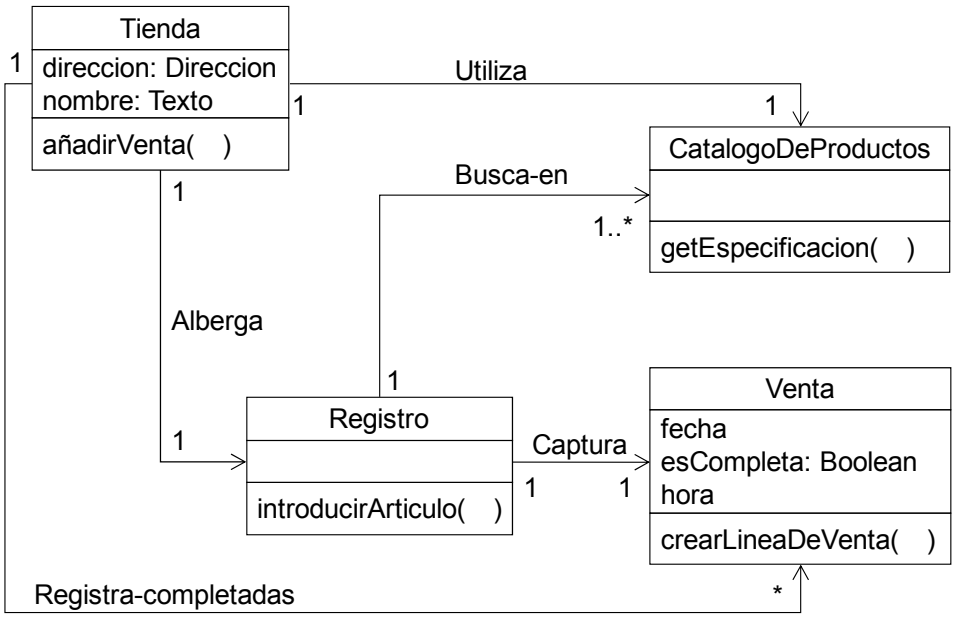


Figura 12. Ejemplo de diagrama de clases de diseño.

Además de las asociaciones y atributos básicos, el diagrama se amplía para representar, por ejemplo, los métodos de cada clase, información del tipo de los atributos y navegación entre los objetos.

El UP no define de manera específica ningún artefacto denominado "diagrama de clases de diseño". El UP define e1 Mode1o de Diseño, que contiene varios tipos de diagramas, que incluye los diagramas de interacción, de paquetes, y los de clases. Los diagramas de clases del Modelo de Diseño del UP contienen "clases de diseño" en términos del UP. De ahí que sea común hablar de "diagramas de clases de diseño", que es más corto que, "diagramas de clases en el Modelo de Diseño".

Como ya se ha mencionado anteriormente, en el Modelo de Dominio del UP, una Venta no representa una definición software, sino que se trata de una abstracción de un concepto del mundo real acerca del cual estamos interesados en realizar una declaración. En cambio, los DCD expresan -para la aplicación software- la definición de las clases como componentes software. En estos diagramas, una *Venta* representa una clase software.

3.1. Cuándo Crear los DCD

Aunque esta presentación de los DCD viene después de la creación de los diagramas de interacción, en la práctica podrían crearse en paralelo. Al comienzo del diseño se podrían esbozar muchas clases, nombres de métodos y relaciones para asignar responsabilidades, antes de la elaboración de los diagramas de interacción. Es posible y deseable elaborar algo de los diagramas de interacción, actualizar entonces los DCD, después extender los diagramas de interacción algo más, y así sucesivamente.

3.2. Creación del Diagrama de Clases de Diseño

3.2.1. Identificación y Representación de las Clases software

El primer paso en la creación de.los DCD como parte del modelo de la solución es identificar aquellas clases que participan en la solución software. Se pueden encontrar examinando todos los diagramas de interacción y listando las clases que se mencionan.

Para la aplicación del PDV son;

Registro	Venta
CatalogoDeProductos	EspecificacionDelProducto
Tienda	LineaDeVenta
Pago	

El siguiente paso es dibujar un diagrama de clases para estas clases e incluir los atributos que se identificaron previamente en el Modelo del Dominio que también se utilizan en el diseño (ver Figura 13).

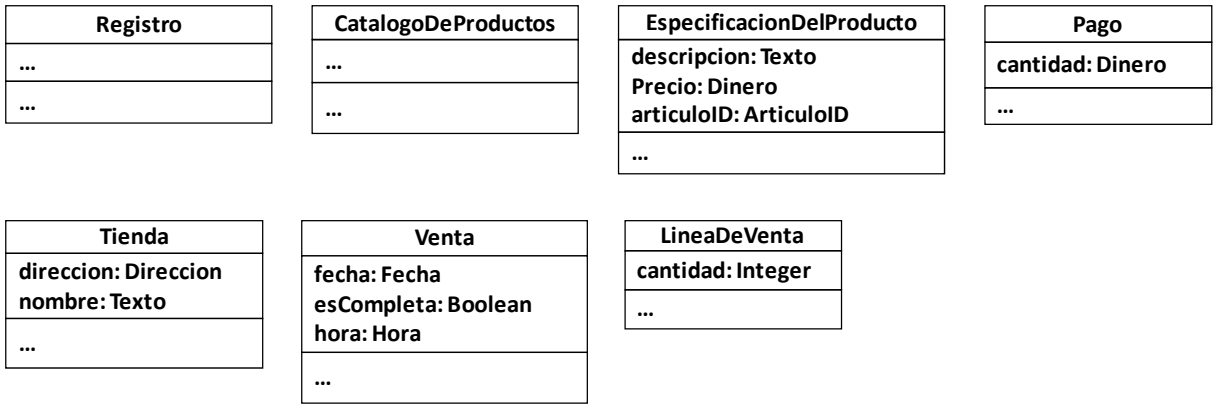


Figura 12. Clases software de la aplicación.

Nótese que algunos de los conceptos del Modelo del Dominio, como Cajero, no están presentes en el diseño. No es necesario -para la iteración actual- representarlos en el software. Sin embargo, en iteraciones posteriores, cuando se aborden nuevos requisitos y casos de uso, podrían formar parte del diseño. Por ejemplo, cuando se írnplementen los requisitos de seguridad y de inicio de sesión, es probable que sea relevante una clase software denominada *Cajero*.

3.2.2. Añadir los Nombres de los Métodos

Se pueden identificar los nombres de los métodos analizando los diagramas de interacción. Por ejemplo, si se envía el mensaje *crearLineaDeVenta* a una instancia de la Venta, entonces la clase Venta debe definir un método *crearLineaDeVenta*.

En general, el conjunto de todos los mensajes enviados a una clase X a lo larga de dos los diagramas de interacción indican la mayoría de los métodos que debe definir la clase X.

La inspección de todos los diagramas de interacción de la aplicación del PDV daría lugar a la asignación de métodos que se muestra en la Figura 13.

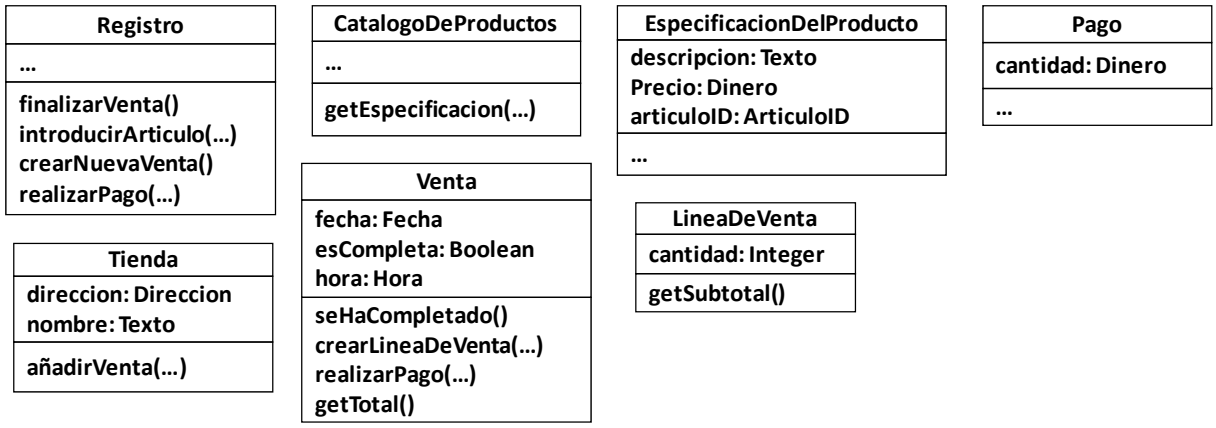


Figura 13. Métodos de la aplicación.

Se deben tener en cuenta las siguientes cuestiones especiales en relación con los nombres de los métodos:

- Interpretación del mensaje create.
- Descripción de los métodos de acceso.
- Interpretación de los mensajes a los multiobjetos.
- Sintaxis dependiente del lenguaje.

El mensaje *create* es una forma independiente del lenguaje, posible en UML para indicar instanciación e inicialización. Al traducir el diseño a un lenguaje de programación orientado a objetos, se debe expresar en función de sus estilos para la instanciación e inicialización. No existe un método *create* ni en C++, ni en Java, ni en Smalltalk. Por ejemplo, en C+ +, implica la asignación automática, o asignación de almacenamiento libre con el operador new, seguido de una llamada al constructor. En lava, implica la invocación del operador new, seguido de una llamada al constructor. Debido a las múltiples interpretaciones, y también debido a que la inicialización es una actividad muy común, es habitual omitir en el DCD los métodos relacionados con la creación y los constructores.

Los **métodos de acceso** recuperan (método de obtención, *accessor*) o establecen (método de cambio, *mutador*) el valor de los atributos. En algunos lenguajes (como Java) es común tener un *accessor* y un *mutator* para cada atributo, y declarar todos los atributos como privados (para imponer la encapsulación de datos). Normalmente, incluye la descripción de estos métodos en el diagrama de clases debido a que generan mucho ruido; para n atributos, hay 2n métodos sin interés. Por ejemplo, no se muestral aunque está presente, el método *getPrecio* (o *precio*) de la *EspecificacionDelProducto* porque se trata simplemente de un método de acceso.

Un mensaje a un **multiobjeto** se interpreta como un mensaje al propio objeto conlenedor/colección. Por ejemplo, el siguiente mensaje al multiobjeto *buscar* se debe interpretar como un mensaje al objeto contenedor/colección, como a un *Map* en lava, un *map* en C++, o un *Dictionary* en Smalltalk (ver Figura 14).

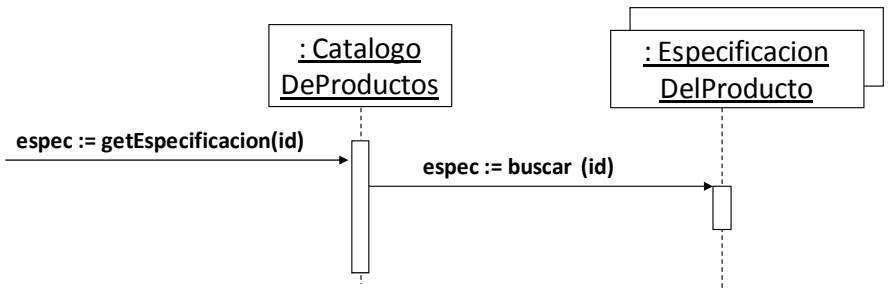


Figura 14. Mensaje a un multiobjeto.

Por tanto, el método *buscar* no forma parte de la clase *EspecificacionDelProducto*, sino de la interfaz del multiobjeto. En consecuencia, no es correcto añadir el método *buscar* a la clase *EspecificacionDelProducto*.

Normalmente, estas interfaces o clases de contenedores/colecciones son elementos de las librerías predefinidas, y no es útil mostrar explícitamente estas clases en el DCD, porque añaden ruido, pero poca información nueva.

Algunos lenguajes, como Smalltalk, tienen una **sintaxis** que es muy diferente del formato UML básico de nombreMetodo(listaParametros). Se recomienda que se utilice el formato UML básico, incluso si el lenguaje de implementación que se planea utilizar una sintaxis diferente. Idealmente, la traducción debería tener lugar en el momento de la generación de código, en lugar de durante la creación de los diagramas de clase.

3.2.3. Añadir Más Información sobre los Tipos

Opcionalmente, todos los tipos de los atributos, parámetros de los métodos y los valores de retorno se podrían mostrar. La cuestión sobre si se muestra o no esta información se debe considerar en el siguiente contexto:

Se debería crear un DCD teniendo en cuenta los destinatarios.

- Si se está creando en una herramienta CASE con generación automática de código, son necesarios todos los detalles y de modo exhaustivo.

- Si se está creando para que lo lean los desarrolladores de software, los detalles exhaustivos de bajo nivel podrían afectar negativamente por el nivel de ruido.

Por ejemplo, ¿es necesario mostrar todos los parámetros y la información de sus tipos? Depende de lo obvia que sea la información para la audiencia a la que está destinada.

El diagrama de clases de diseño de la Figura 15 muestra más información sobre los tipos.

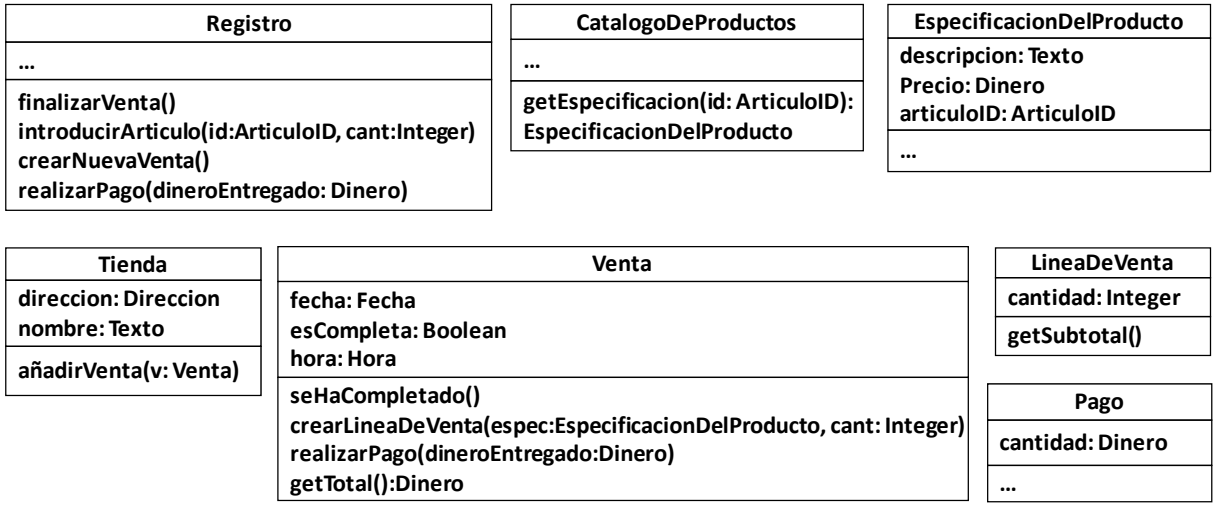


Figura 15. Añadir información acerca de los tipos.

3.2.4. Añadir Asociaciones y Navegabilidad

Cada extremo de asociación se denomina rol, y en los DCD el rol podría se completa con una flecha de navegabilidad. La navegabilidad es una propiedad del rol que indica que es posible navegar unidireccionalmente a través de la asociación desde los objetos de la clase origen a la clase destino. La navegabilidad implica visibilidad (ver Figura 16).

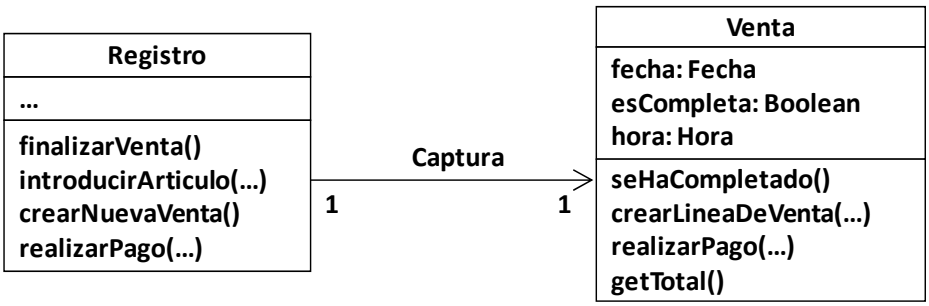


Figura 16. Representación de la navegabilidad.

La interpretación habitual de una asociación con una flecha de visibilidad es la visibilidad de atributo desde la clase origen hasta la clase destino. Durante la implementación en un lenguaje orientado a objetos por lo general se transforma en un atributo de la clase origen que hace referencia a una instancia de la clase destino. Por ejemplo, clase *Registro* definirá un atributo que referencia a una instancia de *Venta*.

Todas, las asociaciones en los DCD deben completarse con flechas de navegabilidad necesarias.

En un DCD, se eligen las asociaciones de acuerdo a un criterio necesito-conocer orientado al software estricto -¿qué asociaciones se requieren para satisfacer la visibilidad y las necesidades de memoria actuales que se indican en los diagramas de interacción?-. Esto contrasta con las asociaciones en el Modelo del Dominio, que se podrían identificar por la intención de mejorar la comprensión del dominio del problema. Una vez más, vemos que existe una diferencia entre los objetivos del Modelo del Diseño y del Modelo del Dominio: uno es analítico, el otro una descripción de los componentes software.

La visibilidad y las asociaciones requeridas entre las clases se dan a conocer mediante los diagramas de interacción. A continuación, se presentan algunas situaciones comunes que sugieren la necesidad de definir una asociación completada con visibilidad de A a B:

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

Por ejemplo, a partir del diagrama de interacción de la Figura 17 que comienza con el mensaje *create* a la *Tienda*, y a partir del contexto más amplio de los otros diagramas de interacción, se puede distinguir que probablemente la *Tienda* debería tener una conexión permanente con las instancias de *Registro* y el *CatalogoDeProductos* que crea. También es razonable que el *CatalogoDeProductos* necesite una conexión permanente con la colección de objetos *EspecijicacionDelProducto* que crea. De hecho, muy a menudo el creador de otro objeto requiere una conexión permanente con él. Por tanto, las conexiones implícitas se presentarán como asociaciones en el diagrama de clases.

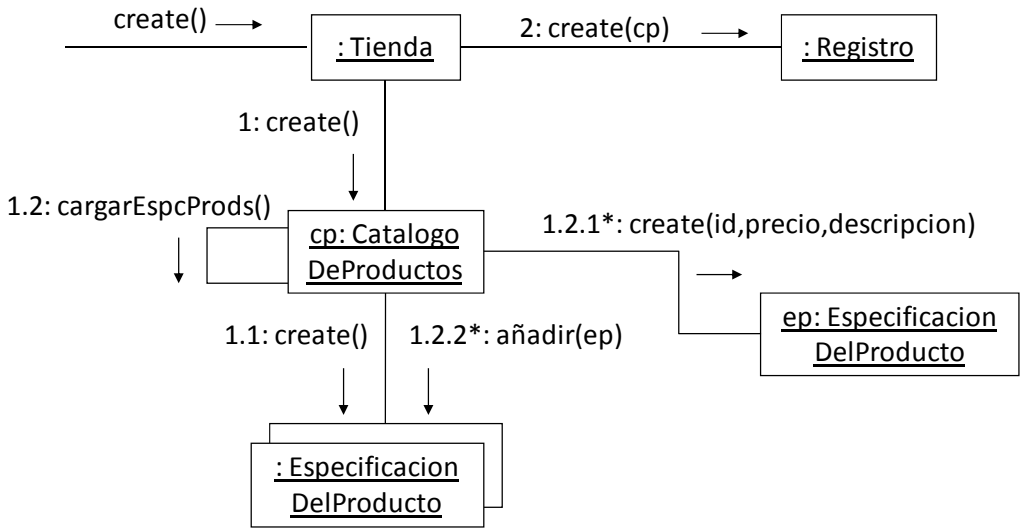


Figura 17. La navegabilidad se identifica a partir de los diagramas de interacción.

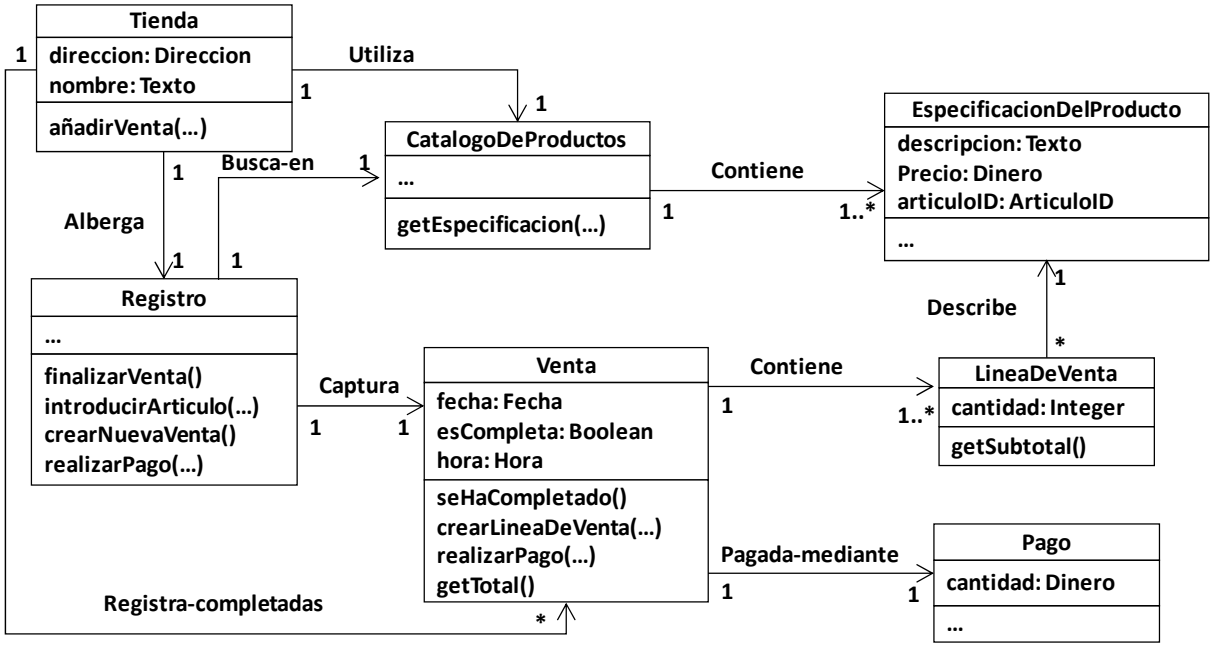


Figura 18. Asociaciones completadas con navegabilidad.

Basado en el criterio anterior para las asociaciones y la navegabilidad, el análisis de todos los diagramas de interacción generados para la aplicación del PDV NuevaEra dará lugar a un diagrama de clases (ver Figura 18) con las siguientes asociaciones (se oculta la información exhaustiva sobre los tipos por claridad).

Nótese que éste no es exactamente el mismo conjunto de asociaciones que se generó para el Modelo del Dominio. Por ejemplo, en el Modelo del Dominio no existía la asociación *Busca-en* entre el *Regístro* y el *CatalogoDeProductos* -no se pensó que fuera una relación importante y permanente en ese momento-. Pero durante la creación de los diagramas de interacción, se decidió que el objeto software *Registro* debería tener una conexión permanente con el *CatalogoDeProductos* software para buscar los objetos *EspecificacionDelProducto*.

3.3. Visibilidad por Defecto en UML

Si no se muestra explícitamente ningún marcador de visibilidad para un atributo o método, no hay un valor por defecto. Si no se muestra nada, en UML significa “sin especificar”. Sin embargo, la convención común es asumir que los atributos son privados (el nombre va precedido del símbolo “-“) y los métodos públicos (el nombre va precedido del símbolo “+”), a menos que se indique otra cosa.

Diagramas de Colaboración

Documentación basada en el texto:

UML y Patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado, 2ª edición, Craig Larman, Ed. Prentice-Hall, 2002 (2003 la edición en Español), Capítulo 15 pp. 185-200

Hay dos tipos de Diagramas de Interacción

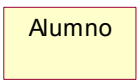
- Diagramas de Colaboración
- Diagramas de Secuencia

Muestran la misma información, es decir, el paso de mensajes entre objetos o interacción entre objetos, pero de distinta forma:

- Diagramas de secuencia: de arriba abajo secuencialmente
- Diagramas de colaboración: en forma de grafo

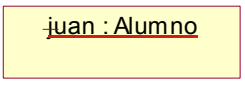
1. Notación Básica

- Clases e instancias: Rectángulos con el nombre dentro.
  - Clase: con mayúscula la primera letra y sin subrayar.



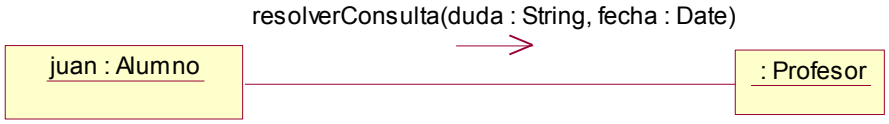
  - Instancia sin nombre: dos puntos y el nombre de la clase, subrayado.

  - Instancia con nombre: nombre de la instancia en minúscula, dos puntos y el nombre de la clase, subrayado.

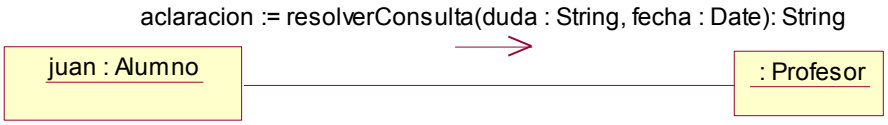


- Enlaces:
  - Una línea que une dos instancias
  - Es una conexión que posibilita el paso de uno o más mensajes.
  - Es una relación de USO entre dos objetos.

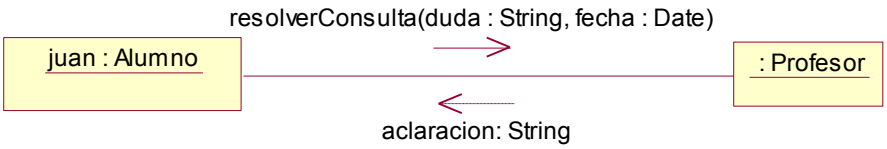
- Mensajes:
  - flecha que indica el sentido del mensaje, con una etiqueta con el nombre del mensaje y sus parámetros entre paréntesis.
  - Se puede mostrar opcionalmente el tipo de un parámetro con dos puntos y el nombre del tipo.



- Valor de retorno:
  - se puede mostrar con el nombre de una variable y una asignación precediendo al nombre del mensaje.
  - Opcionalmente se puede mostrar el tipo del valor devuelto con dos puntos y el nombre del tipo tras los parámetros.

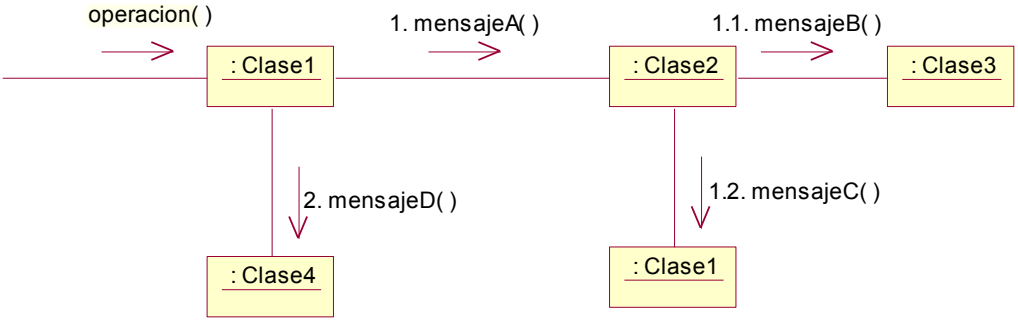


  - También se puede mostrar como otro mensaje específico de retorno

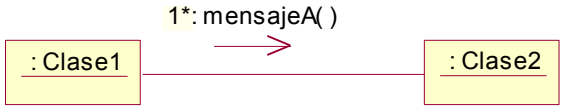


2. Flujo de Control

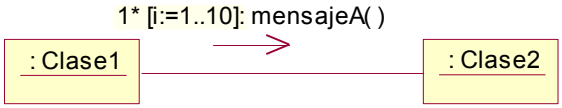
- Secuencia:
  - Cada mensaje va precedido por un Número de Secuencia y dos puntos, excepto el primer mensaje que representa la operación del Sistema.
  - Puede haber mensajes anidados

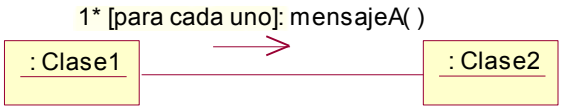


- Iteración:
  - se indica con un * a continuación del número de secuencia del mensaje.

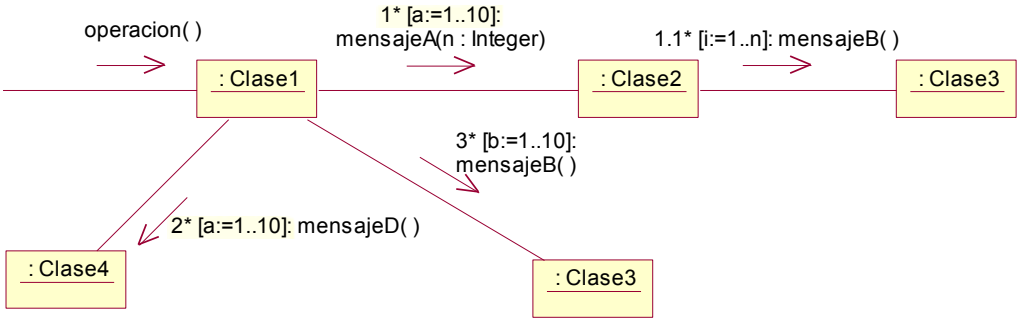


  - Si se quiere indicar un rango posible del número de envíos del mensaje se pone una Cláusula de Iteración

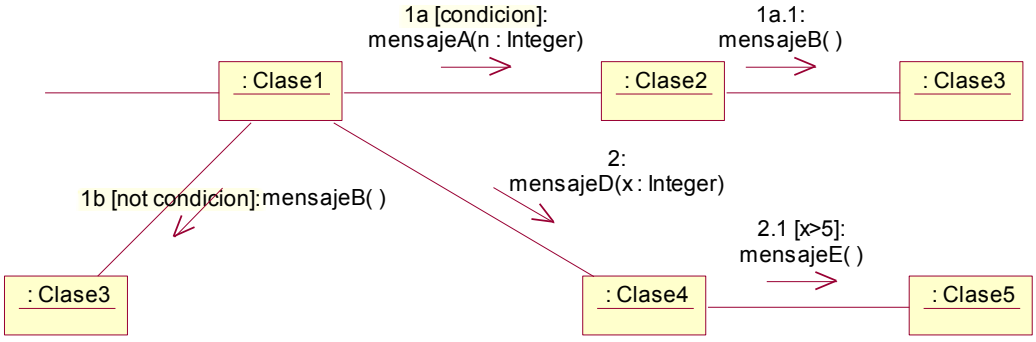


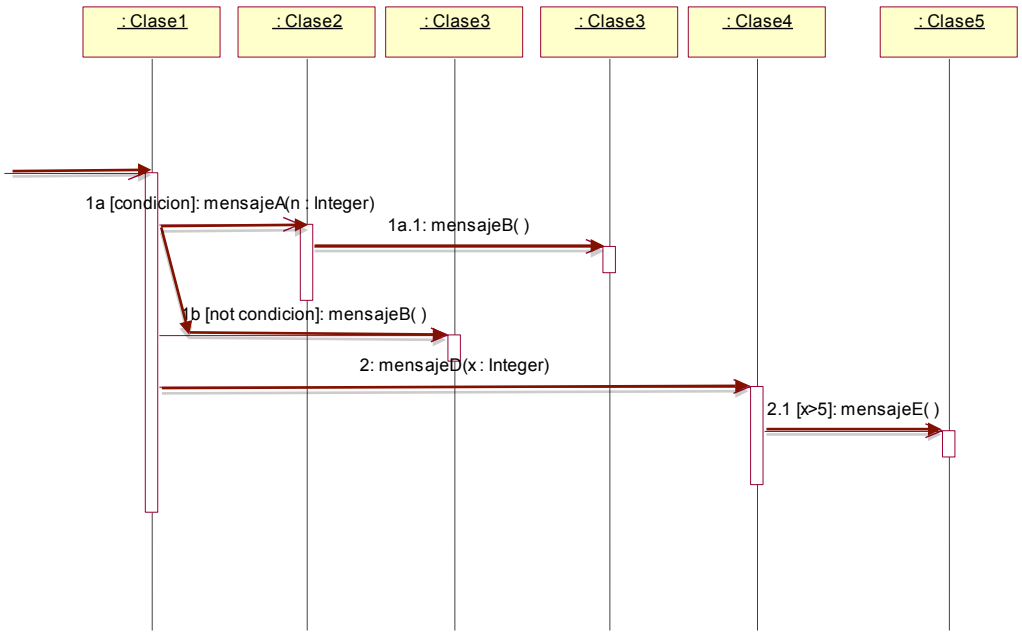


  - Dos mensajes con una misma cláusula de iteración ocurren los dos dentro de una misma iteración

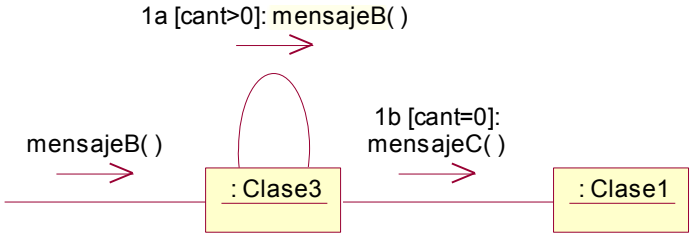


- Condicional:
  - se indica con una Cláusula Condicional
  - Si hay dos caminos mutuamente excluyentes según una condición, uno lleva la cláusula [condicion] y el otro [not condicion], y en el número de secuencia se añade a o b





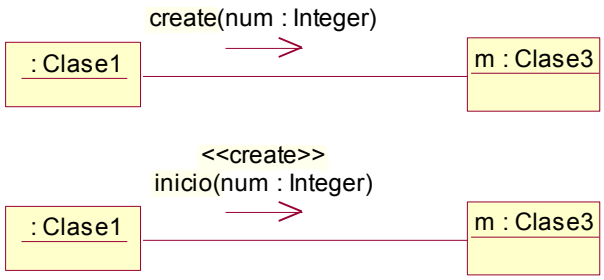
- Recursividad:
  - Un objeto se puede mandar un mensaje a sí mismo (self o this).



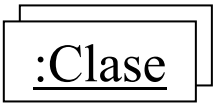
3. Otros Elementos de la Notación

- Creación de instancias:

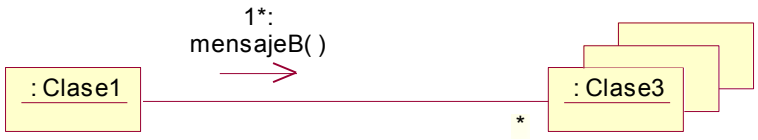
Se muestra enviando un mensaje create () a la instancia que será creada. El mensaje create() es la representación estándar en UML de un método constructor.



- Colecciones de objetos (multiobjetos): se muestra con dos rectángulos superpuestos.



- Cuando se manda un mensaje a un icono colección, se envía al objeto colección en sí mismo, y no se reenvía a todos y cada uno de los objetos de la colección.
- Para indicar el envío del mensaje a cada elemento de la colección:



- A un objeto colección se le puede pedir:
  - que nos diga el número de elementos en la colección,
  - que nos dé un puntero a uno de los objetos,
  - que añada un cierto objeto, etc.

- Envío de mensajes a una Clase – invocación de Métodos de Clase (métodos estáticos)