# Capítulo

# 6

# UML

**Centro Asociado Palma de Mallorca**

**Tutor: Antonio Rivero Cuesta**

# 6 UML - Lenguaje Unificado de Modelado

## 6.1 Introducción.

El UML es un lenguaje universal de modelado de sistemas que se emplea para especificar y modelizar los sistemas software.

## 6.2 Objetivos

Familiarizarse con el lenguaje UML, su utilización para la especificación y modelado.

## 6.3 ¿Qué es UML?

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

Ofrece un estándar para describir un plano del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Presenta información sobre la estructura estática y el comportamiento dinámico de un sistema.

Existen herramientas que pueden ofrecer generadores de código a partir de los diagramas UML.

## 6.4 Orígenes de UML

La notación UML deriva y unifica las tres metodologías de análisis y diseño más extendidas:

- Metodología de Grady Booch.
- Técnica de modelado orientado a objetos.
- Metodología de casos de uso.

## 6.5 Objetivos del UML

Tendremos en cuenta los siguientes objetivos:

- Conseguir la capacidad de modelar sistemas, desde el concepto hasta los artefactos ejecutables, utilizando técnicas orientadas a objetos.
- Cubrir las cuestiones relacionadas con el tamaño inherente a los sistemas complejos y críticos.
- Crear un lenguaje de modelado utilizable tanto por las personas como por las máquinas.

Mediante el fomento del uso del UML, OMG pretende alcanzar los siguientes objetivos:

- Proporcionar a los usuarios un lenguaje de modelado visual expresivo y utilizable para el desarrollo e intercambio de modelos significativo.
- Proporcionar mecanismos de extensión y especialización.
- Ser independiente del proceso de desarrollo y de los lenguajes de programación.
- Proporcionar una base formal para entender el lenguaje de modelado.
- Fomentar el crecimiento del mercado de las herramientas OO.
- Soportar conceptos de desarrollo de alto nivel como pueden ser colaboraciones, frameworks, patterns, y componentes.
- Integrar las mejores prácticas utilizadas hasta el momento.

El UML debemos entenderlo como un estándar para modelado y no como un estándar de proceso software.

Es un leguaje que podemos utilizar en otros campos:

- Sistemas de información de la empresa.
- Bancos y servicios financieros.
- Telecomunicaciones.
- Transporte.
- Defensa/industria aeroespacial.
- Comercio.
- Electrónica médica.
- Ámbito científico.
- Servicios distribuidos en la Web.

## 6.6 Estructura de UML

Los conceptos y modelos de UML pueden agruparse en las siguientes áreas conceptuales:

1. Estructura estática:

   Cualquier modelo preciso debe primero definir su universo:

   - Los conceptos clave de la aplicación.
   - Sus propiedades internas.
   - Las relaciones entre cada una de ellas.

   Los conceptos de la aplicación son modelados como clases. La información almacenada es modelada como atributos. La estructura estática se expresa con diagramas de clases y puede usarse para generar la mayoría de las declaraciones de estructuras de datos en un programa.

2. Comportamiento dinámico:

   La visión de la interacción de los objetos se representa con los enlaces entre objetos junto con el flujo de mensajes y los enlaces entre ellos.

   Este punto de vista unifica:

   - La estructura de los datos.
   - El control de flujo.
   - El flujo de datos.

3. Construcciones de implementación:

   Los modelos UML tienen significado para el análisis lógico y para la implementación física.

   Un componente es una parte física reemplazable de un sistema y es capaz de responder a las peticiones descritas por un conjunto de interfaces.

   Un nodo es un recurso computacional que define una localización durante la ejecución de un sistema.

   Puede contener componentes y objetos.

4. Mecanismos de extensión:

   UML tiene una limitada capacidad de extensión.

5. Organización del modelo:

La información del modelo debe ser dividida en piezas coherentes, para que los equipos puedan trabajar en las diferentes partes de forma concurrente.

Debemos organizar el contenido en paquetes de tamaño modesto.

Los paquetes son unidades organizativas, jerárquicas y de propósito general de los modelos UML.

Pueden usarse para el almacenamiento, control de acceso, gestión de la configuración y construcción de bibliotecas que contengan fragmentos de código reutilizable.

6. Elementos de anotación:

Son las partes explicativas de los modelos UML.

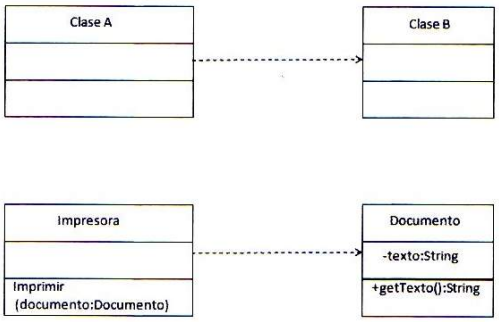
Son comentarios que se pueden aplicar para describir, clasificar y hacer observaciones sobre cualquier elemento de un modelo.

El tipo principal de anotación es la nota que simplemente es un símbolo para mostrar restricciones y comentarios junto a un elemento o un conjunto de elementos.

7. Relaciones:

Existen seis tipos de relaciones entre los elementos de un modelo UML.

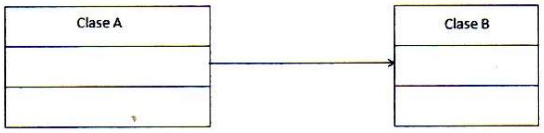
- **Dependencia**. Es una relación semántica entre dos elementos en la cual un cambio a un elemento, el elemento independiente, puede afectar a la semántica del otro elemento, el elemento dependiente. Se representa como una línea discontinua, que puede ser dirigida y a veces tiene uta etiqueta. LA clase A usa la clase B.

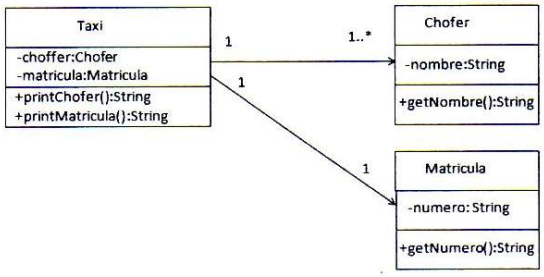



- **Asociación**. Es una relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. Se representa como una línea continua, que puede ser dirigida y a veces tiene uta etiqueta. A menudo se incluyen otros adornos para indicar la multiplicidad y los roles de los objetos involucrados. La multiplicidad de una asociación determina cuantos objetos de cada tipo intervienen en la relación. Cada asociación tiene dos multiplicidades, una para cada extremo de la relación. Para especificar la multiplicidad de una asociación hay que especificar la multiplicidad mínima y la multiplicidad máxima.

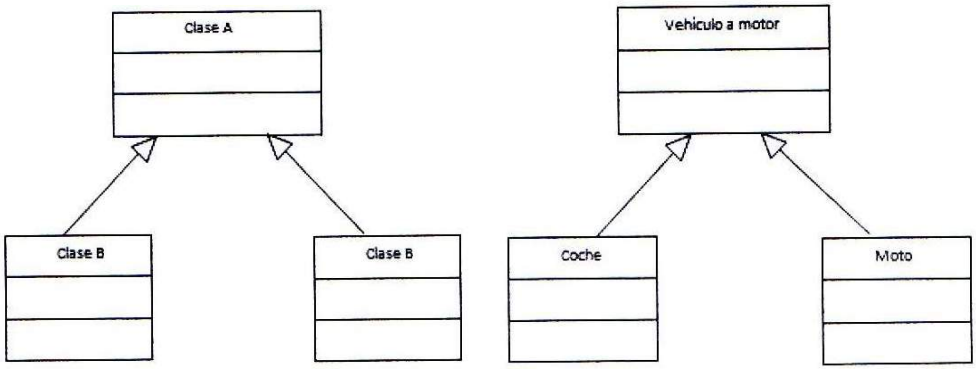
| **Multiplicidad** | **Significado** |
|---|---|
| 1 | Uno y sólo uno |
| 0..1 | Cero o uno |
| N..M | Desde N hasta M |
| * | Cero o varios |
| 0..* | Cero o varios |
| 1..* | Uno o varios, al menos uno |

La clase A está asociada a la clase B. La clase A necesita a la clase B.

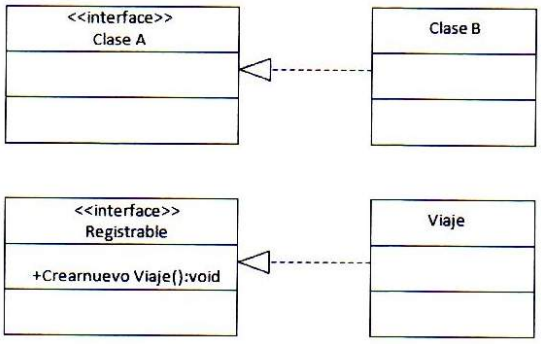


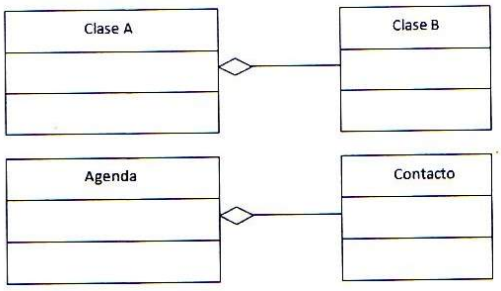

- **Generalización**. Es una relación de especialización / generalización en la cual los objetos del elemento especializado, ***el hijo***, pueden sustituir a los objetos del elemento general, ***el padre***. El hijo comparte la estructura y el comportamiento del padre. Se representa con una línea acabada en punta de flecha vacía. La clase A es una generalización de la clase B o la C. También podemos decir que la clase B y C son especializaciones de la clase A.



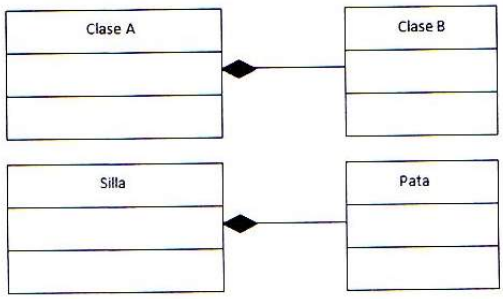

- **Realización**. Es una relación semántica entre clasificadores, donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos sitios: entre interfaces y las clases y componentes que las realizan, y entre los casos de uso y las colaboraciones que los realizan. Se representa como una mezcla entre la generalización y la dependencia, una línea discontinua con una punta de flecha vacía. En este ejemplo la clase B es una realización de la A. Viaje realiza la interface Registrable.



- **Agregación**. Es similar a la relación de agregación, solo varía en la multiplicidad ya que en lugar de ser una relación "uno a uno" es de "uno a muchos". Se representa con una flecha acabada en rombo de color blanco. La clase A agrupa varios elementos del tipo Clase B.




- **Composición**. Es similar a la agregación, pero la relación es más fuerte. Aporta documentación conceptual ya que es una "relación de vida", el tiempo de vida de un objeto está condicionado por el tiempo de vida del objeto que lo incluye. La Clase A agrupa varios elementos del tipo Clase B. el tiempo de vida de los objetos de tipo Clase B está condicionado por el tiempo de vida del objeto de tipo Clase A.




8. Diagramas.

   Se utilizan para representar diferentes perspectivas de un sistema de forma que un diagrama es una proyección del mismo UML.

   UML proporciona un amplio conjunto de diagramas que normalmente se usan en pequeños subconjuntos para poder representar las cinco vistas principales de la arquitectura de un sistema.

## 6.7 Diagramas UML

### 6.7.1 Diagramas de casos de uso

Ilustran la funcionalidad proporcionada por una unidad del sistema.

Describen las relaciones y las dependencias entre un grupo de casos de uso y los actores participantes en el proceso.

No están pensados para el diseño y no pueden describir los elementos internos de un sistema.

Sirven para facilitar la comunicación con los futuros usuarios del sistema y con el cliente.

Resultan especialmente útiles para determinar las características necesarias que tendrá el sistema.

Describen qué es lo que debe hacer el sistema, pero no cómo.

#### Actor

Es una entidad externa, fuera del sistema, que interacciona con el sistema participando y normalmente iniciando en un caso de uso.

Los actores pueden ser gente real, otros sistemas o eventos.

No representan a personas físicas o a sistemas, sino un rol.

Cuando una persona interactúa con el sistema de diferentes maneras, estará representado por varios actores.

## Caso de uso

Describe desde el punto de vista de los actores, un grupo de actividades de un sistema que produce un resultado concreto y tangible.

Son descriptores de las interacciones típicas entre los usuarios de un sistema y ese mismo sistema.

Representan el interfaz externo del sistema y especifican qué requisitos de funcionamiento debe tener éste, únicamente el qué, nunca el cómo.

Hay que tener en cuenta las siguientes reglas:

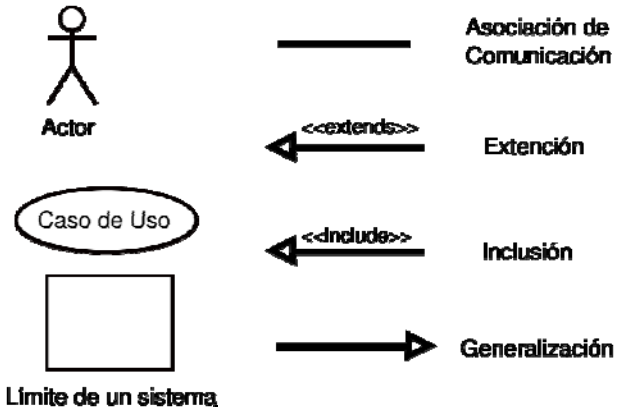
- Cada caso de uso está relacionado como mínimo con un actor.
- Cada caso de uso es un iniciador, un actor.
- Cada caso de uso lleva a un resultado relevante.


Figura 6.8 Actor y caso de uso

Los casos de uso pueden tener relaciones con otros casos de uso.

Los tres tipos de relaciones más comunes entre casos de uso son:

- "**include**" que especifica una situación en la que un caso de uso tiene lugar dentro de otro caso de uso.
- "**extends**" que especifica que en ciertas situaciones, o en algún punto un caso de uso será extendido por otro.
- Generalización que especifica que un caso de uso hereda las características del "super" caso de uso, y pude volver a especificar algunas o todas ellas de una forma muy similar a las herencias entre clases.



En la figura 6.9 se representa un diagrama de casos de uso de un cajero automático con dos actores: un cliente y un empleado de banco.

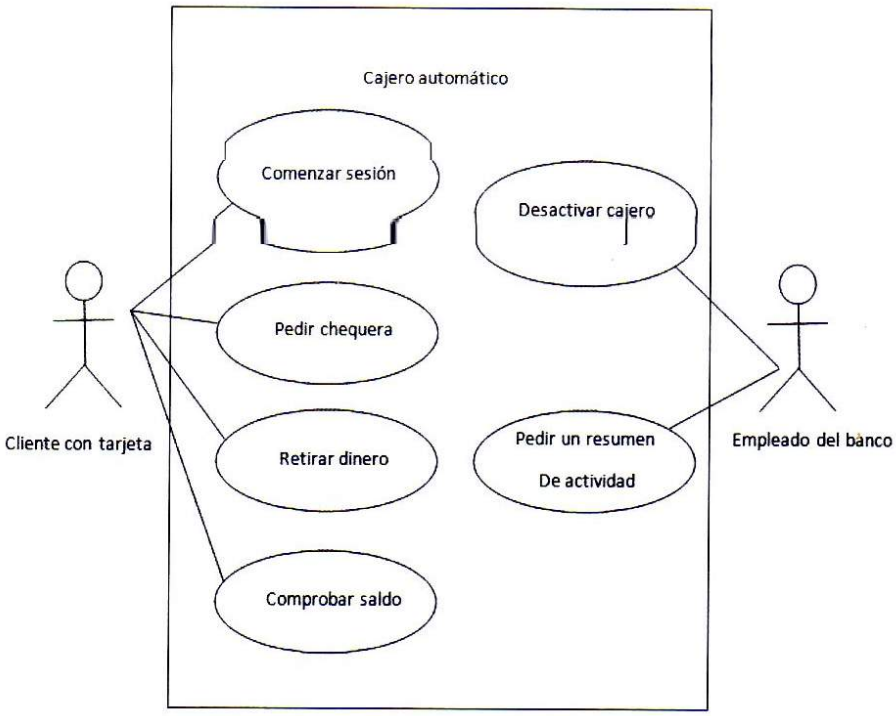
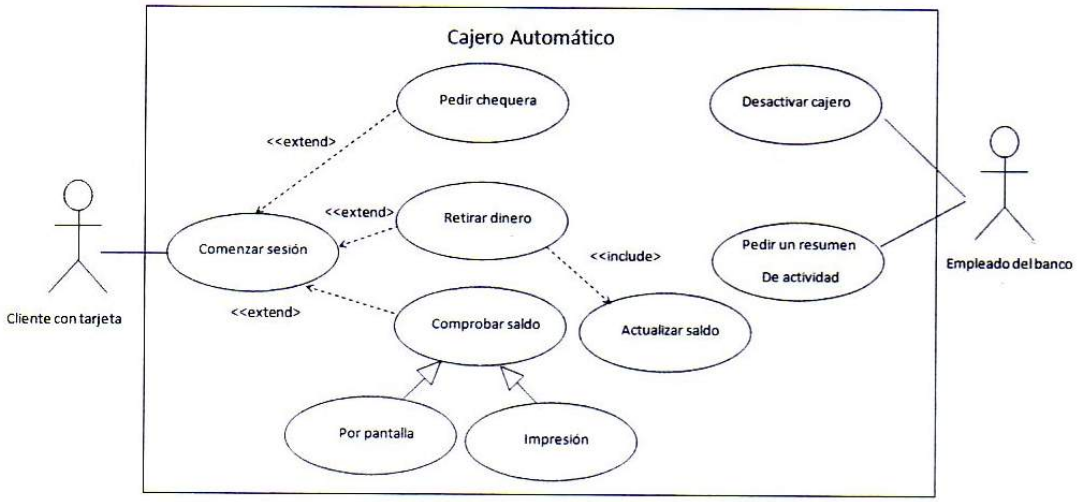



Figura 6.9 Diagrama de casos de usos

En la figura 6.10 se muestra el diagrama de casos de uso del cajero automático presentado en la figura 6.9 extendiendo los casos de uso con las diferentes relaciones.




## Descripción de casos de uso

Son reseñas textuales del caso de uso.

Normalmente tienen el formato de una nota o un documento relacionado de alguna manera con el caso de uso, y explica los procesos o actividades que tienen lugar en el caso de uso.

### 6.7.2 Diagrama de clases

Los diagramas de clases muestran las diferentes clases que componen un sistema y cómo se relacionan unas con las otras.

Los diagramas de clases son diagramas estáticos porque muestran las clases, junto con sus métodos y atributos, así como las relaciones estáticas entre ellas: qué clases conocen a qué otras clases o qué clases son parte de otras clases.

No muestran los métodos mediante los que se invocan entre ellas.

#### Clase

Define los atributos y los métodos de una serie de objetos.

Todos los objetos de esta clase, instancias de clase, tienen el mismo comportamiento y el mismo conjunto de atributos.

Se representan con rectángulos, con el nombre de la clase y también se pueden mostrar los atributos y operaciones de la clase en otros dos compartimentos del rectángulo dentro de la clase.

La clase está formada por atributos y operaciones o métodos.

- Atributos

Se muestran al menos con su nombre y también pueden mostrar su tipo, valor inicial y otras propiedades.

Aparecen calificados en el diagrama dependiendo de su acceso como:

+ Indica atributos públicos.

# Indica atributos protegidos.

− Indica atributos privados.

- Operaciones

Se muestran al menos con su nombre y pueden mostrar sus parámetros y valores de retorno.

Aparecen calificadas en el diagrama dependiendo de su acceso como:

+ Indica operaciones públicas.

# Indica operaciones protegidas.

− Indica operaciones privadas.

- Plantillas

Las clases pueden tener plantillas o ***templates***, un valor usado para una clase no especificada o un tipo.

El tipo de plantilla se especifica cuando se inicia una clase, cuando se crea un objeto.

#### Asociaciones de Clases

Las clases se pueden relacionar con otras de las siguientes formas posibles:

- Generalización.
- Asociación.
- Realización.
- Agregación.
- Composición.

Los diagramas de clases pueden contener más componentes aparte de clases, pueden ser:

- **Interfaces**. Son clases abstractas, son instancias que no pueden ser creadas directamente a partir de ellas. Pueden contener operaciones, pero no atributos. Las clases pueden heredarse de las interfaces pudiendo así realizarse instancias a partir de estos diagramas.
- **Tipo de Datos**. Son primitivas incluidas en algunos lenguajes de programación. No pueden tener relación con clases, pero las clases sí pueden relacionarse con ellos.
- **Enumeraciones**. Son simples listas de valores. No pueden relacionarse con las clases, pero las clases sí pueden hacerlo con ellos.
- **Paquetes**. En lenguajes de programación, representan un espacio de nombres en un diagrama se emplean para representar partes del sistema que contienen más de una clase, incluso cientos de ellas.

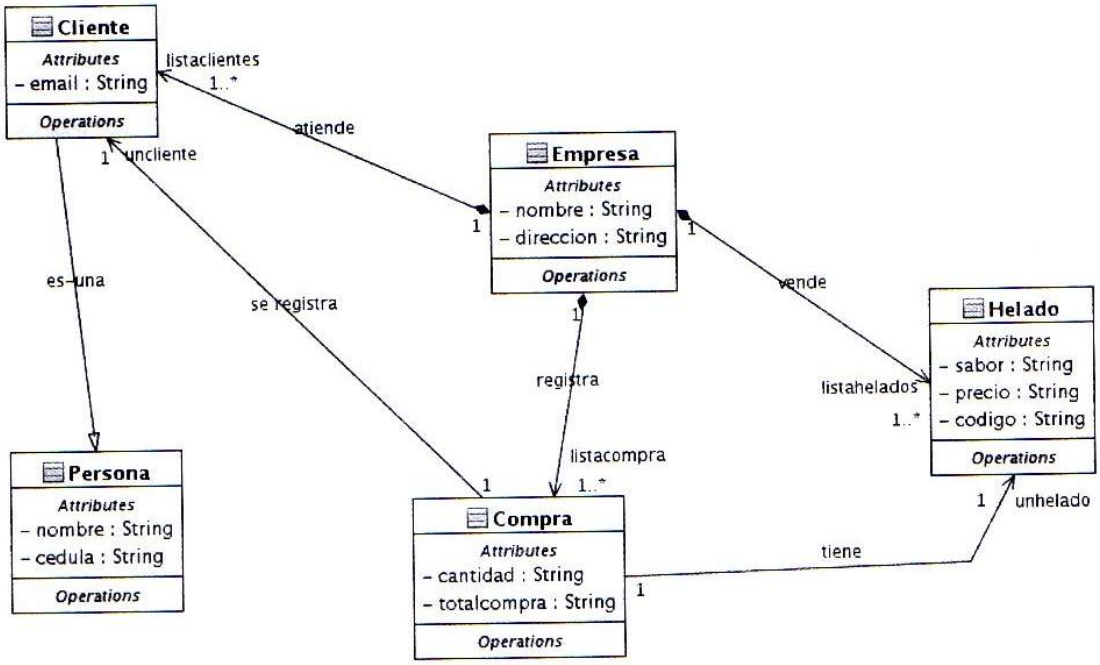



Figura 6.11 Diagrama de clases

### 6.7.3 Diagramas de secuencia

Muestran el flujo de mensajes entre objetos para un determinado caso de uso.

Ponen especial énfasis en el orden y el momento en que se envían los mensajes a los objetos.

Se explica la secuencia de las llamadas que se producen entre los objetos que intervienen.

Pueden tener mayor o menor detalle y representar diferentes llamadas a diferentes objetos.

Tienen dos dimensiones:

- La vertical muestra la secuencia de llamadas ordenadas en el tiempo en el que ocurren.
- La horizontal muestra las diferentes instancias de objetos a las que son enviadas las llamadas.

En la figura 6.12 se puede ver una secuencia de invocaciones para el dibujo del botón de una ventana de una aplicación

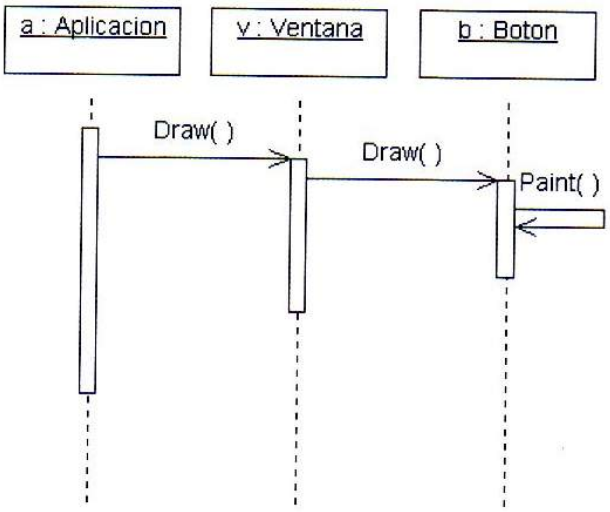



Figura 6.12 Diagrama de secuencia simple

Los objetos están representados por rectángulos con el nombre del objeto dentro y por líneas intermitentes verticales con el nombre del objeto en la parte más alta.

El eje de tiempo también es vertical, incrementándose hacia abajo, de forma que los mensajes son enviados de un objeto a otro en forma de flechas con los nombres de la operación y los parámetros.

Es conveniente dibujar una flecha de retorno de la llamada.

Los mensajes pueden ser:

- Síncronos, el tipo normal de llamada del mensaje donde se pasa el control a objeto llamado hasta que el método finalice.
- Asíncronos, donde se devuelve el control directamente al objeto que realiza la llamada.

Los mensajes síncronos tienen una caja vertical en un lateral del objeto invocante que muestra el flujo del control del programa.

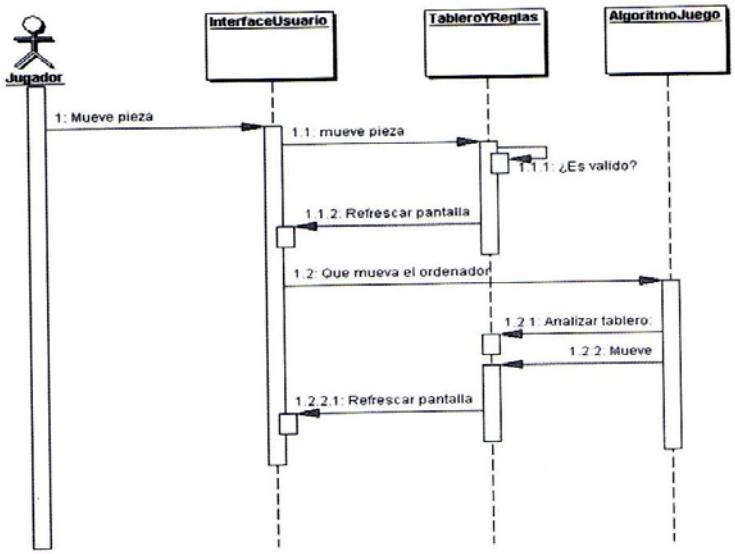



Figura 6.13 Diagrama de secuencia

### 6.7.4 Diagramas de colaboración

Muestran las interacciones que ocurren entre los objetos que participan en una situación determinada.

Se parece a los diagramas de secuencia, pero destacando la forma en que las operaciones se producen en el tiempo.

Los diagramas de colaboración fijan el interés en las relaciones entre objetos y su topología.

Los mensajes enviados de un objeto a otro se representan mediante flechas, mostrando el nombre del mensaje, los parámetros y la secuencia del mensaje.

Están indicados para mostrar una situación o flujo de programa específicos y son unos de los mejores tipos de diagramas para demostrar o explicar rápidamente un proceso dentro de la lógica del programa.

En las figuras 6.14 y 6.15 podemos ver los diagramas de secuencia y el respectivo de colaboración.

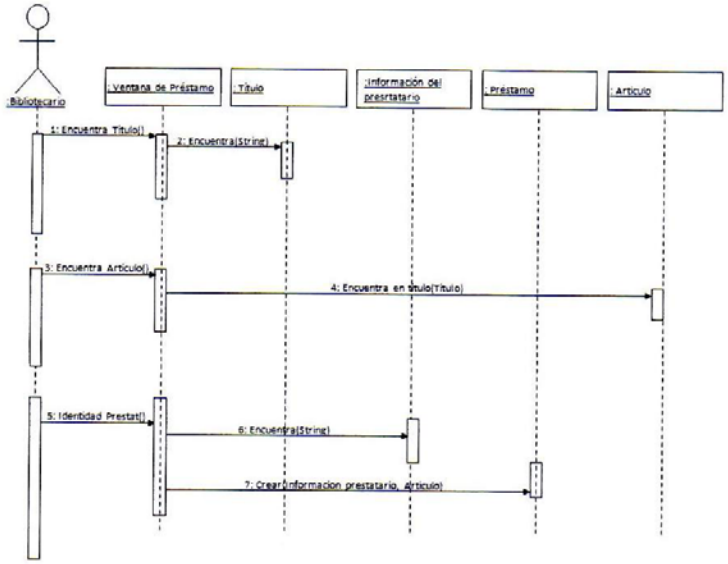



Figura 6.14 Diagrama de secuencia

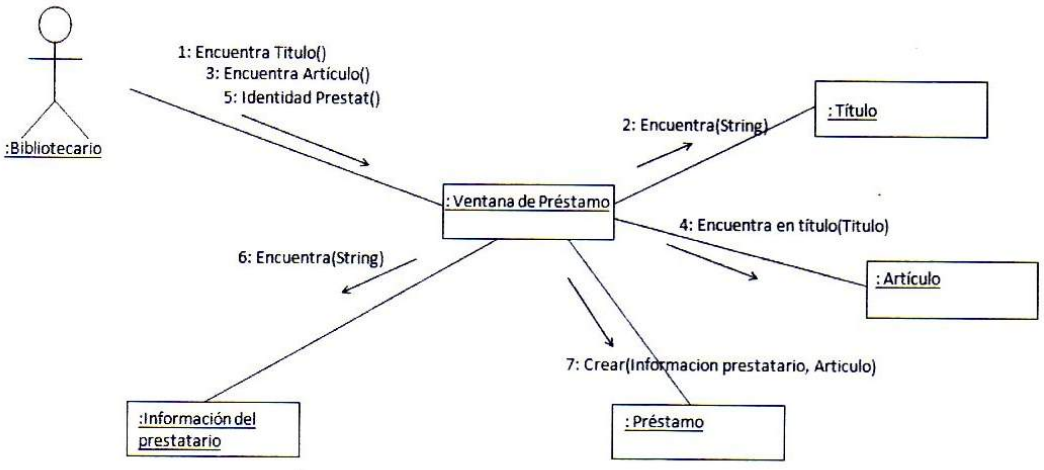



Figura 6.15 Diagrama de colaboración

### 6.7.5 Diagrama de estado

Muestran los diferentes estados de un objeto durante su vida, y los estímulos que provocan los cambios de estado en un objeto.

Ven a los objetos como máquinas de estado o autómatas finitos que pueden estar en un conjunto de estados finitos y que pueden cambiar su estado a través de un estímulo perteneciente a un conjunto finito.

Todos los objetos tienen un estado pero no todos los objetos son susceptibles de generar un diagrama de estados.

Solo aquellos que presenten "***estados interesantes***", es decir tres o más estados, deberán ser modelados en su diagrama de estados

La notación de un diagrama de estados tiene cinco elementos básicos:

- **Punto de inicio**: dibujado con un círculo relleno.
- **Transición entre estados**: dibujado con una línea terminada con punta de flecha abierta.
- **Estado**: dibujado con un rectángulo con los vértices redondeados.
- **Punto de decisión**: dibujado con un círculo no relleno.
- **Puntos de terminación**: dibujados con un círculo con otro relleno en su interior.

Para dibujar un diagrama de estado comenzaremos por el punto de inicio y la línea de transición que lleve hasta el primer estado del objeto.

Después se dibujan cada uno de los estados distribuidos por el diagrama y se conectan con las líneas de transición de estados.

Hay dos tipos especiales de estado: inicio y fin.

No hay ningún evento que pueda devolver a un objeto a su estado de inicio y de la misma forma no hay ningún evento que pueda sacar a un objeto de su estado de fin.

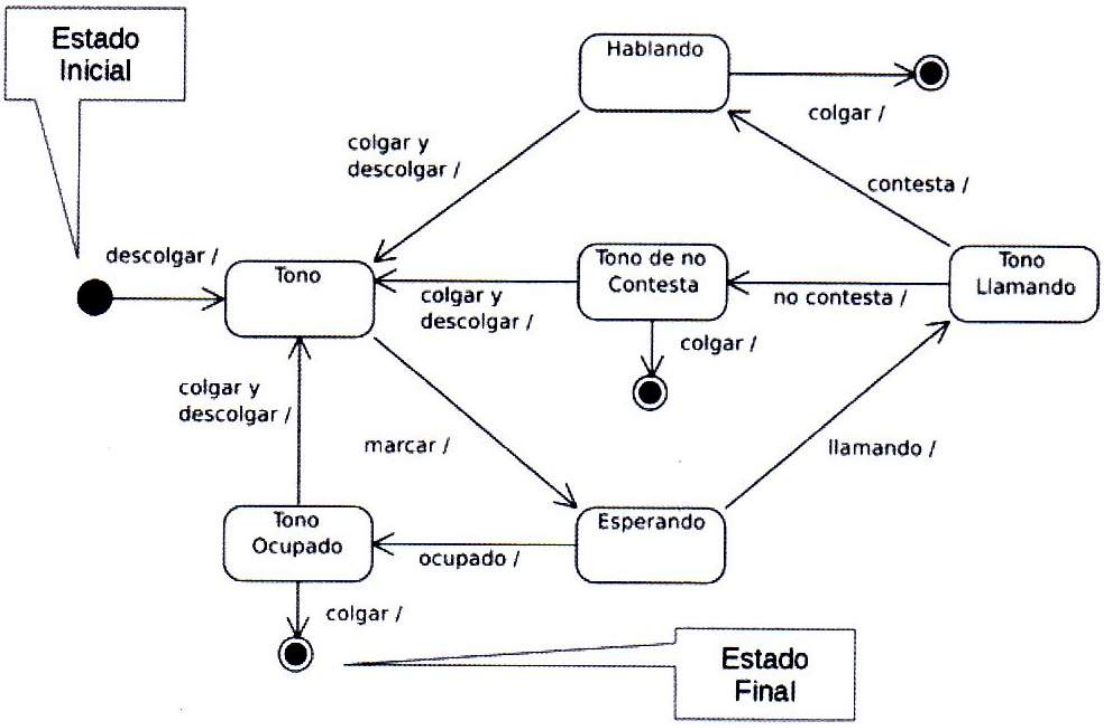
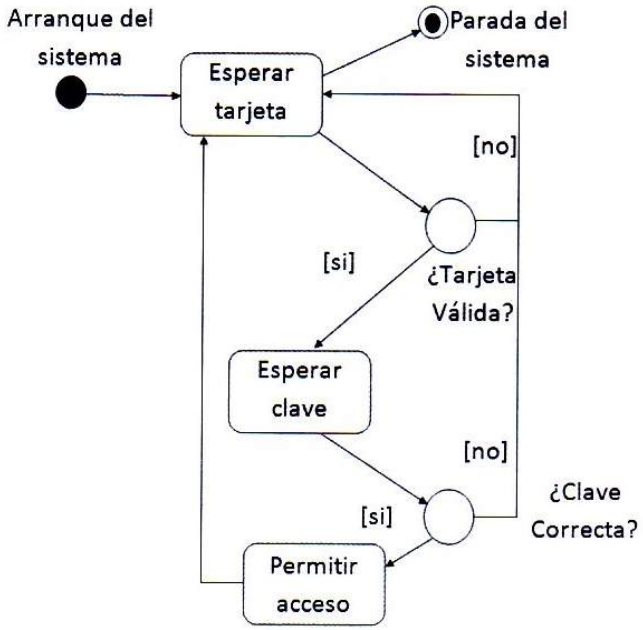



Figura 6.16 Diagrama de transición de estados

En la figura 6.17 podemos ver el diagrama de transición de estados del acceso con tarjeta con dos puntos de bifurcación




### 6.7.6 Diagrama de actividad

Describen la secuencia de las actividades en un sistema.

Son una forma especial de los diagramas de estado que únicamente contienen actividades.

Son similares a los diagramas de flujo procesales con la diferencia de que todas las actividades están claramente unidas a objetos.

Siempre están asociados a una clase.

Soportan actividades tanto secuenciales como paralelas.

La ejecución paralela se representa por medio de iconos de fork/espera, y en el caso de las actividades paralelas, no importa en qué orden sean invocadas, pueden ser ejecutadas simultáneamente una detrás de otra.

## Actividad

Una actividad es un único paso de un proceso.

Es un estado del sistema que tiene actividad interna y al menos una transición saliente.

También pueden tener más de una transición saliente, si tienen diferentes condiciones.

Pueden formar jerarquías, es decir, formada de varias ***actividades de detalle*** en cuyo caso las transiciones entrantes y salientes deberían coincidir con las del diagrama de detalle.

Los diagramas de actividad son indicados para modelar procesos de alto nivel.

Los elementos para definir un diagrama de actividades son similares a los usados en los diagrama de secuencia.

- **Círculo** relleno de inicio.
- **Rectángulos** con los bordes redondeados para determinar las actividades.
- **Flechas** conectoras para unir las actividades.
- **Puntos** de decisión con círculos huecos.
- **Líneas** divisoras o calles para establecer las responsabilidades entre los distintos objetos del sistema.

En la figura 6.18 se ve un diagrama de actividades para la gestión de pedidos en una empresa.

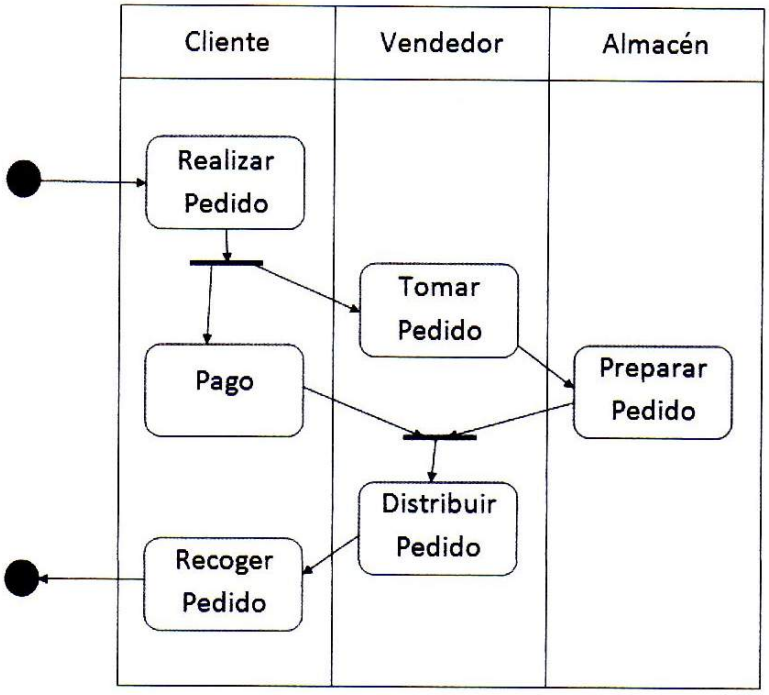



Figura 6.18 Diagrama de secuencia de actividad

### 6.7.7 Diagramas de componentes

Muestran los componentes del software y los artilugios de los que está compuesto, como:

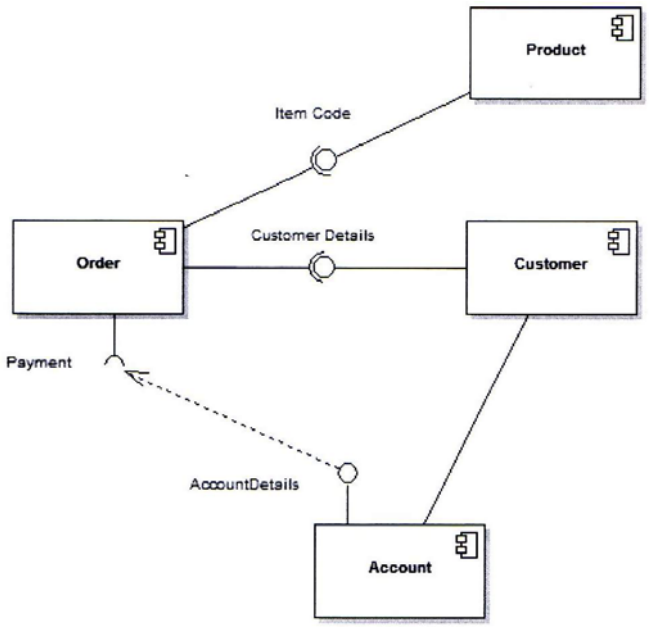
- Los archivos de código fuente.
- Las librerías.
- Las tablas en una base de datos.

Los componentes pueden tener interfaces que permiten asociaciones entre componentes.

Ilustran las piezas de software, controladores embebidos, etc. Que conformarán un sistema.

Un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clase.

Normalmente un componente se implementa por una o más clases en tiempo de ejecución.



### 6.7.8 Diagrama de despliegue

Muestra cómo el sistema se asentará físicamente en el entorno hardware que lo acompaña.

El objetivo es mostrar dónde los componentes del sistema se ejecutarán y cómo se comunicarán entre ellos.

Será un diagrama muy usado por las personas que produzcan el sistema.

La notación es la misma que el diagrama de componentes.

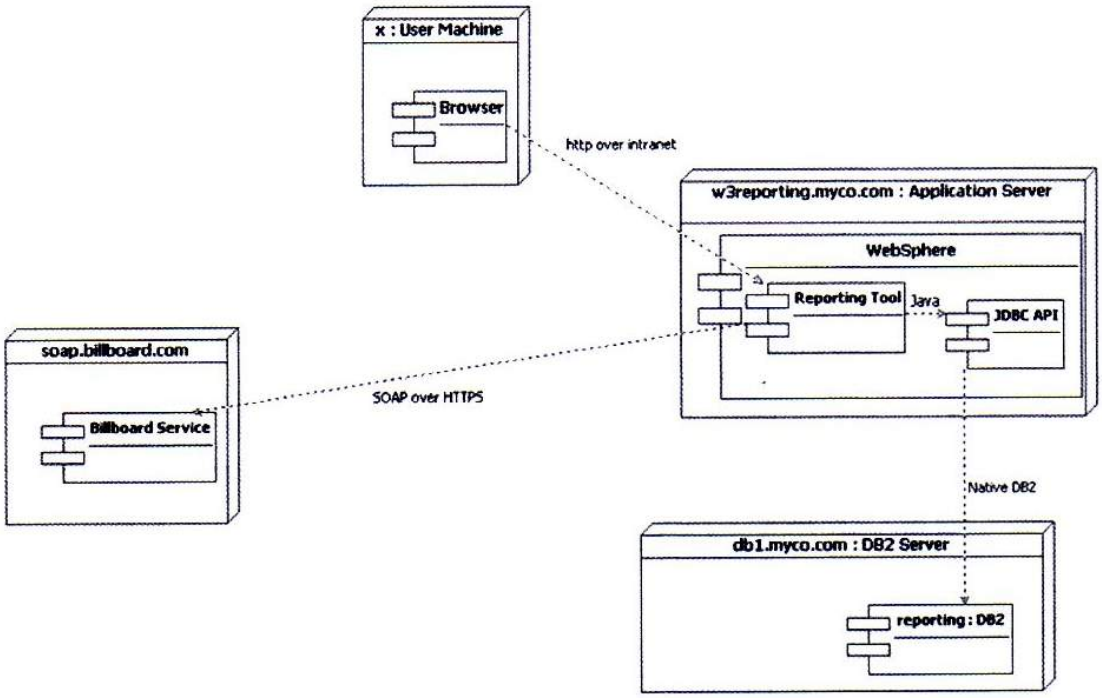



Figura 6.20 Diagrama de despliegue