

# ENTORNOS DE DESARROLLO

## UNIDAD 5: Lenguaje Unificado de Modelado (UML)



# INDICE

1- LENGUAJE UML 2.0	
2- DIAGRAMAS DE CLASES	
3- DIAGRAMAS DE COMPORTAMIENTO.....	18
3.1 CASOS DE USO.....	27-37
3.2 DIAGRAMAS DE SECUENCIAS.....	38-50
3.3 DIAGRAMAS DE COLABORACIONES	
3.4 DIAGRAMA DE ESTADOS	



# Introducción a UML

Para realizar labores de diseño de software, es vital realizar modelados o diagramas representando gráficamente la estructura de la aplicación y su funcionalidad, de una manera fácil de entender y rápida de crear. Antiguamente, los diagramas de cada diseñador eran únicos, ya que, salvo un par de diagramas conocidos por todos (como los diagramas de flujo o los diagramas de entidad-relación), cada diseñador o analista realizaba los diagramas de la manera que mejor los entendía, por lo que podría suponer un problema si varias personas tenían que entender los diagramas que uno o varios diseñadores les presentaban para definir el software.

Con el fin de evitar ese problema se creó UML (Unified Modeling Language), un conjunto unificado de estándares para las diferentes necesidades y usos que un diseñador pudiera tener a la hora de plantear una representación gráfica de un programa. Son diagramas de propósito general que se presuponen conocidos por todos, con unas técnicas de notación conocidas, de modo que cualquiera pueda crear diagramas entendibles por todos.





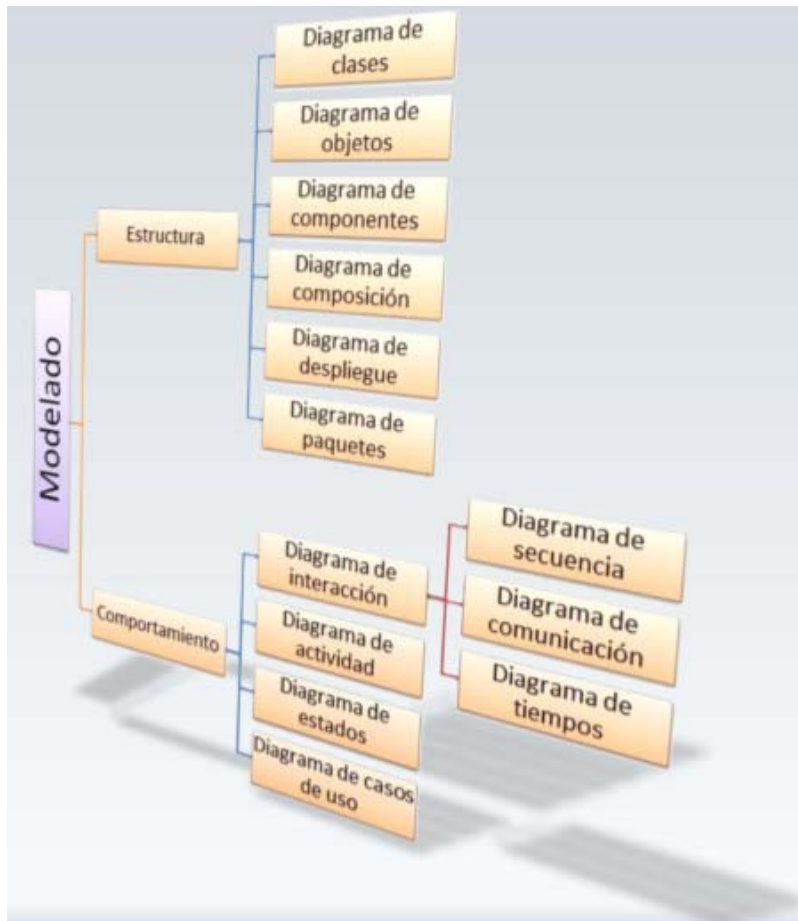
# Introducción a UML. UML 2.0

En la versión 2.0 de UML se definió un completo árbol de superestructuras donde se relacionaban y complementaban todos los tipos de diagramas UML a la hora de definir un software por completo.

Teniendo presente la superestructura de diagramas de UML, podríamos pensar que la tarea de definir y realizar dichos diagramas (y que por supuesto sean coherentes entre sí) sería una tarea abrumadora, pero no es necesario ni se suelen realizar todos los diagramas para modelar un software, por norma general se utilizan solo una serie de diagramas para modelarlo, lo más clásico sería la tríada diagrama de clases, de secuencia y de casos de uso.



# Introducción a UML. UML 2.0





# Diagramas de CLASES



# Diseño de clases en UML

Estos diagramas son particularmente útiles en el desarrollo de la capa del modelo, para mapear de un modo gráfico las entidades y sus relaciones en nuestra aplicación.

## ELEMENTOS:

- ▶ **CLASES.** Agrupan conjunto de objetos con características comunes, llamadas **ATRIBUTOS** y su comportamiento llamados **METODOS**. Tendrán una visibilidad.
- ▶ **RELACIONES.** Entre los elementos del sistema que componen las **CLASES**. Pueden ser **ASOCIACIÓN**, **AGREGACIÓN**, **COMPOSICIÓN** y **GENERALIZACIÓN**.
- ▶ **NOTAS.** Es un cuadro para escribir comentarios para ayudar al entendimiento del diagrama.
- ▶ **ELEMENTOS DE AGRUPACIÓN.** Se utiliza para modelar un sistema grande, agrupando en paquetes que se relacionan entre si.



# Diseño de clases en UML

En primer lugar, debemos percatarnos de que las clases se definen mediante cuadrados, y dichos cuadrados se dividen en tres segmentos horizontales.

- ▶ 1º) tendríamos el nombre de la clase
- ▶ 2º) Seguidamente irían los atributos
- ▶ 3º) Al final, los métodos de la clase.

Es importante resaltar que si alguna clase no posee atributos propios o métodos propios, se deberá seguir dibujando el segmento que le corresponde, incluso si está vacío.

El carácter que precede al nombre del atributo o método se corresponde con su grado de comunicación o **visibilidad**.

+ **Público**, el elemento será visible tanto desde dentro como desde fuera de la clase.

- **Privado**, el elemento de la clase solo será visible desde la propia clase.

#**Protegido**, el elemento no será accesible desde fuera de la clase, pero podrá ser manipulado por los demás métodos de la clase o de las subclases.

~ **paquete/defecto**, define la visibilidad del paquete que puede ser utilizada por otra clase mientras esté en el mismo paquete.



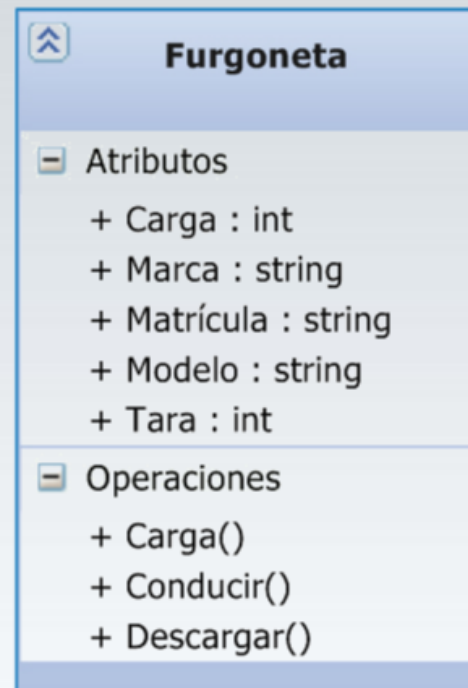


# Diseño de clases en UML

## DISEÑO DE CLASES EN UML

Cada bloque dentro del diagrama representa a una clase; una clase, como sabemos, dispone de unos atributos y de unos métodos asociados. Representaríamos las clases como cajas en donde escribiríamos sus atributos y sus métodos.

Las clases representan a nuestros objetos, los atributos definen las propiedades y características del objeto y los métodos especifican las acciones que podemos realizar con dicho objeto.



# Diseño de clases en UML

## Relaciones

En el diagrama de clases, además de dibujar y representar las clases con sus atributos y métodos, también se representan las relaciones.

Las relaciones se representan con flechas con una forma determinada, y, además, poseen una cardinalidad.

La cardinalidad es un número o símbolo que representa al número de elementos de cada clase en cada relación, pudiendo usar el comodín "\*" para definir un número indeterminado de elementos.





# Diseño de clases en UML

## Relaciones

Cardinalidad	Significado
1	Uno y solo uno
0..1	Cero o uno
X..Y	Desde X hasta Y
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios



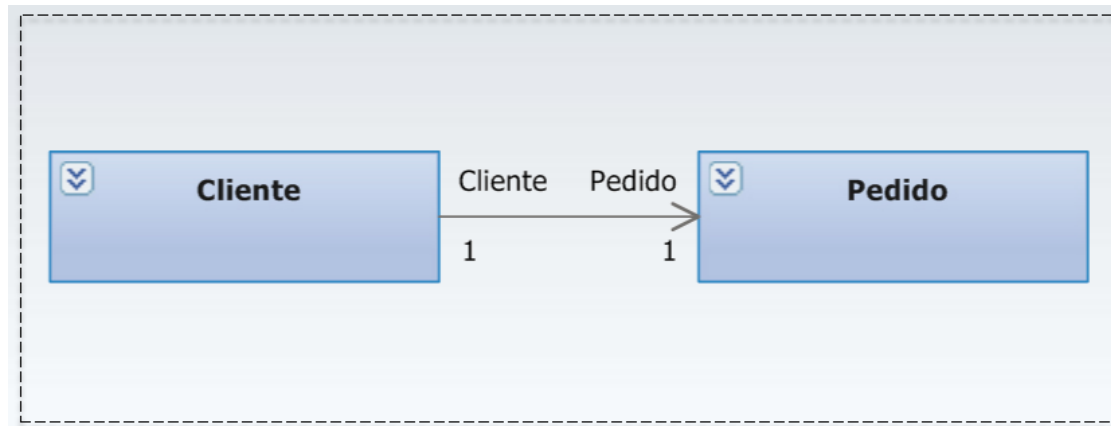


# Diseño de clases en UML

## Relaciones

### Asociación

La asociación es la más básica de las relaciones, no tiene un tipo definido y puede ser tanto una composición como una agregación, además una asociación podría implicar únicamente un uso del objeto asociado. Se representan mediante una flecha simple que también puede tener una cardinalidad.

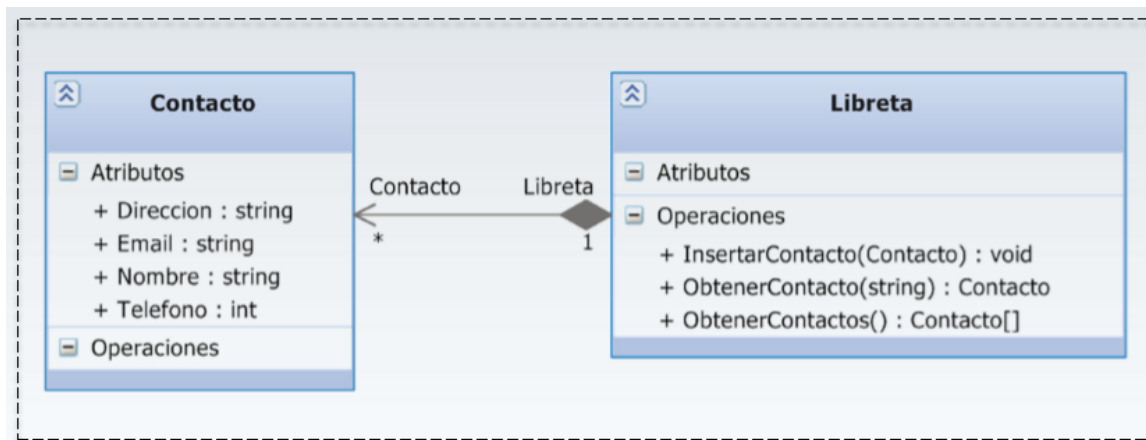


# Diseño de clases en UML

## Relaciones

### Composición

La composición define los componentes de los que se compone otra clase, se define además que la clase que contiene la composición no tiene sentido de existencia si la(s) agregada(s) desaparece(n). Mediante esta relación, denotada por una flecha con un rombo relleno en una de sus puntas, se indica la presencia de las clases origen en la clase destino, donde la clase destino es apuntada por el rombo de la relación.





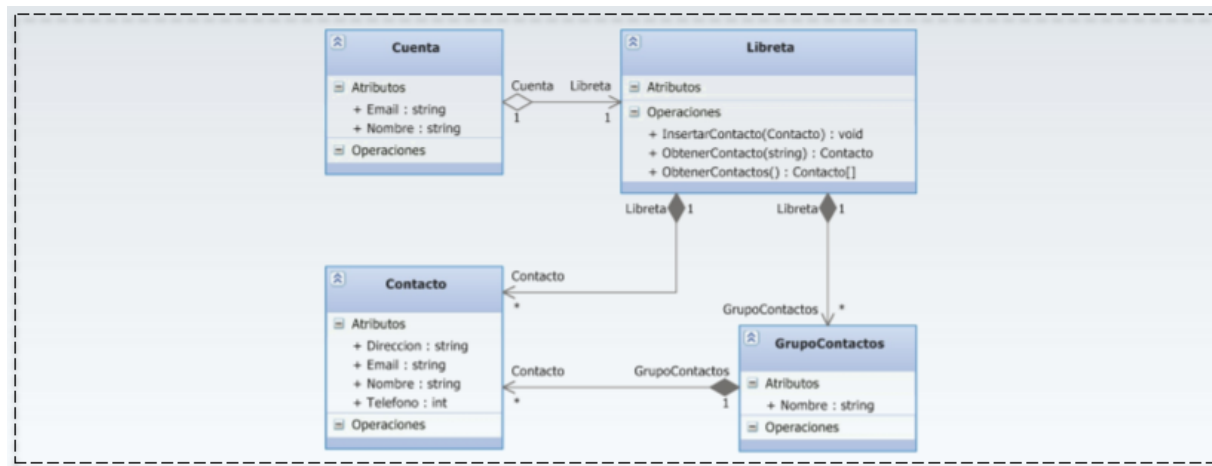
# Diseño de clases en UML

## Relaciones

### Agregación

La agregación es un tipo de asociación que indica que una clase es parte de otra clase. A diferencia que en la composición, la destrucción del compuesto no conlleva la destrucción de los componentes. Habitualmente se da con mayor frecuencia que la composición.

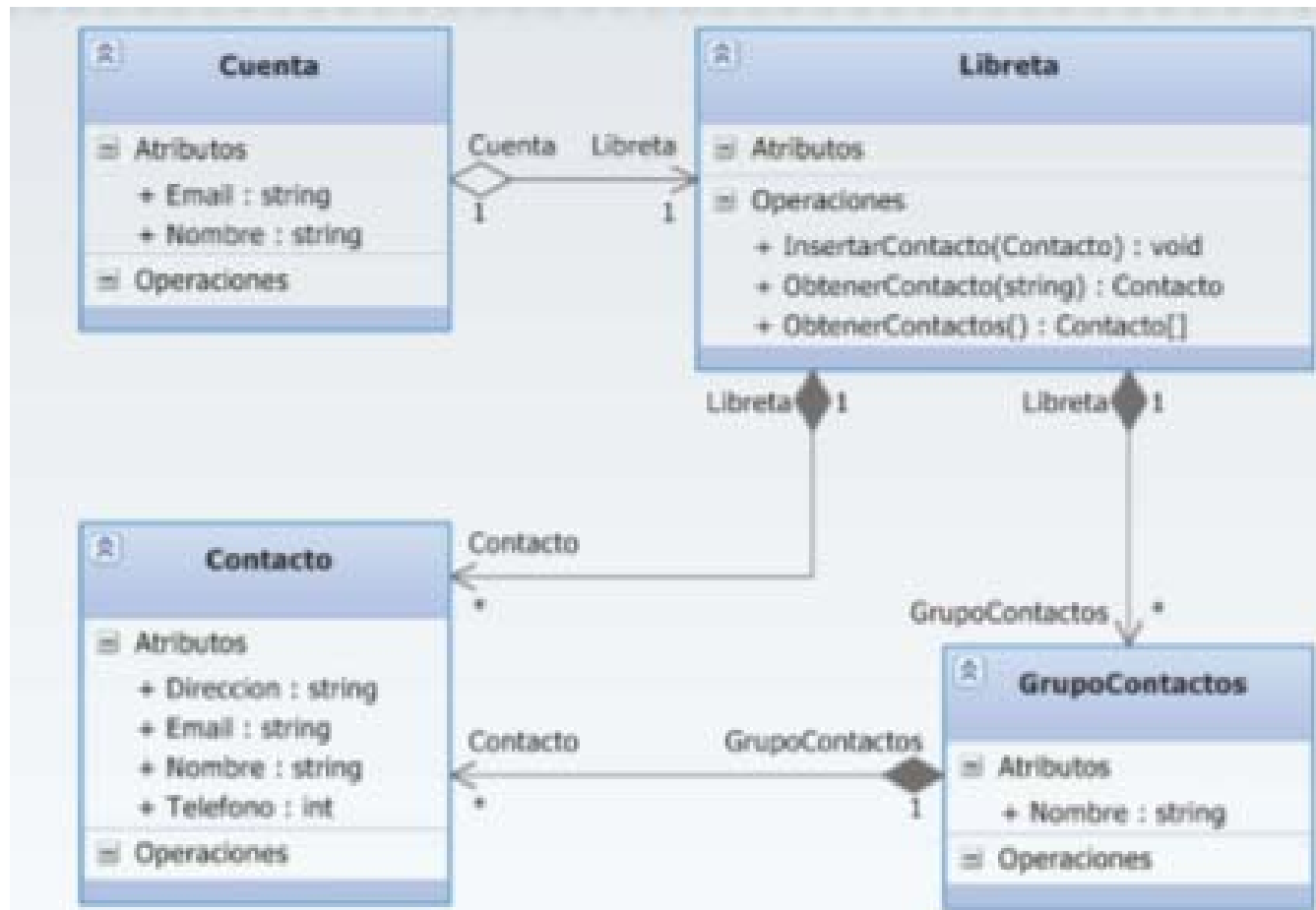
La agregación se representa en UML mediante un rombo de color blanco colocado en el extremo en el que está la clase que representa el "todo"





# Diseño de clases en UML

Relaciones: AGREGACION



# Diseño de clases en UML

## Relaciones

### Agregación vs Composición

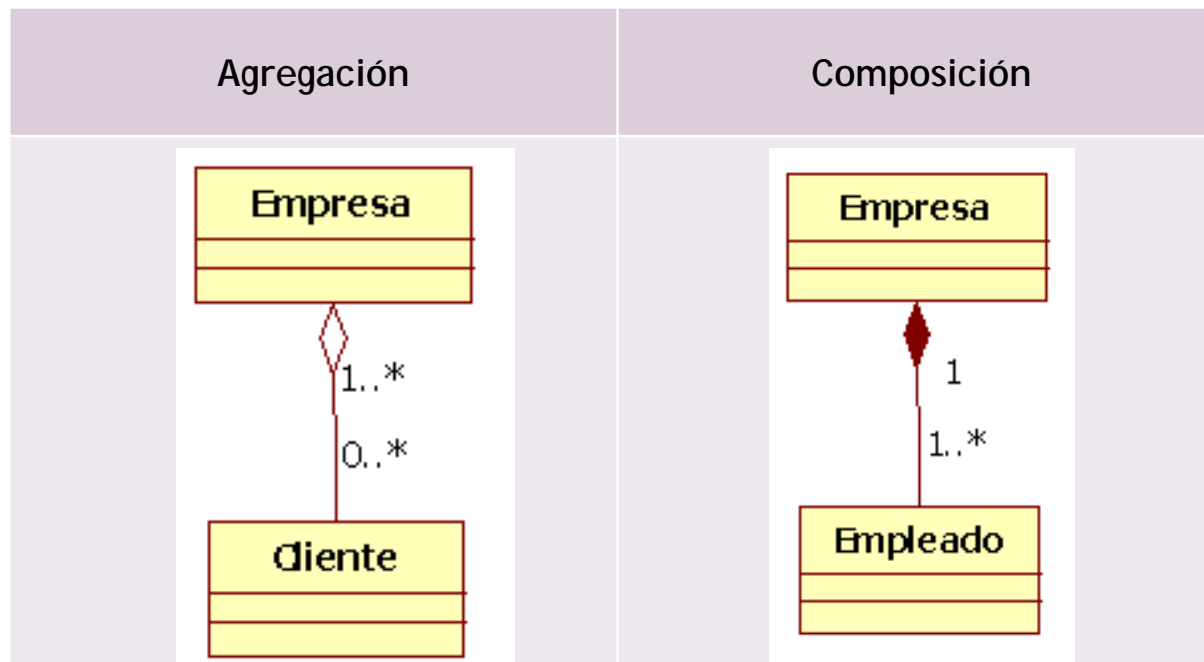
	Agregación	Composición
Varias asociaciones comparten los componentes	Sí	No
Destrucción de los componentes al destruir el compuesto	No	Sí
Cardinalidad a nivel de compuesto	Cualquiera	0..1 ó 1
Representación	Rombo transparente	Rombo negro



# Diseño de clases en UML

## Relaciones

### Agregación vs Composición



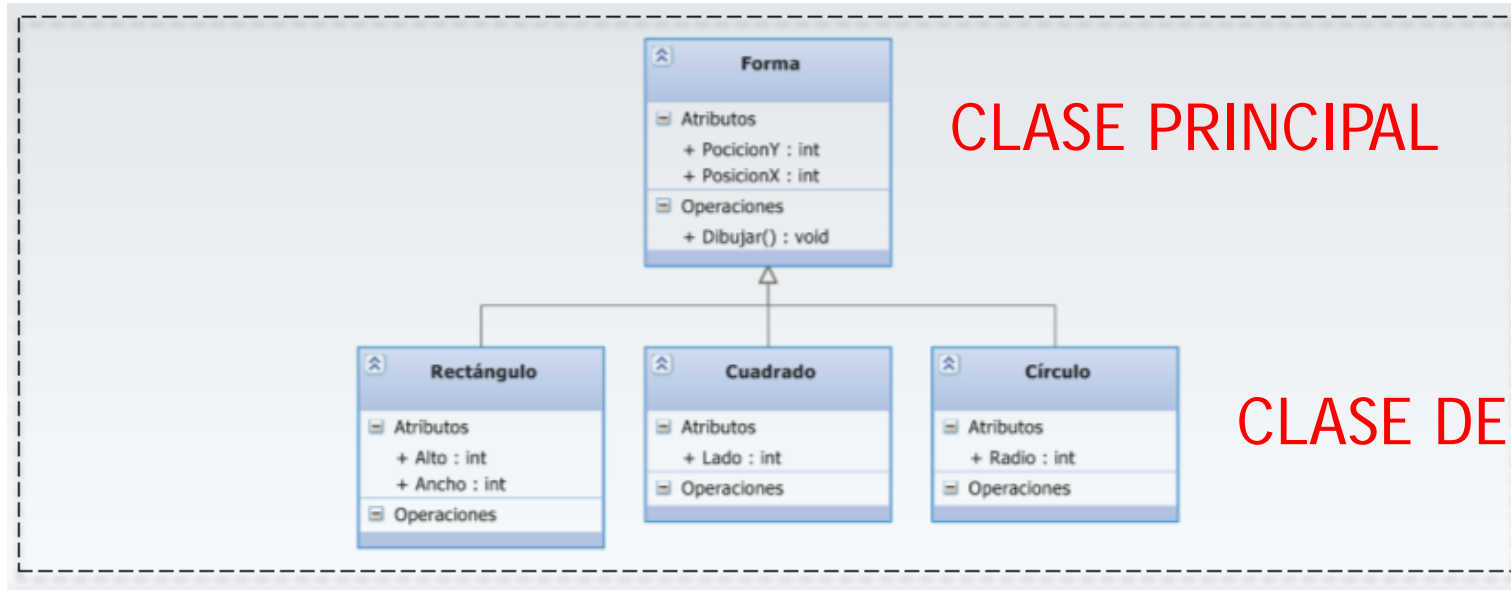


# Diseño de clases en UML

## Relaciones

### Herencia

La herencia es un modo de representar clases y subclases, es decir, clases más específicas de una general, se representan mediante una flecha con una punta triangular vacía.

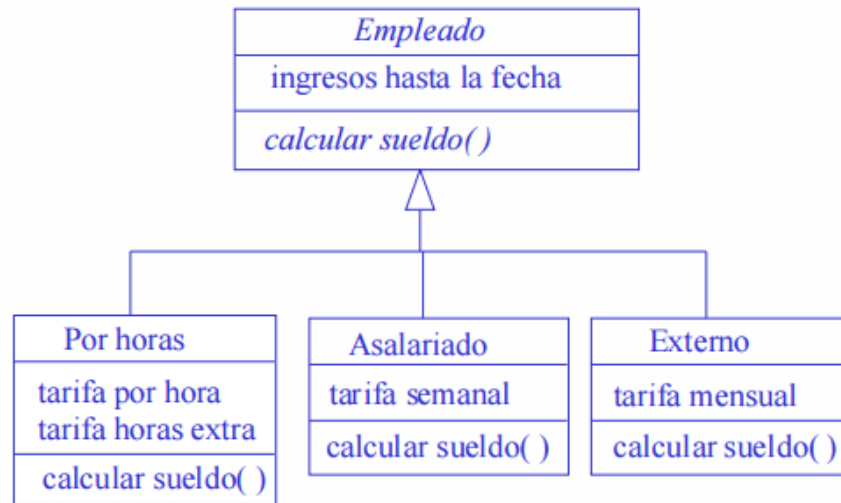


# Diseño de clases en UML

## Clases abstractas

Una clase abstracta es una clase sin instancias directas, es decir que no se puede instanciar. Una clase abstracta es una clase que contiene uno o más métodos abstractos, es decir, que no están implementados. La clase heredera tiene que implementar todos los métodos abstractos de la clase abstracta o ser también una clase abstracta.

Para representar una clase abstracta en UML escribiremos su nombre en cursiva.



# Diseño de clases en UML

Ejemplo 01- Tienda de Música





# Diseño de clases en UML

## Ejemplo 02- EMPRESA

Representa mediante un diagrama de clase las siguientes especificaciones:

- ▶ Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes. Ambos se caracterizan por su nombre y edad.
- ▶ Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.
- ▶ De los clientes además se necesita conocer su teléfono de contacto.
- ▶ La aplicación necesita mostrar los datos de empleados y clientes.



# Diseño de clases en UML

## Interfaces

Una interfaz (interface) contiene la declaración de un conjunto de operaciones sin su implementación correspondiente y se usa para especificar un servicio proporcionado por una clase o un componente. Una interfaz no tiene instancias.

Se representa con el símbolo de un clasificador (rectángulo), precediendo el nombre con el estereotipo <<interface>>, o con una línea con un círculo en el extremo, etiquetado con el nombre de la interfaz.

**InterfaceName**



**<<interface>>**  
**InterfaceName**



# Diagramas de comportamiento





# Diagramas de comportamiento

Existen diversos tipos de diagramas de comportamiento basados en el estándar unificado UML.

Los diagramas de comportamiento son un modo de representar gráficamente los procesos y formas de uso de un programa, con ellos visualizamos los aspectos dinámicos de un sistema, como el flujo de mensajes a lo largo del tiempo, el movimiento físico de los componentes en una red o los diferentes estados y operaciones que transcurren en el ciclo de vida de un programa.

Si en los diagramas de clases veíamos cómo se definían y especificaban las diferentes clases, sus operaciones y relaciones, los diagramas de comportamiento nos dirán cómo interactúan a lo largo del tiempo dichas clases y operaciones.



# Diagramas de comportamiento

Al tratarse de un tipo totalmente diferente de diagramas, no es necesario que lleven una relación directa con el diagrama de clases de un sistema, es más, podría no aparecer ningún nombre del diagrama de clases en un diagrama de comportamiento.

No son diagramas estructurales, son diagramas conceptuales de manejo, de flujo y de la secuencia de un programa, por lo que es muy posible que dependiendo de la profundidad y extensión de un diagrama no aparezcan algunas de las entidades definidas en el diagrama de clases.

Todos los diagramas de comportamiento se utilizan de un modo jerárquico, es decir, en principio, sea el diagrama que sea, se realiza un diagrama sencillo con pocas "etapas", donde vemos el funcionamiento del sistema, separándolo en diferentes secciones que posteriormente tendrán un diagrama más detallado y específico.



# Diagramas de comportamiento

## Diagramas de Actividad

Representan los flujos de trabajo del sistema desde su inicio hasta el fin con las operaciones y componentes del sistema.

Este tipo de diagramas tienen un gran parecido con los clásicos diagramas de flujo que hemos visto con anterioridad y con una notación muy similar. Los diagramas de actividad tienen unas características muy concretas y restrictivas, se componen de tres elementos:

- ▶ estados
- ▶ transiciones
- ▶ nodos.





# Diagramas de comportamiento

## Diagramas de Actividad

### Estados

- Se representan como un rectángulo con los bordes redondeados.
- Definen los diferentes estados o etapas por las que pasa la ejecución del programa

### Transiciones

- Se representan como una flecha unidireccional.
- Son líneas de conexión que enlazan estados entre sí con una dirección.







### Nodos

- Existen diferentes tipos de nodos. Decisión, Barras de sincronización, nodo inicial y nodo final.
- Los de decisión definen caminos alternativos.
- Las barras de sincronización definen actividades que ocurren de manera asíncrona.
- Los nodos iniciales y finales son únicos y deben existir en el diagrama. Indican el estado inicial y final del flujo de trabajo.



# Diagramas de comportamiento

## Diagramas de Actividad

Símbolo	Significado
	Nodo inicial
	Estado/actividad
	Nodo de decisión
	Transición
	Barra de sincronización
	Nodo final





# Diagramas de comportamiento

## Diagramas de Actividad

Las reglas son muy sencillas:

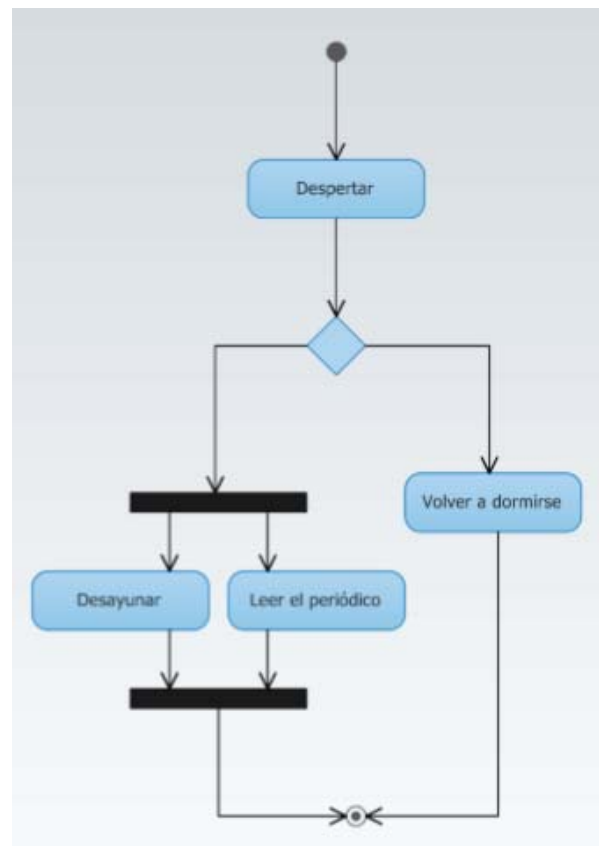
- ▶ Siempre debe haber un único estado inicial y un único estado final
- ▶ Todas las operaciones, transiciones y procesos ocurren entre esos dos puntos.
- ▶ Las transiciones se realizan entre estados y pueden tener nodos de por medio.
- ▶ Los nodos de bifurcación se usan para representar flujos alternativos
- ▶ Las barras de sincronización se utilizan para marcar el inicio de flujos de actividades en paralelo. Debe existir una barra de sincronización al inicio de los flujos en paralelo y otra cuando termina el flujo de actividades en paralelo.





# Diagramas de comportamiento

## Diagramas de Actividad



# Diagramas de comportamiento

## Diagramas de Casos de Uso

Los diagramas de casos de uso representan cómo interactúan los diferentes actores en un sistema para cada caso de uso. Es decir, definen qué acciones puede realizar cada actor dentro de un sistema.

Definiciones:

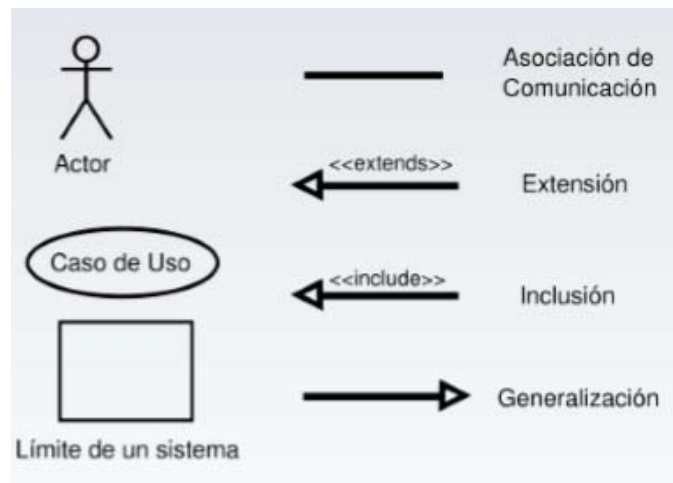
- ▶ **Actor:** Representa un usuario, organización o sistema externo que interactúa con la aplicación o el sistema.
- ▶ **Caso de uso:** Representa las acciones realizadas por uno o varios actores para conseguir un objetivo determinado.
- ▶ **Asociación:** Indica que un actor forma parte de un caso de uso.
- ▶ **Subsistema o componente:** El sistema o la aplicación en los que se está trabajando (o parte de él). Puede ser cualquier cosa, desde una red de gran tamaño hasta una única clase de una aplicación.



# Diagramas de comportamiento

## Diagramas de Casos de Uso

Representación:





# Diagramas de comportamiento

## Diagramas de Casos de Uso

Las relaciones que se pueden realizar entre casos de uso y los actores son realmente muy parecidas, son del mismo tipo pero restrictivas entre sí:

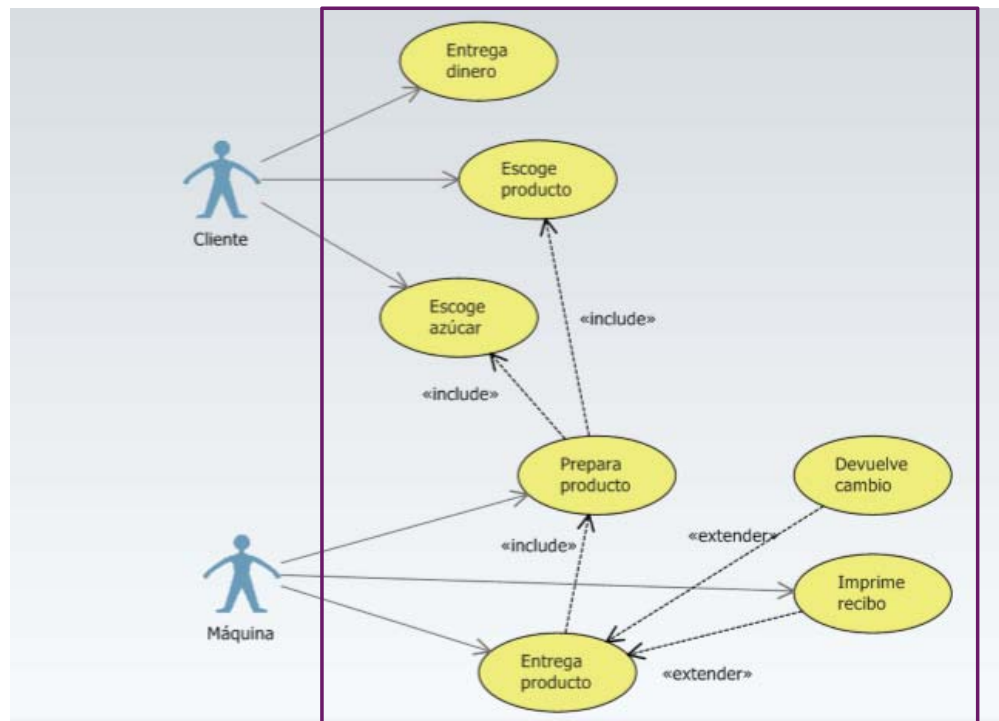
- ▶ **Extensión:** Un caso de uso de extensión agrega objetivos y pasos al caso de uso extendido. Las extensiones solamente funcionan en ciertas condiciones. El caso de uso extendido se encuentra en el extremo con la punta de flecha.
- ▶ **Inclusión:** Un caso de uso de inclusión llama o invoca al caso de uso incluido. La inclusión se usa para mostrar cómo se divide un caso de uso en pasos más pequeños. El caso de uso incluido se encuentra en el extremo con la punta de flecha.
- ▶ **Generalización:** Es un modo de representar la herencia de casos de uso, es decir, un caso de uso puede ser genérico, pero tener formas más específicas del mismo.
- ▶ **Límite de un sistema:** Se utiliza para separar los diferentes sistemas dentro de un diagrama de casos de uso. En caso de que solo haya un sistema, no es necesario establecer el límite de un sistema.



# Diagramas de comportamiento

## Diagramas de Casos de Uso

Diagrama de casos de uso de una máquina expendedora de café

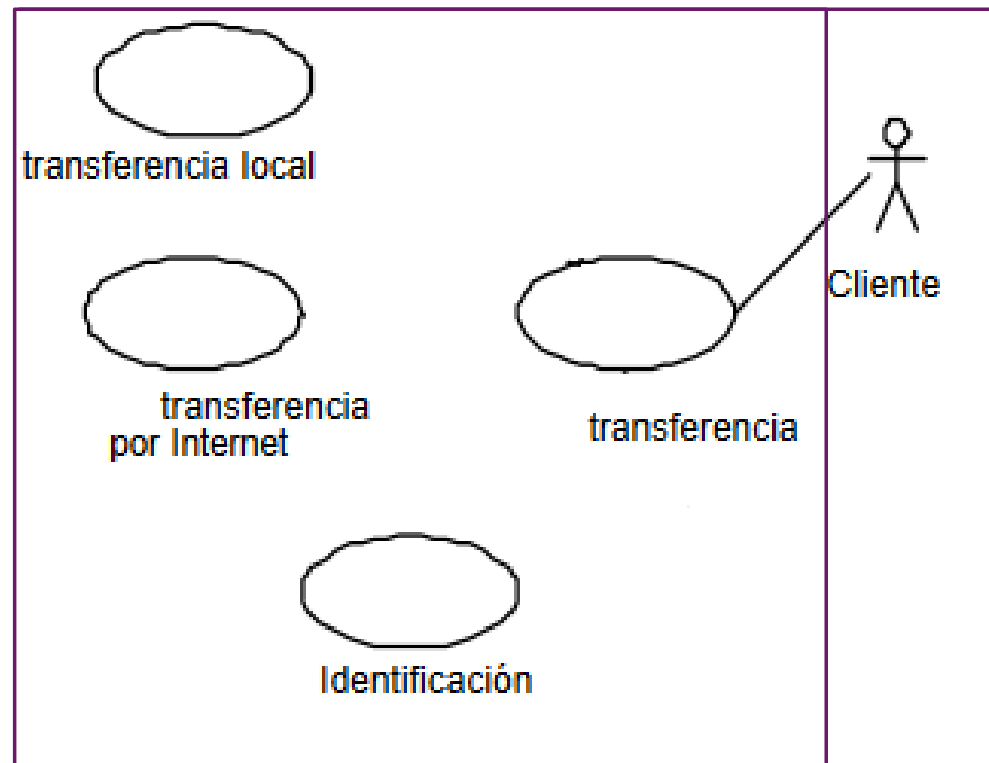




# Diagramas de comportamiento

## Diagramas de Casos de Uso

Ejemplo 1: rellenar los tipos de relación de los casos de uso para el sistema de pago:

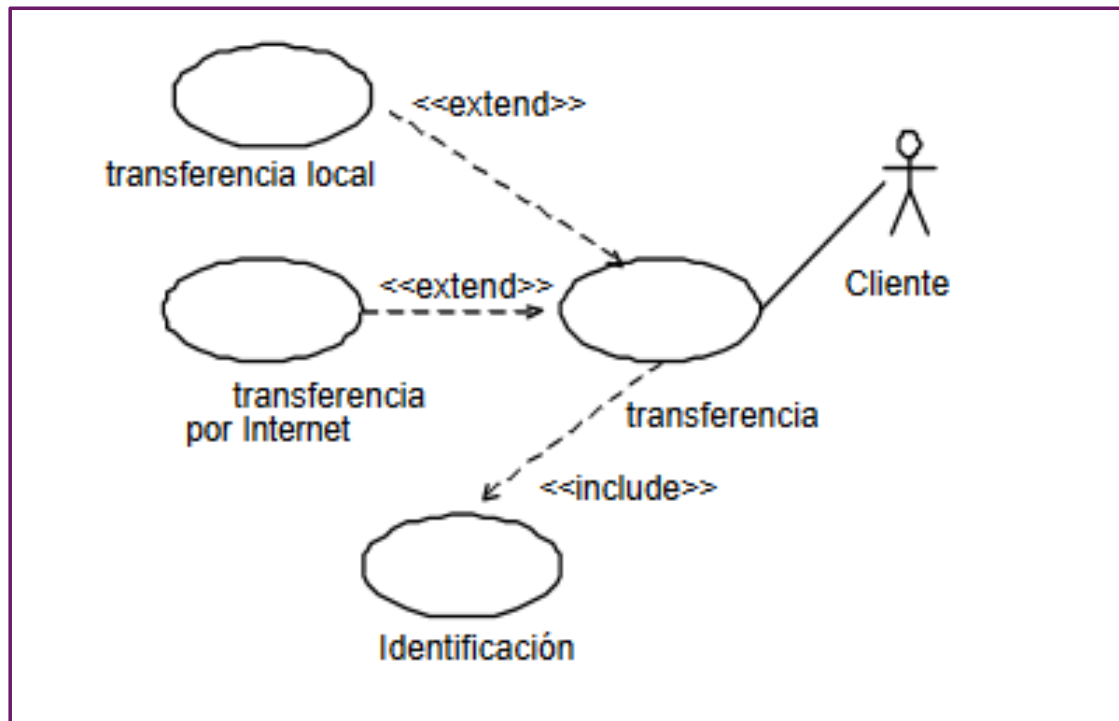




# Diagramas de comportamiento

## Diagramas de Casos de Uso

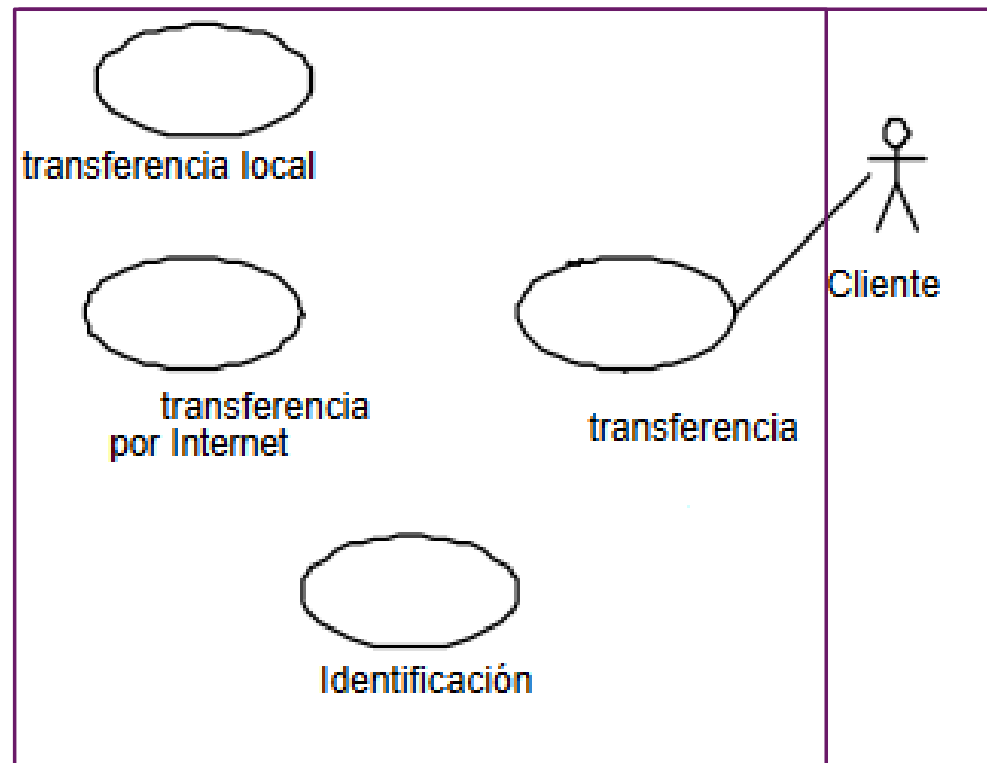
Ejemplo 1: rellenar los tipos de relación de los casos de uso:



# Diagramas de comportamiento

## Diagramas de Casos de Uso

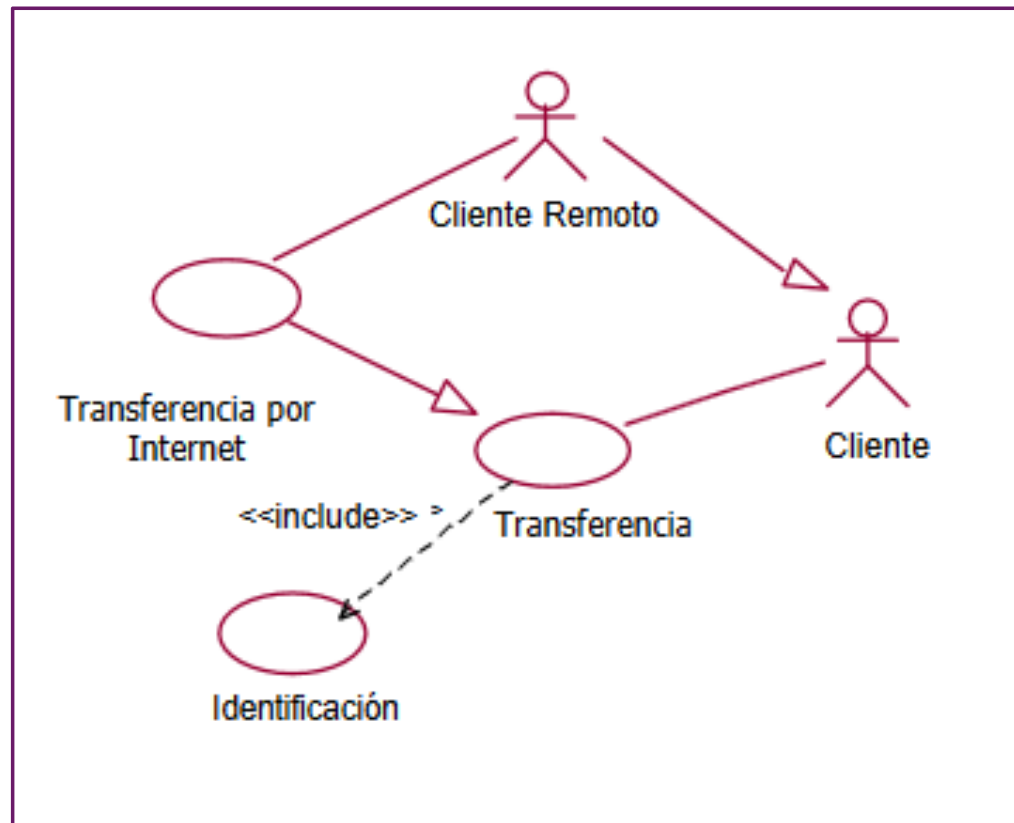
Ejemplo 1b: añade al usuario CLIENTE\_REMOTO.



# Diagramas de comportamiento

## Diagramas de Casos de Uso

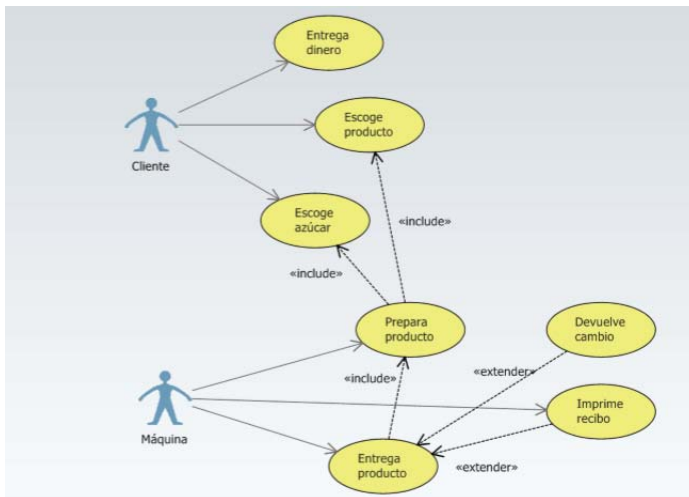
Ejemplo 1b: añade al usuario CLIENTE\_REMOTO.





# Diagramas de comportamiento

## Diagramas de Casos de Uso



Devuelve cambio e imprime recibo extienden a Entrega producto porque son casos de uso que se realizan cuando se entrega el producto, mientras que los casos de uso escoger azúcar y escoger producto son parte del caso de uso de preparar producto.

Visto de otro modo, la máquina necesita que se haya escogido el producto y la cantidad de azúcar para preparar el producto, y la máquina devuelve el cambio e imprime el recibo cuando entrega el producto.



# DFD y CASOS DE USO

## SIMILITUDES Y DIFERENCIAS

- ▶ Un **CASO DE USO** es una función atómica (servicio o transición) ofrecida por el sistema al entorno (ACTORES), mientras que un DFD puede ser detallado en un hijo.
- ▶ Ambos modelan una funcionalidad del sistema pero:
  - ▶ En caso de USO interesa la interacción actor-sistema
  - ▶ En un proceso la transformación que se hace de los flujos de entrada para producir flujos de salida
- ▶ Las relaciones de extensión y de generalización de los casos de uso no tienen correspondencia con los DFDs.
- ▶ Un caso de uso se concibe como una visión externa del sistema, a perspectiva del actor, mientras que un DFD puede mostrar descomposición funcional del sistema



# Diagramas de comportamiento

## Diagramas de Casos de Uso

- ▶ BIBLIOGRAFIA
- ▶ [https://es.wikipedia.org/wiki/Caso\\_de\\_uso](https://es.wikipedia.org/wiki/Caso_de_uso)
- ▶ <https://www.youtube.com/watch?v=ab6eDdwS3rA> (video LucidaChart)





# Diagramas de SECUENCIAS



# Diagramas de comportamiento

## Diagramas de Secuencia

Los diagramas de secuencia modelan la secuencia lógica a través del tiempo de los mensajes entre instancias.

El diagrama de secuencia se estructura mediante líneas de vida, que a su vez representan a un participante en el sistema, ya sea un actor o cualquier otro elemento que participe en el transcurso secuencial de nuestro programa. La parte más importante del diagrama es la línea de vida, esa línea representa una secuencia, una cronología que nos permite visualizar con un rápido vistazo las interacciones, mensajes y participantes, así como el número de ellos a través de toda la vida del programa, desde que se ejecuta hasta que se cierra.



# Diagramas de comportamiento

## Diagramas de Secuencia

- ▶ En un diagrama de secuencias los objetos se colocan de izquierda a derecha en la parte superior.
- ▶ Cada línea de vida de un objeto es una línea discontinua que se desplaza hacia abajo partiendo del objeto.
- ▶ Una línea continua con una punta de flecha, conecta a una línea de vida con otra (en ocasiones los mensajes de respuesta se representan con líneas discontinuas) y representa un mensaje de un objeto a otro.
- ▶ El tiempo se inicia en la parte superior y continúa hacia abajo





# Diagramas de comportamiento

## Diagramas de Secuencia

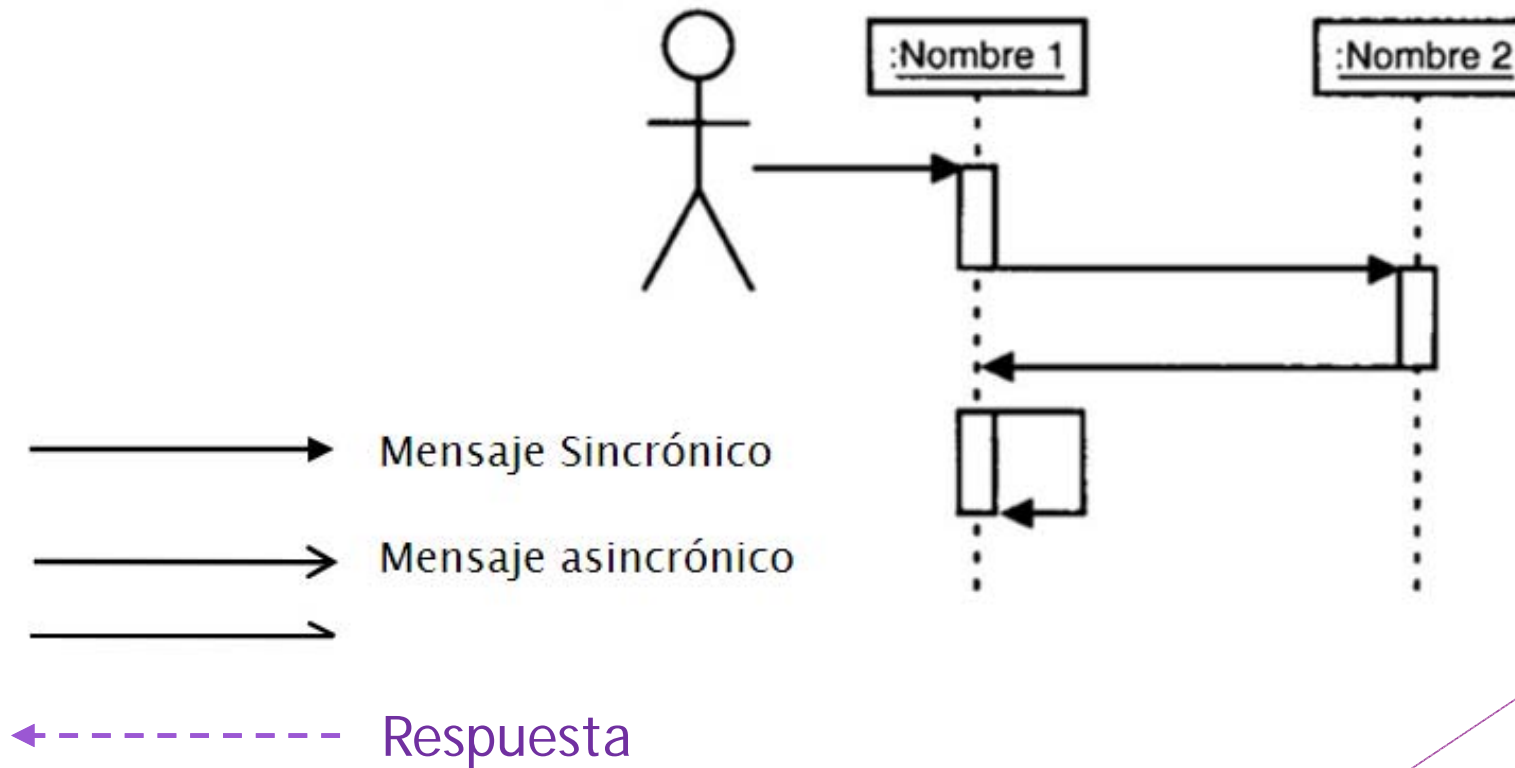
Los diagramas de secuencia muestran dos tipos de mensajes:

- ▶ **Síncronos**: Se corresponden con llamadas a métodos del objeto que recibe el mensaje. El objeto que envía el mensaje queda bloqueado hasta que termina la llamada. Este tipo de mensajes se representan con flechas con la cabeza llena.
- ▶ **Asíncronos**: Estos mensajes terminan inmediatamente, y crean un nuevo hilo de ejecución dentro de la secuencia. Se representan con flechas con la cabeza abierta o con media cabeza. Al igual que los mensajes síncronos, también se representa la respuesta con una flecha discontinua (también podemos encontrar representaciones que utilicen flecha continua para las respuestas).



# Diagramas de comportamiento

## Diagramas de Secuencia



# Diagramas de comportamiento

## Diagramas de Secuencia

### Ejemplo:

Suponga que el usuario de una GUI (Interfaz Gráfica de Usuario) presiona una tecla alfanumérica; si asumimos que utiliza una aplicación como un procesador de textos, el carácter correspondiente deberá de aparecer inmediatamente en la pantalla ¿Qué ocurre para que esto suceda?

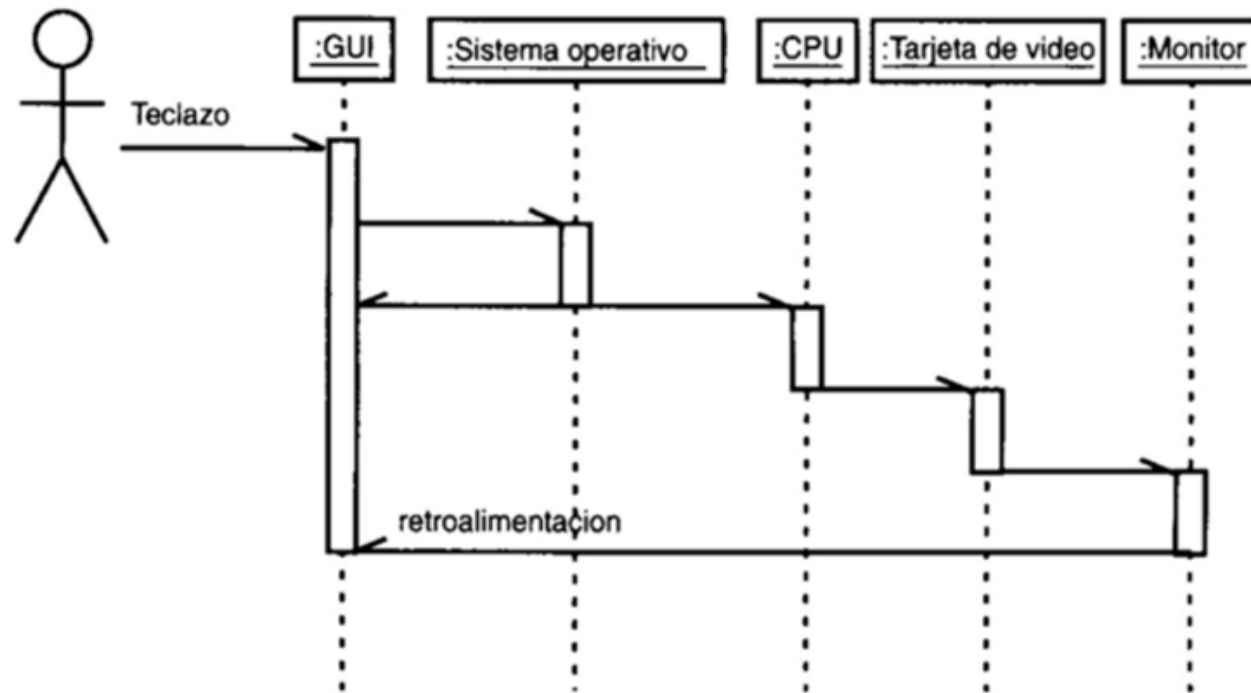
- ▶ El SO notifica al CPU.
- ▶ El SO actualiza la GUI.
- ▶ El CPU notifica a la tarjeta de video.
- ▶ La tarjeta de video envía un mensaje al monitor.
- ▶ El monitor presenta el carácter alfanumérico en la pantalla, con lo que se hará evidente al usuario.





# Diagramas de comportamiento

## Diagramas de Secuencia



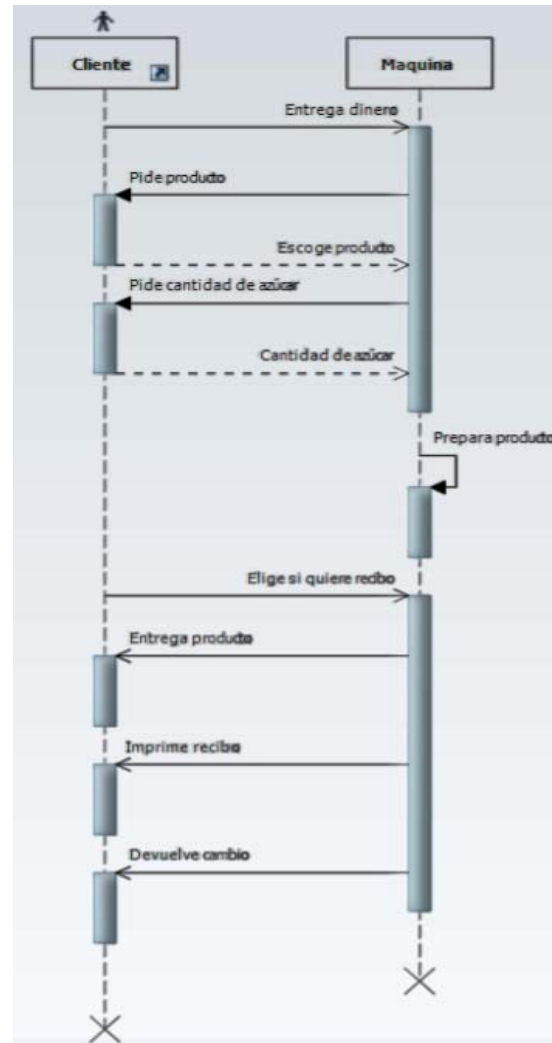
# Diagramas de comportamiento

## Diagramas de Secuencia

En el siguiente diagrama vemos que se han usado los mensajes asíncronos y los mensajes síncronos, básicamente observamos que las opciones que no requieren respuesta se han representado con llamadas asíncronas, aunque no se tiene por qué corresponder de ese modo siempre.

Básicamente, la máquina tiene tres períodos de proceso: cuando le pide los datos al cliente, cuando procesa la solicitud y cuando devuelve los resultados (en este caso un café, el cambio y un recibo) al cliente.

Respuestas con flechas discontinuas.



# Diagramas de comportamiento

## Diagramas de Secuencia

Ejemplo: La máquina de gaseosa

Asumamos que en la máquina de gaseosa hay tres objetos que realizan la tarea:

- ▶ La fachada, la interfaz que la máquina de gaseosa muestra al usuario
- ▶ el registrador de dinero, que lo recolecta
- ▶ el dispensador, que entrega la gaseosa.





# Diagramas de comportamiento

## Diagramas de Secuencia de instancias

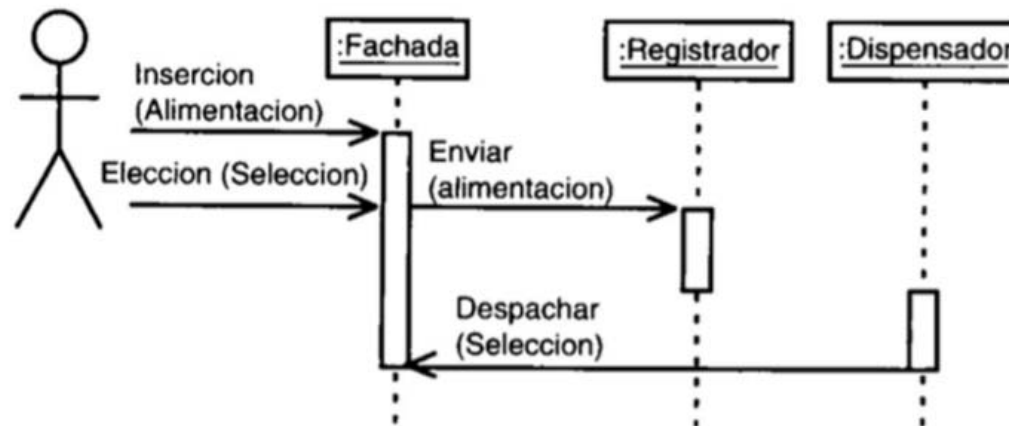
Ejemplo: La máquina de gaseosa

- ▶ El cliente inserta el dinero en la alcancía que se encuentra en la fachada de la máquina.
- ▶ El cliente hace su elección.
- ▶ El dinero viaja hacia el registrador.
- ▶ El registrador verifica si la gaseosa elegida está en el dispensador.
- ▶ Dado que es el mejor escenario, asumamos que si hay gaseosas, y el registrador actualiza su reserva de efectivo.
- ▶ El registrador hace que el dispensador entregue la gaseosa en la fachada de la máquina.



# Diagramas de comportamiento

## Diagramas de Secuencia de instancias



- ▶ Este diagrama de secuencias modela tan sólo el mejor escenario del caso de uso “Comprar Gaseosa”. Por lo tanto, es un **DIAGRAMA DE SECUENCIAS DE INSTANCIAS**.





# Diagramas de comportamiento

## Diagramas de Secuencia genérico

Ejemplo: La máquina de gaseosa

El caso de uso “Comprar gaseosa” podemos tener dos escenarios alternos:

- ▶ Uno de ellos se referiría al hecho de que la máquina no tuviera la gaseosa seleccionada
- ▶ El otro cuando el cliente no cuente con el dinero exacto.

Si se tomara en cuenta todos los escenarios de un caso de uso al momento de crear un diagrama de secuencias, se trataría de un diagrama de secuencias genérico.





# Diagramas de comportamiento

## Diagramas de Secuencia genérico

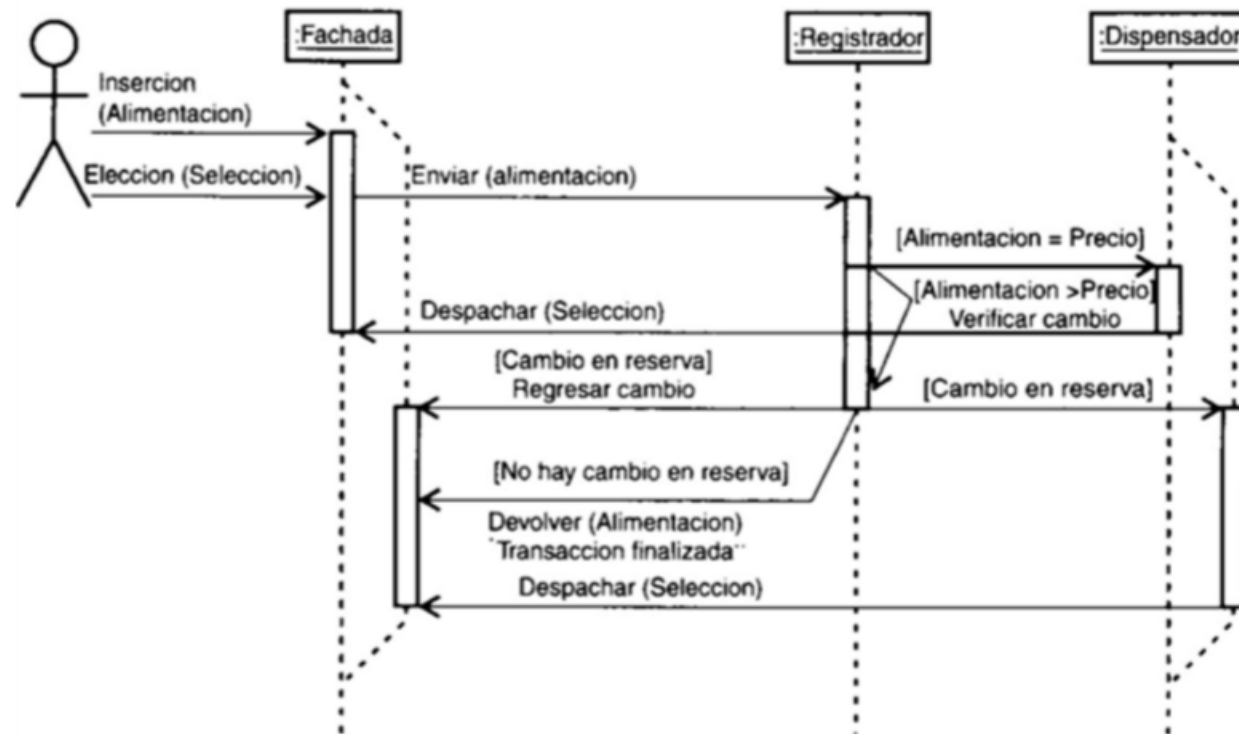
Ejemplo: La máquina de gaseosa para el escenario relacionado con: "Monto Incorrecto":

- ▶ El registrador verifica si la alimentación del usuario concuerda con el precio de la gaseosa.
- ▶ Si el monto es mayor que el precio, el registrador calcula la diferencia y verifica si cuenta con cambio.
- ▶ Si se puede devolver la diferencia, el registrador devuelve el cambio al cliente y todo transcurre como antes.
- ▶ Si la diferencia no se encuentra en la reserva del cambio, el registrador regresará el monto alimentado y mostrará un mensaje que indique al cliente que inserte el monto exacto.
- ▶ Si la cantidad insertada es menor que el precio, el registrador no hace nada y la máquina esperará más dinero.



# Diagramas de comportamiento

## Diagramas de Secuencia genérico





# Diagramas de comportamiento

## Diagramas de Secuencia genérico

Ejemplo: La máquina de gaseosa para el escenario "Sin gaseosa":

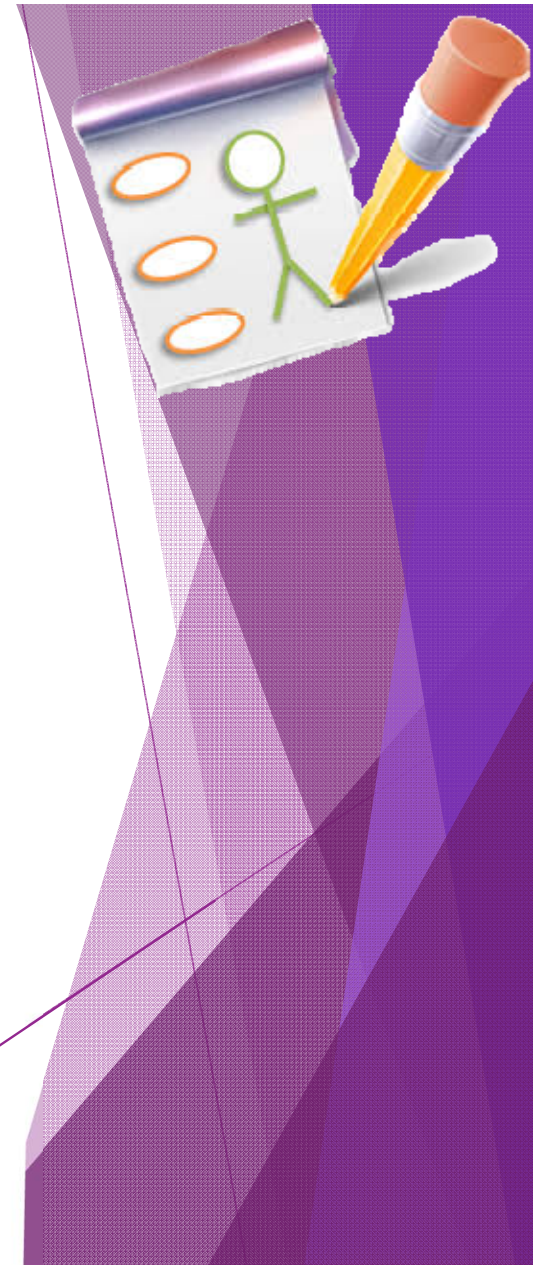
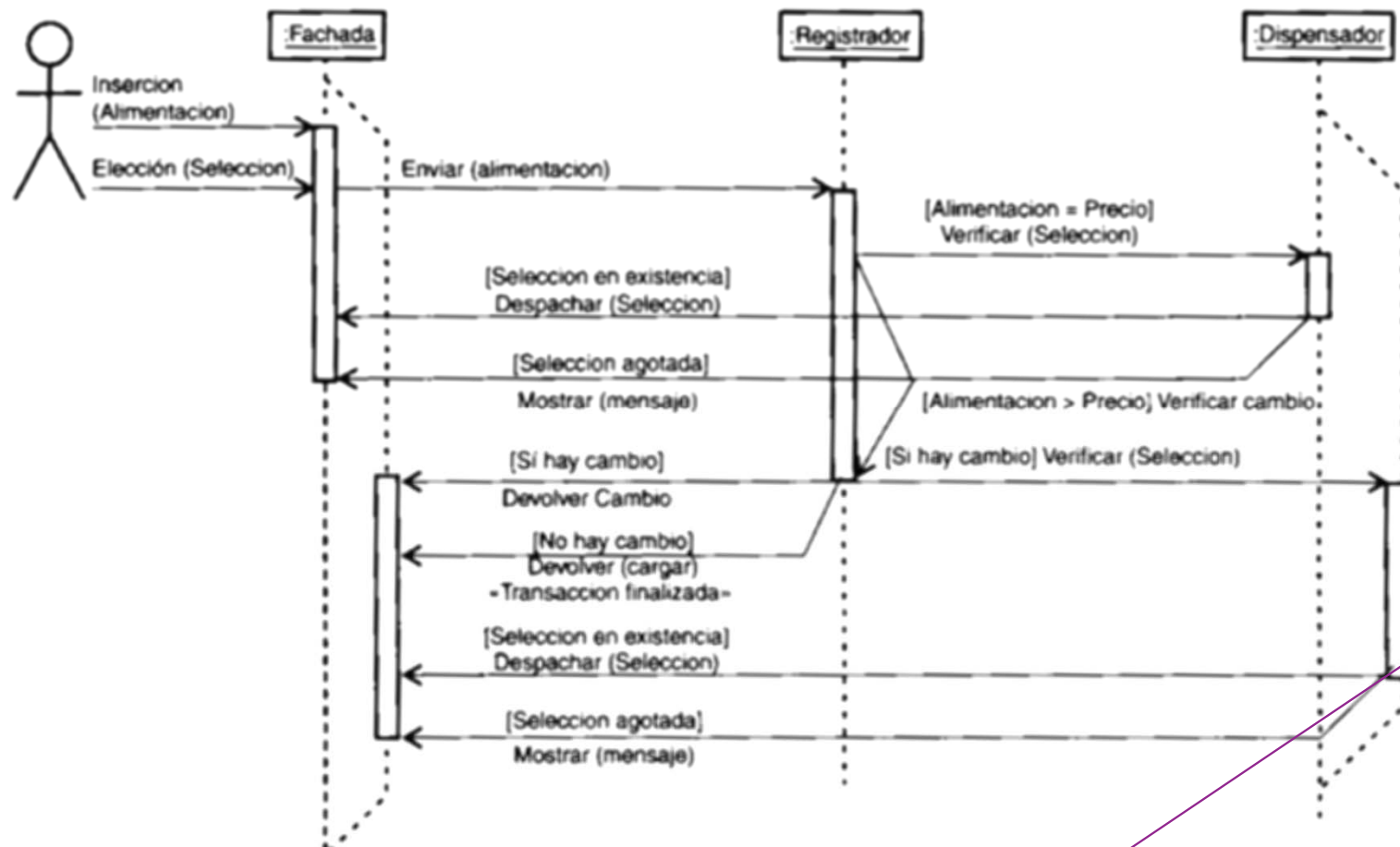
- ▶ Una vez que el cliente elige una marca agotada, la máquina mostrará un mensaje de "agotado".
- ▶ La maquina mostrará un mensaje que solicitará al cliente que haga otra elección.
- ▶ El cliente tendrá la opción de oprimir un botón para que se le regrese su dinero.
- ▶ Si el cliente elige una marca en existencia, todo procederá como en el mejor escenario, si el monto insertado es el correcto. Si no lo es, la máquina seguirá por el escenario del "Monto incorrecto".
- ▶ Si el cliente elige otro marca agotada, el proceso se repetirá hasta que el cliente elija una marca en existencia o presione un botón que le regrese su dinero.





# Diagramas de comportamiento

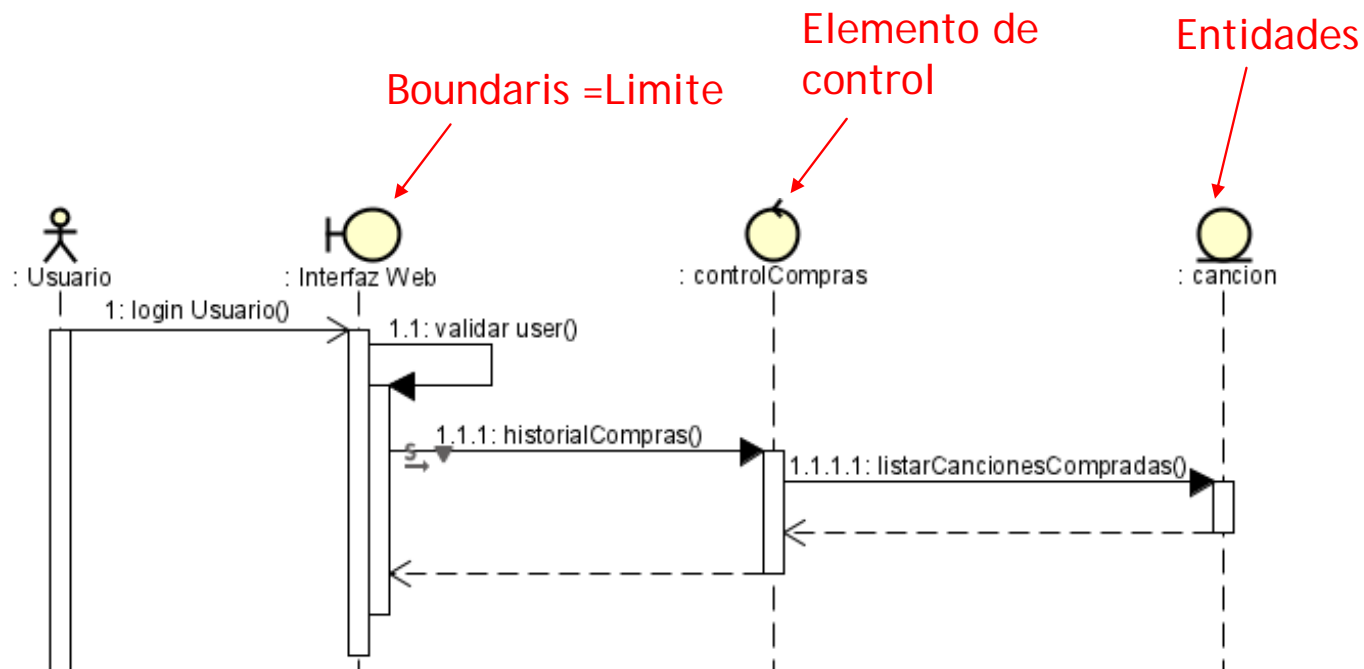
## Diagramas de Secuencia genérico



# Diagramas de comportamiento

## Diagramas de Secuencia con tareas especiales

- ▶ ASTAH, DRAW.IO: Líneas de VIDA



Elementos de Control: Permiten la comunicación entre interfaces y entidades.



# Diagramas de COLABORACIONES





# Diagramas de comportamiento

## Diagrama de colaboraciones

Los diagramas de secuencia son un tipo de diagrama de interacción, que muestran cómo se comunican los objetos entre sí. Existen dos tipos de diagramas de interacción:

- ▶ **Diagrama de Secuencia:** es más adecuado para observar la perspectiva cronológica de las interacciones
- ▶ **Diagrama de Colaboración:** ofrece una mejor visión espacial mostrando los enlaces de comunicación entre objetos

El Diagrama de Colaboración puede obtenerse automáticamente a partir del correspondiente Diagrama de Secuencia (o viceversa)



# Diagramas de comportamiento

## Diagrama de colaboraciones

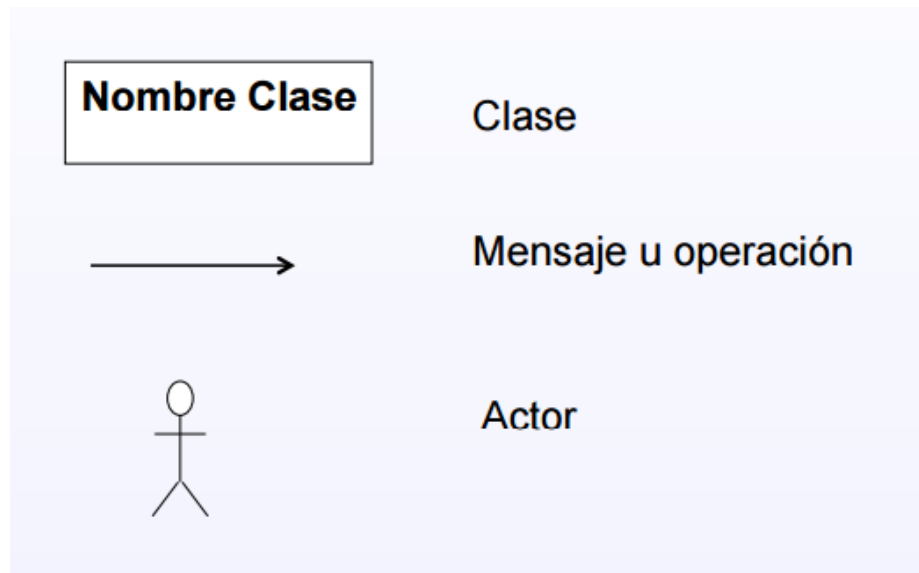
- ▶ El Diagrama de Colaboración modela la interacción entre los objetos de un Caso de Uso
- ▶ Los objetos están conectados por enlaces (links) en los cuales se representan los mensajes enviados acompañados de una flecha que indica su dirección
- ▶ El Diagrama de Colaboración ofrece una mejor visión del escenario cuando el analista está intentando comprender la participación de un objeto en el sistema
- ▶ Son útiles en la fase exploratoria para identificar objetos
- ▶ La distribución de los objetos en el diagrama permite observar adecuadamente la interacción de un objeto con respecto de los demás
- ▶ La estructura estática viene dada por los enlaces; la dinámica por el envío de mensajes por los enlaces



# Diagramas de comportamiento

## Diagrama de colaboraciones

### Representación gráfica

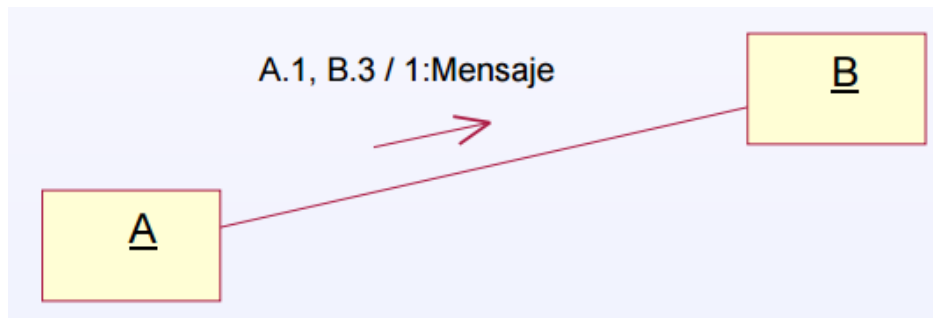




# Diagramas de comportamiento

## Diagrama de colaboraciones. Mensajes

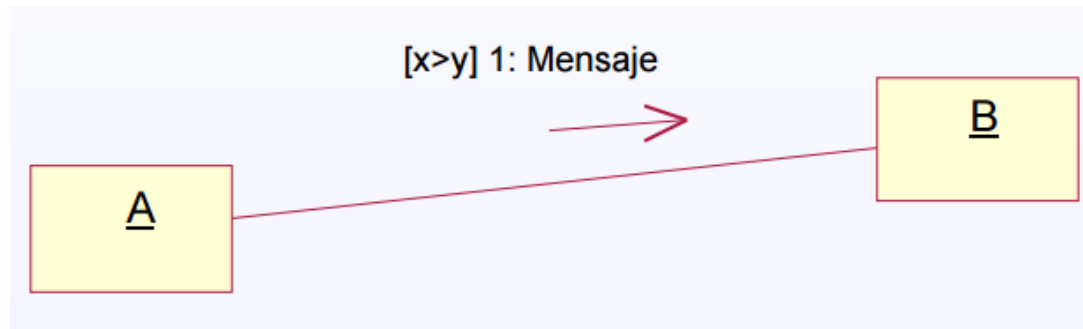
- Un mensaje desencadena una acción en el objeto destinatario
- Un mensaje se envía si han sido enviados los mensajes de una lista (sincronización):



# Diagramas de comportamiento

## Diagrama de colaboraciones. Mensajes

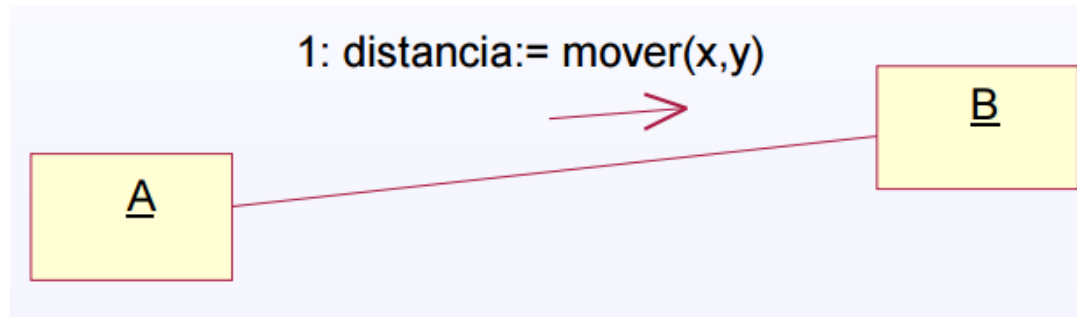
- Un mensaje se envía de manera condicionada:



# Diagramas de comportamiento

## Diagrama de colaboraciones. Mensajes

- Un mensaje que devuelve un resultado:

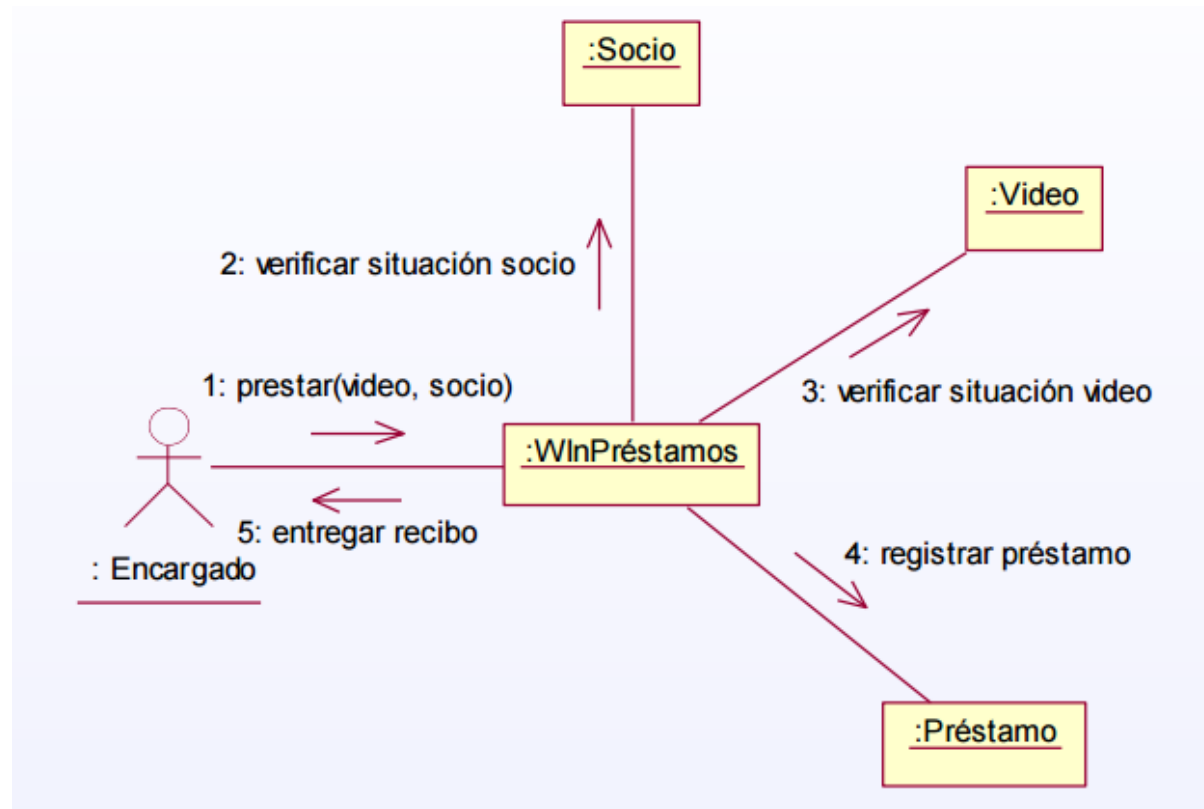




# Diagramas de comportamiento

Diagrama de colaboraciones.

Ejemplo:



# Diagramas de comportamiento

## Diagrama de estado

También llamados Diagrama de Transición de Estados, estos diagramas representan la descripción del comportamiento de un sistema, este describe todos los estados posibles en los que puede estar un objeto específico a lo largo de su ciclo de vida, representándose así su cambio de estado en el tiempo como resultado de los eventos que llegan a él.



# Diagramas de comportamiento

## Diagrama de estado

### Componentes de un diagrama de estado

Este diagrama esta compuesto por:

- **Estados:** Es aquel que influye en el comportamiento y evolución del sistema, los estados siempre han de pertenecer a una clase y representa un resumen de los valores y atributos que puede tener la clase, en si un estado UML describe el estado interno de un objeto de una clase particular. No todos los cambios en los atributos de un objeto deben de estar representados por estados, solo aquellos en lo que el cambio afecta significativamente su comportamiento. Los **Tipos de Estado** son:
  - **Inicio:** Es el estado inicial en el que se inicia el objeto en su ciclo de vida, ningun evento puede retornar un objeto a este estado. Gráficamente esta representado con un circulo negro.





# Diagramas de comportamiento

## Diagrama de estado

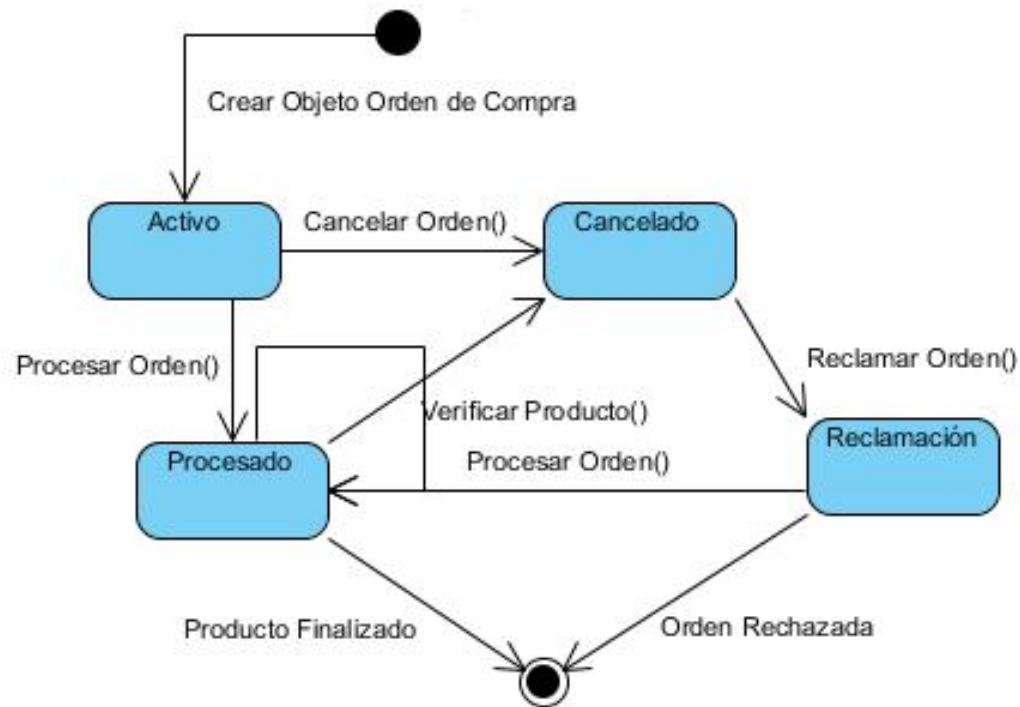
- ▶ **Fin:** Es el estado final en el que queda un objeto al final de su ciclo de vida, ningun evento puede sacar a un objeto de este estado. Gráficamente esta representado con un circulo negro rodeado de otro circulo.
- ▶ **Estado:** Son los diferentes estados por lo que puede pasar un objeto a lo largo de su ciclo de vida, de ellos se puede salir, quedarse en el y retornar. Gráficamente esta representado por un rectangulo.
- ▶ **Eventos:** Son aquellos que dan lugar a un cambio en el comportamiento del sistema o a un momento significativo en su evolución, por ejemplo un metodo de una clase.
- ▶ **Transiciones:** Son las lineas de comunicación, lo que une un estado con otro, ella esta compuesta por los eventos y la accion a ejecutar. La representación gráfica es una flecha en linea con la punta abierta.



# Diagramas de comportamiento

## Diagrama de estado

### Ejemplo

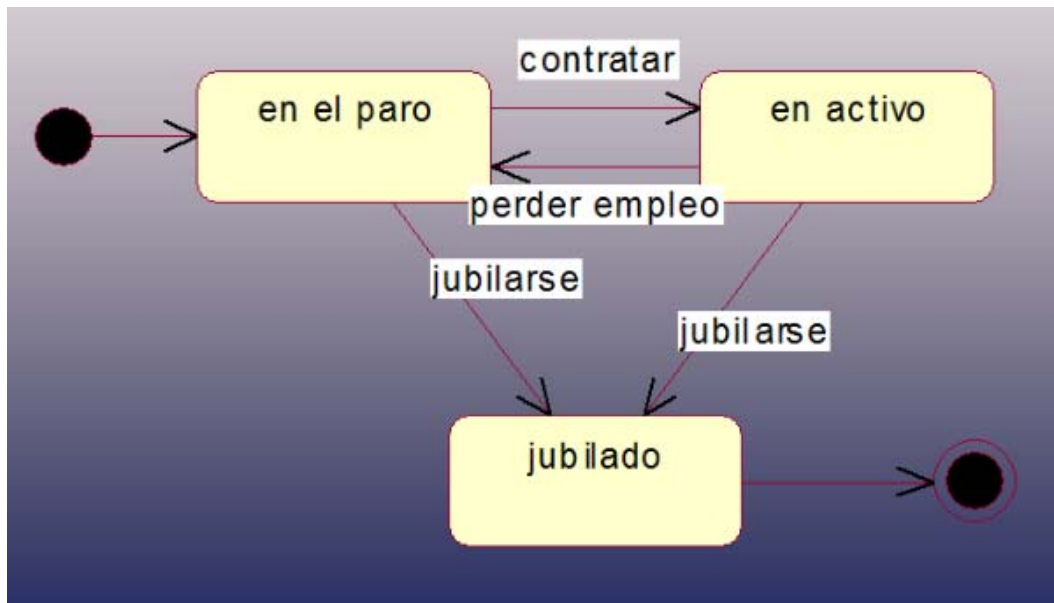




# Diagramas de comportamiento

## Diagrama de estado

### Ejemplo





# HERRAMIENTAS DE MODELADO UML

UMLet: código abierto y libre distribución. Interfaz sencillo de utilizar.

ArgoUML: se distribuye bajo licencia de Eclipse. Soporta diagramas UML 1.4 y genera código para JAVA y C++.

Admite ingeniería directa e inversa.

Astah: Herramienta propietaria, gratuita para estudiantes acreditados. Permite crear proyectos con UML 2.0.

Visual Paradigm for UML (VP-UML): incluye versión de uso no comercial que se distribuye con registro. Capaz de diseñar Bases de datos con ingeniería inversa. Compatible con IDEs de Eclipse, NetBeans, Visual Studio y IntelliJDEA.

