

1. Modelo de Clases
2. Introducción
3. Elementos
4. Clase
  - a. Atributos:
  - b. Métodos:
5. Relaciones entre Clases:
  - a. Herencia (Especialización/Generalización):
  - b. Agregación:
  - c. Asociación:
  - d. Dependencia o Instanciación (uso):
6. Casos Particulares:
  - a. Clase Abstracta:
  - b. Clase parametrizada:
7. Casos de Uso (Use Case)
8. Introducción
9. Elementos
10. Actor:
  - a. Asociación
11. Relaciones:
  - a. Dependencia o Instanciación
12. Introducción
13. Elementos
14. Clase
  - a. Atributos:
  - b. Métodos:
15. Relaciones entre Clases:
  - a. Herencia (Especialización/Generalización):
  - b. Agregación:
  - c. Asociación:
  - d. Dependencia o Instanciación (uso):
16. Casos Particulares:
  - a. Clase Abstracta:
  - b. Clase parametrizada:
17. Casos de Uso (Use Case)
18. Introducción
19. Elementos
20. Actor:
  - a. Asociación
21. Relaciones:
  - a. Dependencia o Instanciación
22. Línea de vida de un objeto
23. Activación
24. Tiempos de transición
25. Mensaje
26. Destrucción de un objeto
27. Métodos recursivos

# Modelo de Clases

## Introducción

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.

Un diagrama de clases esta compuesto por los siguientes elementos:

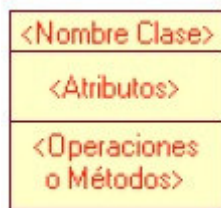
- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

## Elementos

- **Clase**

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

En UML, una clase es representada por un rectángulo que posee tres divisiones:



En donde:

- **Superior**: Contiene el nombre de la Clase
- **Intermedio**: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
- **Inferior**: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

Ejemplo:

Una Cuenta Corriente que posee como característica:

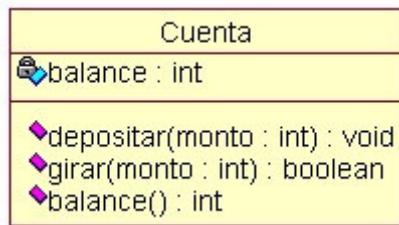
- Balance

Puede realizar las operaciones de:

- Depositar

- Girar
- y Balance


El diseño asociado es:




Atributos y Métodos:

- **Atributos:**

Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

- **public** (+, 
  - **Métodos:**


Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

- **public** (+, 
  - **Relaciones entre Clases:**

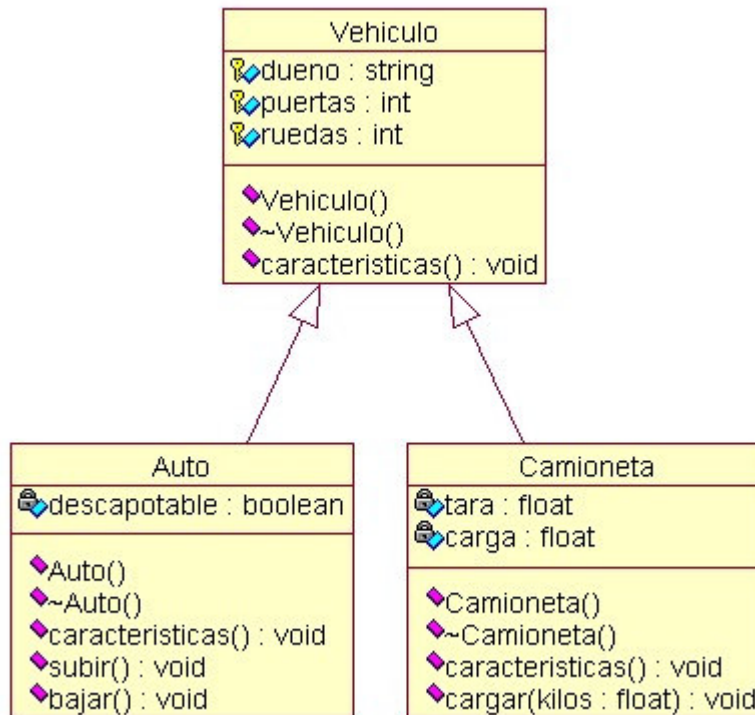
Ahora ya definido el concepto de Clase, es necesario explicar como se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes).

Antes es necesario explicar el concepto de cardinalidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

- o **uno o muchos:** 1..\* (1..n)
- o **0 o muchos:** 0..\* (0..n)
- o **número fijo:** m (m denota el número).

iv. **Herencia (Especialización/Generalización):** 

Indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected), ejemplo:



En la figura se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (Precio, VelMax, etc) además posee algo particular que es Descapotable, en cambio Camión también hereda las características de Vehículo (Precio, VelMax, etc) pero posee como particularidad propia Acoplado, Tara y Carga.

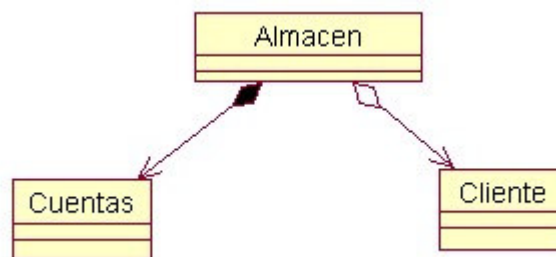
Cabe destacar que fuera de este entorno, lo único "visible" es el método Características aplicable a instancias de Vehículo, Auto y Camión, pues tiene definición pública, en cambio atributos como Descapotable no son visibles por ser privados.

v. **Agregación:** 

Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

- **Por Valor: (Rombo relleno)** Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
- **Por Referencia:** Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).

Un Ejemplo es el siguiente:



En donde se destaca que:

- Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).
- Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

vi. **Asociación:** 

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo:



Un cliente puede tener asociadas muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

vii. **Dependencia o Instanciación (uso):** ----->

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada.

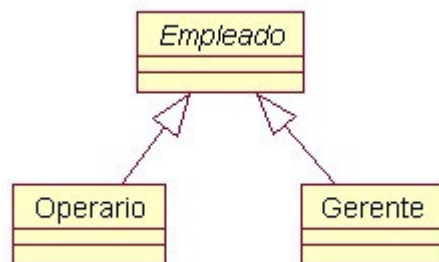
El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación grafica que instancia una ventana (la creación del Objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicación):



Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

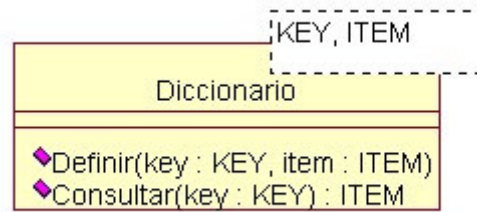
• **Casos Particulares:**

○ **Clase Abstracta:**



Una clase abstracta se denota con el nombre de la clase y de los métodos con letra "itálica". Esto indica que la clase definida no puede ser instanciada pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es definiendo subclasses, que implementan los métodos abstractos definidos.

○ **Clase parametrizada:**



Una clase parametrizada se denota con un subcuadro en el extremo superior de la clase, en donde se especifican los parámetros que deben ser pasados a la clase para que esta pueda ser instanciada. El ejemplo más típico es el caso de un Diccionario en donde una llave o palabra tiene asociado un significado, pero en este caso las llaves y elementos pueden ser genéricos. La genericidad puede venir dada de un Template (como en el caso de C++) o bien de alguna estructura predefinida (especialización a través de clases).

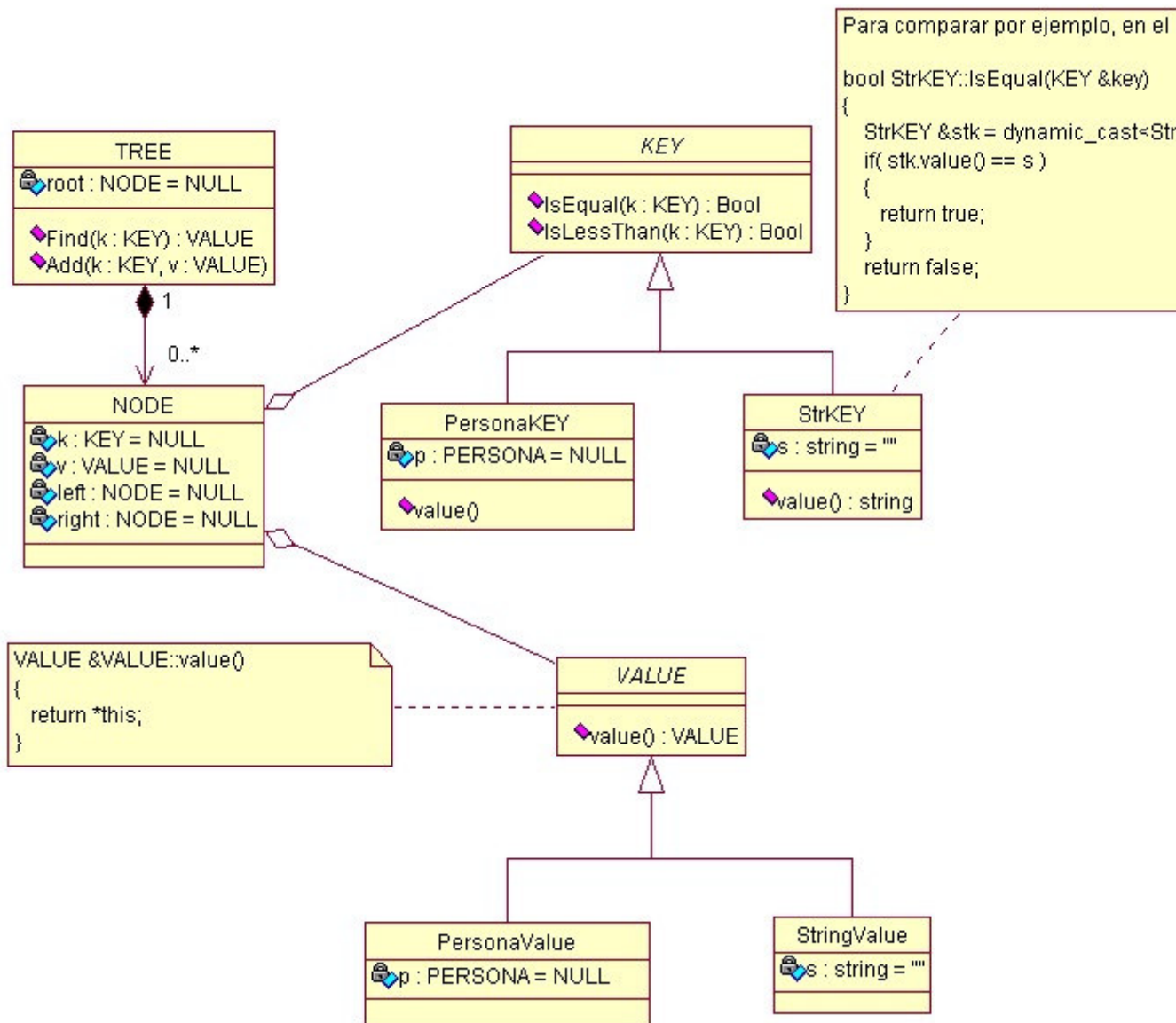
En el ejemplo no se especificaron los atributos del Diccionario, pues ellos dependerán exclusivamente de la implementación que se le quiera dar.

### **Ejemplo:**

Supongamos que tenemos un caso del Diccionario implementado mediante un árbol binario, en donde cada nodo posee:

- key: Variable por la cual se realiza la búsqueda, puede ser genérica.
- item: Contenido a almacenar en el diccionario asociado a "key", cuyo tipo también puede ser genérico.

Para este caso particular hemos definido un Diccionario para almacenar String y Personas, las cuales pueden funcionar como llaves o como item, solo se mostrarán las relaciones para la implementación del Diccionario:



## Casos de Uso (Use Case)

### Introducción

El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos:

- Actor.
- Casos de Uso.
- Relaciones de Uso, Herencia y Comunicación.

### Elementos

- **Actor:**





Una definición previa, es que un **Actor** es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra **rol**, pues con esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

Como ejemplo a la definición anterior, tenemos el caso de un sistema de ventas en que el rol de Vendedor con respecto al sistema puede ser realizado por un Vendedor o bien por el Jefe de Local.

- **Caso de Uso:**



Es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.

- **Relaciones:**

- **Asociación** 

Es el tipo de relación más básica que indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple.

- **Dependencia o Instanciación** 

Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia (se crea). Dicha relación se denota con una flecha punteada.

- **Generalización** 

Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su estereotipo, que puede ser de **Uso** (<<uses>>) o de **Herencia** (<<extends>>).

Este tipo de relación está orientado exclusivamente para casos de uso (y no para actores).

**extends:** Se recomienda utilizar cuando un caso de uso es similar a otro (características).

**uses:** Se recomienda utilizar cuando se tiene un conjunto de características que son similares en más de un caso de uso y no se desea mantener copiada la descripción de la característica.

De lo anterior cabe mencionar que tiene el mismo paradigma en diseño y modelamiento de clases, en donde esta la duda clásica de **usar** o **heredar**.

### Ejemplo:

Como ejemplo esta el caso de una Máquina Recicladora:

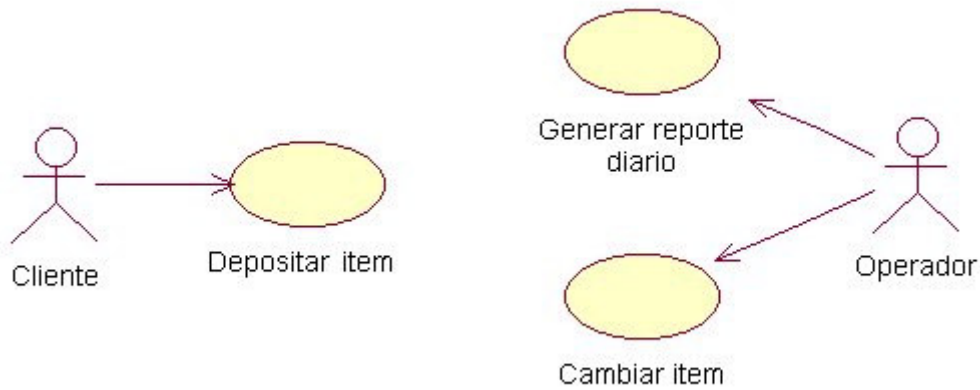
Sistema que controla una máquina de reciclamiento de botellas, tarros y jabas. El sistema debe controlar y/o aceptar:

- Registrar el número de ítemes ingresados.
- Imprimir un recibo cuando el usuario lo solicita:
  - a. Describe lo depositado
  - b. El valor de cada ítem
  - c. Total
- El usuario/cliente presiona el botón de comienzo
- Existe un operador que desea saber lo siguiente:
  - a. Cuantos ítemes han sido retornados en el día.
  - b. Al final de cada día el operador solicita un resumen de todo lo depositado en el día.
- El operador debe además poder cambiar:
  - a. Información asociada a ítemes.
  - b. Dar una alarma en el caso de que:
    - i. Ítem se atora.
    - ii. No hay más papel.

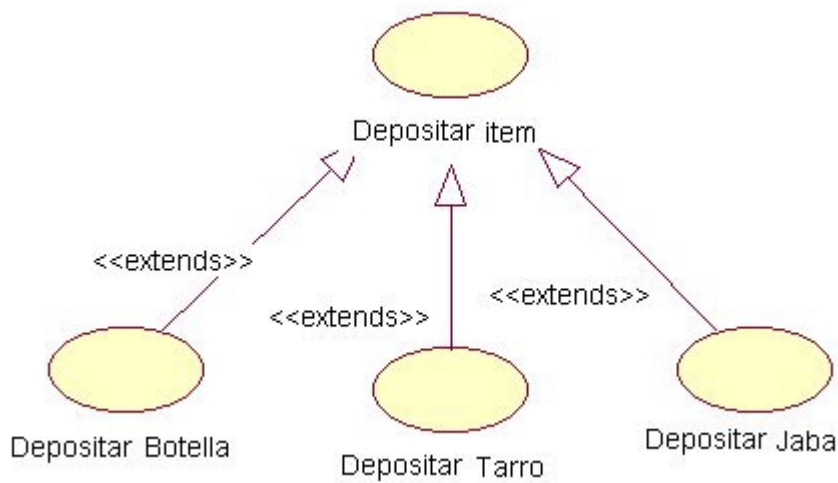
Como una primera aproximación identificamos a los actores que interactúan con el sistema:



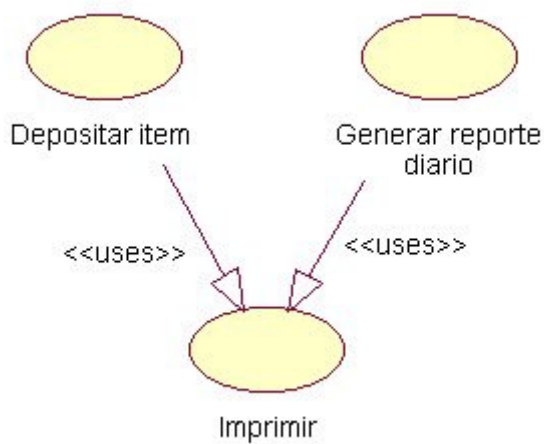
Luego, tenemos que un Cliente puede Depositar Ítemes y un Operador puede cambiar la información de un Ítem o bien puede Imprimir un informe:



Además podemos notar que un item puede ser una Botella, un Tarro o una Jaba.



Otro aspecto es la impresión de comprobantes, que puede ser realizada después de depositar algún item por un cliente o bien puede ser realizada a petición de un operador.



Entonces, el diseño completo del diagrama Use Case es:

# Diagrama de Interacción

## Introducción

El diagrama de interacción, representa la forma en como un Cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente.

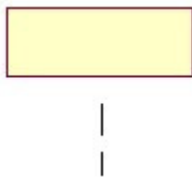
Dicho diagrama puede ser obtenido de dos partes, desde el Diagrama Estático de Clases o el de Casos de Uso (son diferentes).

Los componentes de un diágrama de interacción son:

- Un Objeto o Actor.
- Mensaje de un objeto a otro objeto.
- Mensaje de un objeto a si mismo.

## Elementos

- **Objeto/Actor:**



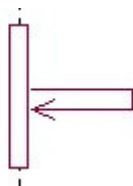
El rectángulo representa una instancia de un Objeto en particular, y la línea punteada representa las llamadas a métodos del objeto.

- **Mensaje a Otro Objeto:**



Se representa por una flecha entre un objeto y otro, representa la llamada de un método (operación) de un objeto en particular.

- **Mensaje al Mismo Objeto:**



No solo llamadas a métodos de objetos externos pueden realizarse, también es posible visualizar llamadas a métodos desde el mismo objeto en estudio.

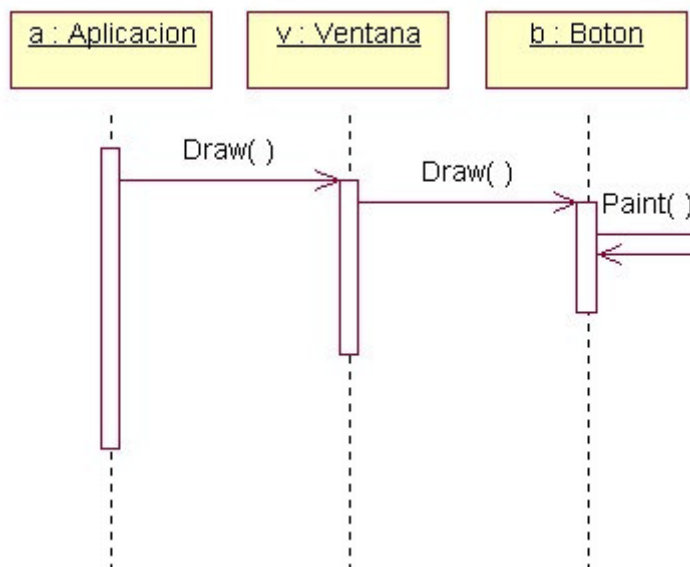
### Ejemplo

En el presente ejemplo, tenemos el diagrama de interacción proveniente del siguiente modelo estático:



Aquí se representa una aplicación que posee una Ventana gráfica, y ésta a su vez posee internamente un botón.

Entonces el diagrama de interacción para dicho modelo es:



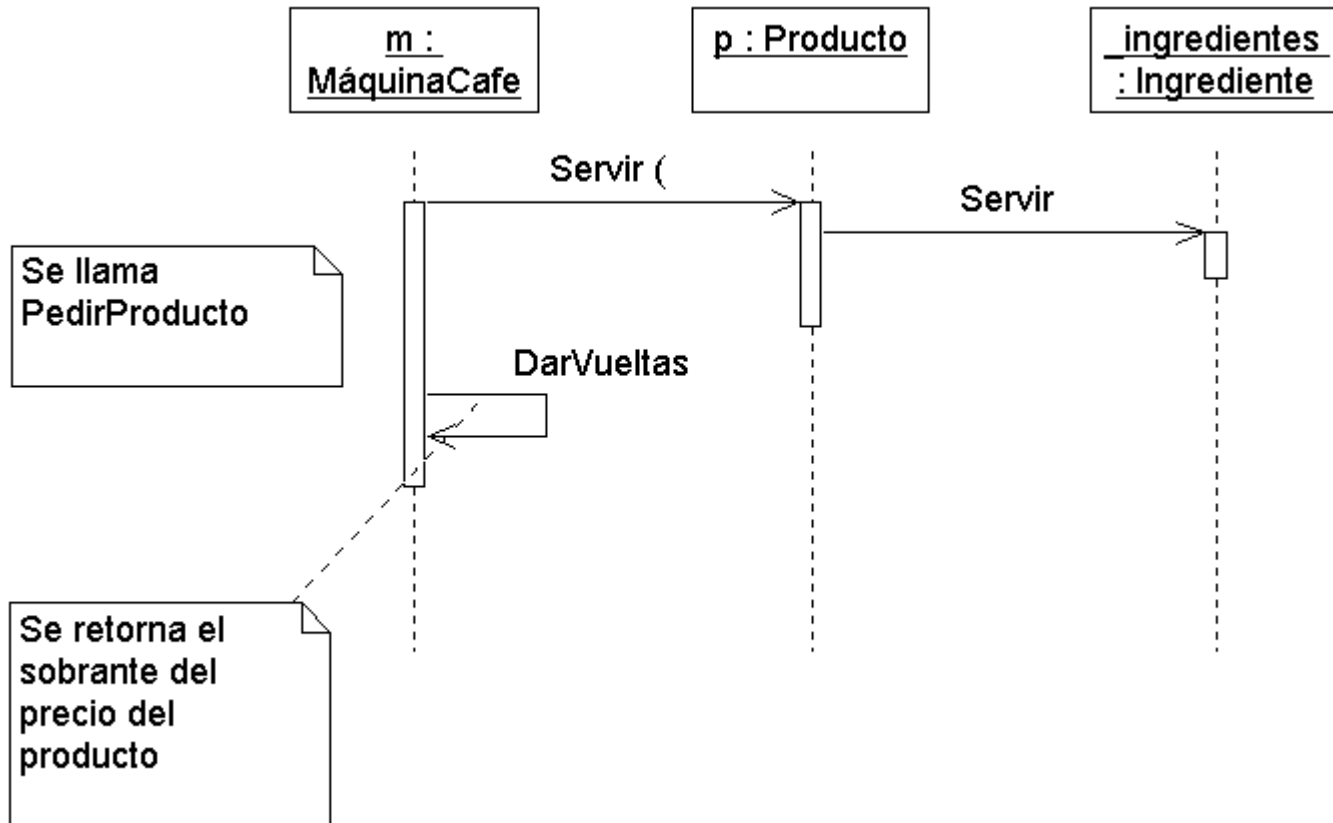
En donde se hacen notar las sucesivas llamadas a Draw() (entre objetos) y la llamada a Paint() por el objeto Botón.

## Conceptos básicos en un Diagrama de Secuencia

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo. Esta descripción es importante porque puede dar detalle a los casos de uso, aclarándolos al nivel de mensajes de los objetos existentes, como

también muestra el uso de los mensajes de las clases diseñadas en el contexto de una operación.

A continuación se muestra un ejemplo de diagrama de secuencia, que da detalle al caso de uso PedirProducto del ejemplo de la cafetera.



### Línea de vida de un objeto

Un objeto se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos (véase activación). El rectángulo de encabezado contiene el nombre del objeto y el de su clase, en un formato *nombreObjeto: nombreClase*. Por ejemplo, el objeto m, instancia de la clase MáquinaCafe envía dos mensajes seguidos para dar respuesta a la operación PedirProducto: Servir al objeto p de la clase Producto y DarVueltas a sí mismo.

### Activación

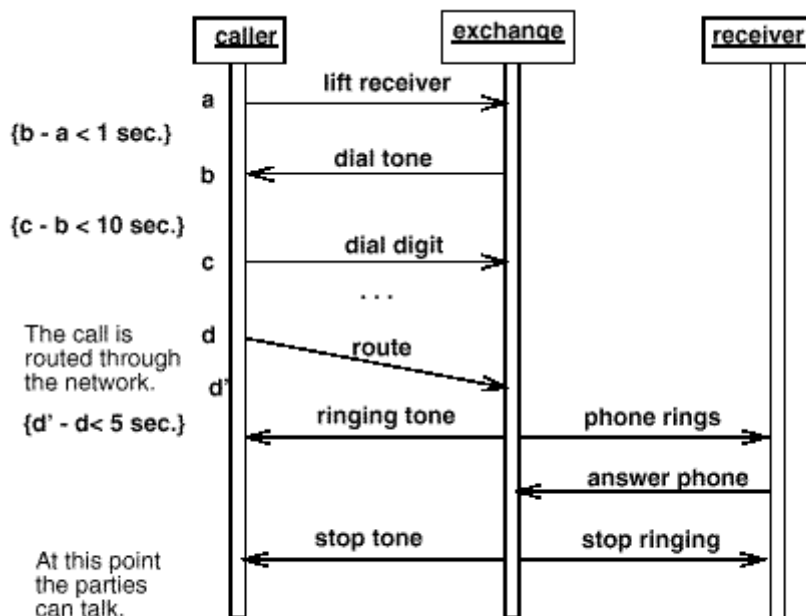
Muestra el periodo de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto. En el ejemplo anterior el objeto **ingredientes** se encuentra activado mientras ejecuta el método correspondiente al mensaje Servir; el objeto **p** se encuentra activo mientras se ejecuta su método Servir (que ejecuta `_ingredientes.Servir`) y el objeto m se encuentra activo mientras se ejecuta `p.Servir` y `DarVueltas`.

## Mensaje

El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta. En el ejemplo anterior el objeto *m* envía el mensaje *Servir* al objeto *p* y un poco más adelante en el tiempo el objeto *m* se envía a sí mismo el mensaje *DarVueltas*.

## Tiempos de transición

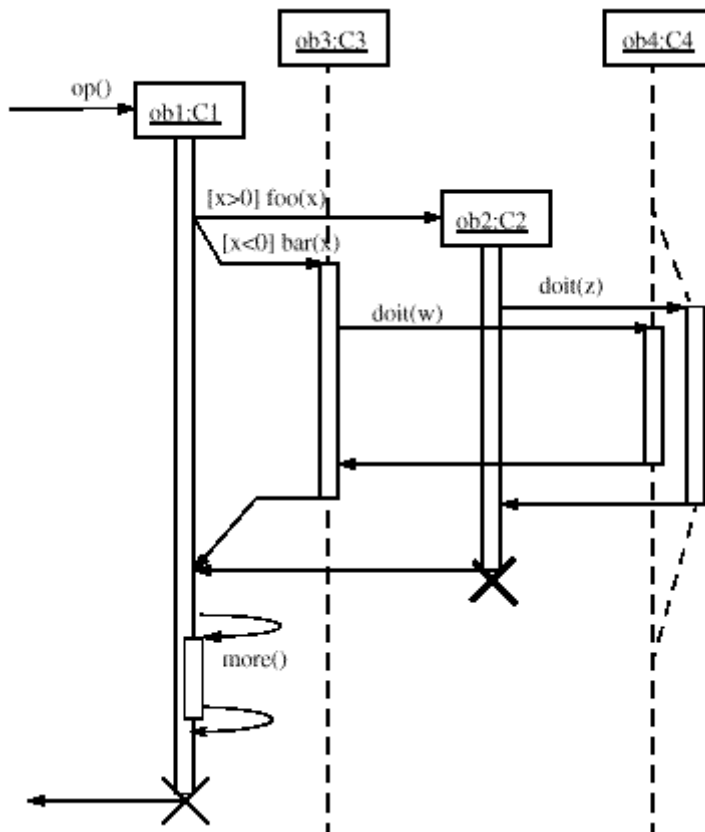
En un ambiente de objetos concurrentes o de demoras en la recepción de mensajes, es útil agregar nombres a los tiempos de salida y llegada de mensajes. Analizando la recepción de una llamada telefónica puede tenerse un diagrama como el siguiente:



En este diagrama se tienen tres objetos concurrentes, el que hace la llamada, la central telefónica y el que recibe la llamada. Se nombran los tiempos de los mensajes que envía o recibe el caller (**a** para descolgar, **b** para el tono de la llamada, **c** para la marcación, **d** para el inicio del enrutamiento de la llamada, **d'** para la finalización del enrutamiento). Estos nombres o tiempos de transición permiten describir restricciones de tiempo (por ejemplo  $b - a < 1 \text{ sec.}$ ) o demoras entre el envío y la recepción (entre **d** y **d'**).

## Condiciones caminos alternativos de ejecución. Concurrencia

En algunos casos sencillos pueden expresarse en un diagrama de secuencia alternativas de ejecución. Estas alternativas pueden representar condiciones en la ejecución o diferentes hilos de ejecución (threads).



En el diagrama anterior se muestran dos casos. **ob1** muestra una condición al enviar un mensaje a **ob3** o a **ob2**, dependiendo de si  $x > 0$  o  $x < 0$ . Estas dos líneas de ejecución se vuelven a unir más adelante, indicando el fin del condicional. Por otra parte **ob4** muestra dos posibles operaciones dependiendo de si se siguió la condición  $x > 0$  o  $x < 0$ . Ya que se presentan en el mismo instante de tiempo, se requiere dividir la línea del objeto en dos (esta misma representación se utiliza para el caso de dos hilos de ejecución).

## Destrucción de un objeto

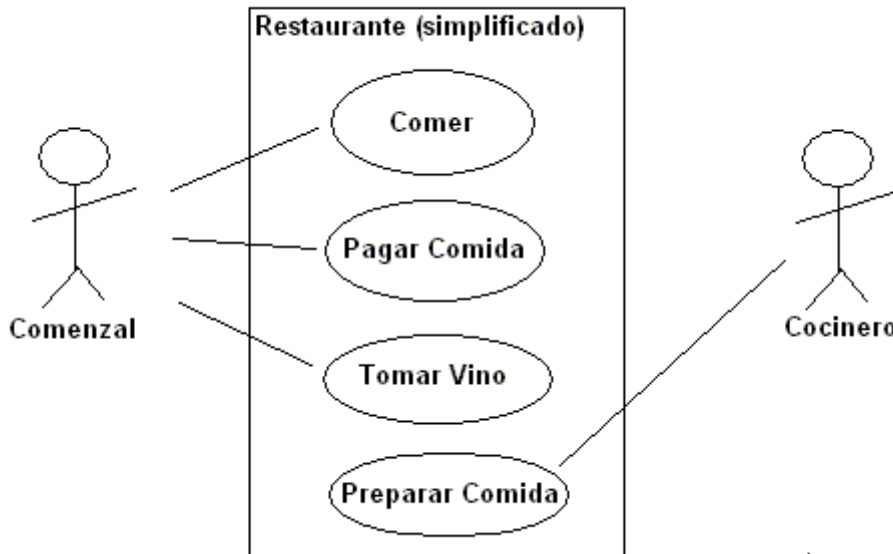
Se representa como una X al final de la línea de ejecución del objeto. Por ejemplo, en el diagrama anterior se muestra el final de **ob2** y de **ob1**.

## Métodos recursivos

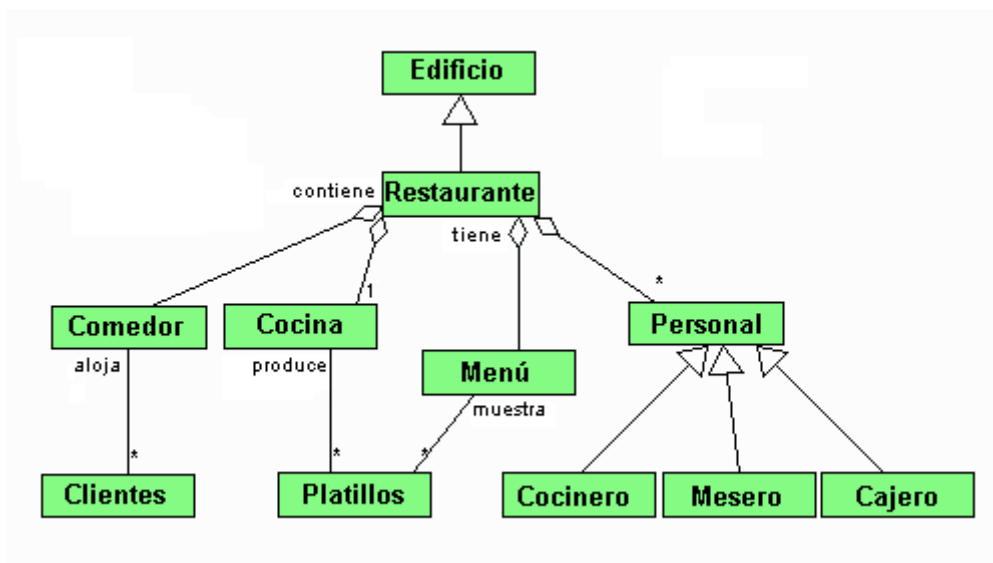
Un ejemplo de un método recursivo es el método **more** en **ob1**. Es un rectángulo un poco salido de la activación principal y con líneas de llamado de mensajes, que indican la entrada y salida de la recursión.

## Imagen:CasoDeUsoUML.png

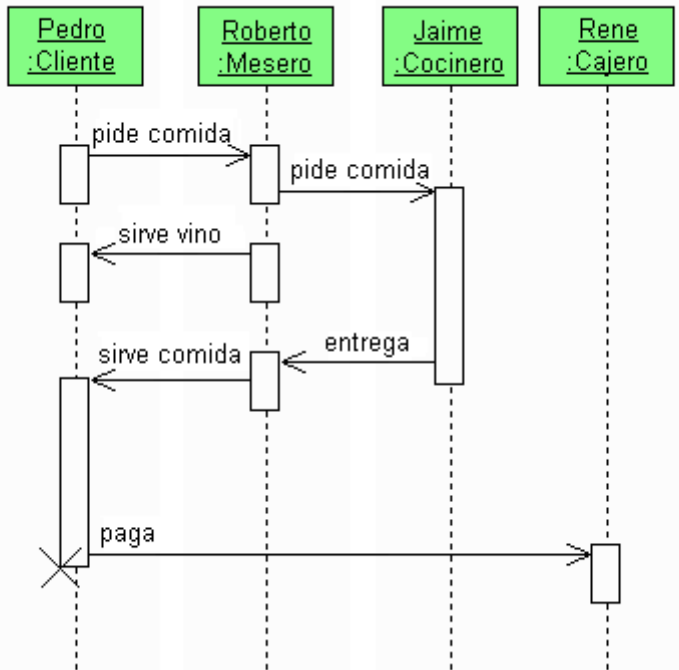




**Imagen:ClasesUML.png**



**Imagen:SecuenciaUML.png**



*Diagrama de clases.* Este diagrama describe la estructura (simplificada) de un sistema de restaurante. El sistema tiene cualquier cantidad de platillos, una cocina, comedor y cualquier número de empleados, todos estos objetos asociados a un restaurante.

*Diagrama de secuencia.* Este diagrama describe la secuencia (simplificada) de mensajes de un sistema de restaurante. El diagrama representa a un cliente pidiendo comida y pagando. *Caso de uso.* Este diagrama describe la funcionalidad (simplificada) de un sistema de restaurante, el comensal puede comer, tomar vino y pagar; solo el cocinero puede preparar la comida.