

# **Resumen UML para el PFC**

## Tabla de contenido

Introducción a UML.....	3
Diagrama de Clases.....	5
3.1.- Creación de clases.....	6
3.2.- Atributos.....	6
3.3.- Métodos.....	7
3.4.- Relaciones entre clases.....	7
3.4.1.- Cardinalidad o multiplicidad de la relación.....	8
3.4.2.- Relación de herencia (Generalización).....	9
3.4.3.- Agregación y composición.....	9
3.4.4.- Atributos de enlace.....	10
3.4.5.- Restricciones.....	10
Elaboración de diagramas de comportamiento.....	12
1.- Introducción.....	12
2.- Diagramas de casos de uso.....	12
2.1.- Elementos del diagrama de casos de uso.....	12
2.2.- Elaboración de casos de uso.....	16
2.3.- Escenarios.....	16
3.-Diagrama de interacción.....	17
3.1.- Diagramas de secuencia.....	17
Anexo I.- Licencias de recursos.....	22

# Introducción a UML

**UML** (*Unified Modeling Language o Lenguaje Unificado de Modelado*) es un conjunto de herramientas que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh, de hecho las raíces técnicas de UML son:

- OMT - Object Modeling Technique (Rumbaugh et al.)
- Método-Booch (G. Booch)
- OOSE - Object-Oriented Software Engineering (I. Jacobson)

UML permite a los desarrolladores visualizar el producto de su trabajo en esquemas o diagramas estandarizados denominados modelos que representan el sistema desde diferentes perspectivas.

## ¿Por qué es útil modelar?

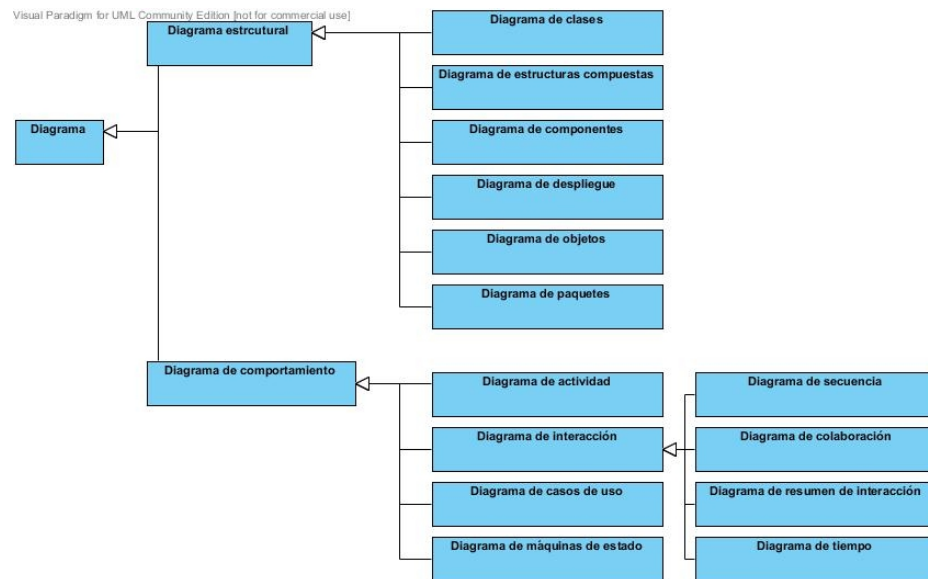
- Porque permite utilizar un lenguaje común que facilita la comunicación entre el equipo de desarrollo.
- Con UML podemos documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones,...) por lo que se dispone de documentación que trasciende al proyecto.
- Hay estructuras que trascienden lo representable en un lenguaje de programación, como las que hacen referencia a la arquitectura del sistema, utilizando estas tecnologías podemos incluso indicar qué módulos de software vamos a desarrollar y sus relaciones, o en qué nodos hardware se ejecutarán cuando trabajamos con sistemas distribuidos.
- Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose modelos precisos, no ambiguos y completos.

UML define un sistema como una **colección de modelos** que describen sus diferentes perspectivas. Los modelos se implementan en una serie de diagramas que son representaciones gráficas de una colección de elementos de modelado, a menudo dibujado como un grafo conexo de arcos (relaciones) y vértices (otros elementos del modelo).

Los diagramas **UML** se clasifican en:

- **Diagramas estructurales.** Representan la visión **estática** del sistema. Especifican clases y objetos y como se distribuyen físicamente en el sistema.
- **Diagramas de comportamiento.** Muestran la **conducta en tiempo** de ejecución del sistema, tanto desde el punto de vista del sistema completo como de las instancias u objetos que lo integran.

En la siguiente imagen aparecen todos los diagramas organizados según su categoría:



### Diagramas estructurales.

- **Diagrama de clases.** Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones.
- **Diagrama de objetos.** Muestra los elementos del modelo estático en un momento concreto, habitualmente en casos especiales de un diagrama de clases o de comunicaciones, y está formado por un conjunto de objetos y sus relaciones.
- **Diagrama de componentes.** Especifica la organización lógica de la implementación de una aplicación, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos.
- **Diagrama de despliegue.** Representa la configuración del sistema en tiempo de ejecución. Aparecen los nodos de procesamiento y sus componentes. Exhibe la ejecución de la arquitectura del sistema. Incluye nodos, ambientes operativos tanto de hardware como de software, así como las interfaces que las conectan, es decir, muestra como los componentes de un sistema se distribuyen entre los ordenadores que los ejecutan. Se utiliza cuando tenemos sistemas distribuidos.
- **Diagrama de estructuras compuestas.** Muestra la estructura interna de una clase, e incluye los puntos de interacción de esta clase con otras partes del sistema.
- **Diagrama de paquetes.** Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. Suele ser útil para la gestión de sistemas de mediano o gran tamaño.

### Diagramas de comportamiento.

- **Diagrama de casos de uso.** Representa las acciones a realizar en el sistema desde el punto de vista de los usuarios. En él se representan las acciones, los usuarios y las relaciones entre ellos. Sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.
- **Diagrama de estado de la máquina.** Describe el comportamiento de un sistema dirigido por eventos. En el que aparecen los estados que puede tener un objeto o interacción, así como las transiciones entre dichos estados. También se denomina diagrama de estado, diagrama de estados y transiciones o diagrama de cambio de estados.
- **Diagrama de actividades.** Muestra el orden en el que se van realizando tareas dentro de un sistema. En él aparecen los procesos de alto nivel de la organización. Incluye flujo de datos, o un modelo de la lógica compleja dentro del sistema.
- **Diagramas de interacción.**

- **Diagrama de secuencia.** Representa la ordenación temporal en el paso de mensajes. Modela la secuencia lógica, a través del tiempo, de los mensajes entre las instancias.
- **Diagrama de comunicación/colaboración.** Resalta la organización estructural de los objetos que se pasan mensajes. Ofrece las instancias de las clases, sus interrelaciones, y el flujo de mensajes entre ellas. Comúnmente enfoca la organización estructural de los objetos que reciben y envían mensajes.
- **Diagrama de interacción.** Muestra un conjunto de objetos y sus relaciones junto con los mensajes que se envían entre ellos. Cada nodo de actividad dentro del diagrama puede representar otro diagrama de interacción.
- **Diagrama de tiempos.** Muestra el cambio en un estado o una condición de una instancia o un rol a través del tiempo. Se usa normalmente para exhibir el cambio en el estado de un objeto en el tiempo, en respuesta a eventos externos.

La herramienta más simple que se puede utilizar para generar diagramas es lápiz y papel, hoy día, sin embargo, podemos acceder a herramientas CASE que facilitan en gran manera el desarrollo de los diagramas UML. Estas herramientas suelen contar con un entorno de ventanas tipo wysiwyg, permiten documentar los diagramas e integrarse con otros entornos de desarrollo incluyendo la generación automática de código y procedimientos de ingeniería inversa.

Podemos encontrar, entre otras, las siguientes herramientas:

- **Rational Systems Developer de IBM:** Herramienta propietaria que permite el desarrollo de proyectos software basados en la metodología UML. Desarrollada en origen por los creadores de UML ha sido recientemente absorbida por IBM. Ofrece versiones de prueba, y software libre para el desarrollo de diagramas UML.
- **Visual Paradigm for UML (VP-UML):** Incluye una versión para uso no comercial que se distribuye libremente sin más que registrarse para obtener un archivo de licencia (bajo licencia LGPL).
  - Incluye diferentes módulos para realizar desarrollo UML, diseñar bases de datos, realizar actividades de ingeniería inversa y diseñar con Agile.
  - Compatible con UML 2.0.
  - Admite la generación de informes en formatos PDF, HTML y otros.
  - Es compatible con los IDE de Eclipse, Visual Studio .net, IntelliJIDEA y NetBeans.
  - Multiplataforma.
- **ArgoUML:** se distribuye bajo licencia Eclipse. Soporta los diagramas de UML 1.4, y genera código para java y C++. Para poder ejecutarlo se necesita la plataforma java. Admite ingeniería directa e inversa.
- **UMLet:** herramienta UML de código abierto y libre distribución. Dispone de un interfaz de usuario sencillo de utilizar

## Diagrama de Clases

El diagrama de clases puede considerarse el más importante de todos los existentes en UML, encuadrado dentro de los diagramas estructurales, representa los elementos estáticos del sistema, sus atributos y comportamientos, y como se relacionan entre ellos. Contiene las clases del dominio del problema, y a partir de éste se obtendrán las clases que formarán después el programa informático que dará solución al problema.

En un **diagrama de clases** podemos encontrar los siguientes elementos:

- **Clases:** agrupan conjuntos de objetos con características comunes, que llamaremos atributos, y su comportamiento, que serán métodos. Los atributos y métodos tendrán una visibilidad

que determinará quién puede acceder al atributo o método. Por ejemplo, una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará.

- **Relaciones:** en el diagrama se representan relaciones reales entre los elementos del sistema a los que hacen referencia las clases. Pueden ser de asociación, agregación, composición y generalización. Por ejemplo, si tenemos las clases persona y coche, se puede establecer la relación conduce entre ambas. O una clase alumno puede tener una relación de generalización respecto a la clase persona.
- **Notas:** se representan como un cuadro donde podemos escribir comentarios que ayuden al entendimiento del diagrama.
- **Elementos de agrupación:** Se utilizan cuando hay que modelar un sistema grande, entonces las clases y sus relaciones se agrupan en paquetes, que a su vez se relacionan entre sí.

### 3.1.- Creación de clases.

Una clase se representa en el diagrama como un rectángulo dividido en tres filas: arriba aparece el nombre de la clase, a continuación los atributos con su visibilidad y después los métodos con su visibilidad.

Nombre Clase
-lista de atributos
+lista de métodos()

### 3.2.- Atributos.

Forman la parte estática de la clase. Son un conjunto de variables para las que es preciso definir:

- Su **nombre**.
- Su **tipo**, puede ser un tipo simple, que coincidirá con el tipo de dato que se seleccione en el lenguaje de programación final a usar, o compuesto, pudiendo incluir otra clase.

Además se pueden indicar otros datos como un **valor inicial** o su **visibilidad**. La visibilidad de un atributo se puede definir como:

- **Público (+):** Se pueden acceder desde cualquier clase y cualquier parte del programa.
- **Privado(-):** Sólo se pueden acceder desde operaciones de la clase.
- **Protegido(#):** Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
- **Paquete(~):** se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación tiene esta característica como es el caso de Java.

### 3.3.- Métodos.

Representan la funcionalidad de la clase, es decir, **qué puede hacer**. Para definir un método hay que indicar como mínimo su **nombre**, **parámetros**, el **tipo que devuelve** y su **visibilidad** (similar a la de los atributos). También se debe incluir una descripción del método que aparecerá en la documentación que se genere del proyecto.

Existen dos métodos particulares:

- El **constructor** de la clase, que tiene como característica que no devuelve ningún valor. El constructor tiene el mismo nombre de la clase y se usa para ejecutar las acciones necesarias cuando se instancia un nuevo objeto.
- El **destructor** de la clase. Cuando no se vaya a utilizar más el objeto, se podrá utilizar un método destructor que libere los recursos del sistema que tenía asignados. La destrucción de los objetos es tratada de formas diferentes en función del lenguaje de programación que se esté utilizando.

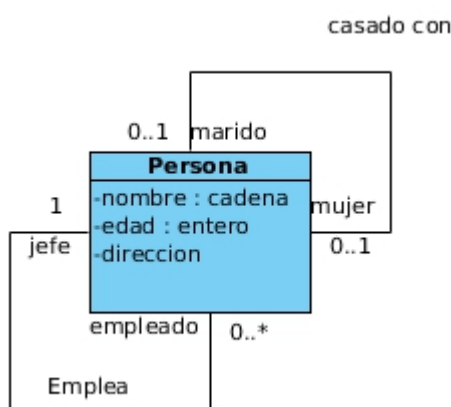
### 3.4.- Relaciones entre clases.

Una relación es una conexión entre dos clases que incluimos en el diagrama.

Se representan como una línea continua. Los mensajes "navegan" por las relaciones entre clases, es decir, los mensajes se envían entre objetos de clases relacionadas, normalmente en ambas direcciones, aunque a veces la definición del problema hace necesario que se navegue en una sola dirección, entonces la línea finaliza en punta de flecha.

Las relaciones se caracterizan por su cardinalidad, que representa cuantos objetos de una clase se pueden involucrar en la relación.

Es posible establecer relaciones unarias de una clase consigo misma. En el ejemplo se ha rellenado en la especificación de la relación los roles y la multiplicidad.



Otros tipos de relaciones que se verán más adelante son:

- De herencia.
- De composición.
- De agregación.

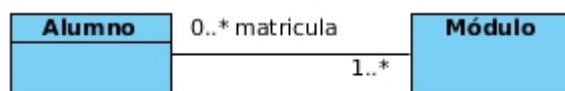
### 3.4.1.- Cardinalidad o multiplicidad de la relación

Un concepto muy importante es la **cardinalidad de una relación**, representa cuantos objetos de una clase se van a relacionar con objetos de otra clase. En una relación hay dos cardinalidades, una para cada extremo de la relación y pueden tener los siguientes valores:

#### Significado de las cardinalidades.

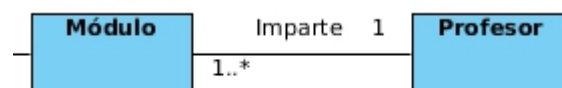
Cardinalidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

Por ejemplo, si tengo la siguiente relación:



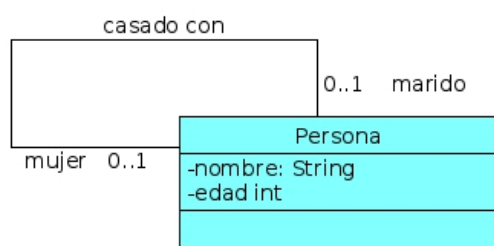
quiere decir que la clase **Alumno** se relaciona con la clase **Módulo** debido a que los alumnos se matriculan en diferentes módulos y en un módulo puede estar matriculado alumnos. La cardinalidad indicada quiere decir que todo alumno está matriculado en al menos un módulo y puede estar matriculado en varios y que en un módulo puede haber varios alumnos matriculados y puede ser que en un módulo no haya nadie matriculado.

O esta otra:



quiere decir que la clase **Profesor** relaciona con la clase **Módulo** debido a que los profesores imparten diferentes módulos y un módulo es impartido por un profesor. La cardinalidad indicada quiere decir que todo profesor imparte al menos un módulo pudiendo impartir varios y, todo módulo es impartido por un profesor y sólo uno.

Y este ejemplo:





Indica que una persona se relaciona con otra persona por la relación "casado con". La cardinalidad indica que una mujer puede estar soltera o casada con una persona y los maridos igual.

### 3.4.2.- Relación de herencia (Generalización)

La herencia es una propiedad que permite a los objetos ser contruidos a partir de otros objetos, es decir, la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados. También recibe el nombre de generalización.

El objetivo principal de la herencia es la reutilización, poder utilizar código desarrollado con anterioridad. La herencia supone una clase base y una jerarquía de clases que contiene las clases derivadas. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes, es por esto que los atributos, métodos y relaciones de una clase se muestran en el nivel más alto de la jerarquía en el que son aplicables.

#### Tipos:

1. **Herencia simple:** Una clase puede tener sólo un ascendente. Es decir una subclase puede heredar datos y métodos de una única clase base.
2. **Herencia múltiple:** Una clase puede tener más de un ascendente inmediato, adquirir datos y métodos de más de una clase.

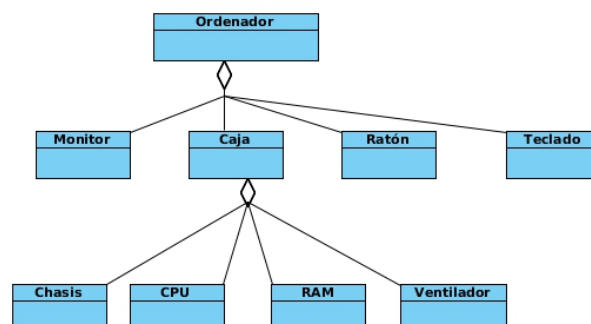
#### Representación:

En el diagrama de clases se representa como una asociación en la que el extremo de la clase base tiene un triángulo.

### 3.4.3.- Agregación y composición.

Muchas veces una determinada entidad existe como un conjunto de otras entidades. En este tipo de relaciones un objeto componente se integra en un objeto compuesto. La orientación a objetos recoge este tipo de relaciones como dos conceptos: la agregación y la composición.

La agregación es una asociación binaria que representa una relación todo-parte (pertenecer a, tener un, es parte de). Los elementos parte pueden existir sin el elemento contenedor y no son propiedad suya. Por ejemplo, un centro comercial tiene clientes o un equipo tiene unos miembros. El tiempo de vida de los objetos no tiene porqué coincidir.



En el siguiente caso, tenemos un ordenador que se compone de piezas sueltas que pueden ser sustituidas y que tienen entidad por si mismas, por lo que se representa mediante relaciones de agregación. Utilizamos la agregación porque es posible que una caja, ratón o teclado o una memoria RAM existan con independencia de que pertenezcan a un ordenador o no.

La composición es una agregación fuerte en la que una instancia ‘parte’ está relacionada, como máximo, con una instancia ‘todo’ en un momento dado, de forma que cuando un objeto ‘todo’ es eliminado, también son eliminados sus objetos ‘parte’. Por ejemplo: un rectángulo tiene cuatro vértices, un centro comercial está organizado mediante un conjunto de secciones de venta...



Para modelar la estructura de un ciclo formativo vamos a usar las clases Módulo, Competencia y Ciclo que representan lo que se puede estudiar en Formación Profesional y su estructura lógica. Un ciclo formativo se compone de una serie de competencias que se le acreditan cuando supera uno o varios módulos formativos.

Dado que si eliminamos el ciclo, las competencias no tienen sentido, y lo mismo ocurre con los módulos hemos usado relaciones de composición. Si los módulos o competencias pudieran seguir existiendo sin su contenedor habríamos utilizado relaciones de agregación.

Estas relaciones se representan con un rombo en el extremo de la entidad contenedora. En el caso de la agregación es de color blanco y para la composición negro. Como en toda relación hay que indicar la cardinalidad.

### 3.4.4.- Atributos de enlace.

Es posible que tengamos alguna relación en la que sea necesario añadir algún tipo de información que la complete de alguna manera. Cuando esto ocurre podemos añadir atributos a la relación.

### 3.4.5.- Restricciones

En ocasiones la relación entre dos clases está condicionada al cumplimiento de algún requisito, o un parámetro de una clase tiene un valor constante ...

Cuando se precisa reflejar una condición que aparece en el enunciado y no disponemos de una notación particular para que quede reflejada en el diagrama de clases, es posible mostrarla mediante una **restricción**.

Las restricciones se incluyen mediante una descripción textual encerrada entre llaves.

**Ejemplo-1.** Supongamos un ejercicio donde el socio de un club de fútbol desea acceder al palco del estadio. Si esta posibilidad sólo está disponible para antiguos

jugadores del equipo, junto a la línea que relaciona las clases socio y palco, se puede incluir la restricción {sólo para ex-jugadores}.

**Ejemplo-2.** Supongamos que la clase producto de un establecimiento tiene un IVA fijo del 21%, sea cual sea el tipo de producto que pone a la venta. Dado que este valor es fijo, podría considerarse una constante. En el diagrama de clases podría indicarse con una restricción del tipo {IVA constante 21%}.

# Elaboración de diagramas de comportamiento.

## ***1.- Introducción.***

En el tema anterior vimos como crear un diagrama de clases para un problema determinado, esto nos ayuda a ver el problema con otra perspectiva y descubrir información nueva, sin embargo no tiene en cuenta elementos como la creación y destrucción de objetos, el paso de mensajes entre ellos y el orden en que deben hacerse, qué funcionalidad espera un usuario poder realizar, o como influyen elementos externos en nuestro sistema. Un diagrama de clases nos da información estática pero no dice nada acerca del comportamiento dinámico de los objetos que lo forman, para incluir éste tipo de información utilizamos los diagramas de comportamiento que incluyen:

- Diagramas de casos de uso.
- Diagramas de actividad.
- Diagramas de estados.
- Diagramas de interacción.
  - Diagramas de secuencia.
  - Diagramas de comunicación/colaboración.
  - Diagramas de interacción.
  - Diagramas de tiempo.

## ***2.- Diagramas de casos de uso.***

Al construir software es esencial saber cuáles son los requerimientos del sistema que se desea crear, y se precisa alguna herramienta que ayude a especificarlos de una manera clara, sistemática y que los clientes puedan entender fácilmente.

Pero, ¿no bastaría con hacer una lista de requerimientos y describirlos exhaustivamente?. No, una descripción textual puede inducir a errores de interpretación y suele dejar cabos sueltos. La solución puede ser los diagramas de casos de uso.

Los diagramas de casos de uso son un elemento fundamental en la etapa de análisis de un sistema desde la perspectiva de la orientación a objetos porque resuelven uno de los principales problemas en los que se ve envuelto el proceso de producción de software: la falta de comunicación entre el equipo de desarrollo y el equipo que necesita de una solución software. Un diagrama de casos de uso nos ayuda a determinar QUÉ puede hacer cada tipo diferente de usuario con el sistema, en una forma que los no versados en el mundo de la informática o, más concretamente el desarrollo de software, pueda entender.

Los diagramas de casos de uso documentan el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto los casos de uso determinan los requisitos funcionales del sistema, es decir, representan las funciones que un sistema puede ejecutar.

Un diagrama de casos de uso es una visualización gráfica de los requisitos funcionales del sistema, que está formado por casos de uso (se representan como elipses) y los actores que interactúan con ellos (se representan como monigotes). Su principal función es dirigir el proceso de creación del software, definiendo qué se espera de él, y su ventaja principal es la facilidad para interpretarlos, lo que hace que sean especialmente útiles en la comunicación con el cliente.

### ***2.1.- Elementos del diagrama de casos de uso***

Los elementos de un diagrama de casos de uso son los siguientes:

- Actores.
- Casos de uso.
- Relaciones

### 2.1.1.- Actores.

Los actores representan un tipo de usuario del sistema. Se entiende como usuario cualquier cosa externa que interactúa con el sistema. No tiene por qué ser un ser humano, puede ser otro sistema informático o unidades organizativas o empresas.

Los actores representan los tipos de usuario que interactúan con el sistema (un ser humano, un PC, una empresa ...) . Es importante entender la diferencia entre actores y los usuarios, por ejemplo, un usuario puede interpretar diferentes roles según la operación que esté ejecutando, cada uno de estos roles representará un actor diferente. Por otro lado, cada actor puede ser interpretado por diferentes usuarios.

Por ejemplo, el dueño de una panadería podrá aparecer en un diagrama de casos de uso con los roles de administrador y de cocinero, a su vez, puede tener otro usuario contratado, de forma que el actor cocinero podrá ser "interpretado" tanto por el dueño como por el empleado.

**Tipos de actores:**

- **Primarios:** interactúan con el sistema para explotar su funcionalidad. Trabajan directa y frecuentemente con el software.
- **Secundarios:** soporte del sistema para que los primarios puedan trabajar. Son precisos para alcanzar algún objetivo.
- **Iniciadores:** es posible que haya casos de uso que no sean iniciados por ningún usuario, en ese caso se podrá considerar un actor "tiempo" o "sistema" que asuma el arranque del caso.

Los actores se representan mediante la siguiente figura:



Es posible que haya casos de uso que no sean iniciados por ningún usuario, o algún otro elemento software, en ese caso se puede crear un actor "Tiempo" o "Sistema".

### 2.1.2.- Casos de uso.

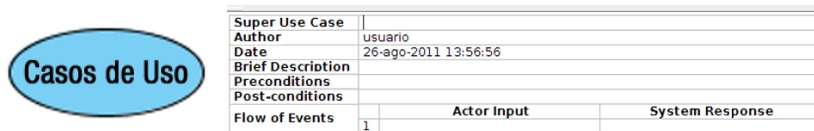
Se utilizan casos de uso para especificar tareas que deben poder llevarse a cabo con el apoyo del sistema que se está desarrollando.

Un caso de uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un actor concreto.

El conjunto de casos de uso forma el "comportamiento requerido" de un sistema. El objetivo principal de elaborar un diagrama de casos de uso no es crear el diagrama en sí, sino la descripción que de cada caso se debe realizar, ya que esto es lo que ayuda al equipo de desarrollo a crear el sistema a posteriori. Junto al diagrama, por cada caso de uso se crea una tabla con una descripción textual, en la que se deben incluir, al menos, los siguientes datos (a los que se denomina contrato).

- **Nombre:** nombre del caso de uso.
- **Actores:** aquellos que interactúan con el sistema a través del caso de uso.
- **Propósito:** breve descripción de lo que se espera que haga.
- **Precondiciones:** aquellas que deben cumplirse para que pueda llevarse a cabo el caso de uso.
- **Flujo normal:** flujo normal de eventos que deben cumplirse para ejecutar el caso de uso exitosamente, desde el punto de vista del actor que participa y del sistema.

- **Flujo alternativo:** flujo de eventos que se llevan a cabo cuando se producen casos inesperados o poco frecuentes. No se deben incluir aquí errores como escribir un tipo de dato incorrecto o la omisión de un parámetro necesario.
- **Postcondiciones:** las que se cumplen una vez que se ha realizado el caso de uso.

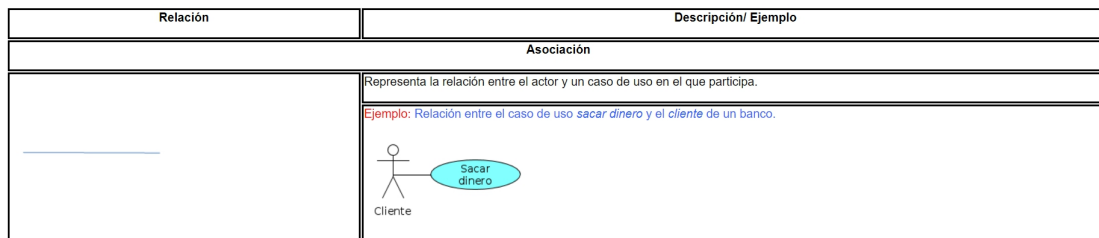




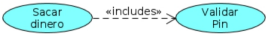




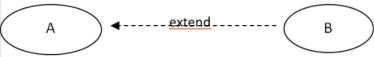


La representación gráfica de un caso de uso se realiza mediante un óvalo o elipse, y su descripción se suele hacer rellenando una o más tablas como la de la imagen (obtenida de la herramienta Visual Paradigm).

### 2.1.3.- Relaciones.

Los diagramas de casos de uso son grafos no conexos en los que los nodos son **actores** y **casos de uso**, y las aristas son las **relaciones** que existen entre ellos. Las relaciones representan qué actores realizan las tareas descritas en los casos de uso, en concreto qué actores inician un caso de uso. Pero además existen otros tipos de relaciones que se utilizan para especificar relaciones más complejas, como uso o herencia entre casos de uso o actores.

Existen diferentes tipos de relaciones entre elementos:



Inclusión (include - use)	
	Se trata de una relación entre casos de uso. La ejecución de un caso de uso implica <b>necesariamente</b> la ejecución del segundo.
<p>include</p>	
	<p>Esta relación es muy útil cuando se desea especificar algún comportamiento común en dos o más casos de uso, aunque es frecuente cometer el error de utilizar esta técnica para hacer subdivisión de funciones, por lo que se debe tener mucho cuidado cuando se utilice.</p> <p><b>Ejemplo 1:</b> Al ejecutar el caso de uso <i>sacar dinero</i>, <b>obligatoriamente</b> se ejecuta el caso de uso <i>validar pin</i> de la tarjeta de crédito.</p>  <p><b>Ejemplo 2:</b> Por ejemplo, a la hora de hacer un pedido se debe buscar la información de los artículos para obtener el precio, es un proceso que <b>necesariamente</b> forma parte del caso de uso, sin embargo también forma parte de otros, como son el que visualiza el catálogo de productos y la búsqueda de un artículo concreto, y dado que tiene entidad por sí solo se separa del resto de casos de uso y se incluye en los otros tres.</p>
<p>En el siguiente gráfico se representa que A usa B, es decir, que <b>A siempre ejecuta B</b>.</p> 	
Extensión (extend)	
	Se trata de una relación entre casos de uso. La ejecución de un caso de uso <b>puede</b> provocar la ejecución del segundo
<p>extend</p>	
	<p>Se utiliza una relación entre dos casos de uso de tipo "<b>extends</b>" cuando se desea especificar que el comportamiento de un caso de uso es diferente dependiendo de ciertas circunstancias.</p> <p>La principal función de esta relación es simplificar el flujo de casos de uso complejos. Se utiliza cuando existe una parte del caso de uso que se ejecuta sólo en determinadas ocasiones, pero no es imprescindible para su completa ejecución. Cuando un caso de uso extendido se ejecuta, se indica en la especificación del caso de uso como un <b>punto de extensión</b>. Los puntos de extensión se pueden mostrar en el diagrama de casos de uso.</p>
	<p><b>Ejemplo 1:</b> <i>Imprimir ticket</i> es consecuencia del caso de uso <i>sacar dinero</i>, pero su ejecución es opcional a que sea requerida por el cliente.</p>  <p><b>Ejemplo 2:</b> Cuando un usuario hace un pedido si no es socio se le ofrece la posibilidad de darse de alta en el sistema en ese momento, pero puede realizar el pedido aunque no lo sea.</p> 
	
A opcionalmente ejecuta B.	
Generalización	
	Se utiliza para representar relaciones de herencia entre casos de uso o actores. No se contemplan generalizaciones combinadas entre actores y casos de uso.
	Se utiliza cuando se tiene uno o más casos de uso que son especificaciones de un caso más general.
	<p>Por ejemplo, entre actores: tanto <i>profesor</i> como <i>alumno</i> son casos particulares del actor <i>persona</i>.</p>  <p><b>Ejemplos, entre casos de uso:</b></p> <p>Un ejemplo de generalización de casos de uso sería la compra de artículos en un comercio, pudiendo considerarse la compra de alimentos o de bebidas. Ambos tipos de compras tendrán las características heredadas del caso de uso compra general, más las particulares definidas para cada caso.</p>

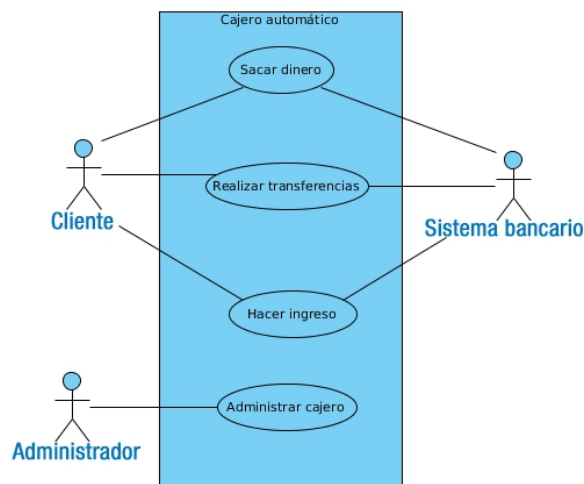
## 2.2.- Elaboración de casos de uso.

En los diagramas de casos se hace una abstracción de la realidad en la que representamos qué cosas pueden hacerse en nuestro sistema y quién las va a hacer.

Necesitamos diagramas cuya información permita al equipo de desarrollo la toma de decisiones adecuadas en la fase de análisis y diseño (cumpliendo los requisitos), así como que sean útiles en la fase de implementación en un lenguaje orientado a objetos.

Partiremos de una descripción lo más detallada posible del problema a resolver y trataremos de detectar aspectos como:

- Usuarios que interactúan con el sistema, para obtener los actores.
- Tareas que realizan estos actores para determinar los casos de uso más genéricos.
- Refinar el diagrama analizando los casos de uso más generales para detectar casos relacionados por inclusión, extensión y generalización.



## 2.3.- Escenarios.

Un caso de uso debe especificar un comportamiento deseado, pero no imponer cómo se llevará a cabo ese comportamiento, es decir, debe decir QUÉ pero no CÓMO. Esto se realiza utilizando escenarios que son casos particulares de un caso de uso.

Un escenario es una ejecución particular de un caso de uso que se describe como una secuencia de eventos. Un caso de uso es una generalización de un escenario.

Por ejemplo, para el caso de uso hacer pedido podemos establecer diferentes escenarios:



Un posible escenario podría ser:

Realizar pedido de unos zapatos y unas botas.

1. El usuario inicia el pedido.
2. Se crea el pedido en estado "en construcción".
3. Se selecciona un par de zapatos "Lucía" de piel negros, del número 39.
4. Se selecciona la cantidad 1.
5. Se recupera la información de los zapatos y se modifica la cantidad a pagar sumándole 45 €.
6. Se selecciona un par de botas "Aymara" de ante marrón del número 40.
7. Se selecciona la cantidad 1.
8. Se recupera la información de las botas y se modifica la cantidad a pagar sumándole 135 €.
9. El usuario acepta el pedido.
10. Se comprueba que el usuario es, efectivamente socio.
11. Se comprueban los datos bancarios, que son correctos.
12. Se calcula el total a pagar añadiendo los gastos de envío.
13. Se realiza el pago a través de una entidad externa.
14. Se genera un pedido para el usuario con los dos zapatos que ha comprado, con el estado "pendiente".

Los escenarios pueden y deben posteriormente documentarse mediante diagramas de secuencia.

### ***3.-Diagrama de interacción***

Una vez conocidos los diagramas de casos de uso, se hace necesario buscar la forma de representar como circula la información, los objetos que participan en los casos de uso, los mensajes que envían, y en el momento en que se producen. Disponer de esta información ayudará con posterioridad en el desarrollo de los diagramas de clases.

Los **diagramas de interacción** son vistas del sistema que muestran como grupos de objetos interactúan para un cierto comportamiento. Captan la ejecución de los casos de uso, representando a los actores que participan y los mensajes que se pasan.

Hay dos tipos de diagramas de interacción: **diagramas de secuencia** y **diagramas de colaboración**.

El diagrama de colaboración contiene la misma información que un diagrama de secuencia, pero la anotación es diferente.

#### ***3.1.- Diagramas de secuencia.***

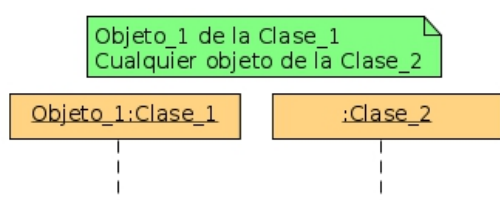
En los **diagramas de secuencia**, los objetos/actores que forman parte del escenario de un caso de uso se representan mediante rectángulos distribuidos horizontalmente en la zona superior del diagrama, a los que se asocia una línea temporal vertical (una para cada actor) de las que salen, en orden, los diferentes mensajes que se pasan entre ellos.

Con esto el equipo de desarrollo puede hacerse una idea de las diferentes operaciones que deben ocurrir al ejecutarse una determinada tarea y el orden en que deben realizarse.

### 3.1.1.- Representación de objetos, línea de vida y paso de mensajes.

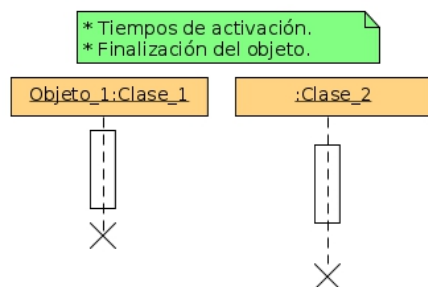
#### Representación de objetos y línea de vida.

En un **diagrama de secuencia**, los **objetos** se dibujan mediante rectángulos y se distribuyen horizontalmente en la parte superior del diagrama. Por cada objeto se identifica su nombre, seguido del símbolo de dos puntos y a continuación el nombre de la clase a la que pertenece. Si no se indica el nombre del objeto, se considera que para el propósito del diagrama es válido cualquier objeto de la clase.



De cada rectángulo sale una línea punteada que representa el paso del tiempo, se denomina **línea de vida**. La línea de vida se prolonga mientras el objeto es relevante en el diagrama, una vez deja de serlo se indica mediante una cruz "X", dejando por tanto de existir a partir de ese momento.

Cuando el objeto toma protagonismo en el intercambio de mensajes, se dice que está **activo** y se indica mediante un recuadro sobre su línea de vida.

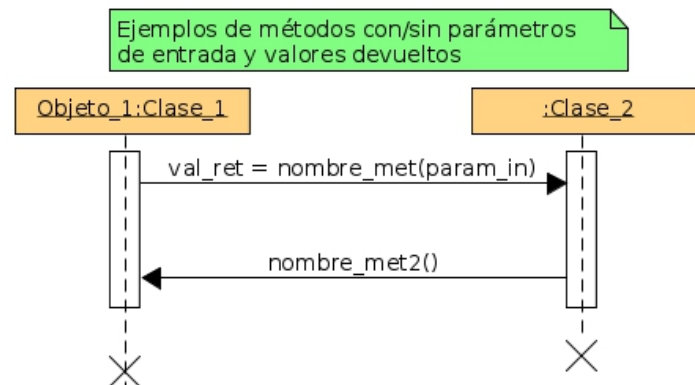


Una línea de vida puede estar encabezada por otro tipo de instancias como el sistema o un actor que aparecerán con su propio nombre. Usaremos el sistema para representar solicitudes al mismo, como por ejemplo pulsar un botón para abrir una ventana o una llamada a una subrutina.

#### Paso de mensajes (Invocación de métodos).

Los **mensajes**, que significan la invocación de métodos, se representan como flechas horizontales que van de una línea de vida a otra, indicando con la flecha la dirección del mensaje. Los mensajes se dibujan desde el objeto que envía el mensaje al que lo recibe, pudiendo ser el mismo objeto emisor y receptor de un mensaje. El orden en el tiempo viene determinado por su posición vertical, un mensaje que se dibuja debajo de otro indica que se envía después, por lo que no se hace necesario un número de

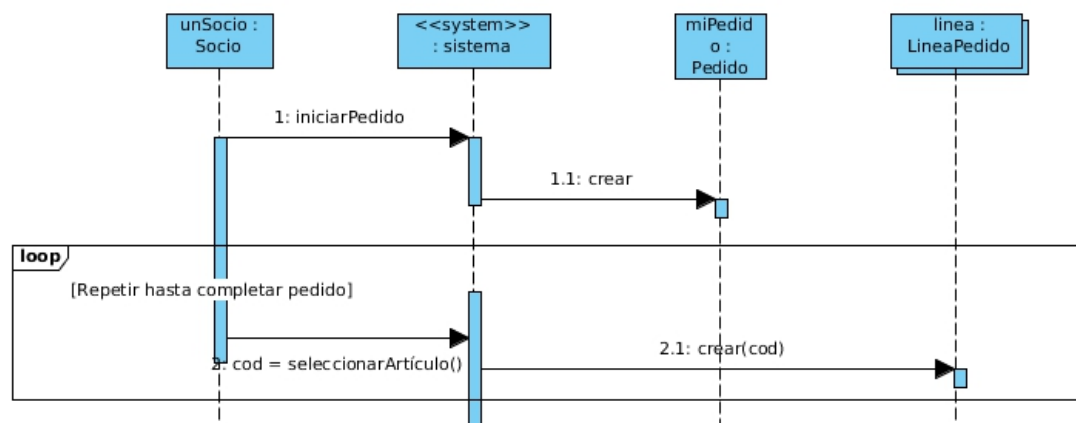
secuencia. Los **mensajes** tendrán un **nombre** y pueden incluir **argumentos de entrada**, **valores devueltos** e **información de control** (condición o iteración).



Una notación alternativa para recoger valores devueltos por los métodos es dibujar una línea de puntos finalizada en flecha, que irá desde el objeto destinatario del mensaje al que lo ha generado, acompañado del texto del valor devuelto.

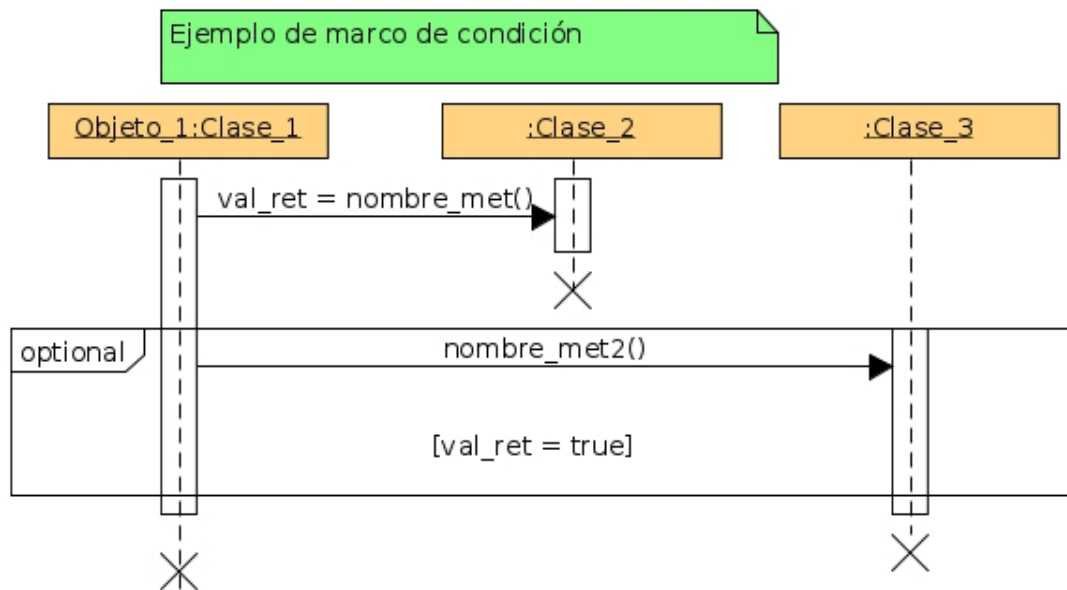
### Condicionales e iteraciones.

Además de presentar acciones sencillas que se ejecutan de manera secuencial también se pueden representar algunas situaciones más complejas como bucles usando marcos, normalmente se nombra el marco con el tipo de bucle a ejecutar y la condición de parada. También se pueden representar flujos de mensajes condicionales en función de un valor determinado.



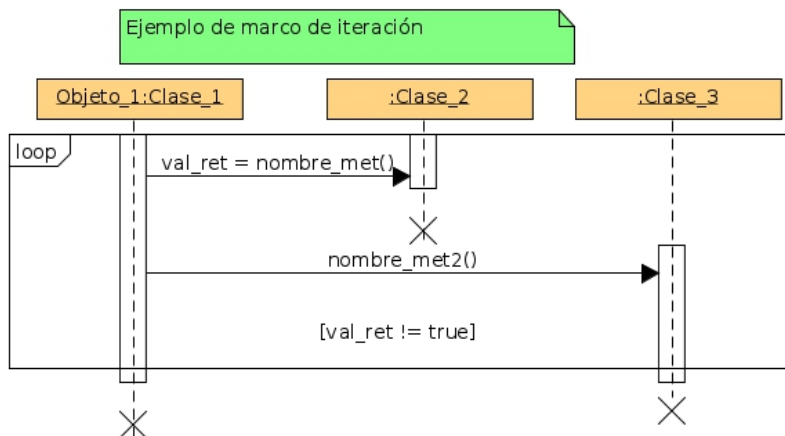
Las secuencias de control; tanto condicionales, como iterativas, se pueden representar usando **marcos**, normalmente se nombra el marco con el tipo de bucle a ejecutar y la condición de parada. También se pueden representar flujos de mensajes condicionales en función de un valor determinado.

La expresión a evaluar para la condición o iteración se representa entre corchetes.



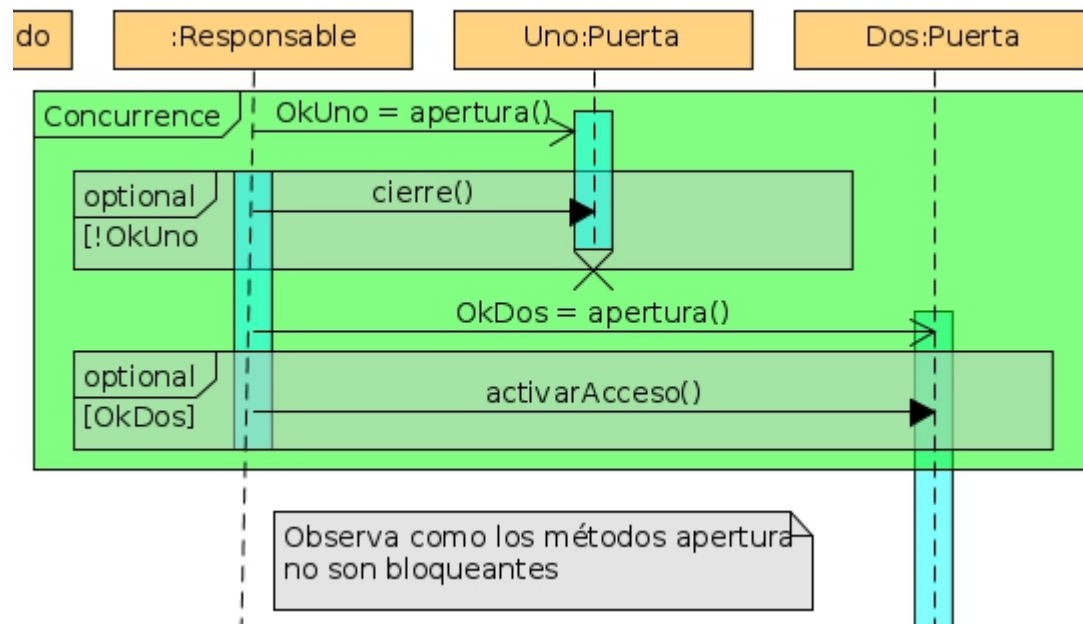
Combinando varios marcos opcionales es posible representar diferentes alternativas en la ejecución de un diagrama de secuencia.

Para el caso de una iteración, tenemos el siguiente ejemplo.



Por defecto los métodos son **bloqueantes**, se entiende que el proceso del diagrama de secuencia completa cada método antes de continuar con el siguiente, es una secuencia de métodos en el tiempo. Pero en ocasiones se producen situaciones en las que se desea mostrar varios procesos en paralelo (**conurrencia**), se puede reflejar mediante el uso de marcos con la etiqueta concurrence.

Junto a los marcos de concurrencia, se hace necesario el uso de **métodos no bloqueantes (asíncronos)**, que permitan en paralelo activar diferentes procesos. La notación utilizada para los métodos asíncronos es una línea finalizada con media cabeza de flecha o en UMLet una línea cuya punta flecha no está rellena.



Por último destacar que se puede completar el diagrama añadiendo **etiquetas** y **notas** en el margen izquierdo que aclare la operación que se está realizando.

# **Anexo I.- Licencias de recursos.**

Todo el material recopilado en este documento se recogió del banco de recursos del ministerio de educación para uso educativo no comercial disponible como recurso web en <https://www.edu.xunta.gal/>