TEMA 3 – MODELADO

1. UML 3
1.1 HISTORIA 3
1.2 CARACTERÍSTICAS 4
1.3 DIAGRAMAS UML 5
1.3.1 DIAGRAMAS ESTRUCTURALES 6
1.3.2 DIAGRAMAS DE COMPORTAMIENTO 8
2. INTRODUCCIÓN Y PARTICIPANTES 11
2.1 INTRODUCCIÓN 11
2.2 PARTICIPANTES 11
2.2.1 AUTORES: 11
2.2.2 CLIENTES: 11
2.2.3 ORGANIZACIONES INVOLUCRADAS: 11
3. OBJETIVOS DEL SISTEMA 12
3.1 EJEMPLOS 13
3.1.1 GESTIÓN DE UN VIDEOCLUB: 13
3.2 EJERCICIOS 14
4. DIAGRAMA DE CASOS DE USO 18
4.1 SUJETO O SISTEMA 18
4.2 CASOS DE USO 19
4.2.1 DESCRIPCIÓN 20
4.3 ACTORES 23
4.3.1 DESCRIPCIÓN 24
4.4 RELACIONES 24
4.4.1 RELACIONES ENTRE ACTORES 25
4.4.2 RELACIONES ENTRE ACTORES Y CASOS DE USO 25
4.4.3 RELACIONES ENTRE CASOS DE USO 26
• GENERALIZACIÓN 26
• INCLUSIÓN 27
• EXTENSIÓN 27
5. SOFTWARE DE CREACIÓN DE DIAGRAMAS 31
6. DIAGRAMA DE PAQUETES 32
6.1 PAQUETES 32
6.2 ELEMENTOS EMPAQUETABLES 33
6.3 RELACIONES 33
6.3.1 GENERALIZACIÓN 33
6.3.2 DEPENDENCIA 34
6.3.3 FUSIÓN 36
7. REQUISITOS DE INFORMACIÓN 39
8. REQUISITOS NO FUNCIONALES 41

9. MATRIZ DE RASTREABILIDAD 44

10. DIAGRAMA DE CLASES 45

10.1 CLASES 45

10.1.1 ELEMENTOS DE LA CLASE 46

10.1.2 REPRESENTACIÓN 47

10.1.3 INTERFAZ, CONTROL Y ENTIDAD: 51

10.2 VISIBLILIDAD 53

10.2.1 Tipos de visibilidad: 54

10.2.2 Representación: 54

10.2.3 Características a tener en cuenta: 54

10.3 TIPO DE DATO 54

10.4 OBJETOS. 55

10.4.1 INSTANCIACIÓN 56

10.4.2 CARACTERÍSTICAS DE LOS OBJETOS: ESTADO, COMPORTAMIENTO E IDENTIDAD 56

10.5 RELACIONES O ASOCIACIONES 56

10.5.1 ASOCIACIÓN 57

- ELEMENTOS 57
- TIPOS DE ASOCIACIONES 60

10.5.2 GENERALIZACIÓN 65

10.5.3 DEPENDENCIA 66

10.6 COMENTARIOS 67

10.7 GENERACIÓN DE CÓDIGO 67

10.8 Ejercicios 68

11. DIAGRAMA DE SECUENCIA 73

11.1 LÍNEA DE VIDA 74

11.2 MENSAJES 76

11.2.1 MENSAJE SÍNCRONO 76

11.2.2 MENSAJE ASÍNCRONO 77

11.2.3 MENSAJE DE RETORNO 77

11.2.4 MENSAJE DE CREACIÓN 78

11.2.5 MENSAJE DE DESTRUCCIÓN 79

11.2.6 MENSAJE REFLEXIVO 79

11.3 OCURRENCIA DE EJECUCIÓN 81

11.4 FRAGMENTOS COMBINADOS 82

11.4.1 ALT 82

11.4.2 OPT 84

11.4.3 LOOP 84

11.4.4 PAR 86

11.4.5 CRITICAL 87

11.5 OCURRENCIA O USO DE INTERACCIÓN 87

11.6 COMENTARIOS 87

12. REFERENCIAS 89

1. UML

Un modelo es una representación abstracta de una especificación, un diseño o un sistema, desde un punto de vista particular. Tiene como objetivo expresar la esencia de algunos aspectos de lo que se está haciendo, sin especificar detalles innecesarios.

UML **OMG**

UML (Wikipedia) **UML (Wikipedia-en)** **OMG (Wikipedia_en)**

Conceptos básicos de UML (IBM)
Acoplamiento y cohexión (latecladeescape)
Ingeniería del software (CodeCompiling)
UML (CodeCompiling)
UML (UNAD)

UML (**Unified Modeling Language** o **Lenguaje Unificado de Modelado**) es un lenguaje visual de modelado para mostrar, especificar, construir y documentar partes de un sistema software desde distintos puntos de vista y que se ha adoptado como estándar a nivel internacional por una gran cantidad de organismos y empresas.

1.1 HISTORIA

- En el año1994, *James Rumbaugh* se une a la compañía *Rational* creada por *Grady Booch.* Los dos eran unos investigadores muy reconocidos en el área de la metodología del software y tenían como objetivo unificar los métodos que habían desarrollado (*Booch* había desarrollado el *Booch Method* y *Rambaugh* el *Object Modeling Technique*) para formar uno más potente y universal.

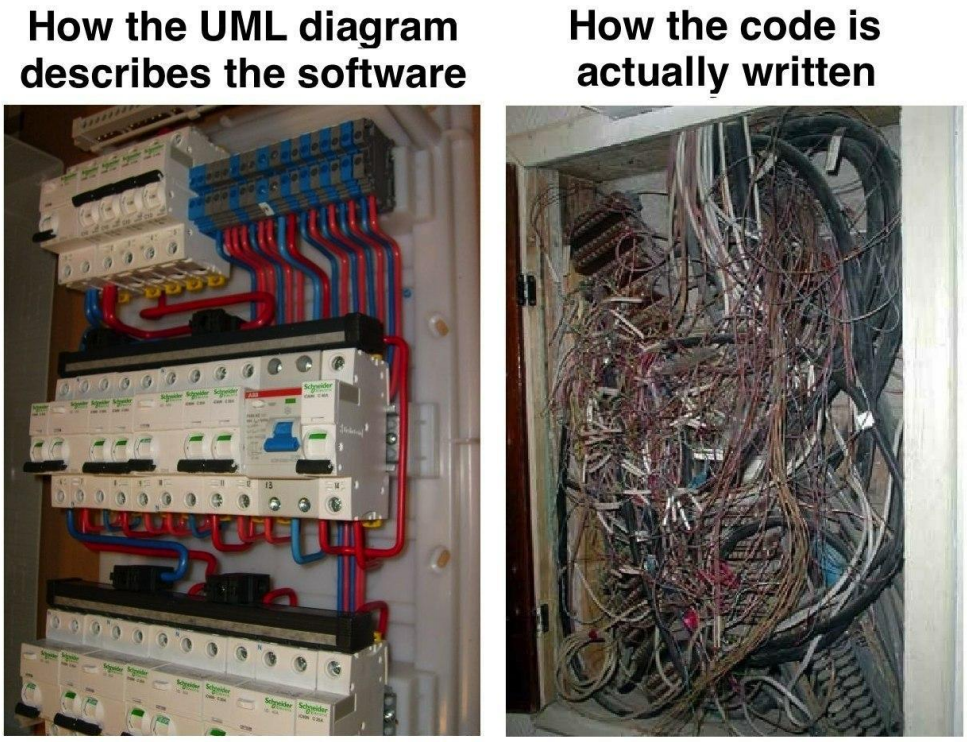
- En 1995 se incorpora a *Rational Ivar Jacobson*, que, como *Grady* y *James*, era un estudioso de la metodología del software de mucho renombre, con una metodología propia llamada *Object-Oriented Software Engeneering*. Los tres juntos publican un documento titulado *Unified Method V0.8*

- En 1997 UML 1.1 fue aprobada por la OMG (*Object Management Group*), un consorcio dedicado al cuidado y establecimiento de diversos estándares de tecnologías orientadas a objetos, convirtiéndose en la notación estándar de facto para el análisis y el diseño orientado a objetos.

- En julio de 2005 se libera UML 2.0

1.2 CARACTERÍSTICAS

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.



Un ***lenguaje de modelado*** es una manera de expresar los distintos modelos que se producen en el entorno de desarrollo.
Tiene:

a. ***Sintaxis***: En un lenguaje de modelado basado en diagramas, la sintaxis determina las reglas y principios que determinan que diagramas son legales. (formas)

b. ***Semántica*** Significado de los elementos y palabras del lenguaje. (fondo)

Tanto la sintaxis como la semántica pueden ser más o menos formales.

Además, UML puede conectarse a lenguajes de programación mediante ***ingeniería directa*** e ***inversa***.

Ingeniería directa: Generamos código a partir del modelo (de clases).
Ingeniería inversa: Generamos el modelo (de clases) a partir del código fuente.

1.3 DIAGRAMAS UML

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***, generalmente, se componen de cuatro tipos de elementos:

Estructuras: Son los nodos del grafo y definen el tipo de diagrama.

Relaciones: Son los arcos del grafo que se establecen entre los elementos estructurales.

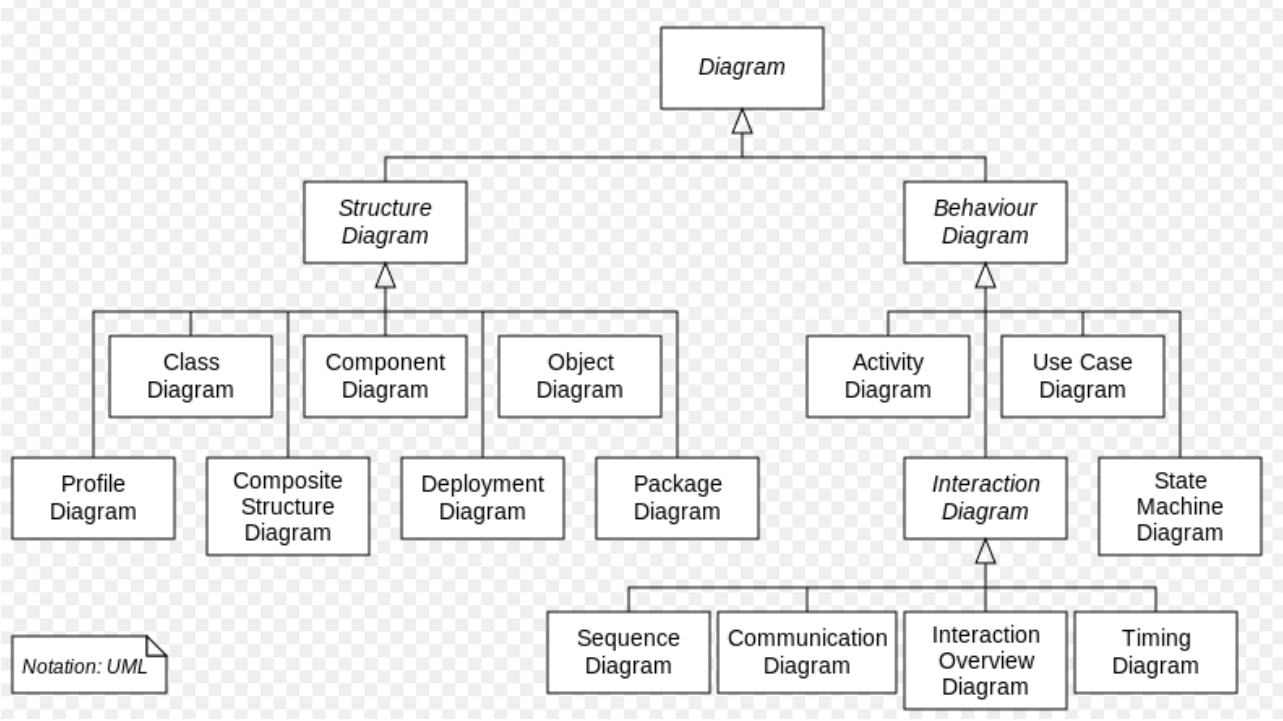
Notas: Se representan como un cuadro donde podemos escribir comentarios que nos ayuden a entender algún concepto que queramos representar.

Agrupaciones: Se utilizan cuando modelamos sistemas grandes para facilitar su desarrollo por bloques (*paquetes*).

y se clasifican en:

Diagramas estructurales: Representan la estructura ***estática*** de los elementos del sistema.
Especifican clases y objetos y como se distribuyen físicamente en el sistema.

Diagramas de comportamiento: muestran la ***conducta (comportamiento)*** en tiempo de ejecución de los elementos del sistema, tanto desde el punto de vista del sistema completo como de las instancias u objetos que lo integran. Dentro de este grupo están los diagramas de interacción.

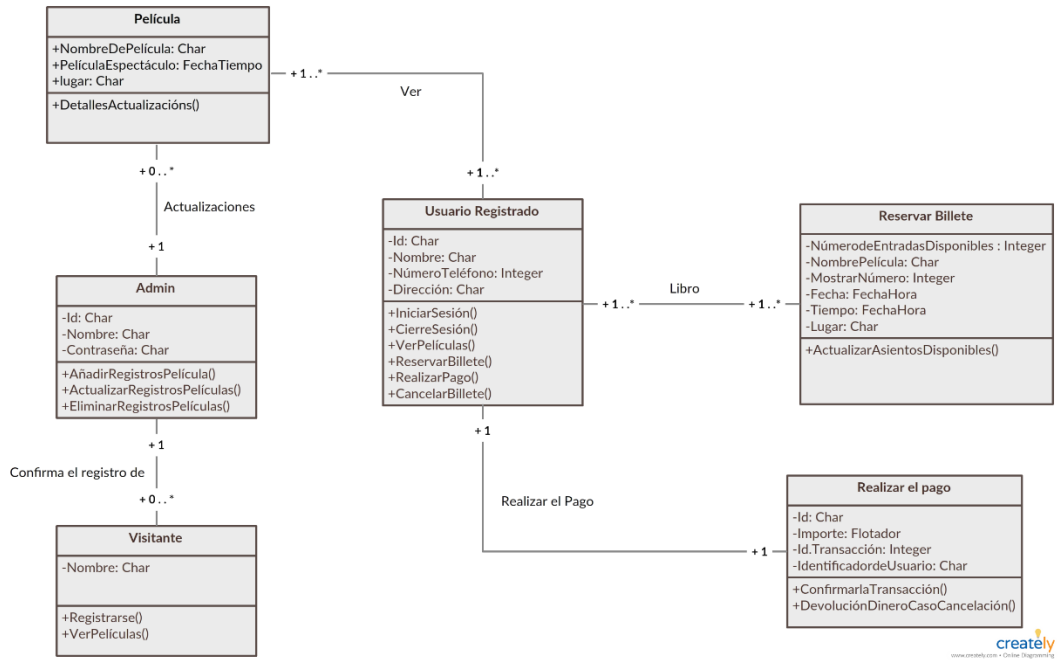


Estándar UML 2.0: En total describe catorce diagramas para modelar diferentes aspectos de un sistema, sin embargo no es necesario usarlos todos, dependerá del tipo de aplicación a generar y del sistema, es decir, se debe generar un diagrama sólo si es necesario.

Diagramas UML

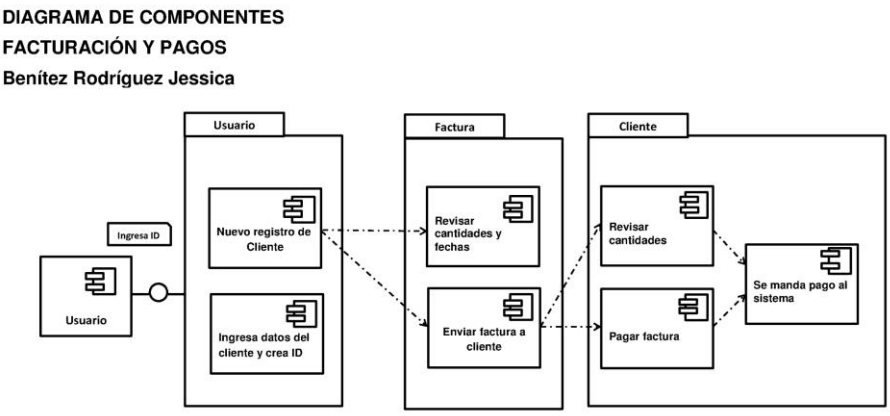
1.3.1 DIAGRAMAS ESTRUCTURALES

- **Diagramas de clases**: Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones. Tiene una prioridad ALTA.

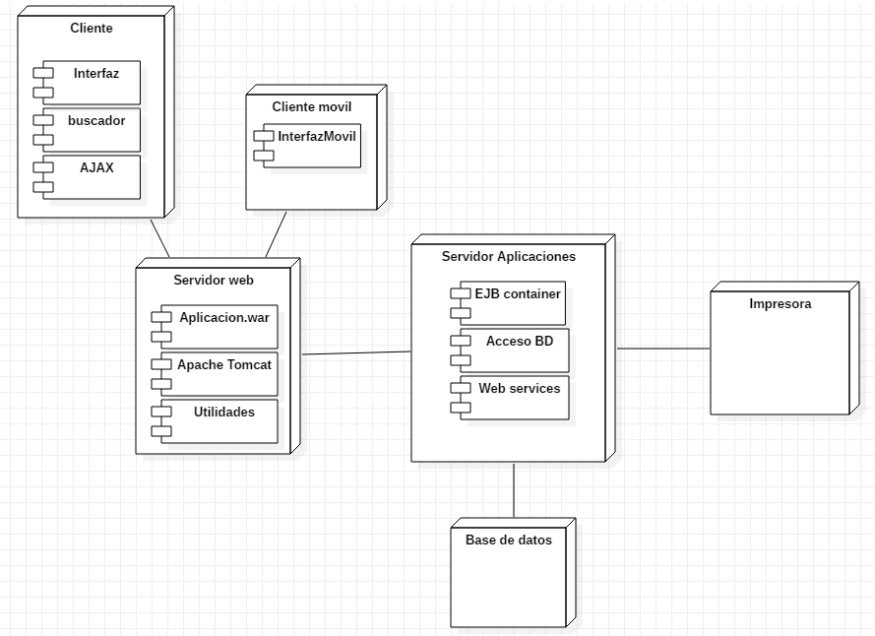


- **Diagrama de objetos**: Muestra los elementos del modelo estático en un momento concreto, habitualmente en casos especiales de un diagrama de clases o de comunicaciones, y está formado por un conjunto de objetos y sus relaciones. Tiene una prioridad ALTA.

- **Diagrama de componentes**: Especifican la organización lógica de la implementación de una aplicación, sistema o empresa, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos. Tiene una prioridad MEDIA.

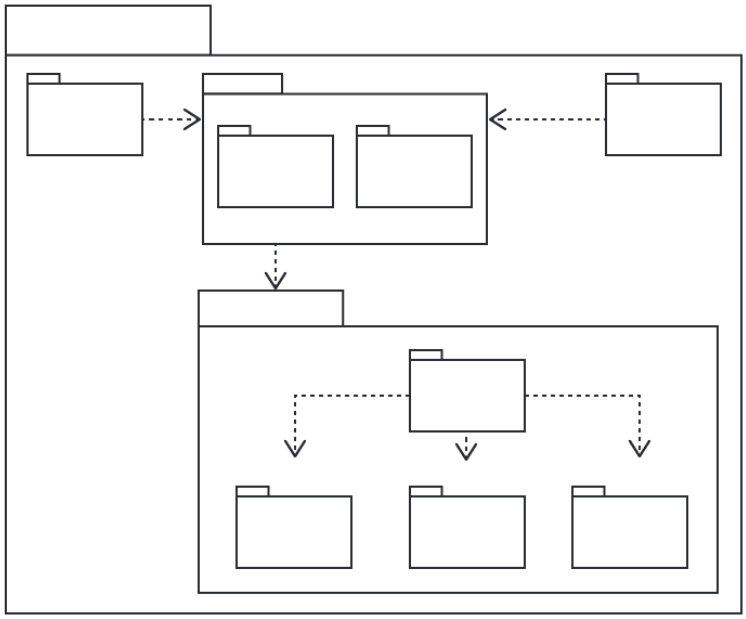


- **Diagramas de despliegue**: Representan la configuración del sistema en tiempo de ejecución. Aparecen los nodos de procesamiento y sus componentes. Exhibe la ejecución de la arquitectura del sistema. Incluye nodos, ambientes operativos sea de hardware o software, así como las interfaces que las conectan, es decir, muestra como los componentes de un sistema se distribuyen entre los ordenadores que los ejecutan. Se utiliza cuando tenemos sistemas distribuidos. Tiene una prioridad MEDIA.



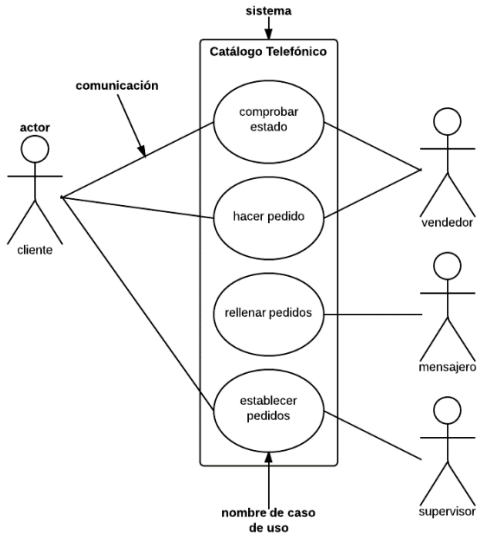
- **Diagrama integrado de estructura** (UML 2.0): Muestra la estructura interna de una clasificación (tales como una clase, componente o caso típico), e incluye los puntos de interacción de esta clasificación con otras partes del sistema. Tiene una prioridad BAJA.

- **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. Tiene una prioridad BAJA.



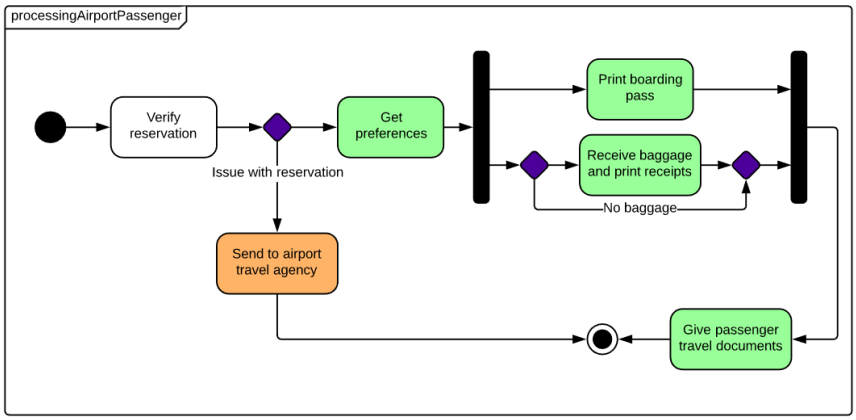
1.3.2 DIAGRAMAS DE COMPORTAMIENTO

- **Diagramas de casos de uso**: Representan 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. Tiene una prioridad MEDIA.

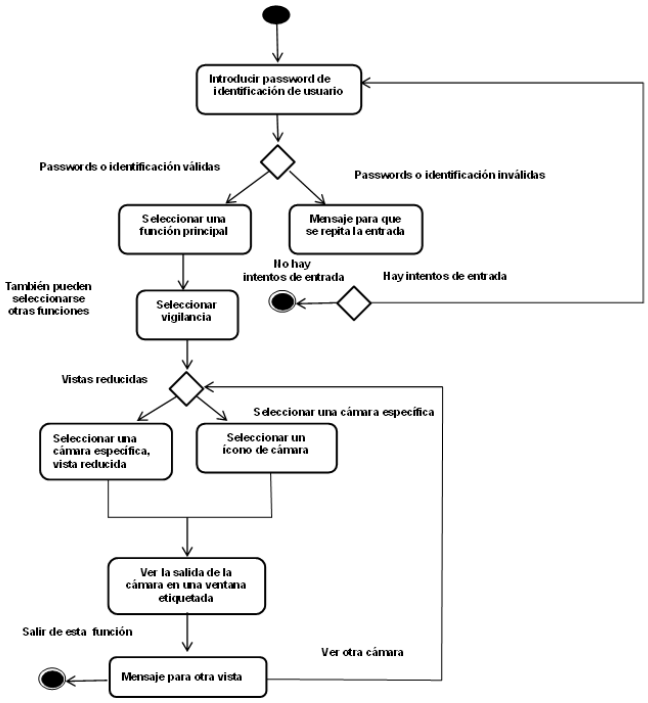


- **Diagramas de estado de la máquina**: Describen el comportamiento de un sistema dirigido por eventos. En el aparecen los estados que pueden tener un objeto o interacción, así como las

transiciones entre dichos estados. Se lo denomina también diagrama de estado, diagrama de estados y transiciones o diagrama de cambio de estados. Tiene una prioridad MEDIA.

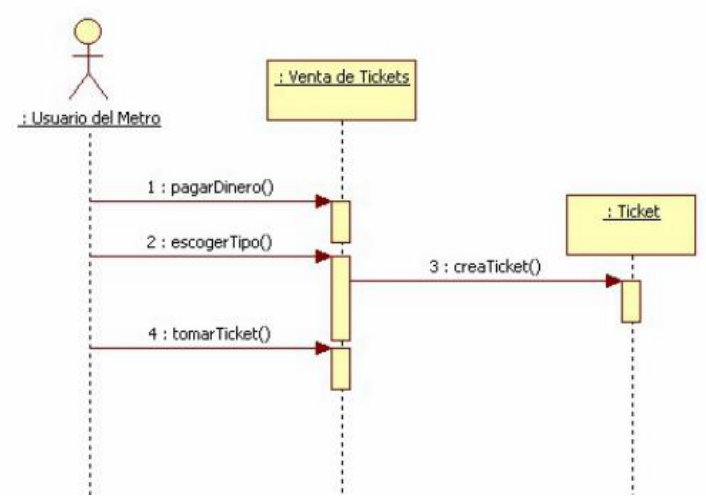


- **Diagrama de actividades**: Muestran el orden en el que se van realizando tareas dentro de un sistema. En él aparecen los procesos de alto nivel de la organización. Incluye flujo de datos, o un modelo de la lógica compleja dentro del sistema. Tiene una prioridad ALTA.

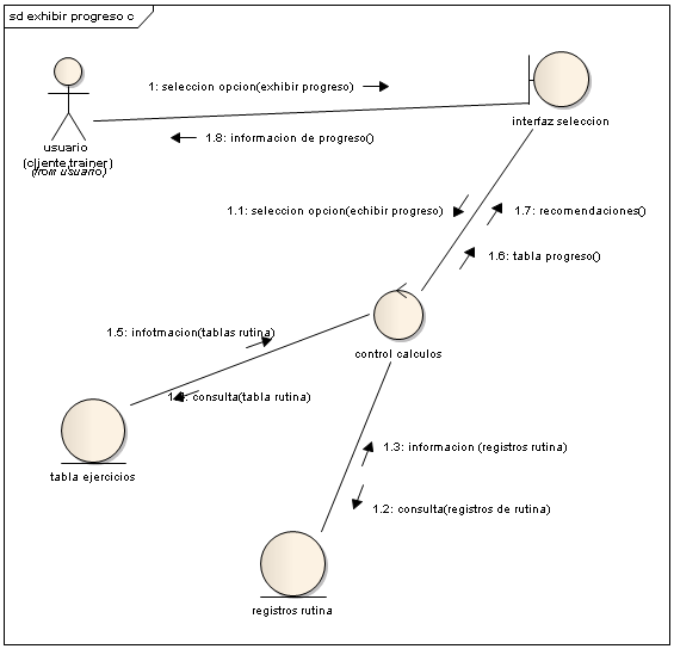


- **Diagramas de interacción:**

  - **Diagramas de secuencia**: Representan la ordenación temporal en el paso de mensajes. Modela la secuencia lógica, a través del tiempo, de los mensajes entre las instancias. Tiene una prioridad ALTA.



- **Diagramas de comunicación/colaboración** (UML 2.0): Resaltan 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. Tiene una prioridad BAJA.



- **Diagrama de interacción**: Muestra un conjunto de objetos y sus relaciones junto con los mensajes que se envían entre ellos. Es una variante del diagrama de actividad que permite mostrar el flujo de control dentro de un sistema o proceso organizativo. Cada nodo de actividad dentro del diagrama puede representar otro diagrama de interacción. Tiene una prioridad BAJA.

- **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. Tiene una prioridad

BAJA.

El objetivo de este tema será enseñar al alumno a modelar el producto software a realizar, y por ello, estará estructurado con este fin.

2. INTRODUCCIÓN Y PARTICIPANTES

Como se ha mencionado anteriormente este tema será una guía para hacer el modelado de un producto software.
A lo largo de la asignatura, cuando se hable del modelado, se hablará de un informe que contenga todos los apartados que vengan a partir de aquí, siguiendo este esquema y estructura.

2.1 INTRODUCCIÓN

Lo primero que se hará será especificar una introducción sobre el producto software que se va a realizar.

Esta introducción contendrá todas las especificaciones del cliente sobre el producto final a desarrollar, es decir, todo lo que se vaya a elaborar a continuación partirá de esta descripción dada, por ello ha de ser clara, concisa y contener toda la información que el cliente crea necesaria para llegar al desarrollo del producto que desea.

Digamos, que, si nos tomamos el modelado como un ejercicio, la introducción sería el enunciado.

2.2 PARTICIPANTES

2.2.1 AUTORES:

Los autores son los individuos que van a realizar el modelado.
Se presentan haciendo uso de una tabla similar a esta:

Participante	**Marcos Unzueta Puente**
Organización	I.E.S Los Sauces
Rol	Analista y arquitecto
Es desarrollador	Sí
Es cliente	No
Es usuario	No
Comentarios	Ninguno

2.2.2 CLIENTES:

Los clientes serán los individuos que han encargado el producto.
Se presentan haciendo una tabla similar a la de autores.

2.2.3 ORGANIZACIONES INVOLUCRADAS:

Las organizaciones interesadas en este desarrollo, ya sea por parte de los clientes o por parte de los autores.
Se presentan haciendo una tabla como la que se muestra a continuación.

Organización	I.E.S Los Sauces
Dirección	AV. DE FEDERICO SILVA , 48 BENAVENTE (ZAMORA)
Teléfono	980 63 06 86
Fax	980 63 08 14
Correo electrónico	
Web	http://ieslossauces.centros.educa.jcyl.es/sitio/
Comentarios	Ninguno

Estas tablas son un prototipo, no un estándar, es decir, los alumnos pueden modificar las filas en función de las necesidades de información, para que finalmente se aporte la mayor cantidad de información posible.

3. OBJETIVOS DEL SISTEMA

Una vez se ha realizado el primer apartado, se procede a utilizar la información expuesta en el apartado de introducción para extraer los objetivos del sistema a desarrollar.

Cuando se habla en este apartado de objetivos, se habla de objetivos específicos, que sean resolubles de manera independiente y que se puedan combinar para resolver el problema original.

Los objetivos se representan con tablas que constan de las siguientes partes:

- **Id del objetivo**: Cada proceso ha de identificarse por un código único.
  Los identificadores de los objetivos comienzan con **OBJ**.

- **Nombre del objetivo**: Nombre descriptivo del objetivo.

- **Versión**: Para tener un control de versiones. Indica el número y la fecha de la versión actual del objetivo.

- **Autores**: Contiene el nombre y la organización de los autores de la versión actual del objetivo. Estos se han definido en el apartado anterior, y de manera ideal, en la documentación se referenciará a la tabla correspondiente al autor definido.

- **Fuentes**: Las fuentes (normalmente clientes) que propusieron la necesidad del objetivo.

- **Descripción**: Se debe completar con la descripción del objetivo.

- **Subobjetivos**: En este campo se han de definir los subobjetivos que dependen del objetivo que está describiendo. En sistemas complejos es probable que se necesite una jerarquía.
  En los casos en los que no sea necesario puede ignorarse este campo.

- **Importancia**: Se indica la importancia del cumplimiento de este requisito para los clientes y usuarios.
  En este campo se puede establecer un valor numérico o algún cómputo de expresiones (vital, importante, normal, quedaría bien …).
  Si aún no se ha establecido la importancia se pueden poner las siglas PD (Por determinar).

- **Urgencia**: Se indica la urgencia del cumplimiento del objetivo.
  Como en el caso anterior, se pueden elegir los valores a establecer, que pueden ser numéricos o expresiones como (inmediatamente, corto plazo, medio plazo, largo plazo …).
  También se puede usar el valor PD.

- **Estado**: Se indica el estado del objetivo desde el punto de vista del desarrollo (En construcción, pendiente de negociación, pendiente de verificación, pendiente de validación [si ya ha sido verificado y está a la espera de validación] o validado [ya ha sido validado por clientes y usuarios]).

- **Estabilidad**: Se estima la probabilidad de que el objetivo sufra cambios en un futuro.

  Se puede medir mediante un valor numérico o mediante una expresión enumerada como alta, media o baja.

  También se puede usar el valor PD.

- **Comentarios**: Cualquier otra información que no encaje en los campos anteriores.

OBJ-001	**Gestión de usuarios**
Versión	1.0 (15/02/2020)
Autores	Marcos Unzueta Puente Leticia Núñez Pérez Gabriel Turrión Hernández
Fuentes	Enunciado del problema
Descripción	El sistema debe gestionar búsquedas, altas, modificaciones y bajas en todos los usuarios que forman parte del sistema.
Subobjetivos	
Importancia	Vital
Urgencia	Corto plazo
Estado	Validado
Estabilidad	Alta
Comentarios	

3.1 EJEMPLOS

3.1.1 GESTIÓN DE UN VIDEOCLUB:

Se va a suponer un producto software que se encargue de gestionar un pequeño videoclub.
Con el objetivo de simplificar un poco el ejemplo y hacerlo más conciso, se han suprimido algunos campos de las plantillas.

OBJ-01	**Gestión de películas**
Descripción	El sistema debe gestionar la información correspondiente a las cintas y películas del videoclub: adquisiciones, retiradas y disponibilidad.

Estabilidad	Alta
Comentarios	

OBJ-02	**Gestión de socios**
Descripción	El sistema debe gestionar la información correspondiente a los socios del videoclub: altas, bajas, modificaciones de datos, sanciones, personas autorizadas y cuentas.
Estabilidad	Alta
Comentarios	

OBJ-03	**Gestión de alquileres**
Descripción	El sistema debe gestionar búsquedas, altas, modificaciones y bajas en todos los usuarios que forman parte del sistema.
Estabilidad	Alta
Comentarios	

3.2 EJERCICIOS

EJERCICIO 1: Realiza el informe de modelado, solo la parte correspondiente a los apartados que hemos visto, del enunciado que se mostrará a continuación:

El sistema para modelar consiste en una primera fase de una aplicación para gestionar de forma completa una explotación agrícola-ganadera.

Estas explotaciones tienen dos partes claramente diferenciadas, pero con una fuerte relación entre sí: una parte agrícola cuyo objetivo es la plantación de cultivos orientados a la producción de pienso; y una parte ganadera dedicada a la producción de cabezas de ganado orientada a la venta de productos cárnicos y derivados.

La explotación puede estar compuesta por diferentes terrenos. Cada uno de los terrenos se denomina con un nombre, tiene un tamaño medido en hectáreas, y se ha de detallar la provincia y el municipio en el que se encuentra. Por otra parte, cada terreno tendrá una finalidad: plantación de cultivo o para ganado extensivo.

En los terrenos agrícolas se podrán plantar cualquier tipo de cultivo, si bien, existirán un máximo de 3 cultivos preferidos (por ejemplo, en un terreno de secano preferentemente podremos plantar trigo, cebada o avena (no tendría sentido plantar tomates); en un terreno de regadío podremos plantar arroz, hortalizas o remolacha (no tendría sentido plantar trigo... sería tirar el dinero y no aprovechar las características del terreno) -esto son sólo ejemplos, podremos plantar cualquier otro producto-).

El cultivo de un terreno puede variar cada año y habrá que registrar qué es lo que se ha cultivado en cada terreno cada año. Cada terreno cultivado produce al final del año productos finales y una cantidad de producción (es decir, un terreno cultivado con la semilla de trigo produce grano de trigo y paja -medidos en toneladas, por ejemplo-). El sistema deberá proporcionar una estimación de producción y realizar recomendaciones de los cultivos para optimizar dicha producción en base a los años anteriores y a los cultivos preferidos para cada terreno.

Con los productos finales de cada terreno se podrá producir pienso propio. Existen diferentes tipos de pienso (para terneros, para vaca adulta, etc.) La diferencia entre cada uno de estos piensos será la proporción y variedad de los productos finales. Por ejemplo, un tipo de pienso podrá tener 50% granos de trigo y 50% granos de cebada. Otro tipo de pienso podrá tener 20% granos de trigo, 30% alfalfa y 50% maíz.

El pienso puede ser propio -como el detallado anteriormente- o comprado a proveedores externos de pienso.

El pienso se almacena en silos, que tienen un identificador, una capacidad (en litros) y una ubicación en un terreno de ganado.

Existe diversa maquinaria (tractores, cosechadoras, empacadoras, abonadoras, etc.) que puede ser utilizada en los diversos terrenos (ya sea agrícolas o de ganado). Cada maquinaria tiene un número de bastidor que la identifica, precio de compra de la máquina, así como la próxima fecha de revisión, y posibles averías registradas. Además, cada máquina tendrá un estado actual (averiada, en funcionamiento, en reparación). Cada posible avería consiste en un comentario descriptivo del problema que le ocurre a la máquina, y un presupuesto del arreglo. El sistema nos deberá proporcionar la información sobre los posibles gastos en una máquina y dar un aviso cuando los presupuestos sobre las posibles averías sumen más que el valor de compra de esta.

Deberá registrarse en qué terreno ha realizado trabajos una determinada máquina, detallando las fechas de inicio y fin de cada uno de los trabajos, así como un texto descriptivo de la tarea realizada y las posibles averías que han sido descubiertas durante la realización de dicha tarea.

La explotación tendrá empleados -de los que se guardará información personal necesaria para el fisco y la Seguridad Social: nombre, apellidos, NIF, domicilio, teléfono, NUSS, etc.- que realizarán tareas en los diferentes terrenos. Se deberá tener registrado los trabajos de cada empleado en cada terreno, guardando la fecha de inicio y la fecha de finalización, así como un texto descriptivo de la tarea realizada.

Durante un año se realizan diversos pedidos a los diferentes proveedores de semillas de la explotación. De los diferentes proveedores (semilleros o de pienso) habrá que guardar todos los datos necesarios para la realización de la facturación (nombre, domicilio fiscal, NIF, teléfono). Los pedidos de semillas irán relacionados con el cultivo plantado en un terreno en ese año, quedando de esta manera referenciado el origen de la semilla plantada en un terreno cada año.

Los terrenos dedicados al ganado tendrán un número de cabezas de ganado (en esta fase sólo vacuno) cada uno. Cada animal está identificado por su número de crotal, número de fuego -si lo tiene-, raza (en esta explotación tan sólo se trabaja con raza morucha, raza limosina y charoláis), sexo (macho o hembra), fecha de nacimiento y peso. Se ha de conocer además los progenitores (madre y padre) de cada animal. La aplicación permitirá realizar búsquedas detalladas en base a la raza de los progenitores. Una búsqueda muy común será encontrar aquellos animales hembra que sean de raza

morucha pura (que serán candidatas para ser madres). Otra búsqueda muy común será encontrar todo aquel animal macho que sea una segunda generación cruzada (su abuela materna es una morucha pura y su abuelo materno un limosín o charoláis. Su madre por tanto será morucha cruzada; por otra parte, su padre será limosín o charoláis). Este animal es muy solicitado por la calidad de su carne. Existen diversos avisos que el sistema lanzará de forma automática cuando un terreno de ganado no tenga animales machos durante más de 4 meses, cuando un terreno de ganado no tenga animales durante más de 2 meses, cuando un terreno de ganado no tenga animales hembra de raza morucha pura durante más de 4 meses, etc.

Cada terreno utilizará el pienso (ya sea comprado o propio) almacenado en sus silos. La aplicación debe mostrar con facilidad diferentes datos estadísticos sobre el consumo de pienso en cada terreno de ganado. Para ello, cada día a las 17:00 se guardará el día sobre el que se toma la medida, la capacidad actual de cada silo, las cabezas de ganado que se encuentran en cada terreno, la temperatura máxima y mínima del día, y los milímetros de precipitaciones caídos ese día.

Las cabezas de ganado se podrán vender a los diferentes tratantes con los que se tiene relación. De cada tratante se conocerán los datos fiscales del mismo. De cada venta habrá que guardar el animal vendido, la fecha de la venta, el precio de venta, el pesaje del animal en el momento de la venta y el objeto de la venta (si es para vida o si es para carne).

Cada vez que nos refiramos a este ejercicio en etapas posteriores del tema lo nombraremos como enunciado 1, para no tener que reescribirlo todas las veces que nos refiramos a él.

EJERCICIO 2: Realiza el informe de modelado, solo la parte correspondiente a los apartados que hemos visto, del enunciado que se mostrará a continuación:

El sistema para modelar consiste en una aplicación tipo booking, que gestione vuelos y alojamientos.

En el caso de los vuelos, el sistema gestiona la información de aerolíneas que han querido formar parte del producto. Cada aerolínea tiene un nombre, una sede y consta de una serie de aviones.

Cada avión tiene un nombre y una capacidad. Cada uno de los aviones realiza una serie de vuelos. Los vuelos tienen una fecha de salida, lugar de origen, un lugar de llegada, un tiempo de duración determinado y un número de plazas ocupadas, así como un precio para cada plaza en el vuelo.

Cada plaza tendrá un número y una letra.

La información del número de plazas que se encuentran disponibles se obtendrá en tiempo real de un API.

La información sobre el precio de cada asiento del vuelo la actualizará un trabajador de la aerolínea manualmente en la aplicación, que también tendrá la posibilidad de modificar todos los valores referentes a su aerolínea.

En el caso de los alojamientos, la aplicación cuenta con aquellos que han decidido formar parte del producto. Existen distintas clasificaciones para el alojamiento (hotel, apartamento o casa particular), hay que valorar cada caso particular, pero cada alojamiento consta de una serie de espacios (una casa completa, una habitación) que tendrán una serie de características específicas (cada espacio ha de

tener al menos tipo que se encuentra dentro de una enumeración, una descripción y un número de plazas) y se podrán alquilar por un tiempo.

Todos los alojamientos tienen, por lo menos, un nombre, una valoración, unas fotos, una descripción y una ubicación.

La información sobre el número de espacios que hay disponibles en el hotel a lo largo del calendario se obtiene en tiempo real de un API.

La información sobre el precio de cada espacio en cada fecha la actualizará un trabajador del alojamiento manualmente en la aplicación, que también tendrá la posibilidad de modificar todos los valores referentes a su alojamiento.

Todos los usuarios han de registrarse previamente en la aplicación y pasar por una pantalla de login.

Los usuarios de la app necesitarán aportar su nombre, apellidos, fecha de nacimiento, sexo, correo electrónico y contraseña.

Los usuarios pueden modificar sus datos personales o eliminar su cuenta de manera permanente.

Existe una opción de recordar contraseña para aquellos usuarios que la hayan olvidado.

El sistema da la opción de visualizar todos los vuelos para una fecha determinada, devuelve una lista ordenada de los vuelos disponibles (que tienen alguna plaza libre) entre esas fechas ordenada de más barato a más caro. Esta funcionalidad da la opción de introducir origen y llegada (mostrará solo los vuelos entre las ubicaciones definidas) o de no introducir ni origen ni llegada y que devuelva simplemente todos los vuelos existentes. El usuario también ha de indicar el número de plazas que desea en el vuelo.

Dese la lista ordenada de vuelos se puede proceder a la reserva de un vuelo.

Cuando hace click en el vuelo se le muestra una pantalla con el dibujo del avión, los asientos libres están en rojo y no se pueden seleccionar y los asientos libres están en verdes. El usuario selecciona los asientos libres que desea y pulsa el botón de reservar.

Al pulsar el botón reservar llevará al usuario a una ventana de pago.

Un usuario tiene sistemas de pago, para poder reservar un vuelo ha de tener al menos uno. El sistema de pago tendrá un número de tarjeta, la clave trasera de la tarjeta, el nombre del propietario y la fecha de vencimiento. Estos sistemas de pago han de poder modificarse, eliminarse o añadirse por parte del usuario.

Antes de que se cargue el dinero al usuario, se vuelve a comprobar mediante el API si las plazas a reservar siguen libres.

El usuario, una vez haya finalizado el pago, habrá realizado una reserva de vuelo. Cada reserva tiene asociada un método de pago de un usuario, un número de plazas (que irán asociadas a su vuelo, avión, compañía…) y un precio pagado.

El sistema permite también generar reservas de alojamientos de una manera similar a la de los vuelos

En el buscador se pondrá fecha de entrada, fecha de salida, ubicación del alojamiento y número de personas. El buscador devolverá una lista de alojamientos que puedan ofrecer el servicio acorde a los parámetros buscados (puede que para números altos de personas tenga que ofrecer varios espacios).

Si el usuario hace click se mostrará la pantalla de pago (este proceso es igual que en los aviones).

La reserva de alojamiento tendrá asociada una serie de espacios, una fecha y un precio pagado.

El usuario tiene la opción de listar todas sus reservas (vuelos y alojamientos) y tiene la opción de cancelarlas cuando desee recibiendo como rembolso el 80% de lo que ha pagado.

Es importante tener en cuenta que los datos de los alojamientos y vuelos son transmitidos a un operario de la app y los introduce a mano, pero se genera una cuenta especial para los alojamientos y aerolíneas que permite la modificación o eliminación de los datos.

Cuando una aerolínea o alojamiento se da de baja, el operario de la app elimina todos sus datos.

Cada vez que nos refiramos a este ejercicio en etapas posteriores del tema lo nombraremos como enunciado 2, para no tener que reescribirlo todas las veces que nos refiramos a él.

4. DIAGRAMA DE CASOS DE USO

Un diagrama de casos de uso se utiliza para modelar los requisitos funcionales del sistema, mostrando así la funcionalidad del mismo.

Requisitos funcionales: Los requerimientos funcionales son declaraciones de los servicios que proveerá el sistema, de la manera en que éste reaccionará a entradas particulares.

A partir de estos diagramas podrán surgir los demás.

Estos diagramas se especifican desde la visión de los agentes externos, describiendo como estos pueden interactuar con el sistema y otorgando una perspectiva exterior a este.

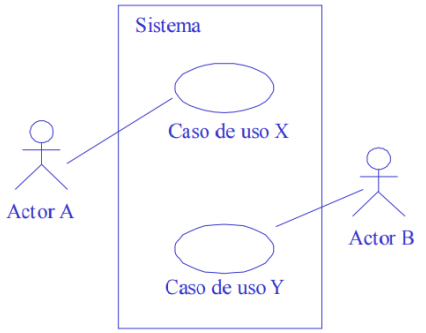
El siguiente paso en el modelado que vamos a hacer va a ser crear un diagrama de casos de uso por cada objetivo y un diagrama de casos de uso general. En muchas ocasiones se usan otras políticas, como hacer diagramas de caso de uso por cada actor.

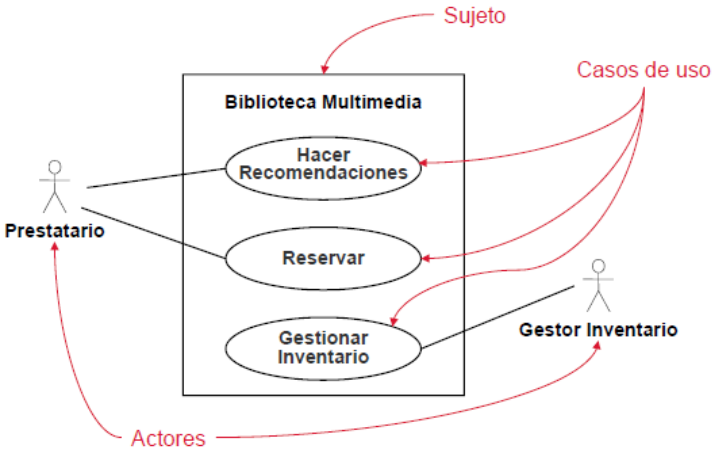
Para poder hacer esto primero vamos a aprender a hacer diagramas de casos de uso.

4.1 SUJETO O SISTEMA

Sistema que se modela

Se representa como un rectángulo que delimita los límites del sistema y contiene los casos de uso, ha de contener un nombre.
Los actores se ubican fuera de los límites del sistema.





4.2 CASOS DE USO

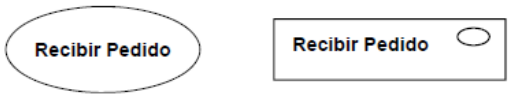
Un caso de uso es un conjunto de acciones llevadas a cabo por el sistema que generan un resultado observable y que dan lugar a un comportamiento del sistema desde el punto de vista del usuario.
El caso de uso es el detalle de un requisito funcional.

Un caso de uso especifica qué hace la aplicación sin entrar en el detalle de cómo lo hace.

Es importante recordar que es necesario hacer una descripción en una plantilla de cada caso de uso (la veremos posteriormente).

Los casos de uso son iniciados por un actor que indica al sistema la activación de este. El caso de uso proporcionará algún valor tangible al usuario.

El caso de uso se representa en UML como una elipse que contiene el nombre del caso de uso (también se puede escribir debajo de la elipse).
Otra notación menos usada es un rectángulo con una elipse en su parte superior derecha y el nombre en su interior.



Notaciones usadas para la representación de casos de uso

El comportamiento de un caso de uso se especificará posteriormente mediante interacciones, actividades, máquinas de estado ...

Es común que los casos de uso contengan variaciones añadiendo a su comportamiento manejo de errores, alternativas y excepciones.

Para proyectos que tengan una dimensión importante, es conveniente organizar los casos de uso en paquetes. Posteriormente veremos cómo se manejan y organizan mediante los diagramas de paquetes.
Un caso de uso puede estar en más de un paquete, es posible que existan relaciones entre casos de uso de diferentes paquetes.

4.2.1 DESCRIPCIÓN

Es importante realizar la plantilla correspondiente a cada caso de uso, ya que, de esta, se extrae una gran cantidad de información que podrá ser utilizada posteriormente para seguir con el modelado, así como con el desarrollo correspondiente.

Para profundizar en el comportamiento de los casos de uso se utilizarán otros diagramas como el de secuencia, del que hablaremos posteriormente.

La plantilla a realizar tendrá los siguientes apartados:

- **Identificador:** Cada caso de uso ha de identificarse por un código único.
  Los identificadores de los casos de uso comienzan con **UC**.
- **Nombre descriptivo:** Nombre descriptivo del caso de uso.
- **Versión**: Para tener un control de versiones. Indica el número y la fecha de la versión actual del caso de uso.
- **Autores**: Contiene el nombre y la organización de los autores de la versión actual del caso de uso.
- **Fuentes**: Las fuentes (normalmente clientes) que propusieron la necesidad del objetivo que dio lugar al caso de uso.
- **Objetivos asociados**: Id de los objetivos asociados al caso de uso.
- **Requisitos asociados**: Id de los requisitos asociados al caso de uso.

- **Descripción**: Si el caso de uso es abstracto debe indicar aquellos casos de uso en los que es incluido o que extiende.
  Si el caso de uso es concreto se ha de indicar el evento de activación que provoca su realización y en caso de que sea incluido desde, o extienda a, se ha de indicar dichos caso de uso.

- **Actores:** Id de aquellos actores que interactúan con el sistema a través del caso de uso.

- **Precondiciones:** Aquellas que deben cumplirse para que pueda llevarse a cabo el caso de uso.

- **Secuencia 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.

- **Postcondiciones:** Las que se cumplen una vez que se ha realizado el caso de uso.

- **Excepciones:** 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.

Secuencia normal	Paso	Acción
	1	El sistema solicita al usuario su nombre de usuario y su clave de acceso
	2	El usuario proporciona el sistema su nombre y su clave de acceso
	3	El sistema comprueba si el nombre de usuario y la clave de acceso son correctas
	4	Si el nombre de usuario y la clave no son correctas, el sistema permite al usuario repetir el intento (pasos 1–3) hasta un máximo de tres veces
	5	Si el nombre de usuario y la clave son correctas, el sistema permite el acceso al usuario
Excepciones	**Paso**	**Acción**
	4	Si el usuario ha intentado tres veces acceder sin éxito, el sistema rechaza el acceso del usuario, a continuación este caso de uso termina

- **Rendimiento:** En este campo puede especificarse el tiempo máximo para cada paso en el que el sistema realice una acción.

- **Importancia:** Se indica la importancia del cumplimiento de este caso de uso para los clientes y usuarios.
  En este campo se puede establecer un valor numérico o algún cómputo de expresiones (vital, importante, normal, quedaría bien …).
  Si aún no se ha establecido la importancia se pueden poner las siglas PD (Por determinar).

- **Urgencia:** Se indica la urgencia del cumplimiento del caso de uso.
  Como en el caso anterior, se pueden elegir los valores a establecer, que pueden ser numéricos o expresiones como (inmediatamente, corto plazo, medio plazo, largo plazo …).
  También se puede usar el valor PD.

- **Estado:** Se indica el estado del caso de uso desde el punto de vista del desarrollo (En construcción, pendiente de negociación, pendiente de verificación, pendiente de validación [si ya ha sido verificado y está a la espera de validación] o validado [Ya ha sido validado por clientes y usuarios])

- **Estabilidad:** Se estima la probabilidad de que el caso de uso sufra cambios en un futuro.
  Se puede medir mediante un valor numérico o mediante una expresión enumerada como alta, media o baja.
  También se puede usar el valor PD.

- **Comentarios:** Cualquier otra información que no encaje en los campos anteriores.

UC-001	**Añadir Empleado**
Versión	1.1 (16/02/2020)
Autores	Marcos Unzueta Puente
Fuentes	Enunciado del problema
Objetivos asociados	OBJ-005 Gestión de empleados
Requisitos asociados	
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando un usuario quiera dar de alta a un nuevo Empleado en el sistema.
Actores	ACT-001 Usuario
Precondición	Estar en la sección Gestión de empleados o haber listado empleados y que el usuario seleccione añadir empleado.
Secuencia normal	1. El sistema muestra un formulario con los datos a rellenar. 2. El actor administrador (ACT-002) rellena los campos requeridos para dar de alta un Empleado (nombre, apellidos, DNI…). 3. El sistema guarda la información referente al empleado. 4. El sistema muestra un mensaje indicando que el proceso de alta se ha realizado con éxito
Poscondición	
Excepciones	1. Si el empleado ya existe (con comprobación del DNI) se le notifica al usuario de que ya hay un empleado con esa identificación.
Rendimiento	
Frecuencia	
Importancia	Alta
Urgencia	Alta
Estado	Completado
Estabilidad	Alta
Comentarios	Ninguno

Sugerencias para escribir casos de uso:

- Mantener los casos de uso breves y sencillos
- Centrarse en el qué, no en el como

EJERCICIO 3: Escribe las plantillas de eliminar empleado o modificar empleado análogas al caso de uso del ejemplo anterior.

4.3 ACTORES

Un actor es algo o alguien externo al sistema e interactúa con él.

Los actores suelen ser usuarios del sistema, temporizadores, dispositivos, incluso, otros sistemas que toman parte en las funciones del principal.

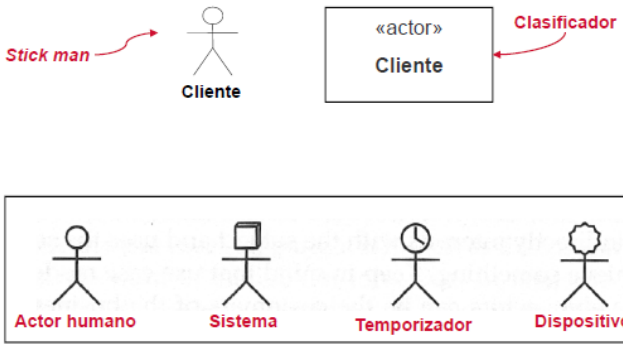
Es importante tener en cuenta que lo que denominamos actor, representa el "papel" o "rol" que un usuario lleva a cabo respecto a su forma de interactuar con el sistema, es decir, una sola persona física puede actuar como actores diferentes.

Los actores se representan mediante el icono de "stick man" o monigote con el nombre del actor cerca del símbolo. Por convenio se suele poner encima o debajo.

Otra manera menos común de representarlo es mediante un rectángulo con el estereotipo <<**actor**>> y el nombre de este.

Los nombres de los actores, por convenio, deben empezar por mayúscula.

Es común y conveniente utilizar otros símbolos para representar tipos de actores, por ejemplo, actores no humanos.



Para establecer los actores implicados hay que identificar las entidades interesadas en interactuar con el sistema.

4.3.1 DESCRIPCIÓN

Los actores también han de describirse, al igual que se hace con los objetivos o los casos de uso.
La plantilla a realizar tendrá los siguientes apartados:

- **Identificador:** Cada actor ha de identificarse por un código único.
  Los identificadores de los actores comienzan con **ACT**.
- **Nombre descriptivo:** Nombre descriptivo del actor.
- **Versión**: Para tener un control de versiones. Indica el número y la fecha de la versión actual del actor.
- **Autores**: Contiene el nombre y la organización de los autores de la versión actual del caso de uso.
- **Fuentes**: Las fuentes (normalmente clientes) que propusieron la necesidad del actor.
- **Descripción**: Rol o papel que representa el actor respecto al sistema.
- **Comentarios:** Cualquier otra información que no encaje en los campos anteriores.

ACT-002	**Administrador**
Versión	1.0(16/02/2020)
Autores	Marcos Unzueta Puente
Fuentes	Enunciado del problema
Descripción	Este actor representa el administrador del sistema, el cual tiene todos los privilegios en el mismo.
Comentarios	

EJERCICIO 4: Identifica los actores que puedan tomar parte en el enunciado 1 y haz las plantillas correspondientes.

EJERCICIO 5: Identifica los actores que puedan tomar parte en el enunciado 2 y haz las plantillas correspondientes.

4.4 RELACIONES

Existen varios tipos de relaciones de UML en los diagramas de casos de uso, que indican conexión entre los elementos que enlaza.

Relación	Notación
Asociación	
Generalización	
Inclusión	«include»
Extensión	«extend»
Realización	

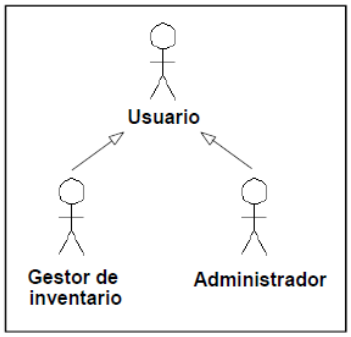
4.4.1 RELACIONES ENTRE ACTORES

Es posible establecer relaciones de **generalización** entre actores.

Los actores más específicos heredan el comportamiento del actor general (o padre) y lo complementan.

Una instancia del actor descendiente se puede usar en los casos en los que se espere una instancia del actor antecesor.

Las relaciones de generalización se representan como una flecha cerrada y rellena de blanco que van de los actores más específicos al general.

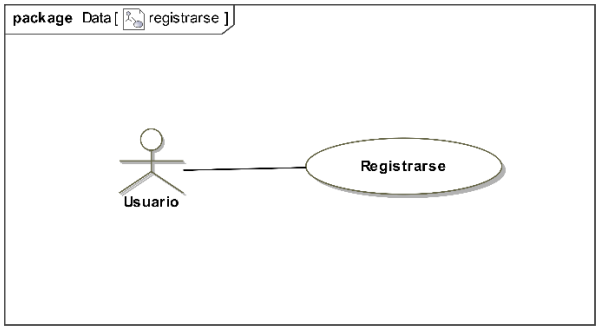


4.4.2 RELACIONES ENTRE ACTORES Y CASOS DE USO

Los actores se conectan a los casos de uso haciendo uso de un tipo de relaciones llamadas **asociaciones**. Estas representan una conexión entre el actor y el caso de uso que inicia.

Generalmente la asociación es una relación uno a uno sin dirección, lo que quiere decir que una instancia de actor se comunica con una instancia de caso de uso y que esta comunicación se puede realizar en ambas direcciones (de actor al caso de uso o del caso de uso al actor).

Se representa mediante una línea continua que une al actor con un caso de uso.



EJERCICIO 6: Genera un diagrama de casos de uso con los siguientes datos:

- El sistema es un restaurante.
- Al restaurante pueden ir comensales normales o críticos de cocina.
- Tanto los comensales normales como los críticos de cocina pueden pedir comida, comer, pagar e ir al baño.
- Los críticos de cocina, además, pueden evaluar el servicio.

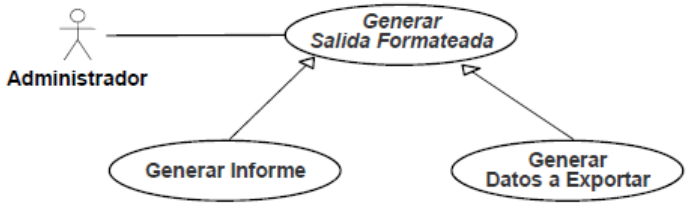
4.4.3 RELACIONES ENTRE CASOS DE USO

- **GENERALIZACIÓN**

La idea y el comportamiento es similar al de la generalización de actores, de tal manera que un caso de uso también se puede especializar en uno o más casos de uso hijos.

Los hijos heredan las relaciones y el comportamiento del caso de uso padre, al cual, puede agregar atributos y operaciones propios.

La notación es idéntica a la generalización de actores.



EJERCICIO 7: Genera un diagrama de casos de uso con los siguientes datos:

- El sistema es un restaurante.
- Al restaurante pueden ir comensales normales o críticos de cocina.
- Tanto los comensales normales como los críticos de cocina pueden pedir primer plato, pedir segundo plato, pedir postre, comer, pagar e ir al baño.
- Los críticos de cocina, además, pueden evaluar el servicio.

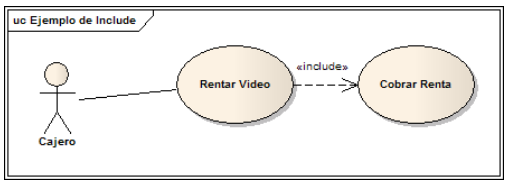
- **INCLUSIÓN**

Los casos de uso pueden incorporar el comportamiento completo de otro caso de uso general.

Esta relación tiene como objetivo indicar la reutilización de comportamientos o porciones de comportamientos comunes a varios casos de uso.

La relación de inclusión se representa mediante una flecha formada por una línea discontinua y con la cabeza abierta, a la cual se añade el estereotipo **«include»**.

Cuando se establece una relación del tipo "include" entre dos casos de uso, estamos diciendo que el caso de uso del que sale la flecha (el caso de uso base) incluye al que apunta (el caso de uso incluido). Esto quiere decir que sin el caso de uso incluido el base no podría funcionar bien



En el ejemplo anterior, para una venta en la caja, la venta no puede considerarse completa si no se realiza el proceso para cobrarla en ese momento. El caso de uso "Cobrar Renta" está incluido en el caso de uso "Rentar Video", o lo que es lo mismo "Rentar Video" incluye (<<include>>) "Cobrar Renta".

EJERCICIO 8: Genera un diagrama de casos de uso con los siguientes datos:

- El sistema es un vendedor.
- El vendedor tiene como única posibilidad llamar a un caso de uso llamado Realizar Venta.
- Realizar venta implica imprimir comprobante y cobrar venta.
- Cobrar venta se puede realizar cobrando al contado, cobrando mediante tarjeta de crédito o cobrando con cheque.
- Cobrar crédito implica que el usuario autorice el pago con tarjeta.

- **EXTENSIÓN**

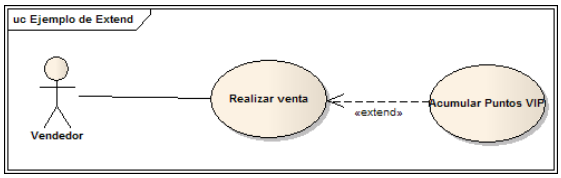
Este tipo de relaciones implican que un caso de uso específico añada acciones opcionales al comportamiento de un caso de uso general. El caso de uso extendido no tiene por qué incluir todo el comportamiento del caso de uso que se extiende.

La diferencia principal respecto a la inclusión reside en que en el "extend" existen situaciones en las cuales el caso de uso de extensión no es indispensable que ocurra, en cambio en la inclusión es necesario que ocurra el caso incluido.

La relación de extensión entre dos casos de uso se representa visualmente como una flecha discontinua con la punta abierta que va desde el caso de uso específico al general.

Ejemplo: Imaginemos una venta en un negocio como el supermercado "Dia". En este supermercado existen carnets que permiten acumular puntos a la hora de realizar las compras,

pero no todo el mundo los tiene.
Un vendedor a la hora de llevar a cabo la acción “Realizar Venta” pregunta al cliente si tiene el carnet VIP o no, en el caso de que lo tenga ha de llevar a cabo la acción de “Acumular Puntos VIP”, en el caso de que no lo tenga finaliza la venta.

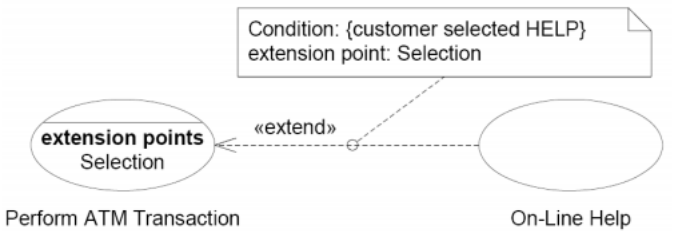


A la hora extender un caso de uso puede ser una buena práctica definir puntos de extensión (extension points), esto decir, especificar el punto del caso de uso donde insertar la extensión para ampliar la funcionalidad siempre ateniéndose las condiciones especificadas.

Los puntos de extensión se disponen en forma lista, la cual se muestra dentro del caso de uso extendido.

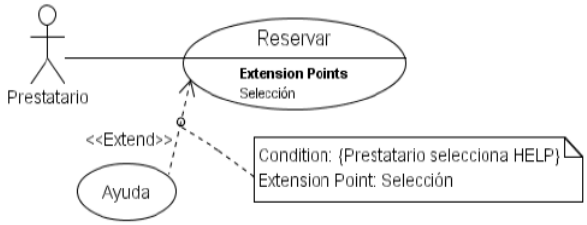
Por otra parte, es conveniente detallar la condición que provoca la ejecución de la extensión. Esto visualmente se representa como una nota conectada mediante una línea discontinua a la relación de dependencia que se refiere.

Cuando el sistema está ejecutando un caso de uso que cuenta con un punto de extensión, se evalúan las condiciones asociadas. En el caso de que se cumpla la condición, se procede a ejecutar la extensión asociada y, una vez terminada la extensión, el flujo de ejecución retorna al caso de uso base que sigue con su ejecución original.

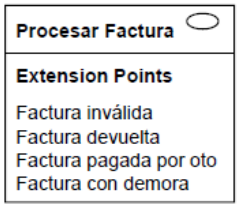


La extensión se representa mediante una flecha discontinua con la punta abierta que va desde el caso de uso que extiende al extendido, el estereotipo usado para señalizar la flecha es **<<extend>>**. Opcionalmente es posible incluir una nota con las condiciones y las referencias a los puntos de extensión.

Los puntos de extensión se representan como una cadena de texto dentro del propio caso de uso siguiendo la sintaxis: **< nombre > [: <explicación>]**



En el caso de que haya una gran cantidad de puntos de extensión resulta conveniente usar la representación alternativa de los casos de uso.



Como reflexión, es de vital importancia tener en cuenta que el fin de las relaciones de extensión e inclusión no consiste en incentivar la división de los casos de uso de manera masiva e innecesaria.

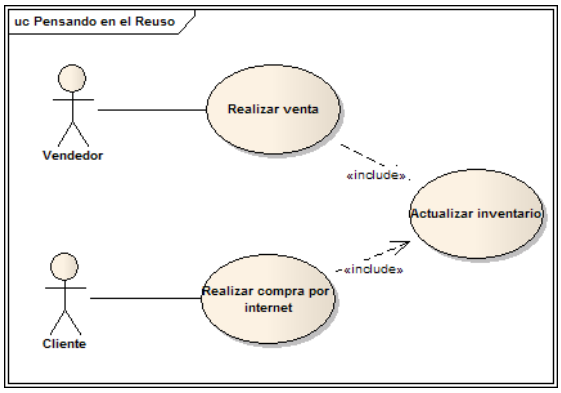
Es necesario ponderar si un división de la funcionalidad del caso de uso va a acarrear más ventajas o más inconvenientes y actuar en consecuencia.

¿Cuándo usar este tipo de relaciones?:

Hay pasos que son iguales en dos o más casos de uso, por tanto, la extracción de esa funcionalidad a un solo caso de uso permitiría reutilizar código, mejorando así sustancialmente el desarrollo.
Tener secciones idénticas sin reutilizar en programación puede provocar que sea necesario escribir y dar mantenimiento a esos pasos en los documentos asociados a cada uno de ellos, a la vez que implica correr el riesgo de que esos pasos se diseñen, programen y prueben de maneras diferentes y con esfuerzos.

La existencia de código duplicado puede dar lugar a errores, ya que es imprescindible realizar varias modificaciones paralelas cada vez que se quiere modificar el código.

No sería inteligente trabajar varias veces en lo mismo.



EJERCICIO 9: Genera un diagrama de casos de uso con los siguientes datos:

- El sistema representa una app tipo Just Eat
- Una persona no identificada ha de darse de alta.
- Desde darse de alta si se hace click en el botón de ayuda se visualizará una pestaña de ayuda y i se pulsa el botón de información sobre privacidad se mostrará una pestaña de información sobre la privacidad.
- El usuario no registrado puede hacer log in, este log in devolverá un error cuando se introduzcan datos incorrectos.
- Cuando se ha hecho el log in la persona no identificada pasa a ser un usuario.

- El usuario puede ver los restaurantes disponibles y la comida disponible en cada restaurante.
- Viendo la comida disponible dentro de un restaurante puede pedir comida al mismo. Cuando pida comida se activará la función que revisará si el usuario tiene fondos suficientes.
- El pago se puede hacer con tarjeta o con paypal.

EJERCICIO 10: Genera un diagrama de casos de uso con los siguientes datos:

- Un juego de dos jugadores en el que cada uno participa con su propia terminal.
- Cuando quieren empezar una partida uno ha de iniciar la partida y otro conectarse.
- En el juego los jugadores pueden mover nave o disparar.
- Cuando uno de los jugadores acierta el disparo la partida finaliza.

EJERCICIO 11: Genera un diagrama de casos de uso con los siguientes datos:

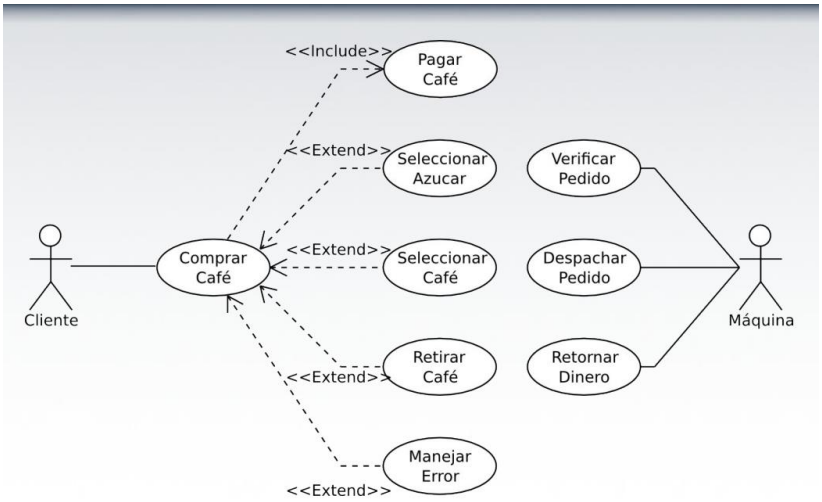
- Una empresa de transporte de mercancías quiere hacer una supervisión de todos los camiones de los que dispone. Para eso ha diseñado una nueva aplicación web llamada Güeraryu.
- En esta aplicación un administrador registra unos GPS especiales con 4G y les asigna un nombre. Básicamente el nombre es la matrícula del camión o furgoneta.
- Para la aplicación un GPS y un vehículo es la misma cosa.
- El administrador programa unas rutas para cada transporte con una fecha/hora de salida, fecha/hora de llegada (prevista). Y de esa manera se puede conocer qué vehículos están operativos en un momento determinado.
- Estos GPS envían cada 5 minutos la localización exacta del dispositivo a la web mediante comunicación 4G.
- El administrador puede acceder a un mapa en el que aparecen en el mismo geolocalizados los distintos camiones y dependiendo de su ubicación puede modificar la ruta. Dentro del mismo mapa puede elegir mostrar solamente aquellos vehículos que tengan una ruta programada en ese momento. También puede pedir mostrar aquellos vehículos que no tienen ninguna ruta programada y de esa manera ver dónde se ubican por si hiciese falta realizar un porte. El administrador generalmente elige el vehículo más cerca del origen de la ruta.

EJERCICIO 12: Genera un diagrama de casos de uso con los siguientes datos:

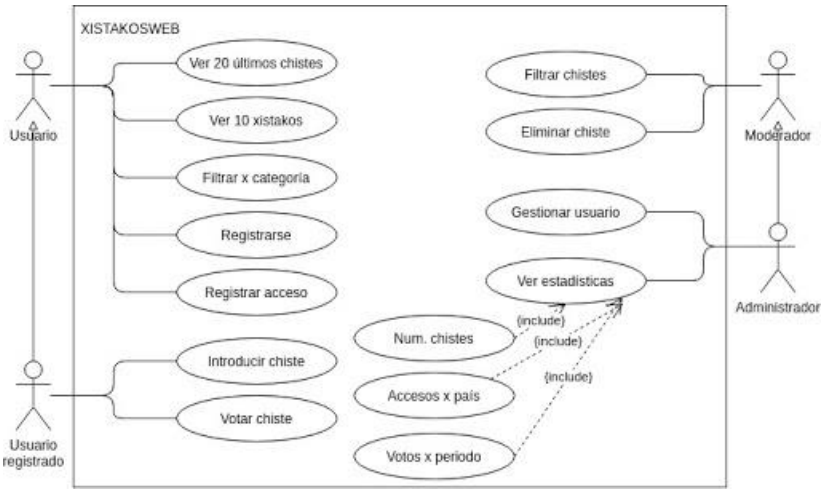
- Se desea desarrollar un sistema de encuentros virtuales (parecido a un chat).
- Cuando se conecta al servidor, un usuario puede entrar o salir de un encuentro.
- Cada encuentro tiene un manager.
- El manager es el usuario que ha planificado el encuentro (el nombre del encuentro, la agenda del encuentro y el moderador del encuentro).
- Cada encuentro puede tener también un moderador designado por el manager.
- La misión del moderador es asignar los turnos de palabra para que los usuarios hablen.
- El moderador también podrá dar por concluido el encuentro en cualquier momento.

- En cualquier momento un usuario puede consultar el estado del sistema, por ejemplo los encuentros planeados y su información.

EJERCICIO 13: Explica el sistema que representa el siguiente diagrama de casos de uso:



EJERCICIO 14: Explica el sistema que representa el siguiente diagrama de casos de uso:



5. SOFTWARE DE CREACIÓN DE DIAGRAMAS

Como parece lógico un trabajo tan tedioso como diseñar decenas de diagramas debía tener una adaptación al software, que nos permitiera reflejar de manera informática nuestro trabajo.

Existen programas de pago muy completos, pero la idea en este curso es la de trabajar con software libre.

Bajo mi punto de vista el mejor programa para desarrollo de diagramas UML es Visual Paradigm, no obstante, hasta donde yo tenía entendido este programa era de pago. Recientemente descubrí que esto no es así del todo y que tiene una versión community.

https://www.visual-paradigm.com/download/community.jsp?platform=windows&arch=64bit

Como nunca he trabajado con esta versión no sé si tendrá alguna limitación, por ello es conveniente tener en cuenta algunas alternativas gratuitas como Modelio y StarUML.

En clase aprenderemos como usar estos programas.

EJERCICIO 15: Realiza los diagramas de casos de uso de cada objetivo del enunciado 1 con sus descripciones correspondientes.

EJERCICIO 16: Realiza los diagramas de casos de uso de cada objetivo del enunciado 2 con sus descripciones correspondientes.

6. DIAGRAMA DE PAQUETES

Son diagramas estructurales cuyo fin es mostrar cómo se organizan y distribuyen en paquetes algunos elementos de un modelo.

Hasta ahora hemos definido los participantes y organizaciones del sistema, los objetivos que pretende cumplir, los actores que forman parte en este, hemos generado los diagramas de casos de uso correspondientes a cada objetivo llegando así a ver sus requisitos funcionales para posteriormente definirlos.

Ahora toca ver como se distribuirían estos casos de uso de manera general, dando una visión global del sistema.

Posteriormente se volverán a usar diagramas de paquetes con otros fines.

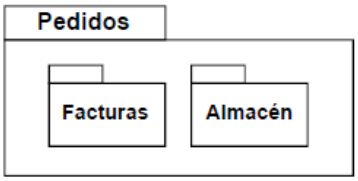
6.1 PAQUETES

Un paquete representa una agrupación de elementos UML relacionados.
Estos elementos pueden ser diagramas, documentos, clases o, incluso, otros paquetes, a los cuales proporciona un espacio de nombres.

Los paquetes se organizan siguiendo jerarquías, de tal manera que el paquete raíz será el que contiene a todos los paquetes, que en su suma contendrán todo el sistema.

Los paquetes, en UML, se representarán como un rectángulo grande que posee un rectángulo pequeño situado en su esquina superior izquierda. El nombre puede aparecer tanto en el rectángulo grande como en el pequeño, no obstante, si el paquete tiene en su interior dibujados otros paquetes conviene poner el nombre en

rectángulo pequeño por claridad, así como situar el nombre en el interior de los paquetes pequeños.



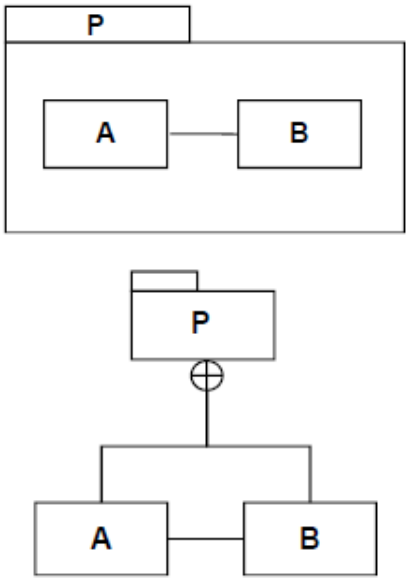
6.2 ELEMENTOS EMPAQUETABLES

Son elementos con nombre que un paquete posee de forma directa.

La visibilidad de los elementos de un paquete indica si son accesibles desde fuera de este.
Se representa colocando delante del nombre del elemento los signos:

+ para visibilidad pública
- para visibilidad privada

Existen dos formas para representar los elementos que contiene un paquete:



6.3 RELACIONES

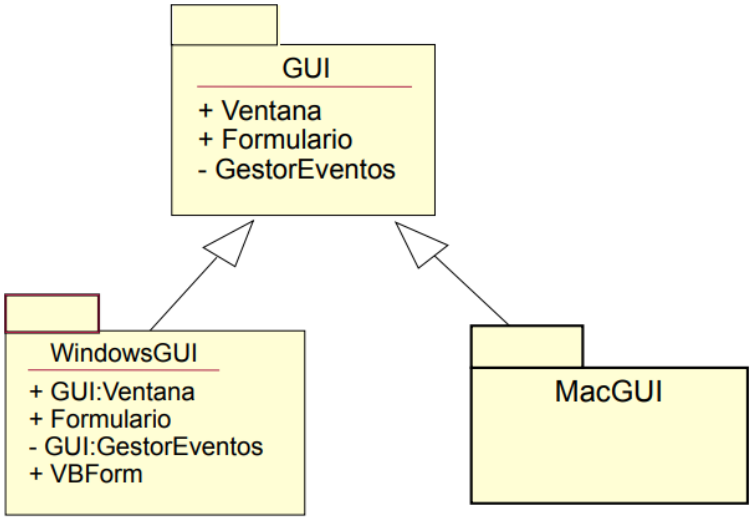
Las relaciones permitidas entre paquetes son generalización, dependencia y fusión.

6.3.1 GENERALIZACIÓN

La generalización es similar a la que aplica en los diagramas de casos de uso, de tal manera que lo que representa esta relación es la herencia.

Un paquete especializado puede usarse en sustitución de uno más general, del cual hereda.

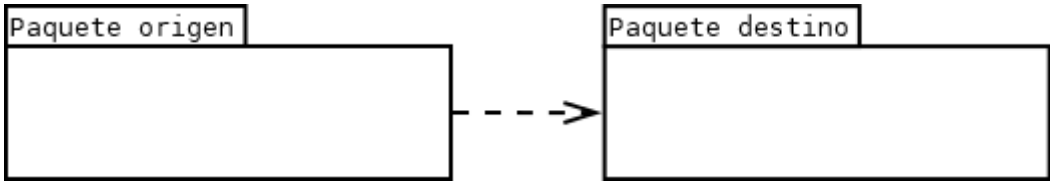
La generalización se representa como en el caso de los diagramas de caso de uso, mediante una flecha cerrada y hueca.



6.3.2 DEPENDENCIA

Las dependencias entre paquetes realmente representan las relaciones existentes entre los elementos que pertenecen a los propios paquetes y que indican que un paquete necesita los elementos de otro para poder funcionar correctamente.

La dependencia se representa mediante el dibujo de una flecha formada por una línea discontinua y con la cabeza abierta, que va desde el paquete que requiere los elementos hasta el paquete que los ofrece.



Existen a su vez varios tipos de relación de dependencia:

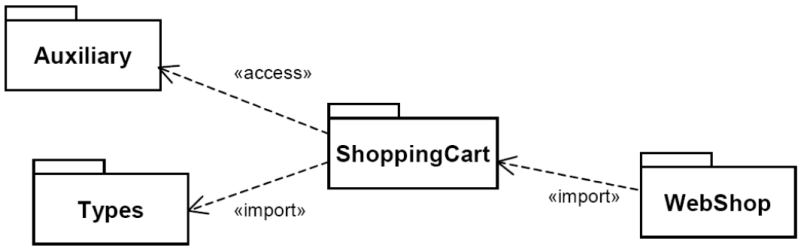
- **Importación**: Brinda a los paquetes la posibilidad importar elementos de otros, de tal manera que el paquete de origen añada el contenido público del paquete destino.

  Para remarcar que la relación de dependencia se trata de una importación se utiliza el estereotipo <<**import**>> en la relación.

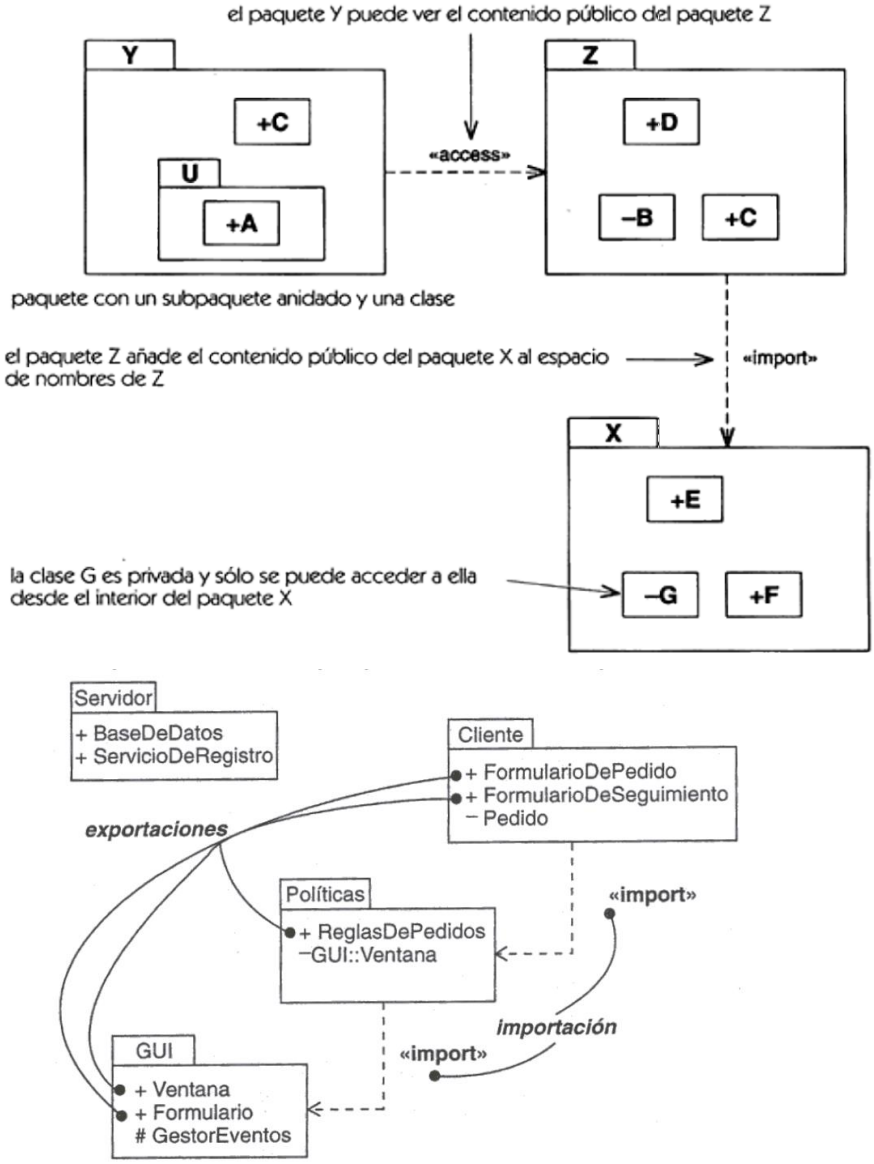
- **Acceso**: Permite a los paquetes acceder a los elementos de otro, pero sin importar el contenido.

  Para remarcar que la relación de dependencia se trata de un acceso se utiliza el estereotipo <<**access**>> en la relación.

- **Exportación**: La parte pública de un paquete son sus exportaciones. No es necesario marcar visualmente las exportaciones.

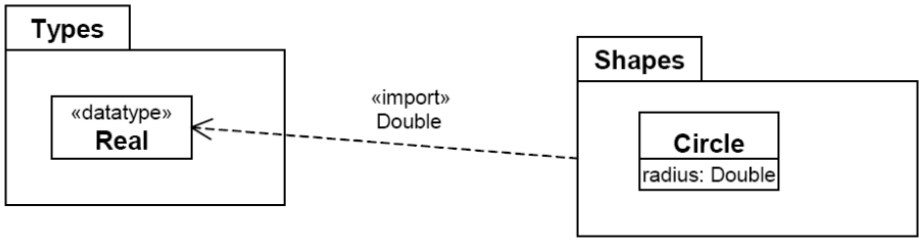


En el caso del ejemplo, los elementos en Types son importados a ShoppingCart.
Los elementos en Types son importados también a WebShop (por transitividad).
Los elementos de Auxiliary sólo son accedidos desde ShoppingCart y, por tanto, no pueden usarse sin calificar desde WebShop.



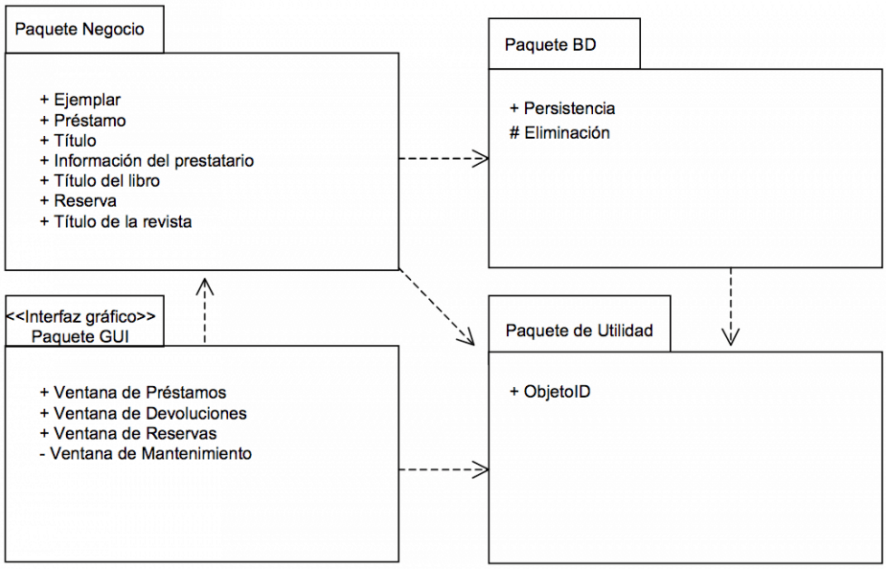
UML ofrece la posibilidad de importar o acceder a elementos de un paquete en vez de al conjunto completo de elementos de un paquete.

Así mismo, se puede asignar un alias a un elemento importado/accedido.



En este ejemplo, el tipo Types:Real está disponible en el paquete Shapes con el nombre Double.

EJERCICIO 17: Explica el sistema que está representando este diagrama de paquetes.

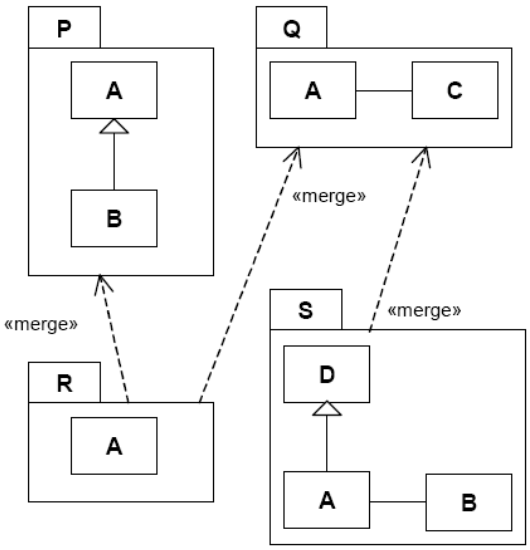


6.3.3 FUSIÓN

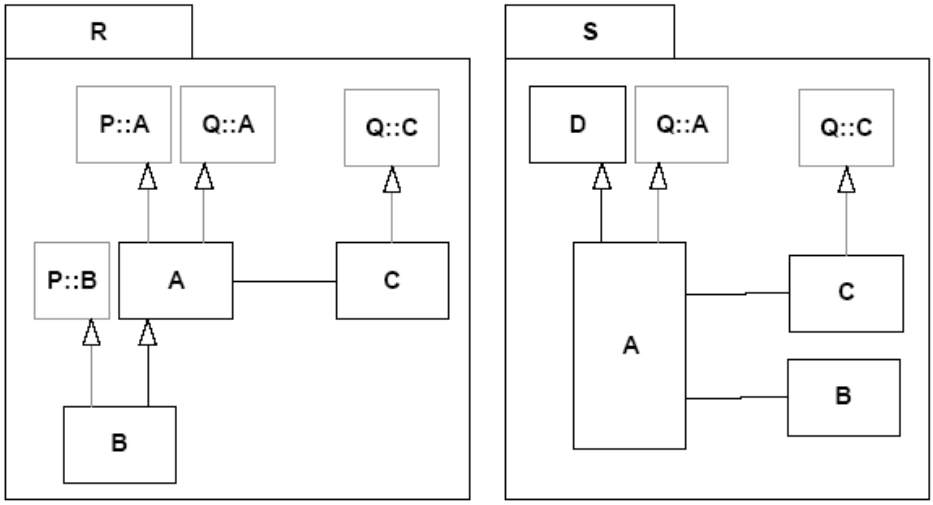
Relación existente entre dos paquetes mediante la cual el contenido del paquete origen pasa a extenderse con el contenido del paquete destino.

Para remarcar que la relación de dependencia se trata de una fusión se utiliza el estereotipo **<<merge>>** en la relación.

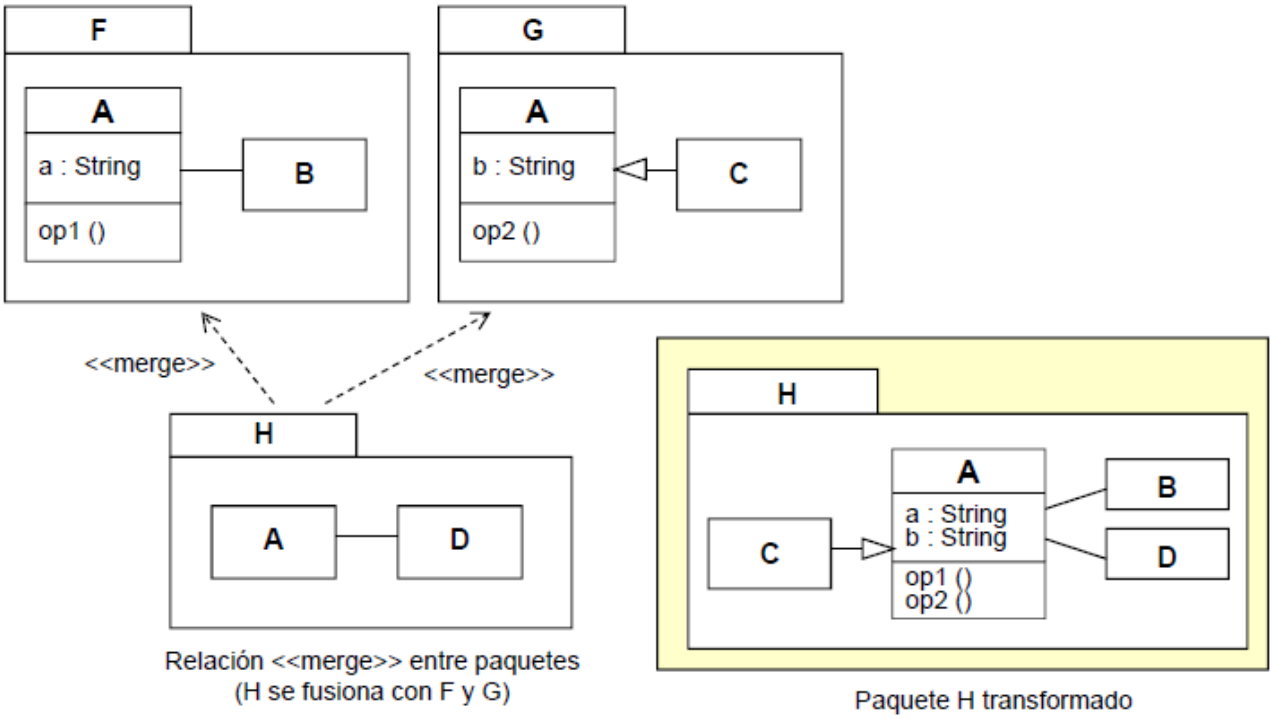
Ejemplo:



Resultado:



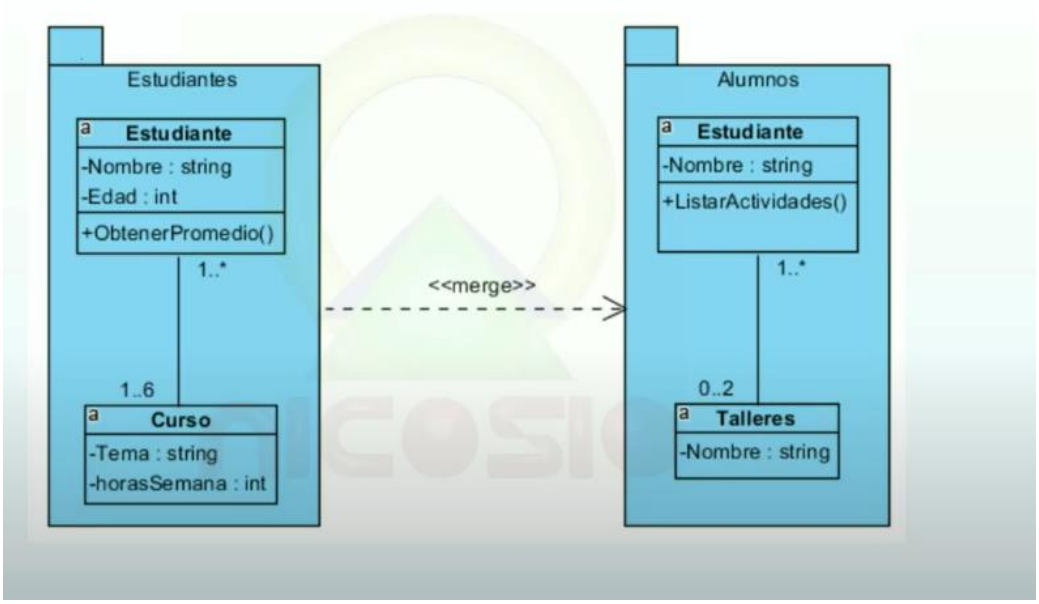
Otro ejemplo:



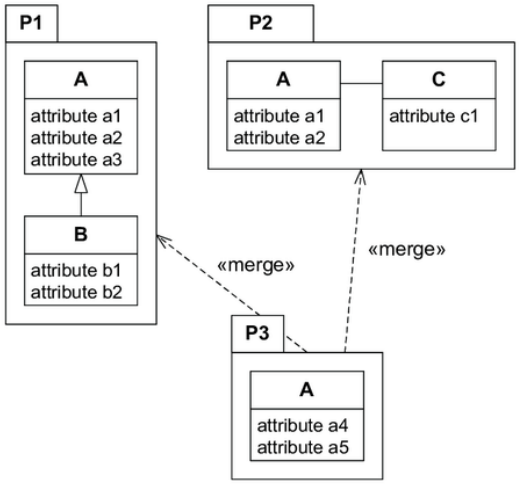
Relación <<merge>> entre paquetes
(H se fusiona con F y G)

Paquete H transformado

EJERCICIO 18: Dibuja el resultado de realizar merge en este diagrama de paquetes.



EJERCICIO 19: Dibuja el resultado de realizar merge en este diagrama de paquetes.



EJERCICIO 20: Dibuja el diagrama de paquetes que se especifica a continuación:

- Vamos a desarrollar el **API Java SE 7**.
- Contiene las siguientes interfaces: Swing, Java 2D, AWT, Accessibility, Input, Image I/O, Print, Sound.
- Contiene los siguientes elementos de integración: IDL, JDBC, JNDI, RMI, RMI-IIOP, Scripting.
- Contiene las siguientes bases: Lag and Util, Collections, Concurrency, Regular Expressions, Reflection, Ref Objects, Logging, JAR, Zip, Instrumentation, Management, Versioning, Preferences, JVM.
- Otras bases extendidas: Beans, Internecionalization, I/O, JMX, JNI, Math, Networking, Endorsed Override, Security, Object Seralization, Extensions, JAXP, JAXB, JAX-WS, SAAJ.

EJERCICIO 21: Realiza los diagramas de paquetes de los casos de uso correspondientes al enunciado 1.

EJERCICIO 22: Realiza los diagramas de paquetes de los casos de uso correspondientes al enunciado 2.

7. REQUISITOS DE INFORMACIÓN

El objetivo de estos requisitos consiste en identificar todos los datos con los que trabaja la organización y que soportan información

En nuestro modelado, el siguiente punto consistirá en identificar y documentar los requisitos de información. Las plantillas que contendrán los requisitos de almacenamiento y gestión de información seguirán el patrón mostrado a continuación.

- **Identificador:** requisito ha de identificarse por un código único.
  Los identificadores de los actores comienzan con **IRQ**.

- **Nombre descriptivo:** Nombre descriptivo del requisito.

- **Versión**: Para tener un control de versiones. Indica el número y la fecha de la versión actual del requisito.
- **Autores**: Contiene el nombre y la organización de los requisitos de la versión actual del caso de uso.
- **Fuentes**: Las fuentes (normalmente clientes) que propusieron la necesidad del actor.
- **Objetivos asociados**: Ha de contener una lista con los objetivos asociados al requisito. Los objetivos a cumplir son los que darán sentido a la existencia de estos requisitos.
- **Requisitos asociados**: En este campo se indicarán otros requisitos que estarán asociados al requisito de información que se está describiendo.
- **Descripción**: Ha de concretar el concepto relevante sobre el que se debe almacenar información.
- **Datos específicos**: Contiene una lista con los datos específicos asociados al campo relevante. En esta sección se han de incluir todos aquellos datos que se consideren oportunos (tipo, descripción, restricciones, ejemplos, etc.).
- **Importancia**: Similar a las plantillas anteriores.
- **Urgencia**: Similar a las plantillas anteriores.
- **Estado**: Similar a las plantillas anteriores.
- **Estabilidad**: Similar a las plantillas anteriores.
- **Comentarios:** Cualquier otra información que no encaje en los campos anteriores.

IRQ-001	**Alumno**
Versión	1.0 (29/05/2016)
Autores	Marcos Unzueta Puente
Fuentes	Enunciado del problema
Objetivos asociados	OBJ-005 …
Requisitos asociados	UC-001 … UC-002 … UC-003 … UC-004 … UC-005 …
Descripción	El sistema deberá almacenar la información relevante a los alumnos.

Datos específicos	Nombre Primer apellido Segundo apellido Número de teléfono Correo de la junta …
Importancia	Importante
Urgencia	Alta
Estado	Completado
Estabilidad	Alta
Comentarios	Ninguno

EJERCICIO 23: Indica la información que se puede recopilar sobre un socio en un sistema software que gestione un videoclub.

EJERCICIO 24: Indica los requisitos de información que pueden darse en un sistema software que gestione un videoclub.

EJERCICIO 25: Indica los requisitos de información que pueden darse en un sistema software que gestione un hospital de mascotas.

EJERCICIO 26: Indica los requisitos de información que pueden darse en un sistema software que gestione las competiciones nacionales correspondientes a un deporte.

EJERCICIO 27: Realiza las tablas de requisitos de información correspondientes al enunciado 1.

EJERCICIO 28: Realiza las tablas de requisitos de información correspondientes al enunciado 2.

8. REQUISITOS NO FUNCIONALES

Se trata de condiciones que el cliente impone al sistema pero que no se corresponden con información a guardar ni con funciones a realizar.
Estos requisitos se corresponden, generalmente, con características de funcionamiento, que, van asociados, principalmente con aspectos referentes a la calidad.

Para poder continuar con el modelado ha llegado el momento de gestionar la información correspondiente a los requisitos no funcionales.

Algunos de los atributos sobre los que se pueden apoyar los requisitos no funcionales son los siguientes:

- Rendimiento
- Disponibilidad
- Durabilidad
- Estabilidad

- Funcionalidad
- Accesibilidad
- Adaptabilidad
- Capacidad
- Integridad de datos
- Documentación
- Operabilidad
- Mantenibilidad
- Conformidad
- Auditabilidad
- Portabilidad
- Seguridad
- Elasticidad
- Legibilidad
- Extensibilidad
- Eficiencia
- Privacidad
- Explotabilidad
- Integrabilidad
- Escalabilidad
- Robustez
- Interoperabilidad
- Garantía
- Reusabilidad
- Compatibilidad

Ejemplos de requisitos no funcionales que pueden solicitar los clientes:

- El sistema deberá soportar un máximo de 1000 usuarios concurrentes sin que el tiempo de respuesta medio aumente más de un 10%.

- El sistema deberá funcionar en ordenadores personales con sistema operativo Linux y entorno gráfico KDE.

- El sistema deberá funcionar en un servidor AS/400 con la siguiente configuración: …

- El sistema deberá utilizar el protocolo TCP/IP para las comunicaciones con otros sistemas.

- La interfaz de usuario del sistema deberá ser consistente con los estándares definidos en IBM's Common User Access.

- La tasa de fallos del sistema no podrá ser superior a 2 fallos por semana.

- El sistema deberá desarrollarse con Oracle 7 como servidor y clientes Visual Basic 4.

- El sistema deberá funcionar en los sistemas operativos Windows 95, Windows 98 y Windows

NT 4.0, siendo además posible el acceso al sistema a través de Internet usando cualquier navegador compatible con HTML 3.0.

EJERCICIO 29: Extrae los requisitos no funcionales de este problema

- Quiero vender música a través de Internet.
- Los usuarios comprarán créditos para adquirir canciones.
- Los usuarios buscarán las canciones que deseen y las pagarán con créditos.
- Los usuarios tendrán algunos días para descargar en su ordenador las canciones que hayan adquirido.
- Quiero hacer ofertas generales (afectan a todos los usuarios) y particulares (afectan a usuarios concretos).

EJERCICIO 30: Extrae los requisitos no funcionales que ha de tener una la app del banco Santander.

EJERCICIO 31: Extrae los requisitos no funcionales que ha de tener el sistema software de una biblioteca.

Las plantillas utilizadas para documentar los requisitos no funcionales contendrán los siguientes elementos.

- **Identificador:** Cada requisito ha de identificarse por un código único.
  Los identificadores de los actores comienzan con **NRF**.
- **Nombre descriptivo:** Nombre descriptivo del requisito.
- **Versión**: Para tener un control de versiones. Indica el número y la fecha de la versión actual del requisito.
- **Autores**: Contiene el nombre y la organización de los requisitos de la versión actual del caso de uso.
- **Fuentes**: Las fuentes (normalmente clientes) que propusieron la necesidad del actor.
- **Objetivos asociados**: Ha de contener una lista con los objetivos asociados al requisito.
- **Requisitos asociados**: En este campo se indicarán otros requisitos que estarán asociados al requisito no funcional que se está describiendo.
- **Descripción**: Ha de concretar la capacidad que deberá presentar el sistema.
- **Importancia**: Similar a las plantillas anteriores.
- **Urgencia**: Similar a las plantillas anteriores.
- **Estado**: Similar a las plantillas anteriores.

- **Estabilidad**: Similar a las plantillas anteriores.

- **Comentarios:** Cualquier otra información que no encaje en los campos anteriores.

NFR-001	Alumno
Versión	1.0 (20/02/2020)
Autores	Marcos Unzueta Puente
Fuentes	Enunciado del problema
Objetivos asociados	OBJ-005 …
Requisitos asociados	UC-001 … UC-002 … UC-003 … UC-004 … UC-005 …
Descripción	El sistema deberá almacenar la información relevante a los coches.
Importancia	Importante
Urgencia	Alta
Estado	Completado
Estabilidad	Alta
Comentarios	Ninguno

EJERCICIO 32: Realiza las tablas de requisitos no funcionales correspondientes al enunciado 1.

EJERCICIO 33: Realiza las tablas de requisitos no funcionales correspondientes al enunciado 2.

9. MATRIZ DE RASTREABILIDAD

Se trata de una matriz que permite saber que requisitos están asociados a cada objetivo.

Seguimos complementando nuestra documentación de modelado software con esta matriz.

	OBJ-001	OBJ-002	OBJ-003	OBJ--004	OBJ 005	OBJ-006	OBJ-007
IRQ-001					X		
IRQ-002		X					
IRQ-…							X
UC-001					X		
UC-002					X		
UC-003					X		
UC-004					X		
UC-…					X		

EJERCICIO 34: Realiza la matriz de rastreabilidad correspondiente al enunciado 1.

EJERCICIO 35: Realiza las matriz de rastreabilidad correspondiente al enunciado 2.

10. DIAGRAMA DE CLASES

El diagrama de clases UML se utiliza para documentar la estructura estática del sistema mostrando sus clases, atributos, operaciones (o métodos) y relaciones.

Esta fase nos ayudará a expresar todos los requisitos que ya hemos identificado y definido anteriormente en forma de objetos y clases.

Los diagramas de clases os resultarán más familiares ya que guardan una gran relación con la programación en lenguajes orientados a objetos y las construcciones a utilizar tendrán su traducción directa en Java (por ejemplo).

Antes de meternos en vereda, vamos a aprender a desarrollar diagramas de clases.

10.1 CLASES

Se trata de la descripción que se asigna a conjunto de objetos similares.
Las clases contienen las características comunes a dichos objetos (métodos, atributos, restricciones, etc.)
Estos objetos descritos, ejercen un rol o roles equivalentes en el sistema.

Podemos decir, por tanto, que una clase tiene como objetivo describir las propiedades y el comportamiento de un conjunto de objetos.

Podemos pensar en una clase coche, cuyos atributos son número de puertas, caballos, marca, modelo, matrícula, etc. A partir de esta clase podemos extraer infinitos objetos, un objeto coche tendrá un valor

definido para cada uno de los atributos y corresponderá a una instancia de la clase (el objeto coche uno tendrá, por ejemplo, cinco puertas, cien caballos, marca Renault, etc.).

EJERCICIO 36: Pensad en una clase "Alumno", usada para la gestión de un centro. ¿Qué atributos tendrá?, escribe objetos.

Un objeto es una instancia de una clase.

Un objeto, realmente, se corresponderá con un siempre elemento, mientras que las clases definirán una familia de elementos similares.

El uso de clases en los lenguajes de programación presenta una serie de ventajas:

- Es el método más cómodo para describir un conjunto de objetos que tienen los mismas comportamientos y propiedades.

- Al contar con una definición de carácter general de todos los objetos podemos prescindir del almacenamiento del código que representa el comportamiento de cada objeto de manera individual.

- Facilita mucho la reutilización de código ya que se permite utilizar clases desde otras o transmitir atributos y/o operaciones por herencia.

SE CONSIDERA UN ERROR GRAVE EN LA PROGRAMACIÓN A OBJETOS TENER DOS COPIAS DE UN MISMO ARTEFACTO DE INGENIERÍA DEL SOFTWARE (CÓDIGO DUPLICADO, PARTE DE UN DIAGRAMA DUPLICADO, PARTE DE UN DOCUMENTO DUPLICADO…). A no ser que exista un motivo muy justificado ha de reutilizarse la información y referenciar a ella en vez de tener varias copias de esta.

10.1.1 ELEMENTOS DE LA CLASE

Los elementos principales para representar a una clase en el tipo de diagrama que estamos tratando serán los siguientes:

- **Nombre**: Cuentan con un nombre **único** dentro de su contenedor (generalmente será un paquete, aunque también puede ser, por ejemplo, otra clase).

- **Atributos**: Son porciones de información que definen una propiedad de la clase.
  Cuando los atributos toman valores concretos describen el estado del objeto.

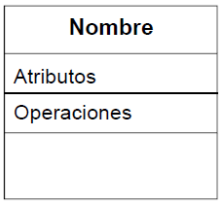
- **Operaciones o métodos**: Definen las acciones que pueden realizar los objetos que se instancien a partir de la clase. Se elaboran a partir de un conjunto de instrucciones.

EJERCICIO 37: Pensad en una clase "Tweet", usada en la aplicación "Twitter". ¿Qué atributos tendrá? ¿Qué operaciones estarán definidas?

EJERCICIO 38: Pensad en una clase "Trabajador", usada en una aplicación que controle la asistencia al trabajo (con horario de entrada y de salida). ¿Qué atributos tendrá? ¿Qué operaciones estarán definidas?

10.1.2 REPRESENTACIÓN

Las clases, en los diagramas de casos de uso de UML, se representan como un rectángulo, el cual, incluye a su vez una serie de compartimentos representados también como rectángulos más pequeños internos a la clase.



- **Compartimento del nombre**:
  Ha de contener al menos el nombre de la clase en negrita y centrado. Puede contener también estereotipos (ej.<<enumeration>>) y propiedades (ej. {persistent}).

  Las convenciones (las cuales debemos de seguir) dictan que los nombres de las clases han de ser sustantivos, las palabras han de concatenarse internamente y la primera letra de cada palabra interna ha de ser en mayúscula.

  Un ejemplo sería una clase llamada "*CocheDeSustitucion*".

- **Compartimento de los atributos**:
  En este apartado se dispone una lista de atributos correspondientes a la clase alineada a la izquierda. Los atributos han de tener, al menos, el nombre, con la posibilidad de añadir información adicional.

  Siguiendo las convenciones, vamos a escribir los nombres de los atributos utilizando el formato camelCase (la primera letra en minúscula, las palabras se concatenan y la primera letra de las siguientes palabras empiezan por mayúscula), se debe evitar que empiecen por caracteres como "_" o "$".

  El nombre elegido ha de ser mnemotécnico, es decir, diseñado para indicar al observador cual es la intención de su uso.

  Se deben evitar los nombres de variable de un carácter excepto para variables temporales.

  Un ejemplo sería un atributo llamado "*numeroDePuertas*".

  Sintaxis utilizada en la descripción de un atributo:
  visibilidad / nombre:tipo [multiplicidad] = valorPorDefecto {cadena de propiedades}

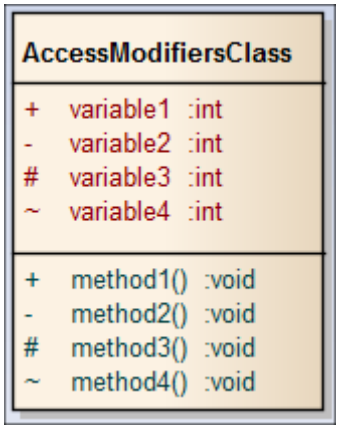
  - **Visibilidad:**
    Expresa si los atributos son visibles en otras clases.

+ Visibilidad Pública

Visibilidad Protegida

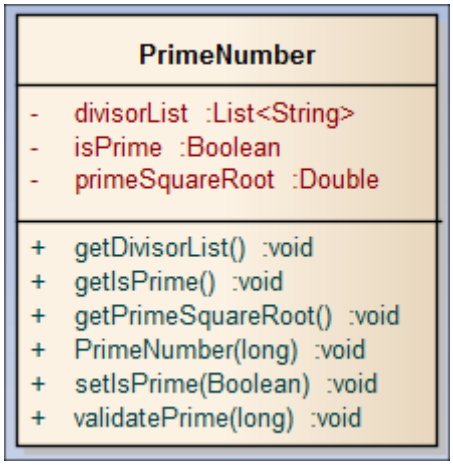
- Visibilidad Privada

~ Visibilidad de Paquete



- **/:**
  Indica si el atributo es derivado. Un atributo derivado será aquel cuyo valor pueda deducirse o derivarse de los valores de otros atributos o entidades... Por ejemplo, el atributo edad podría derivarse a partir del atributo fecha de nacimiento.

- **Tipo:**
  Pueden ser tipos primitivos o específicos de un lenguaje de programación concreto, pudiéndose utilizar cualquier tipo, incluso otras clases.



- **Multiplicidad:**
  Representa el número de valores que puede admitir el atributo. Se especifica entre corchetes usando la notación [M..N] siendo M el valor mínimo y N el valor máximo. Si el número posible de valores es único podrá representarse mediante un único valor. Algunos ejemplos:

  - **1**: El atributo tiene un solo valor. Si no se especifica la multiplicidad esta será la que aplique ya que es el valor por defecto.

- **0..1**: El atributo también puede tener el valor “null”.
- ***:** El atributo representa una colección de valores que puede no contener ningún valor.
- **1..***: El atributo representa una colección de valores que puede ha de contener al menos un valor.
- **N..M**: El atributo representa una colección de valores que puede ha de contener entre N y M valores.

- **ValorPorDefecto:**
  Valor que se le dará en este atributo a cada objeto de la clase cuando se instancie.
  Este valor se podrá modificar posteriormente.
  Si no se da este valor se omite el signo igual.

- **Cadena de propiedades:**
  Lista, separada por comas, de las propiedades de un atributo (readOnly, ordered, sequence…)

- **Compartimento de las operaciones**:
  En este apartado se dispone una lista de las operaciones correspondientes a la clase alineada a la izquierda.

  Las operaciones han de tener, al menos, el nombre, con la posibilidad de añadir información adicional.

  Las convenciones dictan que los nombres de las operaciones han de utilizar el formato camelCase (la primera letra en minúscula, las palabras se concatenan y la primera letra de las siguientes palabras empiezan por mayúscula), se debe evitar que las variables empiecen por caracteres como “_” o “$”.

  Los métodos han de ser verbos.

  Un ejemplo sería una clase llamada “*subirUnaMarcha*”.

  Sintaxis utilizada en la descripción de una operación:
  visibilidad nombre (listaParametros): tipoRetorno {cadena de propiedades}

  - **Visibilidad**:
    Igual que para los atributos.

  - **ListaParametros**:
    Lista separada por comas de los parámetros formales. La sintaxis propia de los parámetros se especificará posteriormente.

  - **TipoRetorno**:

Pueden ser tipos primitivos o específicos de un lenguaje de programación concreto, pudiéndose utilizar cualquier tipo, incluso otras clases. Representa el tipo que corresponderá con el valor que devuelva la función en su return.

- **Cadena de propiedades:**
  Lista separada por comas de las propiedades o restricciones de una operación (isQuery, isPolymorphic…).

Sintaxis utilizada en los parámetros de una operación:
dirección nombre: tipo [multiplicidad] = valorPorDefecto {cadena de propiedades}

- **Direccion**:
  Indica si el parámetro se envía dentro o fuera de la operación (in, out, inout).

- **Tipo:**
  Igual que en los atributos.

- **Multiplicidad:**
  Igual que en los atributos.

- **ValorPorDefecto:**
  Igual que en los atributos.

- **Cadena de propiedades**:
  Igual que en los atributos.

- **Compartimentos adicionales**: Para mostrar propiedades definidas por el usuario.

EJERCICIO 39: Representad la clase “Usuario” con los siguientes atributos:

- Nombre (Obligatorio)
- Primer apellido (Obligatorio)
- Segundo apellido (No obligatorio)
- Contraseña (Obligatorio, se maneja encriptada)
- Fecha de nacimiento (Obligatorio)
- Edad (Se calcula a partir de la fecha de nacimiento)
- Avatar (será un número entero)

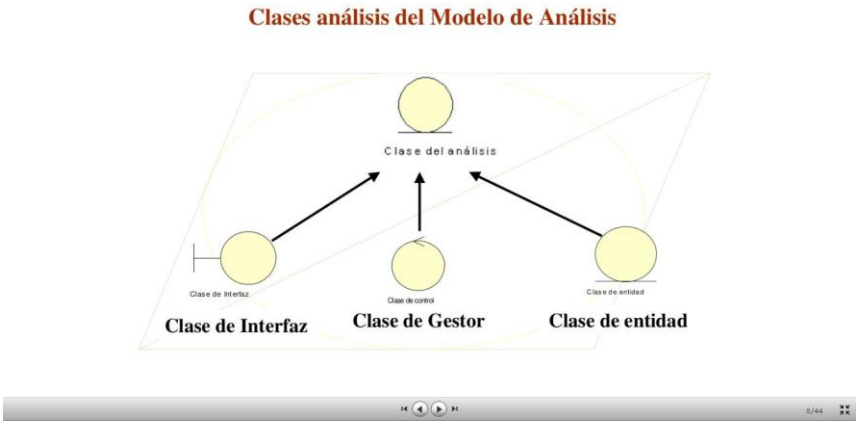
Las operaciones que tendrá serán las siguientes:

- Podrá modificar el avatar, se le pasará como parámetro el nuevo valor de su avatar y lo sustituirá.
- Podrá modificar su segundo apellido, se le pasará como parámetro el nuevo valor de su segundo apellido y lo sustituirá.

EJERCICIO 40: Representad la clase “Mario” de un videojuego de plataformas, asignad los atributos y operaciones que creáis convenientes.

10.1.3 INTERFAZ, CONTROL Y ENTIDAD:

Realmente, si seguimos el patrón Modelo-Vista-Controlador (MVC), cada clase de análisis se puede subdividir en tres.

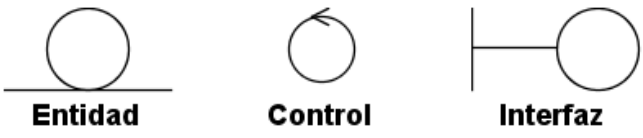


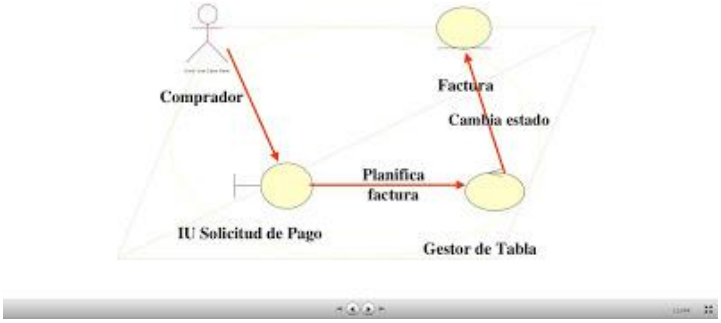
Es decir, todas las clases tienen una vista (interfaz), un modelo (entidad) y un controlador (gestor).

- **Interfaz**: Se encarga de presentar y transmitir la información entre el sistema y los actores.
  Las clases de interfaz modelan las partes del sistema que dependen de sus actores.
  Las clases interfaz, generalmente, representan abstracciones de ventanas, formularios, paneles, interfaces de comunicación, etc.

- **Entidad**: Contiene los objetos de negocio que se almacenan en el sistema.

- **Controlador o gestor**: Se encargan del procesamiento de la información y ponen en contacto la interfaz con la entidad.





EJERCICIO 41: Representa en UML las clases que puedes extraer del siguiente enunciado

(aplicando el patrón MVC)

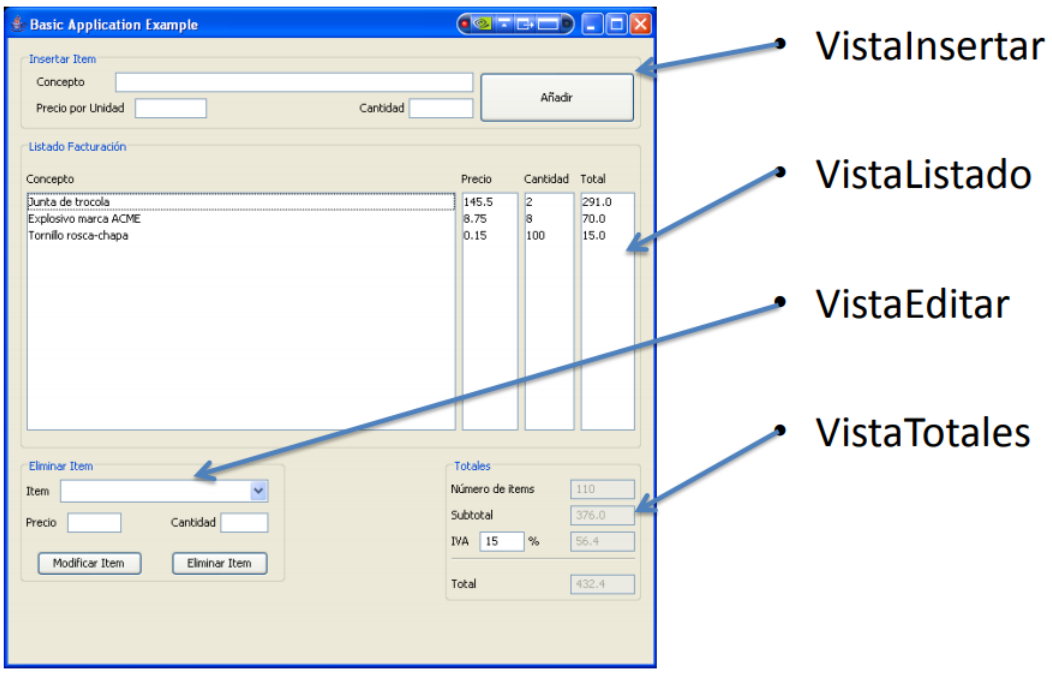
- Se pretende diseñar una calculadora que convierta valores de pesetas a euros y viceversa.

EJERCICIO 42: Representa en UML las clases que puedes extraer del siguiente enunciado (aplicando el patrón MVC).

- Se pretende diseñar una aplicación que dé soporte a la gestión de facturas.
- Cada Factura estará compuesta por Items
- Cada Item tendrá concepto, precio, cantidad
- La Factura además tendrá un porcentaje de IVA aplicable
- Valores como totales, subtotales, importe de IVA y número de items será datos calculados.

- Dos clases del modelo:
  - ModeloItem: Cada una de las líneas de la factura.
  - ModeloFactura: Contenedor de uno o varios Items.

- Cuatro clases Vista:



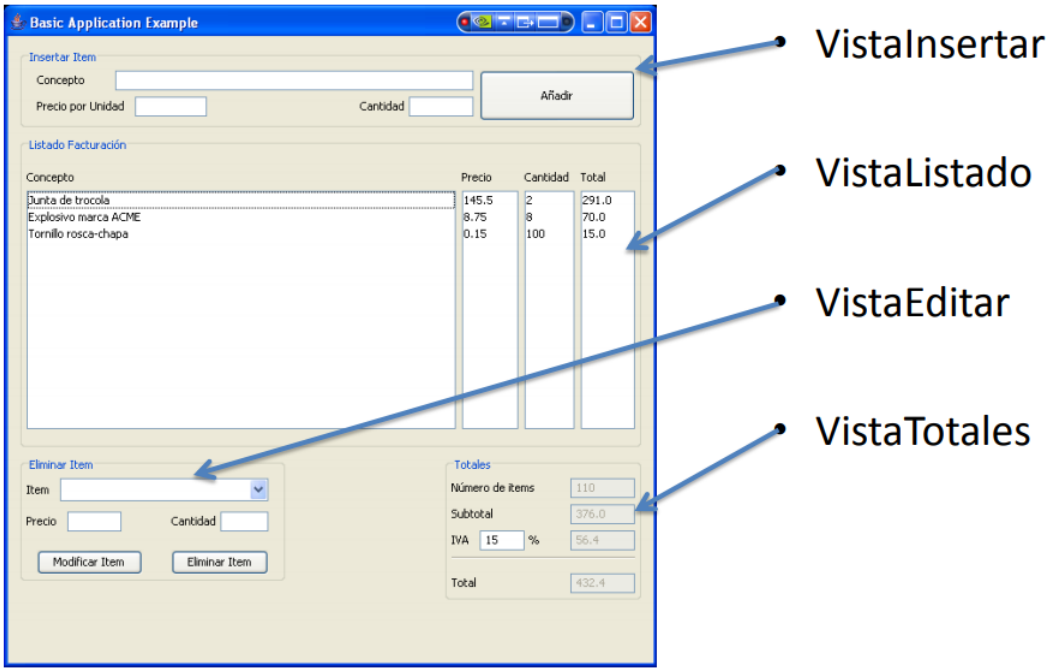
- Cuatro clases Controlador:
  - Una por Vista.

EJERCICIO 43: Representa en UML las clases que puedes extraer del siguiente enunciado (aplicando el patrón MVC).

- Se pretende diseñar una aplicación que dé soporte a la gestión de facturas.
- Cada Factura estará compuesta por Items
- Cada Item tendrá concepto, precio, cantidad
- La Factura además tendrá un porcentaje de IVA aplicable
- Valores como totales, subtotales, importe de IVA y número de items será datos calculados.

- Dos clases del modelo:
  - ModeloItem: Cada una de las líneas de la factura.
  - ModeloFactura: Contenedor de uno o varios Items.

- Cuatro clases Vista:



- Cuatro clases Controlador:
  - Una por Vista.

EJERCICIO 44: Representa en UML las clases que puedes extraer para un proceso mediante el cual un usuario introduce una contraseña, el sistema la encripta, la compara con su versión encriptada en la base de datos y actúa en consecuencia.

10.2 VISIBLILIDAD

Define desde qué contendores se puede acceder al elemento (atributo, método o clase) que va

asociado

10.2.1 Tipos de visibilidad:

Público: Se puede acceder al elemento desde cualquier clase.

Protegido: Sólo se permite acceder al elemento desde operaciones de la propia clase, de clases derivadas o de clases del mismo paquete.

De paquete: Sólo se puede acceder al elemento desde operaciones de la propia clase y clases del mismo paquete.

Privado: Sólo se puede acceder al elemento desde operaciones de la clase.

		Mismo paquete		Otro paquete	
		Subclase	Otra	Subclase	Otra
-	**private**	*no*	*no*	*no*	*no*
#	**protected**	*sí*	*sí*	*sí*	*no*
+	**public**	*sí*	*sí*	*sí*	*sí*
~	*package*	*sí*	*sí*	*no*	*no*

10.2.2 Representación:

La visibilidad, en los diagramas de casos de uso de UML, se representan añadiendo un carácter al elemento del que se quiere indicar.

+ Visibilidad Pública
# Visibilidad Protegida
- Visibilidad Privada
~ Visibilidad de Paquete

10.2.3 Características a tener en cuenta:

En los atributos y operaciones no existe una visibilidad por defecto, por tanto, si la visibilidad no se representa en el diagrama querrá decir que no está definida.

Como norma general, a la hora de definir la visibilidad, tendremos en cuenta que hay que ser lo más ocultistas posible, haciendo que el elemento sea visible a la menor cantidad de clases posible, pero evitando penalizar en rendimiento o funcionalidad. Algunas consecuencias directas que tendrá esta premisa serán las siguientes:

- Los atributos deben ser privados. Se deben modificar mediante métodos de la clase creados a tal efecto.
- Las operaciones que ayudan a implementar parte de la funcionalidad deben ser privadas (si no se utilizan desde clases derivadas) o protegidas (si se utilizan desde clases derivadas).

10.3 TIPO DE DATO

Los tipos de datos pueden ser primitivos o específicos de un lenguaje de programación concreto, pudiéndose

utilizar cualquier tipo, incluso otras clases.
Algunos de los tipos de datos más comunes y usados son los siguientes:

- **Integer**: Representan los números enteros (negativos, positivos y cero).
- **Boolean**: Es un tipo de dato de carácter lógico, acepta los valores verdadero y falso.
- **String**: Este tipo de dato representa una cadena de caracteres.
- **UnlimitedNatural**: Son enteros igual o mayores a cero, el valor de infinito se denota con un asterisco: *.
- **Float**: Representan los números decimales.
- **Date**: Sirve para almacenar fechas.
- **Char**: Almacena un carácter.
- **Listas o arrays**: Serie de elementos del tipo vector, matriz o más dimensiones.

10.4 OBJETOS.

"Un objeto es algo que tiene comportamiento, estado e identidad" (Grady Booch)

Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos:
Primero, los objetos se crean a medida que se necesitan.
Segundo. Los mensajes se mueven de un objeto a otro (o del usuario a un objeto) a medida que el programa procesa información o responde a la entrada del usuario.
Tercero, cuando los objetos ya no se necesitan, se borran y se libera la memoria.

Una ***clase*** es una ***abstracción*** que define las características comunes de un conjunto de ***objetos*** relevantes para el sistema.

Cada vez que se construye un ***objeto*** en un programa informático a partir de una clase se crea lo que se conoce como ***instancia*** de esa ***clase***. Cada instancia en el sistema sirve como modelo de un objeto del contexto del problema relevante para su solución, que puede realizar un trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema, sin revelar cómo se implementan estas características.

La ***encapsulación*** y el ocultamiento aseguran que los datos de un objeto están ocultos, con lo que no se pueden modificar accidentalmente por funciones externas al objeto.

Existe un caso particular de clase, llamada ***clase abstracta***, que, por sus características, no puede ser instanciada. Se suelen usar para definir métodos genéricos relacionados con el sistema que no serán traducidos a objetos concretos, o para definir métodos de base para clases derivadas.

10.4.1 INSTANCIACIÓN

Objeto es la ***instancia*** de una ***clase***. A este proceso se le llama ***instanciación***.

Enlace es la instancia de una ***relación***.

El proceso de creación de un nuevo objeto que pertenece a una clase se llama instanciación de la clase, y al objeto resultante se le llama instancia, por esto es por lo que las variables cuyos valores pertenecen al objeto y pueden variar a lo largo de su tiempo de vida se denominan variables instancia.

La creación de un objeto consiste en la creación de un nuevo elemento con su propio estado y su propia identidad, que se comportará de forma consistente con todos los demás objetos de la clase.

Una clase puede verse como una factoría de objetos.

10.4.2 CARACTERÍSTICAS DE LOS OBJETOS: ESTADO, COMPORTAMIENTO E IDENTIDAD

Un objeto encapsula su propio estado y comportamiento

Un objeto se define por:

Identidad: Aquello que distingue a un objeto entre todos los de su clase.

Su estado: Es la concreción de los atributos definidos en la clase a un valor concreto.

Su comportamiento: Definido por los métodos públicos de su clase.

Su tiempo de vida: Intervalo de tiempo a lo largo del programa en el que el objeto existe. Comienza con su creación a través del mecanismo de instanciación y finaliza cuando el objeto se destruye.

Conceptualmente, un ***objeto*** es una cosa con la que se puede interactuar. Cómo se comporte depende del estado interno actual del objeto. Importa con que objeto se interactúa, normalmente nos dirigimos al objeto por su nombre; es decir un objeto tiene una identidad que lo distingue del resto de objetos.

El ***estado*** de un objeto lo constituyen todos los datos que encapsula en un momento determinado. Un objeto normalmente tiene un número de ***atributos*** cada uno de los cuales tiene un valor.
El conjunto de atributos de un objeto no puede cambiar durante la vida del objeto, aunque los valores de los atributos sí pueden hacerlo.

10.5 RELACIONES O ASOCIACIONES

Una relación es una conexión semántica entre elementos de un modelo.
La instanciación de una relación o asociación es un enlace

Tipos de relaciones:

- **Asociación**: Es una relación estructural que describe una conexión entre objetos.

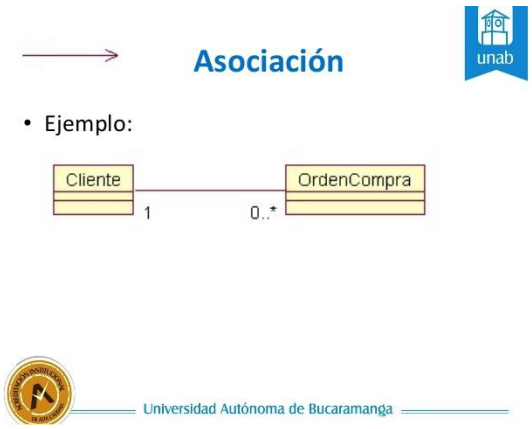
- **Generalización**: Relación tipo herencia, que establecen entre ellas las superclases y las subclases.

- **Dependencia**: Relación entre un cliente y el proveedor del servicio usado por el cliente.

10.5.1 ASOCIACIÓN

Las asociaciones son relaciones estructurales que llevan la información sobre conexiones entre objetos en un sistema.

- **ELEMENTOS**



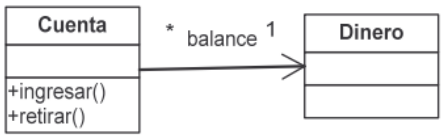
Las asociaciones suelen ser bidireccionales (que aplica en ambos sentidos).

Se suelen representar mediante una línea horizontal que conecta dos objetos, no obstante, en función de una serie de características esto puede cambiar (lo veremos más adelante).

La información que se proporciona en las asociaciones se puede (y en muchas ocasiones se debe) complementar. Para este fin se utilizan una serie de elementos complementarios.

En el centro de la línea de asociación se puede escribir el **nombre** de la relación, al cual se puede añadir un triángulo sólido que indicará el **orden de lectura**.

Hay ocasiones en las que la navegación, en lugar de ser bidireccional, transcurre en un solo sentido. Para representar esto, en la línea, se representará una flecha que indicará el sentido de la asociación y que se denomina **indicador de navegación**. Una x indica que el extremo en el que se sitúa no es navegable.



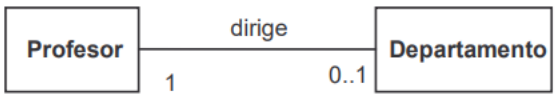

```
class Cuenta
{
  private Dinero balance;

  public void ingresar (Dinero cantidad)
  {
    balance += cantidad;
  }

  public void retirar (Dinero cantidad)
  {
    balance -= cantidad;
  }

  public Dinero getSaldo ()
  {
    return balance;
  }
}
```

La **multiplicidad** de las asociaciones indica cuántos objetos de cada tipo intervienen en la relación.
Cada asociación tiene dos multiplicidades (una para cada extremo de la relación) y dentro de cada una de ellas hay que indicar la multiplicidad mínima y la multiplicidad máxima.

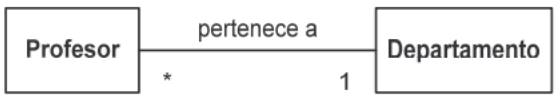
Multiplicidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

Cuando la multiplicidad mínima es 0 se dice que la relación es opcional, es decir, puede ser o no ser.
En cambio, cuando la multiplicidad mínima mayor o igual a 1, se dice que la relación es obligatoria.

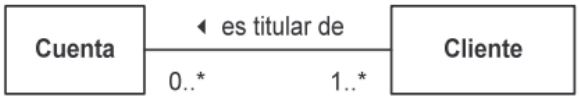
Todo departamento tiene un director. Un profesor puede dirigir un departamento:



Todo profesor pertenece a un departamento. A un departamento pueden pertenecer varios profesores:

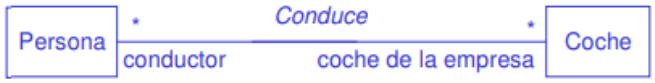


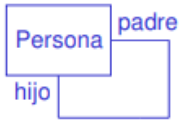
Un cliente puede o no ser titular de una o varias cuentas (Relación opcional). Una cuenta ha de tener un titular como mínimo (Relación opcional),



Otros elementos usados en la asociación, pero no tan populares son los siguientes:

- **Papel/rol**: Consiste en especificar el papel o rol que desempeña cada uno de los extremos (clases) en una asociación A cada rol se le asigna un nombre que permite identificar cada extremo de manera inequívoca.
  El nombre del papel ha de guardar relación con el conjunto de los objetos relacionados.
  Los nombres de papeles solo serán obligatorios en asociaciones entre dos objetos de la misma clase y para distinguir dos asociaciones entre el mismo par de clases, en el resto de los casos serán opcionales.



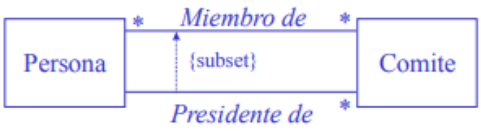


- **Visibilidad:** Igual que en las clases.

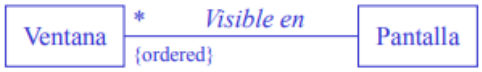
- **Calificador**: El calificador indica el atributo o conjunto de atributos que permitirán discernir qué subconjunto de objetos de cada extremo participará en la asociación.
  Para representar un calificador se dibuja un recuadro en el extremo de la clase que califica en la asociación.

- **Cadena de propiedades**: Algunas de las propiedades aplicables a las relaciones de asociación son las siguientes.

  - **{subset}**: Se utiliza para indicar que la asociación es un subconjunto de otra.



  - **{ordered}:** El extremo representa un conjunto ordenado.

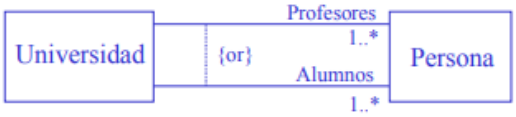


  - **{sequence} o {seq}:** El extremo representa una secuencia.

  - **{or}:** Se utiliza para indicar que los objetos de una clase solo pueden participar en una de las asociaciones pertenecientes a un grupo de estas.

EJERCICIO 45: Explica detalladamente los siguientes diagramas.

a)

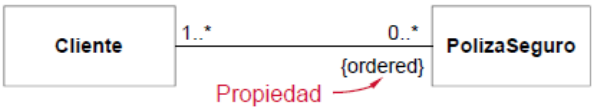


b)

c)

d)

e)



EJERCICIO 46: Realizad un diagrama de clases en el cual exista una clase profesor, una clase departamento, una clase instituto y una clase colegio.

Un profesor puede impartir clase en un instituto o en un colegio.

Todo profesor puede pertenecer a uno o más departamentos, y dentro de los departamentos a los que pertenece puede ser jefe de uno.

- **TIPOS DE ASOCIACIONES**

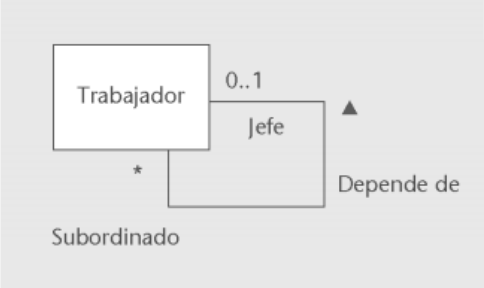
  - **Binarias**: Son aquellas que comunican dos clases.

En el ejemplo mostrado anteriormente se indica que una persona puede trabajar en una empresa o ninguna y que las personas que trabajan en empresas se llaman empleados. Las empresas han de contar con, al menos, un empleado.

La flecha abierta indica que solo se puede acceder (navegar) desde una empresa hacia sus empleados.

- **Reflexiva**:

  Son asociaciones binarias en las cuales, las dos clases que se comunican son la misma. Generalmente esta relación comunica a subgrupos de la clase en la que se encuentra.



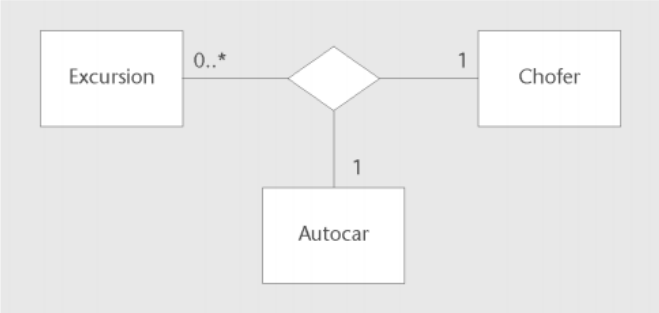
En el ejemplo anterior cada trabajador depende de un jefe. Tanto el jefe como el subordinado son trabajadores.

Cada trabajador puede tener como máximo un jefe, mientras que un jefe puede cualquier número de subordinados.

EJERCICIO 47: Crea otros tres diagramas con ejemplos de relaciones reflexivas.

- **Ternaria**: Representa aquellas relaciones que cuentan con tres roles implicados, al igual que una relación n-aria será aquella que tenga n roles.
-

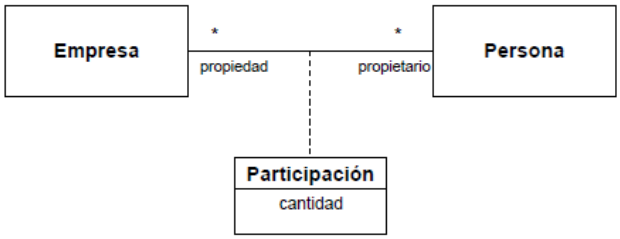
En este tipo de asociaciones, en el punto de confluencia se dibujará un rombo.

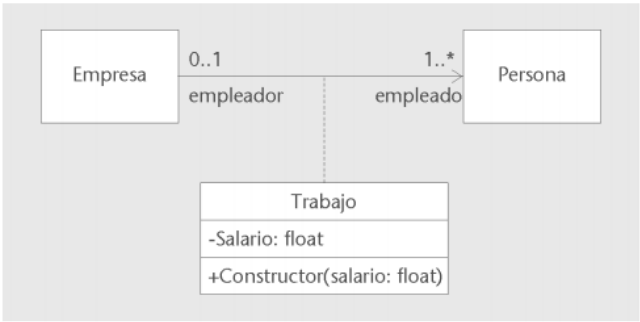


Un chofer determinado puede conducir un autocar determinado en cualquier número de excursiones, pero en una excursión concreta un chófer solo puede conducir un autocar y un autocar solo puede tener un chofer,

EJERCICIO 48: Cada aula alberga uno o más grupos a los que se imparten una o más asignaturas, a su vez cada grupo tiene asignada una o más aulas en donde recibe docencia de una o más asignaturas, y además cada asignatura se imparte en una o más aulas a uno o más grupos.

- **Clases asociación:** Se trata de asociaciones que a su vez tienen comportamiento de clases. Cobran utilidad cuando cada enlace (instancia de la relación) ha de tener sus propios valores, ya sea para los atributos, operaciones propias o sus propias referencias a objetos.
  Se representa de la misma manera que una clase que se une mediante una línea discontinua a la asociación.





EJERCICIO 49: Planteamos un problema en el que existen dos clases, muro y ventana. Cada ventana ha de estar posicionada en un muro y los muros pueden tener varias ventanas. Las ventanas se posicionan con los valores “X” e “Y” del eje cartesiano.

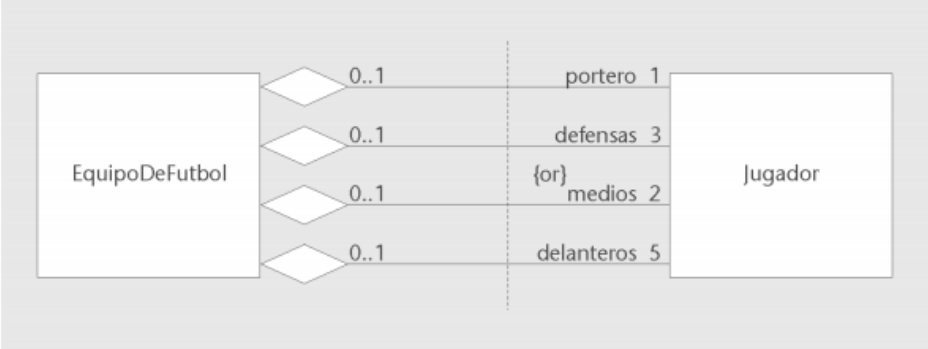
Realiza el diagrama de clases.

- **Agregación:** Representan un caso particular de la asociación binaria en la cual uno de los roles asume el significado de parte y otro el de todo.
  Algunos de los casos en los cuales aplica este concepto son los siguientes:

  - **Acoplamiento-piezas**: Una máquina y sus piezas, un circuito y sus componentes, un sistema software y sus programas, etc. Cada parte tiene un papel concreto y no se puede sustituir.

  - **Continente-contenido**: En el caso anterior las piezas son las que conforman el acoplamiento, en este caso el continente no conforma el contenido.
    Un avión tiene una relación continente contenido con sus pasajeros ya que, aunque no transporte ningún pasajero, el avión sigue siendo un avión.

  - **Colectivo-miembros**: Algunos ejemplos serían un grupo de excursionistas y sus excursionistas, o bien, una promoción de alumnos y dichos alumnos.
    En estos casos los miembros no tienen papeles diferenciados y, por tanto, son intercambiables.

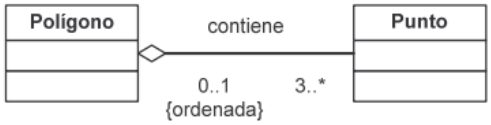


En el ejemplo mostrado anteriormente se representa una composición un poco pintoresca de un equipo de fútbol.

Cada jugador juega en un equipo o ninguno, y un jugador solo puede estar en una de las cuatro posiciones (como indica {or}).

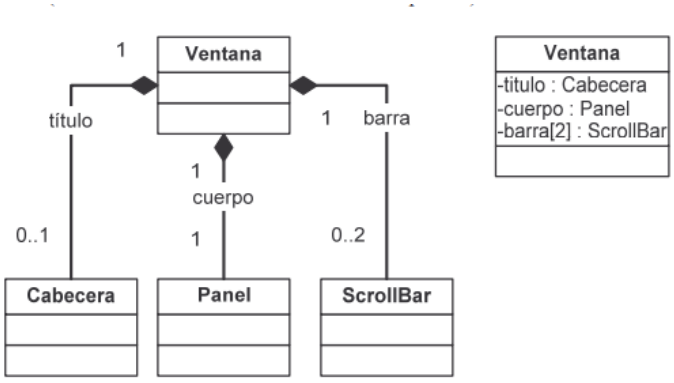
Las posiciones a considerar son portero, defensas, medios o delanteros (no se profundiza en posiciones internas a estos grupos).

Como se puede ver la agregación se representa como un rombo vacío en la parte u objeto compuesto.

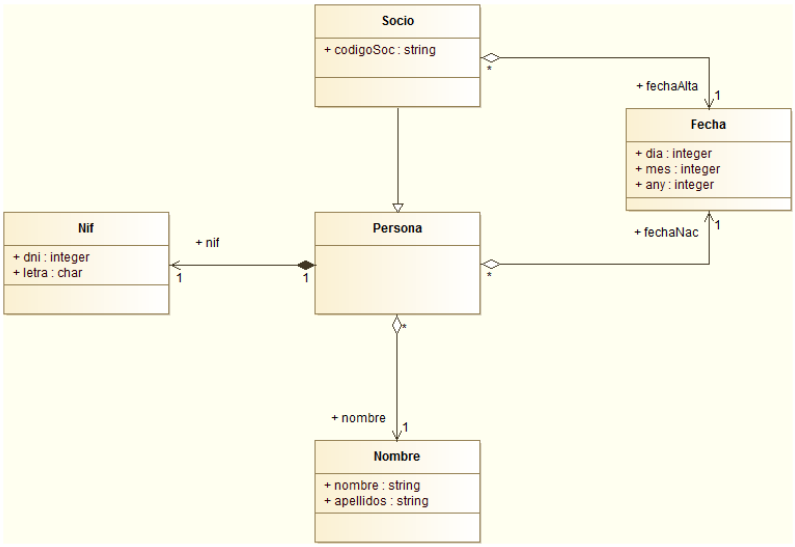


EJERCICIO 50: Planteamos un problema en el que tenemos 4 clases (Coche, llanta, puerta, motor).
Se necesita saber la marca del coche, el año de fabricación y su modelo.
De la llanta se necesita saber la marca, el modelo y el radio en pulgadas.
Del motor se necesita el número de serie y los cilindros.
De la puerta se necesita saber el tipo.
Las clase han de tener sus constructores y sus getters y setters correspondientes.
Cada automóvil ha de tener 4 llantas, 2 o más puertas y un motor.
Cada pieza ha de pertenecer a un automóvil.
Realiza el diagrama de clases.
¿Qué tipo y subtipo de relación es?

- **Composición:**
  Es una asociación agregación con las restricciones adicionales de que un objeto sólo puede ser parte de un compuesto a la vez y que el objeto compuesto es el único responsable de la disponibilidad de todas sus partes.
  Los objetos componentes no tienen vida propia, sino que, cuando se destruye el objeto compuesto del que forman parte también se destruyen.
  Se representan con un rombo en uno de los extremos de la asociación, pero en este caso el rombo está coloreado.



EJERCICIO 51: Explica el sistema que representa el siguiente diagrama.



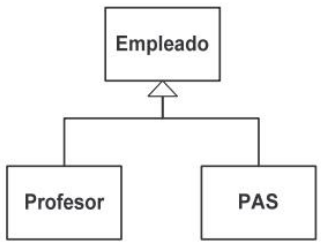
10.5.2 GENERALIZACIÓN

Consiste en una relación entre un elemento general (superclase) y otro más específico (subclase).

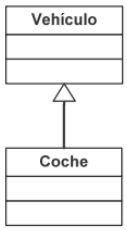
En este contexto, el concepto de generalización tiene su traducción directa con el de herencia.

Cuando un conjunto de subclases tiene en común una serie de atributos y operaciones se crea el concepto de superclases, que contienen todos estos elementos en conjunto y que posteriormente permiten desarrollar las características específicas. Es decir, cada subclase hereda las características de su superclase, aunque también añade su propios atributos y operaciones específicos.

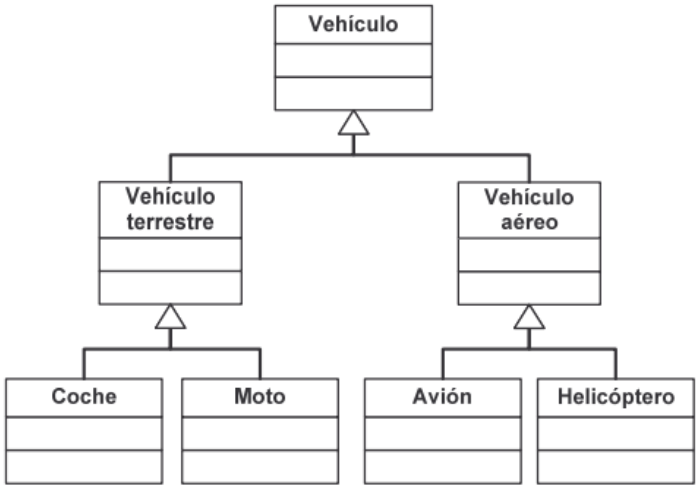
La notación usada en UML para la generalización consiste en una flecha que conecta las subclases con la superclase, yendo la flecha de las subclases a la superclase.



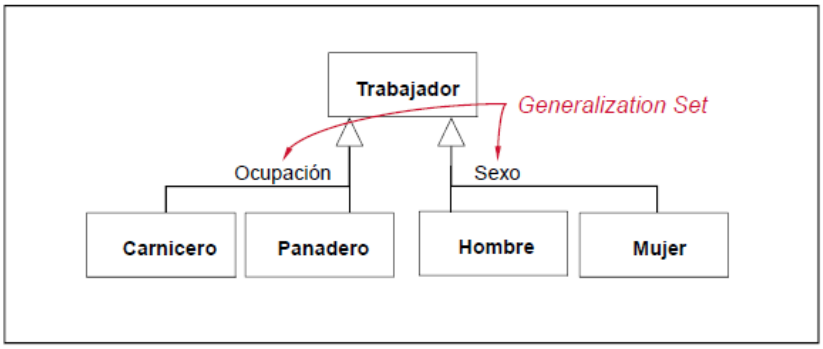

```
public class Empleado
{
   ...
}

public class Profesor extends Empleado
{
   ...
}

public class PAS extends Empleado
{
   ...
}
```



De este último ejemplo se puede extraer la conclusión de que todos los coches son vehículo, y de que algunos vehículos son coches (no todos).



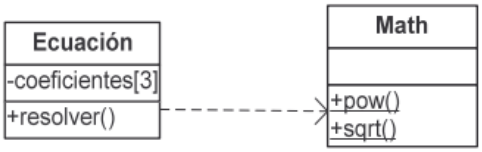
Para diferenciar las relaciones de generalización, en ocasiones, se utilizan conjuntos de generalización.



10.5.3 DEPENDENCIA

La relación de dependencia es la que se genera entre un cliente, que consume un servicio, y el proveedor de este servicio usado por el cliente.

En UML se representa como una línea discontinua con una punta de flecha abierta, que apunta del cliente al proveedor.



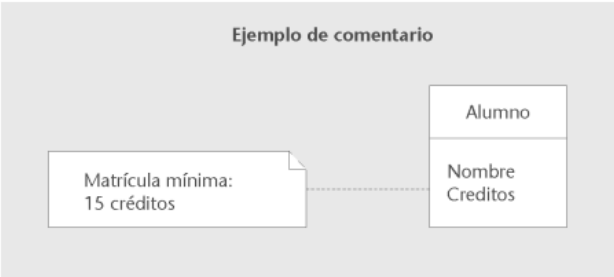
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

En este ejemplo se muestra la resolución de la ecuación de segundo grado mostrada.

10.6COMENTARIOS

Un comentario se trata de una construcción del lenguaje cuyo objetivo es incluir anotaciones legibles sin que estas afecten al propio funcionamiento.

Un comentario, en los diagramas de clases de UML, se introduce dentro de un rectángulo con un vértice doblado, enlazado mediante una línea discontinua al elemento que se refiere.



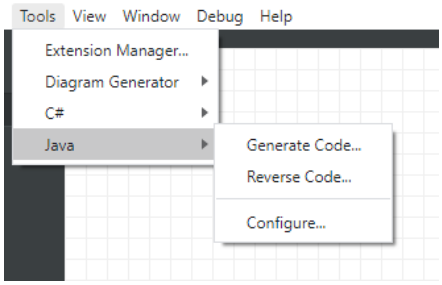
10.7 GENERACIÓN DE CÓDIGO

Es posible generar código en ciertos lenguajes de programación a partir de diagramas.

En este caso veremos la generación de código a partir de los diagramas de clases.

Esta funcionalidad está completamente implementada en Visual Paradigm, pero se corresponde con la parte de pago (no se puede utilizar en la versión de VP que estamos usando), por eso vamos a usar StarUML.

En StarUML hay que instalar las extensiones del lenguaje deseado (C++, Java …) y reiniciar el entorno. Una vez las extensiones estén instaladas aparecerán las opciones correspondientes.



10.8Ejercicios

EJERCICIO 52: Realiza el diagrama de clases correspondiente a Twitter.

EJERCICIO 53: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- El diagrama va a representar la estructura de un robot.
- El robot estará compuesto por varios módulos entre los que se encuentran: rotación, extensión, helicoidal, cámara. Los módulos podrán ser dinámicos (capaces de moverse: rotación, extensión, helicoidal) o estáticos (no se pueden mover: cámara).
- Los módulos tendrán un identificador (1-255) y unas dimensiones (largo, ancho y alto, entre 1 y 200mm).
- Los módulos estarán compuestos de un sistema de control y un sistema de comunicación. Los módulos dinámicos tendrán:
    - Motores (1 o 2).
    - Un parámetro que es el tipo de movimiento que pueden realizar.
    - Una función que es moverse (con parámetro el tipo de movimiento).
- Los módulos estáticos podrán tener sensores (de 0 a 5).
- El sistema de control utiliza el sistema de mensajes para comunicarse.
- Los módulos pueden enviar y recibir mensajes de/hacia el usuario y otros módulos, con un parámetro que es un array de datos a mandar o recibir.
- El sistema de control también utiliza los motores para moverse y los sensores para captar información del medio.

- Se pide utilizar la herencia siempre que se pueda.

EJERCICIO 54: Realiza los diagramas de clases y de casos de uso correspondientes al siguiente enunciado:

- Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (ingeniería, literatura, informática, historia ...), editorial, año y autor.
- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, con retraso o en reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, y por cada día de retraso, se impone una "multa" de dos días sin posibilidad de coger un nuevo libro.
- A la hora de realizar los diagramas, ten en cuenta añadir los métodos necesarios para realizar el préstamo y devolución de libros.

EJERCICIO 55: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- Se propone realizar un modelo simplificado de los distintos miembros de la comunidad universitaria.
- Todos los miembros de la comunidad universitaria se caracterizan por un nombre y un D.N.I.
- Los miembros se dividen en estudiantes o personal de la universidad.
- Todos los estudiantes tienen un número de identificación asociado: el nie.
- En cuanto al personal, todos tienen un salario asignado y a su vez estos pueden ser personal docente investigador (pdi) o personal de administración y servicios (pas).
- Los pdi tienen asignada una asignatura que impartir (se identificará por el título) y los pas un edificio donde trabajan (se identificará por el nombre del edificio).
- Además de los anteriores, existen los doctorandos que son a la vez pdi y estudiantes.
- Los doctorandos se caracterizan por el título de la tesis doctoral sobre la que investigan.

EJERCICIO 56: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- En un juego de ordenador existen 2 tipos de jugadores: los principiantes y los avanzados.
- Todos ellos deben tener un nombre y un número de vidas.
- Los principiantes se desplazan andando a unas coordenadas (x,y).
- Los jugadores avanzados además de andar también pueden conducir un vehículo para desplazarse más rápido a unas coordenadas.
- Cada vehículo tiene asociada una velocidad que puede ser leída y ajustada a un valor dado, pero no puede superar una velocidad máxima dada.
- La velocidad máxima sólo se podrá asignar una vez y no podrá ser modificada.
- Todos los atributos de las clases serán privados y tendrán métodos públicos para acceder a ellos (get/set) salvo que los requisitos indiquen lo contrario.
- Debe existir un método que se llame andar y otro conducir.

EJERCICIO 57: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- En un programa de ordenador, las facturas tienen necesariamente un conjunto de datos del proveedor, un conjunto de datos del cliente, un importe (valor decimal) y una fecha.
- Los datos del cliente son la cadena de caracteres nombre y el entero fiabilidad de pago, mientras que los datos del proveedor son sólo su nombre.
- Dentro de la categoría cliente está el subtipo “cliente moroso”, que lleva también asociado el número decimal deuda.
- Cada clase tendrá los métodos para leer y fijar (“set” y “get”) todos sus atributos (no hace falta incluir en la clase Factura los métodos para fijar datos del cliente o del proveedor).
- También se debe incluir en la clase Factura un método de “Borrado”, que no devuelve ni recibe ningún parámetro.
- Los atributos serán privados y tendrán métodos públicos para acceder a ellos.

EJERCICIO 58: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- Se desea diseñar un diagrama de clases sobre la información de las reservas de una empresa

dedicada al alquiler de automóviles.

- Un determinado cliente puede tener en un momento dado hechas varias reservas.
- De cada cliente se desean almacenar su DNI, nombre, dirección y teléfono.
- Dos clientes se diferencian por un código único.
- Cada cliente puede ser avalado por otro cliente de la empresa.
- Una reserva la realiza un único cliente, pero puede involucrar varios coches.
- Es importante registrar la fecha de inicio y final de la reserva, el precio del alquiler de cada uno de los coches, los litros de gasolina en el depósito en el momento de realizar la reserva, el precio total de la reserva y un indicador de si el coche o los coches han sido entregados.
- Todo coche tiene siempre asignado un determinado garaje que no puede cambiar.
- En los garajes pueden aparcar varios coches, teniendo un valor que almacena la capacidad del mismo (número máximo de coches que pueden aparcar).
- De cada coche se requiere la matricula, el modelo el color y la marca.
- Cada reserva se realiza en una determinada agencia.

EJERCICIO 59: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- Un zoo necesita una aplicación informática para llevar su organización respecto a las especies que posee, los empleados (cuidadores y guías), y los distintos itinerarios de visita que ofrece.
- De las especies interesa saber el nombre en español, el nombre científico y una descripción general.
- Hay que tener en cuenta que una especie puede vivir en diferentes hábitats naturales y que un hábitat puede ser ocupado por diferentes especies.
- Las especies se encuentran en distintas zonas del parque de manera que cada especie está en una zona y en una zona hay varias especies.
- Los diferentes hábitats naturales vienen definidos por el nombre, el clima y el tipo de vegetación predominantes, así como el continente o continentes en los que se encuentran.

- Las zonas del parque en las que se encuentran las distintas especies vienen definidas por el nombre y la extensión que ocupan.

- Los itinerarios discurren por distintas zonas del parque.

- La información de interés para los itinerarios es: código de itinerario, la duración del recorrido, la longitud del itinerario, el máximo número de visitantes autorizado y el número de distintas especies que visita.

- Hay que tener en cuenta que un itinerario recorre distintas zonas del parque y que una zona puede ser recorrida por diferentes itinerarios.

- Los guías del parque vienen definidos por el nombre, dirección, teléfono y fecha en la que comenzaron a trabajar en el zoo.

- Interesa saber qué guías llevan qué itinerarios, teniendo en cuenta que un guía puede llevar varios itinerarios y que un itinerario puede ser asignado a diferentes guías en diferentes horas, siendo éstas un dato de interés.

- Los cuidadores vienen definidos por el nombre, dirección, teléfono y fecha de ingreso en el parque.

- Hay que tener en cuenta que un cuidador puede estar a cargo de varias especies y que una especie puede ser atendida por varios cuidadores, siendo de interés la fecha en la que un cuidador se hace cargo de una especie.

EJERCICIO 60: Realiza el diagrama de clases correspondiente al siguiente enunciado:

- La Policía quiere crear una base de datos sobre la seguridad en algunas entidades bancarias.

- Cada entidad bancaria se caracteriza por un código y por el domicilio de su Central.

- Cada entidad bancaria tiene más de una sucursal que también se caracteriza por un código y por el domicilio, así como por el número de empleados de dicha sucursal.

- Cada sucursal contrata, según el día, algunos vigilantes jurados, que se caracterizan por un Código y su edad.

- Un vigilante puede ser contratado por diferentes sucursales (incluso de diferentes entidades), en distintas fechas y es un dato de interés dicha fecha, así como si se ha contratado con arma o no.

- Por otra parte, se quiere controlar a las personas que han sido detenidas por atracar las sucursales de dichas entidades. Estas personas se definen por una clave (código) y su nombre

completo.

- Alguna de estas personas está integrada en algunas bandas organizadas y por ello se desea saber a qué banda pertenecen, sin ser de interés si la banda ha participado en el delito o no.
- Dichas bandas se definen por un número de banda.
- Así mismo, es interesante saber en qué fecha ha atracado cada persona una sucursal.
- Evidentemente, una persona puede atracar varias sucursales en diferentes fechas, así como que una sucursal puede ser atracada por varias personas.
- Igualmente, se quiere saber qué Juez ha estado encargado del caso, sabiendo que un individuo, por diferentes delitos, puede ser juzgado por diferentes jueces.
- Es de interés saber, en cada delito, si la persona detenida ha sido condenada o no y de haberlo sido, cuánto tiempo pasará en la cárcel.
- Un juez se caracteriza por una clave interna del juzgado, su nombre y los años de servicio.
- NOTA: En ningún caso interesa saber si un vigilante ha participado en la detención de un atracador.

EJERCICIO 61: Realiza el diagrama de clases correspondiente al enunciado 1.

EJERCICIO 62: Realiza el diagrama de clases correspondiente al enunciado 2.

11. DIAGRAMA DE SECUENCIA

Es un diagrama de interacción.

Una interacción consiste en el intercambio de información entre elementos.

Estos diagramas muestran la interacción entre los objetos poniendo especial énfasis en el intercambio de mensajes entre estos.

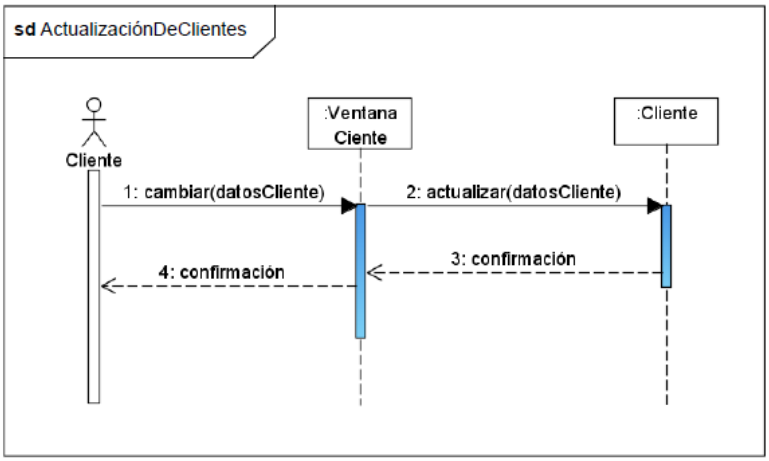
Una vez tenemos realizados todos los diagramas de casos de uso correspondientes se procederá a realizar los diagramas de secuencia de aquellos casos de uso que se consideran más complejos o comprometidos.

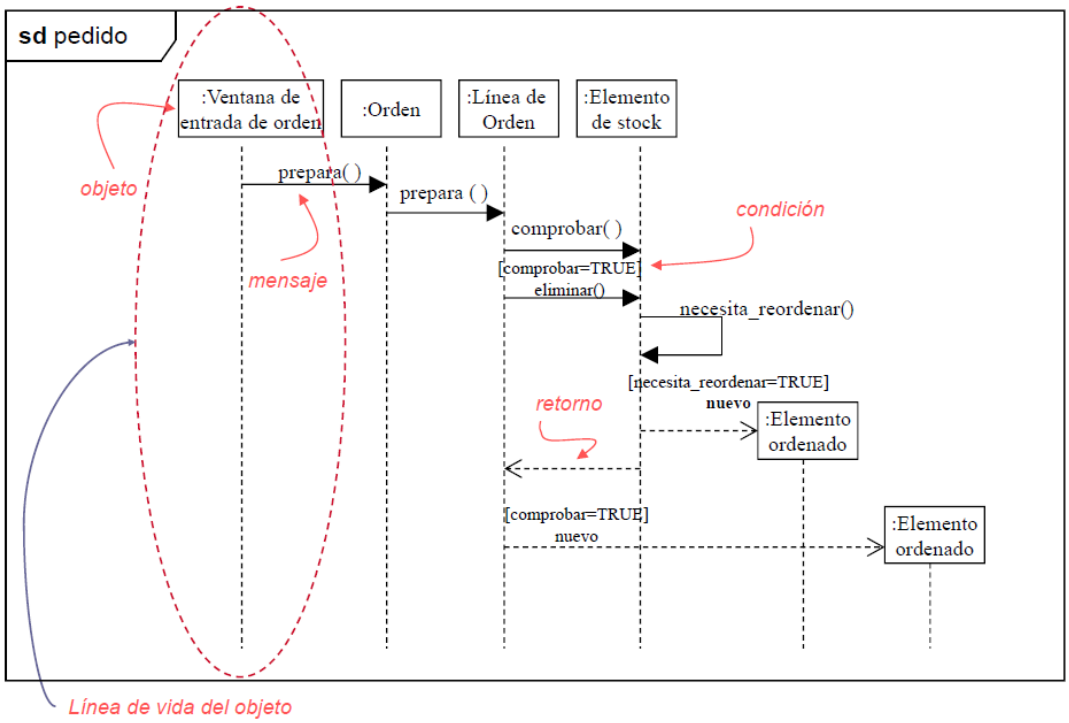
Este tipo de diagramas pueden utilizarse para mostrar un escenario específico o un escenario genérico (que contemple todas las alternativas posibles que pueden presentar ramas, condiciones y bucles).

La representación se realiza dentro de un marco rectangular con el nombre del caso de uso precedido del prefijo

sd.

El eje y del diagrama de secuencia está dispuesto en función del tiempo, de tal manera que los mensajes que se encuentren más arriba respecto del eje Y serán los que primero se ejecuten.



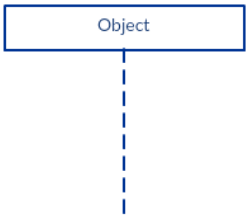


Vamos a aprender a realizar este tipo de diagramas paso a paso.

11.1 LÍNEA DE VIDA

Una línea de vida representa un participante individual en un diagrama de secuencia.

En UML se dibuja como una línea discontinua que contiene el participante en la parte superior de la misma. El participante suele ser un objeto.



Los diagramas de secuencia se forman por varias líneas de vida dispuestas de manera horizontal a lo largo del propio diagrama representando las partes del sistema que interactúan entre sí durante la secuencia.

Cuando el participante, al cual hace referencia la línea de vida, se representa mediante un símbolo de actor querrá decir que el diagrama de secuencia particular es propiedad de un caso de uso.

Se pueden encontrar otros símbolos como los que vamos a ver a continuación, no obstante, estos solo tienen sentido aplicando los conocimientos del patrón Modelo – Vista – Controlador.

Un elemento entidad (o modelo) representará los datos del sistema. Por ejemplo, en una aplicación de servicio al cliente la entidad “Cliente” gestionaría todos los datos relacionados con un cliente.

Un elemento límite representa un límite del sistema, como podrían ser las pantallas de interfaz del usuario, las puertas de la base de datos o los menús con los que interactúan los usuarios.

Los elementos de control ejercen de intermediario entre el modelo (entidad) y las vistas (límites), de tal manera que organiza y programa las interacciones entre ambos ejerciendo a su vez de mediador.

11.2 MENSAJES

Representa una transmisión de información entre objetos

La recepción de un mensaje por parte de un objeto acarreará una consecuencia, la cual implicará la realización de una actividad (ocurrencia de ejecución).

Cuando hablamos de mensajes, nos referimos a un concepto que engloba a las señales, las invocaciones a operaciones, las llamadas a procedimientos remotos, etc.

Es decir, el envío de un mensaje se puede disparar por una señal (se hace click en un botón), por la invocación de una función (se llama a la función calculateArea()), etc.

Los mensajes se representan con flechas entre las líneas de vida de los objetos.

Existen distintos tipos de mensajes y cada uno de ellos será representado de una manera concreta.

Los mensajes pueden tener elementos como nombre, parámetros y valor de retorno.

11.2.1 MENSAJE SÍNCRONO

Los mensajes síncronos son aquellos en los que el emisor espera a que el receptor responda al mensaje antes de continuar con otro.

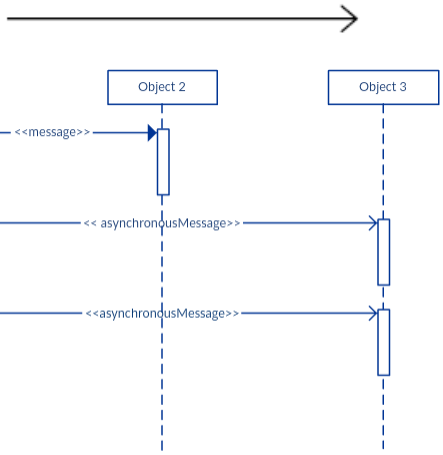
Para representarlo se utiliza una flecha continua con la punta rellena.

Un envío síncrono sería cuando llamamos a una función que tiene un return y usamos el resultado de este en las siguientes líneas de código.

11.2.2 MENSAJE ASÍNCRONO

En el caso de los mensajes asíncronos, el emisor realiza el envío del mensaje y continúa su flujo de ejecución, sin esperar contestación alguna por parte del receptor.

Para representarlo se utiliza una flecha continua con la punta sin rellenar y sin base.

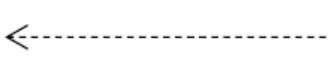


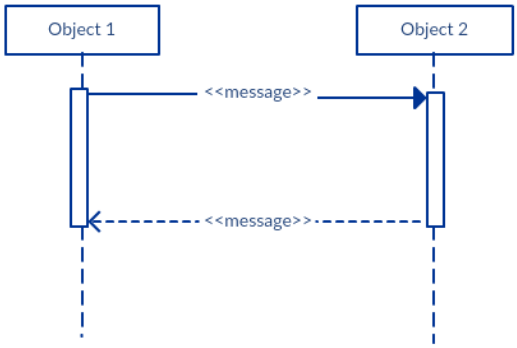
11.2.3 MENSAJE DE RETORNO

Tiene como finalidad indicar que el receptor del mensaje ha recibido el mensaje, lo ha procesado, ha elaborado una respuesta y la está enviando al emisor.

En muchas ocasiones la representación de estos mensajes es opcional, ya que una barra de activación que se dispara con un mensaje síncrono siempre implica un mensaje de retorno.

Para representarlo se utiliza una flecha discontinua con la punta sin rellenar y sin base.





EJERCICIO 63: Se quiere realizar un diagrama de secuencia que se corresponda con la siguiente realidad:

- Se va a dibujar el proceso mediante el cual se imprime un archivo.
- En primer lugar, el usuario va a decir al ordenador que quiere imprimir un archivo.
- El ordenador va a indicar a la impresora que ha de imprimir un archivo y le va a pasar el archivo a imprimir.
- Se devolverá confirmación de la impresión.

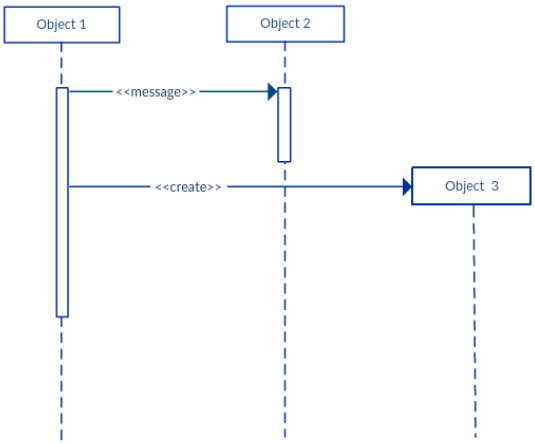
11.2.4 MENSAJE DE CREACIÓN

Este concepto cobra sentido cuando tenemos en cuenta que no todos los objetos viven durante toda la franja temporal que implica el propio diagrama.

En el caso de este tipo de mensajes un objeto le indica a otro que debe comenzar a existir.

La anotación de la casilla de participante eliminado puede utilizarse cuando se necesita mostrar que el participante en particular no existía hasta que se envió la llamada de creación.

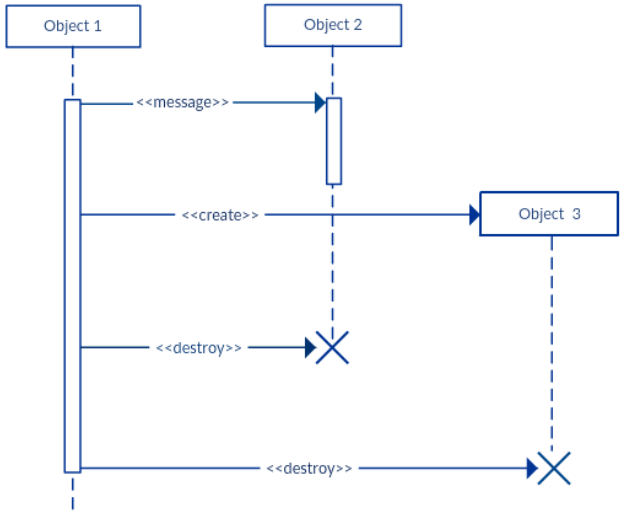
Para representarlo se utiliza una flecha de mensaje con el estereotipo <<**create**>>.



O una flecha discontinua con la punta sin rellenar y sin base (similar a la de retorno, pero en dirección izquierda derecha).

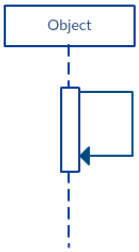
11.2.5 MENSAJE DE DESTRUCCIÓN

Este tipo de mensaje es análogo al anterior, de tal manera que los participantes cuando no se necesitan pueden ser eliminados del diagrama de secuencia, añadiendo una "X" al final de la línea de vida de dicho participante.



11.2.6 MENSAJE REFLEXIVO

Un mensaje reflexivo tiene lugar cuando un objeto se envía un mensaje a sí mismo.
Se indica con una flecha que comienza y termina en la misma línea de vida.



EJERCICIO 64: Se quiere realizar un diagrama de secuencia que se corresponda con la siguiente situación concreta:

- Se va a realizar la representación de un cajero.
- La comunicación realmente será entre una persona, un cajero, la sesión con la que se está trabajando y una base de datos.
- El usuario ha introducido la tarjeta de crédito en el cajero y este le ha pedido la contraseña.
- El usuario ha introducido la contraseña correctamente.
- Se ha generado una sesión.
- La interfaz muestra una serie de opciones al usuario.
- Cuando el usuario realiza una acción la sesión llama a la base de datos.
- Después de una serie de acciones solicitadas se ha pulsado el botón de salir.
- La sesión se ha destruido.
- El cajero indica al usuario que espere unos segundos.
- El cajero ha devuelto la tarjeta al usuario.

EJERCICIO 65: Se quiere realizar un diagrama de secuencia que se corresponda con la siguiente situación concreta:

- Se va a realizar la representación de un cajero.
- La comunicación realmente será entre una persona, un cajero, la sesión con la que se está trabajando y una base de datos.
- El usuario ha introducido la tarjeta de crédito en el cajero y este le ha pedido la contraseña.

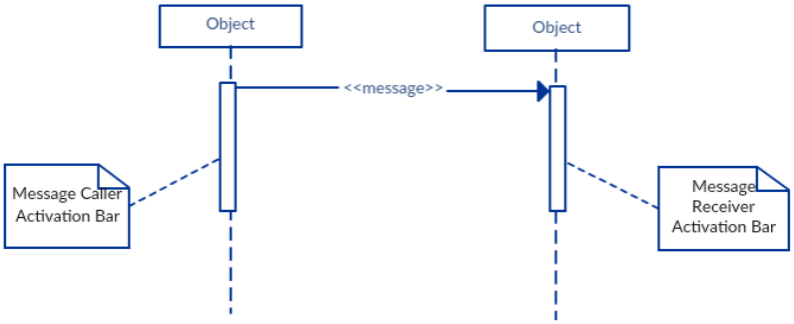
- El usuario ha introducido la contraseña correctamente.
- Se ha generado una sesión.
- La interfaz muestra una serie de opciones al usuario.
- Cuando el usuario realiza una acción la sesión llama a la base de datos.
- La sesión ha estado inactiva más de medio minuto.
- La sesión se ha destruido.
- El cajero indica que por motivos de seguridad la tarjeta ha sido almacenada.

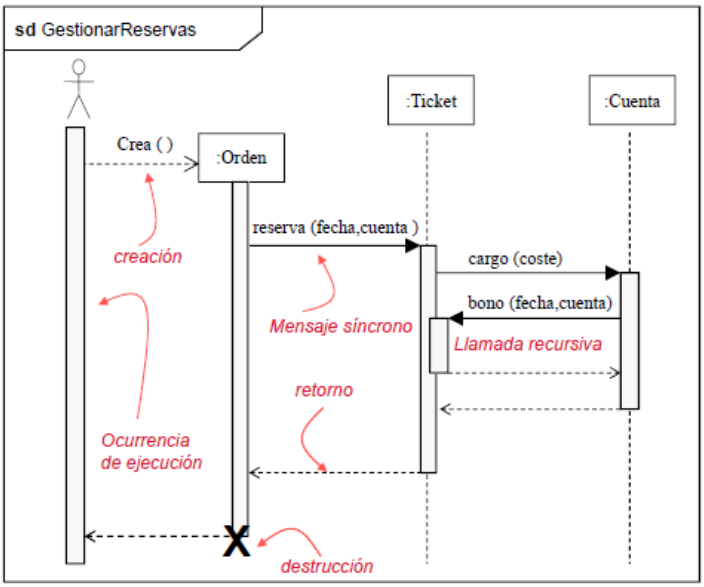
11.3 OCURRENCIA DE EJECUCIÓN

La ocurrencia de ejecución o barra de activación, indica que el objeto está activo o instanciado durante una interacción.

Se dice que un objeto está activado cuando se encuentra ejecutando su propio código o esperando el retorno de otro objeto al cual ha enviado un mensaje.

Su representación es opcional y el símbolo utilizado es un rectángulo estrecho sobre la línea de vida del objeto.





11.4 FRAGMENTOS COMBINADOS

Los fragmentos combinados encapsulan secciones del diagrama de secuencia que cuentan con un comportamiento especial.

Para representar los fragmentos se coloca el conjunto de trazas dentro de un marco (frame). En la parte superior izquierda del símbolo se pone la palabra clave correspondiente al operador.

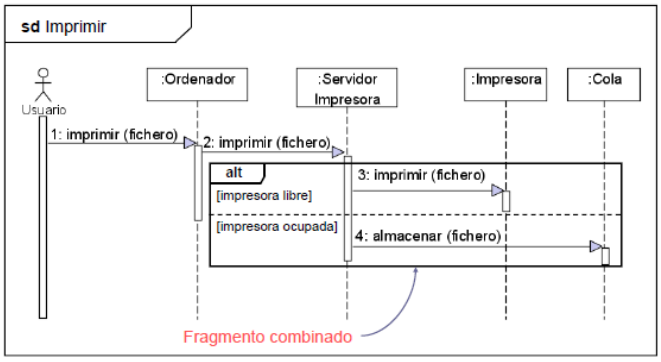


Diagrama de secuencia con un fragmento combinado

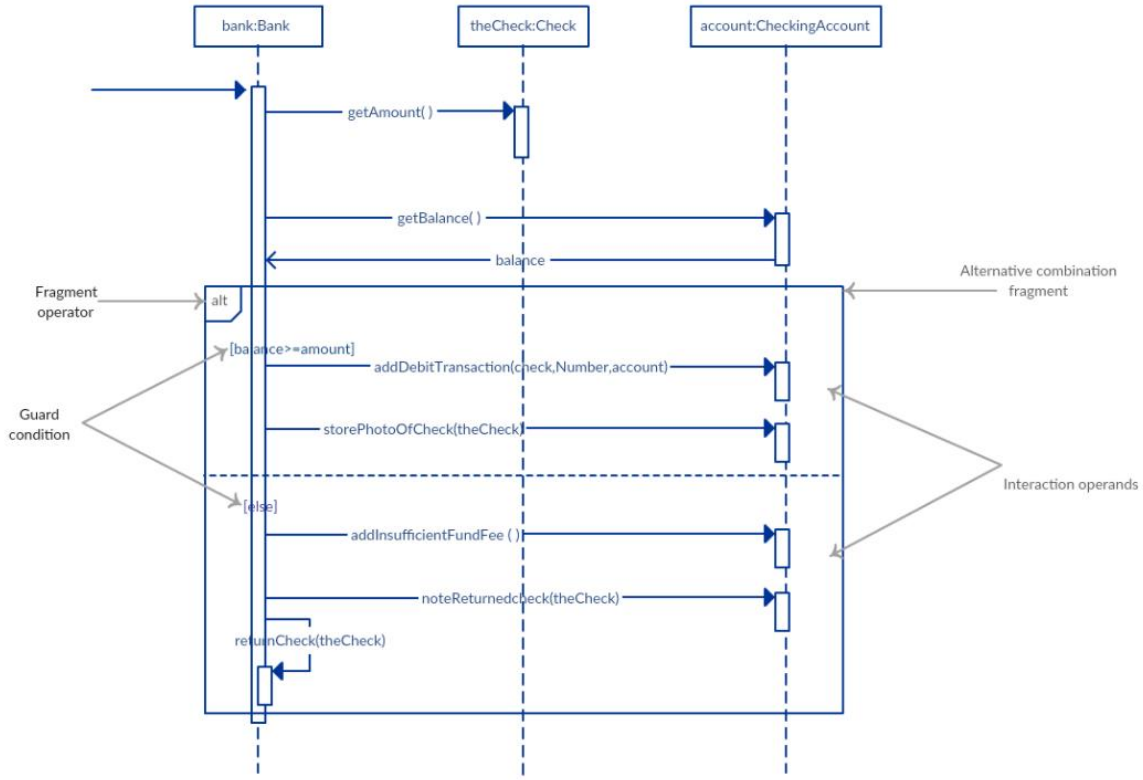
Existen distintos tipos de fragmentos combinados.

11.4.1 ALT

La denominada estructura alternativa se utiliza cuando es necesario elegir entre dos o más secuencias de mensajes.

Las alternativas se representan dividiendo el rectángulo mayor mediante líneas discontinuas formando lo que se llaman operandos de interacción.

Cada una de las alternativas incluirá una condición (o guarda) ubicada en la esquina superior izquierda del recuadro.



EJERCICIO 66: Se quiere realizar un diagrama de secuencia que se corresponda con la siguiente realidad genérica:

- Se va a realizar la representación de un cajero.
- La comunicación realmente será entre una persona, un cajero, la sesión con la que se está trabajando y una base de datos.
- Para comenzar el usuario ha de introducir la tarjeta de crédito en el cajero.
- El cajero solicitará la contraseña al usuario.
- Si el usuario introduce la contraseña de manera errónea el cajero devolverá la tarjeta.
- Si el usuario introduce la contraseña correctamente se generará una sesión.
- Con la sesión generada, la interfaz del cajero mostrará una serie de opciones al usuario.
- Cada vez que un usuario realice una acción se llamará a la base de datos.
- Si la sesión ha estado inactiva más de medio minuto esta se destruirá, el cajero se guardará la

tarjeta e indicará al usuario mediante su interfaz que esta ha sido almacenada.

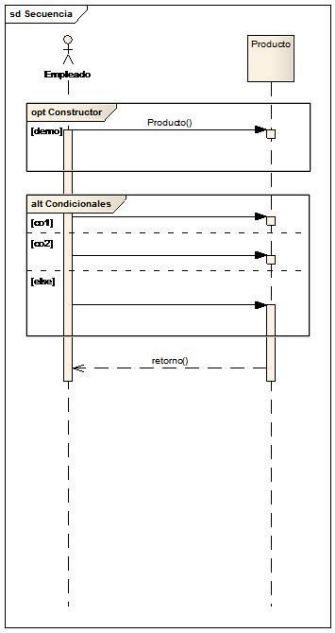
- Si el usuario pulsa el botón de salir la sesión se destruirá, el cajero indicará al usuario que espere unos segundos y devolverá la tarjeta.

11.4.2 OPT

Representa un comportamiento opcional, es decir, que puede ejecutarse este comportamiento o no.

Cabe destacar que los fragmentos opcionales solo se ejecutarán bajo una determinada condición, de lo contrario, la secuencia no se realizará.

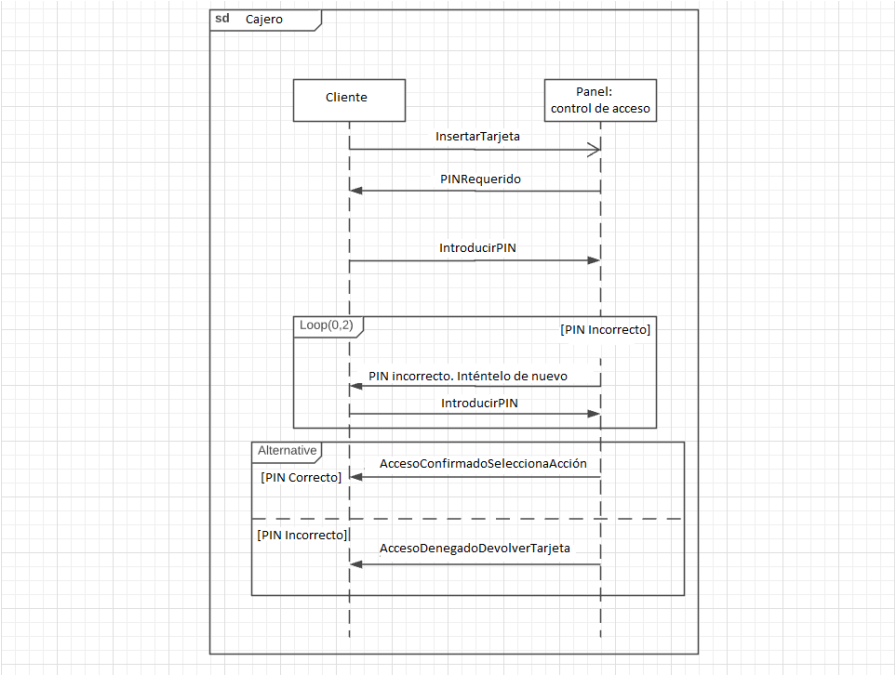
A diferencia del fragmento alternativo, un fragmento de opción no se divide en dos o más operandos.



11.4.3 LOOP

Se utiliza para representar bucles, es decir, comportamiento repetitivo.

La condición que determina si se sigue iterando o no será una prueba booleana, a su vez se pueden añadir una serie de condiciones especiales. Estas condiciones especiales indican el número de iteraciones mínimas y máximas (escritas como minint = [el número] y máximas (escritas como maxint = [el número]). Ej. loop (1,5).



EJERCICIO 67: Explica el diagrama de secuencia visto en la imagen anterior.

EJERCICIO 68: Se quiere realizar un diagrama de secuencia que se corresponda con la siguiente realidad genérica:

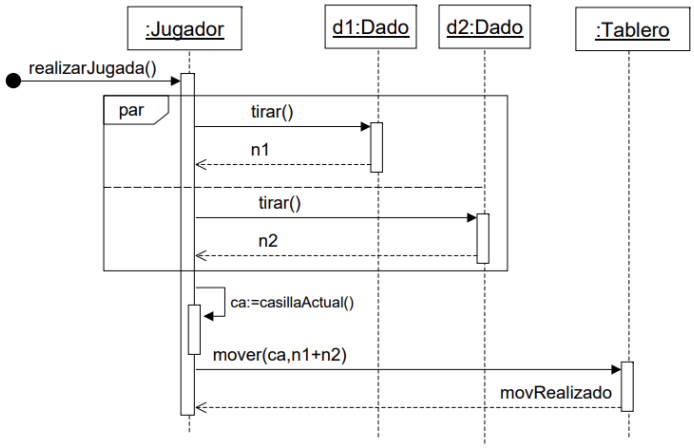
- Se va a realizar la representación de una página de registro de una web.
- El usuario pulsará el botón de registrarse y entonces se creará la página de registro.
- La página de registro pedirá al usuario una serie de datos (nombre, correo electrónico, contraseña, comprobación de contraseña, etc).
- El usuario podrá leer los términos y condiciones de uso haciendo click en el link clickable.
- Si el usuario procede a leer los términos y condiciones de uso se generará un modal que se puede cerrar haciendo click fuera de su área o en la pestañita de cerrar.
- Cuando el usuario pulsa a registrar pueden darse varias circunstancias.
    - Circunstancia 1:
        - Si algún parámetro está vacío se muestra un mensaje de error al usuario y queda a la espera de que el usuario modifique los datos y vuelva a pulsar el botón de registro.

- Circunstancia 2:
  - Si las contraseñas no coinciden se manda un mensaje de error al usuario y la página queda a la espera de que el usuario modifique los datos y vuelva a pulsar el botón de registro.
- Circunstancia 3:
  - Se comprueba el correo electrónico en la base de datos y se comprueba que este ya está registrado.
  - Muestra un mensaje de error al usuario y queda a la espera de que el usuario modifique los datos y vuelva a pulsar el botón de registro.
- Circunstancia 4:
  - Si todos los datos están correctos se almacena al usuario en la base de datos, se nuestra un mensaje de éxito al usuario, se destruye la página de registro y se mueve al usuario a una página de login y eso ya es otra historia.
  - Muestra un mensaje de error al usuario y queda a la espera de que el usuario modifique los datos y vuelva a pulsar el botón de registro.

- Con la sesión generada, la interfaz del cajero mostrará una serie de opciones al usuario.
- Cada vez que un usuario realice una acción se llamará a la base de datos.
- Si la sesión ha estado inactiva más de medio minuto esta se destruirá, el cajero se guardará la tarjeta e indicará al usuario mediante su interfaz que esta ha sido almacenada.
- Si el usuario pulsa el botón de salir la sesión se destruirá, el cajero indicará al usuario que espere unos segundos y devolverá la tarjeta.

11.4.4 PAR

Se utiliza para representar comportamientos que se ejecutan en paralelo.



11.4.5 CRITICAL

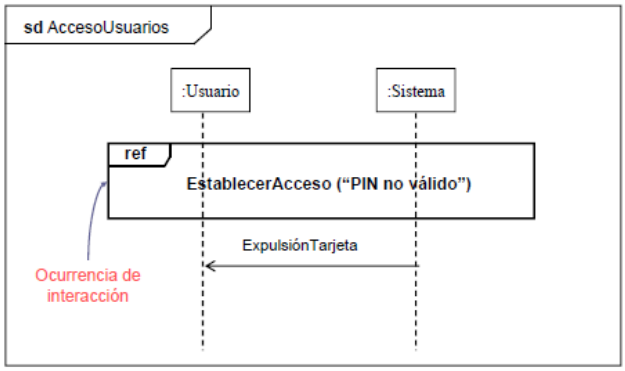
Este tipo de fragmento que solo tiene sentido en comportamientos en paralelo.
Cuando varios procesos en paralelo quieren acceder a un solo recurso hay ocasiones en las que es necesario controlar el acceso a este, evitando los problemas que pueden surgir cuando varios procesos comparten recursos.

11.5 OCURRENCIA O USO DE INTERACCIÓN

Permite cualquier interacción referenciar a una interacción que representa un comportamiento usable en otros contextos.

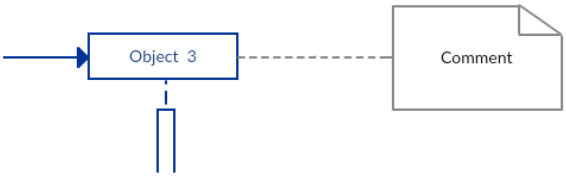
Se representa con el mismo formato que el de los fragmentos combinados haciendo uso de la palabra clave ref.

La sintaxis utilizada será la siguiente: **nombre [(argumentos)] [:valorRetorno]**



11.6 COMENTARIOS

Los diagramas UML, generalmente, permiten la anotación de comentarios.
Este se suele representar como un rectángulo con una esquina doblada, el cual se vincula al objeto relacionado mediante una línea de puntos.



EJERCICIO 69: Realiza el diagrama de secuencia correspondiente al siguiente fragmento de código.

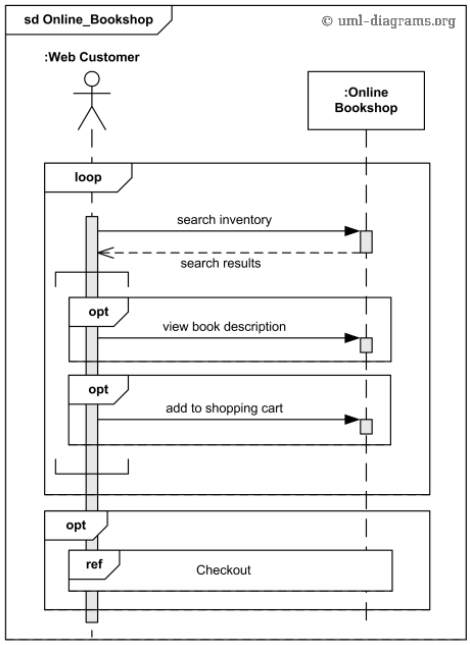
```
public class JuegoLaberinto {
   public Laberinto crearLaberinto () {
      Laberinto lab = new Laberinto();
      Habitacion h1 = new Habitacion();
      Habitacion h2 = new Habitacion();
      Puerta puerta = new Puerta(h1, h2);
      lab.añadeHabitacion(h1);
      lab.añadeHabitacion(h2);
      h1.añadePuerta(puerta);
      return lab;
   }
}
```

EJERCICIO 70: Realiza el diagrama de secuencia correspondiente al siguiente fragmento de código.

```
public class JuegoLaberinto {
   private Laberinto lab;
   private boolean conVentana;

   public JuegoLaberinto() {
      lab = new Laberinto();
      conVentana = true;
   }

   public void crearLaberinto () {
      Habitacion h;
      for (int i=0; i<10; i++) {
         h = new Habitacion();
         if (conVentana == true)
            h.añadeVentana(new Ventana());
         lab.añadeHabitacion(h);
      }
}
```

EJERCICIO 71: Realiza el diagrama de secuencia correspondiente al cuento de los tres cerditos.

EJERCICIO 72: Explica el siguiente diagrama.



EJERCICIO 73: Se quiere realizar un diagrama de secuencia sobre la modificación de los datos de un usuario en una aplicación. Se ha de usar el patrón MVC.

EJERCICIO 74: Realiza los diagrama de secuencia correspondientes a los casos de uso comprometidos del enunciado 1.

EJERCICIO 75: Realiza los diagrama de secuencia correspondientes a los casos de uso comprometidos del enunciado 2.

12. REFERENCIAS

https://www.teatroabadia.com/es/uploads/documentos/iagramas_del_uml.pdf
https://www.ctr.unican.es/asignaturas/mc_oo/doc/m_estructural.pdf
https://ingsotfwarekarlacevallos.wordpress.com/2015/07/02/uml-relaciones-entre-clases/
http://elvex.ugr.es/decsai/java/pdf/3c-relaciones.pdf
http://openaccess.uoc.edu/webapps/o2/bitstream/10609/69245/2/Ingenier%C3%ADa%20del%20software_M%C3%B3dulo%204_UML%20%28I.pdf
https://www.lucidchart.com/pages/es/que-es-un-diagrama-de-paquetes-uml#:~:text=Los%20diagramas%20de%20paquetes%20son,o%2C%20incluso%2C%20otros%20paquetes.
https://ocw.unican.es/pluginfile.php/1403/course/section/1792/is1_t09_trans.pdf
https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/
https://www.ctr.unican.es/asignaturas/MC_OO/Doc/Casos_de_uso.pdf
https://www.seas.es/blog/informatica/tipos-de-relaciones-en-diagramas-de-casos-de-uso-uml/
https://www.abiztar.com.mx/articulos/casos-a-incluir-casos-a-extender.html
https://creately.com/blog/es/diagramas/tutorial-del-diagrama-de-secuencia/
http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.php
http://ocw.uc3m.es/ingenieria-informatica/metodologia-de-desarrollo-visual/course-files/material-del-tema-6
https://www.codecompiling.net/files/slides/UML_clase_06_UML_secuencia.pdf

https://ingsotfwarekarlacevallos.wordpress.com/2015/07/07/uml-diagrama-de-secuencia/
https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagramas-de-secuencia/
https://www.seas.es/blog/informatica/operadores-de-control-yo-marcos-de-interaccion-uml-2/
http://myfpschool.com/ejercicio-uml-ejercicios-resueltos/
https://slideplayer.es/slide/3363740/
https://es.slideshare.net/junsaed/metodologia-elicitacion-6734535
http://www.lsi.us.es/~javierj/cursos_ficheros/NDT/EARF.pdf
https://www.studocu.com/ec/document/universidad-de-las-fuerzas-armadas-de-ecuador/bases-de-datos/practica/ejemplo-de-elicitacion-de-requisitos-de-software/5283818/view
https://diagramasuml.com/casos-de-uso/#Ejemplos_de_un_diagrama_de_casos_de_uso
http://clasev.net/aulavirtual/pluginfile.php/49045/mod_resource/content/1/ejemplos%20-%20casos%20de%20uso.pdf
http://myfpschool.com/examen-uml-diagrama-de-casos-de-uso-gueraryu/
http://www.lsi.us.es/~javierj/cursos_ficheros/metricaUML/CasosUsoUML.pdf
https://lh3.googleusercontent.com/proxy/htmGW-k3omXFdF5jIXKxwm4Jya23LYWKJrQQdiAD7FMtXc6rfSNQ1fR4S6dhTJB0Y5cAUrec5KsXNrQV6lpuvIdxgalNq9DadT8hB-uIByIaroFFLf6klQ
https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-paquetes/
https://www.youtube.com/watch?v=ataioNckj-E
https://www.researchgate.net/profile/Tatjana-Kutzner/publication/312892724/figure/fig5/AS:454473521864708@1485366339849/A-UML-package-merge-example-ISO-2012b-modified.png
http://www.lsi.us.es/docencia/get.php?id=6845
http://www.lsi.us.es/~javierj/cursos_ficheros/02.%20Un%20ejemplo%20de%20requisitos.pdf
http://techuejutlafundamentosdeingenieriafch.blogspot.com/2013/04/3_20.html
https://laurel.datsi.fi.upm.es/_media/docencia/cursos/java/ejercicio_mvc.pdf
https://ocw.unican.es/pluginfile.php/1403/course/section/1792/is1_t08_trans.pdf
https://joanpaon.files.wordpress.com/2013/06/6.png?w=300&h=184
https://joanpaon.files.wordpress.com/2013/07/18.png
http://ocw.uc3m.es/ingenieria-de-sistemas-y-automatica/informatica-industrial-i/ejercicios_modelado.pdf
https://www.uml-diagrams.org/index-examples.html
http://arantxa.ii.uam.es/~eguerra/docencia/0708/00%20Tema%201.pdf
http://ocw.uc3m.es/ingenieria-de-sistemas-y-automatica/informatica-industrial-i/ejercicios_modelado.pdf
https://idoc.pub/documents/ejercicios-resueltos-de-diagramas-de-clases-uml-mwl1323zkvnj
https://repositorio.grial.eu
https://i.pinimg.com/originals/98/85/f2/9885f2d2d82ce9384952270aa6b5d1af.png
https://chpeti20182916533home.files.wordpress.com/2019/03/interaccion.jpg?w=640
https://ingsotfwarekarlacevallos.files.wordpress.com/2015/07/81.png?w=640
https://sites.google.com/site/proyectoanalisisutb/_/rsrc/1403030487949/file-cabinet/5.png
https://i.pinimg.com/originals/d1/50/50/d15050517c2634bd424d1e7c2db0979e.png
https://d2slcw3kip6qmk.cloudfront.net/marketing/pages/chart/UML-state-diagram-tutorial/AirportCheckInExample.PNG
https://d2slcw3kip6qmk.cloudfront.net/marketing/pages/chart/discovery-2020/feature-images/package-diagram-UML-feature-image-2020@2x.png
https://diagramasuml.com/wp-content/uploads/2018/11/despliegue-1.png
https://p.calameoassets.com/150727020819-993c5e9628d745267ca4d61be9703e16/p1.jpg
https://sites.google.com/site/metodologiareq/capitulo-ii/tecnicas-para-identificar-requisitos-funcionales-y-no-funcionales#_Toc324099575
Apuntes de ingeniera del Software de la USAL
https://sekthdroid.wordpress.com/tag/uml-modificadores-de-visibilidad/