ENTORNOS DE DESARROLLO

TEMA 5:

MODELADO

Desarrollo de Aplicaciones Multiplataforma

Curso 2021/2022

Índice

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.

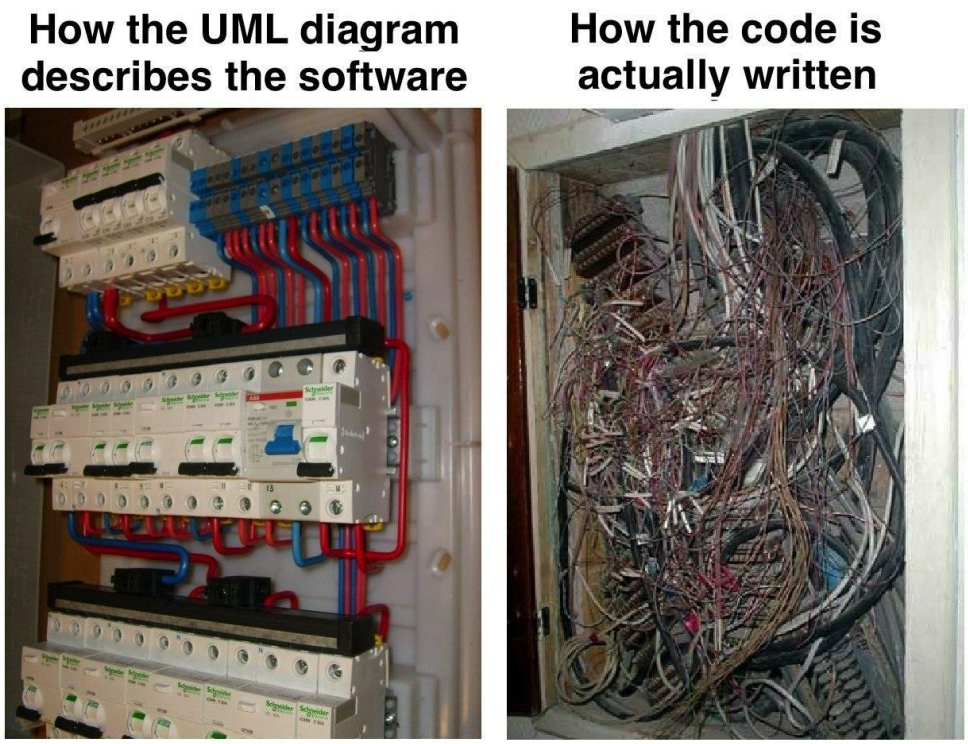
UML (Unified Modeling Language o Lenguaje Unificado de Modelado) es un lenguaje visual de modelado para mostrar, especificar, construir y documentar partes de un sistema software desde distintos puntos de vista y que se ha adoptado como estándar a nivel internacional por una gran cantidad de organismos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

- **Sintaxis:** En un lenguaje de modelado basado en diagramas, la sintaxis determina las reglas y principios que determinan que diagramas son legales. (formas)
- **Semántica** Significado de los elementos y palabras del lenguaje. (fondo)

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:

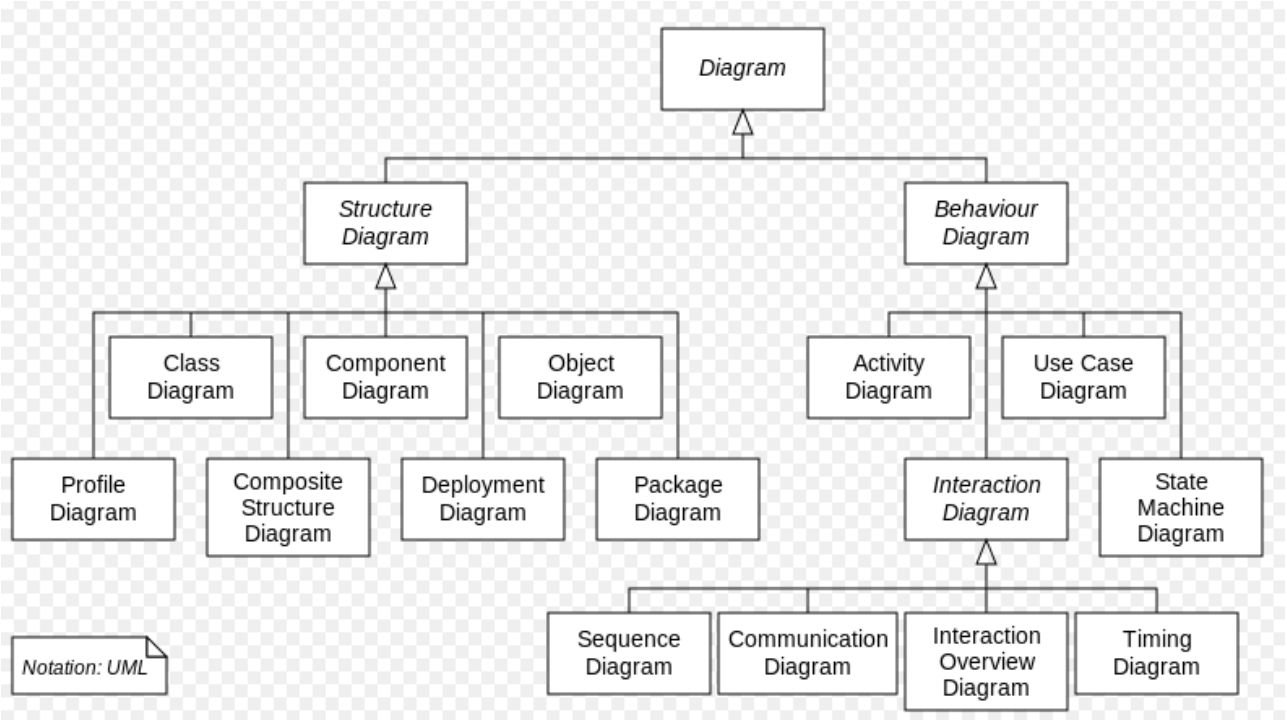
- **Estructuras:** Son los nodos del grafo y definen el tipo de diagrama.

- **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).

El 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.

Los diagramas se clasifican en:

- **Diagramas estructurales**
- **Diagramas de comportamiento**



La mayor parte de la información de los diagramas que se muestran a continuación ha sido extraída de la web de Visual Paradigm (Visual Paradigm, s.f.) y alguna del libro en Entornos de Desarrollo de Comesaña (Comesaña)

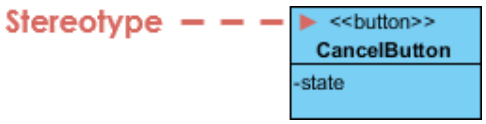
1.3.1. Diagramas estructurales

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

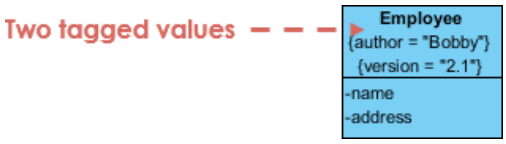
- **Diagramas de perfiles**: Se trata de un mecanismo de extensibilidad que permite ampliar y personalizar UML mediante la adición de nuevos bloques de construcción, la creación de nuevas propiedades y la especificación de nueva

semántica para que el lenguaje se adapte a su dominio de problemas específico.
Tiene tres tipos de mecanismos de extensibilidad:

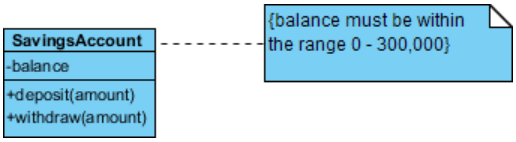
- **Estereotipos**



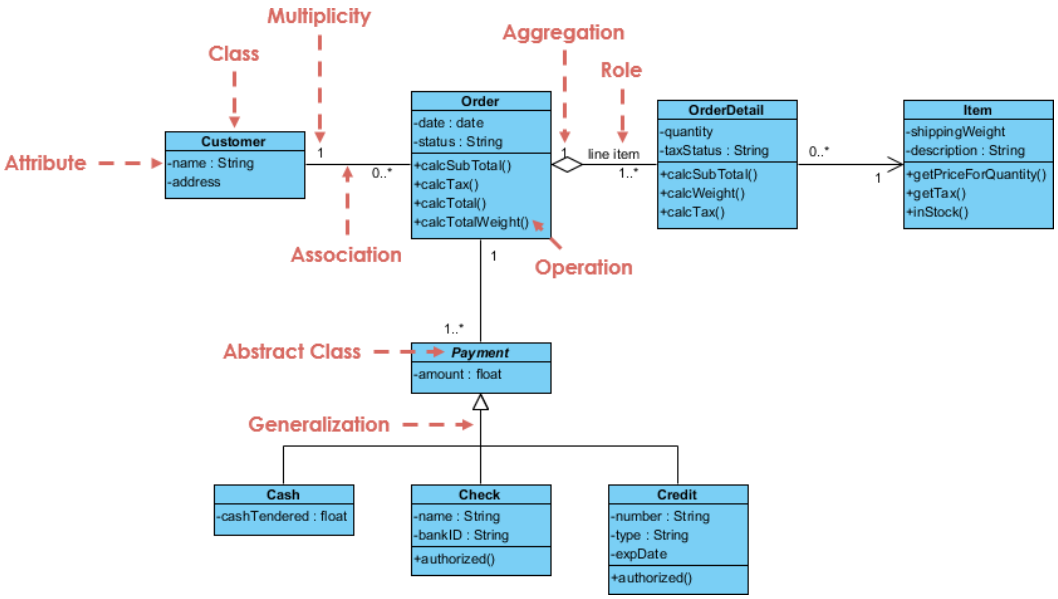
- **Valores etiquetados**



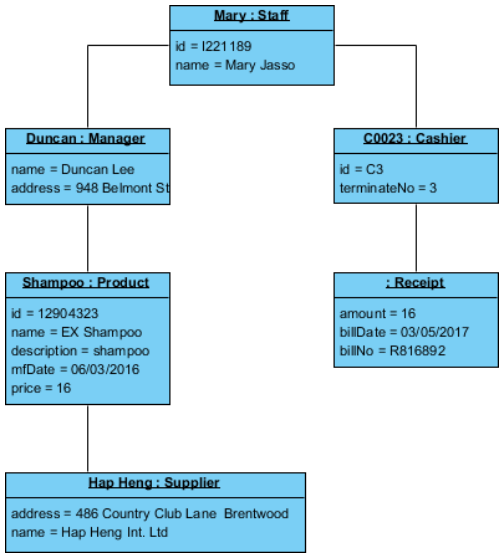
- **Restricciones**



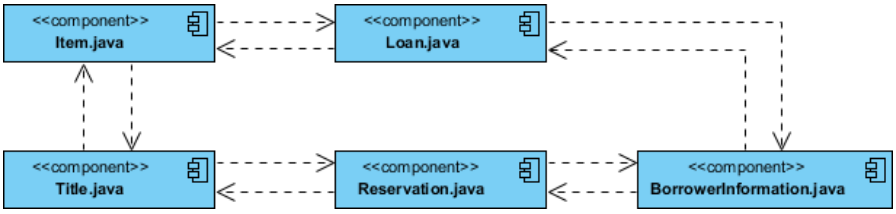
- **Diagramas de clases:** Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones, es una notación gráfica utilizada para construir y visualizar sistemas orientados a objetos. Tiene una prioridad alta.



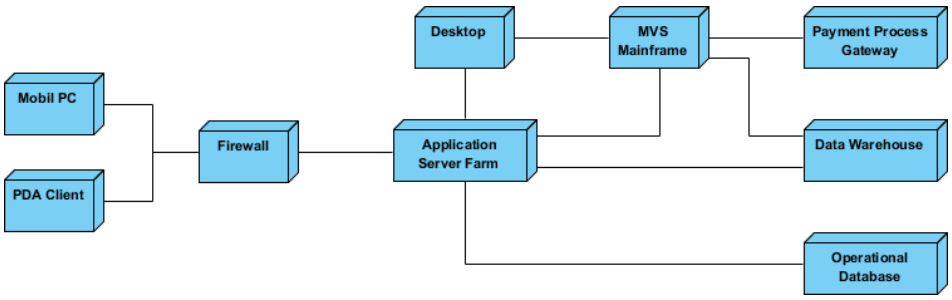
- **Diagrama de objetos:** Un objeto es una instancia de una clase en un momento determinado del tiempo de ejecución que puede tener su propio estado y valores de datos. Asimismo, un diagrama de objetos UML estático es una instancia de un diagrama de clases; muestra una instantánea del estado detallado de un sistema en un momento dado.



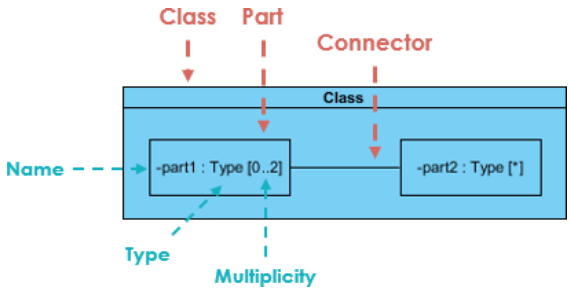
- **Diagrama de componentes:** Especifican la organización lógica de la implementación de una aplicación, sistema o empresa, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos.



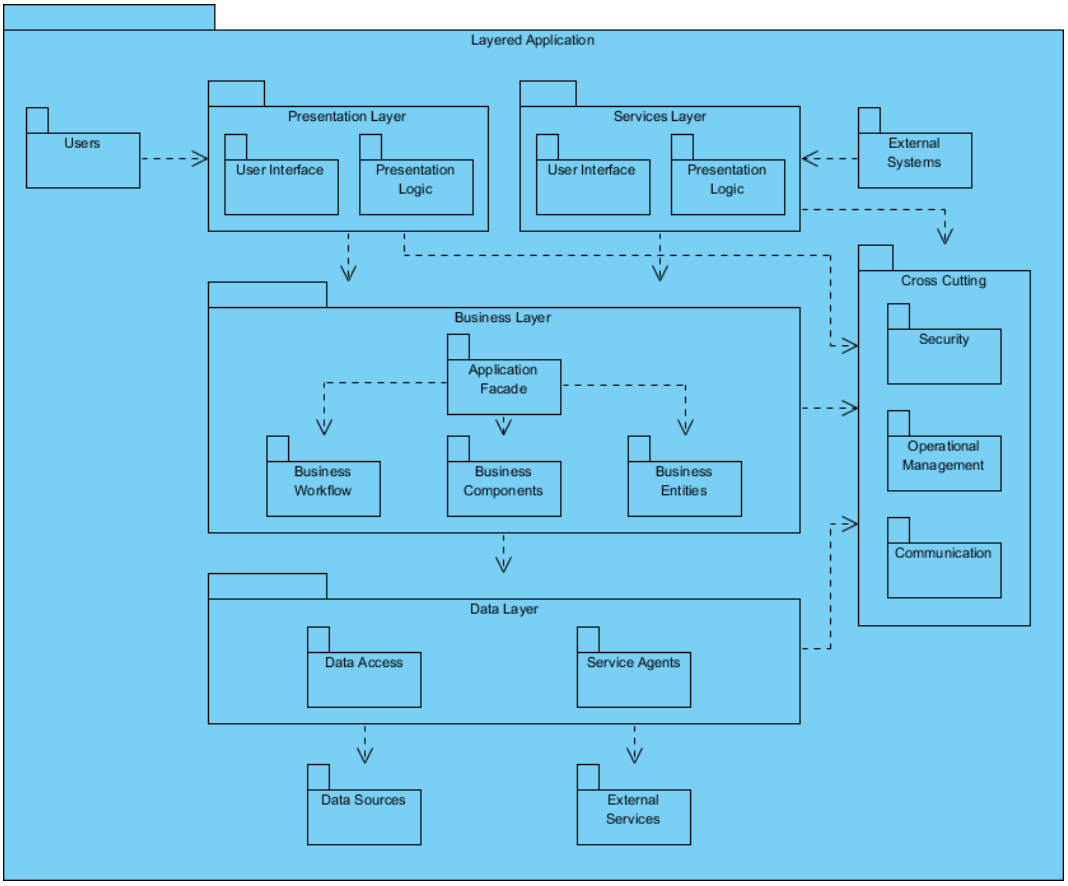
- **Diagramas de despliegue:** Se utilizan para modelar la vista de implementación estática de un sistema.



- **Diagrama integrado de estructura:** Se trata de un diagrama que contiene clases, interfaces, paquetes y sus relaciones, y que proporciona una visión lógica de todo o parte de un sistema de software.
  Un diagrama de estructura compuesta desempeña una función similar a la de un diagrama de clases, pero permite entrar en más detalles al describir la estructura interna de varias clases y mostrar las interacciones entre ellas.



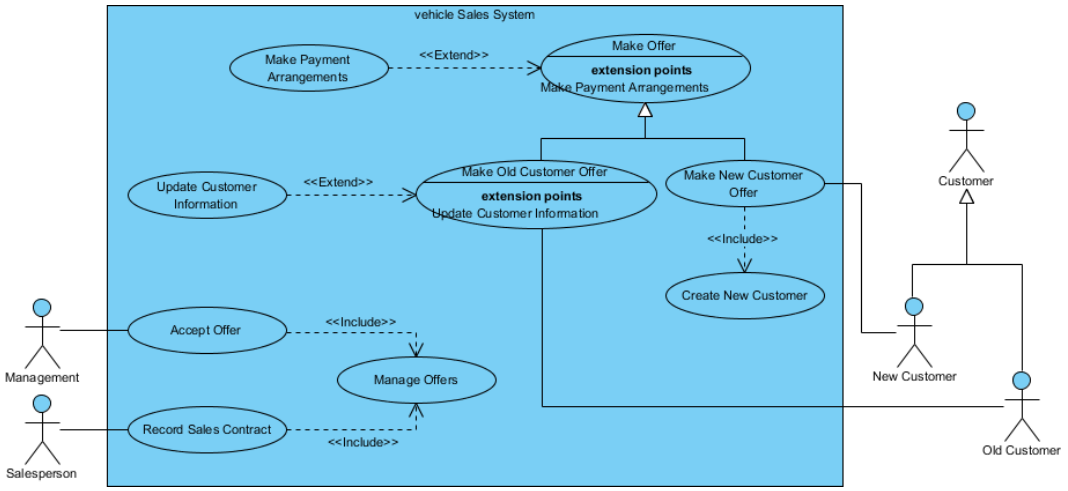
- **Diagrama de paquetes:** Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. Suele ser útil para la gestión de sistemas de mediano o gran tamaño.



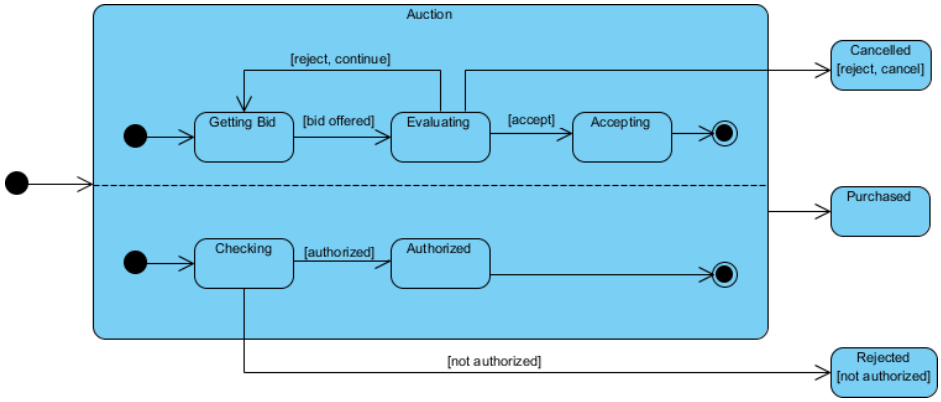
1.3.2. Diagramas de comportamiento

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.

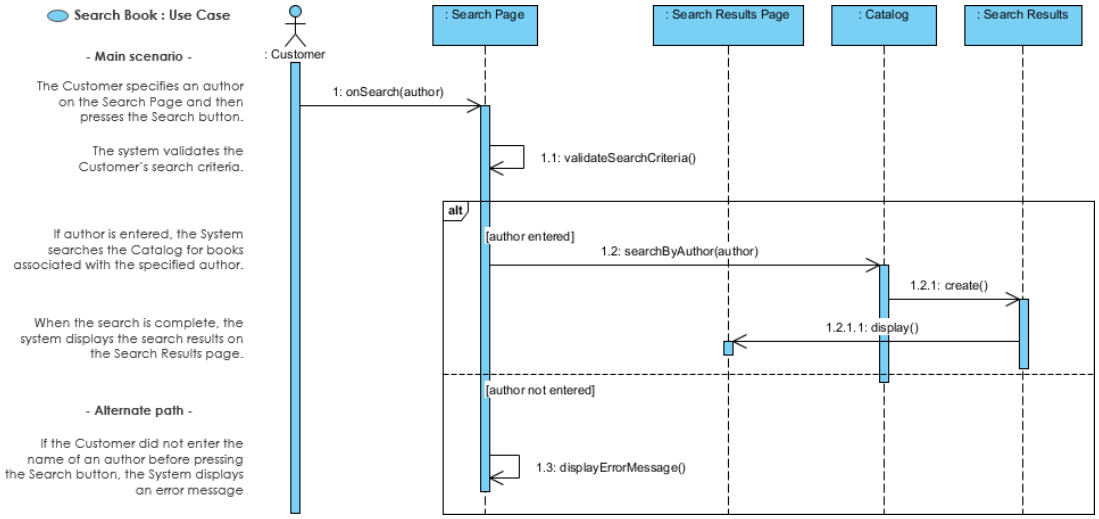
- **Diagramas de casos de uso:** Un diagrama de casos de uso UML es el método más utilizado para especificar los requisitos funcionales de un programa de software en desarrollo. Los casos de uso especifican el comportamiento esperado (qué), y no el método exacto para hacerlo realidad (cómo).



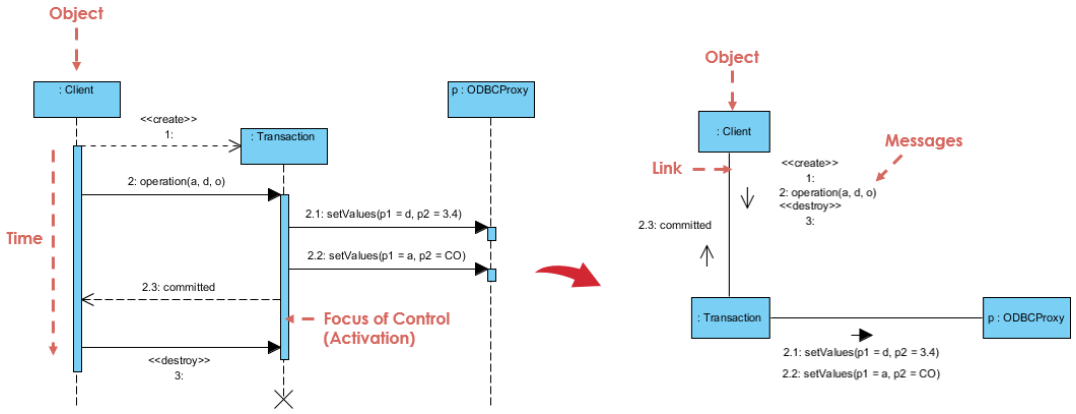
- **Diagramas de estado de la máquina:** El comportamiento de una entidad no es sólo una consecuencia directa de sus entradas, sino que también depende de su estado anterior. La historia pasada de una entidad se puede modelar mejor mediante un diagrama de máquina de estados finitos o tradicionalmente llamado autómata. Los diagramas de máquinas de estado UML (o a veces denominados diagrama de estado, máquina de estado o gráfico de estado) muestran los diferentes estados de una entidad. Los diagramas de máquinas de estado también pueden mostrar cómo una entidad responde a varios eventos cambiando de un estado a otro.



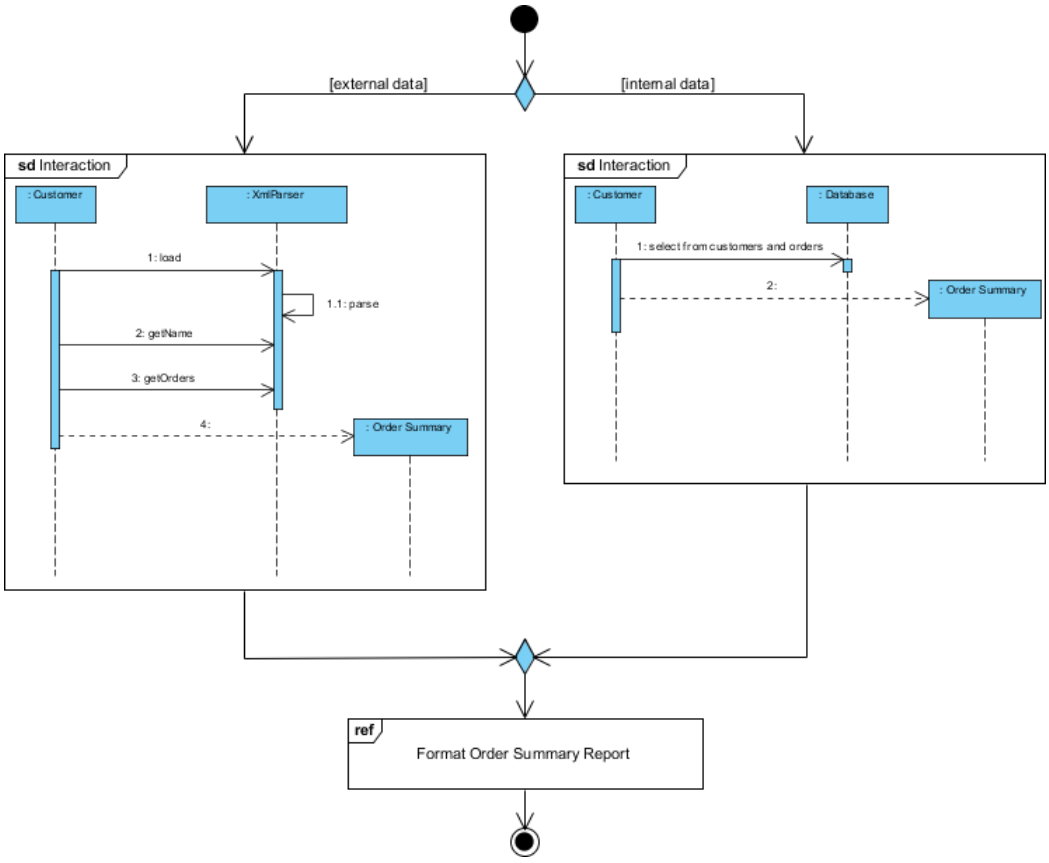
- **Diagrama de actividades:** Muestran el orden en el que se van realizando tareas dentro de un sistema. En él aparecen los procesos de alto nivel de la organización. Incluye flujo de datos, o un modelo de la lógica compleja dentro del sistema.
- **Diagramas de interacción:**
    - **Diagramas de secuencia:** Este tipo de diagramas detallan cómo se realizan las operaciones. Capturan la interacción entre objetos en el contexto de una colaboración. Los Diagramas de Secuencia se centran en el tiempo y muestran el orden de la interacción visualmente utilizando el eje vertical del diagrama para representar en el tiempo qué mensajes se envían y cuándo.



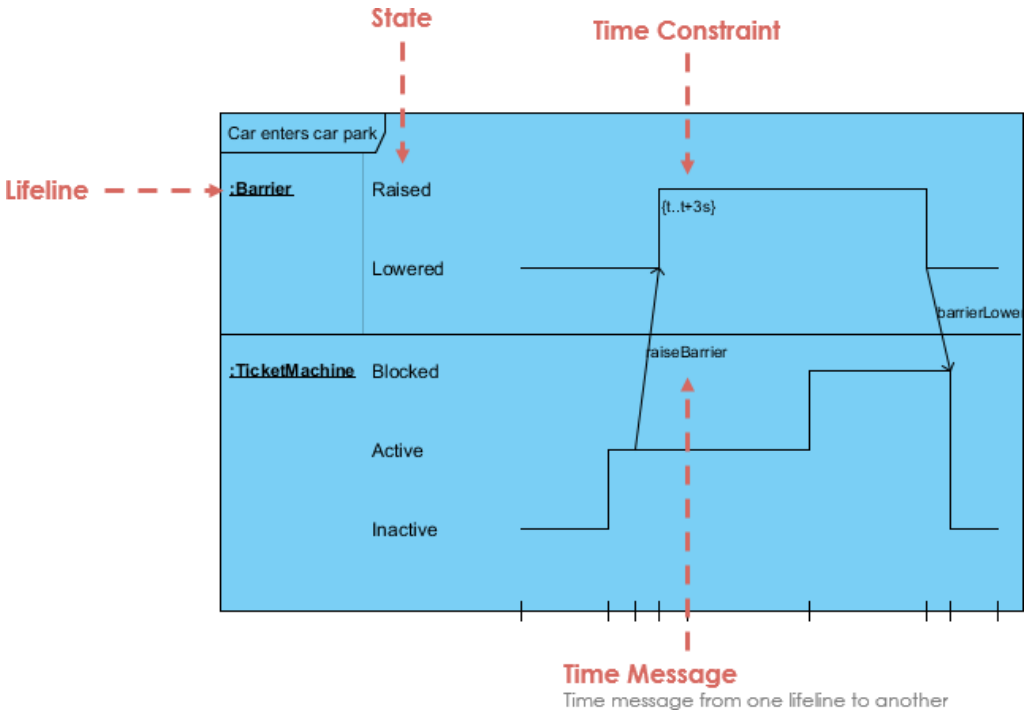
- **Diagramas de comunicación/colaboración :** Resaltan la organización estructural de los objetos que se pasan mensajes. Ofrece las instancias de las clases, sus interrelaciones, y el flujo de mensajes entre ellas. Comúnmente enfoca la organización estructural de los objetos que reciben y envían mensajes.



- **Diagrama general de interacción:** . El Diagrama de Visión General de Interacción se centra en la visión general del flujo de control de las interacciones que también puede mostrar el flujo de actividad entre los diagramas. En otras palabras, usted puede enlazar los diagramas "reales" y lograr un alto grado de navegabilidad entre los diagramas dentro de un Diagrama de Visión General de Interacción.



- **Diagrama de tiempos:** Se usa normalmente para exhibir el cambio en el estado de un objeto en el tiempo, en respuesta a eventos externos.



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.

2.3. 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	

2.3.1. Ejemplo

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	

2.3.2. Ejercicio

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 de viajes que gestione vuelos y alojamientos.

Las aerolíneas y acomodadores que quieren aparecer en la aplicación se ponen en contacto con el equipo administrativo de la app para que les dé de alta y han de facilitar las APIs correspondientes para que la aplicación extraiga la información de sus 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, una duración de vuelo determinada.

Cada avión tiene varias plazas, que tienen un número, una letra. Es necesario conocer también si la plaza está ocupada o no y el precio que tiene.

Respecto a los acomodadores, se entiende por acomodador a un particular o una entidad que ofrece uno o varios alojamientos.

Los alojamientos pueden ser de varios tipos (habitación de hotel, apartamento, casa particular, etc) y es necesario conocer su precio y disponibilidad.

Todos los alojamientos tienen, por lo menos, un nombre, unas fotos, una descripción, un tamaño (número de personas) y una ubicación.

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. Se pueden ordenar los vuelos por duración o por precio (del asiento más barato) y se pueden filtrar los vuelos por aquellos a los que le queda algún asiento disponible (se pueden especificar el número de asientos disponibles).

Dese esta lista se puede acceder a la reserva de un asiento.

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 y si tienen el mismo precio.

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, unas plazas 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 un alojamiento, 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 en la tarjeta con la que ha pagado.

Cuando una aerolínea o alojamiento se da de baja, el operario de la app elimina todos sus datos (hay que tener en cuenta que las reservas que ya se han hecho en esas entidades se han de mantener).

La aplicación genera de manera automática una planificación con las reservas del usuario (tanto de vuelos como de alojamientos).

3. 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.

En este curso vamos a utilizar la versión gratuita de uno de los softwares de creación de diagramas más completo y más reconocido.

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

Existen otras alternativas gratuitas como Modelio , StarUML o, incluso, usar directamente Visual Paradigm.

4. DIAGRAMAS DE CASOS DE USO

Un diagrama de casos de uso UML es el método más utilizado para especificar los requisitos funcionales de un programa de software en desarrollo.

Los requisitos funcionales se tratan de funcionalidades que el cliente exige en el sistema que ha solicitado.

A partir del diagrama de casos de uso 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.

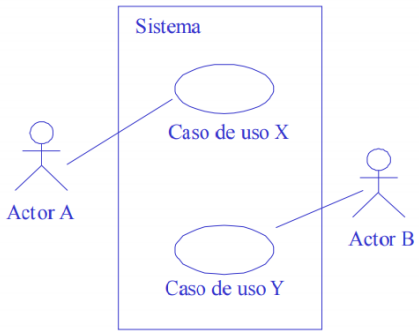
El siguiente paso en el modelado que vamos a hacer va a ser crear un diagrama de casos de uso por cada objetivo y un diagrama de casos de uso general. En muchas ocasiones se usan otras políticas, como hacer diagramas de caso de uso por cada actor.

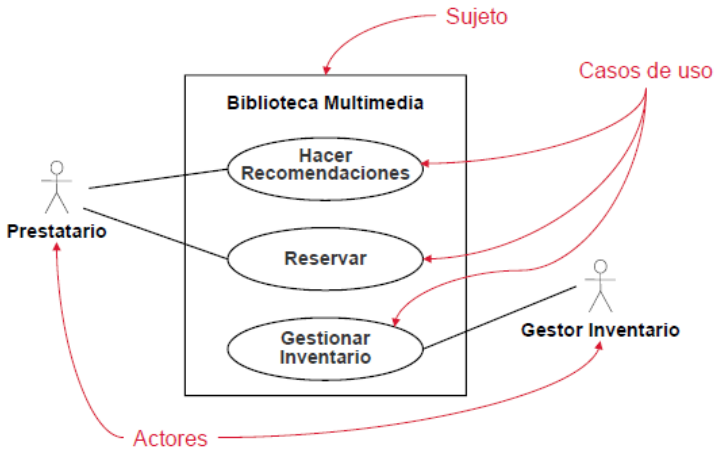
4.1. Sujeto o sistema

Se trata del 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.

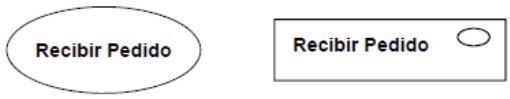
Un caso de uso especifica qué hace la aplicación sin entrar en el detalle de cómo lo hace.

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	El sistema muestra un formulario con los datos a rellenar. El actor administrador (ACT-002) rellena los campos requeridos para dar de alta un Empleado (nombre, apellidos, DNI…). El sistema guarda la información referente al empleado. El sistema muestra un mensaje indicando que el proceso de alta se ha realizado con éxito
Postcondition	
Excepciones	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 cómo.

4.3. Actores

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

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

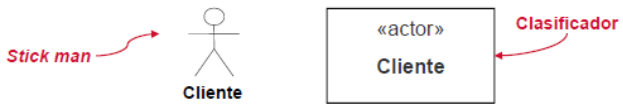
Es importante tener en cuenta que lo que denominamos actor, representa el “papel” o “rol” que un usuario lleva a cabo respecto a su forma de interactuar con el sistema, es decir, una sola persona física puede actuar como actores diferentes.

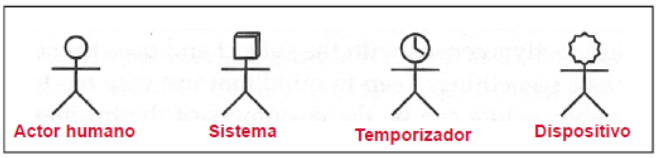
Los actores se representan mediante el icono de “stick man” o monigote con el nombre del actor cerca del símbolo. Por convenio se suele poner encima o debajo.

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.

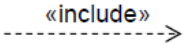
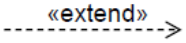
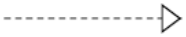
- **Comentarios**: Cualquier otra información que no encaje en los campos anteriores.

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 3: Identifica los actores que puedan tomar parte en el ejercicio grande 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	
Extensión	
Realización	

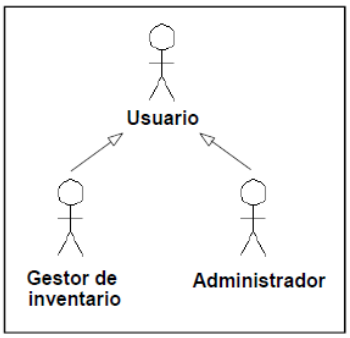
4.4.1. Entre actores

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.



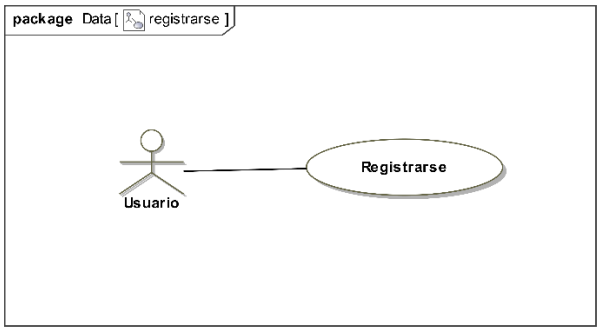
4.4.2. Entre actores y casos de uso

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 4: 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. Entre casos de uso

4.4.3.1. 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 5: 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.

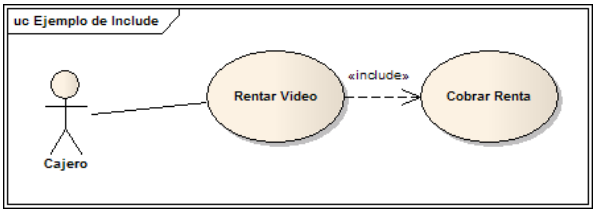
4.4.3.2. Inclusión

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

- El sistema corresponde a una máquina de cobro.
- El sistema tiene como única finalidad realizar la venta del producto.
- Realizar venta implica imprimir comprobante y cobrar venta.
- Cobrar venta se puede realizar en efectivo, mediante tarjeta de crédito o cobrando con cheque.

- Al utilizar la tarjeta de crédito, el sistema siempre solicita al cliente introduzca el pin.

4.4.3.3. Extensión

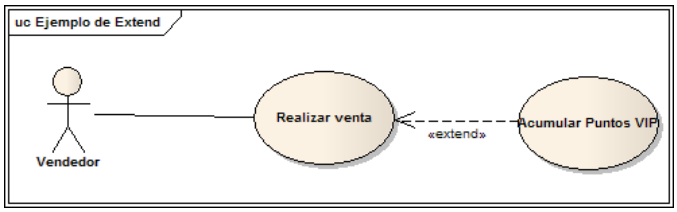
Este tipo de relaciones implican que un caso de uso específico añada acciones opcionales al comportamiento de un caso de uso general. El caso de uso extendido no tiene por qué incluir todo el comportamiento del caso de uso que se extiende.

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.

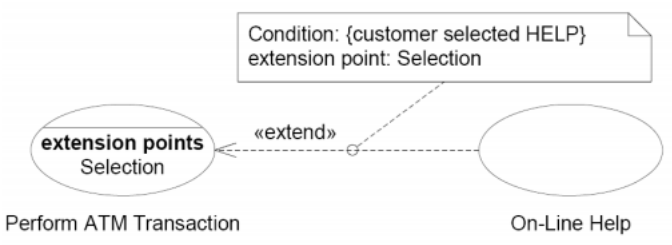


A la hora extender un caso de uso puede ser una buena práctica definir puntos de extensión (extension points), esto decir, especificar el punto del caso de uso donde insertar la extensión para ampliar la funcionalidad siempre ateniéndose las condiciones especificadas.

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

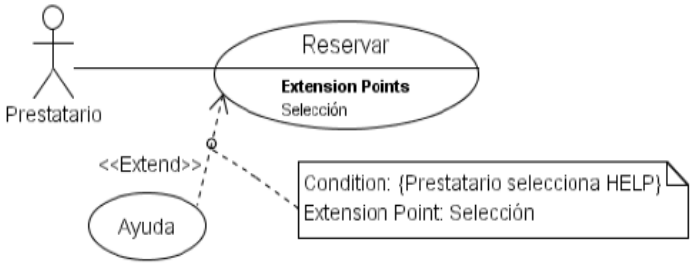
Por otra parte, es conveniente detallar la condición que provoca la ejecución de la extensión. Esto visualmente se representa como una nota conectada mediante una línea discontinua a la relación de dependencia que se refiere.

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.

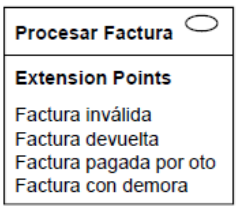


La extensión se representa mediante una flecha discontinua con la punta abierta que va desde el caso de uso que extiende al extendido, el estereotipo usado para señalizar la flecha es **<<extend>>**. Opcionalmente es posible incluir una nota con las condiciones y las referencias a los puntos de extensión.

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.



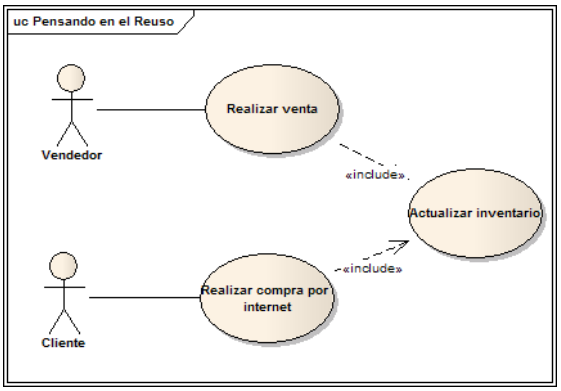
¿Cuándo usar este tipo de relaciones?:

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 7: 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 (bien como cliente o bien como restaurante).
- 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.
- Cuando se ha hecho el log in la persona no identificada pasa a ser un cliente o un restaurante.
- El cliente puede ver los restaurantes disponibles y la comida disponible en cada restaurante.
- Se puede seleccionar la comida para posteriormente pedirla.
- Cuando pida comida se activará la función que revisará si el usuario tiene fondos suficientes, generará un ticket y mandará una notificación al restaurante.
- El pago se puede hacer con tarjeta o con paypal.

- El restaurante puede ver los pedidos pendientes que tiene y aceptar o cancelar el que desee.
- En caso de cancelación se devuelve el dinero pagado al cliente y se le envía una notificación.
- En caso de aceptación se manda una notificación al cliente.

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

- Se trata de un programa para realizar debates online y por escrito entre varios miembros de un canal.
- Un miembro puede crear un debate o visualizar los debates que existen.
- De los debates existentes, se puede unir libremente a uno ya existente.
- Dentro de un debate puede visualizar los mensajes del resto, puntuarlos o escribir un mensaje.
- De los miembros, solo el creador del debate puede darlo por finalizarlo y cerrarlo para no admitir más comentarios.
- Cuando un miembro publica un mensaje el sistema comprueba que no contenga ninguna de las palabras prohibidas, y en caso de que la tenga, le bloquea la cuenta.

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

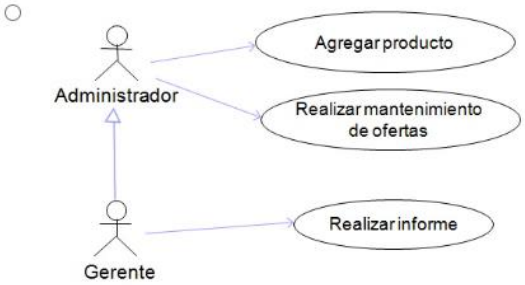
- Una empresa de transporte de mercancías quiere hacer una supervisión de todos los camiones de los que dispone. Para eso ha diseñado una nueva aplicación web llamada Güeraryu.
- En esta aplicación un administrador registra unos GPS especiales con 4G y les asigna un nombre. Básicamente el nombre es la 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).
- 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.

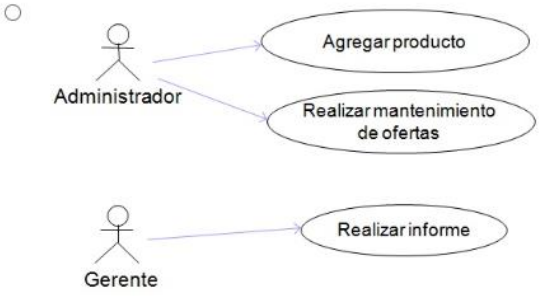
EJERCICIO 10:

1- Dada esta parte del enunciado, ¿cuál es el diseño más pertinente?

... "El administrador del sistema es el encargado de agregar productos a Amazing.com así como también de mantener las ofertas del sitio" ...

... "El gerente puede agregar productos y mantener ofertas, además de realizar informes de interés para los directivos y la empresa"...

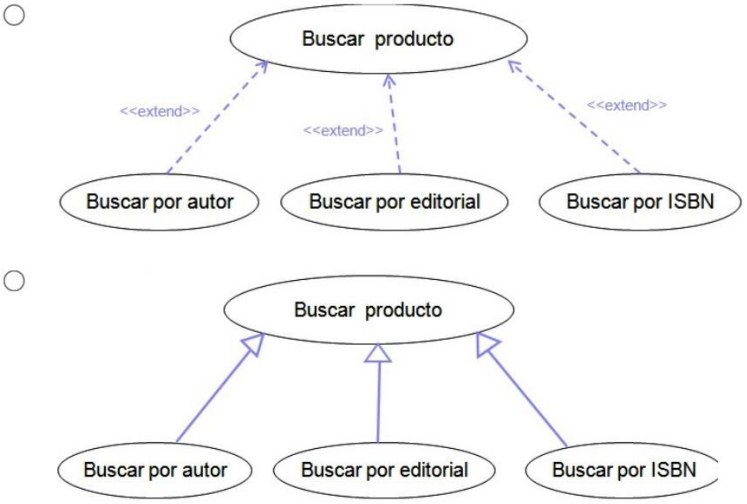




(Región de Murcia, s.f.)

EJERCICIO 11:

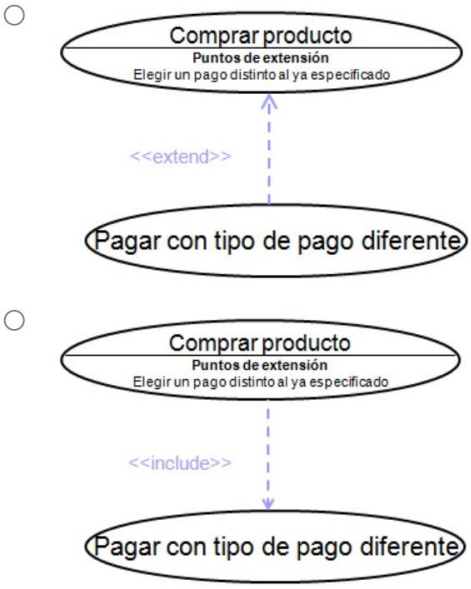
2- ¿Cuál es el diseño correcto para modelar los distintos tipos de búsqueda?



(Región de Murcia, s.f.)

EJERCICIO 12:

3- ¿Cuál es el diseño más pertinente para modelar el pago con un medio o datos diferentes a los almacenados? ¿Utilizando "include" o "extend"?



(Región de Murcia, s.f.)

5. REQUISITOS DE 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 13: Indica los requisitos de información que pueden darse en un sistema software que gestione un videoclub.

EJERCICIO 14: Indica los requisitos de información que pueden darse en un sistema software que gestione un hospital de mascotas.

EJERCICIO 15: Indica los requisitos de información que pueden darse en un sistema software que gestione las competiciones nacionales correspondientes a un deporte.

6. 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 16: 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 17: Extrae los requisitos no funcionales que ha de tener una la app del banco Santander.

EJERCICIO 18: 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

7. 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		

8. MATRIZ DE RASTREABILIDAD

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).

8.1. Clases

Se trata de la descripción que se asigna a una plantilla a partir de la cual se puede generar objetos similares.

Las clases contienen las características comunes a dichos objetos (métodos, atributos, restricciones, etc.)

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 19: Pensad en una clase “Alumno”, usada para la gestión de un centro. ¿Qué atributos tendrá?

Un objeto es una instancia de una clase.

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.

8.1.1. Elementos

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.

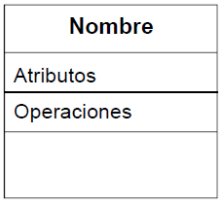
- **Operaciones o métodos:** Definen las acciones que pueden realizar los objetos que se instancien a partir de la clase. Se elaboran a partir de un conjunto de instrucciones.

EJERCICIO 20: Pensad en una clase "Tweet", usada en la aplicación "Twitter". ¿Qué atributos tendrá? ¿Qué operaciones estarán definidas?

EJERCICIO 21: 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?

8.1.2. 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}**

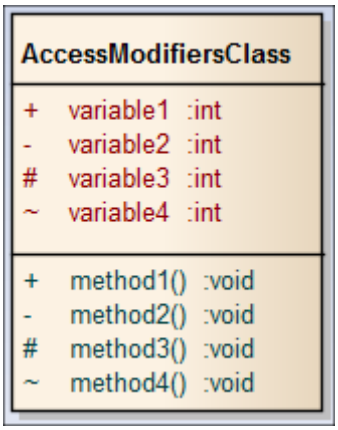
- **Visibilidad:** Expresa si los atributos son visibles en otras clases.

+ Visibilidad Pública

Visibilidad Protegida

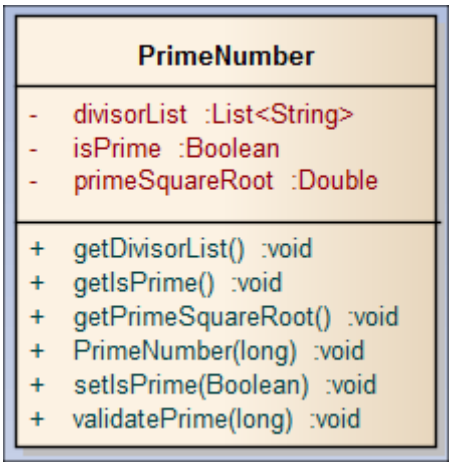
- Visibilidad Privada

~ 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 acciones.
Un ejemplo sería: "*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}

- **Dirección**: 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 22: 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.

8.2. Visibilidad

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

8.2.1. 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*

8.2.2. 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

8.2.3. 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, preferentemente, 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).

8.3. 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.

8.4. Relaciones

Una relación es una conexión semántica entre elementos de un modelo.

La instanciación de una relación o asociación es un enlace-

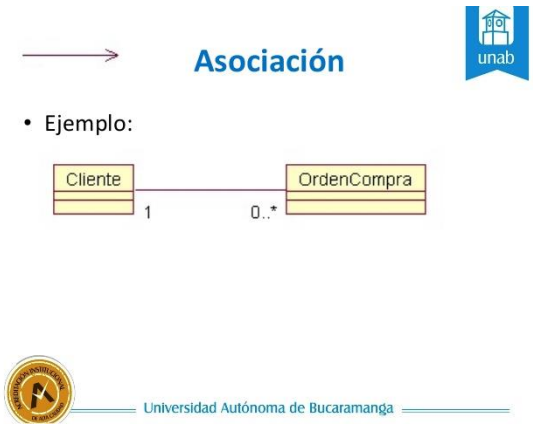
Tipos de relaciones:

- **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.

8.4.1. Asociación

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

8.4.1.1. Elementos

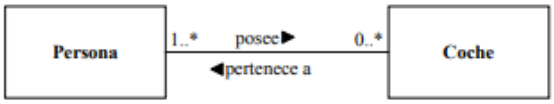


Las asociaciones suelen ser bidireccionales (que aplica en ambos sentidos).

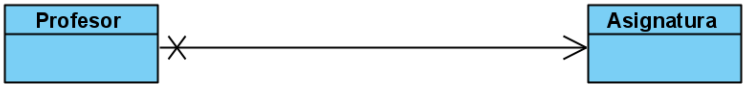
Se suelen representar mediante una línea horizontal que conecta dos objetos, no obstante, en función de una serie de características esto puede cambiar (lo veremos más adelante).

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

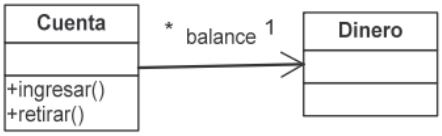
En el centro de la línea de asociación se puede escribir el nombre de la relación, al cual se puede añadir un triángulo sólido que indicará el orden de lectura.



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.



En este caso, desde un objeto asignatura, no se puede acceder al objeto profesor que la imparte.




```
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.

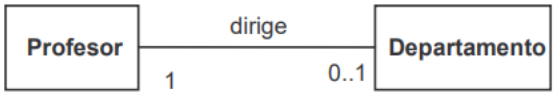
Cada asociación tiene dos multiplicidades (una para cada extremo de la relación) y dentro de cada una de ellas hay que indicar la multiplicidad mínima y la multiplicidad máxima.

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)

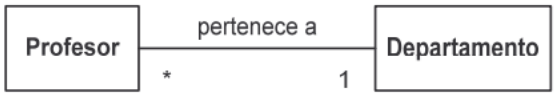
Cuando la multiplicidad mínima es 0 se dice que la relación es opcional, es decir, puede ser o no ser.

En cambio, cuando la multiplicidad mínima mayor o igual a 1, se dice que la relación es obligatoria.

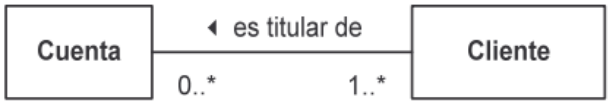
Todo departamento tiene un director. Un profesor puede dirigir un departamento:



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

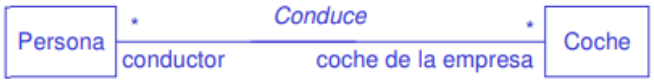


Un cliente puede ser titular de una o varias cuentas o de ninguna (Relación opcional). Una cuenta ha de tener un titular como mínimo (Relación opcional)



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.

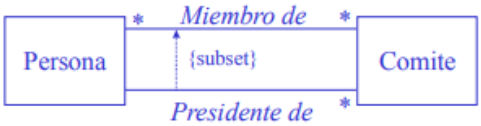


- **Visibilidad:** Igual que en las clases.

- **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.

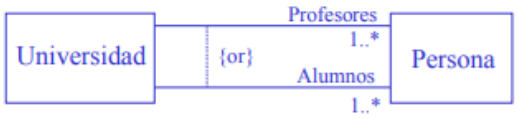
    - **{subset}**: Se utiliza para indicar que la asociación es un subconjunto de otra.



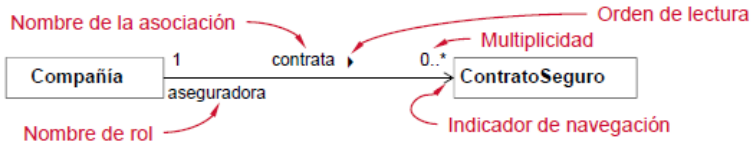
    - **{or}:** Se utiliza para indicar que los objetos de una clase solo pueden participar en una de las asociaciones pertenecientes a un grupo de estas.

EJERCICIO 23: Explica detalladamente los siguientes diagramas.

a)



b)



c)

d)

EJERCICIO 24: 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.

8.4.1.2. 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.

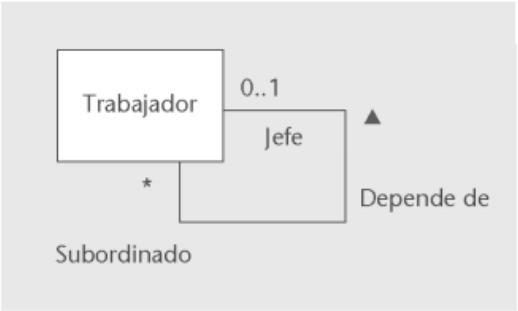
Las empresas han de contar con, al menos, un empleado.

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.

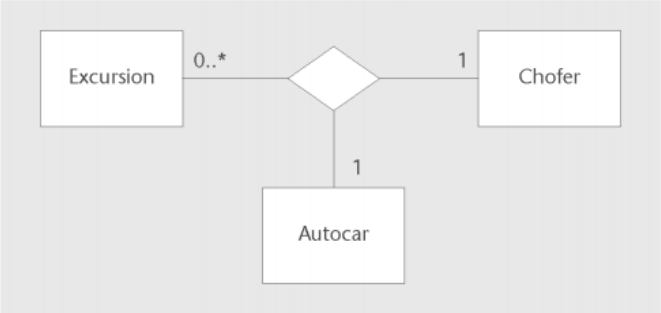


En el ejemplo anterior cada trabajador depende de un jefe. Tanto el jefe como el subordinado son trabajadores.

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

- **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.



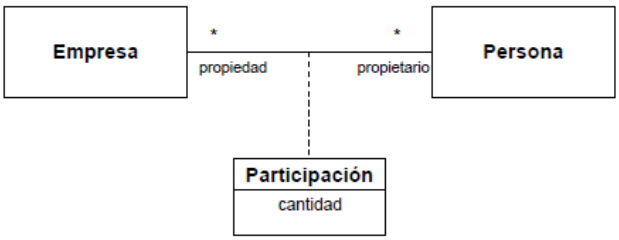
Un chofer determinado puede conducir un autocar determinado en cualquier número de excursiones, pero en una excursión concreta un chófer solo puede conducir un autocar y un autocar solo puede tener un chofer.

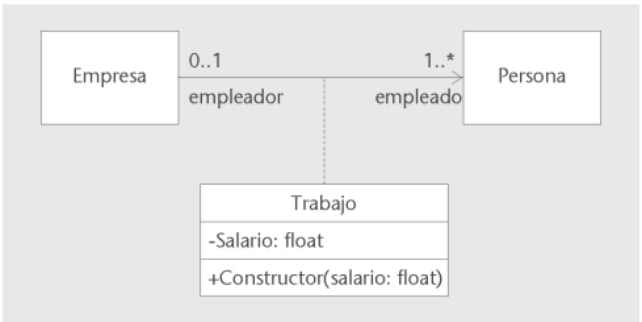
EJERCICIO 25: Cada aula alberga uno o más grupos a los que se imparten una o más asignaturas, a su vez cada grupo tiene asignada una o más aulas en donde recibe docencia de una o más asignaturas, y además cada asignatura se imparte en una o más aulas a uno o más grupos.

- **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 26: Planteamos un problema que se corresponde con la informatización de una página web de venta de entradas para conciertos, en la cual se contempla una clase cliente y una clase entrada.

Los clientes tienen un nombre y un DNI.

Las entradas tienen un nombre, una descripción y un precio.

Es necesario registrar las compras que realizan los clientes en el sistema, por tanto, se registrará la fecha de la misma, el número de entradas compradas y el dinero gastado.

- **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:

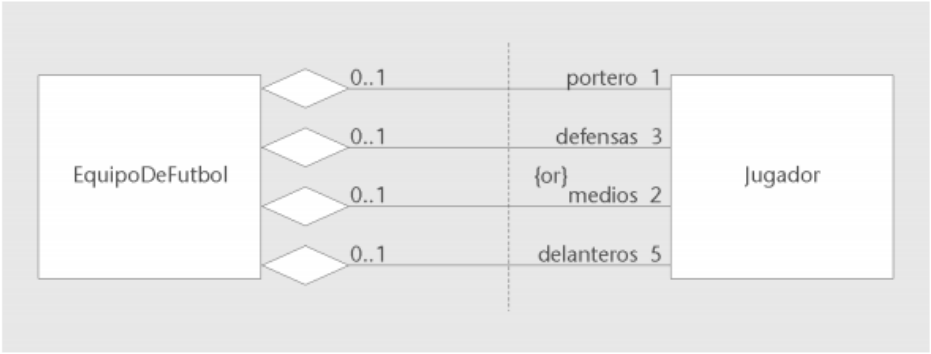
  - **Acoplamiento-piezas**: Una máquina y sus piezas, un circuito y sus componentes, un sistema software y sus programas, etc. Cada parte tiene un papel concreto y no se puede sustituir.

  - **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.

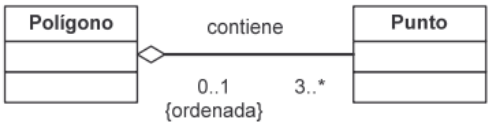


En el ejemplo mostrado anteriormente se representa una composición un poco pintoresca de un equipo de fútbol.

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 27: 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?

EJERCICIO 28: Vamos a replantear el ejercicio 24 con los nuevos conocimientos que tenemos.

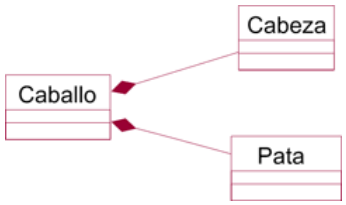
- **Composición:**

  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.

  Por ejemplo, en un hotel, las habitaciones forman parte del mismo y, una habitación por si misma sin el concepto del hotel no tendría sentido.

  Se representan con un rombo en uno de los extremos de la asociación, pero en este caso el rombo está coloreado.



EJERCICIO 29: La gran cadena de hoteles MH ha decidido crear un software para la gestión de sus propiedades.

Se ha de indicar el nombre de la ciudad en la que están los hoteles, los habitantes que tiene, el número de hoteles que existen en la ciudad y la proporción habitantes/hoteles, que se calculará a partir de los parámetros existentes.

Se han de especificar los hoteles de la marca MH, indicando la calle en la que están. En una misma ciudad puede haber varios hoteles.

Las habitaciones de cada hotel, indicando su precio y su tamaño.

Los empleados de la empresa, teniendo en cuenta que un empleado puede trabajar tan solo en un hotel. Se ha de especificar su nombre, su DNI, y su sueldo.

Se pretende que parte de este sistema salga a internet de cara al público, permitiéndole reservar habitaciones de los hoteles de la marca en tiempo real.

De esta manera el sistema registrará clientes, de los cuales es conveniente conocer el nombre y el DNI.

El sistema permitirá a los clientes reservar habitaciones y deberá contener el precio de la reserva y la fecha.

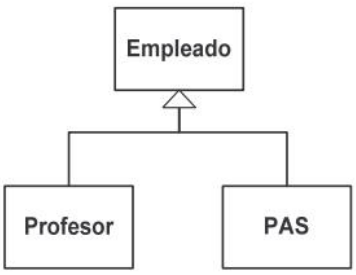
8.4.2. Generalización

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

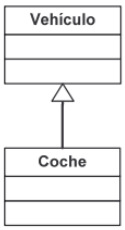
En este contexto, el concepto de generalización tiene su traducción directa con el de herencia.

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.

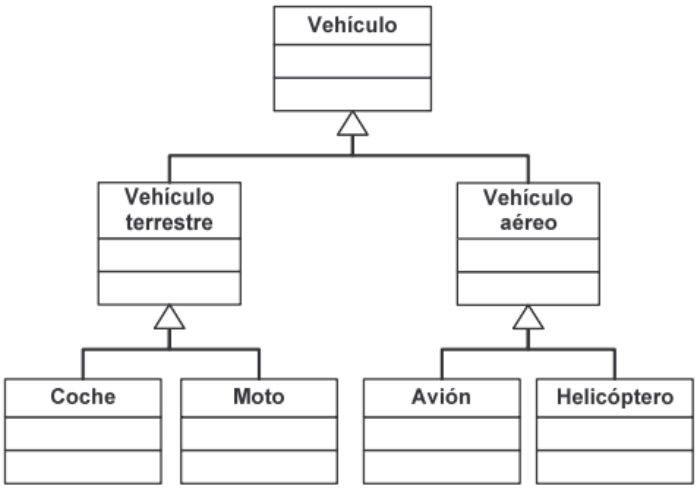
La notación usada en UML para la generalización consiste en una flecha que conecta las subclases con la superclase, yendo la flecha de las subclases a la superclase.



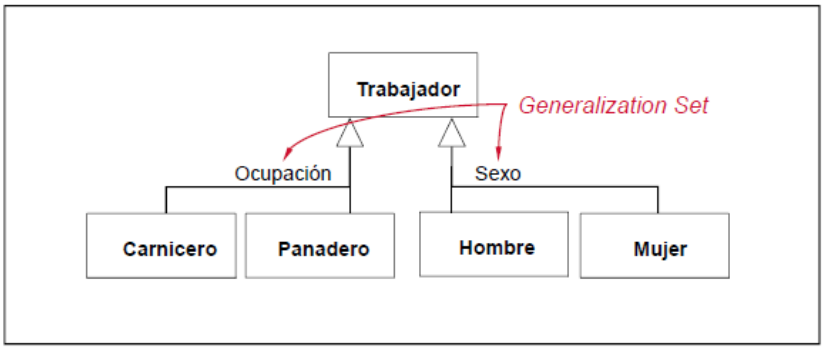

```
public class Empleado
{
   ...
}

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.



EJERCICIO 29 (Ampliación): Respecto al diagrama que hemos hecho en el ejercicio anterior, se contempla que los trabajadores pueden ser recepcionistas, botones o cocineros. De los recepcionistas hay que almacenar el número de teléfono, de los botones la planta en la que trabajan y de los cocineros su plato estrella.

8.4.3. Dependencia

La relación de dependencia es la que se genera entre un usuario, que utiliza una funcionalidad, 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.

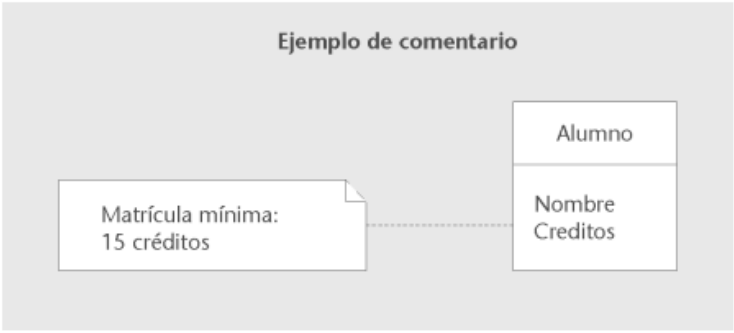
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

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

8.4.4. Comentarios

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.



8.4.5. Generación de código

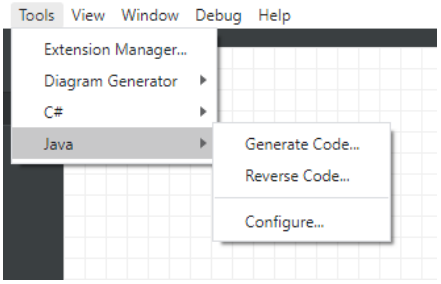
Es posible generar código en ciertos lenguajes de programación a partir de diagramas.

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.



8.4.6. Ejercicios

EJERCICIO 30: 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. Los módulos podrán ser dinámicos (capaces de moverse: rotación [tiene que almacenar el sentido de la rotación], extensión [tiene que almacenar los centímetros que puede extender], helicoidal [tiene que almacenar su velocidad máxima en revoluciones por minuto]) o estáticos (no se pueden mover: cámara [tiene que almacenar la resolución]).
- Los módulos tendrán un identificador (1-255) y unas dimensiones (largo, ancho y alto, entre 1 y 200mm).
- Los módulos dinámicos tendrán:
    - Motores (1 o 2).
    - Una función que es moverse (con un parámetro que es el tipo de movimiento a realizar, que se representa como un entero).
- Los módulos estáticos podrán tener sensores, de los que se almacenará la marca (de 0 a 5).
- Los módulos estarán compuestos de un sistema de control y un sistema de comunicación.
- Los módulos pueden enviar y recibir mensajes de/hacia el usuario y otros módulos, con un parámetro que es una lista de bytes a mandar o recibir haciendo uso del sistema de comunicación.
- El sistema de control utiliza el de comunicación para enviar mensajes a otros módulos.

- 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 31: 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 copias en préstamo.
- Cuando se realiza el préstamo de un libro se apunta la fecha en la que se realiza.
- 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.
- El lector tiene que estar dado de alta para poder alquilar un libro.

EJERCICIO 32: 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 33: 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.