**UNIDAD 05 ENTORNOS DE DESARROLLO**

**DISEÑO ORIENTADO A OBJETOS. DIAGRAMAS**

**1.- Introducción a la orientación a objetos.**

La construcción de software es un proceso cuyo objetivo es dar solución a problemas utilizando una herramienta informática y tiene como resultado la construcción de un programa informático. Como en cualquier otra disciplina en la que se obtenga un producto final de cierta complejidad, si queremos obtener un producto de calidad, es preciso realizar un proceso previo de análisis y especificación del proceso que vamos a seguir, y de los resultados que pretendemos conseguir.

**El enfoque estructurado**.

Sin embargo, cómo se hace es algo que ha ido evolucionando con el tiempo, en un principio se tomaba el problema de partida y se iba sometiendo a un proceso de división en subproblemas más pequeños reiteradas veces, hasta que se llegaba a problemas elementales que se podía resolver utilizando una función. Luego las funciones se hilaban y entretejían hasta formar una solución global al problema de partida. Era, pues, un proceso centrado en los procedimientos, se codificaban mediante funciones[[1]](#footnote-1) que actuaban sobre estructuras de datos[[2]](#footnote-2), por eso a este tipo de programación se le llama [programación estructurada](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/1_introduccin_a_la_orientacin_a_objetos.html#ta1a7d59c-b851-e5dd-30f8-3e276e692799)[[3]](#footnote-3). Sigue una filosofía en la que se intenta aproximar qué hay que hacer, para así resolver un problema.

**Enfoque orientado a objetos**.

La orientación a objetos ha roto con esta forma de hacer las cosas. Con este nuevo [paradigma](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/1_introduccin_a_la_orientacin_a_objetos.html#te4de96a8-16fb-6367-9a2e-65b41132d00a)[[4]](#footnote-4) el proceso se centra en simular los elementos de la realidad asociada al problema de la forma más cercana posible. La abstracción[[5]](#footnote-5) que permite representar estos elementos se denomina objeto, y tiene las siguientes características:

* Está formado por un conjunto de **atributos**, que son los datos que le caracterizan y
* Un conjunto de **operaciones** que definen su comportamiento. Las operaciones asociadas a un objeto actúan sobre sus atributos para modificar su estado. Cuando se indica a un objeto que ejecute una operación determinada se dice que se le pasa un **mensaje**.

Las aplicaciones orientadas a objetos están formadas por un conjunto de objetos que interaccionan enviándose mensajes para producir resultados. Los objetos similares se abstraen en clases, se dice que un objeto es una instancia de una clase.

Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos:

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

**2.- Conceptos de Orientación a Objetos.**

Como hemos visto la orientación a objetos trata de acercarse al contexto del problema lo más posible por medio de la simulación de los elementos que intervienen en su resolución y basa su desarrollo en los siguientes conceptos:

* **Abstracción**: Permite capturar las características y comportamientos similares de un conjunto de objetos con el objetivo de darles una descripción formal. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad, o el problema que se quiere atacar.
* **Encapsulación**:[[6]](#footnote-6) Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.
* **Modularidad**: Propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. En orientación a objetos es algo consustancial, ya que los objetos se pueden considerar los módulos más básicos del sistema.
* **Principio de ocultación**:[[7]](#footnote-7) Aísla las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas. Reduce la propagación de efectos colaterales cuando se producen cambios.
* **Polimorfismo**:[[8]](#footnote-8) Consiste en reunir bajo el mismo nombre comportamientos diferentes. La selección de uno u otro depende del objeto que lo ejecute.
* **Herencia**:[[9]](#footnote-9) Relación que se establece entre objetos en los que unos utilizan las propiedades y comportamientos de otros formando una jerarquía. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
* **Recolección de basura**: Técnica por la cual el entorno de objetos se encarga de destruir automáticamente los objetos, y por tanto desvincular su memoria asociada, que hayan quedado sin ninguna referencia a ellos.

**2.1.- Ventajas de la orientación a objetos.**

Este paradigma tiene las siguientes **ventajas** con respecto a otros:

1. Permite desarrollar software en mucho menos tiempo, con menos coste y de mayor calidad gracias a la reutilización porque al ser completamente modular facilita la creación de código reusable dando la posibilidad de reutilizar parte del código para el desarrollo de una aplicación similar.
2. Se consigue aumentar la calidad de los sistemas, haciéndolos más extensibles ya que es muy sencillo aumentar o modificar la funcionalidad de la aplicación modificando las operaciones.
3. El software orientado a objetos es más**fácil de modificar y mantener** porque se basa en criterios de modularidad y encapsulación en el que el sistema se descompone en objetos con unas responsabilidades claramente especificadas e independientes del resto.
4. La tecnología de objetos facilita la adaptación al entorno y el cambio haciendo aplicaciones escalables[[10]](#footnote-10). Es sencillo modificar la estructura y el comportamiento de los objetos sin tener que cambiar la aplicación.

**2.2.- Clases, atributos y métodos.**

Los objetos de un sistema se abstraen, en función de sus características comunes, en clases. Una clase está formada por un conjunto de procedimientos y datos que resumen características similares de un conjunto de objetos. La clase tiene dos propósitos: definir **abstracciones** y favorecer la **modularidad**.

Una clase se describe por un conjunto de elementos que se denominan **miembros** y que son:

* **Nombre**. Identificativo, relacionado con lo que representa.
* **Atributos**: Conjunto de características asociadas a una clase. Pueden verse como una relación binaria entre una clase y cierto dominio formado por todos los posibles valores que puede tomar cada atributo. Cuando toman valores concretos dentro de su dominio definen el **estado** del objeto. Se definen por su nombre y su tipo, que puede ser simple o compuesto como otra clase.
* **Protocolo**: Operaciones (métodos, mensajes) que manipulan el estado. Un ***método*** es el procedimiento o función que se invoca para actuar sobre un objeto. Un ***mensaje*** es el resultado de cierta acción efectuada por un objeto. Los métodos determinan como actúan los objetos cuando reciben un mensaje, es decir, cuando se requiere que el objeto realice una acción descrita en un método se le envía un mensaje. El conjunto de mensajes a los cuales puede responder un objeto se le conoce como *protocolo del objeto*.

Por ejemplo, si tenemos un objeto icono, tendrá como atributos el tamaño, o la imagen que muestra, y su protocolo puede constar de mensajes invocados por el clic del botón de un ratón cuando el usuario pulsa sobre el icono. De esta forma los mensajes son el único conducto que conectan al objeto con el mundo exterior.

Los valores asignados a los atributos de un objeto concreto hacen a ese objeto ser **único**. La clase define sus características generales y su comportamiento.

**2.3.- Visibilidad.**

El principio de ocultación es una propiedad de la orientación a objetos que consiste en aislar el estado de manera que sólo se puede cambiar mediante las operaciones definidas en una clase. Este aislamiento protege a los datos de que sean modificados por alguien que no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones. Da lugar a que las clases se dividan en dos partes:

1. **Interfaz**: captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.
2. **Implementación**: comprende como se representa la abstracción, así como los mecanismos que conducen al comportamiento deseado.

Existen distintos niveles de ocultación que se implementan en lo que se denomina **visibilidad**. Es una característica que define el tipo de acceso que se permite a atributos y métodos y que podemos establecer como:

* **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
* **Privado**: Sólo se pueden acceder desde operaciones de la clase.
* **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.

Como norma general a la hora de definir la visibilidad tendremos en cuenta que:

* El estado debe ser privado. Los atributos de una clase se deben modificar mediante métodos de la clase creados a tal efecto.
* Las operaciones que definen la funcionalidad de la clase deben ser públicas.
* 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).

**2.4.- Objetos. Instanciación.**

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

Cada vez que se construye un objeto en un programa informático a partir de una clase se crea lo que se conoce como instancia[[11]](#footnote-11) de esa clase. Cada instancia en el sistema sirve como modelo de un objeto del contexto del problema relevante para su solución, que puede realizar un trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema, sin revelar cómo se implementan estas características.

Un objeto se define por:

* **Su estado**: es la concreción de los atributos definidos en la clase a un valor concreto.
* **Su comportamiento**: definido por los métodos públicos de su clase.
* **Su tiempo de vida**: intervalo de tiempo a lo largo del programa en el que el objeto existe. Comienza con su creación a través del mecanismo de **instanciación** y finaliza cuando el objeto se destruye.

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

Ejemplo de objetos:

1. **Objetos físicos**: aviones en un sistema de control de tráfico aéreo, casas, parques.
2. **Elementos de interfaces gráficas de usuario**: ventanas, menús, teclado, cuadros de diálogo.
3. **Animales**: animales vertebrados, animales invertebrados.
4. **Tipos de datos definidos por el usuario**: Datos complejos, Puntos de un sistema de coordenadas.
5. **Alimentos**: carnes, frutas, verduras.

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

**3.- UML.**

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

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

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

**¿Por qué es útil modelar?**

* Porque permite utilizar un lenguaje común que facilita la comunicación entre el equipo de desarrollo.
* Con UML podemos documentar todos los artefactos[[12]](#footnote-12) de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones…) por lo que se dispone de documentación que trasciende al proyecto.
* Hay estructuras que trascienden lo representable en un lenguaje de programación, como las que hacen referencia a la arquitectura del sistema,[[13]](#footnote-13) utilizando estas tecnologías podemos incluso indicar qué módulos de software vamos a desarrollar y sus relaciones, o en qué nodos hardware se ejecutarán cuando trabajamos con sistemas distribuidos.
* Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose modelos precisos, no ambiguos y completos.

Además, UML puede conectarse a lenguajes de programación mediante ingeniería directa[[14]](#footnote-14) e inversa[[15]](#footnote-15), como veremos.

**3.1.- Tipos de 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).

Un diagrama UML se compone 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.

y se clasifican en:

* **Diagramas estructurales**: Representan la visión estática del sistema. Especifican clases y objetos y como se distribuyen físicamente en el sistema.
* **Diagramas de comportamiento**: muestran la conducta en tiempo de ejecución del sistema, tanto desde el punto de vista del sistema completo como de las instancias u objetos que lo integran. Dentro de este grupo están los diagramas de interacción.

En la imagen aparecen todos los diagramas organizados según su categoría. Se han destacado aquellos que perteneces al estándar UML 2.0, más novedosos. En total se describen trece 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.

Diagrama

Descripción generada automáticamente con confianza media

**3.2.- Herramientas para la elaboración de diagramas UML.**

La herramienta más simple que se puede utilizar para generar diagramas es lápiz y papel, hoy día, sin embargo, podemos acceder a [herramientas CASE](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/32_herramientas_para_la_elaboracin_de_diagramas_uml.html#tb286ede0-4fee-8eca-2b56-b74f2df020bb) que facilitan en gran manera el desarrollo de los diagramas UML. Estas herramientas suelen contar con un entorno de ventanas tipo wysiwyg, permiten documentar los diagramas e integrarse con otros entornos de desarrollo incluyendo la generación automática de código y procedimientos de ingeniería inversa.

Podemos encontrar, entre otras, las siguientes herramientas:

* **Rational Systems Developer de IBM**: Herramienta propietaria que permite el desarrollo de proyectos software basados en la metodología UML. Desarrollada en origen por los creadores de UML fue absorbida por IBM. Ofrece versiones de prueba, y software libre para el desarrollo de diagramas UML. Actualmente esta herramienta está obsoleta, IBM no ofrece soporte desde 2010, tiene una herramienta nueva**IBM Rational Rhapsody Developer.**
* Visual Paradigm for UML (VP-UML): Incluye una versión para uso no comercial que se distribuye libremente sin más que registrarse para obtener un archivo de licencia. Incluye diferentes módulos para realizar desarrollo UML, diseñar bases de datos, realizar actividades de ingeniería inversa y diseñar con Agile. Es compatible con los IDE de Eclipse, Visual Studio .net, IntellijDEA y NetBeans. [Multiplataforma](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/32_herramientas_para_la_elaboracin_de_diagramas_uml.html#t68f17394-83c7-b3e7-e70c-380e34be7aa5), incluye instaladores para Windows y Linux.
* **ArgoUML**: se distribuye bajo licencia Eclipse. Soporta los diagramas de UML 1.4, y genera código para java y C++. Para poder ejecutarlo se necesita la plataforma java. Admite ingeniería directa e inversa.

**3.2.1.- Visual Paradigm.**

Para realizar el ejemplo de desarrollo de diagramas de clases que veremos a continuación se ha determinado usar la herramienta Visual Paradigm for UML por los siguientes motivos:

* Incluye una versión para uso no comercial, aunque se debe aclarar que viene con funcionalidad limitada, que se distribuye bajo licencia LGPL. Es posible solicitar una licencia de prueba para treinta días que utilizaremos cuando veamos la parte de ingeniería directa e inversa y generación de código.
* Es multiplataforma.
* Compatible con UML 2.0.
* Admite la generación de informes en formatos PDF, HTML y otros.
* Incluye un módulo para integrarse con NetBeans.
* Permite realizar actividades de ingeniería inversa y directa. Esto junto con la consideración anterior permite generar código en un proyecto NetBeans directamente a partir del diseño de clases, ahorrándonos trabajo.

**4.- Diagramas estructurales**

Dentro de los diagramas estructurales, podemos destacar los siguientes tipos:

* **Diagramas de clases**: Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones. Tiene una prioridad ALTA.
* **Diagrama de objetos**: Muestra los elementos del modelo estático en un momento concreto, habitualmente en casos especiales de un diagrama de clases o de comunicaciones, y está formado por un conjunto de objetos y sus relaciones. Tiene una prioridad ALTA.
* **Diagrama de componentes**: Especifican la organización lógica de la implementación de una aplicación, sistema o empresa, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos. Tiene una prioridad MEDIA.
* **Diagramas de despliegue**: Representan la configuración del sistema en tiempo de ejecución. Aparecen los nodos de procesamiento y sus componentes. Exhibe la ejecución de la arquitectura del sistema. Incluye nodos, ambientes operativos sea de hardware o software, así como las interfaces que las conectan, es decir, muestra como los componentes de un sistema se distribuyen entre los ordenadores que los ejecutan. Se utiliza cuando tenemos sistemas distribuidos. Tiene una prioridad MEDIA.
* **Diagrama integrado de estructura** (UML 2.0): Muestra la estructura interna de una clasificación (tales como una clase, componente o caso típico), e incluye los puntos de interacción de esta clasificación con otras partes del sistema. Tiene una prioridad BAJA.
* **Diagrama de paquetes**: Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. Suele ser útil para la gestión de sistemas de mediano o gran tamaño. Tiene una prioridad BAJA.

**4.1.- Diagramas de clases.**

Dentro de los diagramas estructurales, y de todos en general, es el más importante porque representa los elementos estáticos del sistema, sus atributos y comportamientos, y como se relacionan entre ellos. Contiene las clases del dominio del problema[[16]](#footnote-16), y a partir de éste se obtendrán las clases que formarán después el programa informático que dará solución al problema.

En un diagrama de clases podemos encontrar los siguientes elementos:

* **Clases**: recordemos que son abstracciones del dominio del sistema que representan elementos del mismo mediante una serie de características, que llamaremos atributos, y su comportamiento, que serán métodos. Los atributos y métodos tendrán una visibilidad que determinará quien puede acceder al atributo o método. Por ejemplo, una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará.
* **Relaciones**: en el diagrama representan relaciones reales entre los elementos del sistema a los que hacen referencia las clases. Pueden ser de asociación, agregación y herencia. Por ejemplo, si tengo una clase persona, puedo establecer una relación conduce entre persona y coche.
* **Notas**: Se representan como un cuadro donde podemos escribir comentarios que nos ayuden a entender algún concepto que queramos representar.
* **Elementos de agrupación**: Se utilizan cuando hay que modelar un sistema grande, entonces las clases y sus relaciones se agrupan en paquetes[[17]](#footnote-17), que a su vez se relacionan entre sí.

Ejercicio resuelto

**Crear un diagrama de clases nuevo en Visual Paradigm UML que incluya su nombre y su descripción.**

Para crear un diagrama de clases en VP-UML seleccionamos Archivo >> Nuevo Diagrama y seleccionamos Diagrama de clases. También podemos acceder al Navegador de diagramas, que se encuentra en el panel de la izquierda y en diagramas de clases hacer clic con el botón secundario y Seleccionar Nuevo diagrama de clases. Cuando generamos un diagrama nuevo tenemos que indicar su nombre y una descripción. Esto es importante para la generación de la documentación posterior.

Cuando creamos un diagrama nuevo aparece en blanco en el panel central de la aplicación. Si es necesario cambiar sus propiedades podemos hacerlo seleccionándolo en el Navegador de diagramas, a través de la opción "Abrir <nombre> Specification" del menú contextual. También podemos abrir el menú contextual, haciendo clic con el botón derecho del ratón sobre el panel central de la aplicación.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**4.1.1.- Creación de clases.**

Una clase se representa en el diagrama como un rectángulo divido en tres filas, arriba aparece el nombre de la clase, a continuación, los atributos con su visibilidad y después los métodos con su visibilidad que está representada por el signo menos "–" para los atributos (privados) y por el signo más "+" para los métodos (públicos).

Ejercicio resuelto

**Crear una clase nueva en el diagrama de clases del punto anterior.**

Tabla

Descripción generada automáticamenteCuando generamos un diagrama nuevo aparece area de trabajo donde podemos añadir elementos. Si hacemos clic con el botón derecho podemos crear una nueva clase a partir de la opción "Añadir Forma". Posteriormente, cuando la clase esté creada si hacemos clic con el botón secundario sobre la clase y seleccionamos "Abrir Especificación" podremos añadir atributos y métodos.

**4.1.2.- Atributos.**

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

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

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

* **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
* **Privado**: Sólo se pueden acceder desde operaciones de la clase.
* **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
* **Paquete**: Se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación es Java.

Ejercicio resuelto

**Crear una clase de nombre "Módulo" y que tenga tres atributos:**

**Nombre, de tipo string.**

**Duración de tipo Int.**

**Contenidos de tipo string.**

Crear la clase como hemos visto en el punto anterior y modificar su nombre a "Módulo". Para añadir un atributo a una clase basta con seleccionar Añadir atributo del menú contextual y escribir su nombre. Si queremos añadir más información podemos hacerlo desde la especificación de la clase en la pestaña Atributos, en la imagen vemos la especificación de una clase llamada Módulo, y de su atributo Contenidos para el que se ha establecido su tipo (string) y su descripción. Por defecto la visibilidad de los atributos es privado y no se cambia a menos que sea necesario.

Tenemos la posibilidad de añadir, desde el menú contextual de la clase, con el atributo seleccionado dos métodos llamados getter y setter que se utilizan para leer y establecer el valor del atributo cuando el atributo no es calculado, con la creación de estos métodos se contribuye al encapsulamiento y la ocultación de los atributos.

**4.1.3.- Métodos.**

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

Existen un caso particular de método, el **constructor** de la clase, que tiene como característica que no devuelve ningún valor. El constructor tiene el mismo nombre de la clase y se usa para ejecutar las acciones necesarias cuando se instancia un objeto de la clase. Cunado haya que destruir el objeto se podrá utilizar una función para ejecutar las operaciones necesarias cuando el objeto deje de existir, que dependerán del lenguaje que se utilice.

Ejercicio resuelto

**Añadir a la clase creada anteriormente los métodos:**

**matricular(alumno : Alumno) : void**

**asignarDuración(duracion : int) : void**

El método más directo para crear un método es en el menú contextual seleccionar "Añadir operación" y escribir la signatura del método:

+nombre(<lista\_parámetros>) : tipo\_devuelto

También se puede añadir desde la especificación de la clase en la pestaña Operations.

En la imagen vemos la especificación de la clase y del método "asignarDuración" que asigna el número de horas del módulo.

El signo + en la signatura del método indica que es público. Así queda la clase en el diagrama:

**4.2.- Relaciones entre clases.**

Una **relación** es una conexión entre dos clases que incluimos en el diagrama cuando aparece algún tipo de relación entre ellas en el dominio del problema.

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

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

* De herencia.
* De composición.
* De agregación.

Ejercicio resuelto

**Crea una clase nueva llamada Alumno y establece una relación de asociación con el nombre “matrícula” entre ésta y la clase Módulo.**

Creamos la clase como hemos visto en puntos anteriores. Para crear una relación utilizamos el elemento asociación de la paleta o bien el icono Association >> Class del menú contextual de la clase. Otra forma consiste en hacer clic sobre la clase Alumno, seleccionar Association >> Class y estirar la linea hasta la clase Módulo, aparecerá un recuadro para nombrar la relación.

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

**4.2.1.- Cardinalidad o multiplicidad de la relación**

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

**Significado de las cardinalidades:**

Tabla

Descripción generada automáticamente

Por ejemplo, si tengo la siguiente relación:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

quiere decir que los alumnos se matriculan en los módulos, en concreto, que un alumno se puede matricular en uno a más módulos y que un módulo puede tener ningún alumno, uno o varios.

O esta otra:

Diagrama

Descripción generada automáticamente con confianza media

en la que un profesor puede impartir uno o varios módulos, mientras que un módulo es impartido sólo por un profesor.

**Ejercicio resuelto**

**Establece la cardinalidad de la relación que has creado en el punto anterior para indicar que un alumno debe estar matriculado en al menos un módulo, o varios y que para cada módulo se puede tener ningún alumno, uno o varios.**

Si queremos establecer la cardinalidad abrimos la especificación de la relación y establecemos el apartado Multiplicidad a alguno de los valores que indica, si necesitamos utilizar algún valor concreto también podemos escribirlo nosotros mismos. En el caso que nos ocupa seleccionaremos la cardinalidad 0..\* para los alumnos y 1..\* para los módulos.

**4.2.2.- Relación de herencia.**

La **herencia** es una propiedad que permite a los objetos ser construidos a partir de otros objetos, es decir, la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados.

El objetivo principal de la herencia es la reutilización, poder utilizar código desarrollado con anterioridad. La herencia supone una [clase base](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/422_relacin_de_herencia.html#t617502f1-6a44-3882-e418-bc95e80d7fd1)[[18]](#footnote-18) y una jerarquía de clases que contiene las [clases derivadas](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/422_relacin_de_herencia.html#tc376fd22-04bd-ee01-94ef-24d62a9f1815)[[19]](#footnote-19). Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes, es por esto que los atributos, métodos y relaciones de una clase se muestran en el nivel más alto de la jerarquía en el que son aplicables.

**Tipos**:

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

**Representación**:

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

Ejercicio resuelto

**En nuestro diagrama tenemos Alumnos y Profesores. Aún no hemos hablado de su definición y estructura, pero en nuestro sistema tanto un alumno como un profesor tienen unas características comunes como el nombre, la fecha de nacimiento o el correo electrónico por el hecho de ser personas:**

**Transforma este diagrama para hacer uso de la herencia añadiendo una clase "Persona".**

Podemos utilizar la relación de herencia para crear una clase nueva que se llame Persona y que recoja las características comunes de profesor y alumno. Persona será la clase base y Profesor y Alumno las clases derivadas.

Como los atributos Nombre, FechaNacimiento y correoElectronico se heredan de la clase base no hace falta que aparezcan en las clases derivadas, por lo que las hemos eliminado. Después podemos añadir atributos o métodos propios a las clases derivadas. La relación se añade de igual manera que una relación de asociación, pero seleccionando la opción Generalization.

**4.2.3.- Agregación y composición.**

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

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

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

Diagrama

Descripción generada automáticamente

Texto

Descripción generada automáticamente

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

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

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

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**4.2.4.- Atributos de enlace.**

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

Ejercicio resuelto

Cuando un alumno se matricula de un módulo es preciso especificar el curso al que pertenece la matrícula, las notas obtenidas en el examen y la tarea y la calificación final obtenida. Estas características no pertenecen totalmente al alumno ni al módulo sino a la relación específica que se crea entre ellos, que además será diferente si cambia el alumno o el módulo. Añade estos atributos al enlace entre Alumno y Módulo.

Para modelar esto en Visual Paradigm creamos una clase nueva (Matrícula) junto a Alumno y Módulo, y la unimos a la relación utilizando el icono que rodea a la propia clase "Association class", el diagrama queda así:

Diagrama

Descripción generada automáticamente

**4.3.- Paso de los requisitos de un sistema al diagrama de clases.**

Empezamos identificando objetos que serán las clases del diagrama examinando el planteamiento del problema. Los objetos se determinan subrayando cada nombre o cláusula nominal e introduciéndola en una tabla simple. Los sinónimos deben destacarse. Pero, ¿qué debemos buscar una vez que se han aislado todos los nombres? Buscamos sustantivos que puedan corresponder con las siguientes categorías:

* **Entidades externas** (por ejemplo: otros sistemas, dispositivos, personas) que producen o consumen información a usar por un sistema computacional.
* **Cosas** (por ejemplo: informes, presentaciones, cartas, señales) que son parte del dominio de información del problema.
* **Ocurrencias o sucesos** (por ejemplo: una transferencia de propiedad o la terminación de una serie de movimientos en un robot) que ocurren dentro del contexto de una operación del sistema.
* **Papeles o roles** (por ejemplo: director, ingeniero, vendedor) desempeñados por personas que interactúan con el sistema.
* Unidades organizacionales (por ejemplo: división, grupo, equipo) que son relevantes en una aplicación.
* **Lugares** (por ejemplo: planta de producción o muelle de carga) que establecen el contexto del problema y la función general del sistema.
* **Estructuras** (por ejemplo: sensores, vehículos de cuatro ruedas o computadoras) que definen una clase de objetos o, en casos extremos, clases relacionadas de objetos.

Cuando estemos realizando este proceso debemos estar pendientes de no incluir en la lista cosas que no sean objetos, como operaciones aplicadas a otro objeto, por ejemplo, "inversión de imagen" producirá un objeto en el ámbito del problema, pero en la implementación dará origen a un método. También es posible detectar dentro de los sustantivos **atributos** de objetos, cosa que también indicaremos en la tabla.

Cuando tengamos la lista completa habrá que estudiar cada objeto potencial para ver si, finalmente, es incluido en el diagrama. Para ayudarnos a decidir podemos utilizar los siguientes criterios:

1. La información del objeto es necesaria para que el sistema funcione.
2. El objeto posee un **conjunto de atributos** que podemos encontrar en cualquier ocurrencia del objeto. Si sólo aparece un atributo normalmente se rechazará y será añadido como atributo de otro objeto.
3. El objeto tiene un **conjunto de operaciones** identificables que pueden cambiar el valor de sus atributos y son comunes a cualquier ocurrencia del objeto.
4. Es una entidad externa que consume o produce información esencial para la producción de cualquier solución en el sistema.

El objeto se incluye si cumple todos (o casi todos) los criterios.

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

Descripción generada automáticamente

**4.3.1.- Obtención de atributos y operaciones.**

**Atributos**

Definen al objeto en el contexto del sistema, es decir, el mismo objeto en sistemas diferentes tendría diferentes atributos, por lo que debemos buscar en el enunciado o en nuestro propio conocimiento, características que tengan sentido para el objeto en el contexto que se analiza. Deben contestar a la pregunta "*¿Qué elementos (compuestos y/o simples) definen completamente al objeto en el contexto del problema actual?*"

**Operaciones**

Describen el comportamiento del objeto y modifican sus características de alguna de estas formas:

* Manipulan los datos.
* Realizan algún cálculo.
* Monitorizan un objeto frente a la ocurrencia de un suceso de control.

Se obtienen analizando verbos en el enunciado del problema.

**Relaciones**

Por último habrá que estudiar de nuevo el enunciado para obtener cómo los objetos que finalmente hemos descrito se relacionan entre sí. Para facilitar el trabajo podemos buscar mensajes que se pasen entre objetos y las relaciones de composición y agregación. Las relaciones de herencia se suelen encontrar al comparar objetos semejantes entre sí, y constatar que tengan atributos y métodos comunes.

Cuando se ha realizado este procedimiento no está todo el trabajo hecho, es necesario revisar el diagrama obtenido y ver si todo cumple con las especificaciones. No obstante siempre se puede refinar el diagrama completando aspectos del ámbito del problema que no aparezcan en la descripción recurriendo a entrevistas con los clientes o a nuestros conocimientos de la materia.

**4.4.- Generación de código a partir del diagrama de clases.**

La Generación Automática de Código consiste en la creación utilizando herramientas CASE de código fuente de manera automatizada. El proceso pasa por establecer una correspondencia entre los elementos formales de los diagramas y las estructuras de un lenguaje de programación concreto. El diagrama de clases es un buen punto de partida porque permite una traducción bastante directa de las clases representadas gráficamente, a clases escritas en un lenguaje de programación específico como Java o C++.

Normalmente las herramientas de generación de diagramas UML incluyen la facilidad de la generación, o actualización automática de código fuente, a partir de los diagramas creados.

Ejercicio resuelto

**Traduce el diagrama de clases generado en el punto anterior, tanto desde el SDE para NetBeans de Visual Paradigm como desde VP-UML.**

**Utilizando la integración de VP-UML en NetBeans:**

Antes de hacerlo tendremos que abrir el modelo desde NetBeans, usado VP integrado, crear un proyecto nuevo e importar el proyecto VP-UML que hemos creado.

Se puede hacer de dos formas:

Sincronizar con el código: El código fuente eliminado no se recuperará. Solo se actualizará el código existente.

Forzar sincronizado a código: Se actualizará todo el código que pueda partir del modelo, incluido el de código eliminado del proyecto NetBeans.

Para generar todas las clases y paquetes de un proyecto VP-UML en NetBeans abrimos el proyecto y desplegamos el menú contextual y seleccionamos Update Project to Code. También existe la posibilidad de hacerlo directamente desde una clase en particular.

Si se produce algún problema se muestra en la ventana de mensajes, una vez corregido se vuelve a actualizar.

Este procedimiento produce los archivos .java necesarios para implementar las clases del diagrama.

**Desde VP-UML**

Para generar el código java de un diagrama de clases, utilizamos el menú Herramientas >> Código. Se muestra una ventana en la que podemos configurar el lenguaje, las clases a generar, y otras características básicas relacionadas con la nomenclatura de atributos y métodos.

**4.4.1.- Elección del lenguaje de programación. Orientaciones para el lenguaje java.**

El lenguaje final de implementación de la aplicación influye en algunas decisiones a tomar cuando estamos creando el diagrama ya que el proceso de traducción es inmediato. Si existe algún problema en los nombres de clases, atributos o tipos de datos porque no puedan ser utilizados en el lenguaje final o no existan la generación dará un fallo y no se realizará. Por ejemplo, si queremos utilizar la herramienta de generación de código tendremos que asegurarnos de utilizar tipos de datos simples apropiados, es decir, si usamos Java el tipo de dato para las cadenas de caracteres será String en lugar de string o char\*.

Podemos definir el lenguaje de programación final desde el menú **Herramientas.**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente con confianza media

**4.5.- Generación de la documentación.**

Como en todos los diagramas UML, podemos hacer las anotaciones que consideremos necesarias abriendo la especificación de cualquiera de los elementos, clases o relaciones, o bien del diagrama en si mismo en la pestaña "Specification" o "Comentarios".

La ventana del editor cuenta con herramientas para formatear el texto y darle un aspecto bastante profesional, pudiendo añadir elementos como imágenes o hiperenlaces.

También se puede grabar un archivo de voz con la documentación del elemento usando el icono Grabar.

**Generar informes**

Cuando los modelos están completos podemos generar un informe en varios formatos diferentes (HTML, PDF o Word) con la documentación que hemos escrito. Para generar un informe hacemos:

Desde VP-UML accedemos a Herramientas ->Doc y seleccionamos el tipo de informe que queremos.

Desde  NetBeans seleccionamos Modelin >> Reports >> Report writer.

En ambos casos, una vez que elegimos el tipo de informe, obtendremos la siguiente ventana en la que seleccionamos entre otros:

* El resultado es un archivo (.html, .pdf o .doc) en el directorio de salida que hayamos indicado con la documentación de los diagramas seleccionados.Qué diagramas queremos que intervengan y donde se almacenará el informe.
* La pestaña opciones (Options) permite configurar los elementos que se añadirán al informe, como tablas de contenidos, títulos, etc.
* Las propiedades de la página.
* Si se va a añadir una marca de agua.

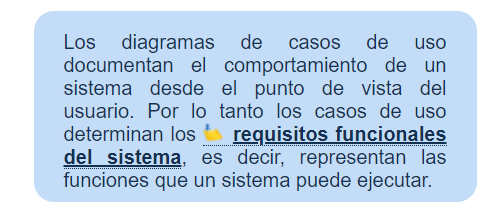
**5.- Diagramas de comportamiento.**

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

* Diagramas de casos de uso.
* Diagramas de actividad.
* Diagramas de máquinas de estado.
* Diagramas de interacción.
  + Diagramas de secuencia.
  + Diagramas de comunicación.
  + Diagramas de interacción.
  + Diagramas de tiempo.

**5.1.- Diagramas de casos de uso.**

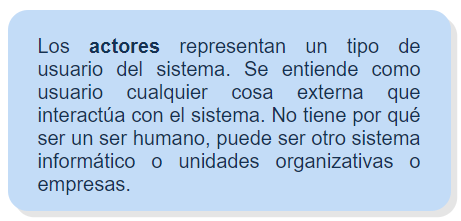
Los diagramas de casos de uso son un elemento fundamental del análisis de un sistema desde la perspectiva de la orientación a objetos porque resuelven uno de los principales problemas en los que se ve envuelto el proceso de producción de software: la falta de comunicación entre el equipo de desarrollo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



Requisitos funcionales: Un requisito es una condición que debe cumplir un proyecto software. Los requisitos funcionales definen las acciones fundamentales que debe realizar el software al recibir información, procesarla y producir resultados. Suelen venir definidos por el cliente.

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

**5.1.1.- Actores.**

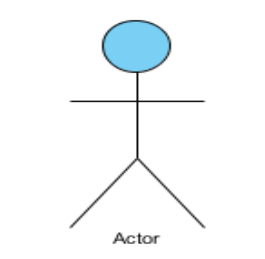


Siempre hay que intentar independizar los actores de la forma en que se interactúa con el sistema. Por ejemplo, un usuario del sistema puede interpretar diferentes roles según la operación que esté ejecutando, cada uno de estos roles representará un actor diferente, es decir, un actor en un diagrama de casos de uso representa un rol que alguien puede estar jugando, no un individuo particular por lo tanto puede haber personas particulares que puedan estar usando el sistema de formas diferentes en diferentes ocasiones. Suele ser útil mantener una lista de los usuarios reales para cada actor.

Tipos de actores:

* **Primarios**: interaccionan con el sistema para explotar su funcionalidad. Trabajan directa y frecuentemente con el software.
* **Secundarios**: soporte del sistema para que los primarios puedan trabajar. Son precisos para alcanzar algún objetivo.
* **Iniciadores**: no interactúan con el sistema pero desencadenan el trabajo de otro actor.

Los actores se representan mediante la siguiente figura:



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

**5.1.2.- Casos de uso.**

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

Un **caso de uso** especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un actor concreto. El conjunto de casos de uso forma el "**comportamiento requerido**" de un sistema.

El objetivo principal de elaborar un diagrama de casos de uso no es crear el diagrama en sí, sino la descripción que de cada caso se debe realizar, ya que esto es lo que ayuda al equipo de desarrollo a crear el sistema a posteriori. Para hacer esto utilizamos, sobre todo otros diagramas que permiten describir la dinámica del caso de uso, como el diagrama de secuencia que veremos después, y una descripción textual, en la que se deben incluir, al menos, los siguientes datos (a los que se denomina **contrato**):

* **Nombre:** nombre del caso de uso.
* **Actores**: aquellos que interactúan con el sistema a través del caso de uso.
* **Propósito**: breve descripción de lo que se espera que haga.
* **Precondiciones**: aquellas que deben cumplirse para que pueda llevarse a cabo el caso de uso.
* **Flujo normal**: flujo normal de eventos que deben cumplirse para ejecutar el caso de uso exitosamente, desde el punto de vista del actor que participa y del sistema.
* **Flujo alternativo**: flujo de eventos que se llevan a cabo cuando se producen casos inesperados o poco frecuentes. No se deben incluir aquí errores como escribir un tipo de dato incorrecto o la omisión de un parámetro necesario.
* **Postcondiciones**: las que se cumplen una vez que se ha realizado el caso de uso.

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

**5.1.3.- Relaciones.**

Los diagramas de casos de uso son [grafos no conexos](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/513_relaciones.html#t1bf25a6d-5a63-d67c-06fa-5c09572cc350)[[20]](#footnote-20) en los que los nodos son actores y casos de uso, y las aristas son las relaciones que existen entre ellos. Representan qué actores realizan las tareas descritas en los casos de uso, en concreto qué actores inician un caso de uso. Pero además existen otros tipos de relaciones que se utilizan para especificar relaciones más complejas, como uso o herencia entre casos de uso o actores.

Existen diferentes tipos de relaciones entre elementos:

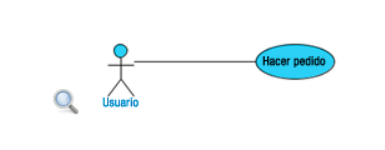
* **Asociación**: representa la relación entre el actor que lo inicia y el caso de uso.
* **Inclusión**: se utiliza cuando queremos dividir una tarea de mayor envergadura en otras más sencillas, que son utilizadas por la primera. Representa una relación de uso, y son muy útiles cuando es necesario reutilizar tareas.
* **Extensión**: se utiliza para representar relaciones entre un caso de uso que requiere la ejecución de otro en determinadas circunstancias.
* **Generalización**: se utiliza para representar relaciones de herencia entre casos de uso o actores.

A continuación las vemos con un poco más de detalle.

**5.1.3.1.- Interacción o asociación.**

Hay una asociación entre un actor y un caso de uso si el actor interactúa con el sistema para llevar a cabo el caso de uso o para iniciarlo.

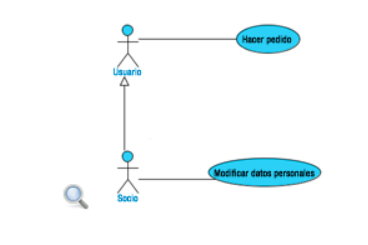
Una asociación se representa mediante un linea continua que une un actor con un caso de uso. Por ejemplo, un usuario de un sistema de venta por Internet puede hacer un pedido, lo que se representa del siguiente modo:



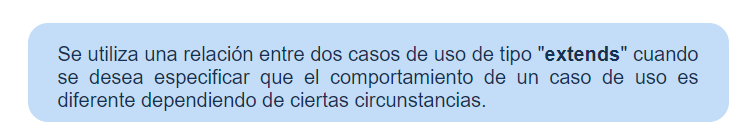
**5.1.3.2.- Generalización.**

Es posible que, igual que con los diagramas de clases, existan casos de uso que tengan comportamientos semejantes a otros que los modifican o completan de alguna manera. El caso base se define de forma abstracta y los hijos heredan sus características añadiendo sus propios pasos o modificando alguno. Normalmente la herencia se utiliza menos en diagramas de casos de uso que en diagramas de clases.

Por ejemplo, el usuario del sistema de venta por Internet puede a su vez darse de alta en la página web para que tengan sus datos registrados a la hora de hacer el pedido, en este caso el usuario es la generalización del socio. Ambos actores pueden hacer un pedido, pero solo el socio puede modificar sus datos en el sistema.

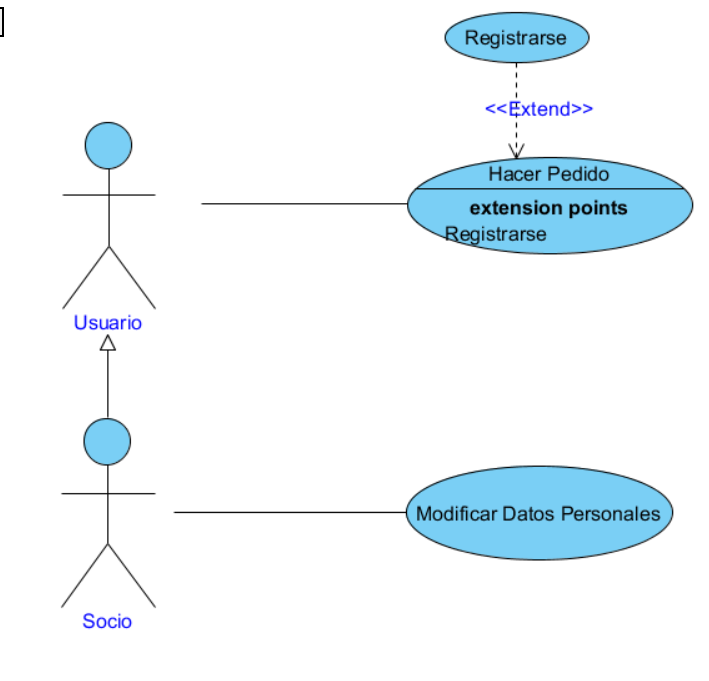


**5.1.3.3.- Extensión.**

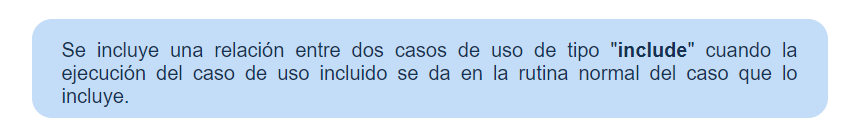


La principal función de esta relación es simplificar el flujo de casos de uso complejos. Se utiliza cuando existe una parte del caso de uso que se ejecuta sólo en determinadas ocasiones, pero no es imprescindible para su completa ejecución. Cuando un caso de uso extendido se ejecuta, se indica en la especificación del caso de uso como un **punto de extensión**. Los puntos de extensión se pueden mostrar en el diagrama de casos de uso.

Por ejemplo, cuando un usuario hace un pedido si no es socio se le ofrece la posibilidad de darse de alta en el sistema en ese momento, pero puede realizar el pedido aunque no lo sea.



**5.1.3.4.- Inclusión.**



Esta relación es muy útil cuando se desea especificar algún comportamiento común en dos o más casos de uso, aunque es frecuente cometer el error de utilizar esta técnica para hacer subdivisión de funciones, por lo que se debe tener mucho cuidado cuando se utilice.

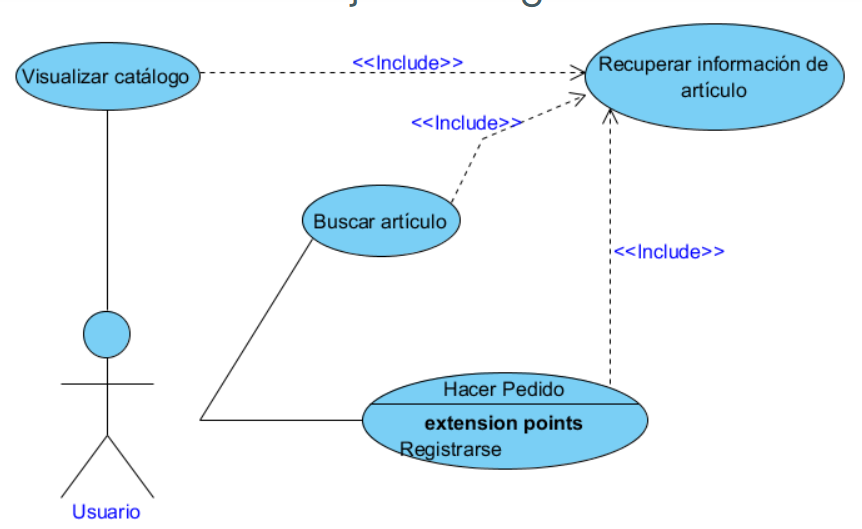
Por ejemplo, a la hora de hacer un pedido se debe buscar la información de los artículos para obtener el precio, es un proceso que necesariamente forma parte del caso de uso, sin embargo también forma parte de otros, como son el que visualiza el catálogo de productos y la búsqueda de un artículo concreto, y dado que tiene entidad por sí solo se separa del resto de casos de uso y se incluye en los otros tres.

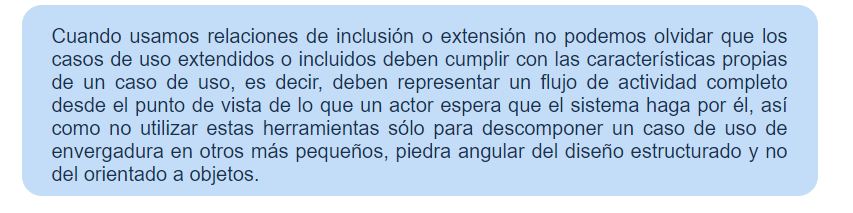
Las ventajas de esta asociación son:

* Las descripciones de los casos de uso son más cortas y se entienden mejor.
* La identificación de funcionalidad común puede ayudar a descubrir el posible uso de componentes ya existentes en la implementación.

Las desventajas son:

* La inclusión de estas relaciones hace que los diagramas sean más difíciles de leer, sobre todo para los clientes.

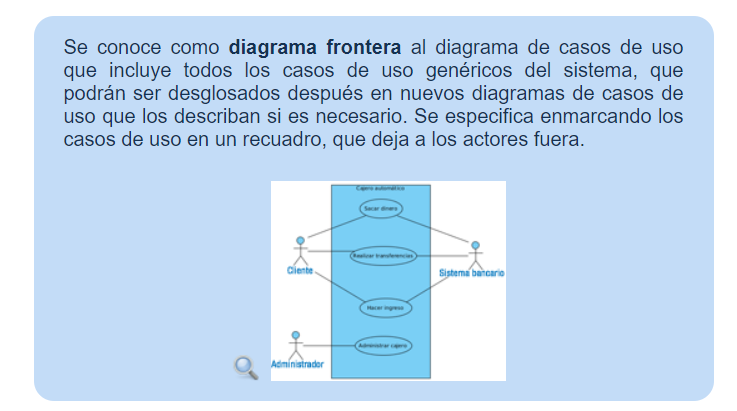




**5.1.4.- Elaboración de casos de uso.**

Sea cual sea el tipo de diagrama que estemos creando, cuando lo hacemos realizamos un proceso de abstracción por el cual representamos elementos de la realidad esquemáticamente, y en el diagrama de casos de uso pasa igual, necesitamos abstraer la realidad en un dibujo, en el que representamos qué cosas pueden hacerse en nuestro sistema y quien las va a hacer. Necesitamos diagramas que incluyan suficiente información para que el equipo de desarrollo tome las decisiones más adecuadas en la fase de análisis y diseño para una construcción de software que cumpla con los requerimientos, así como que sean útiles en la fase de implementación en un lenguaje orientado a objetos.

Partiremos de una descripción lo más detallada posible del problema a resolver y trataremos de detectar quien interactúa con el sistema, para obtener los actores diagrama de casos de uso, a continuación buscaremos qué tareas realizan estos actores para determinar los casos de uso más genéricos. El siguiente paso es refinar el diagrama analizando los casos de uso más generales para detectar casos relacionados por inclusión (se detectan fácilmente cuando aparecen en dos o más casos de uso generales), extensión y generalización. Al diagrama generado se le denomina diagrama frontera.



Finalmente revisamos toda la documentación que hemos generado.

**5.1.5.- Escenarios.**

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

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

Descripción generada automáticamente

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

Un posible escenario podría ser:

Realizar pedido de unos zapatos y unas botas.

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

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

**5.2.- Diagramas de secuencia.**

Los diagramas de secuencia se utilizan para formalizar la descripción de un escenario o conjunto de ellos representando que mensajes fluyen en el sistema así como quien los envía y quien los recibe.

Los objetos y actores que forman parte del escenario se representan mediante rectángulos distribuidos horizontalmente en la zona superior del diagrama, a los que se asocia una línea temporal vertical (una por cada actor) de las que salen, en orden, los diferentes mensajes que se pasan entre ellos. Con esto el equipo de desarrollo puede hacerse una idea de las diferentes operaciones que deben ocurrir al ejecutarse una determinada tarea y el orden en que deben realizarse.

**5.2.1.- Representación de objetos, línea de vida y paso de mensajes.**

**Representación de objetos y línea de vida.**

En un diagrama de secuencia se dibujan las entidades que participan dentro de rectángulos que se distribuyen horizontalmente. De cada rectángulo o entidad sale una línea de puntos que representa el paso del tiempo, se les denomina línea de vida y representan que el objeto existe.

Para formar el nombre de una linea de vida de un objeto se usa el nombre del objeto, que es opcional, seguido del símbolo dos puntos y a continuación el nombre de la clase a la que pertenece.

Texto, Chat o mensaje de texto

Descripción generada automáticamente

Cuando tenemos un objeto que puede tener varias instancias, aparece como un rectángulo sobre otro, como en es el caso de las líneas del pedido que pueden ser varias.

Diagrama

Descripción generada automáticamente

**Invocación de métodos.**

Los mensajes, que significan la invocación de métodos, se representan como flechas horizontales que van de una linea de vida a otra, indicando con la flecha la dirección del mensaje, los extremos de cada mensaje se conectan con una línea de vida que conecta a las entidades al principio del diagrama. Los mensajes se dibujan desde el objeto que envía el mensaje al que lo recibe, pudiendo ser ambos el mismo objeto y su orden viene determinado por su posición vertical, un mensaje que se dibuja debajo de otro indica que se envía después, por lo que no se hace necesario un número de secuencia.

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

**Iteraciones y condicionales.**

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

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

Gráfico, Diagrama, Gráfico de cajas y bigotes

Descripción generada automáticamente

**5.2.2.- Elaboración de un diagrama de secuencia.**

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

**5.3.- Diagramas de colaboración.**

Los diagramas de colaboración son un complemento para los de secuencia cuyo objetivo es mostrar las interacciones entre los objetos del diagrama mediante el paso mensajes entre ellos. Las interacciones entre los objetos se describen en forma de [grafo](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/53_diagramas_de_colaboracin.html#tbeac8a10-1adc-b63b-c158-9646ae5a6282)[[21]](#footnote-21) en el que los nodos son objetos y las aristas son enlaces entre objetos a través de los cuales se pueden enviar mensajes entre ellos. Los objetos se conectan mediante enlaces y se comunican a través de los mensajes.

Como, a diferencia de los diagramas de secuencia, no se incluye una línea temporal los mensajes son numerados para determinar su orden el tiempo.

En este diagrama de colaboración el actor Iniciador manda un mensaje al objeto c1 que inicia el escenario, a continuación, el objeto c1 envía el mensaje mensaje1 que lleva un parámetro al objeto c2 y después el mensaje mensaje2, que no tiene parámetros de nuevo al objeto c2.

**5.3.1.- Representación de objetos.**

Un objeto puede ser cualquier instancia de las clases que hay definidas en el sistema, aunque también pueden incluirse objetos como la interfaz del sistema, o el propio sistema, si esto nos ayuda a modelar las operaciones que se van a llevar a cabo.

Los objetos se representan mediante rectángulos en los que aparece uno de estos nombres.

* **NombreClase:** directamente se puede utilizar el nombre de la clase a la que pertenece el objeto que participa en la interacción. Pero esta representación hace referencia a la clase, el resto son objetos.
* **NombreObjeto**: se puede usar el nombre concreto del objeto que participa en la interacción, normalmente aparece subrayado.
* **:nombreClase**: cuando se coloca el símbolo ":" delante del nombre de la clase quiere decir que hace referencia a un objeto genérico de esa clase.
* **NombreObjeto:nombreClase**: hace referencia al objeto concreto que se nombre añadiendo la clase a la que pertenece.

**5.3.2.- Paso de mensajes.**

Para que sea posible el paso de mensajes es necesario que exista una asociación entre los objetos. En la imagen es posible el paso de mensajes entre el objeto objeto1 y objeto2, además de quedar garantizada la navegación y visibilidad entre ambos.

Un mensaje es la especificación de una comunicación entre objetos que transmite información y desencadena una acción en el objeto destinatario. La sintaxis de un mensaje es la siguiente:

donde:

* **Secuencia**: representa el nivel de anidamiento del envío del mensaje dentro de la interacción. Los mensajes se numeran para indicar el orden en el que se envían, y si es necesario se puede indicar anidamiento incluyendo subrangos.
* **\***: indica que el mensaje es iterativo.
* **Condición de guarda**: debe cumplirse para que el mensaje pueda ser enviado.
* **ValorDevuelto**: lista de valores devueltos por el mensaje. Estos valores se pueden utilizar como parámetros de otros mensajes. Los corchetes indican que es opcional.
* **Mensaje**: nombre del mensaje.
* **Argumentos**: parámetros que se pasan al mensaje.

Como se ve en el ejemplo se puede usar la misma asociación para enviar varios mensajes. Vemos que hay dos mensajes anidados, el 1.1 y el 2.1, se ha usado el nombre de los mensajes para indicar el orden real en el que se envían.

Diagrama

Descripción generada automáticamente

Los mensajes 1, 2 y 3 tiene parámetros y los mensajes 1 y 5 devuelve un resultado.

Se contempla la bifurcación en la secuencia añadiendo una condición en la sintaxis del mensaje:

[Secuencia][\*][CondiciónGuarda]{valorDevuelto} : mensaje (argumentos)

Diagrama

Descripción generada automáticamente

Cuando tenemos una condición se repite el número de secuencia y se añaden las condiciones necesarias, como vemos en la imagen según la condición se enviará el mensaje 1 o el 2, pero no ambos, por lo que coinciden en número de secuencia.

La iteración se representa mediante un \* al lado del número de secuencia, pudiendo indicarse ente corchetes la condición de parada del bucle.

**Nota**: VP-UML modifica el orden en el que aparecen los datos pero no su notación.

**5.3.3.- Elaboración de un diagrama de colaboración.**

Este es el diagrama de colaboración del caso de uso **Hacer pedido**, se ha creado siguiendo el diagrama de secuencia, por lo que no te debe ser muy difícil seguirlo, de hecho, algunas aplicaciones para la creación de estos diagramas permiten la obtención de uno a partir de otro. Debes tener en cuenta que la aplicación modifica un poco la signatura de los mensajes, el valor devuelto se representa al final precedido de dos puntos.

Diagrama

Descripción generada automáticamente

Los aspectos más destacados son los siguientes:

* Las actividades que se repiten o pueden repetirse se marcan con un asterisco y su condición.
* Las condiciones de guarda se escriben en el mismo nombre del mensaje.
* El flujo alternativo de eventos según si el usuario cancela el pedido o no, obliga a modificar los números de secuencia de los mensajes 5 y 6, pasando a tener los mensajes 5a y 6a y 5b y 6b, según la condición. Puedes modificar el número se secuencia de los mensajes abriendo la especificación del diagrama, y seleccionando la pestaña Mensajes, donde puedes editar los números de secuencia haciendo doble clic sobre ellos.
* Al objeto "sistema" se le ha asignado el estereotipo system.

**5.4.- Diagramas de actividad.**

El Diagrama de Actividad es una especialización del Diagrama de Estado, organizado respecto de las acciones, que se compone de una serie de actividades y representa cómo se pasa de unas a otras. Las actividades se enlazan por transiciones automáticas, es decir, cuando una actividad termina se desencadena el paso a la siguiente actividad.

Se utilizan fundamentalmente para modelar el flujo de control entre actividades en el que se puede distinguir cuales ocurren secuencialmente a lo largo del tiempo y cuales se pueden llevar a cabo [concurrentemente](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/54_diagramas_de_actividad.html#t1757a8f6-9100-1bb4-70da-15ee4287f437). Permite visualizar la dinámica del sistema desde otro punto de vista que complementa al resto de diagramas. En concreto define la lógica de control:

* **En el modelado de los procesos del negocio**: Permiten especificar y evaluar el flujo de trabajo de los procesos de negocio.
* **En el análisis de un caso de uso**: Permiten comprender qué acciones deben ocurrir y cuáles son las dependencias de comportamiento.
* **En la comprensión del flujo de trabajo, a través de varios casos de uso**: Permiten representar con claridad las relaciones de flujo de trabajo (workflow) entre varios casos de uso.
* Aclaran cuando se trata de expresar **aplicaciones multihilos**.

Un diagrama de actividades es un grafo conexo en el que los nodos son estados, que pueden ser de actividad o de acción y los arcos son transiciones entre estados. El grafo parte de un nodo inicial que representa el estado inicial y termina en un nodo final.

**5.4.1.- Elementos del diagrama de actividad.**

Normalmente los diagramas de actividades contienen:

* **Estados** de actividad y estados de acción.
  + **Estado de actividad**: Elemento compuesto cuyo flujo de control se compone de otros estados de actividad y de acción.
  + **Estado de acción**: Estado que representa la ejecución de una acción atómica, que no se puede descomponer ni interrumpir, normalmente la invocación de una operación. Generalmente se considera que su ejecución conlleva un tiempo insignificante.
  + Pueden definirse también otro tipo de estados:
    - **Inicial**.
    - **Final**

**Transiciones**: Relación entre dos estados que indica que un objeto en el primer estado realizará ciertas acciones y pasará al segundo estado cuando ocurra un evento específico y satisfaga ciertas condiciones. Se representa mediante una línea dirigida del estado inicial al siguiente. Podemos encontrar diferentes tipos de transacciones:

* **Secuencial o sin disparadores**: Al completar la acción del estado origen se ejecuta la acción de salida y, sin ningún retraso, el control sigue por la transición y pasa al siguiente estado.
* **Bifurcación(Decision node)**: Especifica caminos alternativos, elegidos según el valor de alguna expresión booleana. Las condiciones de salida no deben solaparse y deben cubrir todas las posibilidades (puede utilizarse la palabra clave **else**). Pueden utilizarse para lograr el efecto de las iteraciones.

Diagrama

Descripción generada automáticamente

* **Fusión (Merge node)**: Redirigen varios flujos de entrada en un único flujo de salida. No tiene tiempo de espera ni sincronización.
* **División (Fork node)**: Permiten expresar la sincronización o ejecución paralela de actividades. Las actividades invocadas después de una división son concurrentes.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

* + **Unión (Join node):** Por definición, en la unión los flujos entrantes se sincronizan, es decir, cada uno espera hasta que todos los flujos de entrada han alcanzado la unión.
* **Objetos**: Manifestación concreta de una abstracción o instancia de una clase. Cuando interviene un objeto no se utilizan los flujos de eventos habituales sino flujos de objetos (se representan con una flecha de igual manera) que permiten mostrar los objetos que participan dentro del flujo de control asociado a un diagrama de actividades. Junto a ello se puede indicar cómo cambian los valores de sus atributos, su estado o sus roles.

Diagrama

Descripción generada automáticamente

Se utilizan carriles o calles para ver **QUIENES** son los responsables de realizar las distintas actividades, es decir, especifican qué parte de la organización es responsable de una actividad.

* Cada calle tiene un nombre único dentro del diagrama.
* Puede ser implementada por una o varias clases.
* Las actividades de cada calle se consideran independientes y se ejecutan concurrentemente a las de otras calles.

**5.4.2.- Elaboración de un diagrama de actividad.**

El siguiente diagrama de actividad representa el caso de uso hacer pedido, en el aparecen los elementos que hemos visto en las secciones anteriores.

* En las bifurcaciones se ha añadido la condición que indica si se pasa a una acción o a otra.
* Las acciones Seleccionar artículo y Seleccionar cantidad se han considerado concurrentes.

Diagrama

Descripción generada automáticamente

En este otro diagrama se simplifican las acciones a realizar y se eliminan los objetos para facilitar la inclusión de calles que indican quien realiza cada acción:

Diagrama, Esquemático

Descripción generada automáticamente

**Nota**: Para añadir las calles en Visual Paradigm se utiliza la herramienta del panel "Vertical Swimlane".

**5.5.- Diagramas de estados.**

Basado en los statecharts de David Harel. Representan máquinas de estados ([autómatas de estados finitos](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/55_diagramas_de_estados.html#tc2bbf321-0e59-2f15-de95-2dea2955430a))[[22]](#footnote-22) para modelar el comportamiento dinámico basado en la respuesta a determinados eventos de aquellos objetos que requieran su especificación, normalmente por su comportamiento significativo en tiempo real y su participación en varios casos de uso. El resto de objetos se dice que tienen un único estado.

En relación con el diagrama de estados se cumple que:

* Un objeto está en un estado concreto en un cierto momento, que viene determinado, parcialmente, por los valores de sus atributos.
* La transición de un estado a otro es momentánea y se produce cuando ocurre un determinado evento.
* Una máquina de estados procesa un evento cada vez y termina con todas las consecuencias del evento antes de procesar otro. Si ocurren dos eventos simultáneamente se procesan como si se hubieran producido en cualquier orden, sin pérdida de generalidad.

Un diagrama de máquina de estados expresa el comportamiento de un objeto como una progresión a través de una serie de estados, provocada por eventos y las acciones relacionadas que pueden ocurrir. Por ejemplo, aquí tenemos el diagrama de estados de una puerta.

Diagrama

Descripción generada automáticamente

**5.5.1.- Estados y eventos.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Elementos** de un estado

* Nombre
* Acciones entrada/salida.
* Actividad a realizar.
* Subestados, cuando el estado es complejo y necesita de un diagrama que lo defina.
* Eventos diferidos.

Existen dos tipos de estado especiales, estado inicial y estado final.

* **Estado inicial**: es un pseudoestado que indica el punto de partida por defecto para una transición cuyo destino es el límite de un estado compuesto. El estado inicial del estado de nivel más alto representa la creación de una nueva instancia de la clase.
* **Estado final**: Estado especial dentro de un estado compuesto que, cuando está activo, indica que la ejecución del estado compuesto ha terminado y que una transición de finalización que sale del estado compuesto está activada.

Imagen que contiene interior, foto, hecho de madera, tabla

Descripción generada automáticamente

Tipos de eventos:

* **Señales (excepciones)**: la recepción de una señal, que es una entidad a la que se ha dado nombre explícitamente (clase estereotipada), prevista para la comunicación explícita - y asíncrona- entre objetos. Es enviada por un objeto a otro objeto o conjunto de objetos. Las señales con nombre que puede recibir un objeto se modelan designándolas en un compartimento extra de la clase de ese objeto. Normalmente una señal es manejada por la máquina de estados del objeto receptor y puede disparar una transición en la máquina de estados.
* **Llamadas**: la recepción de una petición para invocar una operación. Normalmente un evento de llamada es modelado como una operación del objeto receptor, manejado por un método del receptor y se implementa como una acción o transición de la máquina de estados.
* **Paso de tiempo**: representa el paso del tiempo (ocurrencia de un tiempo absoluto respecto de un reloj real o virtual o el paso de una cantidad de tiempo dada desde que un objeto entra en un estado). Palabra clave **after**: **after (2 segundos); after 1 ms desde la salida de devInactivo**.
* **Cambio de estado**: evento que representa un cambio en el estado o el cumplimiento de una condición. Palabra clave **when**, seguida de una expresión booleana, que puede ser de tiempo o de otra clase: **when (hora = 11:30); when ( altitud < 1000)**.

**5.5.2.- Transiciones.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Elementos de una transición:

* **Estados origen y destino**: la transición se disparará si, estando en el estado origen se produce el evento de disparo y se cumple la condición de guarda (si la hay), pasando a ser activo el estado final.
* **Evento de disparo**: cuando se produce un evento, afecta a todas las transiciones que lo contienen en su etiqueta. Todas las apariciones de un evento en la misma máquina de estados deben tener la misma signatura. Los tipos de evento los hemos visto en el punto anterior.
* **Condición de guarda**: expresión booleana. Si es falsa, la transición no se dispara, y si no hay otra transición etiquetada con el mismo evento que pueda dispararse, éste se pierde.
* **Acción**: computación atómica ejecutable. Puede incluir llamadas a operaciones del objeto que incluye la máquina de estados (o sobre otros visibles), creación o destrucción de objetos o el envío de una señal a otro objeto.

**5.5.3.- Creación de un diagrama de estados.**

Para ejemplificar la creación de un diagrama de estados vamos a ver el que se asocia al objeto pedido, que cumple con las condiciones que hemos visto al principio, tiene un comportamiento significativo en tiempo real, ya que su situación tanto física, como el sistema, va evolucionando conforme pasa el tiempo, y participa en varios casos de uso (como Hacer pedido y Cumplimentar pedido).

Los diferentes estados en los que puede estar un pedido son:

* **En creación**: es cuando se están seleccionando los productos que formará el pedido.
* **Pendiente**: está en este estado desde que se confirma el pedido hasta que se selecciona para preparar su envío.
* **En almacén**: está en este estado cuando es elaborado el paquete y se ha asignado a una ruta, hasta que se envía a través de la ruta que le corresponde.
* **Servido**: Cuando el pedido es enviado. En este caso se envía una señal física desde el almacén cuando el transporte abandona el almacén.
* **Cancelado**: puede llegarse a esta situación por dos motivos, o bien se cancela mientras se está haciendo por problemas con la tarjeta de crédito, o bien porque, una vez pendiente de su gestión el usuario decide cancelarlo, la diferencia fundamental entre ambos es que en el segundo caso hay que devolver el importe pagado por el pedido al socio que lo ha comprado.

Las transiciones entre estados se producen por llamadas a procedimientos en todos los casos, no intervienen cambios de estado o el tiempo, ni señales.

El diagrama quedaría de la siguiente manera:

Diagrama

Descripción generada automáticamente

[Resumen textual alternativo](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19646/mod_scorm/content/11/ED06_Descripcion_Imagen_Diagrama_estados.html)

En las **transiciones** se ha incluido el nombre de la transición, el evento que la dispara (normalmente hacer clic en algún botón de la interfaz), si existe condición de guarda se pone entre corchetes y la acción a realizar para llegar al siguiente estado junto al símbolo /. En todos los casos el evento de disparo es de tipo llamada (incluye la llamada a una función o pulsar un botón de la interfaz), salvo en el caso de pedido enviado que se controla por una señal que se envía cuando el transporte abandona el almacén.

A los **estados** se les ha añadido la acción a realizar, apartado do/ y en algunos casos la acción de entrada, por ejemplo en el caso del estado pendiente, se debe revisar que los artículos a enviar tengan disponibilidad y la de salida, en el ejemplo disminuir el stock.

**Nota**: para incluir las condiciones de guarda en el diagrama debes rellenar el apartado "Guard" de la especificación, si necesitas añadir alguna acción puedes hacerlo rellenando el apartado "Effect". Los eventos de disparo.

**6.- Ingeniería inversa.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tiene como caso particular la reingeniería que es el proceso de extraer el código fuente de un archivo ejecutable.

La ingeniería inversa puede ser de varios tipos:

* **Ingeniería inversa de datos**: Se aplica sobre algún código de bases datos (aplicación, código SQL, etc.) para obtener los modelos relacionales o sobre el modelo relacional para obtener el diagrama entidad-relación.
* **Ingeniería inversa de lógica o de proceso**: Cuando la ingeniería inversa se aplica sobre el código de un programa para averiguar su lógica (reingeniería), o sobre cualquier documento de diseño para obtener documentos de análisis o de requisitos.
* **Ingeniería inversa de interfaces de usuario**: Se aplica con objeto de mantener la lógica interna del programa para obtener los modelos y especificaciones que sirvieron de base para la construcción de la misma, con objeto de tomarlas como punto de partida en procesos de ingeniería directa que permitan modificar dicha interfaz.

Ejercicio resuelto

**A partir del código que has obtenido en el ejercicio anterior genera el diagrama de clases correspondiente.**

Desde**VP-UML para NetBeans** tendremos acceso a una herramienta que nos permite la transformación de código Java en diagramas de clases. Para ello:

1. Seleccionamos el proyecto en el panel de proyecto.
2. Desde el nodo del proyecto seleccionamos en el menú contextual la opción "**Update UML Model**" lo que iniciará el proceso de ingeniería inversa..

**Desde VP-UML**

Haremos el proceso de ingeniería inversa desde **Herramienta >> Códigol >> Reverse Code (Java o C++)**. Al seleccionar esta opción nos preguntará donde se localiza el código fuente. El proceso no genera el diagrama exactamente igual que el original, es capaz de obtener las clases y las relaciones de herencia. El resto de relaciones tendremos que establecerlas nosotros a mano.

1. Conjunto de sentencias escritas en un lenguaje de programación que operan sobre un conjunto de parámetros y producen un resultado. [↑](#footnote-ref-1)
2. Es el conjunto de una colección de datos y de las funciones que modifican esos datos, que recrean una entidad con sentido, en el contexto de un problema. [↑](#footnote-ref-2)
3. Paradigma de programación que postula que un programa informático no es más que una sucesión de llamadas a funciones, bien sean del sistema o definidas por el usuario. [↑](#footnote-ref-3)
4. Modelo o patrón aplicado a cualquier disciplina científica u otro contexto epistemológico. [↑](#footnote-ref-4)
5. Aislar un elemento de su contexto o del resto de elementos que le acompañan para disponer de ciertas características que necesitamos excluyendo las no pertinentes. Con ello capturamos algo en común entre las diferentes instancias con objeto de controlar la complejidad del software. [↑](#footnote-ref-5)
6. Reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción, permite mejorar la cohesión y se implementa a través del principio de ocultación. [↑](#footnote-ref-6)
7. Relacionado con la ingeniería del software, consiste en el aislamiento del estado, es decir, de los datos miembro de un objeto, de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto. Este aislamiento protege a los datos de que sean modificados por alguien que (…). [↑](#footnote-ref-7)
8. La palabra proviene del griego significa varias formas. En el paradigma de orientación a objetos hace referencia al comportamiento diferente que adopta un objeto según el tipo que tenga, o de un método según sean sus parámetros. [↑](#footnote-ref-8)
9. Propiedad del paradigma de orientación a objetos que permite que un objeto sea construido a partir de otro, reutilizando sus propiedades y métodos y aportando los suyos propios. Puede ser simple o compuesta según se herede de un único objeto o de varios. [↑](#footnote-ref-9)
10. La escalabilidad es la propiedad deseable de un sistema, red o proceso que le permite hacerse más grande sin rehacer su diseño y sin disimular su rendimiento. [↑](#footnote-ref-10)
11. Objeto de una clase, creado en tiempo de ejecución con un estado concreto. [↑](#footnote-ref-11)
12. Un artefacto es una información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software. [↑](#footnote-ref-12)
13. Conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos. Junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos. [↑](#footnote-ref-13)
14. En el contexto del desarrollo de software, la transformación de un modelo en código a través de su traducción de un determinado lenguaje de programación [↑](#footnote-ref-14)
15. En el contexto del desarrollo de software, la transformación del código en un modelo a través de su traducción desde un determinado lenguaje de programación. [↑](#footnote-ref-15)
16. Se define como dominio un área de conocimiento o actividad caracterizada por un conjunto de conceptos y terminología comprendida por los practicantes de ese dominio. El dominio del problema es aquel sobre el que se define el problema a resolver por el sistema que se va a generar. [↑](#footnote-ref-16)
17. Mecanismo de propósito general para organizar elementos en grupos. [↑](#footnote-ref-17)
18. Cuando se utiliza el concepto de herencia es la clase que contiene los métodos y atributos que van a ser heredados por la clase derivada. [↑](#footnote-ref-18)
19. Cuando se utiliza la herencia es la clase que hereda los atributos y métodos de la clase base. [↑](#footnote-ref-19)
20. Grafo en el que no es obligatorio que se pueda trazar un camino (sucesión de arcos que une dos nodos cualesquiera) entre cualquier pareja de nodos a y b. [↑](#footnote-ref-20)
21. Significa trazas. Objeto combinatorio formado por un conjunto de nodos y un subconjunto de líneas seleccionadas del conjunto de líneas que unen cada par de nodos entre sí, denominadas arcos. Cuando dos nodos del grafo están unidos por un arco se dice que existe una relación entre los nodos. [↑](#footnote-ref-21)
22. Modelo matemático que realiza cómputos sobre una entrada para producir una salida. Se representa mediante un grafo cerrado y dirigido cuyos vértices son estados y sus artistas son transiciones, que van etiquetas con un símbolo que representa el evento que desencadena la transición. Parte de un estado inicial y termina en uno o varios estados finales [↑](#footnote-ref-22)