

Diseño orientado a objetos.

Elaboración de diagramas.



Caso práctico

En BK Programación continúan inmersos en el mundo de UML. A pesar de que han trabajado duro y han aprendido bastante acerca de este lenguaje de especificación, Ada se ha dado cuenta de que apenas han empezado a arañar la superficie de todas las posibilidades que les ofrece. De momento ya saben como crear un diagrama de clases bastante completo y como analizar un problema propuesto, sin embargo hay muchos aspectos del problema que no pueden modelar todavía, por ejemplo con solo el diagrama de clases no pueden saber qué se espera del sistema que van a construir, o en qué se deben basar para codificar los métodos, o simplemente, ¿Cómo colaboran los objetos de las clases que han creado para hacer alguna tarea que sea útil?



Ada decide que no pueden parar ahora, y que hay que hacer un esfuerzo final para que los conocimientos del equipo sean globales y puedan enfrentarse a cualquier desarrollo software con solvencia.

Al momento, Ada pone a su equipo manos a la obra.

1.- Introducción a la orientación a objetos.



Caso práctico



Ya en la sala de reuniones...

—Deberíamos empezar por revisar cual es la situación actual. Como ya sabéis existen diferentes lenguajes de programación que se comportan de manera diferente, y esto determina en gran medida el enfoque que se le da al análisis previo. No es lo mismo un lenguaje estructurado que uno orientado a objetos. Tendríamos que conocer las características de ambos enfoques para entender un poco mejor cómo se analizan.

—Es cierto —contesta **Juan** —desde que empecé en el mundo de la informática esto ha cambiado un poco, así que he tenido que ir investigando para adaptarme a los nuevos lenguajes de programación, si queréis, os pongo al día brevemente, ...

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 entretrejan hasta formar una solución global al problema de partida. Era, pues, un proceso centrado en los procedimientos, se codificaban mediante 🧑‍💻 funciones que actuaban sobre 🧑‍💻 estructuras de datos, por eso a este tipo de programación se le llama 🧑‍💻 programación estructurada. 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 el proceso se centra en simular los elementos de la realidad asociada al problema de la forma más cercana posible. La 🧑‍💻 abstracción 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.



Para saber más

Todo acerca del mundo de la orientación a objetos se encuentra en la página oficial del

[Grupo de gestión de objetos.](#)

2.- Conceptos de Orientación a Objetos.



Caso práctico

—De acuerdo, buen resumen **Juan**, sin embargo, los últimos proyectos que han entrado a la empresa se han desarrollado en su totalidad mediante software orientado a objetos, hemos usado PHP con Javascript, pero sobre todo Java, que es un lenguaje basado en objetos, así que sería necesario que analizáramos con un poco más de detenimiento el enfoque orientado a objetos, que características presenta, y que ventajas tiene sobre otros.



—¡Gracias, **Ada**!, también tengo alguna información sobre eso...

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:** 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:** 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:** Consiste en reunir bajo el mismo nombre comportamientos diferentes. La selección de uno u otro depende del objeto que lo ejecute.
- ✔ 🧑 **Herencia:** 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.



Caso práctico

—Además la orientación a objetos cuenta con una serie de ventajas que nos vienen muy bien a los que nos dedicamos a la construcción de software, sobre todo porque nos facilitan su construcción y mantenimiento al dividir un problema en módulos claramente independientes y que, además, cuando ya tenemos suficientemente probados y completos podemos utilizar en otras aplicaciones, la verdad que ahorra bastante tiempo y esfuerzo... — argumenta **Juan**.



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. Es sencillo modificar la estructura y el comportamiento de los objetos sin tener que cambiar la aplicación.

Autoevaluación

¿Cual es la afirmación más adecuada al paradigma de orientación a objetos?

- ☐ Permite crear aplicaciones basadas en módulos de software que representan objetos del entorno del sistema, por lo que no son apropiados para dar solución a otros problemas.
- ☐ Tiene como objetivo la creación de aplicaciones basadas en abstracciones de datos estáticas y de difícil ampliación.
- ☐ Permite crear aplicaciones cuyo mantenimiento es complicado porque las modificaciones influyen a todos los objetos del sistema.
- ☐ Permite crear aplicaciones basadas en módulos que pueden reutilizarse, de fácil modificación y que permiten su ampliación en función del crecimiento del sistema.

2.2.- Clases, atributos y métodos.



Caso práctico

—De acuerdo, ahora conocemos las características básicas y ventajas de usar la orientación a objetos, ¿qué más nos haría falta? ¿Quizá sus estructuras básicas?

—Yo puedo contaros algo sobre eso, —comenta **Juan**— lo estudié en el Ciclo Formativo.



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.

Autoevaluación

Un objeto es una concreción de una clase, es decir, en un objeto se concretan valores para los atributos definidos en la clase, y además, estos valores podrán modificarse a través del paso de mensajes al objeto.

- ☐ Verdadero.
- ☐ Falso.

2.3.- Visibilidad.



Caso práctico

—Pues creo que ya lo tenemos todo...

—No creas, —dice **Ada** que siempre sabe algo más, que el resto desconoce— en orientación a objetos, existe un concepto muy importante, que es el de visibilidad, permite definir hasta qué punto son accesibles los atributos y métodos de una clase, por regla general, cuando definimos atributos los ocultamos, para que nadie pueda modificar el estado del objeto, y dejamos los métodos abiertos, porque son los que permiten el paso de mensajes entre objetos...



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).
-

Autoevaluación

¿Desde dónde se puede acceder al estado de una clase?

- ☐ Desde cualquier zona de la aplicación.
- ☐ Desde la clase y sus clases derivadas.
- ☐ Solo desde los métodos de la clase.

2.4.- Objetos. Instanciación.



Caso práctico

Antonio ha asistido a esta reunión como parte de su formación laboral, pero se encuentra algo perdido entre tantos conceptos:

—A ver, estamos todo el tiempo hablando de que las clases tienen atributos y métodos, luego, que los objetos se pasan mensajes, que son los que modifican los atributos, entonces, ¿no son lo mismo?, ¿qué diferencia hay?



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

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

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.



Citas para pensar

Mientras que un objeto es una entidad que existe en el tiempo y el espacio, una clase representa sólo una abstracción, "la esencia" del objeto, si se puede decir así.

Grady Booch

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.



Caso práctico

Ahora que el equipo conoce los fundamentos de la orientación a objetos llega el momento de ver como pueden poner en práctica los conocimientos adquiridos.

Ada está interesada, sobre todo, en que sean capaces de representar las clases de los proyectos que están desarrollando y como se relacionan entre ellas. Para ello decide comenzar comentando las características de un lenguaje de modelado de sistemas orientados a objetos llamado UML. Este lenguaje permite construir una serie de modelos, a través de diagramas de diferentes visiones de un proyecto.



—Es importante apreciar como estos modelos, nos van a permitir poner nuestras ideas en común utilizando un lenguaje específico, facilitarán la comunicación, que como sabéis, es algo esencial para que nuestro trabajo en la empresa sea de calidad.




Citas para pensar




Una empresa de software con éxito es aquella que produce de manera consistente software de calidad que satisface las necesidades de los usuarios. El modelado es la parte esencial de todas las actividades que conducen a la producción de software de calidad.



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.

¿Porqué es útil modelar?

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

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

3.1.- Tipos de diagramas UML.



Caso práctico

Cuando **María** estudió el ciclo formativo no llegó a ver estas tecnologías con tanto detenimiento, así que está asimilándolo todo poco a poco:

—De acuerdo, UML describe el sistema mediante una serie de modelos que ofrecen diferentes puntos de vista. Pero ¿qué tenemos que hacer para representar un modelo?, ¿en que consiste exactamente?

—Utilizaremos diagramas, que son unos  grafos en los que los nodos definen los elementos del diagrama, y los arcos las relaciones entre ellos.



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 diagramas 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.



Citas para pensar

Un 80% de las aplicaciones se pueden modelar con el 20% de los diagramas UML.

3.2.- Herramientas para la elaboración de diagramas UML.



Caso práctico

—Ahora que conocemos los diagramas que podemos generar para describir nuestro sistema, sería buena idea buscar alguna herramienta que nos ayude a elaborarlos. ¡No sería nada práctico andar todo el día con la libreta a cuestas!

—Lo que nos permite conocer a un buen desarrollador es que siempre hace un buen esquema inicial de cada proyecto, y eso puede hacerse en miles de soportes, desde una libreta a un servilleta, cualquier cosa que te permita hacer un pequeño dibujo, no obstante tienes razón. El uso de herramientas, además de facilitar la elaboración de los diagramas, tiene otras ventajas, como la integración en entornos de desarrollo, con lo que podremos generar el código base de nuestra aplicación desde el propio diagrama.



-¡Guau, eso sí es facilitar el trabajo!

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

Podemos encontrar, entre otras, las siguientes herramientas:

- ✓ **Rational Systems Developer de IBM:** Herramienta propietaria que permite el desarrollo de proyectos software basados en la metodología UML. Desarrollada en origen por los creadores de UML 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**.



Para saber más

Si sientes curiosidad puedes seguir este enlace a la página oficial de [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, incluye instaladores para Windows y Linux.



Para saber más

Aquí tienes el enlace a la página oficial de [Visual Paradigm](#).

- ✔ **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.



Para saber más

Aquí tienes el enlace a la página oficial de [ArgoUML](#).

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.



[Visual Paradigm International](http://www.visual-paradigm.com/), Anagrama de Visual Paradigm. (Todos los derechos reservados)



Debes conocer

En el anexo I dispones de la información para la instalación de Visual Paradigm 12 y su integración con Netbeans.

Autoevaluación

Las herramienta CASE para la elaboración de diagramas UML sirven solo para la generación de los diagramas asociados al análisis y diseño de una aplicación. ¿Verdadero o falso?

- ☐ Verdadero.
- ☐ Falso.

4.- Diagramas estructurales



Caso práctico

En la empresa ya han instalado Visual Paradigm, **Juan y María** están empezando a investigar su funcionamiento, y como utilizarlo desde un proyecto de NetBeans.

—Empecemos por los 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.



Caso práctico

Juan y María han detectado que entre los diagramas estructurales el más importante es el diagrama de clases.

— Fíjate, el diagrama de clases representa la estructura estática del sistema y las relaciones entre las 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, 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, que a su vez se relacionan entre sí.

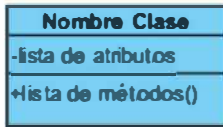


Ejercicio resuelto

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

Mostrar retroalimentación

4.1.1.- Creación de clases.



Visual Paradigm. Clase UML
(Todos los derechos reservados)

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



Citas para pensar

"Una clase es una descripción de un conjunto de objetos que manifiestan los mismos atributos, operaciones, relaciones y la misma semántica."

"Una clase (*Object Modelling and Design [Rumbaugh et al., 1991]*) es un conjunto de objetos que comparten una estructura y un comportamiento comunes."

[Booch G., 1994]



Ejercicio resuelto

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

Mostrar retroalimentación

Autoevaluación

**Al crear una clase es obligatorio definir nombre, atributos y métodos.
¿Verdadero o falso?**

- ☐ Verdadero.
- ☐ Falso.

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.

Mostrar retroalimentación

Autoevaluación

¿Cómo sabemos que los atributos tienen visibilidad privada en el diagrama?

- ☐ Porque aparecen acompañados del símbolo más “+”.
- ☐ Porque aparecen acompañados del símbolo almohadilla “#”.
- ☐ Porque aparecen acompañados del símbolo “~”.
- ☐ Porque aparece acompañado del símbolo menos “-”.

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. Cuando 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

Mostrar retroalimentación

Autoevaluación

¿Cuál es el método que no devuelve ningún tipo de dato?

- ☐ El constructor.
- ☐ Todos los métodos devuelven algo, aunque sea void.
- ☐ ~.

4.2.- Relaciones entre clases.



Caso práctico


—Es fácil, ¿lo ves?, por ejemplo, para la aplicación de venta por Internet, tendríamos como clases socio, pedido o artículo, los socios se caracterizan por sus datos personales, los pedidos por su número, fecha, o localidad de destino y los artículos por el código o su descripción.

—Si eso lo veo claro, pero, ¿cómo lo ponemos todo junto? ¿Cómo se conecta el socio con el pedido y el artículo?



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.

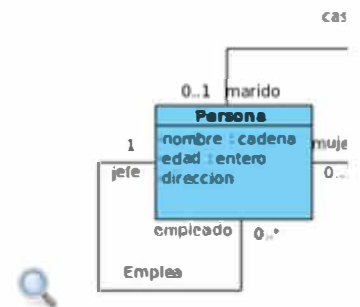


Ejercicio resuelto

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

Mostrar retroalimentación

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.



[Visual Paradigm](#). Clase con relaciones unarias. (Todos los derechos reservados)

Autoevaluación

Para obtener las relaciones de un diagrama nos basamos en la descripción de los requisitos del dominio, pero, ¿se pueden crear relaciones en el diagrama que no aparezcan especificadas en la lista de requisitos del problema?

- ☐ No se puede, las relaciones se deben extraer de la descripción del problema, si no lo hiciéramos así nos estaríamos inventando información.
- ☐ Sí se puede, a veces se infiere información o se conocen cosas del problema que no aparecen en la descripción de los requisitos.

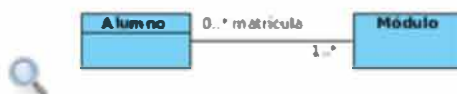
4.2.1.- Cardinalidad o multiplicidad de la relación

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.

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

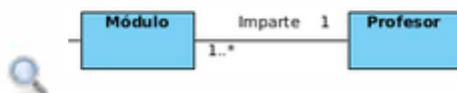
Por ejemplo, si tengo la siguiente relación:



[Visual Paradigm..](#) Cardinalidad I.. (Todos los derechos reservados)

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:



[Visual Paradigm..](#) Cardinalidad II.. (Todos los derechos reservados)

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.

Mostrar retroalimentación

4.2.2.- Relación de herencia.

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

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

Tipos:

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

Representación:

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



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".



[Visual Paradigm](#).. Clases candidatas para herencia. (Todos los derechos reservados)

Mostrar retroalimentación

Autoevaluación

He creado una clase persona cuyos atributo son Nombre, fechaContratación y numeroCuenta. De esta clase derivan por herencia la clase Empleado y JefeDepartamento. ¿Cómo debe declararse un método en la clase Persona que se llame CalculaAntigüedad que se usa sólo para calcular el sueldo de los empleados y jefes de departamento?

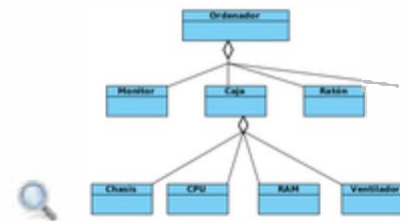
- ☐ Público.
- ☐ Privada.
- ☐ Protegida.
- ☐ Paquete.

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 (pertenecer 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 por qué 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.



Visual Paradigm.. Relación de agregación. (Todos los derechos reservados)

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

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



Visual Paradigm.. Relación de composición. (Todos los derechos reservados)

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.

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.

Mostrar retroalimentación

Autoevaluación

Siguiendo con el ejemplo anterior, para modelar el cálculo de la nota media de un alumno se añade el método *calcularNotaMedia* a la clase *Alumno* que realiza la media de las calificaciones de los módulos en los que el alumno se encuentra matriculado para este curso. ¿Qué visibilidad se debería poner a este método?

- ☐ Público.
- ☐ Privado.
- ☐ Protegido.
- ☐ Paquete.

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

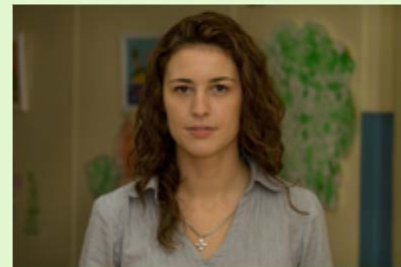


Caso práctico

María y Juan siguen comentando la creación de diagramas de clases.

—Las reservas se utilizan para relacionar los clientes y las habitaciones, eso es sencillo de ver, pero si tenemos un enunciado un poco más largo, puede no ser tan obvio. Quizá podrías darme algún consejo sobre cómo pasar de los requisitos iniciales de una aplicación a un primer diagrama de clases.

—Es verdad, la cosa se complica un poco cuando tenemos más requisitos, pero la clave está en analizar el texto para obtener nombres y continuar el desarrollo a partir de ahí.



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.

Se debe tener en cuenta que la lista no incluye todo, habrá que añadir objetos adicionales para completar el modelo y también, que diferentes descripciones del problema pueden provocar la toma de diferentes decisiones de creación de objetos y atributos.

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.

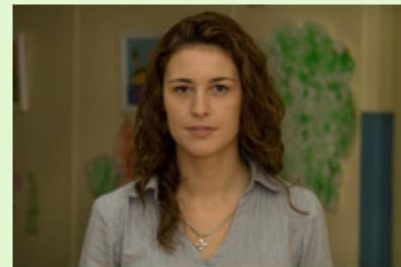
4.4.- Generación de código a partir del diagrama de clases.



Caso práctico

—Bueno, ya tenemos el diagrama, es cierto que es bastante útil para aclarar ideas en el equipo y establecer un plan de trabajo inicial. Además es más fácil empezar a programar porque ya tenemos la línea a seguir, ahora solo falta que empecemos a crear clases y a rellenarlas, ¿Verdad?

—Pues sí, pero aún no lo sabes todo, el diagrama de clases aún te va a dar más facilidades...



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.



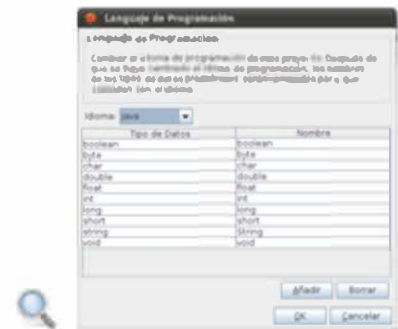
Ejercicio resuelto

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

Mostrar retroalimentación

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

El lenguaje final de implementación de la aplicación influyen 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**.

4.5.- Generación de la documentación.



Caso práctico

Los chicos siguen estudiando características de la herramienta VP-UML...

—Sería estupendo que después de generar un diagrama, en el que verdaderamente te has esforzado, pudieras hacer anotaciones sobre la importancia de cada clase, atributo o relación, para que, posteriormente, pudieras compartir esa información o recordarlo rápidamente cuando estuvieras programando el sistema en caso de tener que consultar algo.

—No solo eso, además de generar documentación exhaustiva, la herramienta permite crear informes con ella, pasándola a un formato más cómodo de leer e interpretar en papel por el equipo de desarrollo.



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

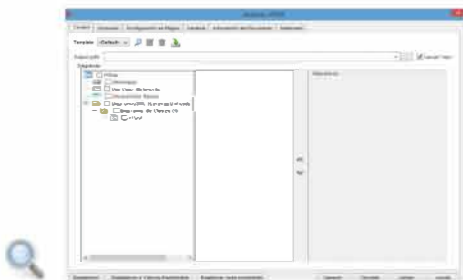
Cuando los modelos están completos podemos generar un informe en varios formatos diferentes (HTML, PDF o Word) con la documentación que hemos escrito. Para generar un informe hacemos:

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.



[Visual Paradigm](#). Ventana para la generación de informes HTML en Visual Paradigm. (Todos los derechos reservados)

5.- Diagramas de comportamiento.



Caso práctico

-Compañeros, creo que ahora no debemos conformarnos con modelar diagramas de clase y nada más, esto no nos da posibilidades de incluir ninguna información acerca del comportamiento de nuestro sistema. ¿Cómo especificamos la funcionalidad? O ¿qué acciones se realizan?, o ¿las restricciones? Necesitamos seguir estudiando diagramas que nos ayuden a especificar la dinámica del sistema. ¿Empezamos ahora mismo?



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.



Caso práctico

-Empezaremos por el principio. Cuando estamos diseñando software es esencial saber cuales son los requerimientos del sistema que queremos construir, y necesitamos alguna herramienta que nos ayude a especificarlos de una manera clara, sistemática, y que nuestros clientes puedan entender fácilmente, ya que es imprescindible que nos pongamos de acuerdo en lo que realmente queremos hacer. ¿Alguna idea?



-¿No bastaría con hacer una lista de requerimientos y describirlos exhaustivamente?

-No, una descripción textual puede inducir a errores de interpretación y suele dejar cabos sueltos, y no contempla otra información, como quien realiza las acciones que describimos, por ejemplo. Necesitamos algo más específico.

Lo que Ada necesita son los 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.

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

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



Reflexiona

Los diagramas de casos de uso se crean en la primera etapa de desarrollo del software, y se enmarcan en el proceso de análisis, para definir de forma detallada la funcionalidad que se espera cumpla el software, y que, además, se pueda comunicar fácilmente al usuario, pero, ¿termina aquí su función?

[Mostrar retroalimentación](#)

5.1.1.- Actores.



Caso práctico

-Como decía, uno de los principales problemas de una descripción textual es que no permite especificar adecuadamente qué personas o entidades externas interactúan con el sistema. Ahora tenemos una herramienta que tiene esto muy en cuenta.



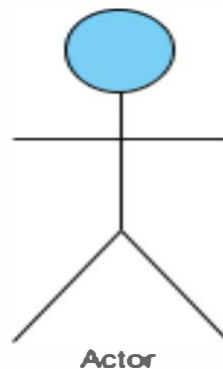
Los **actores** representan un tipo de usuario del sistema. Se entiende como usuario cualquier cosa externa que interactúa con el sistema. No tiene por qué ser un ser humano, puede ser otro sistema informático o unidades organizativas o empresas.

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:** interactúan con el sistema para explotar su funcionalidad. Trabajan directa y frecuentemente con el software.
- ✓ **Secundarios:** soporte del sistema para que los primarios puedan trabajar. Son precisos para alcanzar algún objetivo.
- ✓ **Iniciadores:** 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".

Autoevaluación

¿Un sistema software externo, como una entidad para la autenticación de claves, podría considerarse como un actor en un diagrama de casos de uso?

- ☐ Verdadero.
- ☐ Falso.

5.1.2.- Casos de uso.



Caso práctico

-Vale, pero lo que verdaderamente queremos es identificar la funcionalidad del sistema ¿no?, ¿cómo hace esta herramienta la descripción de la funcionalidad?



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).

Autoevaluación

"Tras comprobar todos los artículos el pedido queda en el almacén a la espera de ser recogido."

¿Dónde incluirías esta afirmación sobre un caso de uso en un contrato?

- ☐ En el flujo de eventos normal.
- ☐ En el flujo de eventos alternativo.
- ☐ En las precondiciones.
- ☐ En las postcondiciones.


5.1.3.- Relaciones.



Caso práctico

-De acuerdo, y ahora ¿Cómo asociamos a los actores con los casos de uso que pueden realizar?



Los diagramas de casos de uso son  grafos no conexos en los que los nodos son actores y casos de uso, y las aristas son las relaciones que existen entre ellos. 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.

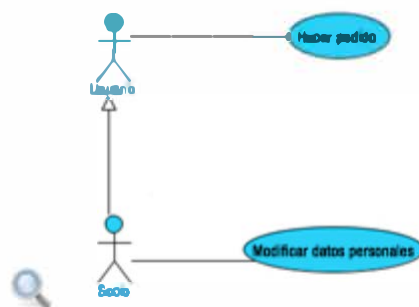
Una asociación se representa mediante una línea continua que une un actor con un caso de uso. Por ejemplo, un usuario de un sistema de venta por Internet puede hacer un pedido, lo que se representa del siguiente modo:



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.

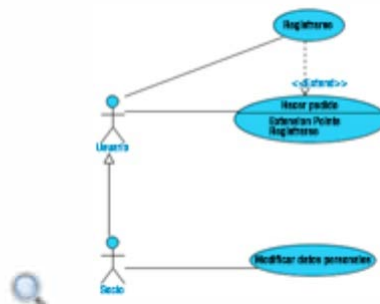


5.1.3.3.- Extensión.

Se utiliza una relación entre dos casos de uso de tipo "**extends**" cuando se desea especificar que el comportamiento de un caso de uso es diferente dependiendo de ciertas circunstancias.

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.

Se incluye una relación entre dos casos de uso de tipo "**include**" cuando la ejecución del caso de uso incluido se da en la rutina normal del caso que lo incluye.

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.

Cuando usamos relaciones de inclusión o extensión no podemos olvidar que los casos de uso extendidos o incluidos deben cumplir con las características propias de un caso de uso, es decir, deben representar un flujo de actividad completo desde el punto de vista de lo que un actor espera que el sistema haga por él, así como no utilizar estas herramientas sólo para descomponer un caso de uso de envergadura en otros más pequeños, piedra angular del diseño estructurado y no del orientado a objetos.

Autoevaluación

Supón el siguiente sistema que modela el caso de uso "Servir pedido" en el que el Empleado de almacén revisa si hay suficientes artículos para hacer el pedido y si todo es correcto, el pedido se embala y se envía:



¿Qué tipo de relación emplearías en el modelo del dibujo?

- ☐ Asociación.
- ☐ Generalización.
- ☐ Extends.
- ☐ Include.

5.1.4.- Elaboración de casos de uso.



Caso práctico

Después de analizar todos los elementos que formen un diagrama de casos de uso el equipo de Ada está preparado para hacer frente a su primer diagrama 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.

Se conoce como **diagrama frontera** al diagrama de casos de uso que incluye todos los casos de uso genéricos del sistema, que podrán ser desglosados después en nuevos diagramas de casos de uso que los describan si es necesario. Se especifica enmarcando los casos de uso en un recuadro, que deja a los actores fuera.



Finalmente revisamos toda la documentación que hemos generado.



Debes conocer

La elaboración de casos de uso no es un proceso inmediato, en la siguiente presentación tienes la descripción de como elaborar el diagrama de casos de uso del sistema con el que estamos trabajando. También tienes un enlace a la descripción del sistema que queremos modelar.

Revísalo con cuidado para comprender los conceptos que acabamos de ver.

[Resumen textual alternativo](#)

Documentación del caso de uso "Hacer pedido" en el Anexo III.

5.1.5.- Escenarios.



Caso práctico

Ada continua la investigación, junto con el equipo de BK programación, que una vez ha creado su primer diagrama de casos de uso, se da cuenta de que realmente es una herramienta muy útil a la hora de definir la funcionalidad de un sistema. Continuando con la investigación descubren una ventaja adicional, utilizando los flujos de eventos, pueden describir interacciones concretas de los actores con el sistema, estas interacciones son los escenarios.



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

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

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

Un posible escenario podría ser:

Realizar pedido de unos zapatos y unas botas.

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

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

5.2.- Diagramas de secuencia.



Caso práctico

María se ha dado cuenta de que los casos de uso permiten, de una manera sencilla, añadir información sobre qué hace el sistema, sin embargo por completa que se la descripción de la secuencia de eventos no permite incluir información útil, como los objetos que intervienen en las tareas, y como se comunican.



-Tendríamos que buscar la forma de representar como circula la información, que objetos participan en los casos de uso, qué mensajes envían, y en que momento, esto nos ayudaría mucho a completar después el diagrama de clases.

Como siempre, Ada tiene una solución.

-Tendremos que investigar los 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.



Reflexiona

Los diagramas de secuencia completan a los diagramas de casos de uso, ya que permiten al equipo de desarrollo hacerse una idea de qué objetos participan en el caso de uso y como interaccionan a lo largo del tiempo.

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



Caso práctico

Ada, orienta a su equipo:

-Bien, ¿qué nos haría falta para poder representar la interacción de los objetos que participan en el caso de uso a lo largo del tiempo?

-Alguna manera de representar los objetos, el paso del tiempo y el paso de mensajes ¿no?



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 línea 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.

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

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.



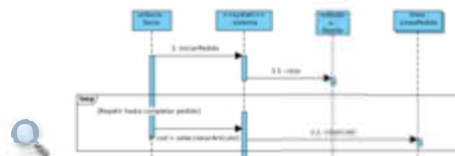
Invocación de métodos.

Los mensajes, que significan la invocación de métodos, se representan como flechas horizontales que van de una línea de vida a otra, indicando con la flecha la dirección del mensaje, los 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.



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.

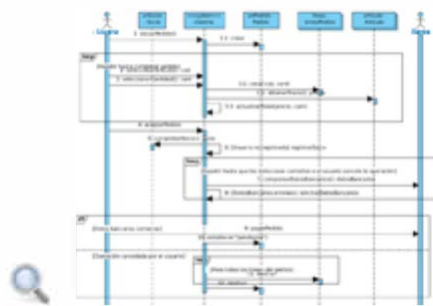


Por último destacar que se puede completar el diagrama añadiendo etiquetas y notas en el margen izquierdo que aclare la operación que se está realizando.

5.2.2.- Elaboración de un diagrama de secuencia.

Vamos a generar el diagrama de secuencia para el caso de uso **Hacer pedido**. En el se establece la secuencia de operaciones que se llevarán a cabo entre los diferentes objetos que intervienen en el caso de uso.

Este es el diagrama ya terminado, en el se han incluido todas las entidades (actores, objetos y sistema) que participan en el diagrama, y se han descrito todas las operaciones, incluidos los casos especiales, como es el registro de usuarios o la gestión de los datos bancarios. También incluye el modelado de acciones en bucle, como es la selección de artículos y de acciones regidas por condición, como es la posibilidad de cancelar el pedido si hay problemas con la tarjeta de crédito.



[Resumen textual alternativo](#)



Debes conocer

En la siguiente presentación puedes encontrar una descripción de como elaborar este diagrama con Visual Paradigm.

[Ver en pantalla completa](#)

Autoevaluación

¿Cual de estos elementos no forma parte de un diagrama de secuencia?

- ☐ Actor.
- ☐ Objeto.
- ☐ Bucle.
- ☐ Evento.


5.3.- Diagramas de colaboración.

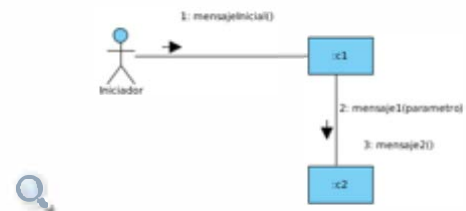


Caso práctico

-El diagrama de secuencia ha aportado información muy valiosa sobre la circulación de mensajes en los casos de uso, sin embargo estaría bien poder mostrar esta información de otra forma en la que se apreciase mejor el anidamiento de los mensajes, y el flujo de control entre objetos, ¿no creéis?



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 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 en 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.



Reflexiona

Los diagramas de colaboración y secuencia utilizan los mismo elementos pero distribuyéndolos de forma diferente, ¿crees que son semejantes?

Mostrar retroalimentación

5.3.1.- Representación de objetos.



Caso práctico

-De acuerdo, mientras investigamos los diagramas de colaboración vamos a ver con un poco más de detalle qué significa la notación que se asigna a los objetos, ¿qué diferencia hay entre usar los dos puntos o no hacerlo? ¿Podemos usar el nombre de una clase, solamente, o es obligatorio indicar el nombre del objeto?



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.

Clase

:objeto

:Clase

objeto:clase

- ✓ **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 nombra añadiendo la clase a la que pertenece.

5.3.2.- Paso de mensajes.

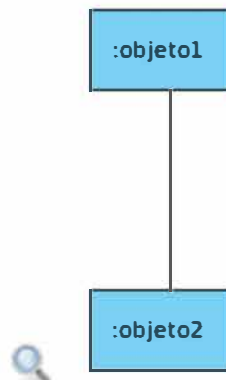


Caso práctico

-Y cuando enviamos un mensaje ¿cómo se representa exactamente?, ¿podemos incluir de alguna forma parámetros en los mensajes o valores devueltos? ¿Y si necesitamos indicar que el mensaje se enviará sólo si se cumple una determinada condición? ¿o que se envía dentro de un bucle?



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.



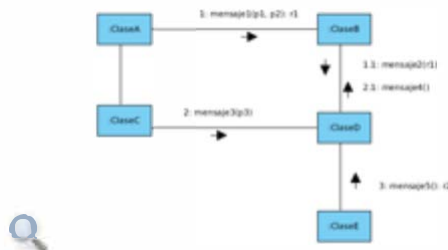
Un mensaje es la especificación de una comunicación entre objetos que transmite información y desencadena una acción en el objeto destinatario. La sintaxis de un mensaje es la siguiente:

```
[secuencia][*] [Condición de guarda] {valorDevuelto} : mensaje (argumentos)
```

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.

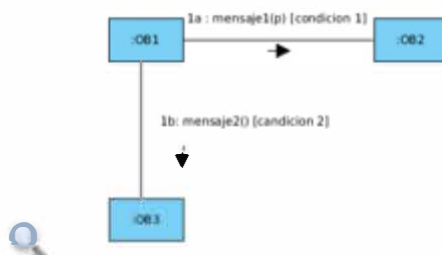
Como se ve en el ejemplo se puede usar la misma asociación para enviar varios mensajes. Vemos que hay dos mensajes anidados, el 1.1 y el 2.1, se ha usado el nombre de los mensajes para indicar el orden real en el que se envían.



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)



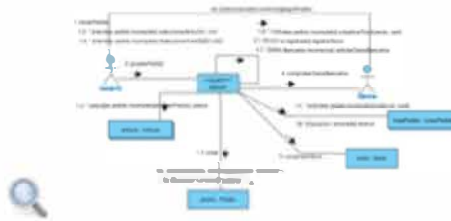
Cuando tenemos una condición se repite el número de secuencia y se añaden las condiciones necesarias, como vemos en la imagen según la condición se enviará el mensaje 1 o el 2, pero no ambos, por lo que coinciden en número de secuencia.

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.



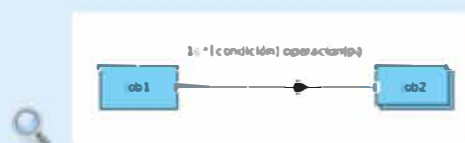
Resumen textual alternativo

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 de 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.

Autoevaluación

Indica qué afirmación no es correcta para el siguiente diagrama:



- ☐ El objeto ob2 es multiobjeto.
- ☐ Se envía un mensaje del objeto 1 al objeto 2.
- ☐ El mensaje operacion(pp) se ejecutará siempre.
- ☐ La operación se puede ejecutar varias veces.

5.4.- Diagramas de actividad.



Caso práctico


Por el momento el equipo de BK no ha tenido problema en seguir lo que Ada les cuenta sobre los diagramas UML. Antonio, que está verdaderamente interesado en el tema hace a Ada la siguiente pregunta:

-¿Que pasaría si quisiera representar sólo las acciones que tienen lugar, prescindiendo de quien las genera, solo el flujo de la actividad del sistema, qué pasa primero, qué ocurre después y qué cosas pueden hacerse al mismo tiempo?

-Pasaría que tendrías que hacer un diagrama 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 cuáles ocurren secuencialmente a lo largo del tiempo y cuáles se pueden llevar a cabo  concurrentemente. 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.



Reflexiona

¿Por qué decimos que el diagrama de actividades visualiza el comportamiento desde otro punto de vista del resto de diagramas?

Mostrar retroalimentación

5.4.1.- Elementos del diagrama de actividad.



Caso práctico

-Vale estoy preparado, ¿qué necesito para tener un diagrama de actividad?



Normalmente los diagramas de actividades contienen:

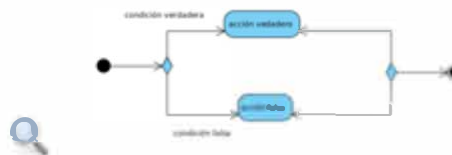
✓ **Estados** de actividad y estados de acción.

- ◆ **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.

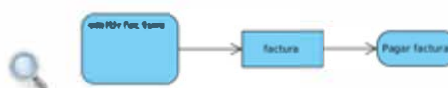


- ◆ **Fusión (Merge node):** Redirigen varios flujos de entrada en un único flujo de salida. No tiene tiempo de espera ni sincronización.
- ◆ **División (Fork node):** Permiten expresar la sincronización o ejecución paralela de actividades. Las actividades invocadas después de una división son concurrentes.



- ◆ **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.



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.

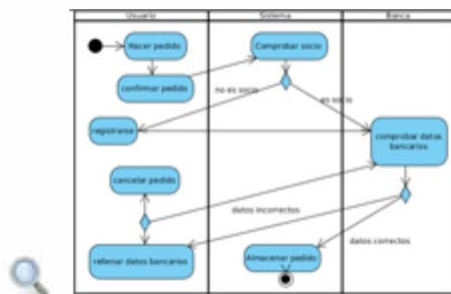
El siguiente diagrama de actividad representa el caso de uso hacer pedido, en el aparecen los elementos que hemos visto en las secciones anteriores.

- ✓ 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.



[Resumen textual alternativo](#)

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:



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

Autoevaluación

Los diagramas de actividades, a diferencia del resto, permiten incluir la concurrencia en la representación del diagrama.

- ☐ Verdadero.
- ☐ Falso.

5.5.- Diagramas de estados.



Caso práctico

Ada espera que su equipo continúe con tan buen ánimo para estudiar un tipo de diagrama más, que completará las diferentes visiones de la dinámica de un sistema que proporciona UML. Son los diagramas de estados, que les permitirán analizar cómo va cambiando el estado de los objetos que tienen una situación variable a lo largo del tiempo.

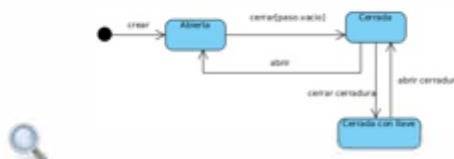


Basado en los statecharts de David Harel. Representan máquinas de estados (autómatas de estados finitos) para modelar el comportamiento dinámico basado en la respuesta a determinados eventos de aquellos objetos que requieran su especificación, normalmente por su comportamiento significativo en tiempo 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.



Autoevaluación

Analiza el diagrama de estados de la puerta, según está dibujado, ¿se puede abrir una puerta que está cerrada con llave directamente?

- ☐ Verdadero.
- ☐ Falso.

5.5.1.- Estados y eventos.



Caso práctico

Ada indica a su equipo que para entender bien la dinámica de un diagrama de estados deben comenzar por analizar sus componentes fundamentales: estados y eventos.



Un **estado** es una situación en la vida de un objeto en la que satisface cierta condición, realiza alguna actividad o espera algún evento.

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.

Un **evento** es un acontecimiento que ocupa un lugar en el tiempo y espacio que funciona como un estímulo que dispara una transición en una máquina de estados. Existen eventos externos y eventos internos según el agente que los produzca.

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.



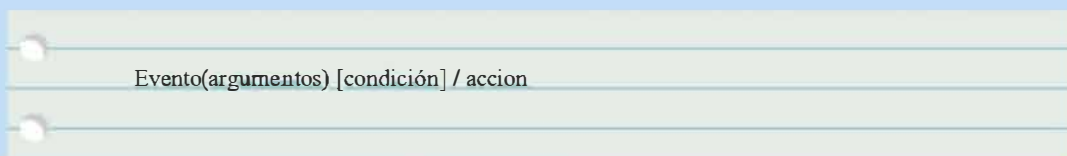
Caso práctico

-De acuerdo, los estados son situaciones específicas en las que se puede encontrar un objeto, y los eventos pueden hacer que un objeto cambie de estado, y, ¿cómo representamos eso?



Una **transición** de un estado A a un estado B, se produce cuando se origina el evento asociado y se satisface cierta condición especificada, en cuyo caso se ejecuta la acción de salida de A, la acción de entrada a B y la acción asociada a la transición.

La signatura de una transición es como sigue:



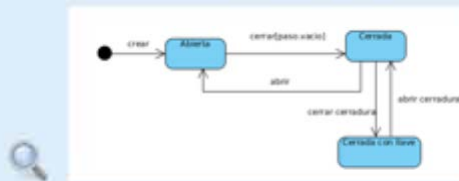
donde todos los elementos son opcionales.

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.

Autoevaluación

Recordemos el diagrama de estado de la puerta:



¿Qué significa la signatura de la transición "cerrar [paso.vacio]"?

- ☐ Que cuando cerremos la puerta el paso quedará vacío.
- ☐ Que para cerrar la puerta el paso debe estar vacío.
- ☐ Que cuando se está cerrado la puerta se vacía el paso.

5.5.3.- Creación de un diagrama de estados.

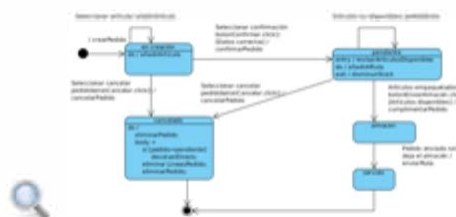
Para ejemplificar la creación de un diagrama de estados vamos a ver el que se asocia al objeto pedido, que cumple con las condiciones que hemos visto al principio, tiene un comportamiento significativo en tiempo real, ya que su situación tanto física, como el sistema, va evolucionando conforme pasa el 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:



Resumen textual alternativo

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.



Caso práctico

Ada está muy satisfecha con el cambio que percibe en su equipo, ahora, cuando tiene que enfrentarse a un proyecto nuevo se esfuerzan en escribir los requerimientos antes y comenzar con un proceso de análisis y creación de diagramas. No obstante, ahora le surge un reto nuevo, una empresa les ha contratado para reconstruir una aplicación que hay que adaptar a nuevas necesidades. Así que los chicos continúan investigando si pueden aplicar el uso de diagramas en este caso también.



La **ingeniería inversa** se define como el proceso de analizar código, documentación y comportamiento de una aplicación para identificar sus componentes actuales y sus dependencias y para extraer y crear una abstracción del sistema e información del diseño. El sistema en estudio no es alterado, sino que se produce un conocimiento adicional del mismo.

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.

Mostrar retroalimentación



Para saber más

Tienes una buena descripción teórica sobre ingeniería inversa en el siguiente documento:

 [Ingeniería Inversa.](#) (0.53 MB)

7.- Enlaces de refuerzo y ampliación.



Para saber más

- ✓ Tutorial sobre Diagramas de Casos de uso en la herramienta case Visual Paradigm I: <https://www.visual-paradigm.com/VPGallery/usecase/index.html>
- ✓ Tutorial sobre Diagramas de Casos de uso en la herramienta case Visual Paradigm II: <https://www.visual-paradigm.com/VPGallery/diagrams/UseCase.html>
- ✓ Tutorial sobre Diagramas de Secuencia en la herramienta case Visual Paradigm: <https://www.visual-paradigm.com/VPGallery/diagrams/Sequence.html>

Anexo I.- Descarga e instalación de Visual Paradigm.

Descarga e instalación de Visual Paradigm

Obtenemos los archivos desde:

[Página de Visual Paradigm](#)

Ofrece dos versiones:

- ✔ **Visual Paradigm for UML (VP-UML)**, versión de 30 días sin Registro
- ✔ **Versión Community-Edition**, para uso no comercial (gratuito).

La versión **Community-Edition** incluye algunas de las funcionalidades de la versión completa, entre las que no se encuentra la generación de código ni la ingeniería inversa.

Para la siguiente unidad también usaremos VP-UML, de modo que si fuera necesario tendríamos que conseguir un nuevo código de activación, esta vez de tipo Community-Edition.

Proceso de instalación

Ejecutaremos el archivo de instalación, que tendrá diferente extensión si es para Windows o para Linux. En nuestro caso suponemos que lo hacemos en un equipo con Ubuntu Desktop 10.10. Se debe tener en cuenta que en el nombre se incluye la versión y la fecha en la que apareció, por lo que estos datos pueden cambiar con el tiempo. Si hacemos la instalación en Windows bastará con hacer doble clic sobre el archivo .exe.

```
usuario@equipo:~/VP/ chmod +x VP_Suite_Linux_5_2_20110611.sh
usuario@equipo:~/VP/sudo ./VP_Suite_Linux_5_2_20110611.sh
```

Mostramos a continuación la instalación sobre un entorno Windows.

Ejecutamos el .exe que hemos descargado desde el enlace anterior.

Iniciar Visual Paradigm

Una vez realizada la instalación tendremos una entrada en el menú **Aplicaciones** llamada **Otras**, si trabajamos con Linux o bien una entrada de menú en el botón Inicio para Visual Paradigm, si es que trabajamos en Windows. En cualquiera de los casos para abrir la herramienta buscamos la opción Visual Paradigm 12

Tras la instalación elegiremos el tipo de licencia que queremos disfrutar durante 30 días, que en nuestro caso será la Enterprise

Evaluation license is provided for evaluation purposes only. Any commercial use is prohibited. The evaluation copy is fully functional except the output produced, such as images and images in reports, will be stamped with watermark.

Please select an edition to evaluate	Enterprise Evaluate	Professional Evaluate	Standard Evaluate	Modeler Evaluate
UML & SysML Toolset	●	●	●	●
BPMN Toolset	●	●	●	●
Collaborative Modeling	●	●	●	●
Glossary management	●	●	●	●
Code/Database Engineering	●	●	●	
Business Modeling	●	●	●	
Advanced Modeling Toolset	●	●	●	
Document Production	●	●	●	
Requirements Gathering Toolset	●	●		
Wireframe	●	●		
Task management	●	●		
Animated Software Design	●	●		
Impact Analysis	●	●		
SoaML Modeling	●	●		
Enterprise Architecture	●			
Business rule	●			
Process Simulation	●			

Back

[Visual Paradigm..](#) Versión de licencia (Todos los derechos reservados)

A continuación tendremos que indicar el Workspace que queremos utilizar.

Workspace Launcher

Select a workspace

A workspace is a directory where your Visual Paradigm projects and settings are stored. Please select the workspace directory to use.

Workspace :

☐ Use this as the default and do not ask again

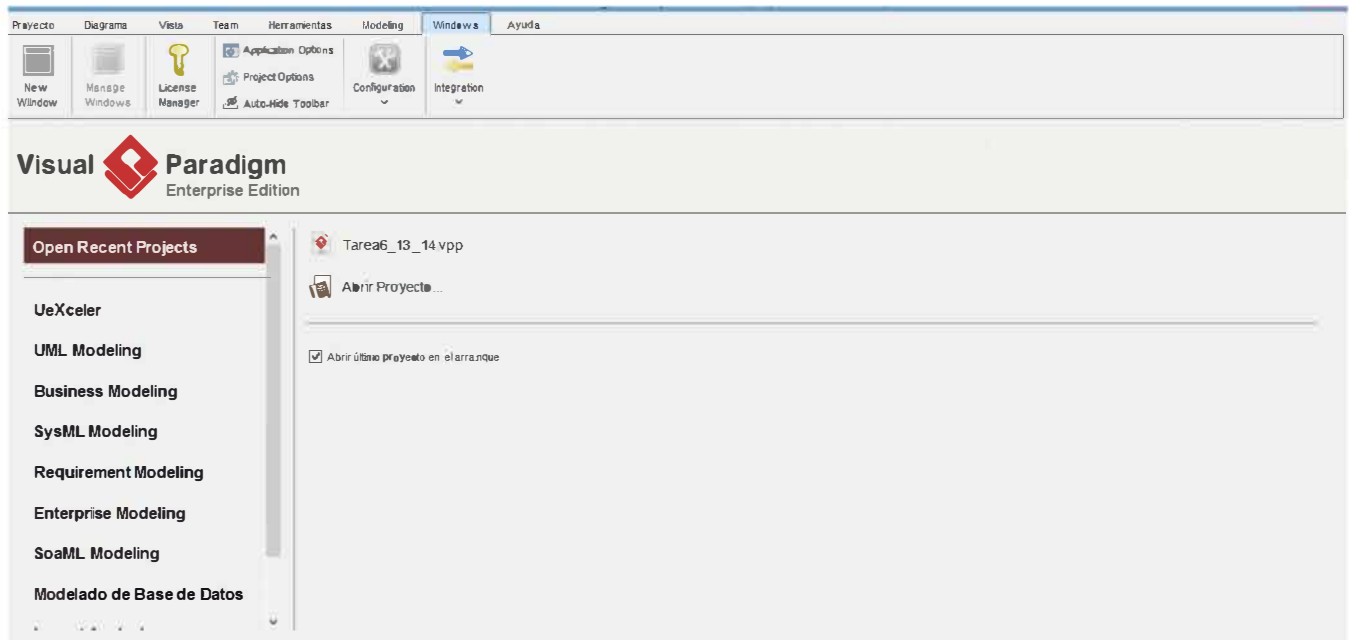
Import Workspace... OK Cancel Help

[Visual Paradigm..](#) WorkSpace (Todos los derechos reservados)

Integrar con Netbeans.

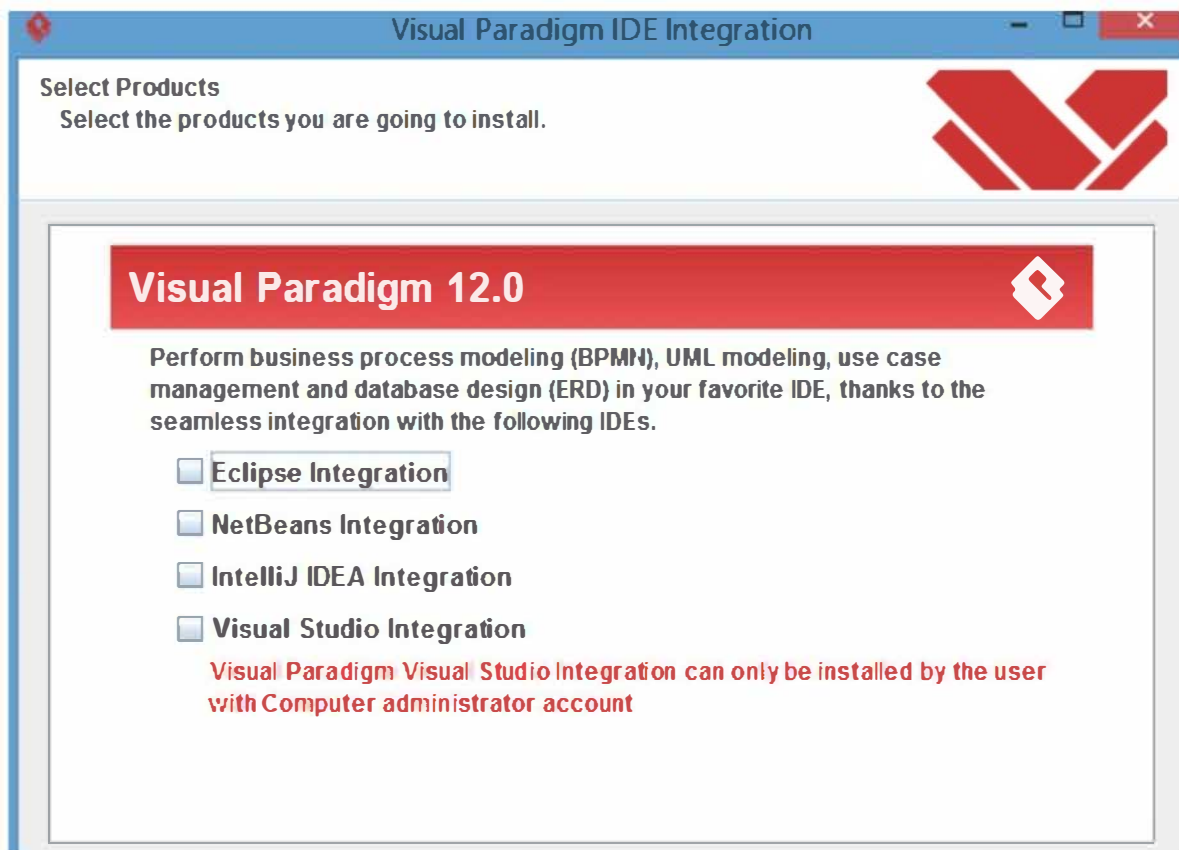
Tras instalar Visual Paradigm hemos de realizar la integración con Netbeans.

Para ello elegimos en la pestaña Windows, la subpestaña **Integration**, y la opción **Integration IDE**.



[Visual Paradigm](#). Integración (Todos los derechos reservados)

Tras ello, elegimos el IDE correspondiente, en nuestro caso **Netbeans**.



[Visual Paradigm..](#) Elección del IDE Netbeans (Todos los derechos reservados)

Se puede dar el problema de que indique que no se tienen permisos para la realización de la integración. Esto se soluciona ejecutando VP como administradores.

Al pulsar el siguiente nos indicará el directorio de Instalación de Netbeans, y, si estamos de acuerdo podremos comenzar la integración.

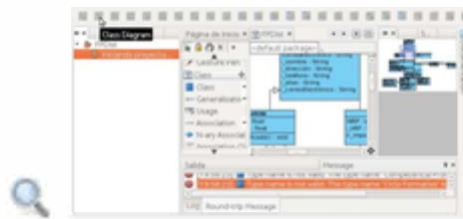
Cuando finalice la instalación, ya tenemos integrado VP en nuestro IDE Netbeans.

Ahora, en el menú emergente que nos ofrece para cada proyecto, dentro de Tools (Herramientas) disponemos de las Opciones de Visual Paradigm

Los proyectos de Visual Paradigm se podrán almacenar en el directorio por defecto, que se denomina vpproject y cuelga del directorio principal del proyecto NetBeans, o en otra ubicación. Nosotros nos quedaremos con la opción por defecto.

También podemos importar un proyecto VP-UML que tengamos ya creado seleccionándolo al crear el proyecto existente.

Una vez creado o importado el proyecto, tendremos una serie de botones en la zona superior derecha que nos permitirán crear los diferentes diagramas de UML, y que queden asociados al proyecto NetBeans.



Anexo II.- Generación del diagrama de clases de un problema dado.

Descripción del problema

El Ministerio de Educación ha encargado a **BK Programación** que desarrolle una plataforma de aprendizaje electrónico para que los alumnos de ciclos formativos a distancia tengan acceso a los materiales y puedan comunicarse con sus profesores. Para que los chicos puedan empezar a crear los primeros diagramas de la aplicación Ada les pasa la siguiente descripción del ámbito del problema:

“Los alumnos y alumnas de Ciclos Formativos a Distancia se matriculan de varios módulos formativos al año. Los módulos formativos son impartidos por profesores y profesoras que pondrán los contenidos del módulo a disposición de los alumnos y alumnas. Para superar un módulo hay que hacer una tarea y un examen que se calificarán de uno a diez, y sacar en ambos casos una puntuación superior a cinco. Los exámenes se componen de 30 preguntas que se eligen y ordenan al azar. Las preguntas tienen un enunciado y cuatro posibles respuestas, sólo una de ellas válida. Un ciclo formativo se compone de una serie de competencias profesionales, que tienen una descripción y que, a su vez, están formadas por uno o varios módulos, que tienen un nombre, y un número de horas. Cuando un alumno o alumna supera los módulos correspondientes a una capacidad se le certifica esa capacidad. Cuando se han superado todos los módulos (y por tanto se han adquirido todas las competencias profesionales) se aprueba el ciclo. Cuando un alumno o alumna finaliza el ciclo se emite un certificado de competencias a su nombre donde aparece la descripción de las competencias que forman el ciclo y la nota media obtenida. Si un alumno o alumna no termina de cursar el ciclo completo puede pedir un certificado que acredite las competencias que sí tenga adquiridas. El alumnado y el profesorado se identifican con un alias en el sistema y se comunican a través de correo electrónico. Por motivos administrativos es necesario conocer el nombre y apellidos, dirección completa y teléfono de todas las personas que participan en el sistema, sea como profesores o como alumnos. Para el profesorado, además, se debe conocer su número de registro personal (NRP)”

Extracción de los sustantivos de la descripción del problema.

Primero subrayamos los sustantivos de la descripción del problema (sin repeticiones) y los pasamos a una tabla: (usaremos sólo alumnos y sólo profesores para simplificar el diseño)

Los alumnos de Ciclos Formativos a Distancia se matriculan de varios módulos formativos al año. Los módulos formativos son impartidos por profesores que pondrán los contenidos del módulo a disposición de los alumnos. Para superar un módulo hay que hacer una tarea y un examen que se calificarán de uno a diez, y sacar en ambos casos una puntuación superior a cinco. Los exámenes se componen de 30 preguntas que se eligen y ordenan al azar. Las preguntas tienen un enunciado y cuatro posibles respuestas, sólo una de ellas válida. Un ciclo formativo se compone de una serie de competencias profesionales, que tienen una descripción y que, a su vez, están formadas por uno o varios módulos, que tienen un nombre, y un número de horas. Al sumar las horas de un ciclo obtenemos las horas del módulo. Cuando un alumno supera los módulos correspondientes a una competencia se le certifica esa competencia. Cuando se han superado todos los módulos (y por tanto se han adquirido todas las competencias profesionales) se aprueba el ciclo. Cuando un alumno finaliza el ciclo se emite un certificado de competencias a su nombre donde aparece la descripción de las competencias que forman el ciclo y la nota media obtenida. Si un alumno no termina de cursar el ciclo completo puede pedir un certificado que acredite las competencias que si tenga adquiridas. Los alumnos y profesores se identifican con un alias en el sistema y se comunican a través de correo electrónico. Por motivos administrativos es necesario conocer el nombre y apellidos, dirección completa y teléfono de todas las personas que participan en el sistema, sea como profesores o como alumnos. Para los profesores, además, se debe conocer su número de registro personal (NRP).

Tabla de sustantivos

Clase/objeto potencial	Categoría
Alumno	Entidad externa o rol
Ciclo Formativo a Distancia	Unidad organizacional
Modulo Formativo	Unidad organizacional
Año	Atributo
Profesor	Entidad externa o rol
Contenidos	Atributo
Tarea	Cosa
Examen	Cosa
Uno	Atributo
Diez	Atributo
Pregunta	Cosa
Enunciado	Atributo
Respuesta	Atributo
Competencia Profesional	Unidad organizacional

Descripción	Atributo
Horas	Atributo
Certificado de competencias	Cosa
Nombre	Atributo
Nota media	Atributo
Alias	Atributo
Sistema	Estructura
Nombre	Atributo
Dirección	Atributo
Teléfono	Atributo
Persona	Rol o entidad externa
Número de registro personal	Atributo

Selección de sustantivos como objetos/clases del sistema

Ahora aplicamos los criterios de selección de objetos. En este apartado es necesario destacar que aunque algunos de los sustantivos que tenemos en el enunciado podrían llegar a convertirse en clases y objetos, como los contenidos de un módulo formativo, se descartan en esta fase porque el enunciado no da suficiente información. El proceso de creación de diagramas no es inmediato, sino que está sujeto a revisiones, cambios y adaptaciones hasta tener un resultado final completo.

Tabla de elección de sustantivos como objetos o clases del sistema.

Clase/objeto potencial	Criterios aplicables
Alumno	2,3,4
Ciclo Formativo a Distancia	1,2,3
Módulo Formativo	1,2,3
Profesor	2,3,4
Tarea	1,2,3
Examen	1,2,3
Competencia Profesional	1,2,3
Pregunta	1,2,3
Certificado de competencias	Falla 2,3 rechazado
Sistema	Falla 1,2,3,4 rechazado
Persona	2,3,4

Obtención de los atributos de los objetos.

Buscamos responder a la pregunta ¿Qué elementos (compuestos y/o simples) definen completamente al objeto en el contexto del problema actual?

Tabla de relación de las clases u objetos con sus atributos.

Clase/objeto potencial	Atributos
Alumno	Nombre, dirección, teléfono, alias, correo electrónico.
Ciclo Formativo a Distancia	Nombre, descripción, horas.
Módulo Formativo	Modulo Formativo
Profesor	Nombre, dirección, teléfono, alias, correo electrónico , NRP.
Tarea	Descripción, calificación.
Examen	Descripción, calificación.
Competencia Profesional	Nombre, descripción.
Pregunta	Enunciado, respuestas, respuesta válida.
Persona	Nombre, dirección, teléfono, alias, correo electrónico.

Obtención de los métodos

Buscamos o inferimos en el enunciado verbos, y actividades en general que describan el comportamiento de los objetos o modifiquen su estado.

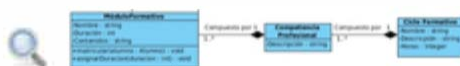
Tabla de clases u objetos del sistema con sus posibles métodos.

Clase/objeto potencial	Métodos
Alumno	CalcularNotaMedia() : void emitirCertificado() : void
Ciclo Formativo a Distancia	
Módulo Formativo	Matricular(Alumno : alumno) : void asignarDuracion(horas: int) : void
Profesor	
Tarea	
Examen	Calificar() añadirPregunta() ordenarPreguntas() crearExamen()
Competencia Profesional	
Pregunta	
Persona	

Obtener relaciones

Con las clases ya extraídas y parcialmente definidas (aún faltan por añadir métodos y atributos inferidos de posteriores refinamientos y de nuestro conocimiento) podemos empezar a construir relaciones entre ellas.

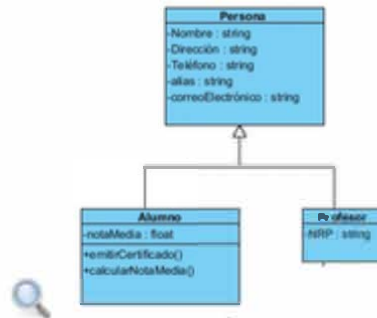
Comenzaremos por las clases que hacen referencia a la estructura de los Ciclos, cada Ciclo se compone de una o más competencias profesionales, que no tienen la capacidad de existir por si mismas, es decir, la competencia no tiene sentido sin su ciclo, por lo que vamos a crear una relación entre ambas clases de composición. De igual manera una competencia profesional se compone de un conjunto de módulos formativos (1 o más) por lo que relacionaremos ambas, también mediante composición.



Un módulo formativo a su vez, contiene un examen y una tarea, que tampoco tienen sentido por si mismos, de modo que también los vamos a relacionarlos mediante composición. El examen por su parte se compone de 30 preguntas, pero éstas pueden tener sentido por si mismas, y pertenecer a diferentes exámenes, además, el hecho de eliminar un examen no va a dar lugar a que las preguntas que lo forman se borren necesariamente, si leemos con atención el enunciado, podemos deducir que las preguntas se seleccionan de un repositorio del que pueden seguir formando parte [... [Los exámenes se componen de 30 preguntas que se eligen y ordenan al azar...], así que en este caso usaremos la relación de agregación para unirlos.



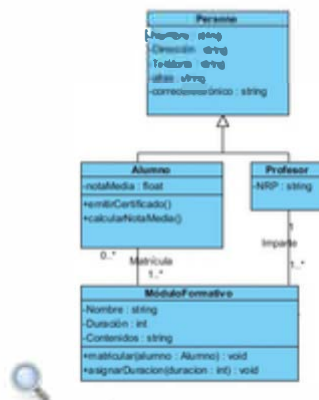
Por otra parte alumnos y profesores comparten ciertas características, por necesidad del sistema, como son los datos personales, o el correo electrónico, esto induce a pensar que podemos crear una abstracción con los datos comunes, que de hecho, ya hemos obtenido del enunciado en la clase persona, que recoge las coincidencias entre alumnos y profesores y añadir una relación de herencia de la siguiente manera:



Por último queda relacionar a alumnos y profesores con los módulos formativos. Un alumno se matricula de un conjunto de módulos formativos, y un profesor puede impartir uno o varios módulos formativos.

Más concretamente, de cara a la cardinalidad, un alumno puede estar matriculado en uno o varios módulos, mientras que un módulo puede tener, uno o varios alumnos matriculados. Por su parte un profesor puede impartir uno o varios módulos, aunque un módulo es impartido por un profesor.

Éste análisis da como resultado lo siguiente:



Añadir Getters, Setters y constructores

Por último añadimos los métodos que permiten crear los objetos de las clases (constructores) así como los que permiten establecer los valores de los atributos no calculados y leerlos (getters y setters), recuerda que para tener éstos métodos completos es necesario que el atributo tenga establecido su tipo, para que sea tenido en cuenta.

Para añadir los getters y setters en Visual Paradigm, basta con desplegar el menú contextual del atributo y seleccionar la opción Create Getter and Setter.

También hay que añadir los métodos que no se infieren de la lectura del enunciado, por ejemplo los que permiten añadir módulos a las competencias, o competencias a los ciclos. Para comprobar estos métodos puedes descargar el diagrama de clases en un proyecto VP-UML un poco más adelante.

Añadir documentación

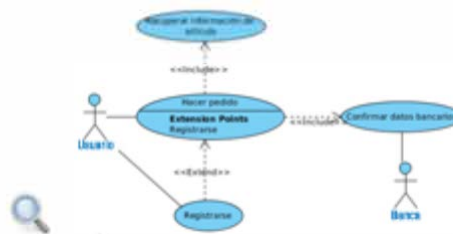
Por último se debe rellenar la documentación de cada clase, atributo y método con una descripción de los mismos que será necesaria para la generación de informes posterior. A continuación se listan las clases con su documentación:

- ✔ **Persona:** Generalización para agrupar las características comunes de alumnos y profesores como personas que interactúan con el sistema. De una persona interesa conocer su nombre, dirección, teléfono, alias y correo electrónico.
- ✔ **Alumno:** Es un tipo de persona. Representa a las personas que se matriculan de un ciclo formativo. Un alumno puede estar matriculado durante varios años en un ciclo. Está matriculado de un ciclo siempre que está matriculado en algún módulo que forme parte del ciclo. Para aprobar un Ciclo hay que superar todos los módulos que lo componen. Para superar un módulo hay que realizar la tarea y aprobar el examen, que está compuesto de 30 preguntas de tipo test, con cuatro respuestas posibles una de las cuales es la correcta. De un alumno interesa almacenar su nota media.
- ✔ **Profesor:** Es un tipo de persona. Representa a las personas que imparten los módulos formativos. Evalúan las tareas que realizan los alumnos y se encargan de poner los contenidos a disposición de los alumnos. De un profesor interesa almacenar su número de registro personal.
- ✔ **Ciclo Formativo a Distancia:** Es uno de los núcleos centrales del sistema. Representa los estudios que se pueden realizar a distancia. Un ciclo formativo se compone de un conjunto de competencias profesionales que se componen a su vez de módulos formativos. Se aprueba un ciclo formativo cuando se adquieren todas las competencias que lo forman. De un ciclo formativos se almacena su nombre, descripción y horas totales.
- ✔ **Competencia Profesional:** Representan el conjunto de conocimientos generales que se adquieren cuando se completa un ciclo formativo. Se componen de módulos profesionales y se adquiere una competencia cuando se superan los módulos que la componen. De una competencia se almacena su descripción.
- ✔ **Módulo Formativo:** Unidades formativas que cursa un alumno. Un módulo formativo tiene una serie de contenidos que el alumno debe estudiar, y una tarea y un examen que el alumno debe hacer. Cuando se aprueban la tarea y el examen se supera el módulo. De un módulo se almacena su nombre, duración y contenidos.
- ✔ **Tarea:** Actividad relacionada con los contenidos de un módulo que debe realizar un alumno. Tiene una puntuación de uno a diez y es evaluada por el profesor. La nota se asigna a cada alumno para la matrícula del módulo al que pertenece la tarea. De una tarea se almacena su descripción.
- ✔ **Examen:** Conjunto de treinta preguntas que se evalúa de uno a diez. Un examen puede desordenarse y calificarse calculando cuantas preguntas son correctas.
- ✔ **Pregunta:** Forman los exámenes de un módulo. Se compone del enunciado y cuatro posibles respuestas de las cuales sólo una es válida.

Anexo III.- Generación de la documentación de un caso de uso

Como se indicaba en los contenidos del apartado 5.1.2, lo más importante en la elaboración de un diagrama de casos de uso, no es el diagrama en sí, sino la documentación de los casos de uso que es lo que permitirá desarrollar otros diagramas que ayuden en la codificación del sistema, y la elaboración de los casos de prueba de caja negra.

A modo de ejemplo vamos a desarrollar la documentación del caso de uso **Hacer Pedido**, ya que, por su complejidad abarca todos los apartados que hemos visto. El ejemplo se hará con la herramienta Visual Paradigm for UML, aunque tu puedes usar la herramienta que consideres más oportuna. Recordamos el aspecto del caso de uso:



Los datos que debemos incluir para elaborar la documentación del caso de uso, el contrato, eran:

- ✓ **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.
- ✓ **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.

Para incluir el nombre, actores, propósito, precondiciones y postcondiciones abrimos la especificación del caso de uso. Esto da lugar a la aparición de una ventana con un conjunto de pestañas que podemos rellenar:

- ✓ En la pestaña "**General**" rellenamos el nombre "**Hacer pedido**", y tenemos un espacio para escribir una breve descripción del caso de uso, por ejemplo:
"El cliente visualiza los productos que están a la venta, que se pueden seleccionar para añadirlos al pedido. Puede añadir tantos artículos como desee, cada artículo añadido modifica el total a pagar según su precio y la cantidad seleccionada."

Cuando el cliente ha rellenado todos los productos que quiere comprar debe formalizar el pedido.

En caso de que el cliente no sea socio de la empresa antes de formalizar la compra se le indica que puede hacerse socio, si el cliente acepta se abre el formulario de alta, en caso contrario se cancela el pedido.

En caso de que se produzca algún problema con los datos bancarios se ofrecerá la posibilidad de volver a introducirlos.

Al finalizar un pedido se añade al sistema con el estado pendiente."

- ✓ En la pestaña "**Valores etiquetados**" encontramos un conjunto de campos predefinidos, entre los que se encuentran el autor, precondiciones y postcondiciones, que podemos rellenar de la siguiente manera:

Autor: *usuario.*

Precondiciones: *Existe una campaña abierta con productos de la temporada actual a la venta.*

Postcondiciones: *Se ha añadido un pedido con un conjunto de productos para servir con el estado "pendiente" que deberá ser revisado.*

También se pueden incluir otros datos como la complejidad, el estado de realización del caso de uso la complejidad que no hemos visto en esta unidad.

Para incluir el resto de los datos en el caso de uso hacemos clic en la opción "**Open Use Case Details...**" del menú contextual, lo que da lugar a la aparición de una ventana con una serie de pestañas. En ellas se recupera la información de la especificación que hemos rellenado antes, además, podemos rellenar el flujo de eventos del caso de uso, en condiciones normales usaríamos la pestaña "Flow of events", sin embargo como esta opción solo está disponible en la versión profesional, utilizaremos la pestaña "**Descripción**", que está disponible en la versión community. Para activarla pulsamos el botón "**Create/Open Description**".

Podemos añadir varias descripciones de diferentes tipos, pulsando el botón "**Nuevo**". Para añadir filas al flujo de eventos pinchamos en el botón. En principio añadimos la descripción principal, luego añadiremos otras alternativas.

Esta es la descripción principal, en ella se describe el flujo normal de eventos que se producen cuando se ejecuta el caso de uso sin ningún problema.

Flujo de eventos normal para el caso de uso Hacer Pedido.

Use Case	Hacer pedido
Author	usuario
Date	26-ago-2011 13:56:56

Brief Description	EL usuario selecciona un conjunto de artículos, junto con la cantidad de los mismos, para crear el pedido. Cuando se formaliza se comprueba que el usuario sea socio. A continuación se comprueban los datos bancarios, se realiza el cobro y se crea el pedido.		
Preconditions	Existe un catálogo de productos disponibles para pedir. El usuario está registrado. Los datos bancarios son correctos.		
Post-conditions	Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.		
		Actor Input	System Response
	1	Inicia el pedido.	
	2		Se crea un pedido en estado "en construcción".
	3	Selecciona un artículo.	
	4	Selecciona una cantidad.	
	5		Recupera la información del artículo para obtener el precio y modifica el precio total del pedido.
Flow of Events	6	El proceso se repite hasta completar la lista de artículos.	
	7	Se acepta el pedido.	
	8		Se comprueba si el usuario es socio, si no lo es se le muestra un aviso para que se registre en el sitio.
	9		Se comprueban los datos bancarios con una entidad externa.
	10		Se genera: calcula el total, sumando los gastos de envío.
	11		Se realiza el pago a través de la entidad externa.
	12		Se almacena la información del pedido con el estado "pendiente".

[Resumen textual alternativo](#)

Añadimos un par de descripciones alternativas para indicar que hacer cuando el usuario no es socio y cuando los datos bancarios no son correctos:

Flujo alternativo para el caso de uso Hacer Pedido cuando el usuario no está registrado.

Author	usuario				
Date	26-ago-2011 17:56:35				
Brief Description	Cuando el usuario hace un pedido si no está registrado se abre la opción de registro para que se dé de alta.				
Preconditions	El usuario no está registrado. Existe un catálogo de artículos para hacer pedido. Los datos bancarios son correctos.				
Post-conditions	El usuario queda registrado. Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.				
				Actor Input	System Response

1		Inicia el pedido.			
2			Se crea un pedido en estado "en construcción".		
3	Flow of Events	Selecciona un artículo.			
4		Selecciona la cantidad.			
5			Recupera la información del artículo para obtener su precio y modifica el precio total a pagar por el pedido.		
6			Añade la información al pedido en creación.		
7		EL proceso se repite hasta completar la lista de artículos del pedido.			
8		Acepta el pedido.			
9			Se comprueba si el usuario es socio.		
10			Se invoca el registro de usuario.		
11			Se comprueban los datos bancarios con una entidad externa.		
12			Se genera: calcula el total, sumando los gastos de envío.		
13			Se realiza el pago a través de la entidad externa.		

Flujo alternativo para hacer pedido cuando los datos bancarios no son correctos.

[illegible]

1		Inicia el pedido.				
2			Se crea un pedido en estado "en construcción".			
3		Selecciona un artículo.				
4		Selecciona una cantidad.				
5	Flow of Events		Recupera la información del artículo para obtener el precio y modifica el precio total del pedido.			
6		El proceso se repite hasta completar la lista de artículos.				
7		Se acepta el pedido.				
8		Acepta el pedido.				
9			Se comprueban los datos bancarios con una entidad externa, fallando la comprobación.			
10			Se solicitan los datos de nuevo.			

11		Introduce los datos de nuevo.		
12			Se repite el proceso hasta que se acepten los datos bancarios o se cancele la operación.	
		13		Se genera: calcula el total, sumando los gastos de envío.
14			Se realiza el pago a través de la entidad externa.	
15		Se almacena la información del pedido con el estado "pendiente".		

[Resumen textual alternativo](#)

El resto de casos se uso se documentan de forma similar hasta completar la descripción formal de la funcionalidad del sistema.

Condiciones y términos de uso de los materiales

Materiales desarrollados inicialmente por el Ministerio de Educación, Cultura y Deporte y actualizados por el profesorado de la Junta de Andalucía bajo licencia Creative Commons **BY-NC-SA**.



Antes de cualquier uso leer detenidamente el siguiente [Aviso legal](#)

Historial de actualizaciones

Versión: 03.00.02	Fecha subida: 11/03/18	Autoría: Sonia Amate Garrido
<p>Ubicación: 3.2.- Herramientas para la elaboración de diagramas UML.</p> <p>Mejora (tipo 1): La herramienta Rational Systems Developer de IBM está obsoleta desde el 2010 como se puede comprobar en https://www-01.ibm.com/software/rational/support/lifecycle/index_by_eos.html y el enlace del para Saber Más no funciona. Sustituir por la herramienta actual de IBM</p> <p>Ajustes realizados en la versión</p> <hr/> <p>Gloria Ortiz</p> <p>Mejora: Corrección en el formato del SCORM.</p>		
Versión: 03.00.01	Fecha subida: 15/03/17	Autoría: Francisco José Gutiérrez Martínez
<p>Ubicación: 7.- Enlaces de refuerzo y ampliación.</p> <p>Mejora (tipo 1): Se añade un nuevo punto, el 7 con enlaces de refuerzo y ampliación con explicaciones prácticas de como realizar diagramas de clases y casos de uso en Visual Paradigm.</p> <p>Ubicación: Todas las páginas.</p> <p>Mejora (tipo 1): Se suprime el pie de página en el elp, pues no debe aparecer en los contenidos, y se genera nuevamente el SCORM</p> <p>Ubicación: No especificada.</p> <p>Mejora (tipo 1): Añadir ejercicios resueltos de realización de diagramas de clases en Visual Paradigm. Explicar mejor los diagramas de casos de clase.</p>		

Versión: 03.00.00	Fecha subida: 28/06/16	Autoría: Aida López Túnez
Ubicación: Todo Mejora (tipo 1): Añadir licencias directamente a las imágenes y quitar Anexo de licencias. Ubicación: Todo Mejora (tipo 1): Arreglar glosario. Ubicación: No especificada. Mejora (tipo 3): Se unifica la anterior unidad 5 y 6		
Versión: 02.00.02	Fecha subida: 11/03/16	Autoría: Silvia Castillo Miranda
Ubicación: No especificada. Mejora (tipo 1): Se suprime el pie de página y otros metadatos y se genera nuevamente el SCORM.		
Versión: 02.00.01	Fecha subida: 10/03/16	Autoría: Francisco José Gutiérrez Martínez
Ubicación: Al inicio, se ha puesto la licencia correctamente tal y como establece el procedimiento de Actualiza Mejora (tipo 1): Se ha puesto la licencia correctamente tal y como establece el procedimiento de Actualización de Materiales. Se añade un nuevo punto, el 5 con enlaces de refuerzo y ampliación.		
Versión: 02.00.00	Fecha subida: 23/02/15	Autoría: Antonio José López Jiménez
Actualización de Unidad y Ejemplos a V12. Cambio de procedimientos segun nuevas versiones. Actualización de Glosario y Corrección de erratas		
Versión: 01.00.01	Fecha subida: 26/05/14	Autoría: Antonio José López Jiménez
Mejora: <i>No especificada.</i>		
Versión: 01.00.00	Fecha subida: 21/04/14	Autoría: MECD MECD
Mejora: <i>No especificada.</i>		