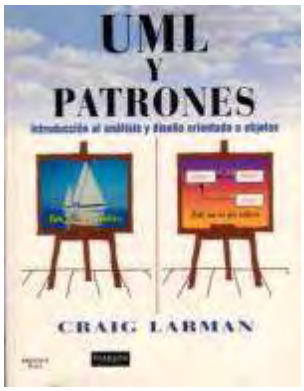
Este documento ha sido generado para facilitar la impresión de los contenidos.
Los enlaces a otras páginas no serán funcionales.

CASO:

Víctor está completamente hecho un lío. Lleva unas semanas aprendiendo cómo hacer diagramas UML, pero no tiene idea de cómo aplicarlos a un caso real. Víctor se queja de que UML no define los pasos a seguir para poder realizar el desarrollo orientado a objetos de un problema. Claro, le dice Carmen, ya que UML sólo es un lenguaje de modelado.

Carmen se acuerda que cuando estudió el módulo de Análisis en el IES Aguadulce, su profesor José Ramón les enseño un proceso a seguir para llevar a cabo el desarrollo orientado a objetos. Este proceso utilizaba UML como lenguaje de modelado y además ese mismo proceso es el que se sigue en SI Andalucía para abordar todos los proyectos que les salen. Como Carmen guarda todos los apuntes del ciclo, le ha propuesto a Víctor que le eche un vistazo y le pregunte las dudas que le puedan surgir, para que luego puedan aplicarlo a un problema sencillo y así que Víctor aprenda la forma en que se deberían hacer las cosas.



¿Qué entiendes por **proceso de desarrollo orientado a objetos**? Pues, un proceso de desarrollo de software es un **método** para organizar las actividades relacionadas con la creación, presentación y mantenimiento de los sistemas de software. A la hora de desarrollar un sistema software es necesario conocer un **lenguaje de programación**, pero eso no basta (el hábito no hace al monje) si lo que nos interesa es desarrollar sistemas robustos y fácilmente mantenibles. Para que nuestro sistema no se desmorone, debemos analizar cuidadosamente el problema y diseñar la solución desde la perspectiva de los objetos.

Como vimos en las unidades anteriores, UML es un **lenguaje de modelado** que nos sirve para expresar las ideas de análisis y diseño orientado a objetos, pero no define una metodología a seguir para lograr nuestros objetivos: un sistema sólido y de calidad, fácil de mantener y que cumpla con los requerimientos. Si somos capaces de seguir un proceso de desarrollo que defina cómo realizar el análisis y el diseño orientado a objetos, entonces seremos capaces de **planificar la construcción** de cualquier sistema y la probabilidad de obtener sistemas de calidad será bastante alta, sobre todo cuando nuestro equipo esté formado por varios desarrolladores, como suele ser común.

Uno de los procesos de desarrollo más extendidos en el desarrollo orientado a objetos es el que propone **Craig Larman**, uno de los gurús del desarrollo orientado a objetos, en su libro "UML y Patrones", editado por Prentice-Hall en el año 1999. El proceso propuesto define una serie de pasos (o patrones como los define Craig Larman) que pueden ser llevados a cabo en cada fase. Además es un proceso dirigido por los casos de usos y un proceso iterativo. Veamos que significa todo esto.

Desarrollo orientado a objetos usando uml y herramientas case

Proceso dirigido por los casos de uso

Para que cualquier proceso de **construcción** de un sistema pueda llevarse a cabo con éxito es necesario que sepamos:

- "¿qué?" pretendemos de nuestro sistema,
- "¿qué?" esperan los diferentes actores que interactúan con el sistema.

Como vimos en las dos unidades anteriores, un **caso de uso** describe una parte de la funcionalidad del sistema. Los casos de uso nos muestran cómo los actores de nuestro sistema esperan que se comporte el mismo. Por tanto, los casos de uso podrían ser un buen instrumento para especificar el "¿qué?" de nuestro sistema.

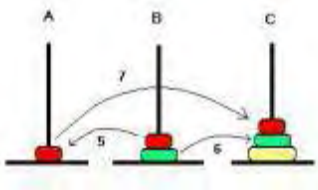
- Pero, **los casos de uso no sólo nos servirán para especificar requisitos del sistema,**
- **sino que también nos guiarán en todo el proceso de desarrollo.**
- Se irán **definiendo casos de uso y los desarrolladores diseñarán e implementarán modelos** que cumplan con esos casos de uso.
- Hecho esto, los encargados de realizar las pruebas **comprobarán que efectivamente el modelo propuesto cumple con dichos casos de uso.**

Por tanto, los casos de uso no solo inician el proceso de desarrollo, sino que permanecen unidos a él durante todo su ciclo de vida. Los casos de uso se especifican, se diseñan y son la base a partir de la cual los probadores construyen los casos de prueba.

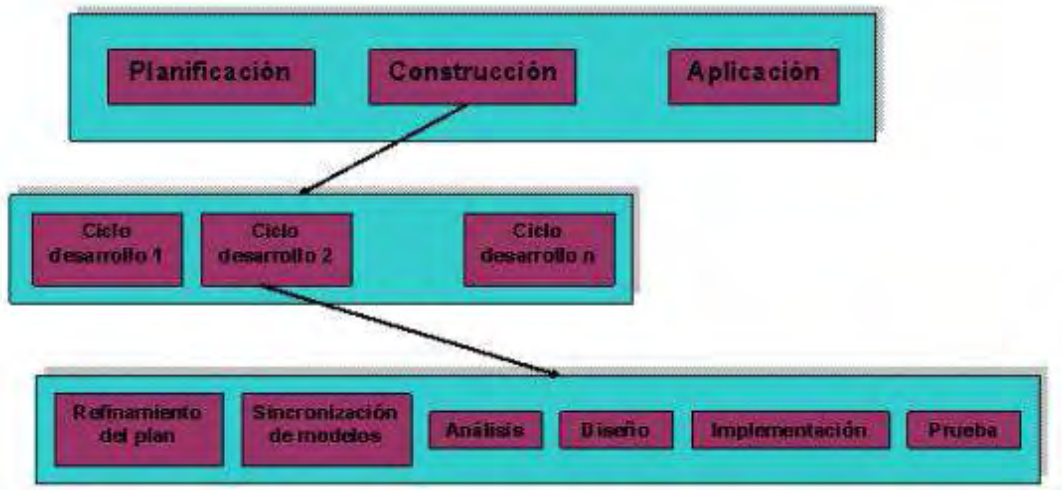
Desarrollo orientado a objetos usando uml y herramientas case

Proceso iterativo

Cuando hablamos de proceso **iterativo** estamos diciendo que es un proceso que en cierto modo repite varias fases del mismo, es decir, que **se basa en el crecimiento y perfeccionamiento secuencial de un sistema a través de múltiples ciclos de desarrollo**. El sistema va creciendo conforme incorporamos nuevas funcionalidades en cada ciclo de desarrollo.



Larman propone tres pasos de alto nivel: planificación, construcción y aplicación (comenzar a usar el sistema). El proceso de construcción será el proceso iterativo en sí y se puede subdividir en varios subprocesos como muestra la siguiente imagen.



Al abordar en cada ciclo un conjunto relativamente pequeño de requerimientos, el proceso iterativo nunca nos abrumará por la complejidad del problema, ya que en cada ciclo tratamos una parte del mismo.

Por tanto, nuestro proceso de desarrollo orientado a objetos constará de una fase **inicial** de planificación y especificación de requisitos, y de una o varias fases de **construcción**. En los siguientes

apartados nos dedicaremos a analizar estas dos fases en el proceso de desarrollo orientado a objetos.

Para saber más

En este enlace podrás encontrar una presentación sobre otro proceso de desarrollo orientado a objetos denominado RUP.

El proceso de desarrollo RUP

http://www.dsic.upv.es/~letelier/pub/p16.ppt [versión en cache]

Autoevaluacion

1 Señala la respuesta correcta en relación a un proceso de desarrollo orientado a objetos.

a) Es un proceso dirigido por los casos de uso y recursivo.
b) El proceso como tal no existe, debemos seguir las directrices que nos marca UML.
c) Es un proceso iterativo y dirigido por los casos de uso que utiliza UML como lenguaje de modelado.
d) Todas son ciertas.

Comprobar

Desarrollo orientado a objetos usando uml y herramientas case

CASO:

Víctor está impaciente por saber cómo abordar ese proceso de desarrollo tan útil según Carmen. Después de todo el tiempo invertido en el aprendizaje del lenguaje UML, ahora no quiere que ese tiempo caiga en saco roto, por lo que quiere aprender todo lo referente a dicho proceso. Ya le ha echado un vistazo a los apuntes que le ha dejado Carmen y sabe que el proceso se compone de varias fases. Así que va a empezar con la primera fase que según Carmen es bastante importante.

¿Qué entiendes por **planificación**? ¿Y por especificación de **requisitos**?

La fase de **planificación y especificación de requisitos** no se diferencia mucho cuando hablamos de desarrollo orientado a objetos o desarrollo estructurado. En ambos casos se trata de identificar **qué queremos** que haga nuestro sistema. Si queremos que el desarrollo de cualquier sistema vaya bien, sea robusto, es primordial comprender las funcionalidades que nuestro sistema debe satisfacer. Si no entendemos cómo queremos que se comporte nuestro sistema, difícilmente podremos diseñar una solución.

Centrándonos en el desarrollo orientado a objetos, esta fase conllevará las siguientes actividades fundamentales:

- **Definición de los requisitos**.
- **Descripción de los casos de uso**.

Autoevaluación

1 La fase de planificación en un proceso de desarrollo orientado a objetos ...

a) Trata de definir cómo nuestro sistema hará las cosas.
b) Trata de definir qué debe hacer nuestro sistema.
c) En el desarrollo orientado a objetos no existe fase de planificación.
d) La fase de planificación no es muy importante y generalmente la podemos obviar.

Comprobar

Desarrollo orientado a objetos usando uml y herramientas case

Definición de los requisitos

La especificación o definición de los requisitos que el sistema debe cumplir es una actividad común en cualquier proceso de desarrollo. Se trata de conocer qué se espera que el sistema realice.

El objetivo de todo el proceso de desarrollo es el **modelado** de un sistema real, mediante un sistema software. Debes conocer perfectamente cuáles son las **necesidades** de nuestro sistema o lo que es lo mismo, cuáles son los requerimientos de nuestro sistema, si quieres llevar a buen puerto este proceso de desarrollo que acabas de comenzar.

En esta fase nos interesa **listar los requerimientos del sistema** y numerarlos, ya que en fases posteriores haremos referencias a estos requerimientos o requisitos.

Estos requerimientos surgen después de una o varias **entrevistas con el cliente** que será el que tendrá que decirnos qué quiere que haga el sistema. Otras veces no habrá cliente, y tendremos que sacar esa información de la observación de un modelo real. Es frecuente que también debamos entrevistarnos con diferentes usuarios del sistema (o actores del mismo), para llegar a una especificación de requisitos completa.

La idea es sacar la máxima **información** posible sobre las funcionalidades esperadas, sobre el punto de vista de cada actor que interactúa con el sistema, sobre los conceptos que se manejan en los distintos escenarios de nuestro problema.

Desarrollo orientado a objetos usando uml y herramientas case

Descripción de los casos de uso

Una vez hemos definido los **requerimientos del sistema**, debemos comprender esos requerimientos y para ello nada mejor que la creación de **casos de uso del sistema**, que como ya sabemos, son documentos narrativos que describen a los actores utilizando nuestro sistema para completar un proceso o satisfacer un objetivo. Además como vimos anteriormente, estos casos de uso serán la columna vertebral de nuestro proceso de desarrollo orientado a objetos.

Como también hemos visto en las unidades anteriores, UML nos proporciona un tipo de **diagrama** para representar la relación entre los diferentes actores de nuestro sistema y los diferentes casos de uso, pero no define un formato para definir o describir un caso de uso en sí.

En las siguientes secciones veremos las diferentes **definiciones** que podemos hacer de un caso de uso, así como el formato que utilizaremos para realizar su descripción. También veremos los diferentes pasos que debemos seguir en esta primera fase de planificación.

Desarrollo orientado a objetos usando uml y herramientas case

Clasificación de los casos de uso

¿Sabrías en qué basarte para clasificar los casos de uso? ¿Sabrías decir qué casos de uso son los más importantes?

A la hora de describir los **casos de uso**, nos será de vital importancia hacer una buena clasificación de los mismos, ya que esta clasificación marcará de alguna forma las diferentes fases de desarrollo (ya que empezaremos con los casos de usos más críticos, más importantes, ...).

- **Casos de uso con respecto a su importancia:**
  - **Casos de uso primarios**: Representan los procesos más importantes de nuestro sistema, sin estos procesos nuestro sistema no sería lo que va a ser.
  - **Casos de uso secundarios**: Representan procesos menores o raros, procesos que se dan con poca frecuencia en nuestro sistema o no son fundamentales para el funcionamiento del mismo.
  - **Casos de uso opcionales**: Representan procesos que podrían no abordarse a la hora de construir nuestro sistema. Son procesos casi sin importancia.

- **Casos de uso con respecto a su nivel de abstracción:**
  - **Casos de uso esenciales**: Un caso de uso es esencial, cuando en su descripción nos dedicamos a describir "**qué**" hará nuestro proceso y no "cómo" lo hará. Los casos de uso esenciales son, por tanto, utilizados en la fase de planificación, ya que pretenden indicarnos qué deberá hacer nuestro caso de uso y en ningún momento dará indicaciones del cómo.

  - **Casos de uso reales**: Un caso de uso real nos da muchos más detalles acerca del cómo llevar a cabo nuestro caso de uso. A veces la distinción entre unos y otros es despreciable, ya que algunas veces tendemos a describir el "cómo", a la vez que el "qué", por lo que algunas veces nos podremos encontrar la descripción de estos casos en la fase de planificación (sobre todo si se trata de un sistema simple, o de un caso de uso relativamente simple). Lo general, es describir este tipo de casos de uso (que generalmente no son más que una especialización de los casos de uso anteriores) en la fase de diseño de una de las fases de desarrollo.

Desarrollo orientado a objetos usando uml y herramientas case

Descripción de alto nivel de un caso de uso

La **descripción de alto nivel** de un caso de uso, describe clara y concisamente el proceso que representa el caso de uso dentro del sistema, pero de una forma muy breve. Es conveniente comenzar con la descripción de alto nivel de los casos de uso para llegar rápidamente a comprender los principales procesos globales, a fin de entender rápidamente el grado de complejidad del sistema.

Ya que UML no especifica el formato que debemos seguir a la hora de describir un caso de uso, nosotros utilizaremos el siguiente formato para llevar a cabo la descripción de alto nivel de nuestros casos de uso:

Caso de uso

Actores

Tipo

Descripción

- En el primer apartado pondremos el **nombre** de nuestro caso de uso. Este nombre debería comenzar con un verbo para subrayar que se trata de un proceso.
- En el apartado **actores**, listaremos los actores que interactúan en este caso de uso.
- En el apartado **tipo** clasificaremos nuestro caso de uso en primario, secundario u opcional.
- En la parte de **descripción**, realizaremos una breve descripción muy general de nuestro caso de uso.

Desarrollo orientado a objetos usando uml y herramientas case

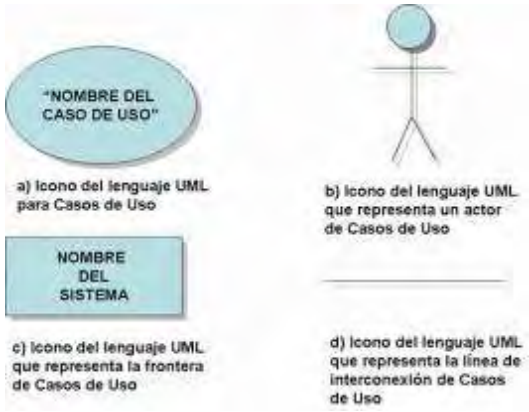
Descripción expandida de un caso de uso

Una **descripción expandida** de un caso de uso es una descripción más detallada del mismo, con la que pretendemos especificar más a fondo **qué es lo que hace** el caso de uso en cuestión. Para realizar esta descripción más detalladamente, lo que hacemos es añadir a la ficha de descripción de alto nivel, otros apartados para poder detallar las peculiaridades de nuestro caso de uso.

Como en el caso anterior, UML tampoco define el formato que debemos utilizar para llevar a cabo esta descripción detallada, por lo que nosotros proponemos el siguiente formato:

Caso de uso
Actores
Propósito
Resumen
Tipo
Referencias cruzadas
Curso normal
Curso alternativo

Como antes, el **nombre** identificará el caso de uso y deberá comenzar con un verbo que denote la acción realizada por el proceso.



- Los **actores**, será una lista de actores que interactúan con el sistema en el caso de uso que se está describiendo. Al ser la descripción expandida, debemos indicar qué actor es el que inicia el caso de uso.
- El **propósito** del caso de uso es la intención que persigue el caso de uso.
- En el apartado **resumen**, repetiremos la descripción hecha del mismo en su formato de alto nivel.
- En el apartado **tipo** clasificaremos nuestro caso de uso en primario, secundario u opcional y en esencial o real.
- Cuando hablamos de **referencias cruzadas**, nos estamos refiriendo a los requerimientos a los que hace referencia este caso de uso. Será una lista de requerimientos.
- El apartado **curso normal**, es la parte principal del formato expandido, y es la principal diferencia de este formato con el de alto nivel. En este apartado debemos describir los detalles de la interacción entre los diferentes actores que intervienen en el caso de uso y el sistema. Comenzaremos con un enunciado con este formato: "Este caso de uso comienza cuando <actor><inicio de evento>", para así subrayar el actor y el evento iniciadores del caso de uso. Describiremos la secuencia de eventos más comunes, mediante la enumeración de acciones de cada uno de los actores. Estas acciones las numeraremos para indicar el orden que siguen unas respecto a otras.

- El apartado **curso alternativo** indica las distintas excepciones que pueden surgir en el curso normal de los eventos, y las acciones a realizar ante estas excepciones.

Autoevaluación

1 Respecto a los casos de uso podemos decir:

a) Existen casos de uso esenciales y casos de uso reales.
b) Los casos de uso esenciales se pueden describir de dos formas: descripción de alto nivel y descripción expandida.
c) En la fase de planificación usamos los casos de uso esenciales.
d) Todas las respuestas anteriores son correctas.

Comprobar

Desarrollo orientado a objetos usando uml y herramientas case

Pasos a seguir en la fase de Planificación

A la hora de abordar esta primera fase de planificación, debemos seguir los siguientes pasos:

1. **Listar las funciones del sistema** o requisitos del mismo.
2. **Definir los límites del sistema**. Los límites del sistema identifican lo que es interno o externo al mismo. Generalmente la frontera suele ser una frontera software/hardware.
3. **Identificar los actores del sistema**. Los actores son entidades externas (de ahí que los identifiquemos después de definir los límites del sistema) que estimulan el sistema con eventos o reciben algo del sistema.
4. **Identificar los casos de uso y describir éstos en su formato de alto nivel**. Para describir los casos de uso en el formato de alto nivel una de las tareas que debemos realizar es clasificar los casos de uso, ya que esa información la debemos incluir en las fichas propuestas para describirlos en el formato de alto nivel.
5. **Realizar el diagrama de casos de uso**. Como ya sabemos, este diagrama representa de forma visual a los actores del sistema, los casos de uso, los límites del sistema y las relaciones entre los casos de uso y los actores o entre casos de uso.
6. **Describir los casos de uso más críticos, importantes o que conllevan un mayor riesgo en su formato expandido**. Si hablamos de un problema simple, podemos describir todos los casos de uso en este formato, pero para los demás problemas que conlleven algo más de complejidad, sólo describiremos los mencionados, dejando la descripción expandida del resto para posteriores ciclos de desarrollo.
7. **Ordenar los casos de uso según la prioridad**, para determinar qué casos de uso tratar en qué ciclo de desarrollo. Empezaremos tratando aquellos casos de uso que nos aporten una mayor compresión global del sistema, aquellos que posean funciones críticas del sistema, los más importantes, ...

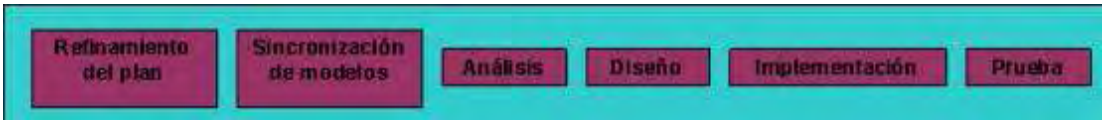
Desarrollo orientado a objetos usando uml y herramientas case

CASO:

Víctor ya tiene claro que la fase de planificación es bastante importante, pues de ella dependen las demás fases. Además ahora le ha quedado claro que todo el proceso de desarrollo está dirigido por los casos de uso, de ahí la importancia de esta fase. Además Víctor ha podido aplicar su conocimiento sobre el lenguaje UML para aplicarlo al proceso de desarrollo y ha notado que en esa primera fase ya se hace uso de uno de esos diagramas.

Pero Víctor sigue impaciente por seguir aprendiendo, y quiere llegar hasta el final. Conforme más se adentra en el proceso de desarrollo, más le está entusiasmando y cada vez le ve más utilidad. Ahora Víctor va a empezar a ver los ciclos de desarrollo y quiere enterarse bien de qué es eso de proceso iterativo. Vamos ayudarle a conseguirlo.

¿Estás tan impaciente como Víctor? ¿Quieres llegar a comprender qué es eso de **proceso iterativo** al igual que Víctor? Como vimos en la introducción de esta unidad la fase de desarrollo propiamente dicha está compuesta por uno o varios ciclos. En cada ciclo de desarrollo abarcaremos un conjunto de casos de uso según su prioridad. Cada ciclo de desarrollo consta de las siguientes fases:



Como podemos ver aquí tenemos actividades como **refinamiento del plan** y **sincronización de modelos**, que sólo se llevarán a cabo cuando estemos hablando de una fase de desarrollo distinta a la primera.

La **primera fase** de desarrollo será la que abarcamos justo después de la fase de planificación. En esta primera fase, no tendremos todavía un plan que refinar y tampoco tendremos modelos que sincronizar. Sin embargo en las siguientes fases de desarrollo, conforme vayamos abarcando más casos de uso o abarcando más complejidad en los mismos, sí tendremos la necesidad de refinar el plan y de sincronizar modelos, ya que pueden aparecer conceptos, asociaciones, ... que no habían aparecido en fases anteriores de

desarrollo.

En esta unidad no trataremos la fase de **implementación** debido a que por su complejidad es tratada en un módulo aparte. Tampoco hablaremos de la fase de pruebas ya que la misma ha sido tratada en unidades anteriores. Nosotros nos vamos a centrar en las dos actividades principales de esta fase iterativa: **análisis y diseño**. En los siguientes apartados describiremos los puntos principales a tratar en cada una de estas fases, los documentos que deberemos generar y el conocimiento que debemos adquirir en cada una de ellas.

Autoevaluación

1 Respecto al ciclo de desarrollo podemos afirmar que...

a) Al igual que la fase de planificación, éste se realiza una sola vez.
b) Nuestro proceso se compone de fase de planificación, fase de análisis y fase de desarrollo.
c) La fase de desarrollo es la fase iterativa de nuestro proceso.
d) En nuestro proceso no existe fase de desarrollo pero sí de diseño.

Comprobar

Te preguntarás qué pasos debes seguir en la fase de **análisis**.

En la fase de análisis de un ciclo de desarrollo se investiga sobre el **problema**, sobre los conceptos relacionados con el subconjunto de casos de uso que se está tratando. Se intenta llegar a una buena comprensión del problema, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación.

Las principales actividades a tratar en esta fase son las que exponemos a continuación y que ampliamos en los apartados siguientes.

- **Definir casos de uso esenciales en su formato expandido**. Ya sabemos que dependiendo de la complejidad del problema y de cada caso de uso, la descripción expandida de los casos de uso esenciales la hemos podido hacer en la fase inicial de planificación.
- **Refinar los diagramas de casos de uso**. Esto no se realizará en la primera fase de desarrollo, ni tampoco en una fase de desarrollo en la que no cambien los casos de usos (sólo cambie su descripción).
- **Definir / Refinar el modelo conceptual**. Volvemos a lo mismo, si es la primera fase de desarrollo, deberemos definir el modelo conceptual, pero si es una fase más avanzada deberemos refinarlo, contemplando los nuevos conceptos, asociaciones o atributos que pudieron surgir en fases anteriores.
- **Definir los diagramas de secuencia del sistema**. Como ya veremos construiremos un diagrama de secuencia para cada caso de uso.
- **Definir los contratos de operación**. Cada caso de uso puede constar de más de una operación, y a la vez una operación puede ser usada por más de un caso de uso. De lo que se trata es de identificar estas operaciones del sistema y definir el contrato de cada operación (que ya veremos lo que es).

En los siguientes apartados vamos a describir las actividades más importantes de este proceso (ya que otras ya las conocemos).

Desarrollo orientado a objetos usando uml y herramientas case

Modelo Conceptual

¿Qué es para ti un **modelo conceptual**?

Un modelo conceptual explica los **conceptos significativos** en el dominio de nuestro problema. Como hemos visto anteriormente, la fase de análisis orientada a objetos trata de investigar sobre el dominio del problema, y una parte importante de esta tarea es identificar los conceptos que componen nuestro problema. Como estamos en la fase de análisis, estos conceptos representarán hechos del mundo real y no componentes software.

En las unidades anteriores se estudió UML, y no se vio ningún diagrama llamado modelo conceptual. Efectivamente, ya que éste es tratado como un **diagrama de estructura estático** (ya veremos más adelante como se puede modelar usando UML).

Veamos las **tareas** que debemos seguir para llegar a construir nuestro modelo conceptual:

- **Identificación de conceptos**. A la hora de identificar los conceptos del dominio de nuestro problema debemos centrarnos en la especificación de requisitos y en el conocimiento general acerca del dominio del problema. Hay categorías típicas que deberían ser incluidas como conceptos (tipo de concepto, objetos físicos, especificaciones, lugares, transacciones, contenedores de otras cosas, cosas de un contenedor, catálogos, etc.). También debemos buscar sustantivos en la definición de requisitos y en las descripciones de los casos de uso. A la hora de nombrar los conceptos deberemos emplear la estrategia del cartógrafo que se puede resumir en:
  - Usar los nombre existentes en el territorio.
  - Excluir las características irrelevantes.
  - No añadir cosas que no están ahí.

- **Identificación de asociaciones**. Una vez que hemos identificado los conceptos es la hora de identificar las **asociaciones** existentes entre estos. Una asociación es una relación entre conceptos que indica una conexión significativa e interesante entre ellos. Las asociaciones que vale la pena mencionar son aquellas que perduran en el tiempo, ya que deben ser conocidas (debido a los requerimientos) o aquellas relaciones típicas (A es una parte física de B, A es una parte lógica de B, A está físicamente contenido en B, A es una descripción de B, A es un elemento de línea en una transacción B, A es miembro de B, A usa o dirige a B, A se comunica con B...).

Una asociación se identifica por un nombre (debe empezar con mayúsculas y si es una frase nominal se construirá usando guiones) y una multiplicidad que define cuántas instancias de un tipo A pueden asociarse a una instancia del tipo B en determinado momento.

- **Identificación de atributos**. Un **atributo** es un valor lógico de un dato de un concepto. Debemos incluir aquellos atributos en que los requerimientos o los casos de uso indican o conllevan la necesidad de recordar información. Los atributos siempre deben ser valores simples, en el momento en el que un atributo tome valores complejos, nos estará indicando que eso no es un atributo, sino que debería ser un concepto. Un **error** común a los desarrolladores de bases de datos es incluir atributos que son llaves ajenas, ya que en esta fase no estamos todavía pensando en cómo vamos a implementar nuestro problema (sería como comparar el modelo Entidad / Relación y el modelo relacional cuando hablamos de diseño de bases de datos).
- **Construcción del modelo conceptual**. Una vez identificados todos los elementos de los que constará nuestro modelo, sólo nos queda construirlo. Dibujaremos todos los conceptos identificados, añadiremos las asociaciones entre éstos (añadiendo su nombre y multiplicidad) y finalmente añadiremos los atributos de cada concepto.

Autoevaluación

1 Un modelo conceptual...

a) Describe los conceptos del mundo real para nuestro problema.
b) Se utiliza en la fase de análisis del ciclo de desarrollo actual.
c) No describe los conceptos software de nuestro problema.
d) Todas son correctas.

Comprobar

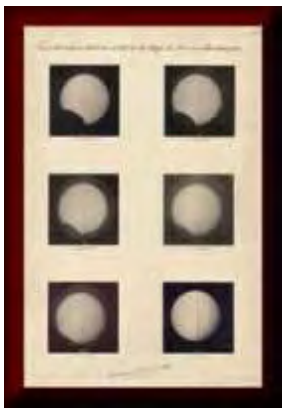
Desarrollo orientado a objetos usando uml y herramientas case

Diagramas de secuencia del sistema

Ya debes saber cómo se construyen los **diagramas de secuencia** utilizando UML, pero ¿en qué te debes basar para construirlos? Como ya sabes los diagramas de secuencia del sistema muestran gráficamente los eventos que fluyen de los actores al sistema. En las unidades anteriores hemos visto los elementos que los componen, y la forma de construirlos. En este apartado daremos algunas directrices para integrar los mismos dentro de nuestro proceso de desarrollo orientado a objetos.

Una vez que tenemos definidos los casos de uso, podemos ver que éstos muestran una **interacción entre los actores y el sistema**. Cada caso de uso nos muestra un escenario en el que actúan los actores y el sistema, comunicándose por medio de eventos que el actor solicita al sistema o que el sistema genera. Como veremos después, cada evento da origen a una operación en respuesta a ese evento.

Como vimos en las unidades anteriores, UML dispone de un diagrama que nos permite representar esta interacción de una manera adecuada. Estos diagramas no son otros que los diagramas de secuencia, los cuáles nos permiten representar un escenario particular entre los actores y el sistema, los eventos que intercambian ambos y su orden.



De lo que se trata es de realizar un diagrama de secuencia por cada caso de uso identificado, para lo cuál seguiremos las siguientes directrices:

- Representaremos el sistema como un **objeto** con una línea debajo.
- Identificaremos los **actores** que intervienen en el caso de uso y también dibujaremos una línea debajo para cada uno de ellos.
- Analizaremos el **curso normal** de eventos del caso de uso y los representaremos en nuestro diagrama. Es conveniente que los eventos se nombren por medio de verbos, para subrayar que estos eventos se corresponden con una operación.
- Es una buena idea incluir la descripción del caso de uso como un comentario en el diagrama de secuencia.

Desarrollo orientado a objetos usando uml y herramientas case

Contratos de Operaciones

Una vez que tenemos identificadas las operaciones del sistema en los diagramas de secuencia, debemos describir mediante **contratos** el comportamiento esperado del sistema en cada una de las operaciones identificadas.

Un contrato contribuye a definir el comportamiento del sistema y describe el efecto que sobre él tienen las operaciones.

Un contrato es un documento que **describe qué es lo que se espera de una operación**. Tiene una redacción en estilo declarativo, y debido a que estamos en la fase de análisis, enfatiza más en el "qué" que en el "cómo". La parte fundamental de un contrato (o la que más nos interesa) es la parte dedicada a las precondiciones y poscondiciones, que indican cómo cambia el estado del sistema cuando se invoca una de sus operaciones.

Para describir un contrato podemos usar la siguiente plantilla:

Nombre
Responsabilidades
Referencias cruzadas
Notas
Excepciones
Salida
Precondiciones
Poscondiciones

- El **nombre** identificará la operación que estamos tratando, y si tiene o no parámetros.
- El apartado **responsabilidades** será una descripción informal de las responsabilidades que la operación debe desempeñar.
- Las **referencias cruzadas** indican los requerimientos que satisface y los casos de uso en los que es utilizada esta operación.
- El apartado **notas** nos permitirá hacer cualquier tipo de comentario que deseemos incluir.
- Las **excepciones** son las situaciones anómalas que pueden surgir y que debemos tener en cuenta, además de indicar cómo actuar ante ellas.

- La **salida** indica los mensajes que se envían fuera del sistema (en la mayor parte de las operaciones del sistema este apartado queda vacío).
- Las **precondiciones** son las suposiciones acerca del estado del sistema antes de ejecutar la operación.
- Las **poscondiciones** indican el estado del sistema después de completar la operación. A la hora de redactar las poscondiciones es mejor utilizar un tiempo pasado, para subrayar que se trata de declaraciones sobre un cambio en el estado que ya ha pasado (se ha creado, en vez de crear ...). Es importante escribir las asociaciones que se crean entre objetos. Es como si hiciésemos una fotografía al sistema antes de aplicar la operación y otra después y describir los cambios de estado producidos.

Por tanto, los pasos que deberemos seguir para construir un **contrato** son los siguientes:

1. Identificaremos las operaciones del sistema a partir de los diagramas de secuencia.
2. Para cada operación identificada deberemos construir un contrato.
3. Empezaremos escribiendo el apartado de responsabilidades, describiendo informalmente el propósito de la operación.
4. A continuación rellenaremos el apartado poscondiciones, describiendo declarativamente los cambios de estado que sufren los objetos en el modelo conceptual.
5. Finalmente completaremos el resto de apartados del contrato.

Autoevaluación

1 La fase de análisis del ciclo de desarrollo actual ...

a) Incluye la construcción del modelo conceptual y del diagrama de casos de uso.
b) Incluye la construcción del modelo conceptual, de diagramas de secuencia y la descripción de los contratos de operación.
c) Incluye la construcción de diagramas de secuencia y la construcción del diagrama de casos de uso.
d) Todas son correctas.

Comprobar

¿Qué diferencias crees que hay entre la fase de análisis y la fase de diseño?

Cómo hemos visto, en la **fase de análisis** se da prioridad al conocimiento de los requerimientos, de los conceptos y de las operaciones relacionadas con el sistema.

Sin embargo, en la **fase de diseño** se da prioridad a la creación de una solución a nivel lógico para satisfacer los requisitos, basándose en el conocimiento adquirido en la fase de análisis.

La fase de diseño se centra en la elaboración de los diagramas de interacción del sistema, para posteriormente construir el diagrama de clases de diseño.

Las actividades a realizar durante la fase de diseño son las siguientes:

- **Definir los casos de uso reales** (si es que no los habíamos descrito ya).
- **Definir informes e Interfaz de usuario**. Aquí se definirá cómo interactuarán los actores con el sistema y que formato tendrá la información de salida. (En esta unidad no trataremos este apartado ya que éste ha sido tratado en otra unidad anterior).
- **Definir los diagramas de colaboración**. Realizaremos un diagrama de colaboración para cada contrato.
- **Definir los diagramas de clases de diseño**. Este es el objetivo principal de esta fase de diseño.
- **Definir el esquema de Base de Datos**. Se describirá la base de datos sobre la que se apoyará nuestro sistema (este apartado tampoco lo trataremos por el mismo motivo; ya ha sido tratado en una unidad anterior)

Por lo que nos vamos a centrar en las dos principales actividades de esta fase de diseño: la construcción de los diagramas de colaboración y de los diagramas de clases de diseño (ya que de los casos de uso reales ya hemos hablado, y hemos dicho que no íbamos a tratar el diseño de interfaces ni el esquema de base de datos que ya han sido tratados en unidades anteriores).

Desarrollo orientado a objetos usando uml y herramientas case

Diagramas de colaboración

¿Recuerdas los diagramas de **colaboración**? ¿Sabías que es una fase muy importante para llegar a un diseño robusto?

Efectivamente, **una de las actividades más importantes en la fase de diseño es la construcción de los diagramas de interacción, que son los encargados de mostrar el intercambio de mensajes entre instancias del modelo de clases para cumplir las poscondiciones establecidas en cada contrato.**

Existen dos **tipos** de diagramas de interacción:

- los de secuencia y
- los de colaboración.

Con ambos podemos expresar lo mismo, pero para esta fase creemos más clarificador la utilización de los segundos, los de colaboración.

Veamos los **pasos** que debemos seguir a la hora de construir un diagrama de colaboración:

- Crearemos un diagrama **separado** para cada operación del sistema en desarrollo en el ciclo de desarrollo actual. Para cada evento del sistema, crearemos un diagrama con él como mensaje inicial.
- Usando los apartados de **responsabilidades y de**

poscondiciones del contrato de operación que estamos tratando y la descripción del caso de uso como punto de partida, diseñaremos un sistema de objetos que intercambian mensajes para llevar a cabo las tareas requeridas por el contrato.

- Si el diagrama se complica, **lo dividiremos** en dos diagramas más pequeños. Para ello se termina la secuencia de mensajes en un mensaje determinado, y en el segundo diagrama se comienza con el mensaje que terminó el primero. Debe indicarse en el primer diagrama que el resto de la interacción se detalla en el segundo.

Es normal realizar un borrador de diagrama de clases de diseño, o realizar éste en paralelo que hacemos el diagrama de colaboración. La asignación de responsabilidades a los distintos objetos es una habilidad clave, y que se va adquiriendo según aumenta la experiencia en el desarrollo orientado a objetos. Craig Larman también propone el uso de los patrones GRASP (General Responsibility Asignment Software Patners - patrones generales de software para asignar responsabilidades) para la asignación de responsabilidades.

Llamamos **responsabilidad** a una obligación de una clase. Las responsabilidades están ligadas a las obligaciones de un objeto en cuanto a su comportamiento. Básicamente, estas responsabilidades pueden ser de tipo **conocer** o de tipo **hacer**:

- **Conocer**:
  - Conocer datos privados encapsulados.
  - Conocer los objetos relacionados.
  - Conocer las cosas que pueden calcular o derivar.
- **Hacer**:
  - Hacer algo él mismo.
  - Iniciar una acción en otros objetos.
  - Controlar y coordinar actividades en otros objetos.

La idea es que digamos **de qué objeto** es cada una de las distintas responsabilidades encontradas en el sistema, y lo representemos en los diagramas. Las responsabilidades de tipo conocer se pueden inferir normalmente del modelo conceptual.

Una responsabilidad no es lo mismo que un método, pero los métodos se implementan para satisfacer responsabilidades.

Autoevaluación

1 Sobre los diagramas de colaboración podemos decir que...

a) Debemos dedicarles todo el tiempo necesario para definir las responsabilidades, ya que es una tarea muy importante en la fase de diseño.

b) No tenemos porque realizarlos, ya que como en la fase de análisis realizamos los diagramas de secuencia y ambos son diagramas de interacción, podríamos pasar de uno a otro sin problemas.

c) Se realizan en la fase de análisis justo después de construir los diagramas de secuencia.

d) Todas son correctas.

Comprobar

Desarrollo orientado a objetos usando uml y herramientas case

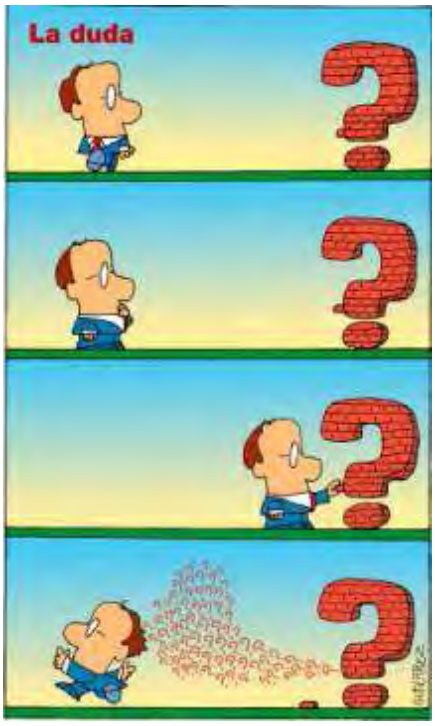
Diagramas de clases de diseño

¿Recuerdas **cómo se construían** los diagramas de clases de diseño?

Como ya sabemos de las unidades anteriores los diagramas de clases de diseño muestran la especificación para las clases software de una aplicación, a diferencia del modelo conceptual que define los conceptos del mundo real.

Como también sabemos, los diagramas de clases de diseño nos muestran información acerca de:

1. **Clases**: El diagrama muestra las clases software, sus atributos, sus métodos. También se muestra la **visibilidad** (privada, protegida o pública) de estos atributos y métodos. Generalmente todos los atributos de una clase son privados, y la clase dispone de métodos para leer y modificar el atributo, aunque por claridad esto no se suele mostrar en los diagramas. También es posible representar en los diagramas los tipos de los atributos, de los valores devueltos por los métodos y de los parámetros de éstos.
2. **Relaciones de dependencia para representar visibilidad entre clases**: Los diagramas de clases también nos muestran la **visibilidad** que los objetos deben tener entre ellos para que estos puedan desempeñar sus funciones. Para que un **objeto A** pueda invocar un método de otro **objeto B**, el objeto **B** debe ser visible al objeto **A**. Hay varias clases de visibilidad: de atributo (la más normal), la de parámetro, la local y la global.
3. **Navegabilidad**: Como vimos en las unidades anteriores, en los diagramas de clases de diseño también se representa la **navegabilidad**, que es una propiedad de un rol (extremo de una asociación) que indica que es posible "navegar" unidireccionalmente a través de la asociación, desde objetos de la clase origen a objetos de la clase destino. Esto implica visibilidad, normalmente de atributo en la clase origen. En la implementación se traducirá en un atributo, que sea una referencia a la clase destino, en la clase origen. Las situaciones en las que debemos definir una asociación de navegabilidad de A a B son:
   - **A** envía un mensaje a **B.**
   - **A** crea una instancia de **B.**
   - **A** necesita mantener una conexión con **B.**



Ahora que ya sabemos la información que debemos mostrar en los diagramas de colaboración, veamos los **pasos a seguir en su construcción**:

1. **Identificaremos todas las clases** participantes en la solución software. Esto se lleva a cabo analizando los diagramas de colaboración.
2. Las representaremos en un **diagrama de clases**.

3. **Duplicaremos los atributos que aparezcan** en los conceptos asociados del modelo conceptual.
4. **Añadiremos los métodos que aparezcan** en los diagramas de colaboración.
5. **Añadiremos la información de tipo a los atributos y métodos**.
6. **Añadiremos las asociaciones necesarias** para soportar la visibilidad de atributos requerida.
7. **Añadiremos flechas de navegabilidad a las asociaciones** para indicar la dirección de visibilidad de los atributos.
8. **Añadiremos relaciones de dependencia** para indicar visibilidad no correspondiente a atributos.

Una vez concluidas estas actividades (y sin olvidar que debemos pasar por las fases de implementación y pruebas, aunque aquí no las estemos tratando) habremos concluido el ciclo de desarrollo actual. Lo siguiente a realizar, es escoger nuevos casos de uso que aún no hayan sido tratados y comenzar otro ciclo de desarrollo partiendo de esos requerimientos impuestos por los casos de uso tratados en el mismo.

Autoevaluación

1 El diagrama de clases de diseño...

a) Es igual que el modelo conceptual, pero también incluye las operaciones.
b) Nos muestra la especificación que tendrán nuestras clases software.
c) Realizamos un diagrama por clase software de nuestro sistema.
d) Todas son verdaderas.

Comprobar

Desarrollo orientado a objetos usando uml y herramientas case

Caso:

__Víctor__ se ha empapado de todo el proceso de desarrollo orientado a objetos, con la ayuda de __Carmen__. Ésta le ha preguntado ¿qué tal? y __Víctor__ le responde que cree que lo tiene bastante claro y que está dispuesto a formar parte de un equipo que trabaje con este proceso.

Pero __Carmen__, que es más veterana en estos temas sabe que uno no tiene las cosas del todo claras hasta que no se enfrenta a un problema real. __Carmen__ se ha enterado que a __SI Andalucía__ le ha surgido un proyecto bastante sencillo sobre la gestión de un videoclub y cree que es una buena forma de poner a prueba los conocimientos que __Víctor__ ha adquirido, por lo que le propone que sean ellos los encargados de llevar a cabo este proyecto, siguiendo las directrices del proceso de desarrollo que ha enseñado a __Víctor__ y el mismo que sigue __SI Andalucía__ en todos sus proyectos.

Para **ejemplificar** todo este proceso de desarrollo que hemos expuesto en los apartados anteriores, lo mejor es seguirlo aplicado a un **caso real** de estudio. En los siguientes apartados nos dedicaremos a seguir el proceso de desarrollo orientado a objetos para un caso real: **informatizar el terminal de un videoclub.**

Hemos escogido un problema simple, ya que de lo que se trata es de seguir las diferentes fases del proceso sin perderse en otras complejidades añadidas del problema.

Caso:

Lo primero que __Víctor__ y __Carmen__ han hecho es reunirse

informalmente con el cliente, ya que aunque todos sabemos cómo funciona "más o menos" un videoclub, no todos funcionan igual. Además ese "más o menos" no es suficiente para comenzar nuestro proceso de desarrollo. En la reunión se preguntaba todo tipo de dudas acerca del funcionamiento del videoclub a nuestro cliente y como conclusión ***Víctor*** *y* ***Carmen*** *llegaron a tener escrita la siguiente información referida a su funcionamiento y/o necesidades:*

"El videoclub dispone de copias de películas, que alquila a sus socios.

De cada película suele haber más de una copia.

Los datos que se quieren almacenar de cada película son: nombre, director, duración, género, productora, precio.

Las películas también tienen un código. Cada copia tiene un código de copia.

Se deben poder dar de alta películas y copias de películas.

También debe ser posible modificar datos de la película como por ejemplo el precio.

El videoclub dispone de ***socios****, de los cuáles conocemos los siguientes datos: código, nombre, dni, dirección, teléfono. Se podrán dar de alta nuevos socios que presenten una copia del DNI. También podremos modificar datos de un cliente como por ejemplo reflejar un cambio de teléfono o de domicilio.*

Los socios alquilan copias de películas, y se apunta la fecha de alquiler, ya que las películas se pagan al devolverse. Cuando un socio devuelve una copia, se le dice lo que debe, éste paga y se apunta que ese socio alquiló dicha película en la fecha en la que la alquiló y la fecha de devolución para posibles consultas. "

Por tanto, partiremos de esta información y trataremos de desarrollar nuestro sistema orientado a objetos que solucione el problema que estamos dispuestos a abordar. Los siguientes apartados tratarán por tanto de la fase de planificación y del ciclo de desarrollo.

Desarrollo orientado a objetos usando uml y herramientas case

Caso:

Víctor *está emocionado después de la entrevista con el cliente. Está deseoso de empezar a trabajar y* ***Carmen*** *le dice que se tranquilice y que deben seguir todas las fases que conforman el proceso de desarrollo visto anteriormente.*

Víctor *ya lo sabe, y no es que quiera ir deprisa, sino que tiene ganas de aplicar dicho proceso. También sabe que lo primero que tiene que hacer son las actividades propias de la fase de planificación, por lo que le dice a* ***Carmen*** *que por qué no se toman un café y empiezan a definir los requisitos del problema que están abordando, para poder continuar con las demás actividades de esta primera fase.*

Si es posible, el café con un par de donuts de chocolate - le dice ***Carmen****.*

En este apartado llevaremos a cabo la fase de **planificación** para nuestro sistema. Por tanto nos dedicaremos a hacer una especificación de los **requisitos**, describir los casos de uso y realizar el diagrama de casos de uso. Empezaremos por la especificación de los requisitos.

Desarrollo orientado a objetos usando uml y herramientas case

Especificación de los requisitos

Después de analizar las notas tomadas, podemos observar que nuestro sistema tiene **tres objetivos** fundamentales:

- La gestión de socios,
- La gestión de películas y copias y
- La gestión de alquileres y devoluciones.

De estos tres objetivos principales y de la información de que disponemos podemos decir que los requisitos de nuestro sistema son los siguientes:

- **R1**: El sistema debe almacenar información sobre las películas (código, nombre, director, duración, productora, género y precio).
- **R2**: El sistema nos permitirá modificar datos de una película.
- **R3**: El sistema debe permitirnos añadir copias de una película.
- **R4**: El sistema debe almacenar la información sobre nuevos socios (código, nombre, DNI, teléfono y dirección).
- **R5**: El sistema nos ofrecerá la posibilidad de modificar datos de los socios.
- **R6**: El sistema nos permitirá anotar un alquiler de una copia por un socio en una fecha dada.
- **R7**: El sistema permitirá anotar la devolución de una copia por un socio en una fecha dada (guardando también la fecha en la que fue alquilada), informándole del precio y cobrando el mismo.

Desarrollo orientado a objetos usando uml y herramientas case

Límites del sistema y actores

¿Te acuerdas cuáles eran los **pasos** que debíamos dar una vez que teníamos definidos los requisitos de nuestro problema?

Ya tenemos claros cuáles son esos requisitos de nuestro sistema, por lo que ahora debemos definir los **límites** de nuestro sistema. En nuestro caso, los límites de nuestro sistema serán los límites de hardware / software de nuestro Terminal de gestión de video club, que llevará a cabo las funcionalidades de nuestro sistema orientado a objetos. Por tanto será el empleado del video club el que interactuará con el mismo.

Si el empleado del video club interactúa con nuestro sistema quiere decir que claramente el **empleado** va a ser un actor de nuestro sistema.

Un **socio** de nuestro video club también lo consideraremos actor de nuestro sistema. Aunque el socio no interactuará directamente con nuestro sistema software, sí lo hace con nuestro modelo del mundo real.

Resumiendo:

- **Límites del sistema**: Los límites de hardware que representa el Terminal de gestión de videoclub.
- **Actores del sistema**: Empleado y Socio.

Desarrollo orientado a objetos usando uml y herramientas case

Identificación, clasificación de los casos de uso y descripción en su formato de alto nivel

A la hora de identificar los casos de uso de nuestro sistema nos fijaremos en los actores del mismo y sus interacciones con nuestro sistema. Si además nos fijamos en los **requisitos** de nuestro sistema, parece que los casos de uso están bastante claros y podrían resumirse en:

- Dar de alta película.
- Modificar película.
- Añadir copia.

- Dar de alta socio.
- Modificar socio.
- Alquilar copia.
- Devolver copia.

Una vez que tenemos identificados los casos de uso, vamos a clasificarlos en **primarios**, **secundarios** u opcionales. Las actividades más frecuentes del videoclub son el alquiler y la devolución de películas, realizándose más raramente las otras actividades. Además de ser las actividades más frecuentes, son las más importantes, por lo que estos casos de uso los clasificaremos como primarios y los demás como secundarios. Pasemos a describir los mismos en su formato de alto nivel. Puedes ver cómo lo hemos hecho en el enlace al siguiente recurso:

Recurso

Autoevaluación

1 Para nuestro caso de estudio...

a) Los requisitos nos los escribió el cliente.
b) Los límites del sistema los define el cliente y de ellos depende todo el proceso de desarrollo.
c) Los casos de uso los hemos definido a partir de los límites del sistema.
d) Todas son falsas.

Comprobar

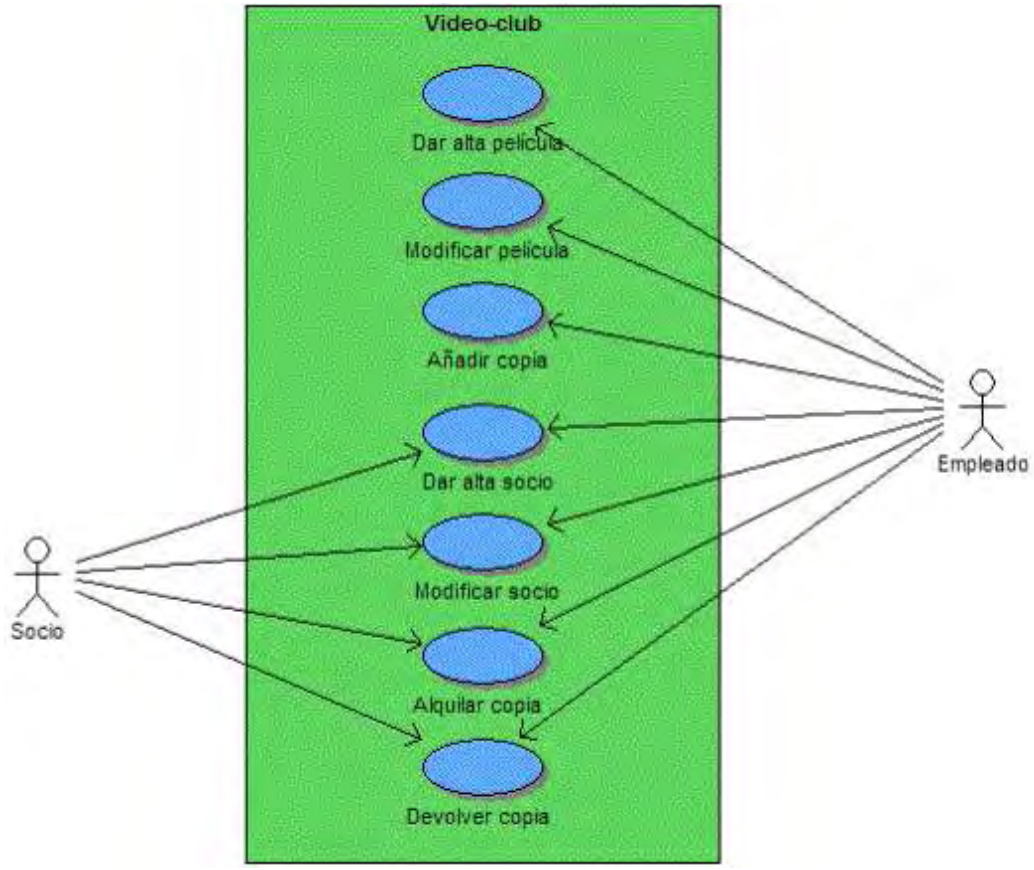
Desarrollo orientado a objetos usando uml y herramientas case

Construcción del diagrama de casos de uso

¿Sabrías construir el **diagrama de casos** de uso a partir de la información obtenida en los apartados anteriores?

Una vez identificados los actores del sistema, los límites del mismo y los casos de uso, es la hora de construir nuestro diagrama de casos de uso, que será el encargado de mostrar gráficamente como interactúan los actores con nuestro sistema por medio de los casos de uso.

El **diagrama** que surge para nuestro caso de estudio es el siguiente:



Desarrollo orientado a objetos usando uml y herramientas case

Descripción expandida de los casos de uso

Debido a que nuestro problema es bastante simple y los casos de uso no conllevan mucha complejidad, ahora nos dedicaremos a describir todos los casos de uso en su **formato expandido** (si fuesen muchos o muy complejos, escogeríamos los más importantes).

La descripción que hemos hecho de nuestros casos de uso siguiendo las pautas de los apartados anteriores es la que se muestra seleccionando el siguiente enlace:

Descripción de casos de uso

Con esto damos por finalizada esta primera fase de planificación de nuestro proceso de desarrollo orientado a objetos. Como hemos podido ver, se han detectado dos casos de uso como primarios, por lo que serán estos dos casos de uso los que escogeremos para realizar el primer paso del ciclo de desarrollo iterativo. Este ciclo llevará sus respectivas fases de análisis y diseño.

Vamos a ello.

Desarrollo orientado a objetos usando uml y herramientas case

Ya sabemos que nuestro proceso de desarrollo es un proceso **iterativo** y que consta de varios **ciclos de desarrollo**. En este **primer** ciclo ya hemos comentado que vamos a tratar los dos casos de uso primarios, ya que son los más importantes y críticos.

Por otro lado sabemos que la fase de **análisis** dentro de un ciclo de

desarrollo sigue investigando sobre el problema, más que investigar en cómo se implementará.

De apartados anteriores sabemos los pasos que debemos seguir en esta fase de análisis de un ciclo de desarrollo. Uno de estos pasos era describir los casos de uso en su formato expandido de nuestro problema, pero debido a que nuestro problema era bastante simple, esto lo hicimos en la fase de planificación, por lo que esta tarea ya la tenemos de la fase anterior.

Nos vamos a centrar en las otras actividades de esta fase de análisis que describiremos en los siguientes apartados.

Desarrollo orientado a objetos usando uml y herramientas case

Construcción del modelo conceptual

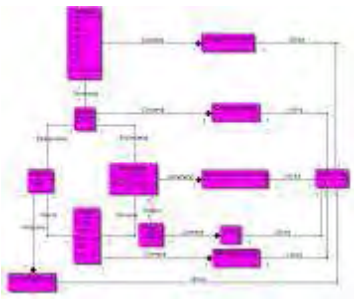
A la hora de construir el **modelo conceptual** para nuestro caso de estudio, debemos:

- identificar los conceptos del dominio de nuestro problema,
- identificar las asociaciones y su multiplicidad e
- identificar los atributos.

Sabemos que hay algunas categorías de conceptos que son candidatas a ser representadas. Además en los modelos conceptuales solemos incluir conceptos como "CatalogoX" indicando un contenedor de conceptos X. En nuestro caso de estudio aparecen los conceptos CatalogoPeliculas, CatalogoCopias y CatalogoSocios, como contenedores de dichos conceptos. Además hemos incluido LibroAlquileres como contenedor de Alquileres, HistorialDevoluciones como contenedor de Devoluciones y Caja como contenedor de Pagos. De ahí que utilicemos asociaciones de agregación indicando que son contenedores. También hemos representado el concepto video-club.

Las asociaciones y su multiplicidad las sacamos de las relaciones existentes entre los conceptos.

Los atributos los identificamos mirando los requisitos de nuestro problema y el conocimiento que vamos adquiriendo del mismo.

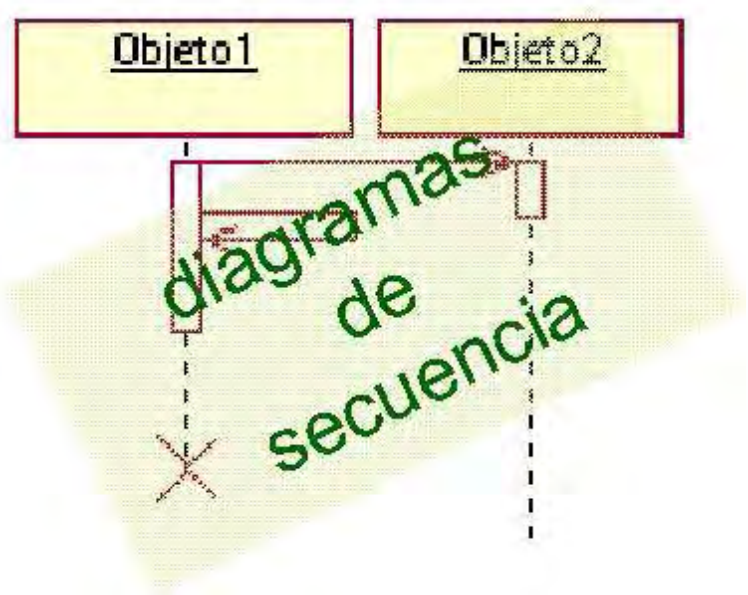


Un posible modelo conceptual para nuestro caso de estudio sería el siguiente (haz clic sobre la imagen para verla a pantalla completa)

Desarrollo orientado a objetos usando uml y herramientas case

Definición de los diagramas de secuencia del sistema

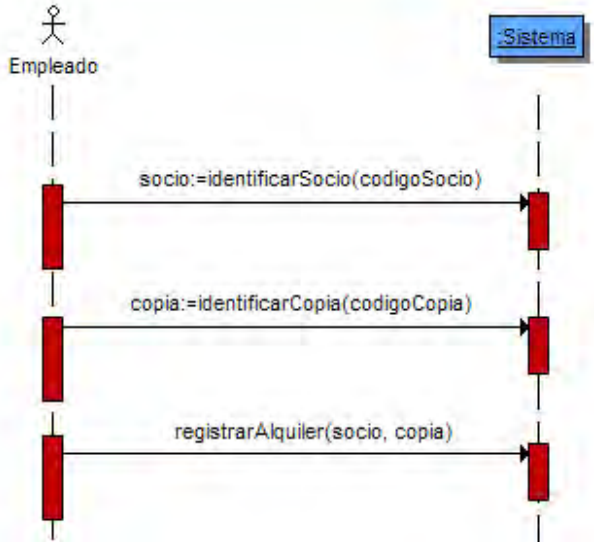
La siguiente **actividad** que debemos realizar es la **construcción de los diagramasde secuencia** para los casos de uso que estamos tratando en el ciclo actual de desarrollo, que en particular para este primer ciclo de desarrollo serán los dos casos de uso clasificados como primarios: Alquilar copia y Devolver copia.



Para realizar estos diagramas de secuencia partiremos del caso de uso y en particular nos centraremos en el curso normal de eventos descrito en dicho caso de uso. De ahí identificaremos los mensajes que se intercambian entre los actores y el sistema. En estos diagramas de secuencia sólo representamos los **actores** que interactúan directamente con el sistema, por lo que Socio no aparecerá en ellos, ya que como mencionamos anteriormente un Socio no interactúa directamente con el sistema sino que es el Empleado el que lo hace. Los diagramas resultantes son los siguientes:

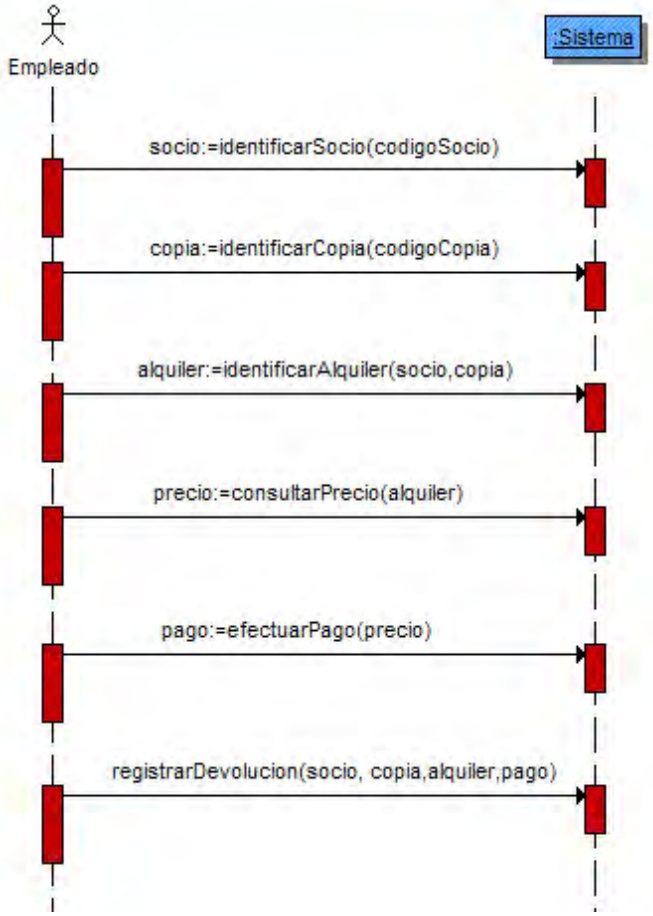
- **Diagrama de secuencia "Alquilar copia"**

Caso de uso Alquilar copia.
Actores Socio (iniciador), Empleado.
Propósito Realizar el alquiler de una copia de una copia por un socio.
Resumen El usuario solicitará alquilar una copia para lo cuál facilitará su código de socio y el código de la copia a alquilar. El empleado lo introducirá en el sistema y éste registrará dicho alquiler.
Tipo Primario.
Referencias cruzadas R6.
Curso normal
1.- Este caso de uso comienza cuando un socio se acerca al Terminal del video club con una copia que ha elegido para alquilar.
2.- El empleado identifica al socio pidiéndole su código de socio e introduciéndolo en el sistema.
3.- El sistema nos muestra la información referente a este socio.
4.- El empleado corroborará los datos con el.
5.- El socio informa de la copia que desea alquilar.
6.- El empleado introduce el código de la copia en el sistema y aceptará la operación.
7.- El sistema registra el alquiler de la copia por el socio en la fecha de hoy.
8.- El empleado entrega la copia al socio.
Curso alternativo



- **Diagrama de secuencia "Devolver copia"**

Caso de uso Devolver copia.
Actores Socio (iniciador), Empleado.
Propósito Realizar la devolución de una copia por parte de un cliente.
Resumen El usuario solicita devolver una copia para lo cuál facilita el código de la copia y su código de socio.
El empleado informará del precio, cobrará y el sistema registrará dicha devolución.
Tipo Primario.
Referencias cruzadas R7.
Curso normal
1.- Este caso de uso comienza cuando un socio se acerca al Terminal del video club con una copia para devolver.
2.- El empleado identifica al socio pidiéndole su código de socio y lo introduce en el sistema.
3.- El sistema nos muestra la información referente a este socio.
4.- El empleado corroborará los datos el.
5.- El socio nos dirá la copia que quiere devolver.
6.- El empleado introducirá el código de la copia que el socio quiere devolver.
7.- El sistema nos mostrará el precio de dicho alquiler.
8.- El empleado informará al socio del precio.
9.- El socio pagará la devolución de dicha copia.
10.- El empleado aceptará la operación y esta quedará registrada en el sistema.
Curso alternativo
3a.- El código de socio no existe, por lo que abortaremos la operación.



Desarrollo orientado a objetos usando uml y herramientas case

Definición de los contratos del sistema

Partiendo de los diagramas de secuencia, y una vez identificadas las operaciones del sistema, ahora debemos describir mediante **contratos** el comportamiento esperado del sistema para cada operación.

Para describir los contratos empezaremos describiendo las responsabilidades de nuestro contrato y las precondiciones y poscondiciones para luego describir las demás secciones. Es importante que no se nos olvide indicar las asociaciones que se forman en el apartado de poscondiciones.

La descripción de los contratos que hemos realizado es la que sigue, disponible seleccionando el siguiente enlace:

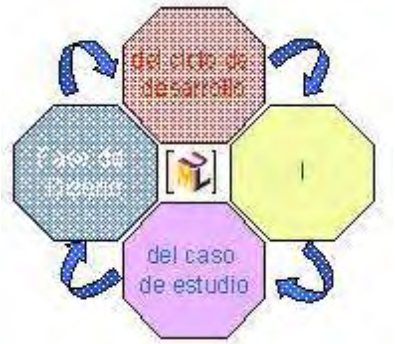
Descripción de los contratos

Desarrollo orientado a objetos usando uml y herramientas case

Ya vimos las actividades a realizar en esta fase ¿pero las tienes claras? ¿Sabes a qué debemos dedicarnos en la misma?

Pues en esta fase es en la que nos dedicaremos a la **creación de una solución lógica** para satisfacer los requisitos, basándonos en el conocimiento adquirido en la fase de análisis del ciclo de desarrollo actual.

Una de las **actividades** de esta fase era describir los casos de uso reales de nuestro problema, pero debido a que nuestro problema era bastante simple los casos de uso de alto nivel y los casos de uso reales son iguales, exceptuando que deberíamos describir las interfaces con las que interactuarán los actores y los informes que se mostrarán a los mismos, pero como ya dijimos anteriormente no nos vamos a detener en esta tarea ya que queda fuera del propósito de esta unidad. Por tanto la descripción de los casos de uso asumimos que es la misma (en verdad es la misma a falta de las interfaces).



También sabemos que no nos vamos a parar en la definición de las interfaces ni del esquema de la base de datos, ya que como en el caso anterior queda fuera del propósito de esta unidad.

Pasemos a centrarnos en las principales actividades de esta fase de diseño.

Desarrollo orientado a objetos usando uml y herramientas case

Diagramas de colaboración

La construcción de los **diagramas de colaboración** es una tarea crucial en el desarrollo orientado a objetos, por lo que deberíamos dedicarle el tiempo necesario para llegar a un buen diseño.

Los diagramas de colaboración los construiremos partiendo de los contratos del sistema. Debemos asignar las responsabilidades adecuadas a cada objeto (qué objetos son los encargados de crear otros objetos, qué objetos tienen el conocimiento, ...).

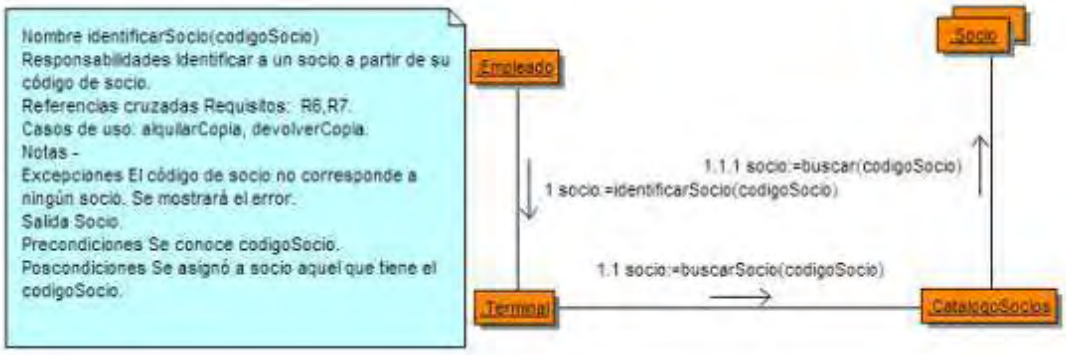
Veamos los diagramas de colaboración que hemos construido para cada contrato.

En ellos hemos incluido un objeto denominado Terminal, que será el que recibirá los eventos del sistema generados por el empleado. Recuerda que en el modelo conceptual teníamos un concepto denominado video-club (pertenecía al dominio del mundo real), pero aquí hemos preferido utilizar el objeto Terminal que representará nuestro Terminal físico (hardware y software).

- **Diagrama de colaboración identificarSocio.**

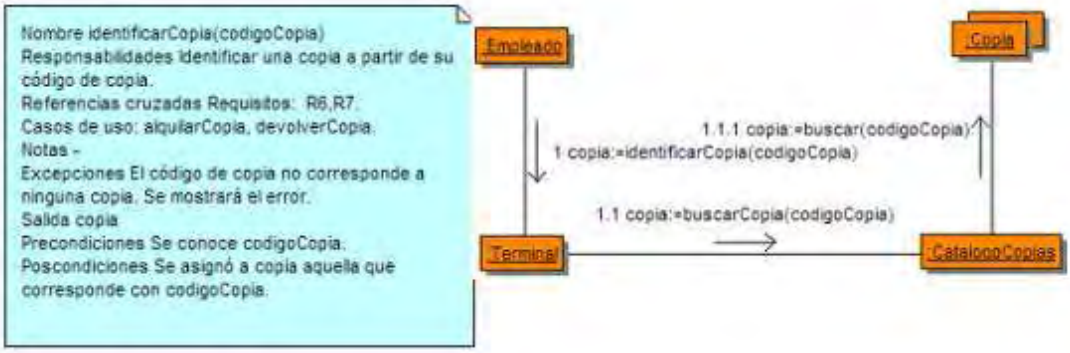
Este diagrama es bastante sencillo. La responsabilidad de buscar un socio la tendrá la clase Terminal

que será la que tiene conocimiento del CatalogoSocios. El Terminal enviará un mensaje a la clase CatalogoSocios indicando que quiere buscar un socio a partir de su código. Esta clase a su vez, enviará el mensaje buscar a la colección de socios que posee.



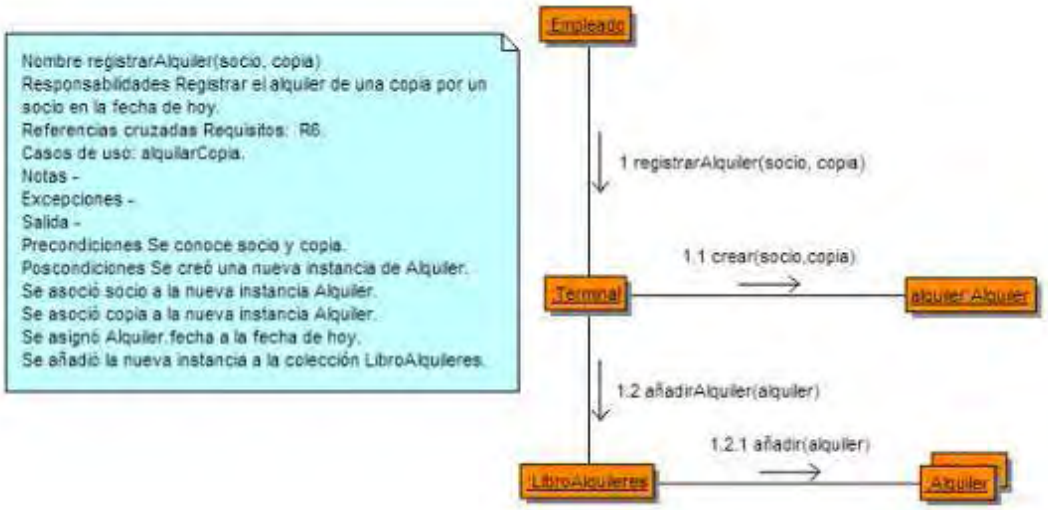
- **Diagrama de colaboración identificarCopia.**

Este diagrama también es bastante sencillo. La responsabilidad de buscar una copia la tendrá la clase Terminal que será la que tiene conocimiento del CatalogoCopias. El Terminal enviará un mensaje a la clase CatalogoCopia indicando que quiere buscar una copia a partir de su código. Esta clase a su vez, enviará el mensaje buscar a la colección de copias que posee.



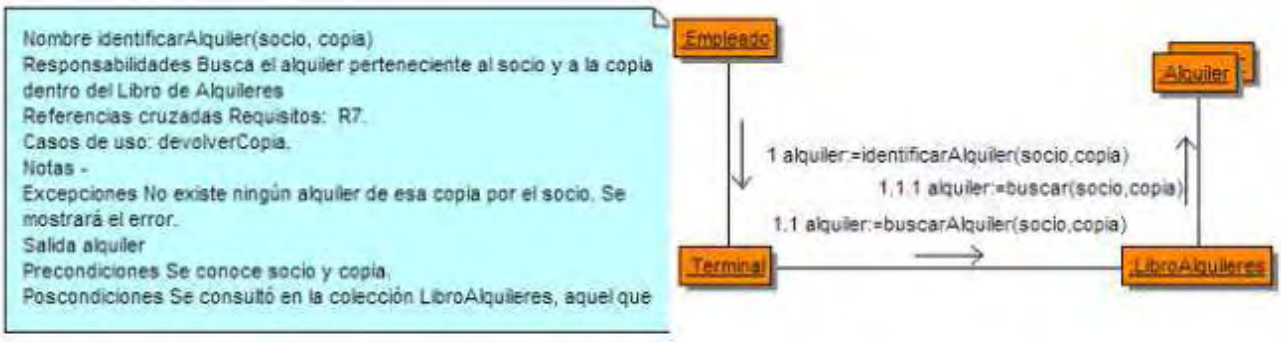
- **Diagrama de Colaboración registrarAlquiler.**

Este diagrama tampoco es mucho más complicado. Debido a que el Terminal también tendrá conocimiento sobre el LibroAlquileres, el Terminal será el encargado de crear la instancia del nuevo Alquiler y posteriormente añadirlo a LibroAlquileres, para que éste lo añada a su colección de Alquileres.



- **Diagrama de Colaboración identificarAlquiler.**

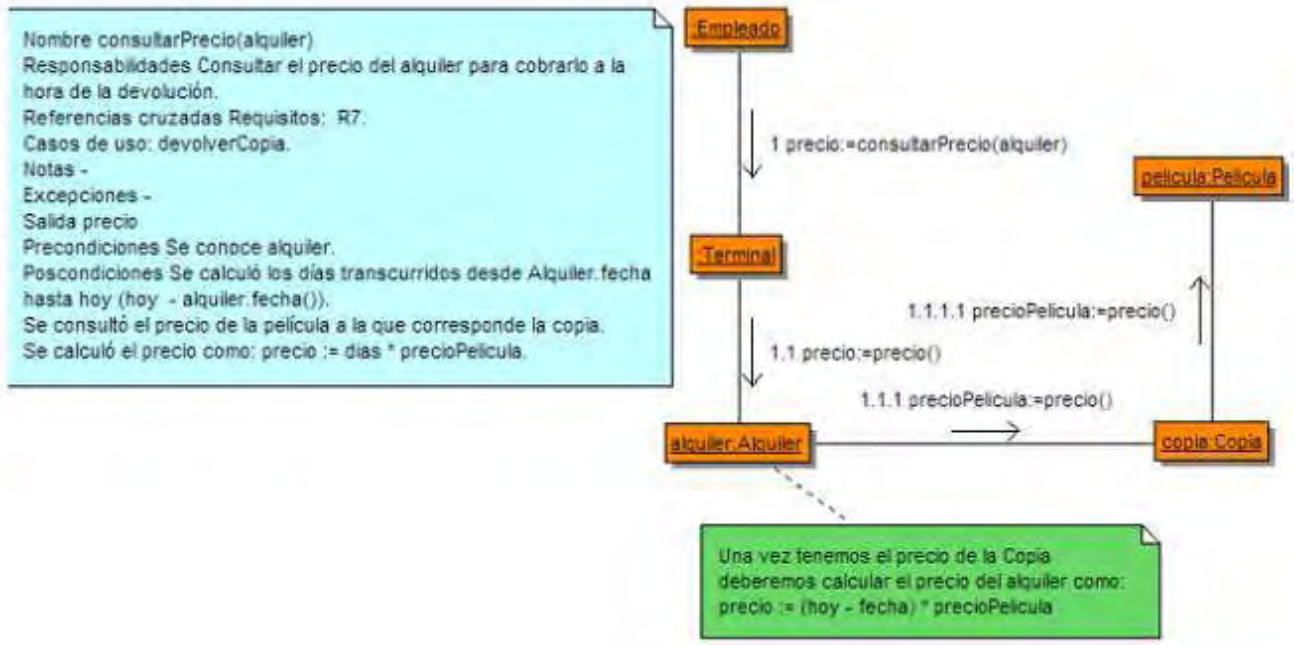
Este diagrama es muy parecido a los dos primeros. El Terminal es el que conoce LibroAlquileres, por lo que éste enviará un mensaje buscarAlquiler a LibroAlquileres para que éste lo busque en la colección de Alquileres que posee.



- **Diagrama de Colaboración consultarPrecio.**

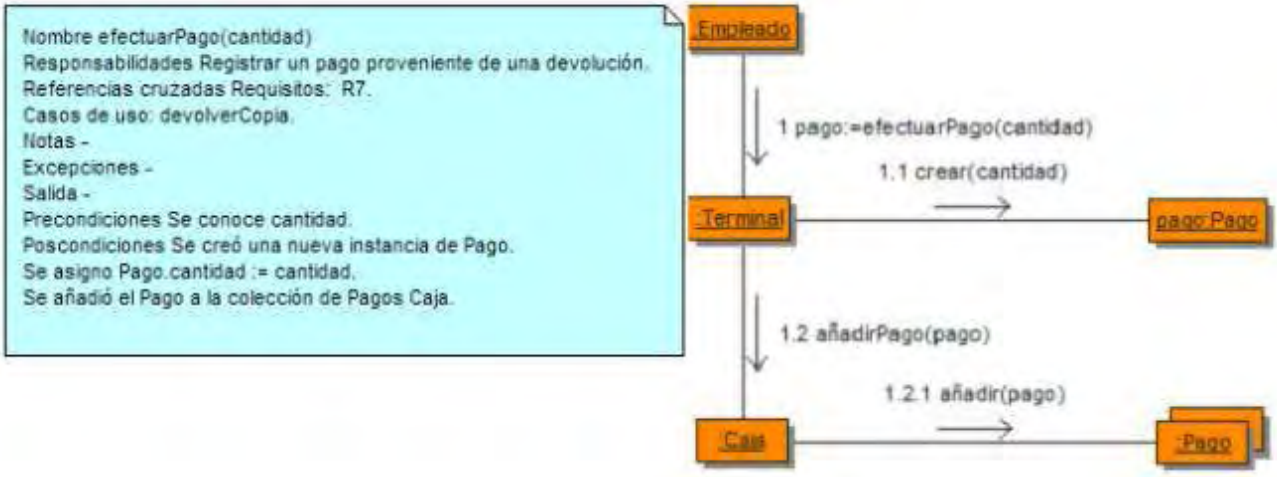
Este diagrama muestra cómo se van delegando las responsabilidades en las distintas clases para llegar al objetivo.

- Un Alquiler es el responsable de conocer su precio, ya que conoce la fecha del mismo y sabe la copia sobre la que se realiza el alquiler.
- Además una copia conocerá la película de la que es copia.
- Una película conoce su precio. Por tanto, el alquiler le dice a copia que calcule su precio, y ésta a su vez se lo dice a su película.
- Una vez que el alquiler conoce el precio de la película, calcular el precio del alquiler es trivial, ya que sólo deberemos saber los días transcurridos y multiplicarlos por el precio de la película.



- **Diagrama de Colaboración efectuarPago.**

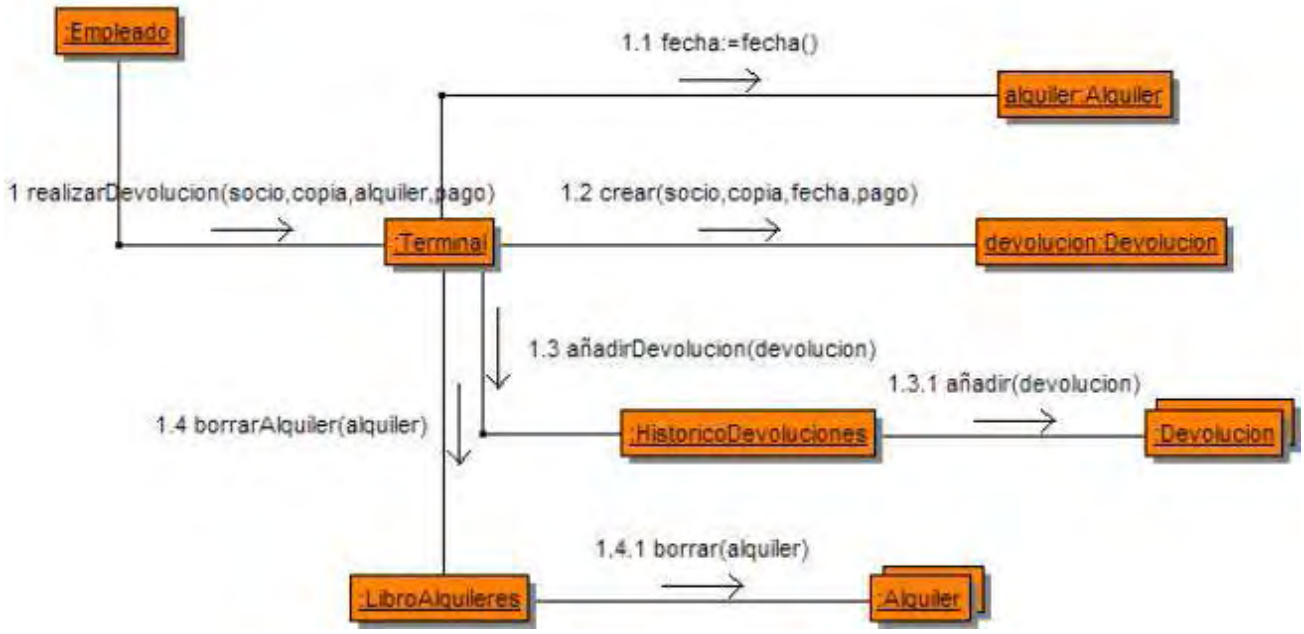
Este diagrama también es bastante simple. El Terminal creará una instancia de Pago, y añadirá ésta a Caja que a su vez la añadirá a la colección de Pagos que Caja posee.



- **Diagrama de colaboración registrarDevolución.**

Éste es uno de los diagramas más complicados de este ciclo de desarrollo, pero que tampoco lo es mucho después de haber visto los anteriores. Lo primero que hace es averiguar la fecha del alquiler, enviando el mensaje fecha() al alquiler. Lo siguiente es crear una instancia de la clase Devolución que posteriormente se añade al HistóricoDevoluciones para que éste la añada a su colección de Devoluciones. Finalmente borraremos el Alquiler del LibroAlquileres.

Nombre registrarDevolucion(socio, copia, alquiler, pago)
Responsabilidades Registrar la devolución de una copia por parte de un socio, anotando la fecha en la que la alquiló y la fecha de hoy como fecha de devolución y su precio. Borrar el alquiler de la copia por dicho socio.
Referencias cruzadas Requisitos: R7.
Casos de uso: devolverCopia.
Notas -
Excepciones -
Salida -
Precondiciones Se conoce socio, copia, alquiler y pago.
Poscondiciones Se creó una nueva instancia de Devolución.
Se asoció el socio a la Devolución.
Se asoció la copia a la Devolución.
Se asoció el pago a la Devolución.
Se asignó Devolucion.fecha_alquiler := Alquiler.fecha().
Se asignó Devolucion.fecha_decolucion:=hoy.
Se borró Alquiler de la colección LibroAlquileres.
Se añadió Devolucion a la colección HistoricoDevoluciones.



Desarrollo orientado a objetos usando uml y herramientas case

Diagrama de clases de diseño

Ésta es la última de las actividades de esta fase de **diseño** para el ciclo de desarrollo actual. Para construir el diagrama de clases de diseño, seguiremos los pasos descritos en el apartado dedicado a dichos diagramas dentro de esta unidad.

La idea es:

- partir de los diagramas de colaboración, para identificar las clases participantes y
- representarlas en un diagrama de clases de diseño.
- Luego duplicar los atributos que aparezcan en el modelo

conceptual.

- Para cada diagrama de colaboración iremos viendo los mensajes que una clase envía a otra y los añadiremos como métodos de la clase a la que se envían. En los diagramas de clase no se suelen mostrar los métodos que provienen de mensajes del tipo crear, ya que éstos se corresponderán con constructores de la clase y lo único que consiguen es hacer más complejo el diagrama sin aportarnos más información.
- Seguidamente añadiremos la información de tipo a los atributos y métodos.
- Finalmente añadiremos las asociaciones necesarias para soportar la visibilidad de atributos (si una clase envía un mensaje a otra, o lo que es lo mismo invoca un método de esa clase, la primera debe tener visibilidad sobre la segunda para poder invocar dicho método).

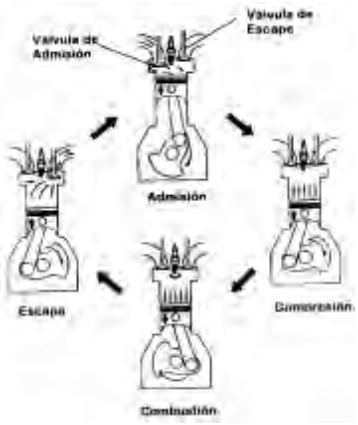
El **diagrama** resultante después de seguir estos pasos es el siguiente (haz clic sobre la imagen para verla a pantalla completa).

Con la realización de este diagrama podemos dar por **concluido** este ciclo de desarrollo (aunque no debemos olvidar que nos quedaría la generación del código a partir del diagrama de clases de diseño, la implementación de los métodos que ya sabemos qué y cómo tienen que hacer las cosas y las pruebas. Cuando en apartados posteriores hablemos del modelado usando herramientas CASE os daremos un ejemplo de cómo estas herramientas hacen el trabajo de generación de esqueletos de las clases para nosotros).

Ahora, para continuar con el desarrollo de nuestro sistema orientado a objetos, deberíamos hacer una nueva iteración en el ciclo de desarrollo, ya que nuestro sistema todavía no cumple todos los requisitos (algo normal, ya que en este ciclo sólo abordamos dos casos de uso, que aunque eran los más importantes no reflejaban todos los requisitos de nuestro sistema).

Desarrollo orientado a objetos usando uml y herramientas case

Y con esto, ¿ya hemos terminado nuestro trabajo?



Pues no, ya que como sabes, el proceso de desarrollo orientado a objetos que estamos siguiendo es un proceso **iterativo**, y estas iteraciones se hacen sobre los ciclos de desarrollo.

En los apartados anteriores abordamos el primero de los ciclos (además de una primera fase de planificación). En este primer ciclo escogimos los casos de uso más significativos, importantes y/o críticos. Pero **debemos continuar iterando hasta que no nos queden casos de uso que abordar, con lo que podremos dar por concluido nuestro proceso de desarrollo** (sin olvidar que generalmente esto nunca acaba, ya que siempre surgen mejoras, nuevas funcionalidades, fallos, etc...).

Para abordar este segundo ciclo lo primero que deberíamos hacer es **escoger dos o tres casos de uso** sobre los que aplicar todo el ciclo. Como hemos visto un ciclo de desarrollo consta de una fase de análisis y una fase de diseño, y a su vez cada fase consta de una serie de actividades que debemos realizar. Por tanto deberíamos repetir

los pasos dados en los apartados 8 y 9 de esta unidad, pero aplicados esta vez a los nuevos casos de uso.

Como el primer ciclo aborda los casos de uso más críticos, se puede decir que la parte más difícil ya la hemos realizado. Ya sólo nos queda ir maquillando nuestro sistema para que al final cumpla con todos los requisitos.

Podríamos exponer aquí cómo realizar el siguiente ciclo de desarrollo (e incluso un tercer ciclo, ya que con eso abarcaríamos todos los casos de uso), pero eso haría que nos extendiéramos demasiado, sin aportar conocimientos nuevos para ti. Por lo que hemos decidido **dejarte a ti este segundo ciclo de desarrollo**, para que te sirva de práctica, y vayas perdiendo el "miedo escénico". Así que ya te contaremos en la tarea de esta unidad lo que debes realizar.

Por tanto, y sin extendernos más, vamos a pasar a ver qué es eso de las herramientas CASE y cómo podemos utilizarlas para llevar a cabo un proceso de desarrollo orientado a objetos como el que hemos estado estudiando en esta unidad.

Desarrollo orientado a objetos usando uml y herramientas case

Caso:

***Carmen** está muy contenta debido al gran progreso que ha tenido **Víctor**, y lo bien que se ha defendido a la hora de enfrentarse a su primer proyecto. Pero **Carmen** nota que **Víctor** no está tan contento, y además lo ve atareado pasando diagramas con una herramienta de diseño gráfico. **Carmen** se acerca y le pregunta qué le pasa y **Víctor** responde que está intentado pasar los diagramas al ordenador para poder imprimirlos, ya que antes los habían realizado sobre papel, y a **Víctor** le obsesiona la limpieza.*

***Carmen** se sonríe y le dice a **Víctor** que está perdiendo el tiempo. Entonces **Víctor** pone cara de asombrado y le pregunta por qué cree que está perdiendo el tiempo. **Carmen** le dice que existen en el mercado herramientas que permiten representar los diagramas UML que ellos han hecho sobre papel de una manera más cómoda que una herramienta de diseño gráfico. Además le comenta que estas herramientas no sólo permiten modelar dichos diagramas, sino que además son capaces de generar la documentación y el código de nuestro problema y todo de una forma simple.*

A estas alturas te estarás preguntando ¿y después de tanto análisis y diseño, tengo que ponerme a picar el código en mi ordenador? Entonces ¿no he estado perdiendo el tiempo? La respuesta es NO.

Lo primero es que todo el proceso que hemos seguido nos ha ayudado a **comprender** perfectamente nuestro problema, para así poder llegar a realizar un diseño adecuado, que finalmente nos llevará a la implementación de dicho sistema.

Pero lo segundo es que **existen en el mercado herramientas que nos permiten generar el esqueleto de todas las clases utilizadas en nuestro diseño y generar toda la documentación necesaria**. Esto que a lo mejor parece una tontería para un ejemplo simple como podía ser nuestro caso de estudio, no es una tontería, sino que es indispensable para llegar a diseños robustos que puedan ser tratados por varios programadores a la vez cuando hablamos de sistemas mucho más complejos.

Estas herramientas se denominan herramientas CASE (Computer Aided Software Engineering - Ingeniería del software asistida por ordenador) cuya finalidad es automatizar los aspectos clave de todo el proceso de desarrollo de un sistema, aumentando la productividad en el desarrollo reduciendo costes (tiempo y dinero). Su objetivo es conseguir la generación automática de programas desde una especificación de alto nivel.

Para saber más

Para ampliar conocimientos sobre UML y herramientas CASE te recomendamos que visites la página de Vico. En ella podrás encontrar algún ejemplo, e incluso cursos sobre UML.

Página de Vico

http://www.vico.org/

Desarrollo orientado a objetos usando uml y herramientas case

Objetivos de las herramientas CASE

Los **objetivos** más importantes de estas herramientas son:

1. Mejorar la **productividad** en el desarrollo y mantenimiento del software.
2. Aumentar la **calidad** del software.
3. Mejorar el **tiempo** y coste de desarrollo y mantenimiento de los sistemas informáticos.
4. Mejorar la **planificación** de un proyecto.
5. Aumentar la biblioteca de **conocimiento** informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
6. **Automatizar** el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
7. Ayuda a la **reutilización** del software, portabilidad y estandarización de la documentación.
8. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
9. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Desarrollo orientado a objetos usando uml y herramientas case

Clasificación de las herramientas CASE

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar en base a los parámetros siguientes:

- Amplitud de las **tareas** que automatizan.
- Las **fases** del ciclo de vida del desarrollo de sistemas que cubren.

Si nos atenemos a la primera clasificación podríamos tener:

- **TOOLKIT**: Es una colección de herramientas integradas que permiten automatizar un conjunto de tareas de algunas de las fases del ciclo de vida del sistema informático: Planificación estratégica, análisis, diseño y generación de programas.
- **WORKBENCH**: Son conjuntos integrados de herramientas que dan soporte a la automatización del proceso completo de desarrollo del sistema informático. Permiten cubrir el ciclo de vida completo. El producto final aportado por ellas es un sistema en código ejecutable y su documentación.

Si nos centramos en la segunda clasificación, tendríamos:

- **UPPER CASE**: Abarca las fases de planificación y requerimientos de desarrollo funcional.
- **MIDDLE CASE**: Fases de análisis y diseño.
- **LOWER CASE**: Fase de generación de código, test e implantación.

Para saber más

Te presentamos una interesante monografía sobre herramientas CASE, que completa

lo expuesto en esta unidad y abunda en los conceptos expuestos aquí.

Herramientas CASE

http://www.monografias.com/trabajos14/herramicase/herramicase.shtml [versión en cache]

Autoevaluación

Una herramienta CASE es...

a) Una aplicación informática.
b) Una herramienta que nos ayuda en el desarrollo de nuestras aplicaciones.
c) Una herramienta que nos permite generar código a partir del modelado de nuestro problema.
d) Todas las respuestas anteriores son correctas.

Comprobar

Diferentes alternativas existentes en el mercado

¿Crees que hay muchas herramientas CASE o es sólo un concepto teórico?

Pues actualmente la **oferta de aplicaciones** informáticas disponibles en el mercado que pueden ser consideradas CASE es amplísima y también muy variada en cuanto a concepción, diseño y cantidad de funciones implementadas. Mientras unas sólo incluyen unas pocas capacidades CASE o se concentran en una sola fase del ciclo de vida del software, otras en cambio abarcan todas las fases, desde el diseño preliminar hasta el mantenimiento y explotación. Por supuesto esto también se refleja en el coste económico y de aprendizaje de las mismas.

En el siguiente enlace podrás ver una tabla con enlaces a algunas de las aplicaciones CASE actuales:

Tabla con enlaces

Como ves la lista es muy extensa, pero en este momento nos interesan aquellas aplicaciones CASE que nos ayuden en nuestro **proceso de desarrollo orientado a objetos**. En el siguiente apartado empezarás a conocer en la práctica una de ellas, **BOUML**, que nos servirá como herramienta CASE para el diseño de nuestros sistemas orientados a objetos. No olvides que lo aprendido para esta aplicación será aplicable a casi cualquier otra.

Para saber más

Para que te hagas una idea de la envergadura del tema que estás estudiando, y compruebes la inmensa variedad de soluciones existentes en el mercado actual es muy interesante que visites el siguiente enlace que recopila enlaces a herramientas CASE clasificadas por categorías. Impresionante, ¿Verdad?

Lista de aplicaciones CASE clasificadas

http://www.cs.queensu.ca/Software-Engineering/toolcat.html [versión en cache]

Caso:

***Víctor** se ha alegrado mucho al conocer que hay herramientas que permiten hacer el trabajo más engorroso que es la representación de los modelos. Además le satisface que estas herramientas sean capaces de generar documentación y código.*

*Pero **Víctor** ha visto la lista de herramientas CASE y está totalmente confundido, ya que no sabría por cuál decidirse, por lo que le pregunta a **Carmen**. Ella le contesta que casi todas permiten modelar y generar documentación y código. **Carmen** le comenta a **Víctor**, que, como ya sabe, en **SI Andalucía** se apuesta por el software libre siempre que éste cumpla las expectativas esperadas. **Carmen** le dice que la herramienta que llevan tiempo utilizando en **SI Andalucía** es una herramienta CASE de uso libre, que proporciona mucha versatibilidad y que es bastante ligera. **Víctor** le pregunta:*

¿Y de que herramienta estamos hablando?

***Carmen** le responde:*

Pues la herramienta se llama BOUML y verás que es muy sencilla de utilizar.

De todas las herramientas vistas ¿cuál es la que mejor se adecua a nuestras expectativas sin tener que desembolsar una gran cantidad de dinero?

Como hemos visto en el apartado anterior hay **muchas herramientas** CASE disponibles y cada una con unas funcionalidades y características propias. Para seguir nuestro proceso de desarrollo orientado a objetos nos interesa una herramienta CASE que nos permita trabajar con el lenguaje de modelado UML, que ha sido el que hemos utilizado para seguir nuestro proceso de desarrollo.

Nosotros hemos optado por una herramienta llamada **BOUML**, y esta elección no ha sido caprichosa, sino que nos hemos basado en las características de la misma y los beneficios que nos aporta.

Veamos cuáles han sido las decisiones en las que nos hemos basado para elegir dicha herramienta.

- BOUML es una herramienta que cumple con las especificaciones del estándar UML 2.0, por lo que nos viene como anillo al dedo para nuestro problema.
- BOUML es capaz de generar código en los lenguajes de programación C++, Java e IDL. El que más nos interesa en éste y posteriores módulos es el lenguaje Java, que será el utilizado en el módulo denominado "Programación en Lenguajes Estructurados".
- BOUML es software libre distribuido bajo la licencia GNU General Public License, por lo que su utilización es libre y gratuita y además puede ser redistribuida y/o modificada dentro de los términos de dicha licencia.
- BOUML es multiplataforma, por lo que puede ser ejecutada bajo los Sistemas Operativos Unix/Linux/Solaris, MacOS y Windows.
- BOUML es una herramienta bastante rápida y no requiere una gran cantidad de memoria para tratar con cientos de clases.

Como veis los beneficios son muchos y su coste nulo (si exceptuamos el coste de aprendizaje que nos supone). Por tanto, ¡ vamos a pasar a la acción!.

Desarrollo orientado a objetos usando uml y herramientas case

Descarga e instalación de BOUML

Como hemos dicho anteriormente BOUML es una herramienta libre por lo que la podemos descargar y utilizar sin coste económico alguno que repercuta en nuestros bolsillos.

Para saber más

Ésta es la página principal de BOUML, que te recomendamos que visites para navegar por sus distintas secciones.

Página principal de BOUML

http://bouml.free.fr/

Desde este enlace te podrás descargar la última versión de la herramienta para cualquier plataforma. Es imprescindible que la descargues, porque tendrás que utilizar esta herramienta a lo largo de los siguientes apartados, y para la tarea de la unidad.

Zona de descarga de la aplicación BOUML

http://bouml.free.fr/download.html

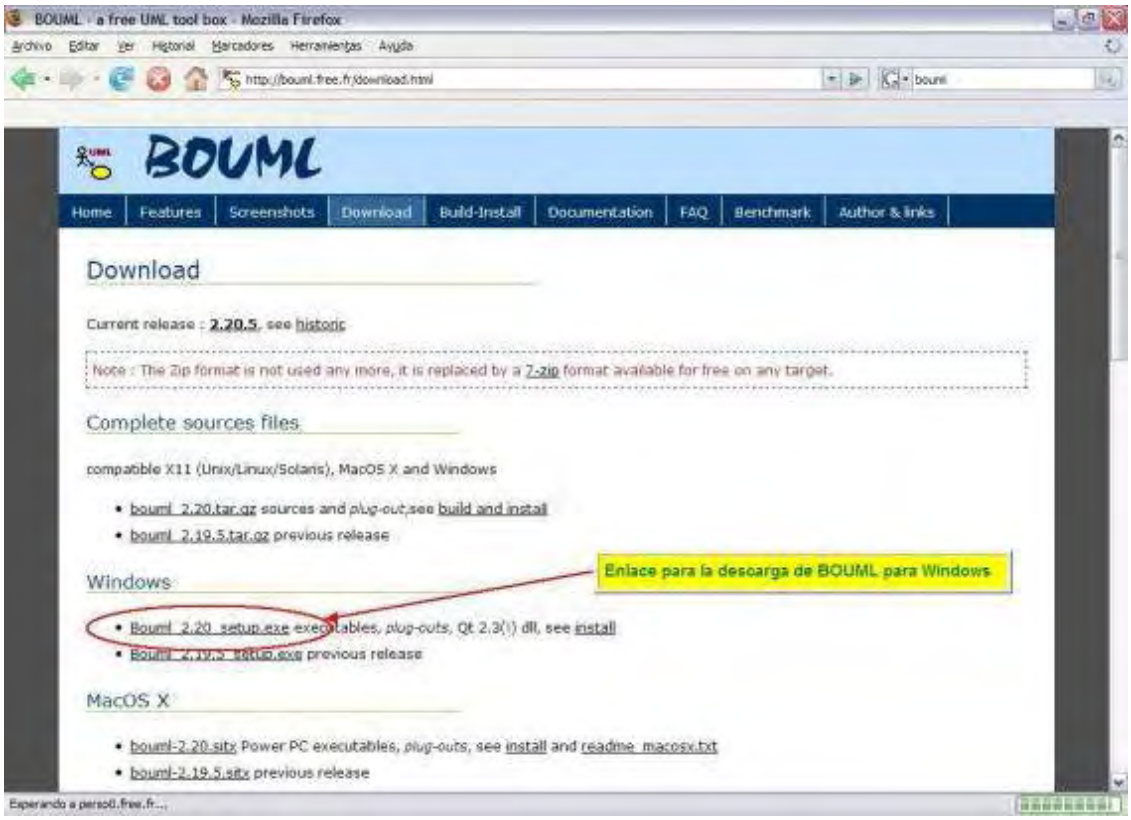
En este enlace podrás acceder a la documentación de BOUML, en la que puedes encontrar un tutorial y el manual de referencia de la herramienta.

Zona de descarga de la documentación

http://bouml.free.fr/documentation.html

A continuación te mostramos algunas imágenes comentadas sobre la página principal de BOUML, donde nos encontramos las distintas secciones, y en la cual podemos acceder, entre otras, a la zona de descargas de la aplicación y a la zona de descargas del manual y los tutoriales.

Una vez accedemos al enlace de descarga pasamos a las siguiente página en la que podemos descargarnos la herramienta para cualquier plataforma.



Ésta es la página a la que se accede al pinchar sobre la pestaña "Documentation" desde la página principal de BOUML.

Resulta muy conveniente que te detengas en los diferentes **enlaces** de la página de BOUML. Está en inglés, como suele ocurrir en este tipo de páginas, pero no es difícil de entender. Además, como en otras ocasiones, te animamos a practicar con este idioma, ya que es la lengua de la informática, y es importante para tu formación como profesional habituarte a trabajar con herramientas y documentación disponible en inglés.

Una vez descargado el instalador de BOUML debes ejecutarlo e instalarlo. Es un proceso muy sencillo, pero para que no tengas dudas te hemos preparado una animación con los detalles del proceso.

 **Detalles del proceso**

Autoevaluación

1 Para usar BOUML...

a) Debemos tener una clave de activación del producto, para poder funcionar con él.
b) Podemos descargarlo y utilizarlo durante un periodo de prueba de 30 días.
c) Podemos descargarlo, pero no podemos pasárselo a nuestros amigos.
d) Podemos descargarlo y redistribuirlo bajo los términos de la licencia GPL.

Comprobar

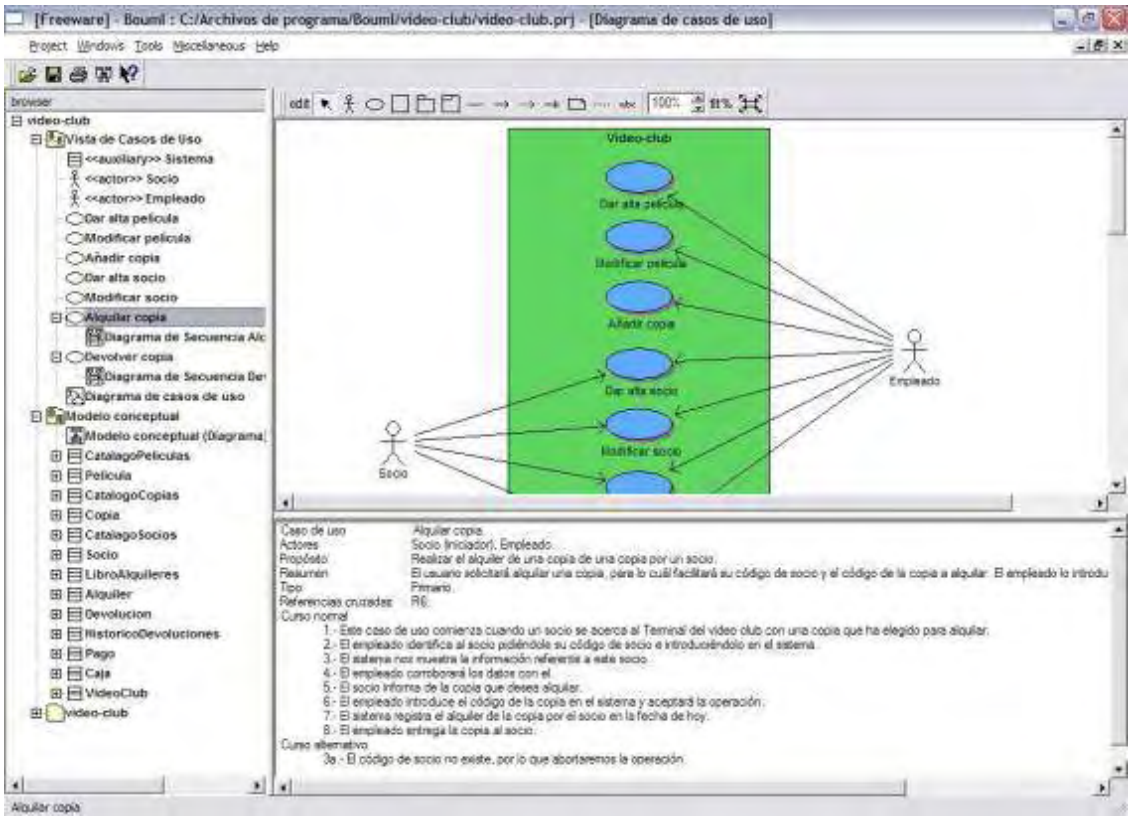
Desarrollo orientado a objetos usando uml y herramientas case

Primeros pasos con BOUML

¿Cómo empiezo a trabajar con BOUML?

Ahora que ya hemos descargado BOUML y lo hemos instalado en nuestro ordenador, es la hora de **arrancar la aplicación** y empezar a trabajar con ella. Después de la instalación se crea un acceso directo en nuestro escritorio (en el caso de Windows) y haciendo doble clic sobre el mismo se empezará a ejecutar nuestra herramienta, y nos aparecerá la pantalla principal de la aplicación con sus tres paneles vacíos.

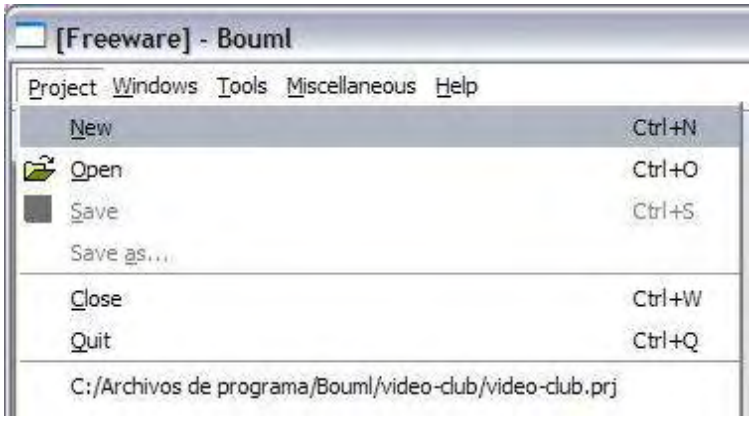
Una vez hecho esto nos aparecerá la pantalla principal de BOUML, cuyo aspecto es el siguiente (este aspecto es después de abrir un proyecto, ya que si es un proyecto nuevo todos los paneles aparecerán vacíos).



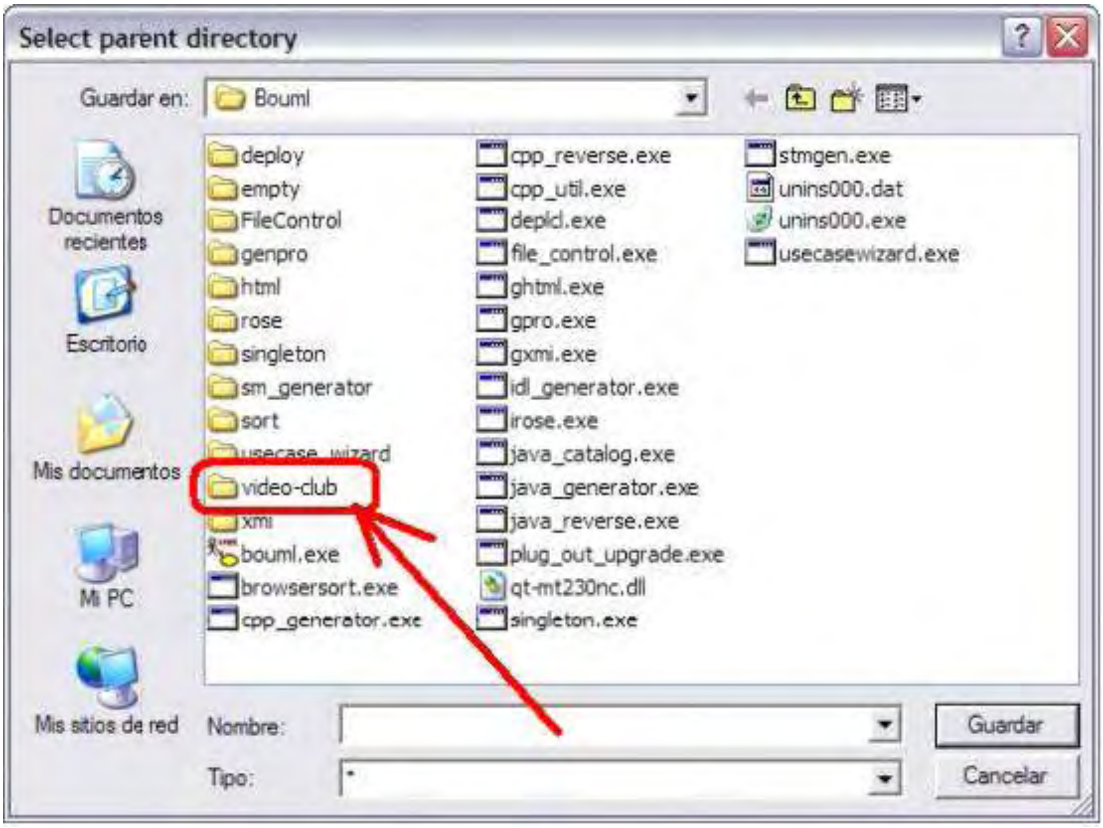
Como podemos observar la pantalla principal está dividida en tres partes:

- **Parte izquierda**: Aparece un árbol de navegación para nuestro proyecto.
- **Parte superior derecha**: En ella aparecen los diagramas.
- **Parte inferior izquierda**: En esta parte nos aparecen los comentarios que hallamos hecho al objeto seleccionado.

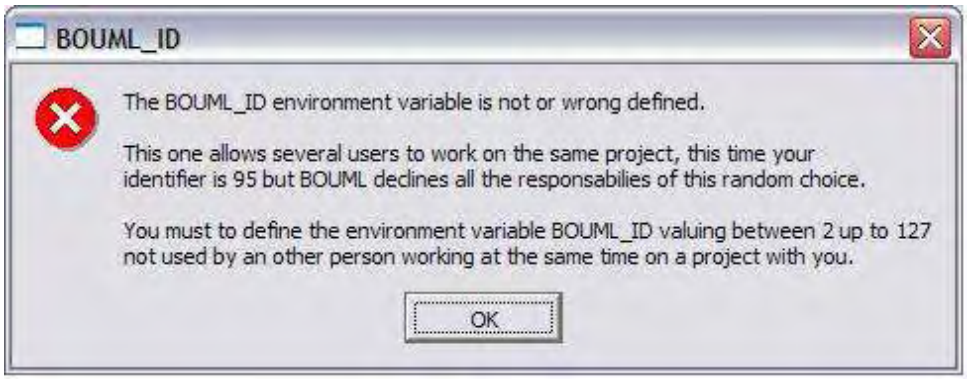
Lo siguiente que debemos hacer es **crear un nuevo proyecto** o abrir un proyecto ya existente. Esto se hace como con cualquier otra aplicación, por medio del menú "Project":



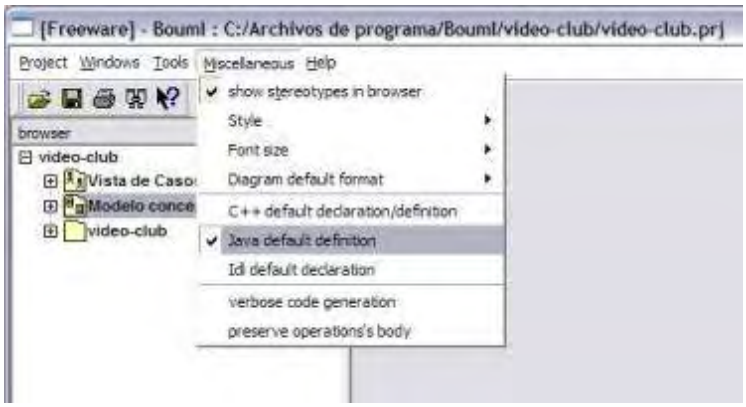
Si elegimos abrir, se nos preguntará donde está almacenado nuestro archivo **.prj** que representa al proyecto. Si elegimos nuevo, se nos preguntará el nombre de nuestro proyecto y el directorio en el que lo queremos guardar. Con el nombre que le demos, BOUML creará un nuevo directorio con ese nombre y ahí almacenará los archivos del proyecto.



Veremos que lo primero que nos aparece es un mensaje de error diciéndonos que no hemos definido un identificador utilizado para accesos concurrentes, que en principio podemos obviar.



Luego se nos avisará de que debemos escoger un lenguaje para la generación de código y nosotros **escogeremos Java**, por medio del menú miscellaneous.



Para clarificar un poco más la creación de nuestro proyecto, os proponemos que veáis esta **simulación** sobre la creación de nuestro proyecto de videoclub, para luego seguir creando los demás

artefactos necesarios para llegar a nuestro objetivo.

Simulación sobre la creación de nuestro proyecto de videoclub

---

Desarrollo orientado a objetos usando uml y herramientas case

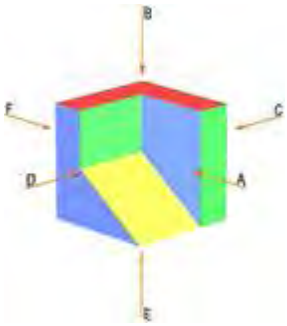
Contenedores y vistas en BOUML

Ya hemos visto como crear un proyecto con nuestra herramienta, pero, ¿cómo **organizar** todos nuestros diagramas y modelarlos para llegar a nuestro objetivo?

BOUML organiza el trabajo por medio de la utilización de y/o vistas en las que podremos incluir los objetos para poder modelar los diagramas necesarios y así seguir nuestro proceso de desarrollo orientado a objetos.

El contenedor que utiliza BOUML es el paquete. El proyecto en sí mismo es un paquete y será la raíz de nuestro árbol de navegación. Un paquete se utiliza para la generación de código y el proyecto se utiliza para crear otras vistas de las que no deseamos generar código. En estos contenedores podemos incluir las siguientes vistas:

- Otro **paquete**.
- **Vista de casos de uso**. La vista de casos de uso se utiliza para definir los actores de nuestro sistema, los casos de uso, los diagramas de casos de uso, de secuencia, de colaboración y de objetos. Esta vista la utilizaremos para nuestra primera fase de planificación y para algunas de las tareas de la fase de análisis de un ciclo de desarrollo.
- **Vista de clases**. La vista de clases puede contener clases, actividades, estados, diagramas de clases de diseño, de objetos, de secuencia y de colaboración. Esta vista se utiliza para la fase de diseño de un ciclo de desarrollo y también la podemos utilizar para representar el modelo conceptual de nuestro problema.
- **Vista de componentes**. La vista de componentes puede contener componentes y diagramas de componentes. Esta vista no la utilizaremos ya que tampoco hemos utilizado estos tipos de diagramas en nuestro proceso de desarrollo.
- **Vista de despliegue**. Esta vista puede contener nodos, artefactos y diagramas de despliegue. En nuestro proceso de desarrollo nosotros no hemos utilizado los diagramas de despliegue ya que hemos tratado con problemas sencillos en los que el despliegue no ha sido necesario modelarlo y porque éstos quedan fuera del alcance de esta unidad. Pero sí utilizaremos esta vista a la hora de generar código, como veremos más adelante.



Una vez que ya tenemos más claro cómo se organiza el trabajo con BOUML vamos a pasar a modelar nuestro caso de estudio con BOUML, para lo cuál explicaremos cómo crear los diferentes diagramas necesarios, para luego finalmente hacer una pequeña reseña sobre la generación de código.

Desarrollo orientado a objetos usando uml y herramientas case

Diagramas de casos de uso

Te preguntarás: ¿Cómo empiezo a **modelar** nuestro caso de estudio?

Como ya sabemos el diagrama de casos de uso lo realizamos en la fase de planificación de nuestro proceso. Pero, antes hemos tenido que identificar los actores, los casos de uso y los límites del sistema. Como todo eso ya lo hemos hecho para nuestro caso de estudio, en este apartado nos vamos a dedicar a ver

cómo podemos modelarlo con BOUML.

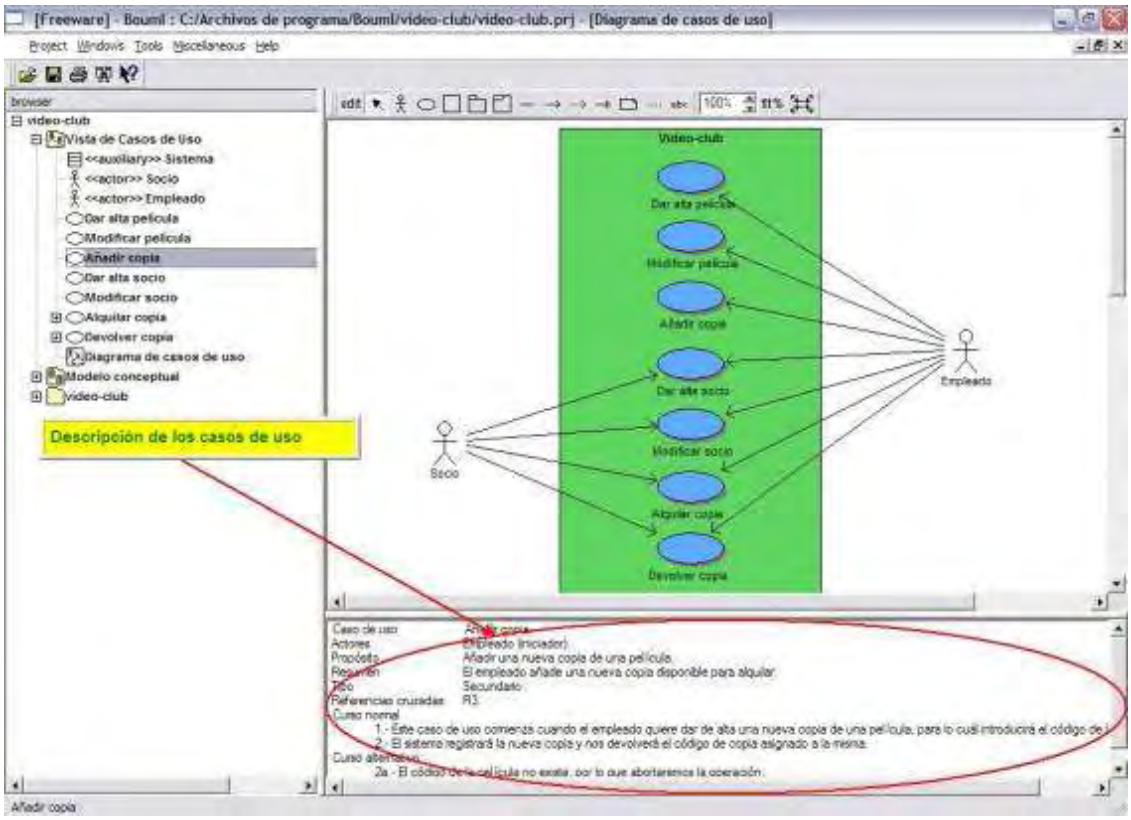
Debido a que los diagramas de casos de uso no van a generar código, no hará falta que metamos la vista de casos de uso dentro de un paquete, así que la colgaremos directamente de nuestro proyecto. Veamos los pasos a seguir para modelar nuestro diagrama de casos de uso.

1. Primero **crearemos una vista de casos de uso** y la renombraremos a "Vista de casos de uso".
2. Una vez hecho esto **crearemos los actores de nuestro sistema**, que en nuestro caso son **Socio** y **Empleado**.
3. Seguidamente debemos **crear los casos de uso**, que para nuestro problema son seis:
   - Dar alta película,
   - Modificar película,
   - Añadir copia,
   - Dar alta socio,
   - Modificar socio,
   - Alquilar película y
   - Devolver película.
4. **Crearemos un diagrama de casos de uso** y lo renombraremos para llamarlo "Diagrama de Casos de uso".
5. **Añadiremos los casos de uso al diagrama** arrastrándolos desde el árbol de navegación hasta el diagrama de casos de uso.
6. **Añadiremos los actores a dicho diagrama**.
7. **Crearemos las asociaciones pertinentes entre actores y casos de uso**.
8. Finalmente **definiremos los límites del sistema**. También podemos **maquillar nuestro diagrama dándole un poco de color** utilizando el menú contextual "edit drawing setting" del diagrama, al cuál accedemos por medio del botón derecho del ratón.

Veamos una simulación de cómo llevar a cabo todas estas tareas para nuestro caso de estudio.

Simulación de creación de un Diagrama de casos de uso

Otra de las tareas que debíamos realizar con respecto a los casos de uso era describirlos, y BOUML nos permite que realicemos la descripción de los mismos utilizando el panel de descripción que como recordaréis era el panel inferior derecho de la ventana de la aplicación, o haciendo doble clic en el caso de uso y rellenando el campo descripción.



Desarrollo orientado a objetos usando uml y herramientas case

Modelo conceptual

¿Has oído hablar de un diagrama en UML que se denomine "**modelo conceptual**"?

Efectivamente, UML no permite representar el modelo conceptual como tal, pero podemos usar un diagrama de clase un poco modificado para llevar a cabo esta tarea.

Para representar nuestro modelo conceptual usaremos por tanto un diagrama de clases colgándolo directamente de nuestro proyecto, ya que del mismo no generaremos código.

La modificación del diagrama de clases que debemos realizar es a la hora de visualizarlo, diciéndole a BOUML que no **queremos ver las operaciones de la clase, sino sólo los atributos**. Veamos los pasos a seguir para construir el mismo:

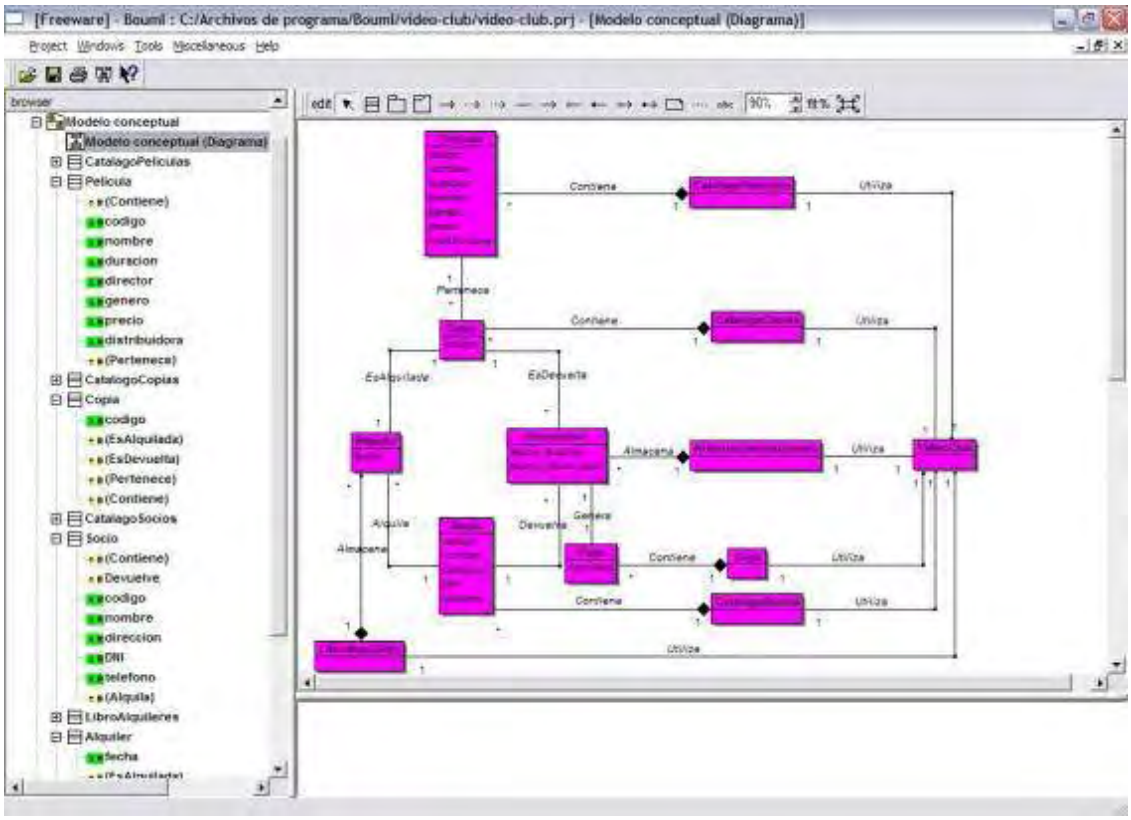
1. Primero **crearemos una vista de clases** y la renombraremos a "Modelo conceptual".
2. Una vez hecho esto **iremos creando clases** (que para nosotros eran conceptos), **colocándonos en la vista recién creada** y con el botón derecho elegiremos "new class" del menú contextual y le iremos dando los nombres adecuados.
3. **Para cada clase le añadiremos los atributos necesarios**, seleccionando la clase y con el botón derecho del ratón elegiremos del menú contextual "**add attribute**" y le daremos el nombre adecuado.
4. Seguidamente **crearemos un diagrama de clases**, por medio del menú contextual de la vista "**new class diagram**", y lo renombraremos a "Modelo conceptual (diagrama)".
5. **Iremos añadiendo las clases al diagrama arrastrándolas al mismo** y las iremos colocando convenientemente.
6. Finalmente **añadiremos las asociaciones entre clases y haciendo doble clic sobre las mismas podremos darle nombre y asignarle su multiplicidad**. También podemos **maquillar nuestro diagrama dándole**

un poco de color utilizando el menú contextual "**edit drawing setting**" del diagrama, al cuál accedemos por medio del botón derecho del ratón. Además desde este menú es desde donde podemos decirle a BOUML que **no represente las operaciones.**

Como antes, os proponemos que veáis esta **simulación** para aclarar el proceso de creación de nuestro modelo conceptual (la simulación no abarca la creación completa de este modelo, sino que sólo ejemplificamos la creación de algunas clases, sus atributos y sus asociaciones, además de darle un poco de color y decirle que no represente las operaciones. Lo que queda es continuar con las clases restantes y sus asociaciones).

Simulación del proceso de creación del modelo conceptual

El aspecto final, después de añadir todas las clases, sus atributos y asociaciones debería ser parecido al que sigue (notar que el árbol de navegación no se ve entero debido a su extensión).



Autoevaluación

1 Respecto a BOUML podemos decir que...

a) Al igual que UML no define un diagrama llamado Modelo conceptual.

b) Aunque no define el modelo conceptual como tal, podemos modelar dicho diagrama por medio de un diagrama de clases de diseño.

c) Permite modificar la forma en que se representa el diagrama de clases, para que no represente las operaciones y así modelar el modelo conceptual.

d) Todas las respuestas anteriores son correctas.

Comprobar

Desarrollo orientado a objetos usando uml y herramientas case

Diagramas de secuencia

Como recordarás el **modelo conceptual** los realizábamos en la **fase de análisis** del ciclo de desarrollo actual, al igual que el modelo conceptual. Otra de las actividades de nuestro proceso era representar un diagrama de secuencia por cada caso de uso que estemos tratando en este ciclo (que para nuestro caso de estudio eran dos casos de uso; los clasificados como primarios).

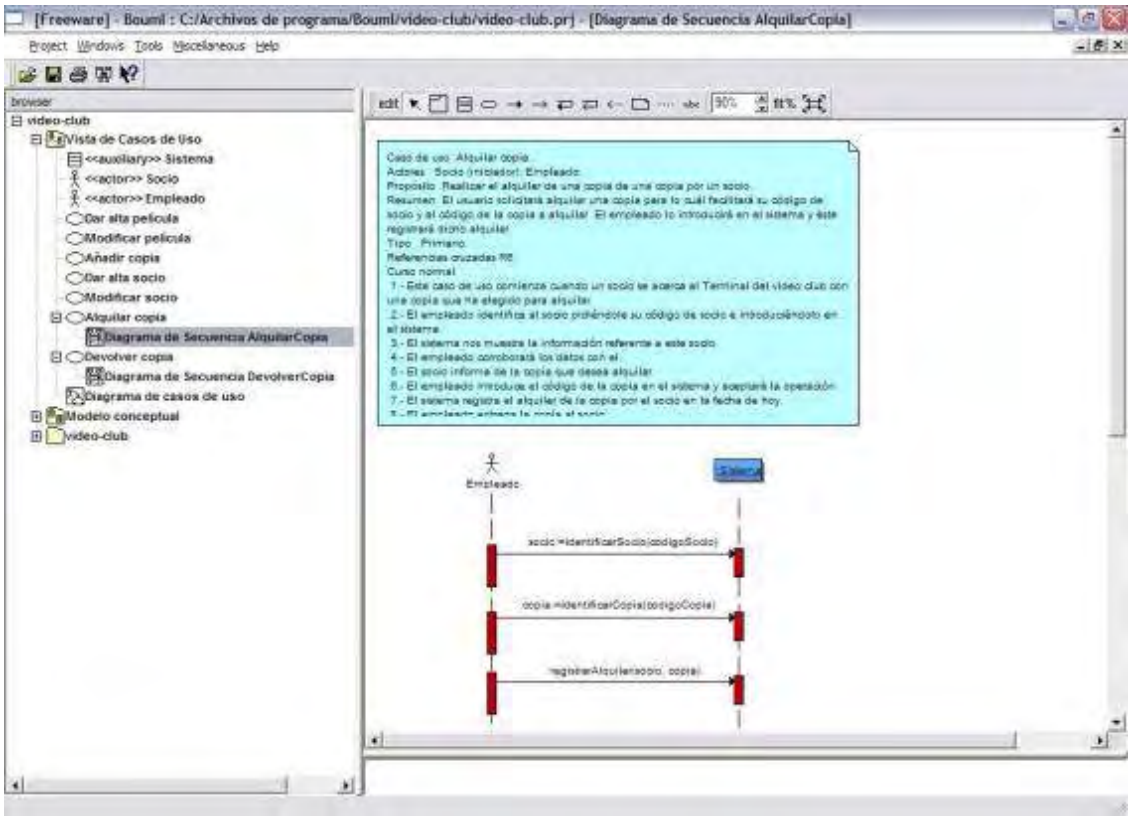
Para representar los **diagramas de secuencia**, usaremos la vista de casos de uso ya que para estos diagramas tampoco generaremos código y además como están asociados a un caso de uso los incluiremos dentro del caso de uso (ya que un caso de uso también actúa como contenedor). Veamos los pasos a seguir para construir un diagrama de secuencia:

1. Lo primero que haremos es **crear un diagrama de secuencia dentro del caso de uso al que éste se asocia**. Para ello seleccionaremos el caso de uso en cuestión y elegiremos "new sequence diagram" del menú contextual que nos aparece al hacer clic con el botón derecho del ratón.
2. **Renombraremos el diagrama** por medio de la opción "edit" del menú contextual de dicho diagrama.
3. Como los diagramas de secuencia utilizan **un actor que denominamos Sistema,deberemos crearlo** como hicimos en el apartado de diagramas de casos de uso. La única diferencia es que este actor tendrá una función distinta a los demás actores, por lo que **deberemos cambiarle su estereotipo a "auxiliary", editando el actor** haciendo doble clic sobre el mismo.
4. Lo siguiente es **arrastrar los actores que intervienen directamente en el caso de uso**, que para nuestro caso de estudio serán Empleado y Sistema.
5. Ahora lo que nos queda es **ir añadiendo los distintos mensajes que se intercambian el actor** (Empleado en nuestro caso de estudio) **y el sistema**. Como en los casos anteriores también **podemos maquillar un poco nuestro diagrama para darle un poco de colorido**.
6. También podemos **insertar una nota en la que podemos incluir el texto del caso de uso**.

Como en los apartados anteriores os incluimos una simulación que detalla el proceso de creación de uno de nuestros diagramas de secuencia. Hemos elegido para la simulación el diagrama de secuencia AlquilarCopia.

Simulación del diagrama de secuencia AlquilarCopia

A continuación os mostramos el aspecto que debería tener el diagrama que acabamos de realizar:



Como recordarás la última actividad a realizar en la fase de análisis era crear los contratos de operación, pero ni UML, ni BOUML en este caso, tienen un mecanismo para describir dichos contratos, por lo que estos los escribiremos en un procesador de textos y posteriormente los podremos añadir como notas a su diagrama de secuencia asociado.

Desarrollo orientado a objetos usando uml y herramientas case

Diagramas de colaboración

¿Te acuerdas de los **diagramas de colaboración** que definimos para nuestro caso de estudio? ¿Te gustaría saber como modelarlos con nuestra herramienta?

Vamos a ver como podemos llevar a cabo esta tarea. Como ya sabes con estos diagramas de colaboración entramos en la fase de diseño del ciclo de desarrollo actual. Ya sabes que la elaboración de estos diagramas es una tarea bastante importante para llegar a un diseño robusto.

Debido a que ya estamos en la **fase de diseño** estos diagramas estarán incluidos dentro de una vista de clases. Además debemos incluir estos diagramas dentro de un paquete si nuestro objetivo es la generación de código (objetivo lógico cuando hablamos de desarrollo orientado a objetos).

Si recuerdas cuando hablábamos de estos diagramas, decíamos que empezaban con un mensaje dirigido hacia una clase, pero resulta que BOUML no permite tener un mensaje que no provenga de ninguna clase, por lo que en estos diagramas incluiremos al empleado como generador de todos los mensajes. Veamos los pasos a seguir para la elaboración de dichos diagramas.

1. Lo primero que haremos es **crear un paquete llamado video-club**.
2. **Crearemos una vista de clases** dentro de dicho paquete a la que llamaremos "Vista de diseño".
3. **Añadiremos las clases que intervengan** en dicho diagrama si es que aún no las hemos creado. En este caso aún no tenemos que añadir ni atributos ni operaciones a dichas clases. Si tenemos que representar un multiobjeto (colección), añadiremos la clase que contiene dos o tres veces y las pondremos una encima de otra.
4. **Crearemos enlaces entre las clases que se vayan a enviar mensajes y les añadiremos los**

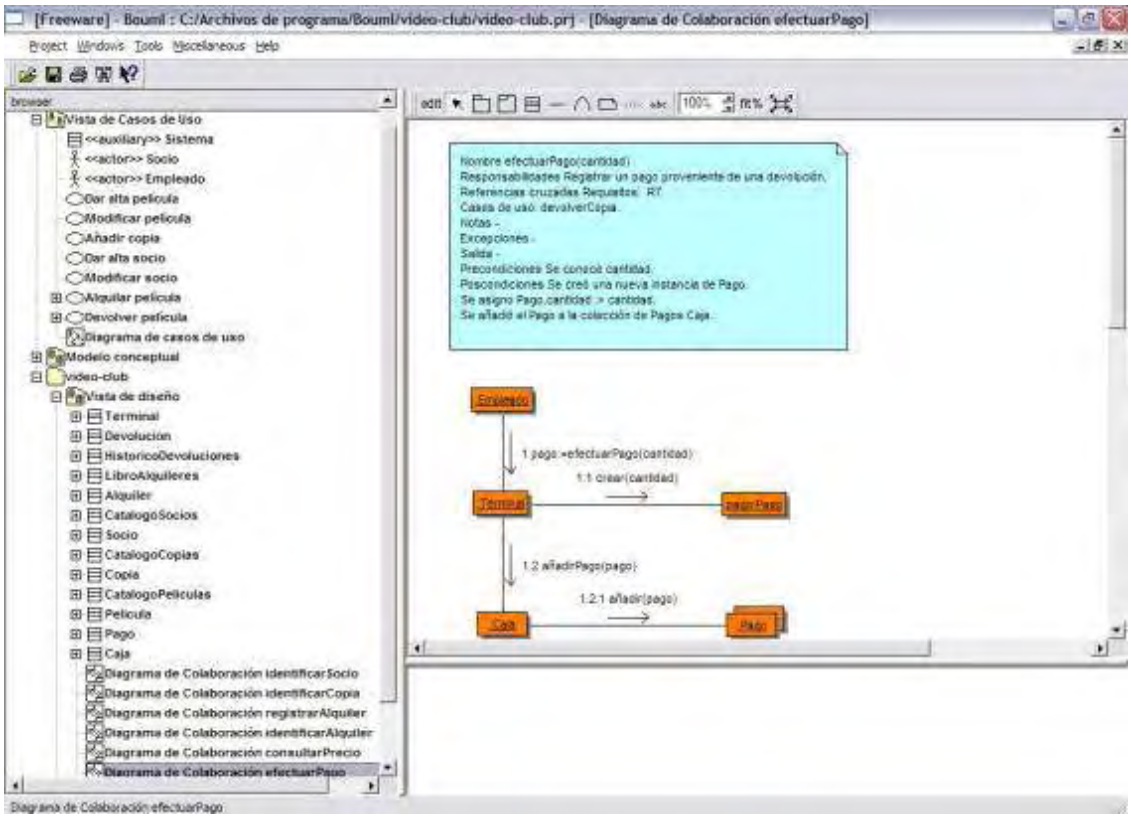
mensajes que éstas se envíen.

5. Finalmente **le daremos un poquito de color** a nuestro diagrama y **cambiaremos la forma en la que se numeran los mensajes para hacerlo de forma jerárquica que es como propone UML**. Esto último lo haremos eligiendo "**yes" para la opción "show hierarchical rank**" de la opción del menú "**edit drawing setting**". Esta opción la podemos cambiar a nivel de diagrama (y deberemos hacerlo para todos los diagramas) o podremos hacerlo para la vista en la pestaña "**collaboration**".

Como en los apartados anteriores os incluimos una simulación que detalla el proceso de creación de uno de nuestros diagramas de colaboración. Hemos elegido para la simulación el diagrama de colaboración efectuarPago.

Simulación del diagrama de colaboración efectuarPago

Y aquí os mostramos el aspecto que debería tener dicho diagrama.



Autoevaluación

1 Un diagrama de colaboración con BOUML...

a) Debe estar incluido en un paquete obligatoriamente.
b) Debemos incluir en él un paquete para generar código.
c) Lo incluimos dentro de un paquete para posteriormente generar código partiendo de estos diagramas y del diagrama de clases de diseño.
d) Todas son verdaderas.

Comprobar

Diagrama de clases de diseño

Te preguntarás ¿cómo realizo el **diagrama de clases de diseño** ya que tengo casi todo el trabajo hecho?

Pues muy fácil, ya que es muy similar a la realización del modelo conceptual, que como recordarás lo modelamos mediante un diagrama de clases aunque un poco modificado.

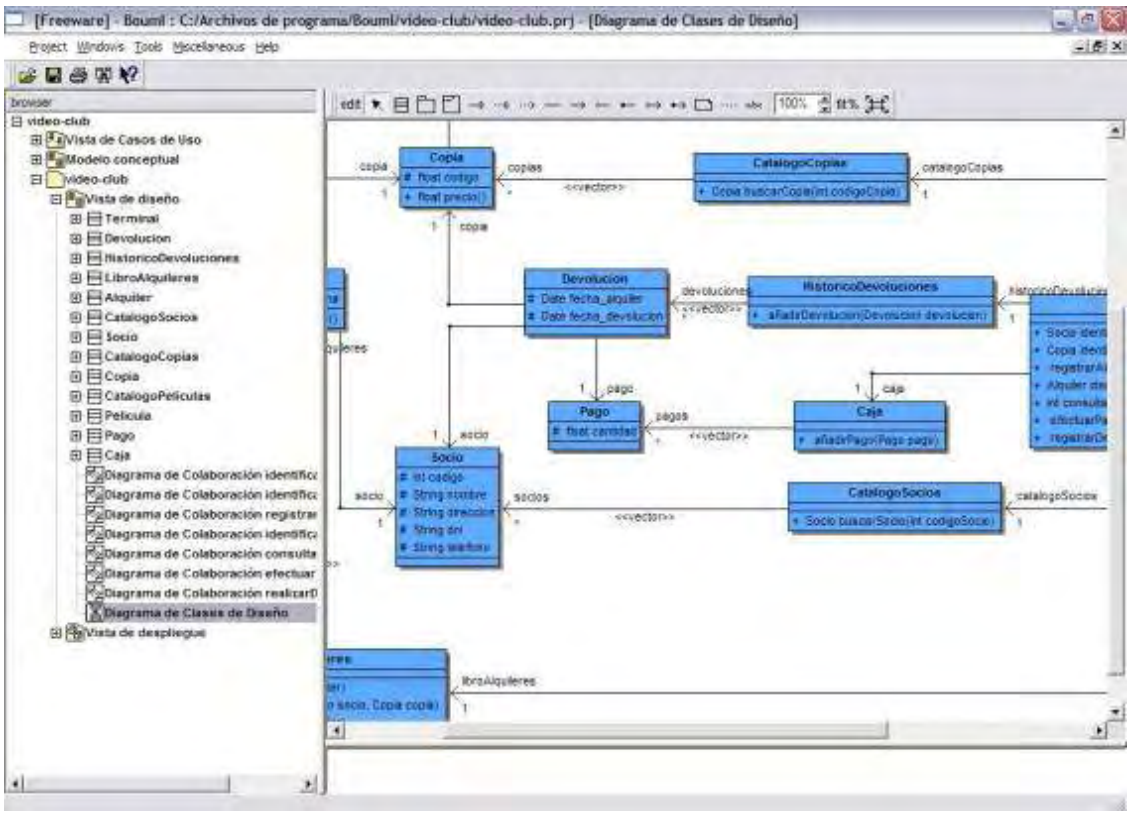
En el diagrama de clases sí queremos mostrar las **operaciones**, además de los atributos. También nos interesa mostrar la visibilidad de los atributos y de las operaciones, así como la definición de los mismos utilizando para ello el lenguaje Java. Veamos los pasos a seguir para representar este diagrama mediante BOUML:

1. Primero **añadiremos un diagrama de clases de diseño a nuestra vista de diseño** creada anteriormente. Lo renombraremos para llamarlo "Diagrama de clases de diseño".
2. **Añadiremos los atributos necesarios a cada clase**, además de indicar los tipos de cada uno de ellos.
3. **Añadiremos las operaciones necesarias a cada clase**, indicando el tipo de sus parámetros y el valor devuelto por las mismas si fuese necesario.
4. **Crearemos las asociaciones entre las distintas clases** (estas pueden ser unidireccionales o bidireccionales). En este diagrama nos interesa más que darle un nombre a la asociación, darle nombre a los roles y **definir su multiplicidad** (los roles serán el nombre que tendrán los atributos en la clase debido a la visibilidad de atributo generada por las asociaciones).
5. Finalmente le **daremos un poco de color,** le diremos que **dibuje las clases para el lenguaje Java**, que muestre la visibilidad de los atributos y operaciones y que muestre la definición completa de los mismos.

Como en los apartados anteriores os incluimos una simulación que detalla el proceso de creación del diagrama de clases de diseño para nuestro caso de estudio.


Creación del diagrama de clases de diseño

Y aquí os mostramos el aspecto que debería tener dicho diagrama.



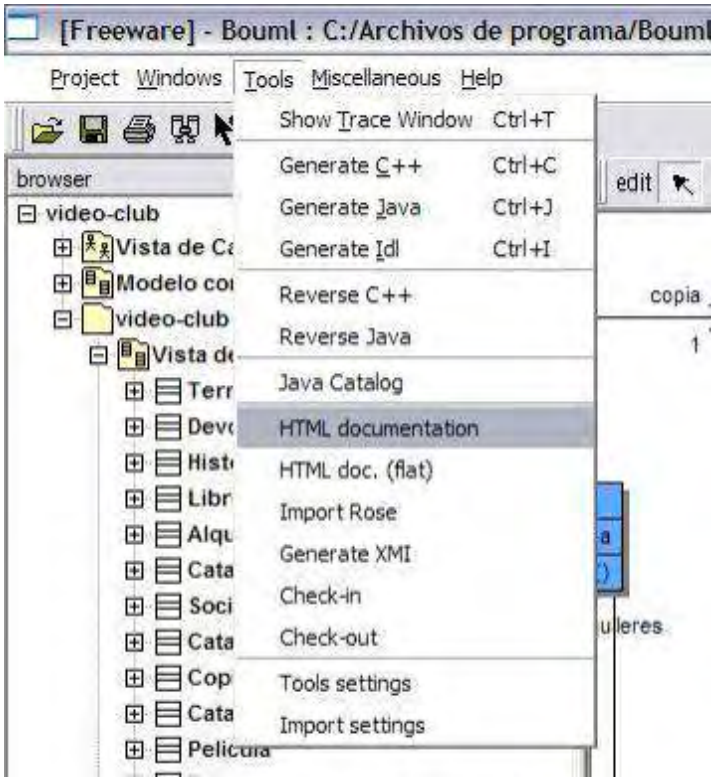
Desarrollo orientado a objetos usando uml y herramientas case

Generación de documentación y código

¿Recuerdas cuáles eran los objetivos de utilizar una herramienta CASE? Efectivamente, entre otros eran la **generación de documentación y la generación de código**.

La generación de documentación es una tarea bastante importante en nuestro proceso de desarrollo orientado a objetos, ya que nos permitirá compartir el conocimiento adquirido sobre nuestro problema y las soluciones adoptadas con todo nuestro equipo de desarrollo.

Con BOUML, la generación de documentación es trivial, ya que **simplemente deberemos elegir en el menú "Tools" la opción "HTML documentation"** y BOUML generará para nosotros toda la documentación de nuestro proyecto en formato HTML en el directorio que nosotros elijamos.



A continuación os mostramos la documentación generada para nuestro caso de estudio. Está en un archivo comprimido ya que consta de varios archivos. Para visualizarla deberás descomprimirla y en tu navegador preferido abrir el archivo "index.html" o el archivo "index-withframe.html" dependiendo si quieres verla sin marcos o con ellos (te recomendamos que lo veas sin marcos, ya que la documentación está más clara).

Documentación generada

Como hemos dicho antes, BOUML también nos permite **generar código para los lenguajes Java,C++ e IDL**. Nosotros nos centraremos en la generación de código para el lenguaje Java.

¿Te acuerdas que antes de definir los diagramas de colaboración creamos un paquete?

Pues ese paquete que generamos no tenía otra intención que la posterior **generación de código**. También recordarás que a la hora de hacer el diagrama de clases no representábamos los constructores o las operaciones de acceso y modificación de atributos,

ya que no aportaban información al diagrama y lo único que hacían era enredarlo. Antes de la generación podemos añadir estas operaciones si nos van a hacer falta.

Debemos decir que BOUML no hace magia, por lo que el código generado no será un código que funcione directamente sin más. **El código generado serán los esqueletos de las clases y de las operaciones o métodos de las mismas**. Para que nuestro código sea funcional **deberemos implementar los cuerpos de dichas operaciones**, pero esta tarea no nos debería suponer un gran esfuerzo dado que el proceso que hemos seguido nos ha llevado a tener un conocimiento exhaustivo de nuestro problema y deberemos saber qué se debe hacer en cada operación, aunque no la hallamos implementado.

Veamos los pasos que debemos seguir para llegar a nuestro código generado:

1. **Editaremos las propiedades de generación a nivel de proyecto para decirle el directorio** donde queremos que se genere nuestro código.
2. Debido a que el lenguaje de generación será Java, **debemos decir el nombre del paquete Java y el directorio del mismo.**
3. Lo siguiente es **crearuna nueva vista de despliegue para nuestro proyecto**. La renombraremos para darle un nombre más significativo.
4. Ahora **debemos asociar la clase de despliegue recién creada a nuestra vista de diseño**.
5. **Para cada clase crearemos un artefacto fuente**.
6. Por último le **diremos a BOUML que genere nuestro código Java**.

Como en los apartados anteriores os incluimos una simulación que detalla el proceso de generación de código para nuestro caso de estudio.

Simulación que detalla el proceso de generación de código

Y aquí podéis ver el código generado para nuestro caso de estudio. Es un fichero comprimido en el que se encuentran todos los ficheros .java que representan a nuestras clases.

Código generado

Y con esto y un bizcocho damos por finalizado nuestro proceso de desarrollo orientado a objetos para nuestro caso de estudio. Esperamos que te haya quedado claro y que seas capaz de enfrentarte a cualquier otro problema real y llegar a un diseño robusto, como al que hemos llegado con nuestro caso de estudio.

Autoevaluación

1 BOUML permite...

a) Generar código para lenguajes como Java.
b) Generar documentación partiendo del modelado de nuestro problema.
c) Hacer ingeniería inversa partiendo de un código en lenguajes como Java.
d) Todas las respuestas anteriores son correctas.

Comprobar