CURSO DE DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Entornos de desarrollo

Quedan rigurosamente prohibidas, sin la autorización escrita de los titulares de «Copyright», bajo las sanciones establecidas en las leyes, la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la reprografía y el tratamiento informático, y la distribución de ejemplares de ella mediante alquiler o préstamo públicos. Diríjase a CEDRO (Centro Español de Derechos Reprográficos, http://www.cedro.org) si necesita fotocopiar o escanear algún fragmento de esta obra.

INICIATIVA Y COORDINACIÓN
IFP Innovación en Formación Profesional

Supervisión editorial y metodológica:
Departamento de Producto de Planeta Formación

Supervisión técnica y pedagógica:
Departamento de Enseñanza de **IFP** Innovación en Formación Profesional

Módulo: Entornos de desarrollo / Desarrollo de aplicaciones web

© Planeta DeAgostini Formación, S.L.U.
Barcelona (España), 2017

INTRODUCCIÓN AL MÓDULO

Un **entorno de desarrollo** es el conjunto de procedimientos y herramientas utilizadas para la creación o mantenimiento de programas informáticos (software). Si bien un entorno de desarrollo debe abarcar la entrada de código de programa a través de un editor de código, las pruebas a través de un compilador y un depurador y su empaquetado final para la entrega al usuario utilizando un constructor de interfaz gráfica (GUI), hay otros aspectos que debe tener en cuenta para ser un nexo útil entre la fase de análisis y la fase de instalación que un desarrollo de software debe contemplar. En ocasiones, el concepto puede también aplicarse a un entorno y elementos físicos que permiten alcanzar los objetivos planteados por un requerimiento de cliente.

Se etiqueta como IDE (*Integrated Development Environment* o Entorno de Desarrollo Integrado) al entorno de programación que contiene un conjunto de procesos, instrumentos y normas que trabajan de forma coordinada para facilitar al programador el control de las diferentes etapas del desarrollo en las que está directamente implicado: entrada de código de programa y pruebas y preparación (empaquetado) del producto final para su inmediata utilización. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien pueden utilizarse para varios.

En algunos lenguajes de programación, un IDE puede funcionar como un sistema en tiempo de ejecución donde se permite utilizar el lenguaje de programación de forma interactiva sin necesidad de trabajo orientado a archivos de texto.

UNIDAD FORMATIVA 3

- Elaboración de diagramas de clases
- Elaboración de diagramas de comportamiento

5. Elaboración de diagramas de clases

En esta unidad vamos a desarrollar conceptos de uso frecuente en la **programación orientada a objetos**, que es una forma de simular el funcionamiento de la realidad mediante modelos que simplifican los elementos reales a los que pretenden imitar. En ocasiones, no se trata de una simplificación si no de una complejidad mayor o de una distorsión de dicha realidad; en ese caso, estaríamos hablando de simulaciones ficticias. Esta forma de abordar nuestro entorno tendrá, además, características de reusabilidad, es decir, no hay que inventar lo que ya está inventado sino utilizarlo en nuestro proyecto, o modularidad, lo que permitirá entrar código en un objeto de forma independiente al resto de objetos.

Nos centraremos, pues, en una serie de modelos con características propias, que podrán ser cambiados por otros o bien comunicarse con el resto de la comunidad de modelos existentes. En definitiva, se trata de una nueva forma de entender el desarrollo informático, que rompe por completo con la forma tradicional de entender el código fuente como un simple listado de instrucciones que va ejecutándose en un orden determinado.

5.1 Clases, atributos, métodos y visibilidad

La **programación orientada a objetos** es un método de programación basado en las clases y su jerarquía, así como en las relaciones existentes entre los objetos del proyecto. Veamos a continuación cuáles son sus conceptos básicos:

- **Clase.** Una definición muy mecanicista de la realidad podría hacerse mediante grupos de objetos con características y conductas diferentes (perros, gatos, semáforos, coches); sin embargo, existen múltiples objetos que poseen características similares. Estas características comunes suelen agruparse para formar un tipo de objeto (clase). Como abstracción de la realidad a la que pretende representar, la clase sería como una sección del plano de un arquitecto, una estructura o prototipo en el que están definidos los datos y métodos que nos permitirán trabajar con dichos datos (Figura 5.1). Este concepto es muy importante, y, a la hora de programar, el dato está envuelto en una clase. Una clase puede entrarse por un programador o bien puede existir ya en las librerías, que el propio lenguaje posee y que habremos instalado previamente en nuestro ordenador. Así pues, para trabajar con las clases deberemos tener en cuenta que definen a un grupo de objetos, y que comparten una estructura y una conducta.

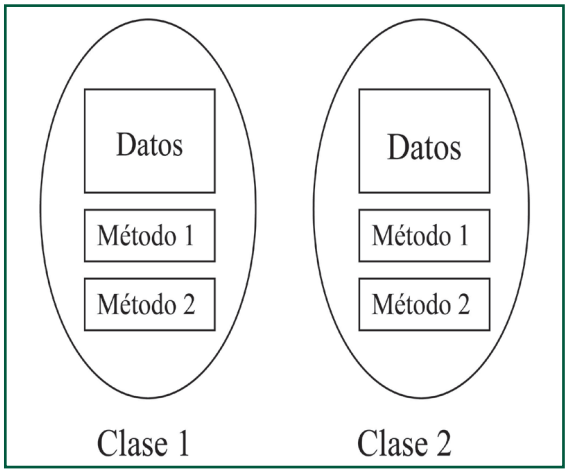


Figura 5.1
Las clases contienen los datos y métodos que nos permiten trabajar.

- **Atributo.** Se trata de una característica aplicada a un elemento de la clase.
- **Método.** Como elemento de la clase, el método contiene un procedimiento que fijará la conducta manifestada por los objetos pertenecientes a dicha clase.
- **Visibilidad.** La visibilidad de una propiedad o un método puede mencionarse como la posibilidad que tengan de ser accedidos. Palabras clave, como *Public* o *Private*, de los códigos definen si puede acceder cualquiera o no (Figura 5.2).

```
class ⟨ClassVariable⟩
    attr
      ⟨NombreAtributo1⟩
        :
      ⟨NombreAtributoN⟩
    meth ⟨Modelo1⟩ ⟨Declaración⟩ end
            :
    meth ⟨ModeloN⟩ ⟨Declaración⟩ end
end
```

Figura 5.2
Podemos declarar no visible un método de una clase.

5.2 Objetos. Instanciación

Los **objetos** de la realidad como un caballo o un pájaro son ejemplos de objetos. Ambos poseen una serie de características o estados: nombre, color, peso, etcétera, y ambos tienen también diferentes conductas: uno galopa, el otro vuela; uno relincha y otro pía. Siempre podemos encontrar objetos de la realidad que poseen

estados o conductas compartidas por otros objetos. Una bombilla, por ejemplo, sólo tiene dos estados: encendido y apagado, y, por lo tanto, dos posibles conductas: encenderse y apagarse. Los objetos de la realidad pueden llegar a poseer una gran complejidad, sin embargo todos se reducen a una lista de posibles estados y a una lista de posibles conductas que pueden hacer.

Los objetos tratados en la programación orientada a objetos presentan las mismas particularidades de estado y conducta que hemos mencionado para los ya existentes en la realidad (Figura 5.3):

- **Objeto de software.** Un objeto de software almacena su estado en variable, y manifiesta su conducta a través de los métodos que operan con los estados de los objetos y canalizan la comunicación entre otros objetos.

- **Instanciación.** Es la creación de una instancia real, esto es, de un objeto a partir de una clase. Se instancia la clase con la creación de un objeto concreto de dicha clase. Este objeto, a diferencia de la clase, es ejecutable por el ordenador y le hemos dado un nombre y colocado en algún lugar concreto de nuestro proyecto.

Figura 5.3
Los objetos de la realidad tienen posibles estados y posibles conductas.

5.3 Relaciones. Herencia, composición, agregación

Las clases y sus instancias, al igual que los objetos de la realidad, se relacionan entre sí. Dichas relaciones deben indicar claramente la naturaleza de cada entidad que conforma nuestro proyecto. Destacamos a continuación algunas de ellas:

- **Herencia.** La herencia define las relaciones entre las clases en un lenguaje orientado a objetos: podemos tener unas clases base (o también llamadas *superclases*), de las que pueden derivar otro tipo de clases (subclases). Los métodos y las propiedades definidas en las superclases son compartidas por las subclases. Por ejemplo, podemos definir la superclase "Animal", de manera que todos los seres vivos pertenecientes al reino animal puedan ser clasificados en esa superclase. A continuación, podemos crear subclases (perro, gato, etc.) que compartan las características de la superclase, pero que contengan rasgos específicos propios de cada subclase.

- **Composición.** Es una forma de asociación en la que un objeto forma parte integral de otro. Así, en la vida real, un objeto está compuesto de muchas partes. En el caso del software ocurre lo mismo, es decir, el objeto formado tiene validez mientras los objetos componentes también la tengan; el objeto contenido desaparecerá si el objeto que los contiene desaparece. Por ejemplo, los departamentos de una universidad mantienen una relación de "composición" con dicha universidad, y que, en caso de su desaparición, sus departamentos también desaparecerían, pues no tienen "vida propia" fuera de la universidad.

- **Agregación.** En la agregación también tenemos objetos contenidos y objetos que lo contienen, pero la relación es parametrizada y su existencia, o inexistencia, no les afecta. Si el objeto contenedor desaparece, los contenidos siguen existiendo (Figura 5.4). Por ejemplo, los profesores que forman parte de la universidad tienen una relación de agregación con ella, y aunque ésta desapareciese, ellos no desaparecerían, ya que podrían incorporarse a otra universidad.

Recuerda

Cuando hablamos de agregación, al eliminar el objeto contenedor, los componentes siguen existiendo. Cuando hablamos de composición, al eliminar el objeto contenedor, desaparecerán también los componentes.

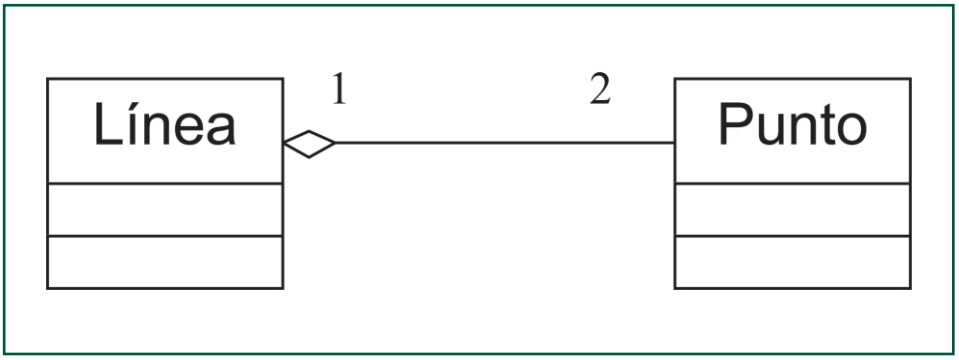


Figura 5.4
Agregación: la clase *Línea* está compuesta por dos puntos. En la agregación, se utiliza un rombo transparente a diferencia de la composición, que utiliza un rombo relleno.

5.4 Diagramas UML. Diagramas estructurales

El principal objetivo de **UML** (*Unified Modeling Language*) es presentar de una forma gráfica (símbolos y conectores) diagramas de procesos, que son muy utilizados en el modelado de programas informáticos y diagramas de flujo. Según sea la fase

en la que nos encontremos del ciclo de desarrollo de un proyecto de software, la probabilidad de generar unos diagramas u otros variará, si bien con las siguientes excepciones:

- **Recogida de requerimientos.** Se trata de una fase inicial que necesita que conozcamos el sistema en el que nos vamos a mover:
  - **Diagramas de secuencia.** Muestran la interacción entre las clases entendiendo por ello intercambio de mensajes.
  - **Diagramas de casos de uso.** Son modelos de funcionalidad del sistema con relación a la interacción con el usuario.
  - **Diagramas de colaboración.** La interacción entre objetos se representa como una serie de mensajes (Figura 5.5).

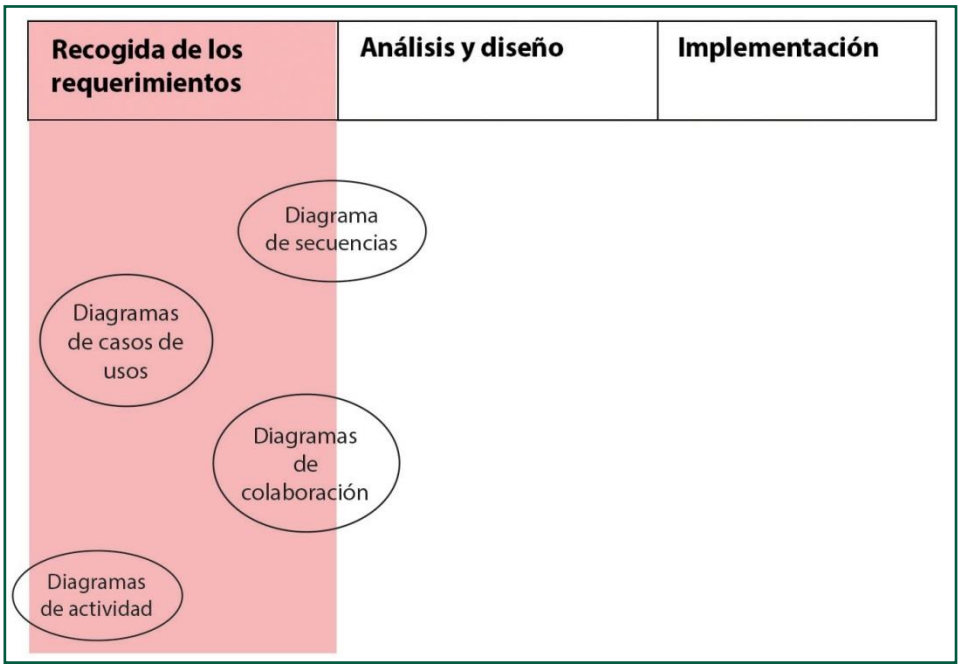


Figura 5.5
Ubicación de los diagramas en la fase de desarrollo de recogida de requerimientos. Algunos diagramas pueden hallarse en más de una etapa.

  - **Diagramas de actividad.** Flujo de datos de un sistema desde una acción hasta su respuesta.

- **Análisis y diseño.** Es la fase en la que trabajan más los programadores (Figura 5.6):
  - **Diagramas de estados.** Describen la conducta dinámica de un sistema como respuesta a estímulos externos.
  - **Diagramas de distribución.** Contemplan los diferentes recursos físicos que posee el sistema en momentos concretos, pues dichos recursos pueden ser dinámicos.

- **Diagramas de clases.** Definen la estructura de un sistema. Describen las clases como conjuntos de objetos que comparten atributos, operaciones, métodos, etc.
- **Diagramas de componentes.** Describen la organización de los componentes de software.
- **Diagramas de actividad.** Son modelos del flujo de un sistema desde una acción hasta una respuesta.

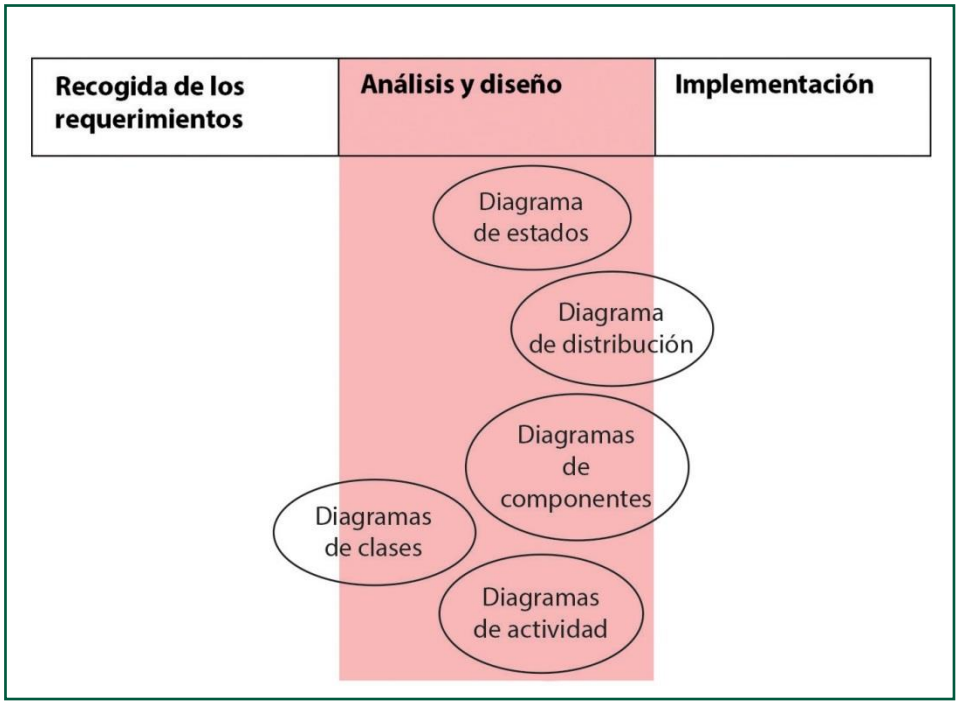


Figura 5.6
Ubicación de los diagramas en la fase de desarrollo de análisis y diseño, algunos diagramas pueden hallarse en más de una etapa.

Los diagramas pueden agruparse en:

- **Diagramas de comportamiento.** Aquellos que nos muestran la forma que tienen de reaccionar ante los estímulos externos. Son diagramas dinámicos, basados en su devenir en tiempo de ejecución. Por ejemplo, mediante un diagrama podremos comprobar los diferentes estados de una máquina inyectora de plásticos en una línea de producción durante las veinticuatro horas de un día laborable; es decir, podremos observar su evolución durante la jornada de trabajo para conocer su comportamiento y su relación con otros elementos externos.

- **Diagramas estructurales.** Presentan modelos estáticos del objeto, como clases, paquetes o componentes, aunque también pueden ser utilizados como modelos en tiempo de ejecución. Por ejemplo, la máquina del ejemplo anterior es la misma durante el cambio de turno que cuando emite la alarma avisando de que

se ha terminado la materia prima y no puede seguir fabricando. Sea cual sea el estado en que se encuentre dicha máquina, las características base de la máquina forman parte de su estructura (tamaño, color).

5.5 Notación de los diagramas de clases

El objetivo principal de un diagrama de clases es describir las clases en un modelo. Se trata de un diagrama estructural, por lo que define la estructura del sistema a través de la presentación de las clases. En una aplicación orientada a objeto veremos que las clases tienen atributos, operaciones y relaciones con otras clases. Mediante el diagrama de clases podremos describir todos esos elementos de una forma clara. La principal figura que representa una clase es el rectángulo (Figura 5.7).

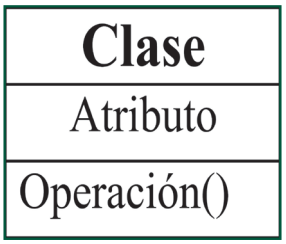


Figura 5.7
Representación de una clase.

Dividimos el rectángulo en tres secciones.: la parte superior se reserva para el nombre de la clase; la parte central contiene la lista de los atributos, y la parte inferior se reserva a la lista de operaciones o métodos. Sin embargo no deben mostrarse todos los atributos y operaciones que la clase contenga, sino sólo los que tengan sentido para el objetivo del diagrama que estamos realizando. Por ejemplo:

Clase. Perro: tiene unos atributos comunes (4 patas, 2 ojos, olfato desarrollado, oído fino) y unas operaciones (conductas) concretas (ladrar, comer, dormir…).

Los **atributos** suelen ser sustantivos (estatura, edad); mientras que las operaciones suelen ser verbos, ya que son acciones que realiza o recibe el objeto. Tanto los atributos como las operaciones tienen un grado de comunicación y visibilidad con el entorno, que se define mediante los siguientes tipos:

- ***Public.*** Son visibles desde fuera y desde dentro de la clase.
- ***Private.*** Los atributos y métodos sólo son accesibles por métodos de la propia clase.
- ***Protected.*** No podrán ser accesibles por métodos fuera de la clase, pero sí por subclases que hayan derivado de la superclase principal a través de la herencia.

Veamos un ejemplo:

- Público. Los atributos y operaciones de un perro son públicos en tanto que éstos sean visibles por otras clases (hombres, mujeres, gatos); el ladrido es una acción pública.
- Privado. Los perros emiten determinados olores que sólo los detectan otros perros.
- *Protected*. Un ser humano puede observar una acción determinada del perro (una emoción), pero no por pertenecer a su clase, sino por derivar de una superclase común (animal emocional); sin embargo, otros animales derivados de superclases no emocionales no podrán apreciar dicha conducta.

Las relaciones entre clases pueden ser de varios tipos:

- De cero a varios (0 ... n) siendo n un número indeterminado para cada caso (Figura 5.8).
- De uno a varios (1 ... n).
- Número fijo (n).
- Herencia.

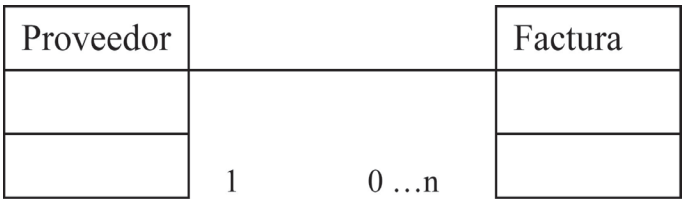


Figura 5.8
Relación de clases 0 a varios. Un proveedor puede no habernos facturado o habernos facturado muchas veces.

En caso de una herencia, hablaremos de **superclase** para definir la primera clase sin derivar, y hablaremos de **subclases** para referirnos a aquellas derivadas de la superclase. Las subclases, además de atributos y métodos (*public* y *protected*) de la superclase, pueden tener sus propios atributos y métodos. Por ejemplo:

- Superclase: seres humanos.
- Subclase: niños, ancianos y toda la clasificación que queramos utilizar que comparta la característica de la superclase.

Relación de clases 0 a varios. Un proveedor puede no habernos facturado o habernos facturado muchas veces.

5.6 Instalación de Visual Paradigm UML integrado en NetBeans

Veamos cómo sería la instalación de un módulo que nos permitirá utilizar los diagramas de clases. Para ello instalamos el módulo Visual Paradigm UML integrado en el entorno de desarrollo de NetBeans. Seguiremos estos pasos:

- Vamos a la URL www.visual-paradigm.com/download.
- Descargamos el archivo de instalación.
- Ejecutamos el programa de instalación.
- Ejecutamos Visual Paradigm en el menú Programas de Windows.
- Se presenta una pantalla para la entrada de la clave de licencia. Hacemos clic en el botón 'Try without key'.
- En el menú "Herramientas", seleccionamos la opción "Integración IDE".
- Escogemos NetBeans en integración IDE.
- Al finalizar la integración, que puede tardar unos minutos, abrimos NetBeans.
- Abrimos un proyecto de Java en NetBeans.
- Finalmente comprobamos que en el menú "Herramientas" (*Tools*) existen nuevas opciones VP-UML EE.

5.7 Herramientas de diseño de diagramas

Existen en el mercado numerosas **herramientas de asistencia gráfica para diagramas**, por lo que es imposible mencionarlas todas. Veamos, sin embargo, cuáles son las principales:

- **Microsoft Visio.** Programa comercial considerado de propósito general, aunque posee capacidad para los diagramas UML.
- **JDeveloper.** *Plug-in* no comercial para el entorno de desarrollo NetBeans. Diseño de casos de uso, clases, actividad y diagramas de secuencia.

- **Visual Paradigm.** SDE for NetBeans. *Plug-in* comercial. Facilidad de migración de diseños ya realizados. Capacidad de trabajo en grupo (Figura 5.9).

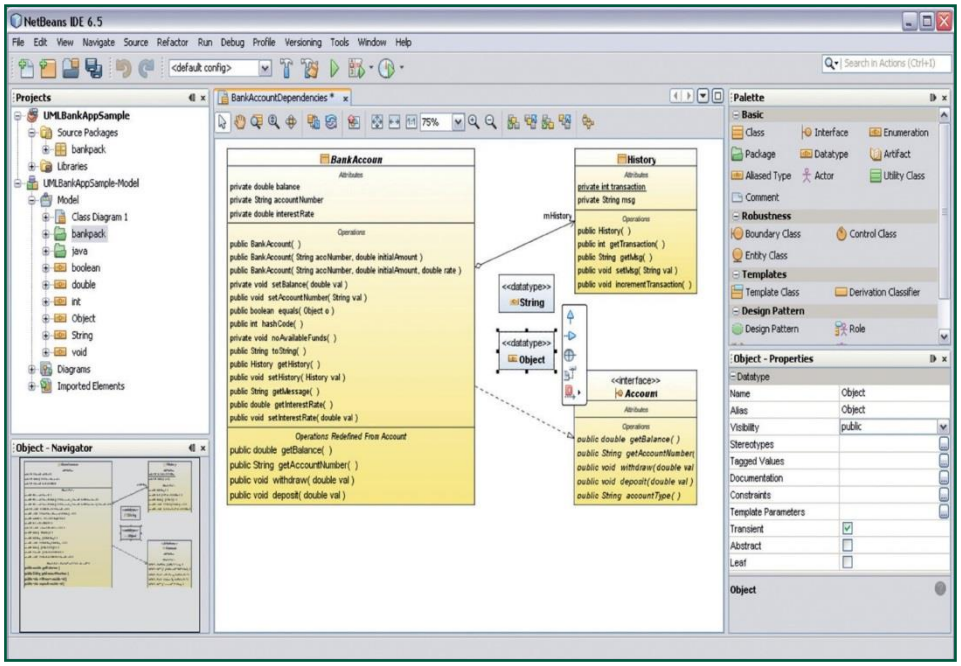


Figura 5.9
Plug-in de diseño de diagramas UML dentro del entorno de desarrollo NetBeans.

- **Bouml.** No comercial. Compatible con C++, Java, PHP, entre otros. Sigue UML 2.0 Standard.

- **Enterprise Architect.** Comercial. Análisis UML y herramienta de diseño. Trazabilidad desde los requerimientos hasta la publicación. Permite el trabajo en equipo.

- **MDT-UML2Tools.** Integración en entorno de desarrollo Eclipse. Permite diagramas de estructura, comportamiento e interacción.

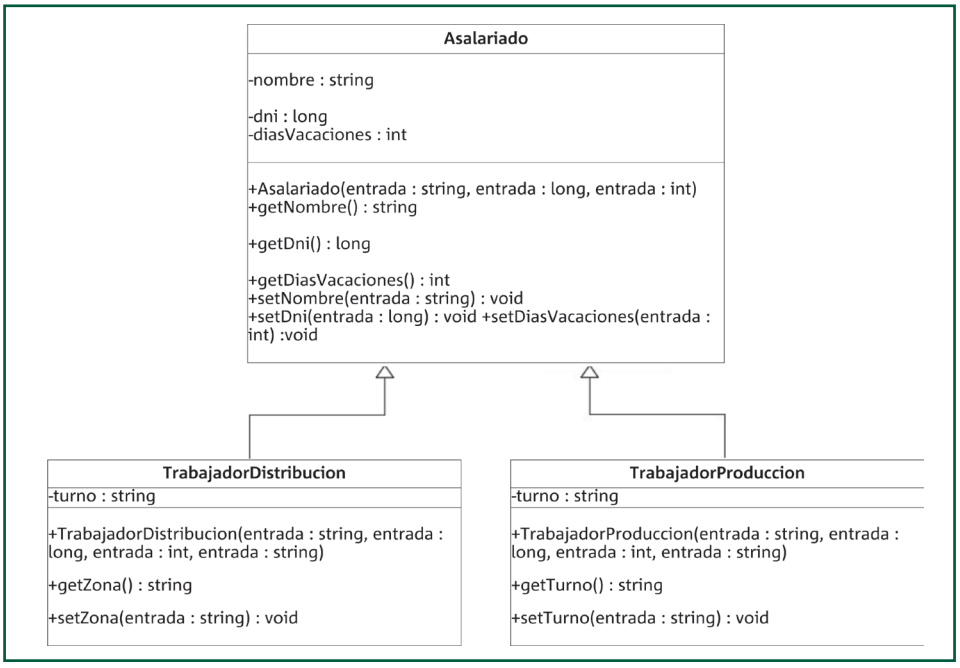
- **SharpDevelop.** Entorno de desarrollo no comercial para los lenguajes de programación C#, Visual Basic .NET, F#, Python, Ruby, Boo y C++.

5.8 Diagrama de clases a partir de sus especificaciones

Veamos un ejemplo de una relación de herencia. Imaginemos el siguiente escenario: queremos representar a los trabajadores de una empresa mediante clases. Dentro de estas hay varias categorías en relación con el cargo que desempeñan.

Definimos una clase “Asalariado”, que nos permita representar a todos los trabajadores de la firma. A partir de esa clase, definimos dos subclases “TrabajadorDistribucion” y “TrabajadorProduccion”, que representan a los trabajadores de dicha firma, que trabajan en las divisiones de distribución y producción respectivamente. Los trabajadores de distribución tienen asignada una zona (o región) donde deben distribuir los productos. Los de producción trabajan por turnos (“Mañana”, “Tarde”, “Noche”), pero en cambio no trabajan por zonas.

El diagrama de clases en UML que nos permite representar la anterior situación podría ser el siguiente (vamos a interesarnos por conocer los siguientes datos: nombre, DNI y número de días de vacaciones que disfrutará cada trabajador):



Representaremos el diagrama mediante tres bloques que van a contener toda la información que necesitamos: en un primer bloque, situaremos la información referente a la clase “Asalariado”; en el segundo, tendremos la información de la clase “TrabajadorDistribución”, y, en un tercer bloque, referenciaremos la información de la clase “TabajadorProduccion”.

Podemos verificar que las clases están vinculadas por una relación de herencia. Por lo tanto, un objeto de la clase “TrabajadorDistribucion” “es - un” “Asalariado”, del mismo modo que un objeto de la clase “TrabajadorProduccion” “es - un”“Asalariado”.

Para saber más

Para acceder a una visión general y resumida del Lenguaje Unificado de Modelado UML, consulta el siguiente enlace de la Universidad Politécnica de Valencia: http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

En el diagrama podemos ver cómo las relaciones de herencia en UML se representan por medio de una línea terminada en una flecha cuyo centro es blanco. Además, todos los atributos que aparecen en la clase base ("Asalariado", en nuestro caso) no deben aparecer en las clases derivadas. De igual manera, los métodos que aparezcan en la clase base y cuyo comportamiento no varíe en las clases derivadas tampoco aparecerán en las mismas. En las clases derivadas solo debemos incluir los atributos y métodos que no aparecían en la clase base.

Siguiendo el gráfico anterior, disponemos de tres clases: "Asalariado", "TrabajadorDistribucion" y "TrabajadorProduccion" entre las cuales hemos definido una relación de herencia.

5.9 Generación de código a partir de diagramas de clases

Los **diagramas de clases** son los diagramas UML más conocidos. Los aspectos estructurales del objeto que estamos modelando (atributos, operaciones y relaciones entre objetos) nos dan una idea bastante aproximada de cómo estamos configurando nuestro nuevo 'modelo de sistema'.

Existen varias herramientas en el mercado que generan código a partir de diagramas de clases, algunas incluso incorporadas, mediante *plug-in*, a entornos de desarrollo conocidos. La base estructural que tienen los diagramas de cla- ses no permiten que las herramientas que generan código a partir de ellos puedan ir más allá del diseño general de una base de datos, la configuración de ficheros, campos y demás elementos de estructura. Para poder completar el código, pues, sería necesario incluir aspectos de comportamiento (operaciones), de mayor complejidad. Por ello, muchos programadores se quedan en la codificación automática del área relacionada con la estructura a partir de los diagramas de clases.

En el entorno NetBeans, también en Eclipse, y generando Java, podemos utilizar SDE de Visual Paradigm. También Microsoft Visual Studio posee herramientas para la generación de código automático en sus lenguajes propios.

5.10 Generación de diagramas de clases a partir de código

La ingeniería inversa, como proceso general de análisis de una tecnología concreta, proporciona una información de su estructura interna y de sus componentes. Esta forma de analizar los objetos tecnológicos ha sido utilizada en la industria desde hace muchos años y con diferentes objetivos. En el mundo del software, también

encontramos este tipo de procesos que, mediante descompiladores y otras herramientas, intentan obtener información formal de la estructura y acciones de un objeto. Aunque es posible obtener dicha información a partir de los objetos, es más probable que queramos obtener cierta documentación a partir de un código fuente en el caso de que, por ejemplo, dicha documentación haya existido en el pasado pero que, en la actualidad, se considere perdida.

Existen herramientas en el mercado que ofrecen todo tipo de ayudas e integración en entornos de desarrollo. Documentar diseños a partir de código fuente no es tarea sencilla y no debemos pretender grandes proezas de las herramientas; pero sí, en cambio, una ayuda suficiente como para retomar un diseño documental que había desaparecido. De hecho, estas herramientas generan no sólo estructura estática de nuestro sistema, sino también conducta dinámica relativa a las operaciones que pueden realizarse. Como generadores de diseño, parece más sencilla la creación de diagramas estructurales (diagramas de clases) junto con sus relaciones y jerarquías con otras clases (Figura 5.10).

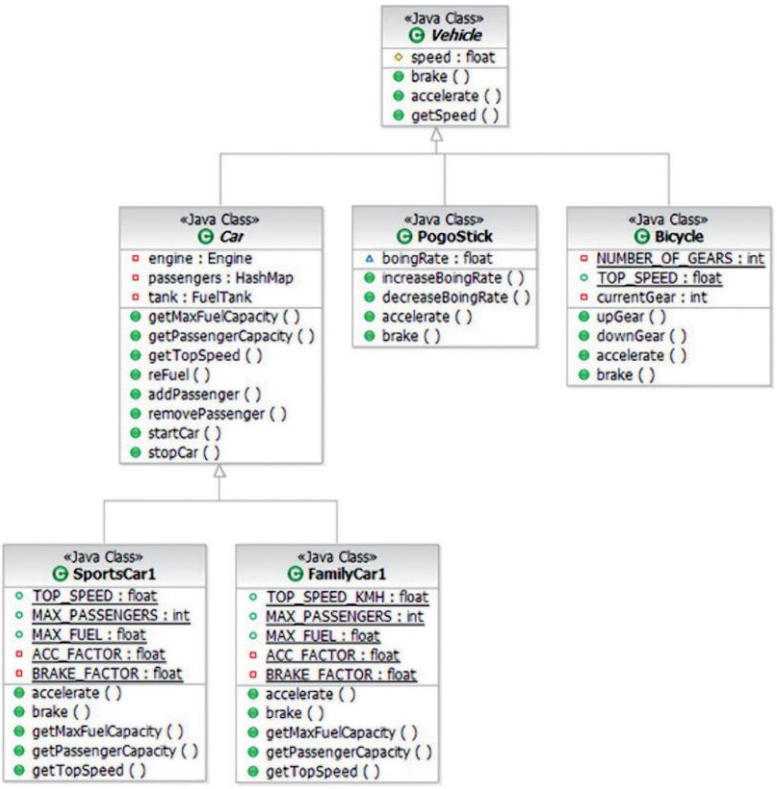


Figura 5.10
Diagrama de herencias desde código Java mediante IBM Rationmal Sofware Architect.

Resumen

Debemos entender el diseño de un sistema informático como el intento de modelar una parcela de la realidad a la que pretendemos emular. La creación de clases, así como la instanciación en objetos, con su nombre, sus atributos y sus conductas determinadas, hacen de este modelado, y, por tanto, de la programación orientada a objetos en general, una forma más intuitiva y cercana a la realidad.

Los diagramas de clase, siguiendo la normas UML (*Unified Modelling Language*), permiten dar una idea de la estructura del sistema que pretendemos crear. Mediante los atributos (características aplicadas a dicha clase) y los métodos (procedimientos que fijarán la conducta manifestada por los objetos pertenecientes a dicha clase), acabarán por configurar los modelos a partir de los cuales emanarán los objetos pertenecientes a la clase en cuestión. Una vez formado el modelo de clase, definimos su grado de visibilidad (el nivel de accesibilidad que tendrán sus propiedades y métodos).

Configuradas las clases y sus instancias emanadas, encontramos diversas relaciones entre ellas:

- Herencia mediante la que una clase base o superclase puede derivar en otro tipo de clases o subclases.

- Composición. Es una forma de asociación en la que un objeto forma parte integral de otro. Si desapareciera el todo, también lo harían sus partes.

- Agregación. Similar a la composición, con la diferencia de que la desaparición del objeto contenedor no afecta a sus componentes, que siguen existiendo.

Estos elementos y relaciones se representan mediante este lenguaje de modelado (UML) haciendo que todos los participantes del proyecto entiendan lo mismo al utilizar un lenguaje común.

Las herramientas informáticas también han abordado el diseño de los modelos y no se han conformado con la asistencia gráfica a la hora de diseñar, sino que también pueden generar código fuente a partir de los diseños previos y viceversa, generar los diseños a partir de un código.

Como siempre, será el presupuesto económico y el tiempo disponible, lo que marque la dedicación que podamos prestar a todas aquellas actividades que enriquecen la mera entrada de código en un ordenador.

Ejercicios de autocomprobación

Indica si las siguientes afirmaciones son verdaderas (V) o falsas (F):

1. Las clases definen a un grupo de objetos, y comparten diversas estructuras y conductas.

2. Como elemento de la clase, el método contiene un procedimiento que fijará la conducta manifestada por los objetos pertenecientes a dicha clase.

3. En la agregación también tenemos objetos contenidos y objetos que lo contienen, pero la relación es parametrizada y su existencia, o inexistencia, no les afecta.

4. Los diagramas de estados describen la conducta dinámica de un objeto como respuesta a estímulos externos.

5. Los diagramas estructurales no presentan modelos estáticos del objeto, como clases, paquetes o componentes, pero pueden ser utilizados como modelos en tiempo de ejecución.

6. La base estructural que tienen los diagramas de clases permite que las herramientas que generan código a partir de ellos puedan ir más allá del diseño general de una base de datos, la configuración de ficheros, campos y demás elementos de estructura.

7. En caso de una herencia, hablaremos de superclase para definir la primera clase sin derivar, y hablaremos de subclases para referirnos a aquellas derivadas de la superclase.

Completa las siguientes afirmaciones:

8. La programación orientada a_____es una forma de _____ el funcionamiento de la realidad mediante_____que simplifican los elementos reales a los que pretenden imitar.

9. Un objeto de software almacena su estado en_____, y manifiesta su conducta a través de los métodos que operan con los_____de los objetos y canalizan la_____entre otros objetos.

10. Los diagramas de_____son aquellos que nos muestran la forma que tienen de reaccionar ante los_____externos. Son diagramas _____, basados en su devenir en tiempo de ejecución.

Las soluciones a los ejercicios de autocomprobación se encuentran al final de esta Unidad Formativa. En caso de que no los hayas contestado correctamente, repasa la parte de la lección correspondiente.

6. Elaboración de diagramas de comportamiento

A lo largo de las unidades anteriores hemos hecho hincapié en los modelos que representaban a las estructuras de la realidad, consideradas, por definición, como componentes inicialmente estáticos. A partir de dicha estructura, aparentemente estable, aparecen nuevos agentes de cambio que los modifican y hacen que se relacionen los unos con los otros. Por ello, con los diagramas estructurales no tenemos suficiente para hacernos una idea de la realidad al que el modelo se está refiriendo. Por lo tanto, para poder completar la simbología representativa de la realidad deberemos recurrir a los **diagramas de comportamiento** que, como su nombre indica, nos darán una idea de cambio, de movimiento, de trabajar con las estructuras. En esta unidad hablaremos de comunicación entre objetos, así como de cambios de estado y flujo de datos entre los elementos del modelo. La creatividad de los diseñadores cobra mayor importancia, porque si crear una buena estructura es un comienzo necesario en el diseño del modelo, no lo es menos el saber descubrir todas las interacciones y movimientos entre las dependencias y jerarquías que existan o, lo que es lo mismo, simbolizar la forma en que dichos elementos "viven" dentro de ese modelo.

6.1 Tipos. Campos de aplicación

A mediados del siglo XX, la ingeniería de sistemas surge como una declaración de diferentes disciplinas científicas para optimizar el desarrollo de productos complejos. Los primeros sectores en aplicarla fueron el armamentístico y aeroespacial. Sin embargo, en la actualidad, son pocos los fabricantes que pueden permitirse prescindir de ella. En un principio, se trataba de documentar al máximo la creación de productos; en la actualidad, se trata de trabajar con modelos (simuladores de realidades o futuras realidades).

Recuerda

Los diagramas de estructura nos muestran cómo son las cosas, cómo son los objetos que vamos a definir en nuestro modelo, mientras que los diagramas de comportamiento nos muestran de qué forma "viven", interactúan o se comunican dichos objetos en nuestro sistema.

Si un sistema es un conjunto de elementos con un objetivo común, los **diagramas de comportamiento** son una buena herramienta para simplificar y plasmar de una forma clara el funcionamiento de un sistema concreto. Podemos clasificar los diagramas de comportamiento en dos grupos: **diagramas de comportamiento** propiamente dichos y **diagramas de interacción** (Figura 6.1); no obstante, este último se considera un subgrupo del anterior:

- **Diagramas de comportamiento**
  - **Diagrama de actividad.** Se utiliza para mostrar la traza de operaciones que se realiza en un proceso del sistema cuestionado.

-**Caso de uso.** Muestra como un elemento cliente (llamado actor) interactúa con el sistema.
-**Máquina de estados.** Muestra el comportamiento de un objeto en cuanto a sus cambios de estado resultantes de la aplicación secuencial de eventos.

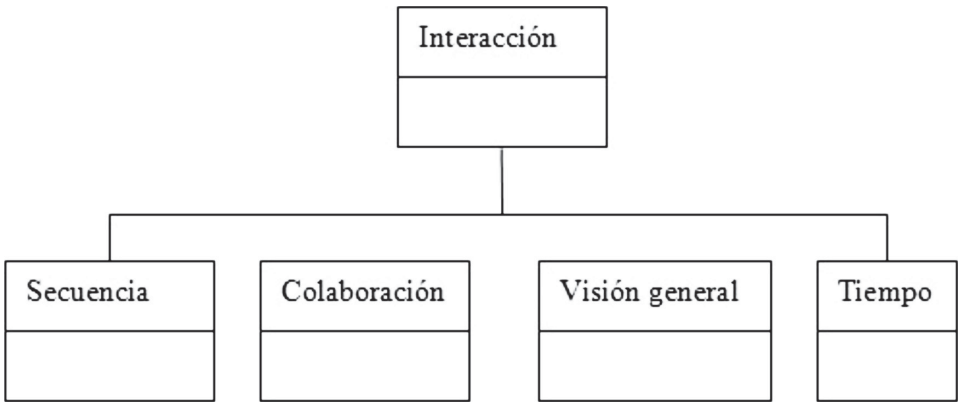


Figura 6.1
Subgrupo de interacción de los diagramas de comportamiento.

- **Diagramas de interacción**
  -**Secuencias.** Los diagramas de secuencia ofrecen una visión de las interacciones de los objetos, ordenadas según el momento en el que ocurren.
  -**Colaboración.** Los diagramas de colaboración muestran la interacción y vinculaciones que existen entre los objetos.
  -**Interacción general.** Focaliza la visión general del flujo de control de las interacciones mediante el uso de marcos adicionales. Se considera una variante del diagrama de actividad.
  -**Tiempo.** El diagrama de tiempo visualiza el comportamiento de los objetos en el transcurso del tiempo (Figura 6.2).

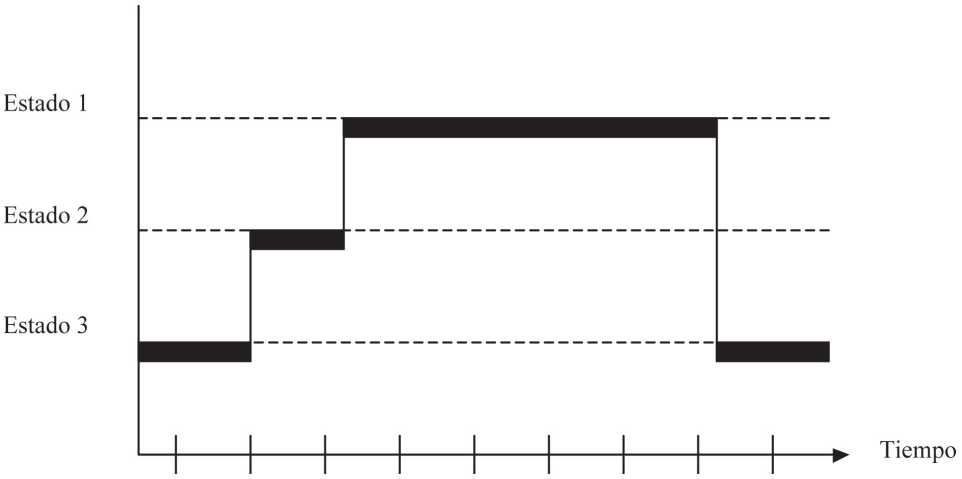


Figura 6.2
Diagrama de tiempo. Los estados del objeto se mantienen estables durante periodos finitos.

6.2 Diagramas de casos de uso

El modelo de **diagrama de caso de uso** presenta los requerimientos de un sistema mediante la captura de las comunicaciones entre usuarios y otros "agentes" que posean influencia en el sistema que estamos modelando.

Los diagramas de caso de uso muestran la interacción entre el sistema y entidades externas a dicho sistema. Estas entidades externas se conocen con el nombre de **actores**. Un actor representa una serie de roles de tipo "humano" (en este caso estaríamos hablando probablemente de un usuario humano), o también de entidades hardware, o incluso de otros sistemas que pueden interactuar con el nuestro. A los actores podemos representarlos con un dibujo o caricatura, o también con un rectángulo, tipo clase, con el nombre del actor (Figura 6.3).

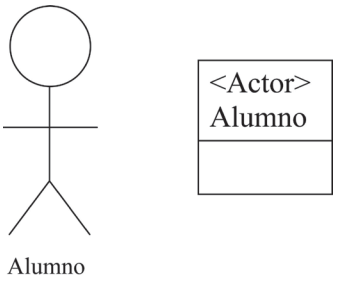


Figura 6.3
Forma de representar a los actores de un modelo.

El caso de uso es una unidad con sentido que nos ofrece una visión extraordinaria de las conductas observables en nuestro sistema. El formato general del diagrama de caso de uso es una elipse, pero con la posibilidad de indicar mediante una flecha el flujo del control.

Los conectores pueden tener múltiples valores en cada extremo del gráfico (Figura 6.4), en el que puede verse que un alumno puede matricularse una sola vez de una asignatura en el colegio; pero, en cambio, el colegio puede tener muchos alumnos matriculándose de la misma asignatura.

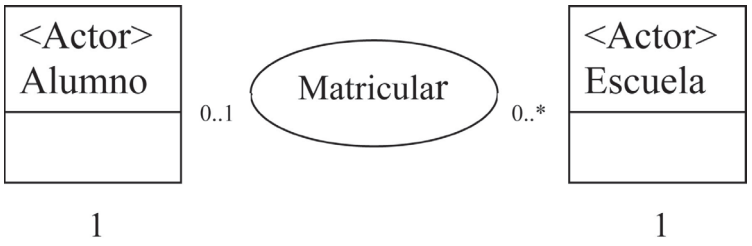


Figura 6.4
El caso de uso "matricular" interacciona con actores externos.

Veamos a continuación cuáles son los principales componentes de un caso de uso (Figura 6.5):

- **Nombre y descripción.** Suele ser un verbo o una frase corta que describa brevemente su naturaleza.

- **Relación de comunicación.** Además de la relación entre actores externos y elementos propios del caso de uso podemos encontrar también relaciones entre otros casos de uso; conteniendo, por tanto, la funcionalidad de ese otro. Cuando esto ocurre, y siempre que se ejecute el caso de uso; se asume que el incluido también lo hará. Por ejemplo, el caso de uso "Matricular" puede tener otro implícito llamado 'Identificar alumno'.

- **Requerimientos.** Representan las funcionalidades que el caso de uso debe proveer al usuario final (actor). Es una declaración de intenciones de que el caso de uso da valor al sistema.

- **Limitaciones.** Las limitaciones son las condiciones previas que deben darse para poder aplicar el caso de uso. También podemos hablar de condiciones posteriores al caso de uso, esto es, circunstancias que deben ser ciertas después de realizarse el caso de uso.

- **Escenarios.** Un escenario es una descripción formal de un flujo de eventos acaecidos durante la ejecución de un caso de uso. Esta descripción textual, entre lo que ocurre con los actores y el sistema, correlaciona el diagrama de secuencias.

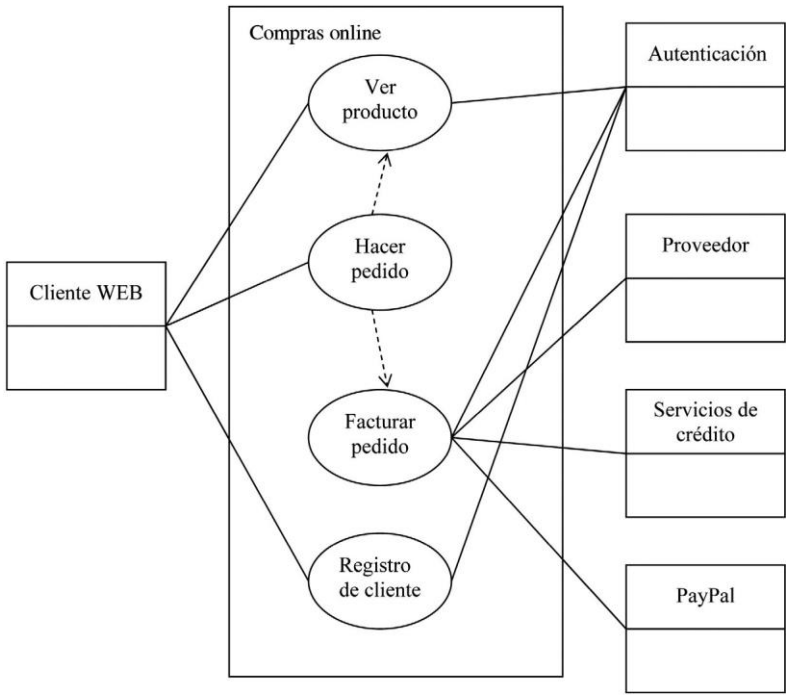


Figura 6.5
Los compradores por Internet (actores), los proveedores (actores) y las entidades de cobro (actores) interactúan con casos de uso (elispses) dento del escenario *Compras on line*.

De la misma manera que un caso de uso puede incluir a otro, podremos también hacer extensiones de uno de ellos. Cuando un alumno solicita la matriculación en alguna asignatura, puede ser necesaria la aprobación de algún elemento externo al sistema (como el director o secretario) que compruebe que el alumno cumple con los requisitos para cursar dicha asignatura (Figura 6.6).

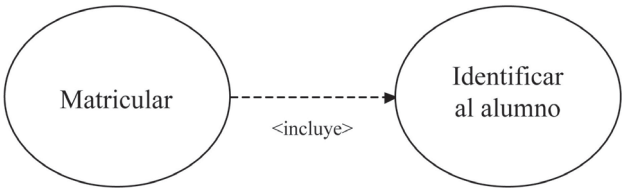


Figura 6.6
Ejemplo de inclusión de casos de uso. En este caso, "Matricular" implica necesariamente identificar al alumno.

6.3 Diagramas de secuencia

Un **diagrama de secuencias** es un tipo de diagrama de interacción que muestra el progreso de los objetos representado por unas líneas, llamadas **líneas de vida**, que se despliegan hacia abajo en el espacio reservado para nuestro modelaje. Los diagramas de secuencia muestran la comunicación entre objetos y las circunstancias en las que se produce (Figura 6.7).

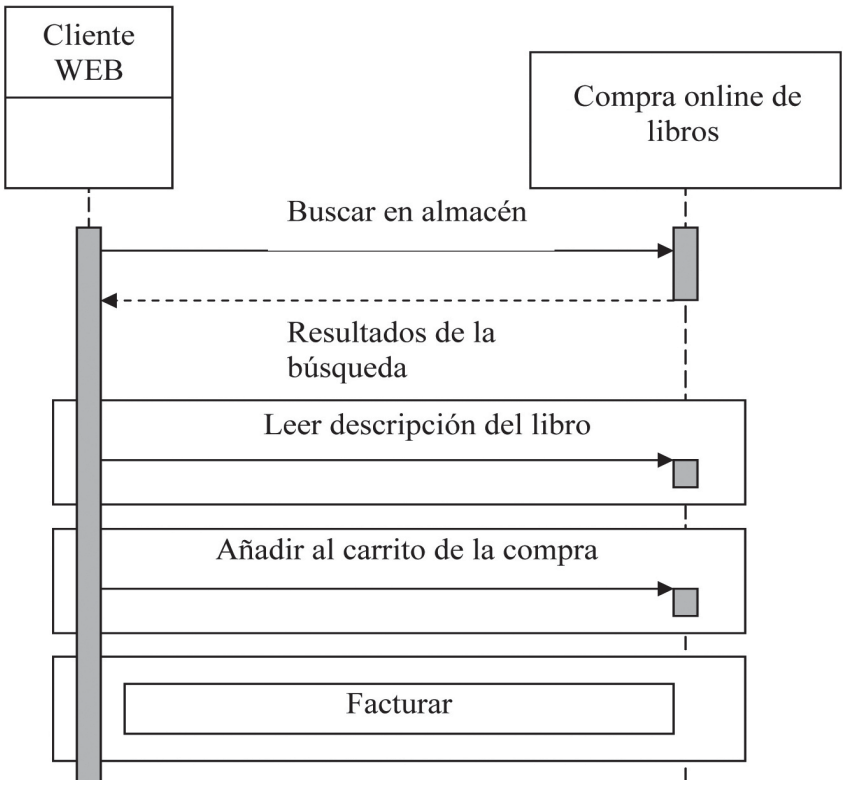


Figura 6.7
Diagrama de secuencia de la compra de un libro por Internet.

- **Línea de vida.** Simboliza el progreso en el tiempo que un objeto realiza como participante del sistema que estamos modelando. Se plasma en el diseño mediante una línea discontinua hacia abajo que parte del símbolo del objeto (Figura 6.8).

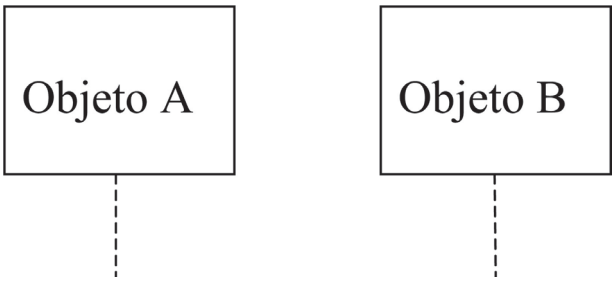


Figura 6.8
Despliegue de líneas de vida hacia abajo.

- **Mensajes y envío de mensajes.** En el transcurso de la línea de vida de un objeto, puede ocurrir que un objeto emita o reciba un mensaje. Este mensaje desencadenará la activación de determinadas operaciones en el objeto receptor. En general, podemos clasificar los mensajes en dos categorías:
  - **Síncrono.** El objeto emisor del mensaje espera constatar que el receptor ha recibido dicho mensaje.
  - **Asíncrono.** El objeto emisor no espera que el mensaje sea recibido y continúa su ejecución de procedimientos si los tuviere.

- **Activación.** A causa de la recepción de un mensaje, se activa una ocurrencia en el objeto. Éste se muestra mediante un rectángulo vertical tan largo como su duración en el tiempo. Tal como muestra la figura 6.9, MSG 1 es un mensaje síncrono (representado por una flecha con cabeza rellenada); MSG 2 es un mensaje asíncrono (representado por una flecha con cabeza normal), y MSG 3 es un mensaje de retorno asíncrono (línea discontinua).

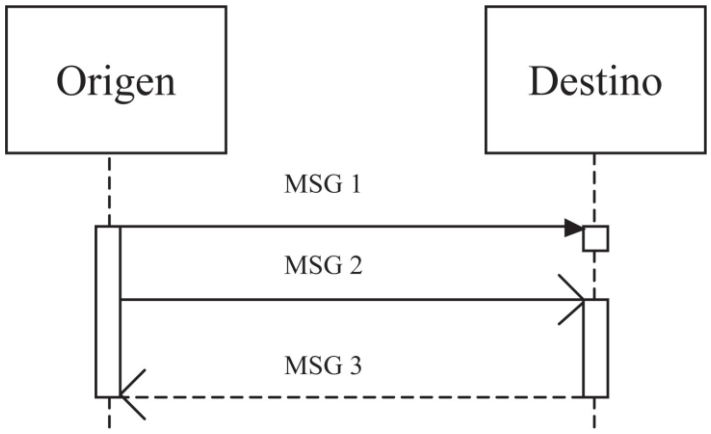


Figura 6.9
Diagrama de secuencia con las líneas de vida, activaciones (rectángulos) y envío y recepción de mensajes entre objetos.

6.4 Diagramas de colaboración

Los **diagramas de colaboración**, también conocidos como **diagramas de comunicación**, describen las interacciones entre los objetos mediante la secuenciación de los mensajes y los roles que adquieren cada uno de ellos. Los diagramas de colaboración son una combinación de los diagramas de clase, los de secuencia y los casos de uso. Puesto que en este diagrama no existe el concepto de tiempo, es preciso secuenciar mediante números el orden de los mensajes. De este modo, podemos ver con mayor detalle la forma estructural (clase) con sus dinamismos (comportamiento). Veamos a continuación algunos de los elementos que componen los diagramas de colaboración:

- **Mensajes.** Debido a que no existe una forma explícita de indicar el tiempo transcurrido, es necesario numerar los mensajes secuencialmente, utilizando si es preciso notación anidada (1.2, 1.3). En caso de que exista una condición para el mensaje, se indicará entre corchetes después de haber indicado previamente el número de secuencia; en caso de la existencia de un bucle, se indicará con un asterisco (Figura 6.10).

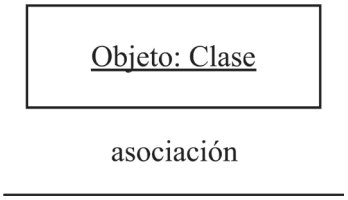


Figura 6.10
Representación de roles y símbolo de asociación.

- **Roles de clase.** Describen cómo se comportan los objetos. Para representarlos se utiliza el símbolo de objeto, pero sin listar los atributos.

- **Asociación.** La asociación de roles (simbolizada por una línea) muestra cómo se comportarán ambos roles en una situación concreta (Figura 6.11).

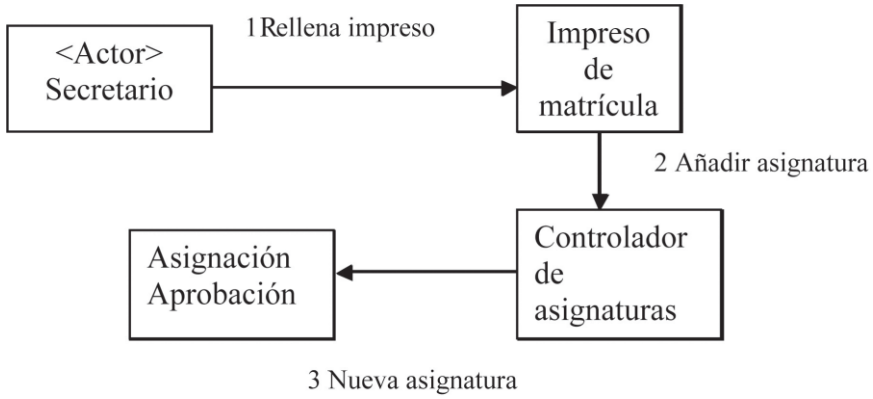


Figura 6.11
Los mensajes secuenciados fluyen, entre los roles, en un diagrama de colaboración.

6.5 Diagramas de actividades

El **diagrama de actividades** se utiliza para mostrar una secuencia de actividades. Se suele indicar el inicio y el final de la actividad, detallando todos los caminos que bifurquen mientras dura su realización.

Es muy interesante para detallar **actividades paralelas**, es decir, aquellas ejecutadas simultáneamente; lo que permite, por lo tanto, determinar sus ejecuciones y dependencias entre sí y tener una buena visualización del comportamiento dinámico del sistema. Veamos de qué se compone el diagrama de actividades (Figura 6.12):

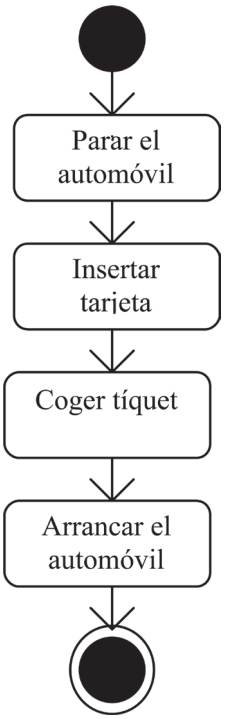


Figura 6.12
Pagar el peaje de la autopista. Simbología inicio y final, y transiciones con flechas entre actividades.

- **Estados iniciales y finales.** El inicio del diagrama se representa con un círculo relleno, del que surgirá el circuito de actividades con sus transiciones. Por último, un círculo relleno circunscrito en uno sin rellenar indica el estado final, y fin de diagrama.

- **Actividad.** La actividad es un consenso que tenemos sobre una determinada secuencia de conducta. El alcance de dicha actividad puede ser muy general o más detallado. La actividad se representa con un rectángulo de puntas redondeadas.

- **Acción.** Es un paso fundamental dentro de una actividad. Utiliza el mismo símbolo, aunque se coloca dentro del de la actividad.

- **Transiciones.** Los momentos de cambio de actividad están simbolizados mediante una flecha de cabeza normal. La transición indica qué actividad o acción es la que sigue a continuación.

- **Decisiones.** Las decisiones, bifurcaciones en el flujo de actividades en función de que se cumpla una determinada condición, se representan con un cuadrado con un vértice de base. En el ejemplo que muestra la figura 6.13 puede verse que no siempre el flujo de actividades es tan sencillo: podría ocurrir que, al introducir la tarjeta de pago, el lector no funcionara; en estos casos, los flujos alternativos deben tenerse muy en cuenta.

- **Condición necesaria.** Mediante corchetes puede mencionarse una condición que se debe cumplir para que el flujo continúe.

- **Combinación (Fusión).** Dos flujos alternativos, creados por una decisión previa, vuelven a converger. El símbolo es el mismo que el utilizado para la decisión.

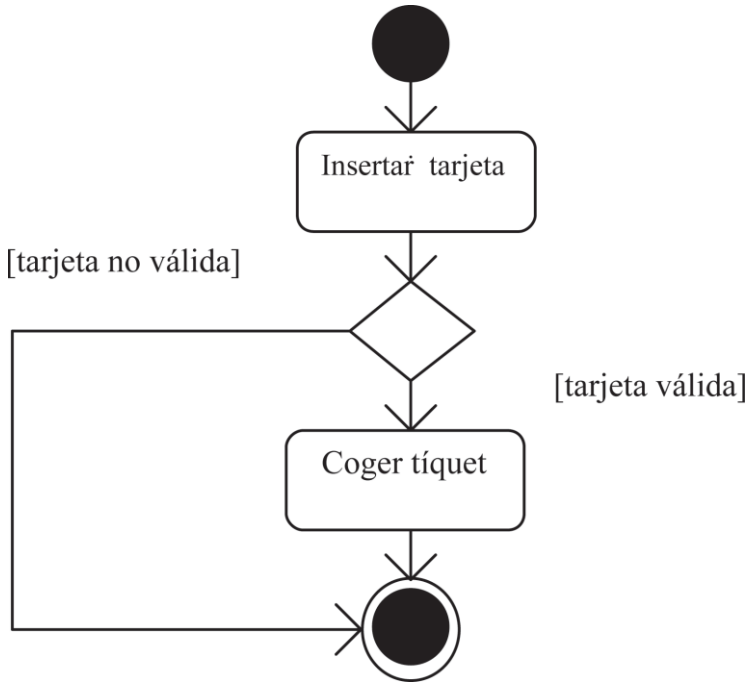


Figura 6.13
Área de decisión en las actividades relacionadas con el pago con tarjeta de crédito en una autopista.

6.6 Diagramas de estado

Un diagrama de estado (diagrama de máquina de estado) se encarga de mostrar el comportamiento que tiene un objeto mientras atraviesa todos los eventos que sufre y que pueden hacerle reaccionar de alguna forma (Figura 6.14).

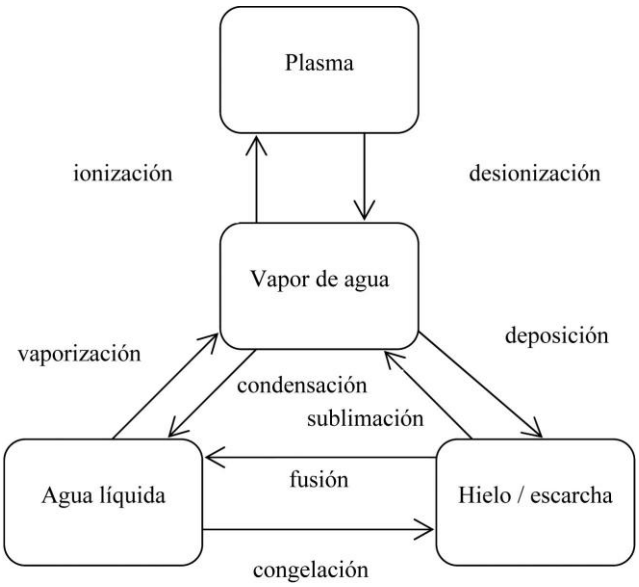


Figura 6.14
Diagrama de máquina de estado de los estados del agua.

Veamos a continuación de qué partes está compuesto un diagrama de estado:

- **Estado.** Un estado de objeto se simboliza con un rectángulo de puntas redondeadas con el nombre del estado en su interior. Por ejemplo, una bombilla tendría dos estados, que se representarían mediante dos rectángulos de puntas redondeadas: uno para el estado encendido y otro para el estado de apagado.

- **Eventos.** Los sucesos que pueden cambiar los estados son finitos. En el caso de la bombilla, el estado puede cambiar al responder al evento "encender" cuando está apagada, y al responder al evento "apagar" cuando está encendida.

- **Señal.** Un mensaje enviado por un objeto que provoca una transición en otro objeto se considera una señal; se considera la señal también como un objeto.

- **Transiciones.** Las transiciones de un estado a otro se simbolizan mediante una flecha. Las transiciones pueden tener los siguientes elementos:
  - **Disparador.** Es la causa de la transición. Como hemos visto puede ser una señal, un evento o un cambio en alguna condición.
  - **Guarda.** Condición necesaria para que el disparador cause la transición.
  - **Efecto.** Acción en el objeto que se desencadena por la transición (Figura 6.15).

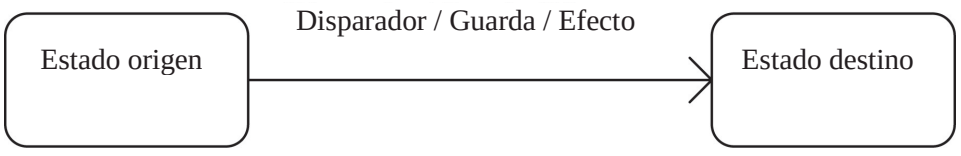


Figura 6.15
La transición de un estado a otro necesita de un disparador (causa), de una condición necesaria (Guarda) y produce un efecto desencadenante.

6.7 Borrado de tablas y tipos

A partir del modelo tradicional de base de datos (el relacional), se ha evolucionado hacia la base de datos objeto relacional, lo que significa que esta nueva base de datos trabaja con los conceptos del modelo relacional, así como el de orientado a objetos:

- **Base de datos relacional.** Los datos están guardados en tablas, y cada una de ellas puede relacionarse con la otra. La tabla de cliente puede contener el dato código de cliente. La tabla pedido de cliente puede contener también el dato código de cliente. Ese dato relaciona ambas tablas.

- **Base de datos objeto relacional.** A partir del lenguaje UML es posible diseñar una base de datos objeto relacional. Se trata de una base más flexible para definir las relaciones entre los datos; es decir, su complejidad, las relaciones y los tipos de datos, herencia, colecciones, composición y asociación.

- **Tipos de datos** (Figura 6.16):
  - **Objeto.** Define una entidad del mundo real. Tiene un nombre, unos atributos y unos métodos.
  - **Colección.** Grupo de datos del mismo tipo.

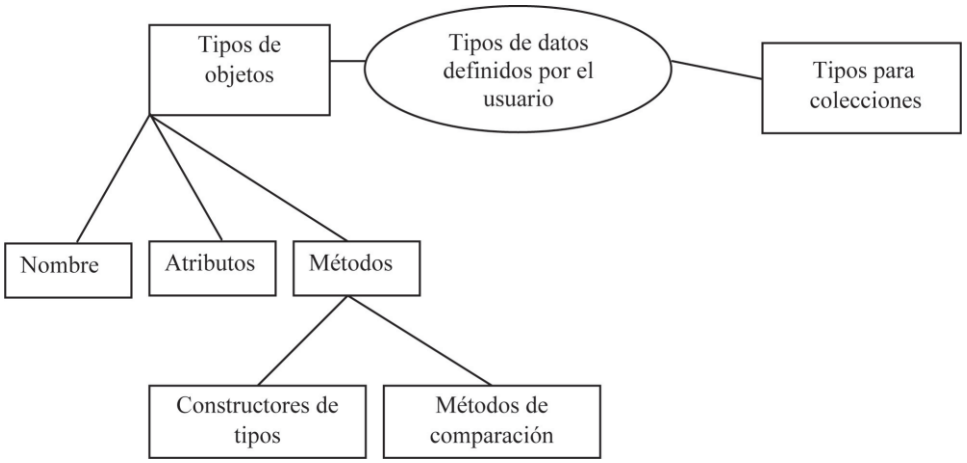


Figura 6.16
Tipos de datos en una base de datos objeto relacional.

- **Borrado de tablas.** La base de datos objeto relacional contiene en los tipos de datos nombre, atributos y métodos. El lenguaje SQL (*Structured Query Language*) con sus variantes de acuerdo con la base de datos concreta permite interrogar y operar con la base de datos:
  - **Borrado de objetos (filas):**
    - DELETE FROM Nombre de la tabla
  - **Borrado de la tabla:**
    - DROP TABLE Nombre de la tabla

Resumen

Los diagramas de comportamiento, a diferencia de los de estructura, nos dan una idea dinámica de cómo funciona un sistema. Los casos de uso presentan 'actores' dentro de un escenario y se comunican con los objetos.

Dentro de los diagramas de comportamiento destacan los diagramas de secuencia y sus líneas de vida, en los que vemos su evolución y dinamismo teniendo el tiempo como un elemento importante.

En los diagramas de colaboración, como forma de comportamiento, el rol y la comunicación juegan un papel determinante, y los diagramas de actividades, quizás los más parecidos a lo que hemos visto con anterioridad en los análisis informáticos, son ajenos al lenguaje UML.

Los diagramas de estado, con sus señales y disparadores, nos brindan la oportunidad de ahondar en la causa – efecto de las reacciones entre objetos y completan la visión dada de los principales diagramas de comportamiento.

Finalmente, la base de datos objeto relacional, que despliega todo el potencial teórico que el lenguaje UML nos ofrece, nos permite comunicarnos con los objetos mediante diferentes lenguajes, siendo uno de los más populares el SQL.

Ejercicios de autocomprobación

Indica si las siguientes afirmaciones son verdaderas (V) o falsas (F):

1. Los diagramas de caso de uso muestran la interacción entre el sistema y entidades externas a dicho sistema. Estas entidades externas se conocen con el nombre de actores.

2. El formato general del diagrama de caso de uso es un cuadrado, pero con la posibilidad de indicar mediante una flecha el flujo del control.

3. Un escenario es una descripción abstracta de un flujo de eventos acaecidos durante la ejecución de un caso de uso.

4. Un diagrama de secuencias es un tipo de diagrama de interacción que muestra el progreso de los objetos representado por unas líneas, llamadas líneas de vida.

5. La línea de vida se plasma en el diseño mediante una flecha hacia abajo que parte del símbolo del objeto.

6. Los diagramas de colaboración son una combinación de los diagramas de clase, los de secuencia y los casos de uso.

7. Los roles de clase describen cómo se comportan los objetos. Para representarlos se utiliza el símbolo de objeto, pero sin listar los atributos.

Completa las siguientes afirmaciones:

8. La interacción general focaliza la visión general del flujo de _____ de las interacciones mediante el uso de _____ adicionales. Se considera una variante del diagrama de _____.

9. Un diagrama de _____ de estado se encarga de mostrar el _____ que tiene un objeto mientras atraviesa todos los _____ que sufre y que pueden hacerle reaccionar de alguna forma.

10. Las decisiones son_____en el flujo de actividades en función de que se cumpla una determinada condición, y se representan con un _____ y con un_____de base.

Las soluciones a los ejercicios de autocomprobación se encuentran al final de esta Unidad Formativa. En caso de que no los hayas contestado correctamente, repasa la parte de la lección correspondiente.

Soluciones de los ejercicios de autocomprobación

Unidad 5

1. F. Las clases definen a un grupo de objetos, y comparten una estructura y una conducta.
2. V
3. V
4. F. Los diagramas de estados describen la conducta dinámica de un sistema como respuesta a estímulos externos.
5. F. Los diagramas estructurales presentan modelos estáticos del objeto, como clases, paquetes o componentes, aunque también pueden ser utilizados como modelos en tiempo de ejecución.
6. F. La base estructural que tienen los diagramas de clases no permite que las herramientas que generan código a partir de ellos puedan ir más allá del diseño general de una base de datos, la configuración de ficheros, campos y demás elementos de estructura.
7. V
8. objetos, simular, modelos.
9. variable, estados, comunicación.
10. comportamiento, estímulos, dinámicos.

Unidad 6

1. V
2. F. El formato general del diagrama de caso de uso es una elipse, pero con la posibilidad de indicar mediante una flecha el flujo del control.
3. F. Un escenario es una descripción formal de un flujo de eventos acaecidos durante la ejecución de un caso de uso.
4. V
5. F. La línea de vida se plasma en el diseño mediante una línea discontinua hacia abajo que parte del símbolo del objeto.
6. V
7. V
8. control, marcos, actividad.
9. máquina, comportamiento, eventos.
10. bifurcaciones, cuadrado, vértice.

Índice

MÓDULO: ENTORNOS DE DESARROLLO

UNIDAD FORMATIVA 3