

Desarrollo de software.

Caso práctico

En BK Programación todos han vuelto ya de sus vacaciones.

Les espera un septiembre agitado, pues acaban de recibir una petición por parte de una cadena hotelera para desarrollar un proyecto software.

Ana, la supervisora de proyectos de BK Programación, se reúne con Juan y María (trabajadores de la empresa) para empezar a planificar el proyecto.

Ana, cuya especialidad es el diseño gráfico de páginas web, acaba de terminar el Ciclo de Grado Medio en Sistemas Microinformáticos y Redes y realizó la FCT en BK Programación. Trabaja en la empresa ayudando en los diseños, y aunque está contenta con su trabajo, le gustaría participar activamente en todas las fases en el proyecto. El problema es que carece de los conocimientos necesarios.

Antonio se ha enterado de la posibilidad de estudiar el nuevo Ciclo de Grado Superior de Diseño de Aplicaciones Multiplataforma a distancia, y está dispuesta a hacerlo. (No tendría que dejar el trabajo).

Le comenta sus planes a su amigo Antonio (que tiene conocimientos básicos de informática), y éste se une a ella.

Después de todo... ¿qué pueden perder?



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Software y programa. Tipos de software.

Caso práctico

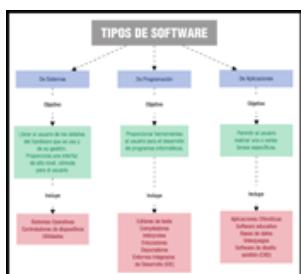
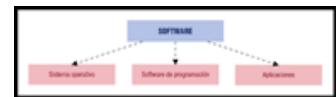
Todos en la empresa están entusiasmados con el proyecto que tienen entre manos. Saben que lo más importante es planificarlo todo de antemano y elegir el tipo de software más adecuado. Ana les escucha hablar y no llega a entender por qué hablan de "tipos de software". ¿Acaso el software no era la parte lógica del ordenador, sin más? ¿Cuáles son los tipos de software?



Es conocido que el ordenador se compone de dos partes bien diferenciadas: Hardware y Software.

El software es el conjunto de programas informáticos que actúan sobre el hardware para ejecutar lo que el usuario deseé.

Según su función se distinguen **tres tipos de software**: software de sistema , software de programación y aplicaciones.



Descripción de la imagen

	<p>El software de sistema es el software base que ha de estar instalado y configurado en nuestro ordenador para que las aplicaciones puedan ejecutarse y funcionar. El principal tipo de software de sistema es el sistema operativo. Algunos ejemplos de sistemas operativos son: Windows, Linux, Mac.</p>
	<p>El software de programación es el conjunto de herramientas que nos permiten desarrollar programas informáticos.</p> <p>Algunos ejemplos son los editores de texto/código, <u>compiladores</u>, <u>intérpretes</u>, entornos de desarrollo integrados (IDE).</p>



Las **aplicaciones informáticas** son un conjunto de programas que tienen una finalidad más o menos concreta.

Son ejemplos de aplicaciones los procesadores de textos, las hojas de cálculo, el software para reproducir música, los videojuegos, etc.

En definitiva, un **programa** es un conjunto de instrucciones escritas en un lenguaje de programación, que indican a la máquina qué operaciones realizar sobre unos determinados datos.

En este tema, nuestro interés se centra en ver cómo se desarrollan las aplicaciones informáticas.

A lo largo de esta primera unidad vas a aprender los conceptos fundamentales de software y las fases del llamado ciclo de vida de una aplicación informática.

También aprenderás a distinguir los diferentes lenguajes de programación y los procesos que ocurren hasta que el programa funciona y realiza la acción deseada.

Para saber más

En el siguiente enlace encontrarás más información de los tipos de software existente, así como ejemplos de cada uno que te ayudarán a profundizar sobre el tema.

[Ampliación sobre los tipos de software.](#)

Reflexiona

Hay varios sistemas operativos en el mercado: _____ Linux, _____ Windows, _____ Mac OS X etc. El más conocido es Windows. A pesar de eso, ¿por qué utilizamos cada vez más Linux?

2.- Relación hardware-software.

Caso práctico

Después de saber ya diferenciar los distintos tipos de software, Ana se le plantea otra cuestión: El software, sea del tipo que sea, se ejecuta sobre los dispositivos físicos del ordenador. ¿Qué relación hay entre ellos?



Como sabemos, al conjunto de dispositivos físicos que conforman un ordenador se le denomina hardware.

Al hablar de un ordenador, la relación hardware-software es inseparable. El software se ejecuta sobre los dispositivos físicos y éstos precisan del software para proporcionar sus funciones.



La primera arquitectura hardware se estableció en 1946 por John Von Neumann, véase en la siguiente figura los principales bloques que la conforman.

En la actualidad, los equipos todavía se basan en esos mismos conceptos.

Esta relación software-hardware la podemos poner de manifiesto desde dos puntos de vista:



[Descripción de la imagen](#)

Desde el punto de vista del sistema operativo

El sistema operativo es el encargado de coordinar al hardware durante el funcionamiento del ordenador, actuando como intermediario entre éste y las aplicaciones que están corriendo en un momento dado.

Todas las aplicaciones necesitan recursos hardware durante su ejecución (tiempo de CPU, espacio en memoria RAM, tratamiento de interrupciones, gestión de los dispositivos de Entrada/Salida, etc.). Será siempre el sistema operativo el encargado de controlar todos estos aspectos de manera "oculta" para las aplicaciones (y para el usuario).

a. Desde el punto de vista de las aplicaciones

Como ya sabemos, una aplicación no es otra cosa que un conjunto de programas y que éstos están escritos en algún lenguaje de programación que el hardware del equipo debe interpretar y ejecutar.

Hay multitud de lenguajes de programación diferentes (como ya veremos en su momento). Sin embargo, todos tienen algo en común: estar escritos con sentencias de un idioma que el ser humano puede aprender y usar fácilmente. Por otra parte, el hardware de un ordenador sólo es capaz de interpretar señales eléctricas (ausencias o presencias de tensión) que, en informática, se traducen en secuencias de 0 y 1 (código binario).

Esto nos hace plantearnos una cuestión: ¿Cómo será capaz el ordenador de "entender" algo escrito en un lenguaje que no es el suyo?.

Como veremos a lo largo de esta unidad, tendrá que pasar algo (un proceso de traducción de código) para que el ordenador ejecute las instrucciones escritas en un lenguaje de programación.

Autoevaluación

Para fabricar un programa informático que se ejecuta en una computadora:

- Hay que escribir las instrucciones en código binario para que las entienda el hardware.
- Sólo es necesario escribir el programa en algún lenguaje de programación y se ejecuta directamente.
- Hay que escribir el programa en algún Lenguaje de Programación y contar con herramientas software que lo traduzcan a código binario.
- Los programas informáticos no se pueden escribir: forman parte de los sistemas operativos.

Incorrecta, ya que el ser humano no tiene capacidad para escribir programas usando 1 y 0.

No es correcta porque el hardware no entiende ese lenguaje.

Muy bien. Esa es la idea...

No es cierta ninguna de las dos afirmaciones.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

3.- Fases a seguir en el desarrollo del software.

Caso práctico

En BK programación ya están manos a la obra. Ada reúne a toda su plantilla para desarrollar el nuevo proyecto.

Ella sabe mejor que nadie que no será sencillo y que habrá que pasar por una serie de etapas. Ana no quiere perderse la reunión, quiere descubrir por qué hay que tomar tantas anotaciones y tantas molestias antes incluso de empezar.

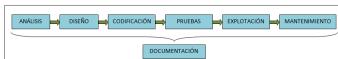


En los puntos siguientes veremos como elegir un modelo de ciclo de vida para el desarrollo de nuestro software.

Independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad.

Entendemos por Desarrollo de Software todo el proceso que ocurre desde que se concibe una idea hasta que un programa está implementado en el ordenador y funcionando.

El proceso de desarrollo, que en un principio puede parecer una tarea simple, consta de una serie de pasos de obligado cumplimiento, pues sólo así podremos garantizar que los programas creados son eficientes, fiables, seguros y responden a las necesidades de los usuarios finales (aquellos que van a utilizar el programa).



Es muy importante dedicar los recursos necesarios en las primeras etapas del desarrollo del software. Avanzar a las etapas finales sin un análisis y diseño libres de errores, implicará que se propaguen durante toda la vida del proyecto y como consecuencia el producto obtenido sea de mala calidad.

Estas etapas son:

Fase	Tareas
Análisis 	<p>Analizar las necesidades de la aplicación a generar.</p> <p>Consensuar todo lo que se requiere del sistema, siendo el punto de partida para las siguientes etapas. Estos requisitos “deberían” ser cerrados para el resto del desarrollo.</p> <p>Se especifican los requisitos funcionales y no funcionales del sistema (ANÁLISIS DE REQUISITOS).</p>
Diseño 	<p>Se divide el sistema en partes y se determina la función de cada una.</p> <p>Realizar los algoritmos necesarios para el cumplimiento de los requisitos planteados en el proyecto.</p> <p>Determinar las herramientas a utilizar en la codificación.</p>

Codificación y compilación	Implementar el código fuente y obtener los ficheros en código máquina que es capaz de entender el ordenador. 
Pruebas 	Los elementos, ya programados, se enlazan para componer el sistema y se comprueba que funciona correctamente y que cumple con los requisitos, antes de ser entregado al usuario final.
Verificación en cliente / Explotación 	Instalamos, configuramos y probamos la aplicación en los equipos del cliente. Es la fase donde el usuario final utiliza el sistema en un entorno de preproducción o pruebas. Si todo va correctamente, el producto estará listo para ser pasado a producción.
Mantenimiento 	Se mantiene el contacto con el cliente para actualizar y modificar la aplicación en el futuro. Se trata de modificaciones al producto, generando nuevas versiones del mismo.
Documentación 	Las tareas de documentación están presentes a lo largo de todo el ciclo de vida del proyecto, por lo que muchos autores no la consideran una etapa en sí misma. De todas las etapas, se documenta y guarda toda la información.

La construcción de software es un proceso que puede llegar a ser muy complejo y que exige gran coordinación y disciplina del grupo de trabajo que lo desarrolle.

Autoevaluación

¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?

- Sí.**
- No.**

Incorrecto. Recuerda que hay que dejar siempre una "puerta abierta" para volver atrás e introducir modificaciones.

Muy bien, vas captando la idea.

Solución

1. Incorrecto
2. Opción correcta

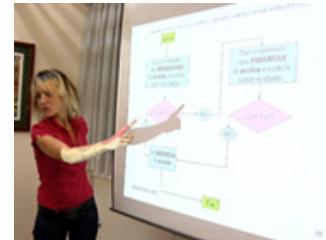
3.1.- Análisis.

Caso práctico

En la reunión de BK acerca del nuevo proyecto Ada, la supervisora, dejó bien claro que lo primero y más importante es tener claro qué queremos que haga el software y con qué herramientas contamos: lo demás vendría después, ya que si esto no está bien planteado, ese error se propagará a todas las fases del proyecto.

—¿Por dónde empezamos? —pregunta Juan.

—ANÁLISIS de REQUISITOS —contesta Ada.



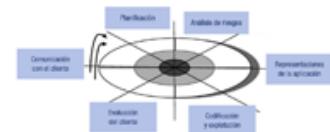
Esta es la primera fase del proyecto. Una vez finalizada, pasamos a la siguiente (diseño).

Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.

¿Qué se hace en esta fase?

En líneas generales, de esta fase obtendremos dos salidas:

- **Documento especificación de requisitos software**, que considerará tanto requisitos funcionales como no funcionales del sistema.
 - **Funcionales**: qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
 - **No funcionales**: tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.



Considerando un programa para la gestión de ventas de una cadena de tiendas, podríamos considerar como ejemplo de requisitos:

Funcionales	No funcionales
Si se desea que la lectura de los productos se realice mediante códigos de barras.	Los PCs suministrados deberán ser de color azul por tratarse del color corporativo.
Si se van a detallar las facturas de compra y sus formatos.	La venta online tendrá que garantizar un servicio ininterrumpido a lo largo de año. La disponibilidad deberá ser 24x7.
Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.	Los materiales entregables deberán cumplir la normativa requerida por la comunidad europea en el sector del comercio.
Si se desea un control del stock en almacén.	La empresa debe hacer sus desarrollos de acuerdo a algún tipo de certificación.
La interfaz tiene que ser fácil de usar para usuarios con pocos conocimientos de informática.	

- **Documento de diseño de arquitectura**, que contiene la descripción de la estructura relacional global del sistema y la especificación de lo que debe hacer cada una de sus partes, así como la manera en que se combinan unas con otras. En ocasiones este documento se genera como una de las primeras tareas de la fase de diseño.

Es imprescindible una buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas. Y habrá que asegurar que se definen aspectos como los siguientes:

- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del usuario cliente y del sistema.
- Relación de objetivos prioritarios y temporización.
- Mecanismos de actuación ante contingencias.
-

Citas para pensar

Todo aquello que no se detecte, o resulte mal entendido en la etapa inicial provocará un fuerte impacto negativo en los requisitos, propagando esta corriente degradante a lo largo de todo el proceso de desarrollo e **incrementando su perjuicio cuanto más tardía sea su detección** *(Bell y Thayer 1976)(Davis 1993)*.

3.2.- Diseño.

Caso práctico

Juan está agobiado por el proyecto. Ya han mantenido comunicaciones con el cliente y saben perfectamente qué debe hacer la aplicación. También tiene una lista de las características hardware de los equipos de su cliente y todos los requisitos. Tiene tanta información que no sabe por dónde empezar.

Decide hablar con Ada. Su supervisora, amable como siempre, le sugiere que empiece a dividir el problema en las partes implicadas.

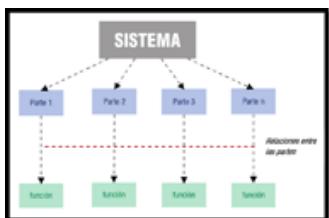


—Vale, Ada, pero, ¿cómo lo divido?

En este punto, ya sabemos lo que hay que hacer, el análisis ya ha definido los requisitos y el documento de análisis arquitectónico identifica cómo dividir el programa para afrontar su desarrollo, pero ¿Cómo hacerlo?.

Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas.

Decidir qué hará exactamente cada parte.



En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.

En este punto, se deben tomar decisiones importantes, tales como:

- Entidades y relaciones de las bases de datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del Sistema Gestor de Base de Datos.
- Etc.

En líneas generales, de esta fase obtendremos dos salidas:

- **Documento de Diseño del Software**, que recoge información de los aspectos anteriormente mencionados.
- **Plan de pruebas** a utilizar en la fase de pruebas, sin entrar en detalles de las mismas.

Nota: en ocasiones, el diseño de arquitectura, que aquí ha sido tratado como una labor a realizar en la fase de análisis, es considerado como una primera tarea de la fase de diseño

Citas para pensar

Design is not just what it looks like and feels like. Design is how it works.

Steve Jobs ("El diseño no es sólo lo que parece y cómo parece. Diseño es cómo se trabaja").

Reflexiona

Según estimaciones, las organizaciones y empresas que crecen más son las que más dinero invierten en sus diseños.

3.3.- Codificación.

Caso práctico

En BK, ya tienen el proyecto dividido en partes.

Ahora llega una parte clave: codificar los pasos y acciones a seguir para que el ordenador los ejecute. En otras palabras, programar la aplicación. Saben que no será fácil, pero afortunadamente cuentan con herramientas CASE que les van a ser de gran ayuda. A Ana el gustaría participar, pero cuando se habla de "código fuente", "ejecutable", etc. sabe que no tiene ni idea y que no tendrá más remedio que estudiarlo si quiere colaborar en esta fase del proyecto.



Durante la fase de codificación se realiza el proceso de programación.

Consiste en elegir un determinado lenguaje de programación y una vez elegido, codificar toda la información anterior (es decir, indicar paso a paso usando un lenguaje de programación, las tareas que debe realizar el ordenador). Al realizar esto se obtiene lo que se llama **código fuente**.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

1. Modularidad: que esté dividido en trozos más pequeños.
2. Corrección: que haga lo que se le pide realmente.
3. Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
4. Eficiencia: que haga un buen uso de los recursos.
5. Portabilidad: que se pueda implementar en cualquier equipo.

```
echo introduce otro numero
read b
if [ $a -eq $b ]
then
{
    echo $a es igual a $b
}
else
{
    if [ $a -lt $b ]
    then
    {
        echo $a es menor q
    }
    else
    {
        echo $a es mayor q
    }
}
```

3.4.- Compilación.

El código fuente es el conjunto de instrucciones que la computadora deberá realizar, escritas por los programadores en algún lenguaje de alto nivel.

Este conjunto de instrucciones no es directamente ejecutable por la máquina, sino que deberá ser traducido al lenguaje máquina, que la computadora será capaz de entender y ejecutar.

En esta fase se hace una traducción de todo el código fuente con el objetivo de pasarlo a lenguaje máquina. Esta traducción es absolutamente necesaria debido a que es el lenguaje que entiende el ordenador.

El proceso de traducción de código fuente a código objeto puede realizarse de dos formas:

1. **Compilación:** El proceso de traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá que enlazar. El software responsable se llama **compilador**.
2. **Interpretación:** El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente. No existe código objeto intermedio. El software responsable se llama **intérprete**. El proceso de traducción es más lento que en el caso de la compilación, pero es recomendable cuando el programador es inexperto, ya que da la detección de errores es más detallada.

3.5.- Pruebas.

Caso práctico

María reúne todos los códigos diseñados y los prepara para implementarlos en el equipo del cliente.

Juan se percata de ello, y le recuerda a su amiga que aún no los han sometido a pruebas. Juan se acuerda bien de la vez que le pasó aquello: *hace dos años, cuando fue a presentar una aplicación a sus clientes, no paraba de dar errores de todo tipo... los clientes, por supuesto, no la aceptaron y Juan perdió un mes de duro trabajo y estuvo a punto de perder su empleo...*



“—No tan deprisa María, tenemos que PROBAR la aplicación”.

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas.

Normalmente, éstas se realizan sobre un conjunto de datos de prueba, que consisten en un conjunto seleccionado y predefinido de datos límite a los que la aplicación es sometida.

La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.

Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

PRUEBAS UNITARIAS

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente).

Como resultado de las pruebas unitarias se genera un **documento de procedimiento de pruebas**, que tendrá como partida el plan de pruebas de la fase de diseño. Éste incluye los resultados obtenidos y deben ser comparados con los resultados esperados que se habrán determinado de antemano.

JUnit es el entorno de pruebas unitarias para Java.

PRUEBAS DE INTEGRACIÓN

Consiste en la puesta en común de todos los programas desarrollados una vez pasadas las pruebas unitarias de cada uno de ellos. Para las pruebas de integración se genera un **documento de procedimiento de pruebas de integración**, que podrá partir de un plan de pruebas de integración si durante la fase de análisis fue generado. Al igual que en las pruebas unitarias los resultados esperados se compararán con los obtenidos.

En el siguiente [enlace](#) encontrarás información interesante sobre los tipos de prueba que debemos hacer a nuestro software.

Autoevaluación

Si las pruebas unitarias se realizan con éxito, ¿es obligatorio realizar las de integración?

- Sí, si la aplicación está formada por más de cinco módulos diferentes.
- Sí, en cualquier caso.

Incorrecto. Una aplicación formada por dos módulos ya es susceptible de producir errores en su interrelación.

Muy bien, vas captando la idea.

Solución

1. Incorrecto
2. Opción correcta

3.6.- Explotación.

Caso práctico

Llega el día de la cita con la cadena hotelera. Ada y Juan se dirigen al hotel donde se va a instalar y configurar la aplicación. Si todo va bien, se irá implementando en los demás hoteles de la cadena.

Ada no quiere que se le pase ni un detalle: lleva consigo la guía de uso y la guía de instalación.

Después de todas las fases anteriores, una vez que las pruebas nos demuestran que el software es fiable, carece de errores y hemos documentado todas las fases, el siguiente paso es la explotación.

Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.

La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.

La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.

En el proceso de instalación, los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados.

Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación.

En este momento, se suelen llevar a cabo las Beta Test, que son las últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo.

Una vez instalada, pasamos a la fase de configuración.

En ella, asignamos los parámetros de funcionamiento normal de la empresa y probamos que la aplicación es operativa. También puede ocurrir que la configuración la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación. Y también, si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla. (Si el software es "a medida", lo más aconsejable es que la hagan aquellos que la han fabricado).

Una vez se ha configurado, el siguiente y último paso es la fase de producción normal. La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.



Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente: será el momento crítico del proyecto.

Reflexiona

Realizas un proyecto software por vez primera y no te das cuenta de documentarlo. Consigues venderlo a buen precio a una empresa. Al cabo de un par de meses te piden que actualices algunas de las funciones, para tener mayor funcionalidad. Estás contento o contenta porque eso significa un ingreso extra. Te paras un momento... ¿Dónde están los códigos? ¿Qué hacía

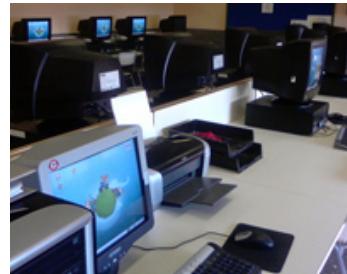
exactamente la aplicación? ¿Cómo se diseñó? No lo recuerdas... Probablemente hayas perdido un ingreso extra y unos buenos clientes.

3.7.- Mantenimiento.

Caso práctico

Ada reúne por última vez durante estas semanas a su equipo. Todos celebran que el proyecto se ha implementado con éxito y que sus clientes han quedado satisfechos.

—Esto aún no ha terminado —comenta Ada—, nos quedan muchas cosas por hacer. Esta tarde me reúno con los clientes. ¿Cómo vamos a gestionar el mantenimiento de la aplicación?



Sería lógico pensar que con la entrega de nuestra aplicación (la instalación y configuración de nuestro proyecto en los equipos del cliente) hemos terminado nuestro trabajo.

En cualquier otro sector laboral esto es así, pero el caso de la construcción de software es muy diferente.

La etapa de mantenimiento es la más larga de todo el ciclo de vida del software.

Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo. Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó.

Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores.

Por todo ello, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

El mantenimiento se define como el proceso de control, mejora y optimización del software.

Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo.

Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

Perfectivos: Para mejorar la funcionalidad del software.

Evolutivos: El cliente propone mejoras para el producto. Implica nuevos requisitos.

Adaptativos: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, nuevas condiciones especificadas por organismos reguladores, etc.

Correctivos: Resolver errores detectados. Sería utópico pensar que esto no vaya a suceder.

Autoevaluación

¿Cuál es, en tu opinión, la etapa más importante del desarrollo de software?

- El análisis de requisitos.
- La codificación.
- Las pruebas y documentación.
- La explotación y el mantenimiento.

Efectivamente. Si esta etapa no está lograda, las demás tampoco lo estarán.

Incorrecto, no necesariamente. Hoy día contamos con ayudas automatizadas (CASE).

No es correcto. Es importante, pero si el análisis no es correcto, no nos servirán de mucho.

No es cierto porque si el software no hace lo que se le pide, éstas no tendrán sentido.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

3.8.- Documentación.

Caso práctico

Ada ha quedado dentro de dos días con su cliente. Pregunta a María por todos los dossiers de documentación. La pálida expresión de la joven hace que Ada arda en desesperación: "— ¿No habéis documentado las etapas? ¿Cómo voy a explicarle al cliente y sus empleados el funcionamiento del software? ¿Cómo vamos a realizar su mantenimiento?".



Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.

En realidad, la documentación no debe ser considerada como una etapa más en el desarrollo del proyecto, la elaboración de documentos es constante durante todo su ciclo de vida.

Documentar un proyecto se hace necesario para dar toda la información a los usuarios de nuestro software y para poder acometer futuras revisiones.

Una correcta documentación permitirá pasar de una etapa a otra de forma clara y definida. También se hace imprescindible para la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

De acuerdo al destinatario final de los documentos, podemos distinguir tres tipos:

Guías técnicas. Dirigidos a personal técnico en informática (analistas y programadores).	
Aspectos que quedan reflejados:	El diseño de la aplicación. La codificación de los programas. Las pruebas realizadas.
¿Cuál es su objetivo?	Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.
Guías de uso. Dirigidos a usuarios que van a usar la aplicación(clientes).	
Aspectos que quedan reflejados:	Descripción de la funcionalidad de la aplicación. Forma de comenzar a ejecutar la aplicación. Ejemplos de uso del programa. Requerimientos software de la aplicación. Solución de los posibles problemas que se pueden presentar.
¿Cuál es su objetivo?	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.

Guías de instalación. Dirigidos al personal informático responsable de la instalación.	
Aspectos que quedan reflejados:	Puesta en marcha. Explotación. Seguridad del sistema.
¿Cuál es su objetivo?	Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

Reflexiona

Según estimaciones, el 26% de los grandes proyectos de software fracasan, el 48% deben modificarse drásticamente y sólo el 26% tienen rotundo éxito. La principal causa del fracaso de un proyecto es la falta de una buena planificación de las etapas y mala gestión de los pasos a seguir. ¿Por qué el porcentaje de fracaso es tan grande? ¿Por qué piensas que estas causas son tan determinantes?

[Mostrar retroalimentación](#)

Porque los errores en la planificación inicial se propagarán en cascada al resto de etapas del desarrollo.

4.- Ciclos de vida del software.

Se puede definir el **ciclo de vida de un proyecto (ciclo de vida del software)** como: conjunto de fases o etapas, procesos y actividades requeridas para ofrecer, desarrollar, probar, integrar, explotar y mantener un producto software.

Al principio, el desarrollo de una aplicación era un proceso individualizado, carente de planificación donde únicamente se hacía una codificación y prueba/depuración del código a desarrollar. Pero pronto se detectaron muchos inconvenientes:

- Dependencia total de la persona que programó.
- Se desconoce el progreso y la calidad del proyecto.
- Falta de flexibilidad ante cambios.
- Posible incremento del coste o incluso imposibilidad de completarlo.
- Puede no reflejar las necesidades del cliente.

De ahí surgió la necesidad de hacer desarrollos más estructurados, aportando valor añadido y calidad al producto final, apareciendo el concepto de **ciclo de vida del software**.

Algunas ventajas que se consiguen son:

- En las primeras fases, aunque no haya líneas de código, invertir en pensar el diseño es avanzar en la construcción del sistema, pues facilitará la codificación.
- Asegura un desarrollo progresivo, con controles sistemáticos, que permite detectar defectos con mucha antelación.
- El seguimiento del proceso permite controlar los plazos de entrega retrasados y los costes excesivos.
- La documentación se realiza de manera formal y estandarizada simultáneamente al desarrollo, lo que facilita la comunicación interna entre el equipo de desarrollo y la de éste con los usuarios.
- Aumenta la visibilidad y la posibilidad de control para la gestión del proyecto.
- Supone una guía para el personal de desarrollo, marcando las tareas a realizar en cada momento.
- Minimiza la necesidad de rehacer trabajo y los problemas de puesta a punto.

Diversos autores han planteado distintos modelos de ciclos de vida, pero los más conocidos y utilizados son los citados a continuación:

1.- Modelo en Cascada

Es el modelo de vida clásico del software.

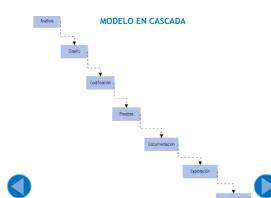
Se pasa de unas etapas a otras sin retorno posible, cualquier error en las fases iniciales ya no será subsanable aunque sea detectado más adelante.

Este escaso margen de error lo hace prácticamente imposible de utilizar. Sólo es aplicable en pequeños desarrollos.

Todos los requisitos son planteados para hacer un único recorrido del proyecto.

Características:

- Requiere conocimiento previo y absoluto de los requerimientos del sistema.
- No hay retornos en las etapas: si hay algún error durante el proceso hay que empezar desde el principio.
- No permite modificaciones ni actualizaciones del software.
- Es un modelo utópico.



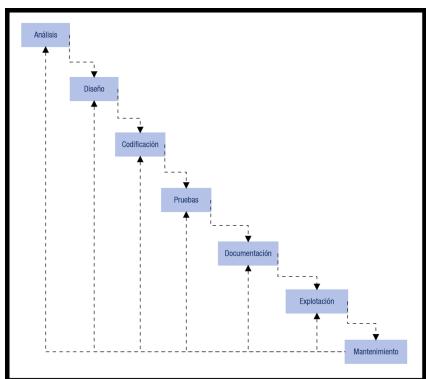
2.- Modelo en Cascada con Realimentación

Es uno de los modelos más utilizados. Proviene del modelo anterior, pero se introduce una realimentación entre etapas, de forma que podamos volver atrás en cualquier momento para corregir, modificar o depurar algún aspecto. No obstante, si se prevén muchos cambios durante el desarrollo no es el modelo más idóneo.

Es el modelo perfecto si el proyecto es rígido (pocos cambios, poco evolutivo) y los requisitos están claros.

Características:

- Es uno de los modelos más utilizados.
- Se puede retornar a etapas anteriores para introducir modificaciones o depurar errores.
- Idóneo para proyectos más o menos rígidos y con requisitos claros.
- Los errores detectados una vez concluido el proyecto pueden provocar que haya que empezar desde cero.



3.- Modelos Evolutivos

Son más modernos que los anteriores. Tienen en cuenta la naturaleza cambiante y evolutiva del software.

Distinguimos dos variantes:

3.1.- Modelo Iterativo Incremental

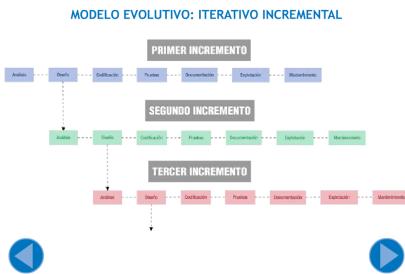
Está basado en el modelo en cascada con realimentación, donde las fases se repiten y refinan, y van propagando su mejora a las fases siguientes.

Cada iteración cubre una parte de los requisitos requeridos, generando versiones parciales y crecientes para el producto software en desarrollo. Cada versión obtenida será el punto de partida para la siguiente iteración.

Los incrementos a considerar en cada vuelta ya vienen establecidos desde las etapas iniciales.

Características:

- Actualmente, no se ponen en el mercado los productos completos, sino versiones.
- Permite una evolución temporal.
- Vemos que se trata de varios ciclos en cascada que se repiten y se refinan en cada incremento.
- Las sucesivas versiones del producto son cada vez más completas hasta llegar al producto final.



3.2.- Modelo en Espiral

Es una combinación del modelo anterior con el modelo en cascada. En él, el software se va construyendo repetidamente en forma de versiones que son cada vez mejores, debido a que incrementan la funcionalidad en cada versión. Es un modelo bastante complejo.

Características:

- Se divide en 6 zonas llamadas regiones de tareas: Comunicación con el cliente, Planificación, Análisis de riesgos, Representación de la aplicación, Codificación y explotación y Evaluación del cliente.
- Se adapta completamente a la naturaleza evolutiva del software.
- Reduce los riesgos antes de que sean problemáticos.



3.3.- Modelos ágiles

Se trata de modelos que están ganando gran presencia en los desarrollos software. Muy centrados en la satisfacción del cliente, muestran gran flexibilidad a la aparición de nuevos requisitos, incluso durante el desarrollo de la aplicación. Al tratarse de metodologías evolutivas, el desarrollo es incremental, pero los incrementos son cortos y están abiertos al solapado de unas fases con otras. La comunicación entre los integrantes del equipo de trabajo y de éstos con el cliente son constantes. Un ejemplo de metodología ágil es [Scrum](#).

Autoevaluación

Si queremos construir una aplicación pequeña, y se prevé que no sufrirá grandes cambios durante su vida, ¿sería el modelo de ciclo de vida en espiral el más recomendable?

- Sí.
- No.

Incorrecto, si la aplicación no sufrirá grandes cambios, este modelo no es recomendable.

Efectivamente, por las características de esta aplicación, pensaríamos mejor en el modelo en cascada con realimentación.

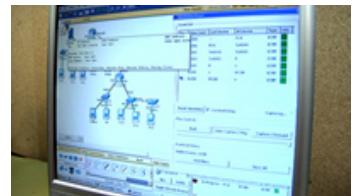
Solución

1. Incorrecto
2. Opción correcta

5.- Herramientas de apoyo al desarrollo del software.

En la práctica, para llevar a cabo varias de las etapas vistas en el punto anterior contamos con herramientas informáticas, cuya finalidad principal es automatizar las tareas y ganar fiabilidad y tiempo.

Esto nos va a permitir centrarnos en los requerimientos del sistema y el análisis del mismo, que son las causas principales de los fallos del software.



Las herramientas **CASE (Computer Aided Software Engineering)** son un conjunto de aplicaciones que se utilizan en el desarrollo de software con el objetivo de automatizar las fases del desarrollo y reducir tanto costes como tiempo del proceso. Como consecuencia, se consigue mejorar la productividad, la calidad del proceso y el resultado final.

¿En qué fases del proceso nos pueden ayudar? En el diseño del proyecto, en la codificación de nuestro diseño a partir de su apariencia visual, detección de errores...

En concreto, estas herramientas permiten:

- Mejorar la planificación del proyecto.
- Darle agilidad al proceso.
- Poder reutilizar partes del software en proyectos futuros.
- Hacer que las aplicaciones respondan a estándares.
- Mejorar la tarea del mantenimiento de los programas.
- Mejorar el proceso de desarrollo, al permitir visualizar las fases de forma gráfica.

CLASIFICACIÓN

Normalmente, las herramientas CASE se clasifican en función de las fases del ciclo de vida del software en la que ofrecen ayuda:

U-CASE: ofrece ayuda en las fases de planificación y análisis de requisitos.

M-CASE: ofrece ayuda en análisis y diseño.

L-CASE: ayuda en la programación del software, detección de errores del código, depuración de programas y pruebas y en la generación de la documentación del proyecto.

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder...

Pulsa este [enlace](#) para ver una ampliación de los tipos y ayudas concretas de la herramientas CASE.

Para saber más

En el siguiente enlace se presenta una ampliación de los tipos y ayudas concretas de la herramientas CASE.

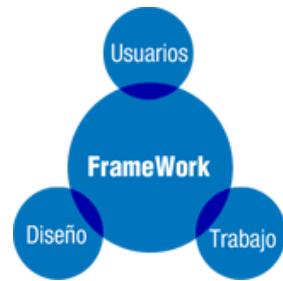
[Ayudas concretas de CASE.](#)

6.- Frameworks.

Un Framework es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero.

Se trata de una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.

Con el uso de framework podemos pasar más tiempo analizando los requerimientos del sistema y las especificaciones técnicas de nuestra aplicación, ya que la tarea laboriosa de los detalles de programación queda resuelta.



Ventajas de utilizar un framework:

Desarrollo rápido de software.

Reutilización de partes de código para otras aplicaciones.

Diseño uniforme del software.

Portabilidad de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.

Inconvenientes:

Gran dependencia del código respecto al framework utilizado (sin cambiarnos de framework, habrá que reescribir gran parte de la aplicación).

La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.

Para saber más

El uso creciente de frameworks hace que tengamos que estar reciclando los constantemente. En el siguiente enlace, hay un documento muy interesante de sus principales características, ventajas y formas de uso:

[Características de frameworks.](#)

Ejemplos de Frameworks:

- **.NET** es un framework para desarrollar aplicaciones sobre Windows. Ofrece el "Visual Studio .net" que nos da facilidades para construir aplicaciones y su motor es el ".Net framework" que permite ejecutar dichas aplicaciones.
- **Spring de Java**. Es un conjunto de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones Java.
- **Qt**. Framework multiplataforma para el lenguaje C++. Admite adaptaciones para ser utilizado en otros lenguajes.
- **Angular**. Framework de Javascript para aplicaciones web.

7.- Lenguajes de programación.

Caso práctico

Una de los aspectos del proyecto que más preocupa a Ada es la elección del lenguaje de programación a utilizar.

Necesita tener muy claros los requerimientos del cliente para enfocar correctamente la elección, pues según sean éstos unos lenguajes serán más efectivos que otros.

Ya dijimos anteriormente que los programas informáticos están escritos usando algún lenguaje de programación. Por tanto, podemos definir un Lenguaje de Programación como un idioma creado de forma artificial, formado por un conjunto de símbolos y normas que se aplican sobre un alfabeto para obtener un código, que el hardware de la computadora pueda entender y ejecutar. Es decir, un lenguaje de programación es el conjunto de:

Alfabeto: conjunto de símbolos permitidos.

Sintaxis: normas de construcción permitidas de los símbolos del lenguaje.

Semántica: significado de las construcciones para hacer acciones válidas.

Hay multitud de lenguajes de programación, cada uno con unos símbolos y unas estructuras diferentes. Además, cada lenguaje está enfocado a la programación de tareas o áreas determinadas. Por ello, la elección del lenguaje a utilizar en un proyecto es una cuestión de extrema importancia.

Como ya sabemos, es en la etapa de diseño cuando típicamente se elige el lenguaje de programación a utilizar y en la fase de desarrollo cuando se hace uso de ellos.

Los lenguajes de programación son los que nos permiten comunicarnos con el hardware del ordenador.

En otras palabras, es muy importante tener muy clara la función de los lenguajes de programación. Son los instrumentos que tenemos para que el ordenador realice las tareas que necesitamos.

Los lenguajes de programación han sufrido su propia evolución, como se puede apreciar en la figura siguiente:



Para saber más

En el siguiente enlace, verás la evolución entre los distintos tipos de Lenguajes de Programación en la historia.

[Evolución de los Lenguajes de Programación.](#)

7.1.- Características de los lenguajes de programación

Características de los Lenguajes de Programación

Lenguaje máquina:

Sus instrucciones son combinaciones de unos y ceros.
Es el único lenguaje que entiende directamente el ordenador. (No necesita traducción).
Fue el primer lenguaje utilizado.
Es único para cada procesador (no es portable de un equipo a otro).
Hoy día nadie programa en este lenguaje.

Lenguaje ensamblador:

Sustituyó al lenguaje máquina para facilitar la labor de programación.
En lugar de unos y ceros se programa usando mnemotécnicos (instrucciones complejas).
Necesita traducción al lenguaje máquina para poder ejecutarse.
Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo.
Es difícil de utilizar.

Lenguaje de alto nivel basados en código:

Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación.
En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del idioma inglés.
(Necesita traducción al lenguaje máquina).
Son más cercanos al razonamiento humano.
Necesita traducción al lenguaje máquina.
Son utilizados hoy día, aunque la tendencia es que cada vez menos.

Lenguajes visuales:

Están sustituyendo a los lenguajes de alto nivel basados en código.
En lugar de sentencias escritas, se programa gráficamente usando el ratón y diseñando directamente la apariencia del software.
Su correspondiente código se genera automáticamente.
Necesitan traducción al lenguaje máquina.
Son completamente portables de un equipo a otro.

7.2.- Clasificación de los lenguajes de programación

Ya sabemos que los lenguajes de programación han evolucionado, y siguen haciéndolo, siempre hacia la mayor usabilidad de los mismos (que el mayor número posible de usuarios lo utilicen y exploten).

La elección del lenguaje de programación para codificar un programa dependerá de las características del problema a resolver.

Podemos clasificar los distintos tipos de Lenguajes de Programación en base a distintos criterios:

Según lo cerca que esté del lenguaje humano

- **Lenguajes de programación de alto nivel.** Los lenguajes de programación de alto nivel se caracterizan por traducir los algoritmos a un lenguaje mucha más fácil de entender para el programador. Estos lenguajes no están orientados a la máquina (donde se va a ejecutar) lo cual hace que los programas escritos con este tipo de lenguaje se puedan ejecutar en cualquier tipo de ordenador. Tienen el inconveniente de que este conjunto de instrucciones no es directamente ejecutable por la máquina, sino que deberá ser traducido al lenguaje máquina, que la computadora será capaz de entender y ejecutar.
- **Lenguajes de programación de bajo nivel.** Los lenguajes de bajo nivel son lenguajes que aprovechan al máximo los recursos del ordenador donde se van a ejecutar, Los lenguajes de más bajo nivel son los lenguajes máquinas, A este nivel le sigue el lenguaje ensamblador.
 - **Lenguaje máquina.** Se programan sus registros directamente con 0s y 1s. Difícil de programar y resolver errores.
 - **Lenguaje ensamblador.** Se trata de un primer nivel de abstracción respecto al lenguaje máquina, aunque conceptualmente está mucho más cercano al equipo que al razonamiento humano.

Estos lenguajes tienen importantes inconvenientes, por ejemplo:

- Su adaptación a la máquina (donde se va a ejecutar) hace que no se pueda utilizar en otro ordenador al menos que tenga las mismas características que el ordenador (para el que se hizo).
- Dificultad en el manejo de dicho lenguaje.

Según su forma de ejecución

Un ordenador realizará cada una de las instrucciones indicadas en un programa. Dicho programa podrá estar escrito en muchos lenguajes de programación pero solo hay uno que entiende el ordenador: el lenguaje máquina.

Si el programa no está escrito en dicho lenguaje habrá que hacer una traducción para que el ordenador lo entienda.

Según como se realice dicha traducción, se pueden dividir en dos grupos:

- **Lenguajes compilados:** Son aquellos lenguajes que se traducen a través de un programa traductor llamado compilador. Dicho programa traduce todo el programa al lenguaje máquina, generando un nuevo fichero ejecutable, que contiene la misma información que el fichero original (código fuente) pero escrito en lenguaje máquina. Si no hay cambios en el fichero original, no hará falta volver a realizar, de nuevo, dicha traducción. Por ejemplo: C, Pascal, etc.
- **Lenguajes interpretados:** Un programa escrito en un lenguaje interpretado supone el tener que traducir el programa cada vez que se quiera ejecutar. A los programas, que realizan dicha traducción, se les llama intérpretes. Dichos programas no generan un fichero ejecutable, por ello, es necesario realizar la traducción cada una de las veces que se ejecute el programa. Por ejemplo: Basic, etc.

La ventaja de los lenguajes compilados frente a los interpretados es que no hace falta traducir el programa cada vez; sin embargo, su inconveniente es que no podremos ejecutar el programa hasta que no haya ningún error. Los intérpretes van traduciendo y ejecutando instrucción a instrucción lo cual hace que se pueda

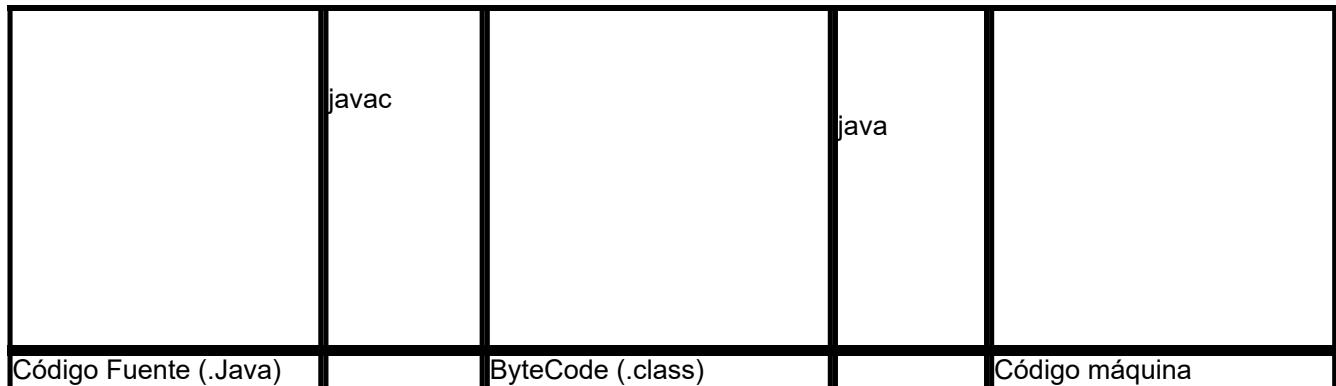
ejecutar el programa (aunque haya errores). En el momento que llegue a dicha instrucción errónea, se parará el programa.

Para obtener las ventajas de ambos tipos de lenguajes, algunos combinan estas dos tareas:

1. Primero, el programa original (en Java, los ficheros donde se guarda el programa fuente tienen extensión “java”) se hace una primera traducción pero no al lenguaje máquina sino a un lenguaje intermedio (en Java es bytecodes). De dicha traducción se obtiene un fichero (en Java tienen extensión “class”). Esto equivaldría a la fase de compilación.
2. En una segunda fase, dicho archivo es traducido (interpretado) en cada ejecución.

Esto es lo que realiza, por ejemplo, Java. A estos lenguajes se les llama **lenguajes intermedios**.

Durante todo este proceso (que se realizan en los lenguajes intermedios), el código pasa por diferentes estados. La siguiente figura muestra el proceso de transformación del código fuente a máquina para el lenguaje Java.



Por lo tanto tendremos:

Código Fuente	<p>Es el escrito por los programadores en algún editor de texto. Se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesarias que debe seguir el ordenador para implementar el algoritmo.</p> <p>Se trata de información conceptualmente más cercana al programador que a la máquina, que no es capaz de ejecutarlo directamente.</p> <p>Un aspecto muy importante en esta fase es la elaboración previa de un algoritmo, que lo definimos como un conjunto de pasos a seguir para obtener la solución del problema. El algoritmo lo diseñamos en <u>pseudocódigo</u> y con él, la codificación posterior a algún Lenguaje de Programación determinado será más rápida y directa.</p> <p>Para obtener el código fuente de una aplicación informática:</p> <ol style="list-style-type: none"> 1. Se debe partir de las etapas anteriores de análisis y diseño. 2. Se diseñará un <u>algoritmo</u> que simbolice los pasos a seguir para la resolución del problema. 3. Se elegirá una Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar. 4. Se procederá a la codificación del algoritmo antes diseñado. <p>La culminación de la obtención de código fuente es un documento con la codificación de todos los <u>módulos</u>, <u>funciones</u>, bibliotecas y <u>procedimientos</u> necesarios para codificar la aplicación.</p> <p>Puesto que, como hemos dicho antes, este código no es inteligible por la máquina, habrá que TRADUCIRLO, obteniendo así un código equivalente pero ya traducido a código binario que se llama código objeto. Que no será directamente ejecutable por la computadora si éste ha sido compilado.</p> <p>Un aspecto importante a tener en cuenta es su licencia. Así, en base a ella, podemos distinguir dos tipos de código fuente:</p> <ul style="list-style-type: none"> • Código fuente abierto. Es aquel que está disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
----------------------	--

	<ul style="list-style-type: none"> • Código fuente cerrado. Es aquel que no tenemos permiso para editarlo.
Código Objeto	<p>Se trata de un código intermedio para los lenguajes intermedios, por lo tanto no es directamente ejecutable por el equipo. Para que la máquina pueda ejecutar el programa es necesario transformarlo al lenguaje que ésta maneja con un intérprete. El código objeto se obtiene mediante el uso de un compilador.</p> <p>El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora. Es un código intermedio entre el código fuente y el ejecutable.</p> <p>En definitiva, es el resultado de traducir código fuente a un código equivalente formado por unos y ceros que aún no puede ser ejecutado directamente por la computadora.</p> <p>Consiste en un <u>bytecode</u> (código binario) que está distribuido en varios archivos, cada uno de los cuales corresponde a cada programa fuente compilado.</p> <p>Sólo se genera código objeto una vez que el código fuente está libre de errores sintácticos y semánticos.</p>
Código Ejecutable	<p>Es el código máquina resultante de enlazar los archivos de código objeto con ciertas <u>rutinas</u> y <u>bibliotecas</u> necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y de proceder a su ejecución.</p> <p>Los programas compilados no producen código objeto. El proceso de compilación ya realiza el paso de fuente a ejecutable directamente.</p> <p>Para obtener un sólo archivo ejecutable, habrá que enlazar todos los archivos de código objeto, a través de un software llamado <u>linker</u> (enlazador) y obtener así un único archivo que ya sí es ejecutable directamente por la computadora.</p> <p>También es conocido como código máquina y ya sí es directamente inteligible por la computadora.</p> <p>En el esquema de generación de código Generación de código ejecutable, vemos el proceso completo para la generación de ejecutables.</p> <p>A partir de un editor, escribimos el lenguaje fuente con algún Lenguaje de programación. (En el ejemplo, se usa Java).</p> <p>A continuación, el código fuente se compila obteniendo código objeto o bytecode.</p> <p>Ese bytecode, a través de la máquina virtual (se verá en el siguiente punto), pasa a código máquina, ya directamente ejecutable por la computadora.</p>

Lenguajes imperativos vs declarativos

- En la **programación imperativa** se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema. P.e. **Basic, C, C++, Fortran, Pascal, Perl, PHP, Java**, lenguajes ensamblador.
- En la **programación declarativa** las sentencias que se utilizan describen el problema que se quiere solucionar, pero no las instrucciones necesarias para hacerlo. Algunos ejemplos son:
 - Lógicos: **prolog**.
 - Algebraicos: **SQL**.
 - Funcionales: **Haskell**.

Según la técnica de programación utilizada

- **Lenguajes de programación estructurados.** Usan la técnica de programación estructurada. Ejemplos: **Pascal, C, etc.**

- **Lenguajes de programación orientados a objetos (POO).** Usan la técnica de programación orientada a objetos. Ejemplos: **C++ (C plus plus), Java, Ada, C#, .Net**, etc.
- **Lenguajes de programación visuales.** Basados en las técnicas anteriores, permiten programar gráficamente, posteriormente se obtiene un código equivalente de forma automática. Ejemplos: **Visual Basic.Net, C#, .Net**, etc.

A pesar de la inmensa cantidad de lenguajes de programación existentes, Java, C, C++, PHP y Visual Basic concentran alrededor del 60% del interés de la comunidad informática mundial.

Autoevaluación

Para obtener código fuente a partir de toda la información necesaria del problema:

- Se elige el Lenguaje de Programación más adecuado y se codifica directamente.
- Se codifica y después se elige el Lenguaje de Programación más adecuado.
- Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.

Incorrecta. La codificación directa nos llevará mucho tiempo y tendremos demasiados errores.

No es correcta. Antes de programar tenemos que saber qué Lenguajes de Programación vamos a utilizar.

Muy bien. El diseño del algoritmo (los pasos a seguir) nos ayudará a que la codificación posterior se realice más rápidamente y tenga menos errores.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

Autoevaluación

Relaciona los tipos de código con su característica más relevante, escribiendo el número asociado a la característica en el hueco correspondiente.

Ejercicio de relacionar

Tipo de código.	Relación.	Características.
-----------------	-----------	------------------

Tipo de código.	Relación.	Características.
Código Fuente	<input type="checkbox"/>	1. Escrito en Lenguaje Máquina pero no ejecutable.
Código Objeto	<input type="checkbox"/>	2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable.
Código Ejecutable	<input type="checkbox"/>	3. Escrito en Lenguaje Máquina y directamente ejecutable.

Enviar

El código fuente escrito en algún lenguaje de programación de alto nivel, el objeto escrito en lenguaje máquina sin ser ejecutable y el código ejecutable, escrito también en lenguaje máquina y ya sí ejecutable por el ordenador, son las distintas fases por donde pasan nuestros programas.

Caso práctico

Juan y María ya han decidido el Lenguajes de Programación que van a utilizar.

Saben que el programa que realicen pasará por varias fases antes de ser implementado en los equipos del cliente. Todas esas fases van a producir transformaciones en el código. ¿Qué características irá adoptando el código a medida que avanza por el proceso de codificación?

7.2.1.- Lenguajes de programación estructurados.

Aunque los requerimientos actuales de software son bastante más complejos de lo que la técnica de programación estructurada es capaz, es necesario por lo menos conocer las bases de los Lenguajes de Programación estructurados, ya que a partir de ellos se evolucionó hasta otros lenguajes y técnicas más completas (orientada a eventos u objetos) que son las que se usan actualmente.

La programación estructurada se define como una técnica para escribir lenguajes de programación que permite sólo el uso de tres tipos de sentencias o estructuras de control:

- Sentencias secuenciales.
- Sentencias selectivas (condicionales).
- Sentencias repetitivas (iteraciones o bucles).

Los lenguajes de programación que se basan en la programación estructurada reciben el nombre de lenguajes de programación estructurados.

La programación estructurada fue de gran éxito por su sencillez a la hora de construir y leer programas.

La Programación estructurada evolucionó hacia la Programación modular, que divide el programa en trozos de código llamados módulos (siguiendo la conocida técnica "divide y vencerás") con una funcionalidad concreta, que podrán ser reutilizables. A su vez, luego triunfaron los lenguajes orientados a objetos y de ahí a la programación visual (siempre es más sencillo programar gráficamente que en código, ¿no crees?).

VENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA

- Los programas son fáciles de leer, sencillos y rápidos.
- El mantenimiento de los programas es sencillo.
- La estructura del programa es sencilla y clara.

INCONVENIENTES

- Todo el programa se concentra en un único bloque (si se hace demasiado grande es difícil manejarlo).
- No permite reutilización eficaz de código, ya que todo va "en uno". Es por esto que a la programación estructurada le sustituyó la programación modular, donde los programas se codifican por módulos y bloques, permitiendo mayor funcionalidad.

La Programación estructurada evolucionó hacia la Programación modular, que divide el programa en trozos de código llamados módulos con una funcionalidad concreta, que podrán ser reutilizables.

7.2.2.- Lenguajes de programación orientados a objetos.

Después de comprender que la programación estructurada no es útil cuando los programas se hacen muy largos, es necesaria otra técnica de programación que solucione este inconveniente. Nace así la Programación Orientada a Objetos (en adelante, P.O.O.).

Los lenguajes de programación orientados a objetos tratan a los programas no como un conjunto ordenado de instrucciones (tal como sucedía en la programación estructurada) sino como un conjunto de objetos que colaboran entre ellos para realizar acciones.

En la P.O.O. los programas se componen de objetos independientes entre sí que colaboran para realizar acciones.

Su primera desventaja es clara: no es una programación tan intuitiva como la estructurada.

Algunas de sus características principales son:

- Se define clase como una colección de objetos con características similares. Si en nuestro programa interactúan un conjunto de personas, la clase a implementar será Persona y los objetos serán personas en particular (Noa, Valeria, Mila ...).
- Las clases definen una serie de atributos, que caracterizan a cada objeto persona. Por ejemplo, el atributo edad tendrá el valor 4 para Noa, 7 para Valeria
- Las clases definen una serie de métodos que corresponden a las acciones que pueden llevar a cabo los objetos. Por ejemplo, en la clase Persona se puede definir el método Saludar, que puede ser utilizado por Mila para dar los buenos días. Estos métodos contendrán el código que da respuesta a las acciones que implementan.
- Mediante llamadas a los métodos, unos objetos se comunican con otros produciéndose un cambio de estado de los mismos. Esto se denomina envío de mensajes.
- El código es más reutilizable.
- Si hay algún error, es más fácil de localizar y depurar en una clase que en un programa entero.

Algunos términos relativos a la programación orientada a objetos son:

Clase
Objeto
Mensaje
Método
Evento (sistema)
Propiedad o atributo
Estado interno
Abstracción
Encapsulamiento
Poliformismo
Herencia

Algunos lenguajes orientados a objetos son: **Ada, C++ (C plus plus), VB.NET, C#, Java, PowerBuilder**, etc.

Nota: otros módulos del ciclo te permitirán conocer la programación orientada a objetos en profundidad.

8.- Máquinas virtuales.

Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

Esta capa de software desempeña un papel muy importante en el funcionamiento de los lenguajes de programación, tanto compilados como interpretados.

Con el uso de máquinas virtuales podremos desarrollar y ejecutar una aplicación sobre cualquier equipo, independientemente de las características concretas de los componentes físicos instalados. Esto garantiza la portabilidad de las aplicaciones.

Las funciones principales de una máquina virtual son las siguientes:

Conseguir que las aplicaciones sean portables.

Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.

Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos.

Cumplimiento de las normas de seguridad de las aplicaciones.

CARACTERÍSTICAS DE LA MÁQUINA VIRTUAL

- Cuando el código fuente se compila se obtiene código objeto (bytecode, código intermedio).
- Para ejecutarlo en cualquier máquina se requiere tener independencia respecto al hardware concreto que se vaya a utilizar.
- Para ello, la máquina virtual aísla la aplicación de los detalles físicos del equipo en cuestión.
- Funciona como una capa de software de bajo nivel y actúa como puente entre el bytecode de la aplicación y los dispositivos físicos del sistema.
- La Máquina Virtual verifica todo el bytecode antes de ejecutarlo.
- La Máquina Virtual protege direcciones de memoria.

La máquina virtual actúa de puente entre la aplicación y el hardware concreto del equipo donde se instale.

8.1.- Entornos de ejecución.

Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. En ocasiones pertenece al propio sistema operativo, pero también se puede instalar como software independiente que funcionará por debajo de la aplicación.

Es decir, es un conjunto de utilidades que permiten la ejecución de programas.

Se denomina runtime al tiempo que tarda un programa en ejecutarse en la computadora.

Durante la ejecución, los entornos se encargarán de:

Configurar la memoria principal disponible en el sistema.

Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados. Considerando que las bibliotecas son el conjunto de subprogramas que sirven para desarrollar o comunicar componentes software pero que ya existen previamente y los subprogramas serán aquellos que hemos creado a propósito para el programa.

Depurar los programas: comprobar la existencia (o no existencia) de errores semánticos del lenguaje (los sintácticos ya se detectaron en la compilación).

Funcionamiento del entorno de ejecución:

El Entorno de Ejecución está formado por la máquina virtual y los API's (bibliotecas de clases estándar, necesarias para que la aplicación, escrita en algún Lenguaje de Programación pueda ser ejecutada). Estos dos componentes se suelen distribuir conjuntamente, porque necesitan ser compatibles entre sí.

El entorno funciona como intermediario entre el lenguaje fuente y el sistema operativo, y consigue ejecutar aplicaciones.

Sin embargo, si lo que queremos es desarrollar nuevas aplicaciones, no es suficiente con el entorno de ejecución.

Adelantándonos a lo que veremos en la próxima unidad, para desarrollar aplicaciones necesitamos algo más. Ese "algo más" se llama entorno de desarrollo.

Autoevaluación

Señala la afirmación falsa respecto de los entornos de ejecución:

- Su principal utilidad es la de permitir el desarrollo rápido de aplicaciones.
- Actúa como mediador entre el sistema operativo y el código fuente.
- Es el conjunto de la máquina virtual y bibliotecas necesarias para la ejecución.

Muy bien, lo has entendido perfectamente.

Incorrecto, eso es verdad.

No es correcto, eso es verdad.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

8.2.- Java runtime environment.

En esta sección se va a explicar el funcionamiento, instalación, configuración y primeros pasos del Runtime Environment del lenguaje Java (se hace extensible a los demás lenguajes de programación).

Concepto.

Se denomina JRE al Java Runtime Environment (entorno en tiempo de ejecución Java).

El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma.

Componentes.

JRE está formado por:

Una Máquina virtual Java (JMV o JVM si consideramos las siglas en inglés), que es el programa que interpreta el código de la aplicación escrita en Java.

Bibliotecas de clase estándar que implementan el API de Java.

Las dos: JMV y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.

Lo primero es descargarnos el programa JRE. (Java2 Runtime Environment JRE 1.6.0.21). Java es software libre, por lo que podemos descargarnos la aplicación libremente.

Una vez descargado, comienza el proceso de instalación, siguiendo los pasos del asistente.

Debes conocer

El proceso de descarga, instalación y configuración del entorno de ejecución de programas. En el siguiente enlace, se explican los pasos para hacerlo bajo el sistema operativo Linux.

[Instalación y configuración del JRE de Java.](#)

Anexo I.- Sentencias de control de la programación estructurada.

SENTENCIAS SECUENCIALES

Las sentencias secuenciales son aquellas que se ejecutan una detrás de la otra, según el orden en que hayan sido escritas.

Ejemplo en lenguaje C:

```
printf ("declaración de variables");
int numero_entero;
espacio=espacio_inicio + velocidad*t tiempo;
```

SENTENCIAS SELECTIVAS (CONDICIONALES)

Son aquellas en las que se evalúa una condición. Si el resultado de la condición es verdad se ejecutan una serie de acción o acciones y si es falso se ejecutan otras.

if → señala la condición que se va a evaluar

then → Todas las acciones que se encuentren tras esta palabra reservada se ejecutarán si la condición del **if** es cierta (en C, se omite esta palabra).

else → Todas las acciones que se encuentren tras esta otra palabra reservada se ejecutarán si la condición de **if** es falsa.

Ejemplo en lenguaje C:

```
if (a >= b)
c= a-b;
else
c=a+b;
```

SENTENCIAS REPETITIVAS (ITERACIONES O BUCLES)

Un bucle iterativo de una serie de acciones harán que éstas se repitan mientras o hasta que una determinada condición sea falsa (o verdadera).

while → marca el comienzo del bucle y va seguido de la condición de parada del mismo.

do → a partir de esta palabra reservada, se encontrarán todas las acciones a ejecutar mientras se ejecute el bucle (en C, se omite esta palabra).

done → marca el fin de las acciones que se van a repetir mientras estemos dentro del bucle (en C, se omite esta palabra).

Ejemplo en lenguaje C:

```
int num;
num = 0;
while (num<=10) { printf("Repetición numero %d\n", num);
num = num + 1;
};
```

Anexo II.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: Scott Schram. Licencia: CC by 2.0. Procedencia: http://www.flickr.com/photos/schram/21742249/		Autoría: fsse8info. Licencia: CC by -SA 2.0. Procedencia: http://www.flickr.com/photos/fsse-info/3276664015/
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Verónica Cabrerizo. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Verónica Cabrerizo. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.

	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.

	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Francisco Palacios. Licencia: CC by -NC-ND 2.0. Procedencia: http://www.flickr.com/photos/wizard_/3303810302/	Autoría: Verónica Cabrerizo. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.	Autoría: Ministerio de Educación. Licencia: Uso Educativo-nc. Procedencia: Elaboración propia.

Instalación y uso de entornos de desarrollo.

Caso práctico



Tras el éxito del anterior proyecto, en BK están recibiendo más peticiones de creación de software que nunca. Ana y Antonio, que ya hace unas semanas que están estudiando el Ciclo de Diseño de Aplicaciones Multiplataforma, piensan que este es un buen momento para participar activamente en los proyectos, pues a sus compañeros no les vendría nada mal un poco de ayuda.

Ada confía en ellos, pero aún es pronto. Por lo menos, ya conocen las fases por las que tiene que pasar todo el desarrollo de aplicaciones, pero eso no será suficiente.

María, sin embargo, no piensa lo mismo y decide darles una oportunidad trabajando en la fase de codificación de un nuevo proyecto de la empresa.

Ana se muestra muy ilusionada y no piensa desperdiciar esta gran oportunidad. Sabe que tiene a su disposición los llamados entornos de desarrollo que le facilitarán su futura tarea.

¿Cómo influirá el conocimiento de esta herramienta en el futuro de Ana y Antonio? A través de esta unidad, veremos si nuestros amigos van logrando ganarse un puesto en la empresa, y de paso, la confianza de Ada.

La fase de codificación es compleja, pero Ana y Antonio están aprendiendo a dominar los llamados entornos integrados de desarrollo de software.



[Ministerio de Educación y Formación Profesional.](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

[Siguiente »](#)

1.- Concepto de entorno de desarrollo. Evolución histórica.

Caso práctico



Todos en la empresa están sorprendidos del entusiasmo de Ana ante los nuevos proyectos que BK programación tiene por delante. Juan, que acabó el Ciclo Superior de Desarrollo de Aplicaciones Informáticas (DAI) hace algunos años, se muestra inquieto porque es consciente de que en sólo unos cuatro años han salido muchas herramientas nuevas en el mercado y necesita reciclarse. Escucha a Ana decir que está estudiando los entornos de desarrollo.

—Yo también debería ponerme al día —piensa Juan.

En la unidad 1 se trataron las fases a seguir en un proceso de desarrollo de software.

La fase de codificación se puede llevar a cabo casi exclusivamente con un editor de texto y un compilador. Pero prácticamente la totalidad de programadores, terminan haciendo uso de algún entorno de desarrollo integrado para crear aplicaciones.

Un entorno integrado de desarrollo (**IDE**), es un tipo de software compuesto por un conjunto de herramientas de programación.

En concreto, el **IDE** entre otras aplicaciones se compone de:

- Editor de código de programación.
- Accesos al compilador desde botones u opciones de menu.
- Acceso a la ejecución del programa desde botones u opciones de menu.
- Depurador.
- Constructor de interfaz gráfico.

Los primeros entornos de desarrollo integrados nacieron a principios de los años 70, y se popularizaron en la década de los 90.

Tienen el objetivo de ganar fiabilidad y tiempo en los proyectos de software. Proporcionan al programador una serie de componentes con la misma interfaz gráfica, con la consiguiente comodidad, aumento de eficiencia y reducción de tiempo de codificación.

Normalmente, un **IDE** está dedicado a un determinado lenguaje de programación. No obstante, las últimas versiones de los **IDE** tienden a ser compatibles con varios lenguajes (por ejemplo, **Eclipse**, **NetBeans**, **Microsoft Visual Studio**) mediante la instalación de plugins adicionales.

En este tema, nuestro interés se centra en conocer los entornos de desarrollo, los tipos (en función de su licencia y del lenguaje de programación hacia el cual están enfocados). Veremos cómo se configuran y cómo se generan ejecutables, haciendo uso de sus componentes y herramientas.

Reflexiona

Según datos, casi todas las personas que empiezan a programar utilizan un editor simple de textos y un compilador-depurador instalado en su equipo. Sin embargo, prácticamente todas acaban utilizando un entorno de desarrollo.

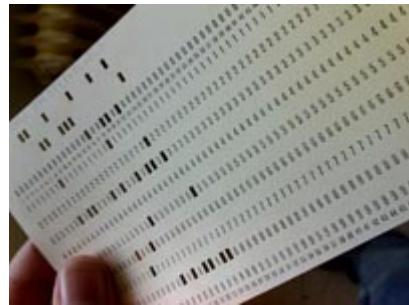
[« Anterior](#) [Siguiente »](#)

1.1.- Evolución Histórica.

En las décadas de utilización de la tarjeta perforada como sistema de almacenamiento el concepto de **Entorno de Desarrollo Integrado** sencillamente no tenía sentido.

Los programas estaban escritos con diagramas de flujo y entraban al sistema a través de las **tarjetas perforadas**. Posteriormente, eran compilados.

El primer lenguaje de programación que utilizó un IDE fue el **BASIC** (que fue el primero en abandonar también las tarjetas perforadas o las cintas de papel).



Este primer IDE estaba basado en consola de comandos exclusivamente (normal por otro lado, si tenemos en cuenta que hasta la década de los 90 no entran en el mercado los sistemas operativos con interfaz gráfica). Sin embargo, el uso que hace de la gestión de archivos, compilación y depuración; es perfectamente compatible con los IDE actuales.

A nivel popular, el primer IDE puede considerarse que fue el IDE llamado **Maestro**. Nació a principios de los 70 y fue instalado por unos 22.000 programadores en todo el mundo. Lideró este campo durante los años 70 y 80.

El uso de los entornos integrados de desarrollo se ratifica y afianza en los 90 y hoy en día contamos con infinidad de IDE, tanto de licencia libre como no.

Tipos de entornos de desarrollo más relevantes en la actualidad.

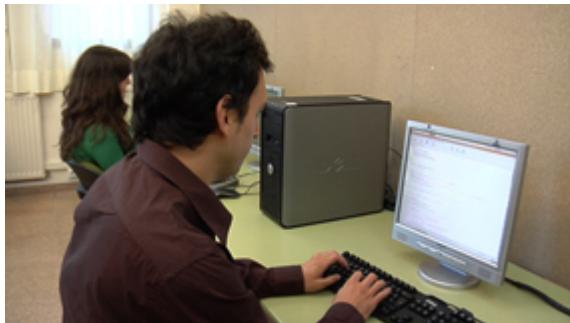
Entorno de desarrollo	Lenguajes que soporta	Tipo de licencia
NetBeans.	C/C++, Java, JavaScript, PHP, Python.	De uso público.
Eclipse.	Ada, C/C++, Java, JavaScript, PHP.	De uso público.
Microsoft Visual Studio.	Basic, C/C++, C#.	Propietario.
C++ Builder.	C/C++.	Propietario.
JBuilder.	Java.	Propietario.

No hay unos entornos de desarrollo más importantes que otros. La elección del IDE más adecuado dependerá del lenguaje de programación que vayamos a utilizar para la codificación de las aplicaciones y el tipo de licencia con la que queramos trabajar.

[« Anterior](#) [Siguiente »](#)

2.- Funciones de un entorno de desarrollo.

Caso práctico



Juan, que asume por fin su desconocimiento, habla con Ana para que le pase sus apuntes de entornos de desarrollo. Ésta se muestra encantada, y le anima a matricularse al ciclo de Desarrollo de Aplicaciones Multiplataforma (DAM) a distancia. Juan se muestra reacio (ya ha estudiado el ciclo y durante cuatro años ha cumplido con éxito en la empresa). Pero piensa que quizás debería reciclarse si no quiere quedarse

atrás en los proyectos. Juan aprendió a programar usando un editor simple de textos, ¿qué ventajas tendrá programando con un IDE?

Como sabemos, los entornos de desarrollo están compuestos por una serie de herramientas software de programación, necesarias para la consecución de sus objetivos. Estas herramientas son:

- ✓ Un editor de código fuente.
- ✓ Un compilador y/o un intérprete.
- ✓ Automatización de generación de herramientas.
- ✓ Un depurador.

Las funciones de los IDE son:

- ✓ Editor de código: coloración de la sintaxis.
- ✓ Auto-completado de código, atributos y métodos de clases.
- ✓ Identificación automática de código.
- ✓ Herramientas de concepción visual para crear y manipular componentes visuales.
- ✓ Asistentes y utilidades de gestión y generación de código.
- ✓ Organización de los archivos fuente en unas carpetas y compilados a otras.
- ✓ Compilación de proyectos complejos en un solo paso.
- ✓ Control de versiones: tener un único almacén de archivos compartido por todos los colaboradores de un proyecto. Ante un error, mecanismo de auto-recuperación a un estado anterior estable.
- ✓ Soporta cambios de varios usuarios de manera simultánea.
- ✓ Generador de documentación integrado.
- ✓ Detección de errores de sintaxis en tiempo real.



Otras funciones importantes son:

- ✓ Ofrece refactorización de código: cambios menores en el código que facilitan su legibilidad sin alterar su funcionalidad (por ejemplo cambiar el nombre a una variable).
- ✓ Permite introducir automáticamente tabulaciones y espaciados para aumentar la legibilidad.
- ✓ Depuración: seguimiento de variables, puntos de ruptura y mensajes de error del intérprete.
- ✓ Aumento de funcionalidades a través de la gestión de sus módulos y plugins.
- ✓ Administración de las interfaces de usuario (menús y barras de herramientas).

- ✓ Administración de las configuraciones del usuario.
- ✓ Empaquetar software para su posterior despliegue o instalación en el entorno de ejecución.

Autoevaluación

Un entorno integrado de desarrollo está compuesto por:

- Editor de código y traductor.
- Editor de código, compilador e interfaz de comandos.
- Editor de código, compilador, intérprete, depurador e interfaz gráfica.
- Interfaz gráfica, editor de código y depurador.

Incorrecta, se compone de más herramientas.

No es correcta porque la interfaz es gráfica.

Muy bien. Esa es la idea.

No es del todo correcta: faltaría el traductor de código (compilador e intérprete).

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

[« Anterior](#) [Siguiente »](#)

3.- Entornos integrados libres y propietarios.

Caso práctico



Juan ha buscado por Internet distintos entornos de desarrollo para aplicarlos en la fase de codificación.

—Cuidado —le dice Ada—. Ya sabes que es de vital importancia el tema de la Licencia de Software. Hay Entornos de desarrollo de licencia libre y otros no, y este aspecto es fundamental ni no queremos tener problemas.

Entornos Integrados Libres

Entornos Integrados libres son aquellos con licencia de uso público. No hay que pagar por ellos, y aunque los más conocidos y utilizados son **Eclipse** y **NetBeans**, hay bastantes más.

Por otra parte, los entornos integrados de desarrollo propietarios necesitan licencia. No son free software, hay que pagar por ellos. El más conocido y utilizado es **Microsoft Visual Studio**, desarrollado por **Microsoft** (sólo disponible en plataformas **Windows**).

La mayor parte de ellos, aunque no todos, están disponibles en diferentes plataformas.

Entornos de desarrollo libres más relevantes en la actualidad

IDE	Algunos lenguajes que soporta	URL
Eclipse	Ada, C/C++, Java, JavaScript, PHP	https://www.eclipse.org/
NetBeans	C/C++, JavaScript, PHP, HTML5 ...	https://netbeans.org/
CodeLite	C/C++, PHP, Node.js	https://codelite.org
JDeveloper	Java, HTML, XML, SQL, PL/SQL, Javascript, PHP, UML ...	http://www.oracle.com/technetwork/developer-tools/jdev/overview/index.html

IntelliJ	Java, Groovy, Perl, Scala, XML/XSL, Python ...	https://www.jetbrains.com/idea/
Microsoft Visual Studio	C#, Visual Basic, F#, C++, HTML, JavaScript, TypeScript, Python, ...	https://visualstudio.microsoft.com/es/vs/community/ https://code.visualstudio.com/

Para saber más

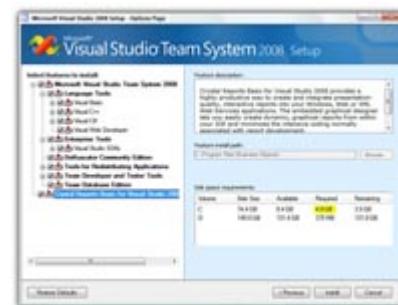
En el siguiente enlace encontrarás un documento muy interesante, en inglés, donde se detallan todos los entornos de desarrollo existentes en la actualidad con todas sus características: licencias, sistemas operativos donde pueden ser instalados y configurados, lenguajes que soporta, desarrolladores y última versión estable.

[Entornos de desarrollo actuales.](#)

Entornos Integrados Propietarios

Son aquellos entornos integrados de desarrollo que necesitan licencia. No son free software, hay que pagar por ellos.

El más conocido y utilizado es Microsoft Visual Studio, que usa el framework .NET y es desarrollado por Microsoft.



Entornos de desarrollo propietarios más relevantes en la actualidad

IDE	Algunos lenguajes que soporta	URL
Microsoft Visual Studio	C++, C#, Visual Basic ...	https://visualstudio.microsoft.com/es/
C++ Builder	C++	https://www.embarcadero.com/es/
IntelliJ	Java, Groovy, Perl, Scala, ML/XSL, Python, Ruby, Sql ...	https://www.jetbrains.com/idea/
QtCreator	C++ con framework QT	https://www.qt.io/

Autoevaluación

Relaciona los siguientes entornos de desarrollo con sus características, escribiendo el número asociado a la característica en el hueco correspondiente.

Ejercicio de relacionar

Entorno de desarrollo.	Relación	Características.
Microsoft Visual Studio.	<input type="checkbox"/>	1. Libre. Soporta C/C++, Java, PHP, Javascript, Python.
NetBeans.	<input type="checkbox"/>	2. Propietario. Soporta Basic, C/C++, C#.
C++ Builder.	<input type="checkbox"/>	3. Propietario. Soporta C/C++.

Enviar

En la elección del entorno de desarrollo más adecuado para desarrollar un proyecto de software influye el tipo de licencia del entorno y los lenguajes de programación que soporta.

« Anterior Siguiente »

4.- Estructura de entornos de desarrollo.

Caso práctico



Juan aprendió a programar utilizando un editor de textos, un compilador y un depurador. Todas estas herramientas se instalaban de forma independiente. A Ana le cuesta creer que los programadores tuvieran que buscar estas herramientas e instalarlas por separado. —En un entorno se integran todas estas cosas y muchas más, y sin salir del mismo puedes programar en varios lenguajes y puedes documentar y.... —Ya lo veo, —le replica Juan—. ¿Cuántos componentes tiene el entorno en total?

Los entornos de desarrollo, ya sean libres o propietarios, están formados por una serie de componentes software que determinan sus funciones.

Estos componentes son:



Componentes	Funciones
Editor de textos.	Resaltado y coloreado de la sintaxis del código. Funciones de completado automático de código. Inserción automáticamente paréntesis, corchetes, tabulaciones y espaciados. Ayuda y listado de parámetros de funciones y métodos de clase
Compilador/intérprete.	Detección de errores de sintaxis en tiempo real.
Depurador.	Ejecución del programa paso a paso, definición de puntos de ruptura y seguimiento de variables. Opción de depurar en servidores remotos.
Generador automático de herramientas.	Herramientas para la visualización, creación y manipulación de componentes visuales y todo un arsenal de asistentes y utilidades de gestión y generación código.

Interfaz gráfica.

Brinda la oportunidad de programar en varios lenguajes con un mismo IDE. Es una interfaz agradable que puede acceder a innumerables bibliotecas y plugins, aumentando las opciones de nuestros programas.

Para saber más

En el siguiente enlace accederás a una página web donde se detallan todos los componentes del entorno de desarrollo, junto con sus funciones.

[Estructura de Entornos de Desarrollo](#)

[« Anterior](#) [Siguiente »](#)

5.- Instalación de entornos integrados de desarrollo.

Caso práctico



Juan está decidido a aprender a usar un entorno de desarrollo. Después de documentarse, piensa que lo idóneo es trabajar con un IDE libre. Además, el tema del sistema operativo que soporta es importante. Juan quiere trabajar bajo Linux, y se decide por el entorno NetBeans. Ahora bien, ¿Qué hay que hacer para instalarlo?

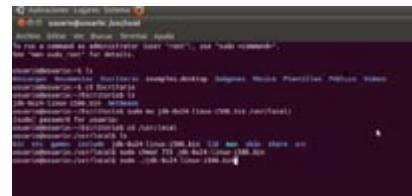
Se va a describir cómo instalar **Netbeans**. IDE con mucha presencia en el mercado y de libre distribución. Dispone de distribuciones bajo diferentes plataformas.

Para compilar los programas en **Java** desde este IDE que se verán en las próximas secciones, habrá que haber instalado primero el paquete **JDK de Java**.

[« Anterior](#) [Siguiente »](#)

5.1.- Instalación de JDK.

La instalación del IDE NetBeans, ya sea en Linux, Windows o Mac OS X, requiere la instalación previa del **JDK** compatible con la versión de NetBeans que se quiera instalar.



JDK son las siglas de **Java Development Kit**: Kit de desarrollo de Java. Consiste en la plataforma del entorno, imprescindible para que éste pueda ser instalado y ejecutado.

En el caso del lenguaje **Java**, se indicaba en temas anteriores que tras la compilación del código fuente se obtiene otro llamado **bytecode**. Para que el **bytecode** pueda ser interpretado, el equipo deberá tener instalado el **JRE (Java Runtime Environment)**, definido en wikipedia como sigue:

JRE es un conjunto de utilidades que permite la ejecución de programas [Java](#).

En su forma más simple, el entorno en tiempo de ejecución de Java está conformado por una **Máquina Virtual de Java o JVM**, un conjunto de bibliotecas Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada. El JRE actúa como un "intermediario" entre el sistema operativo y Java.

La **JVM** es el programa que ejecuta el código Java previamente compilado (bytecode) mientras que las librerías de clases estándar son las que implementan el API de Java. Ambas JVM y **API** deben ser consistentes entre sí, de ahí que sean distribuidas de modo conjunto.

Un usuario sólo necesita el JRE para ejecutar las aplicaciones desarrolladas en lenguaje Java, mientras que para desarrollar nuevas aplicaciones en dicho lenguaje es necesario un entorno de desarrollo, denominado JDK, que además del JRE (mínimo imprescindible) incluye, entre otros, un compilador para Java.

El **JRE** es desarrollado y distribuido de forma gratuita por **Oracle**.

Pero si nuestra voluntad es convertirnos en desarrolladores de código **Java**, no será suficiente el **JRE**, tendremos que instalar el **JDK (Java Development Kit)**, también distribuido por Oracle y que wikipedia define como:

Java Development Kit (JDK) es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red.

Los programas más importantes que se incluyen

- **appletviewer.exe**: es un visor de [applets](#) para generar sus vistas previas, ya que un applet carece de método main y no se puede ejecutar con el programa java.
- **javac.exe**: es el compilador de Java.
- **java.exe**: es el masterescuela (intérprete) de Java.
- **javadoc.exe**: genera la documentación de las clases Java de un programa.

Para instalar el JDK accede a la siguiente dirección:

<https://www.oracle.com/technetwork/es/java/javase/overview/index.html>

En el apartado de Updates verás las distintas versiones del JDK de Java SE (Standard Edition). Un número más alto indica que es más reciente. Elige una de ellas y descarga el fichero adecuado para tu plataforma. En la página de descarga también vienen las instrucciones de instalación por si tuvieses algún problema.

Apunta la ruta donde se instala el JDK pues lo necesitarás más adelante.

Configuración de las variables de entorno

El **JDK** podrá ser utilizado por diversas aplicaciones entre las que se encuentran los **IDEs**. Al ser la ubicación del **JDK** en el sistema de archivos configurable durante la instalación, las aplicaciones que hacen uso del mismo no saben donde localizarlo.

Para resolverlo, se definen variables de entorno, cuyo nombre será de conocimiento público y por tanto común para cualquier aplicación que quiera utilizarlas. En particular, las propuestas a continuación tienen como cometido informar de las rutas elegidas en la instalación para el **JDK** y facilitar el acceso a sus ejecutables.

Configuración de las variables de entorno en Linux (con bash)

Desde un terminal en **Linux** se dan de alta/modifican las variables de entorno **JAVA_HOME** y **PATH**.

Acción	Orden en consola linux
Abrir un terminal Linux .	
Ganar privilegios de administrador.	sudo su + Contraseña
Con un editor de texto (por ejemplo nano), acceder al fichero /etc/bash.bashrc.	nano /etc/bash.bashrc
Introducir las variables de entorno: JAVA_HOME=/usr/java/jdk1.8.0_171 PATH=\$PATH:/usr/java/jdk1.8.0_171/bin En este caso, la ruta del jdk en JAVA_HOME y también añadimos la ruta de los ejecutables en PATH	

```
root@debian:/etc# more /etc/bash.bashrc
JAVA_HOME=/usr/java/jdk1.8.0_171
PATH=$PATH:/usr/java/jdk1.8.0_171/bin
```

Tras reiniciar, se puede comprobar que han quedado activadas desde terminal con los comandos:

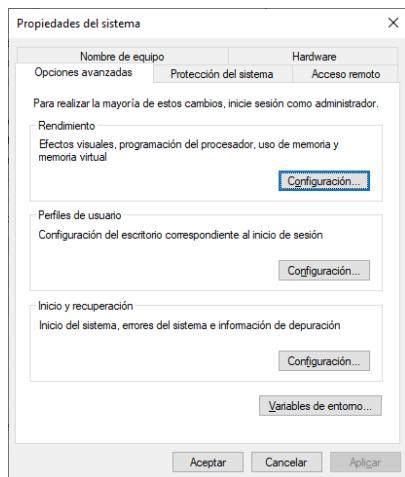
- echo \$PATH
- echo \$JAVA_HOME

Configuración de las variables de entorno en Windows (ejemplo en Windows 10).

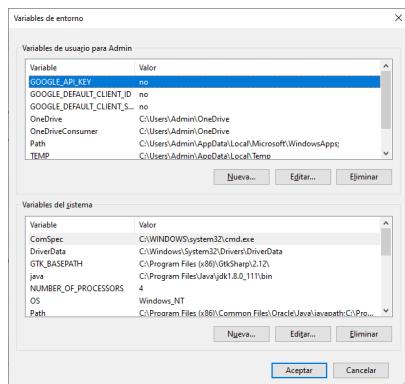
1. Ir a configuración de Windows.
2. Poner en el buscador: "configuración avanzada del sistema":



Se visualizará esta ventana:



A continuación, se pulsa en "variables de entorno"



Después elegimos Nueva (en variables de Sistema) e introducimos los datos de la variable de entorno, su nombre: JAVA_HOME y su valor (la ruta donde está instalado el jdk). En este ejemplo es un j . Después Aceptar.

Como la variable PATH ya existe, seleccionamos la variable PATH y elegimos Editar. Añadimos la ruta del directorio bin de nuestro jdk. Ponemos un punto y coma y luego %JAVA_HOME%\bin. Para finalizar, Aceptar.

Configuración de las variables de entorno en MAC OS

En este enlace puedes ver un ejemplo:

https://sintaxispragmatica.wordpress.com/2014/04/07/establecer-la-variable-java_home-en-mac-osx/

Habrá que editar también la variable PATH:

<https://professor-falken.com/mac/como-anadir-una-nueva-ruta-a-la-variable-path-en-tu-mac/>

Autoevaluación

En tu opinión, ¿Por qué crees que la instalación del JDK sólo la puede realizar el root del sistema?

- Porque se trata de un archivo binario de sistema.
- Porque ningún archivo puede ser ejecutado por un usuario que no sea el root.
- Porque estamos trabajando en la terminal del sistema.

Efectivamente. Es un archivo con extensión .bin y sólo puede ser manipulado por el root.

No. Los ejecutables normales pueden ser ejecutados por cualquiera que tenga permisos para ello.

Incorrecta. La terminal es una forma de trabajar cuando tenemos que modificar permisos y realizar acciones que no podemos desde la interfaz gráfica.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

« Anterior Siguiente »

5.2.- Instalación de entornos de desarrollo.

Una vez tenemos instalado el JDK en nuestro equipo, ya tenemos preparado el contexto en el que se instalará el entorno NetBeans.

Recuerda que para trabajar con un IDE que trabaja con lenguaje Java, necesitarás tener el JDK instalado.

[« Anterior](#) [Siguiente »](#)

5.2.1.- Instalación de NetBeans.

Caso práctico



Juan ya ha instalado el JDK.

—Uff, me ha costado un poco... —le comenta a Ana.
—Hace tiempo que no trabajaba en la terminal de Linux y se me habían olvidado algunas órdenes básicas. Ana le comenta que ya tiene el equipo preparado para instalar NetBeans. Decide pasarle los apuntes del ciclo a distancia para que Juan no tenga que perder mucho tiempo buscando los comandos necesarios.

En el caso del IDE Netbeans, instalaremos la última versión disponible en la siguiente dirección:

<https://netbeans.apache.org/download/index.html>

Elige el adecuado para tu plataforma. Descarga el instalador y ejecútalo. No es necesario configurar nada. Al instalarse puedes elegir dónde instalarlo y decirle dónde está el jdk (lo detectará si lo tienes instalado).

En el "Anexo III.- Instalación JDK y NetBeans" se explica detalladamente como instalar el JDK y el NetBeans conjuntamente.

[« Anterior](#) [Siguiente »](#)

5.2.2.- Instalación de Eclipse.

Recuerda que necesitarás tener el JDK instalado.

Descarga de Eclipse:

<https://www.eclipse.org/downloads/packages/>

Elige el Eclipse IDE for Java developers para tu plataforma. Si tienes problemas con la última versión de Eclipse, en esa misma página puedes encontrar versiones anteriores que quizá te sirvan.

Te habrás descargado un zip. Descomprime el fichero y ejecuta eclipse con el fichero ejecutable llamado eclipse. Al arrancar, detecta automáticamente si el JDK está instalado. A continuación, deberás elegir el **Workspace**.

Selección del Workspace de trabajo.

Eclipse organiza los proyectos en espacios de trabajo. Se trata de poner en común diferentes bloques de código bajo algún criterio en particular (funcionalidades comunes, mismo cliente, clases desarrolladas por un mismo programador ...).

Eclipse asocia cada **Workspace** con un directorio en el sistema de archivos, a partir del cual irán "colgando" nuevos directorios, uno por proyecto creado.

Al iniciar **Eclipse** solicita la ubicación del **Workspace**. Elige la que consideres oportuna.

Una vez haya arrancado puedes configurar el idioma si lo deseas

Configuración del idioma.

Para poner **Eclipse** en español se va a utilizar la solución propuesta en el proyecto **babel** de **Eclipse**. Los pasos a seguir podrás encontrarlos en la url:

<https://www.eclipse.org/babel/downloads.php>

Bajo la etiqueta de "**Installing the language packs**", se dan las siguientes instrucciones.

- En **Eclipse** accede al asistente Help/Install New Softwaree
- Añade el repositorio **Babel de Eclipse**.

<https://download.eclipse.org/technology/babel/update-site/R0.17.0/2019-06/>

No te impacientes, la carga se toma un poco tiempo mostrando el mensaje Pending

- Una vez cargado, seleccionar los paquetes **babel para Eclipse in spanish** y pulsar next
- Reinicia **Eclipse**.

[« Anterior](#) [Siguiente »](#)

6.- Configuración y personalización de entornos de desarrollo.

Caso práctico



Juan está consternado. NetBeans parece albergar tanta información que no sabe por donde empezar. Le gustaría personalizar la configuración de su primer proyecto en el IDE (que va a ser un aplicación de Java). ¿Cómo lo hace? ¿Qué parámetros puede configurar?

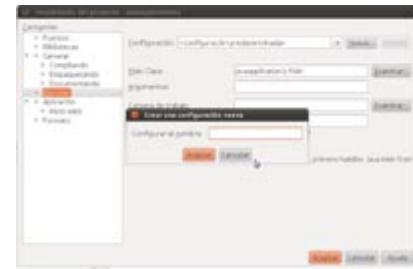
Una vez tenemos instalado nuestro entorno de desarrollo podemos acceder a personalizar su configuración.

Al abrir un proyecto existente, o bien crear un nuevo proyecto, seleccionaremos un desplegable con el nombre de "configuración" desde el que podremos personalizar distintas opciones del proyecto.

Podemos personalizar la configuración del entorno sólo para el proyecto actual, o bien para todos los proyectos, presentes y futuros.

Parámetros configurables del entorno:

- ✓ Carpeta o carpetas donde se alojarán todos los archivos de los proyectos (es importante la determinación de este parámetro, para tener una estructura de archivos ordenada).
- ✓ Carpetas de almacenamiento de paquetes fuente y paquetes prueba.
- ✓ Administración de la plataforma del entorno de desarrollo.
- ✓ Opciones de la compilación de los programas: compilar al grabar, generar información de depuración.
- ✓ Opciones de empaquetado de la aplicación: nombre del archivo empaquetado (con extensión .jar, que es la extensión característica de este tipo de archivos empaquetados) y momento del empaquetado.
- ✓ Opciones de generación de documentación asociada al proyecto.
- ✓ Descripción de los proyectos, para una mejor localización de los mismos.
- ✓ Opciones globales de formato del editor: número de espaciados en las sangrías, color de errores de sintaxis, color de etiquetas, opción de autocompletado de código, propuestas de insertar automáticamente código.
- ✓ Opciones de combinación de teclas en teclado.
- ✓ Etc.



En el "Anexo IV- Configuración y personalización de NetBeans" se explica en detalle como hacer estas configuraciones.

Debes conocer

Busca una guía donde explique cómo acceder a los parámetros de configuración personalizada de los proyectos en NetBeans, y las opciones entre las que podemos elegir para decidir cómo queremos trabajar en un proyecto software:

« Anterior Siguiente »

7.- Gestión de módulos.

Caso práctico



Después de haber probado a configurar algunos aspectos del entorno, ahora Juan desea empezar a programar. Tiene un trabajo pendiente en JavaScript, pero observa que, tristemente, este lenguaje no es soportado por NetBeans.

—¿Cómo que no? —Le dice Ana. —Basta con encontrar el módulo de JavaScript (estructuras del lenguaje más bibliotecas asociadas) y añadirlo como complemento al entorno. Entonces sí que podrás programar (también) en ese lenguaje.

A Juan le parece fascinante.

Con la plataforma dada por un entorno de desarrollo como NetBeans podemos hacer uso de módulos y plugins para desarrollar aplicaciones.

En la página oficial de NetBeans encontramos una relación de módulos y plugins, divididos en categorías.

Seleccionando la categoría Lenguajes de Programación, encontraremos aquellos módulos y plugins que nos permitan añadir nuevos lenguajes soportados por nuestro IDE.

Un módulo es un componente software que contiene clases de Java que pueden interactuar con las API del entorno de desarrollo y el manifest file, que es un archivo especial que lo identifica como módulo.



Los módulos se pueden construir y desarrollar de forma independiente. Esto posibilita su reutilización y que las aplicaciones puedan ser construidas a través de la inserción de módulos con finalidades concretas. Por esta misma razón, una aplicación puede ser extendida mediante la adición de módulos nuevos que aumenten su funcionalidad.

Existen en la actualidad multitud de módulos y plugins disponibles para todas las versiones de los entornos de desarrollo más utilizados. En las secciones siguientes veremos dónde encontrar plugins y módulos para NetBeans 6.9.1 que sean de algún interés para nosotros y las distintas formas de instalarlos en nuestro entorno.

También aprenderemos a desinstalar o desactivar módulos y plugins cuando preveamos que no los vamos a utilizar más y cómo podemos estar totalmente actualizados sin salir del espacio de nuestro entorno.

Veremos las categorías de plugins disponibles, su funcionalidad, sus actualizaciones...

Autoevaluación

¿Cómo crees que influye el hecho de tener módulos y plugins disponibles en el éxito que tenga un IDE?

- Contribuyen al éxito del entorno.
- No influyen en el éxito del entorno.

Efectivamente. Poder añadir funcionalidades concretas según lo que necesitemos hacen que el IDE sea muy aceptado por los usuarios.

No. La aceptación de un entorno será mayor si permite que el usuario decida qué funcionalidades desea incluir.

Solución

1. Opción correcta
2. Incorrecto

[« Anterior](#) [Siguiente »](#)

7.1.- Añadir.

Caso práctico

Ya sabemos que podemos añadir funcionalidades a nuestro entorno. Pero ni Juan ni Ana saben cómo hacerlo. Piden ayuda a María, que decide ayudarles.

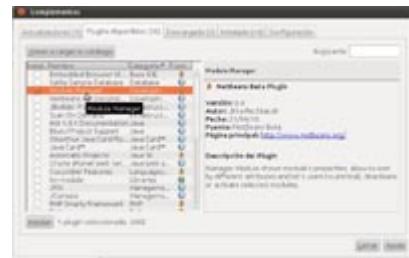
—Añadir módulos y plugins es muy sencillo, prestad atención.



Añadir un módulo va a provocar dotar de mayor funcionalidad a nuestros proyectos desarrollados en NetBeans.

Para añadir un nuevo módulo tenemos varias opciones:

1. Añadir algún módulo de los que NetBeans instala por defecto.
2. Descargar un módulo desde algún sitio web permitido y añadirlo.
3. Instalarlo on-line en el entorno.



Por supuesto, una cuarta posibilidad es crear el módulo nosotros mismos (aunque eso no lo veremos aquí).

Sin embargo, lo más usual es añadir los módulos o plugins que realmente nos interesan desde la web oficial de NetBeans. El plugin se descarga en formato .nbm que es el propio de los módulos en NetBeans. Posteriormente, desde nuestro IDE, cargaremos e instalaremos esos plugins. A esta manera de añadir módulos se le conoce como adición off-line.

También es habitual instalarlos on-line, sin salir del IDE.

La adición on-line requiere tener instalado el plugin Portal Update Center en NetBeans 6.9.1 y consiste en instalar complementos desde nuestro mismo IDE, sin tener que descargarlos previamente.

Al final del tema, en "Anexo V (apartado a).- Adición de módulo en NetBeans" se explica en detalle como añadir un plugin en Netbeans.

Debes conocer

Navegar y familiarizarse por la plataforma web que NetBeans pone a disposición de los desarrolladores es fundamental para estar al día de las últimas funcionalidades que podemos añadir a nuestro entorno mediante la instalación de plugins

Búsqueda online de plugins para NetBeans



« Anterior Siguiente »

7.2.- Eliminar.

Cuando consideramos que algún módulo o plugin de los instalados no nos aporta ninguna utilidad, o bien que el objetivo para el cual se añadió ya ha finalizado, el módulo deja de tener sentido en nuestro entorno. Es entonces cuando nos planteamos eliminarlo.

Eliminar un módulo es una tarea trivial que requiere seguir los siguientes pasos:

1. Encontrar el módulo o plugin dentro de la lista de complementos instalados en el entorno.
2. A la hora de eliminarlo, tenemos dos opciones:
 1. Desactivarlo: El módulo o plugin sigue instalado, pero en estado inactivo (no aparece en el entorno).
 2. Desinstalarlo: El módulo o plugin se elimina físicamente del entorno de forma permanente.

Esta es la ventana, desde el gestor de complementos de NetBeans, que nos aparece cuando queremos eliminar un módulo del entorno.

Siempre nos pedirá elegir entre dos opciones: desactivar o desinstalar.

En este ejemplo, se opta por desactivar el complemento, como podemos ver en la imagen.



Al final del tema, en "Anexo VI.- Eliminar módulos en NetBeans" se detalla como quitar un plugin en NetBeans"

Autoevaluación

Para añadir un módulo desde la web oficial de NetBeans:

- Hay que instalar el plugin Update Center.
- Hay que conectar con la web desde Netbeans y instalar on-line.
- Hay que encontrar el complemento, descargarlo y luego instalarlo en el IDE.
- No se pueden descargar los complementos desde ahí.

Incorrecta, ese plugin sólo es necesario en adiciones on-line.

No es correcta porque la instalación tiene que ser off-line.

Muy bien. Esa es la idea.

Incorrecta. Sí se pueden descargar plugins y módulos para nuestro IDE.

Solución

1. Incorrecto

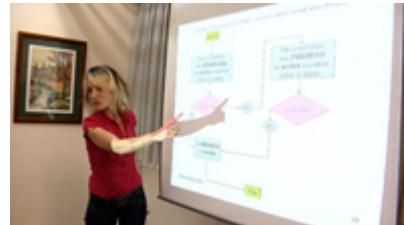
- 2. Incorrecto
- 3. Opción correcta
- 4. Incorrecto

[« Anterior](#) [Siguiente »](#)

7.3.- Funcionalidades.

Caso práctico

—Para que sepas qué puedes encontrar en los complementos de NetBeans, te recomiendo que tengas claras las funcionalidades que ofrece, teniendo en cuenta que se van ampliando día a día, —le comenta Ana a Juan.



Los módulos y plugins disponibles para los entornos de desarrollo, en sus distintas versiones, tienen muchas y muy variadas funciones.

Podemos clasificar las distintas categorías de funcionalidades de módulos y plugins en los siguientes grupos:

1. **Construcción de código:** facilitan la labor de programación.
2. **Bases de datos:** ofrecen nuevas funcionalidades para el mantenimiento de las aplicaciones.
3. **Depuradores:** hacen más eficiente la depuración de programas.
4. **Aplicaciones:** añaden nuevas aplicaciones que nos pueden ser útiles.
5. **Edición:** hacen que los editores sean más precisos y más cómodos para el programador.
6. **Documentación de aplicaciones:** para generar documentación de los proyectos en la manera deseada.
7. **Interfaz gráfica de usuario:** para mejorar la forma de presentación de diversos aspectos del entorno al usuario.
8. **Lenguajes de programación y bibliotecas:** para poder programar bajo un Lenguaje de Programación que, en principio, no soporte la plataforma.
9. **Refactorización:** hacer pequeños cambios en el código para aumentar su legibilidad, sin alterar su función.
10. **Aplicaciones web:** para introducir aplicaciones web integradas en el entorno.
11. **Prueba:** para incorporar utilidades de pruebas al software.

Autoevaluación

¿Qué categoría de funcionalidad de NetBeans te parece más interesante? ¿Por qué?

- Todas son igual de interesantes porque aumentan la funcionalidad.
- Depende de la tarea a realizar y el nivel del usuario.

No es del todo correcto. Según el nivel del usuario y la tarea unas serán más importantes que otras.

Muy bien. Esa es la idea.

Solución

1. Incorrecto
2. Opción correcta

Para saber más

En el siguiente vídeo, se hace un repaso de la adición de nuevas funcionalidades a NetBeans:

<https://www.youtube.com/embed/8icMxyazHHk>

[Resumen textual alternativo](#)

[« Anterior](#) [Siguiente »](#)

7.4.- Herramientas concretas.

- ✓ Importador de Proyectos de NetBeans: permite trabajar en lenguajes como JBuilder.
- ✓ Servidor de aplicaciones GlassFish: Proporciona una plataforma completa para aplicaciones de tipo empresarial.
- ✓ Soporte para Java Enterprise Edition: Cumplimiento de estándares, facilidad de uso y la mejora de rendimiento hacen de NetBeans la mejor herramienta para crear aplicaciones de tipo empresarial de forma ágil y rápida.
- ✓ Facilidad de uso a lo largo de todas las etapas del ciclo de vida del software.
- ✓ NetBeans Swing GUI builder: simplifica mucho la creación de interfaces gráficos de usuarios en aplicaciones cliente y permite al usuario manejar diferentes aplicaciones sin salir del IDE.
- ✓ NetBeans Profiler: Permite ver de forma inmediata ver cómo de eficiente trabajará un trozo de software para los usuarios finales.
- ✓ El editor WSDL facilita a los programadores trabajar en servicios Web basados en XML.
- ✓ El editor XML Schema Editor permite refinar aspectos de los documentos XML de la misma manera que el editor WSDL revisa los servicios Web.
- ✓ Aseguramiento de la seguridad de los datos mediante el Sun Java System Acces Manager.
- ✓ Soporte beta de UML que cubre actividades como las clases, el comportamiento, la interacción y las secuencias.
- ✓ Soporte bidireccional, que permite sincronizar con rapidez los modelos de desarrollo con los cambios en el código conforme avanzamos por las etapas del ciclo de vida de la aplicación.
- ✓ Etc.



Para saber más

Amplía las herramientas concretas que ofrece NetBeans para el desarrollo de aplicaciones multiplataforma.

Visita la web oficial:

[Información herramientas concretas de NetBeans](#)

Autoevaluación

¿En qué fases del desarrollo de software ayudan los entornos integrados de desarrollo?

- En codificación, pruebas, documentación, explotación y mantenimiento.
- En codificación y documentación.
- En análisis y documentación.

Efectivamente. Ofrece funcionalidades concretas en todas esas fases.

No. También ayudan a la gestión de pruebas al software y mantenimiento.

Incorrecta. El análisis es una tarea donde hay que concretar todos los aspectos que queremos que la aplicación resuelva, y eso sólo puede hacerlo una persona (analista).

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

[« Anterior](#) [Siguiente »](#)

8.- Uso básico de entornos de desarrollo.

Caso práctico

—En qué partes se divide el espacio principal del entorno? Vamos a echar un vistazo, — le comenta Juan a Antonio. (A Juan le gusta explicárselo a su compañero, ahora que va descubriendo las ventajas de los IDE).



En el sitio principal del entorno de desarrollo de NetBeans nos encontramos con la siguiente ventana, que aparece cuando seleccionamos archivo, nuevo proyecto, java:



Vemos que el espacio se divide en dos ventanas principales.

✓ Ventana Izquierda: ventana de proyectos.

Aquí irá apareciendo la relación de proyectos, archivos, módulos o clases que vayamos abriendo durante la sesión.

Cada proyecto comprende una serie de archivos y bibliotecas que lo componen.

El principal archivo del proyecto Java es el llamado **Main.java**. Es el archivo por donde empieza la ejecución. No tiene porque llamarse Main pero es aconsejable para que cualquier persona que vea el proyecto sepa porque fichero (clase) empieza la ejecución.



✓ Ventana derecha: espacio de escritura de los códigos de los proyectos.

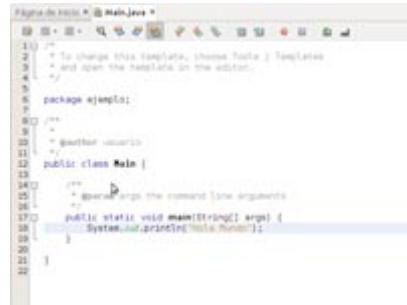
Aquí aparece el esqueleto propio de un programa escrito en lenguaje Java.

Se ha añadido el código:

```
System.out.println("Hola Mundo");
```

Y veremos su significado en las siguientes páginas. De momento, saber que para escribir cualquier código, hay que hacerlo en esta ventana.

BARRA DE HERRAMIENTAS: Desde aquí podremos acceder a todas las opciones del IDE.



```
Paleta de herramientas > Main.java <
```

```
1) /* To change this template, choose Tools | Templates  
2) and open the template in the editor.
```

```
3) package ejemplo;
```

```
4) /**  
5)  * Author: vassallo  
6) */  
7) public class Main {  
8)     /**  
9)      * Prints the command line arguments.  
10)     */  
11)    public static void main(String[] args) {  
12)        System.out.println("Hola mundo!");  
13)    }  
14} 
```



[« Anterior](#) [Siguiente »](#)

8.1.- Edición de programas.

Caso práctico

—Vamos a hacer el primer ejemplo —comenta Ana, entusiasmada—. Después de todo, no debemos perder de vista la finalidad de la herramienta, ESCRIBIR PROGRAMAS!



En este sencillo ejemplo se ve una modificación de las líneas de código en la ventana de codificación del archivo **Main.java** del proyecto **ejemplo** que acabamos de crear.

Las dos líneas que aparecen resaltadas se han escrito sobre la ventana y, tal y como significan en lenguaje Java, su ejecución implicará que sendos mensajes encerrados entre comillas y entre paréntesis saldrán impresos.

No hay que decir que la programación en Java no es objeto del presente módulo, pero puedes probar con algunos ejemplos en Java que tengas de otros módulos.

```
package ejemplo;
public class Main {
    public static void main(String[] args) {
        System.out.println("Creando el primer ejemplo");
    }
}
```

Mientras escribimos en el editor de textos nos percatamos de varias características de NetBeans que ya hemos señalado en páginas anteriores:

- ✓ Autocompletado de código.
- ✓ Coloración de comandos.
- ✓ Subrayado en rojo cuando hay algún error y posibilidad de depuración y corrección de forma visual, mediante un pequeño ícono que aparece a la izquierda de la línea defectuosa.

Al final del tema, en el "Anexo VII.- Ejemplo de edición de código" se explica, en detalle, como escribir código en NetBeans.

Debes conocer

El proceso de edición de un programa desde que arranca el entorno hasta que está libre de errores sintácticos.

En el siguiente documento tienes el código del ejemplo de arriba.

[Pequeño ejemplo de edición de código](#)

Este es otro ejemplo (pide dos números y te visualiza la suma de ambos números)

[Ejemplo de la suma de dos números](#)[« Anterior](#) [Siguiente »](#)

8.2.- Generación de ejecutables.

Una vez tenemos el código plasmado en la ventana de comandos y libre de errores de sintaxis, los siguientes pasos son: compilación, depuración, ejecución.

Para ejecutar el programa sólo tienes que pulsar shift+F6. De esta forma se ejecutará el código que tengas delante.

Al ejecutar el ejemplo anterior, el resultado es:



Si a este ejemplo le añadimos la funcionalidad de JFrame, el resultado de la ejecución es:



Este ejemplo (hecho con JFrame) aparece en el siguiente documento, al que accederás siguiendo el siguiente enlace:

[Pequeño ejemplo de ejecución de código](#)

Al final del tema, en el "Anexo VIII.- Ejecución de un programa en NetBeans" se detalla como ejecutar un programa en NetBeans, de otra forma que no sea pulsando shift+F6

Autoevaluación

Los pasos que debemos dar para generar un ejecutable son:

- Ejecución directa.
- Ejecución, una vez que el editor esté libre de errores sintácticos.
- Una vez que el editor esté libre de errores, compilar, depurar y ejecutar.

Incorrecto. La ejecución directa nunca es recomendable (como mucho, sólo en programas muy pequeños).

No es del todo correcto. Además de los errores sintácticos, existen otro tipo de errores que debemos detectar antes de proceder a la ejecución del programa.

Así es. Estos serían los pasos y la secuencia correcta de actuación para generar ejecutables.

Solución

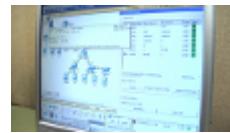
1. Incorrecto
2. Incorrecto
3. Opción correcta

[« Anterior](#) [Siguiente »](#)

9.- Actualización y mantenimiento de entornos de desarrollo.

Caso práctico

—Por último, es de vital importancia el mantener y actualizar el entorno de desarrollo — comenta Ana—. Deberíamos tener permanentemente actualizados todos los complementos y realizar un correcto mantenimiento a las bases de datos asociadas a nuestros proyectos.



El mantenimiento del entorno de desarrollo es una tarea fundamental que requiere tener todos sus componentes periódicamente actualizados.

También es de vital importancia realizar copias de seguridad sobre las bases de datos de nuestros proyectos por si ocurriera algún error o proceso defectuoso poder restaurarlos.

El mantenimiento y las actualizaciones se hacen de forma on-line. En NetBeans contamos con el complemento llamado Auto Update Services. Lo podemos encontrar en el siguiente enlace:

[Complementos de Netbeans](#)

Una vez instalado, nos permitirá realizar continuas revisiones del entorno y actualizaciones de todos los plugins.

Most downloaded	Top rated	Recently added or updated
JBoss Seam [114,400]	200-Plus [4.8/5] (4,416)	ADLT Cellviewer [2011-08-04]
NetBeans GlassFish [101,231]	Slope [4.8/5] (4,415)	Class At Project [2011-09-04]
Visual Web Flow [201,486]	NetBeans EJB Tools [4.7/5] (4,287)	Categories Code [2011-09-03]
Apache Felix [201,486]	UI-Editor [4.7/5] (4,146)	Address Book [2011-09-03]
Fusio [1,422]	Wicket editor [4.7/5] (4,145)	Books Library [2011-09-03]
NetBeans SeamANTA [201,140]	NetBeans [4.7/5] (4,137)	More... [2011-09-03]
Show more...	Show more...	Show more...

Para añadir módulos y plugins on-line, hay que tener este complemento instalado en el entorno.

La gestión de las bases de datos asociadas a nuestros proyectos es muy importante. Habrá que realizarles copias de seguridad periódicamente, para asegurar su restauración en caso de fallos en el sistema, y mantenerlas actualizadas para su posible portabilidad futura a nuevas versiones del entorno que utilicemos.

Autoevaluación

¿Cuál es la razón, en tu opinión, de que salgan nuevas versiones de los entornos de desarrollo tan rápidamente?

- Para adaptarse a la evolución del hardware.
- Para incluir y modificar funcionalidades del entorno.

Incorrecto. Esto no es significativo en la evolución de los entornos de desarrollo.

Así es. Cada nueva versión tiene mejoras que permite aumentar la funcionalidad del entorno.

Solución

1. Incorrecto
2. Opción correcta

[« Anterior](#) [Siguiente »](#)

Anexo I.- Crear, compilar y ejecutar un programa (sin el uso de un entorno de desarrollo).

Vamos a **crear, compilar y ejecutar** nuestro primer programa en java.

Vamos a hacerlo usando un editor de textos como es el bloc de notas y el JDK. No vamos a usar ningún entorno de desarrollo.

Para ello será necesario haber configurado correctamente las variables de entorno JAVA_HOME y PATH como se indica en los contenidos de la unidad.

Paso 1: Utilizando un editor de textos crear el fichero **Hola.java** con el siguiente código:

```
public class Hola {
    public static void main(String[ ] args) {
        System.out.println("Hola");
    }
}
```

Es muy importante que el nombre del fichero sea **Hola.java** (respeta mayúsculas y minúsculas)

Salva el archivo en alguna ruta de tu sistema de archivos.

Paso 2: Deberás abrir un terminal. Más adelante, se ofrece ayuda en este paso

Paso 3: Una vez te encuentres en el directorio/carpeta donde está el fichero Hola.java hay que ejecutar el comando:

javac Hola.java

De esta forma compila el fichero, generando el fichero "Hola.class."

Paso 4: Una vez que tenemos el fichero "Hola.class" hay que interpretarlo con el comando "**java**". *Hay que poner esta instrucción:*

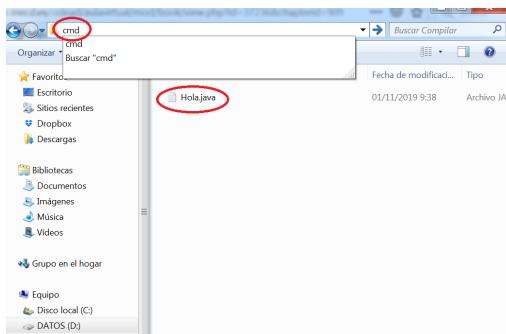
java Hola.

Resultado de la compilación y ejecución (ejemplo en Linux).

```
debian@debian:~/Hola$ more Hola.java
public class Hola {
    public static void main (String args[]){
        System.out.println("Hola");
    }
}
debian@debian:~/Hola$ javac Hola.java
debian@debian:~/Hola$ java Hola
Hola
debian@debian:~/Hola$ ls
Hola.class Hola.java
debian@debian:~/Hola$
```

**Ayuda para el paso 2:
En Windows:**

Lo haremos desde la ubicación del fichero Hola.java para tener un acceso más sencillo a él desde el terminal. Abrimos el explorador y nos situamos en la ubicación del fichero. En la barra de direcciones escribimos cmd y pulsamos ENTER.



Se abre el terminal.

También se puede acceder al terminal desde Inicio, tecleando cmd y ENTER en el campo de texto. Pero de este modo tendrás que usar el comando cd para situarte en el directorio/carpeta donde se encuentra el fichero Hola.java

En Linux

<https://www.comoinstalarlinux.com/como-abrir-una-terminal-en-ubuntu-linux-mint-centos-debian/>

En MAC OS

<https://www.soydemac.com/abrir-terminal-mac/>

NOTA:

Para que el paso 3 resulte más sencillo puedes abrir el terminal en el directorio/carpeta donde se encuentre el fichero Hola.java. Si no es así, deberás usar el comando cd en el terminal para situarte en esa ruta. En las siguientes URLs puedes ver ejemplos de uso de ese comando para que puedas situarte en el directorio/carpeta correcto.

En Windows: <https://www.abrillave.com/cmd/comando-cd.php>.

En Linux: <https://www.solvetic.com/tutoriales/article/7190-como-usar-el-comando-cd-en-linux/> .

En MAC OS: <https://www.faq-mac.com/2003/06/guia-del-terminal-unix-command-line-para-usuarios-mac-parte-iii/>

Con esta práctica habrás visto que no hemos usado el IDE "NetBeans" en ningún momento. Solamente hemos usado un editor de textos (como puede ser el "bloc de notas") y los programas "javac" y "java".

JavaC es un compilador que genera un fichero con el mismo nombre del fichero original pero con extensión class. Traduce el fichero escrito en Java y lo pasa a bytecode.

java es un intérprete que traduce los bytecode a código máquina, para que el ordenador lo entienda y lo pueda ejecutar. Como intérprete que es, va traduciendo línea a línea y ejecutando. No genera ningún fichero nuevo (los intérpretes no generan ficheros).

« Anterior Siguiente »

Anexo II.- Crear, compilar y ejecutar un programa con varias clases (sin el uso de un entorno de desarrollo).

Esta práctica nos servirá de apoyo para poder entender, a través de un ejemplo, algunos de los conceptos que hemos estado viendo hasta el momento, tales como:

- Lenguaje de alto nivel. Un acercamiento al lenguaje Java.
- Código fuente y código máquina.
- Traductor. En este caso se trata del compilador de Java.
- Código interpretable o bytecode. Intérprete.
- Conceptos asociados con la programación orientada a objetos.

No forma parte de este ejercicio, entender el código **Java** en su totalidad. Dichos contenidos serán tratados en otros módulos del ciclo.

Esta práctica, igual que hemos hecho en la anterior, se va hacer usando un editor de textos (como, por ejemplo, el bloc de notas) y el JDK. No se va a usar ningún entorno de desarrollo.

Ejercicio Propuesto

Un profesor de autoescuela pretende enseñar a un estudiante la combinación de colores entre los que puede ir cambiando un semáforo.

En el aula existe un ordenador que simula el comportamiento del semáforo, al que podemos consultar el color que tiene en cada momento.

En la secuencia de ejecución, el profesor preguntará al estudiante el color que tiene el semáforo. Para responder, el estudiante obtiene el resultado del ordenador.

Cuando damos solución a un problema propuesto mediante un programa, éste puede ser implementado de muy diversas formas. Cualquiera de ellas será igualmente válida siempre y cuando cumpla los requisitos que se le piden (funcionales, de rendimiento, disponibilidad,...).

A continuación se propone una de las posibles soluciones. Pulsa [aquí](#) para descargar el fichero **semáforo.zip** que incluye el código fuente.

[Mostrar retroalimentación](#)

Identificando actores / objetos involucrados

De acuerdo a lo solicitado en el enunciado, es posible plantear el problema haciendo uso de tres actores (profesor, estudiante, ordenador). Para cada uno de ellos se desarrollará una clase **Java**.

Se crearán cinco ficheros de código fuente, que una vez compilados nos proporcionarán otros tantos ficheros en código intermedio (**bytecode de Java**).

Para la ejecución del programa, el **intérprete de Java** hará uso de los ficheros **bytecode** creados, junto a otras funciones pertenecientes a las bibliotecas de Java.

Cuando queremos implementar una solución, se crea un “proyecto”, cuyos contenidos quedan almacenados en el sistema de archivos del ordenador en un directorio (semaforo en nuestro caso).

A partir de aquí, se pueden ver subdirectorios, que organizan los diferentes archivos del proyecto en paquetes (paquetes clases y principal para este ejemplo).

Por último, cada paquete puede contener un conjunto de ficheros, donde se implementan las clases. Podrá ser una o varias por fichero (típicamente fichero .java implementa una clase).

Ejemplo de visualización de ficheros en Linux:

```
Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/semaforo# ls ./principal/
ClaseColor.java
root@debian:/home/debian/Escritorio/semaforo# ls ./clases/
estudiante.java ordenador.java persona.java profesor.java
root@debian:/home/debian/Escritorio/semaforo#
```

Descripción de las clases

Clase ClaseColor (ClaseColor.java)

```
1 package principal;
2 import clases.Profesor;
3 // Clase color, el profesor pregunta a un alumno por un color entre
4 public class ClaseColor {
5     public static void main(String[] args) {
6         Profesor teacher = new Profesor();
7         String color = teacher.preguntacolor();
8         System.out.println("La respuesta recibida es: " + color);
9     }
10 }
```

Consideraciones de la clase ClaseColor:

- La clase ClaseColor está contenida en el fichero ClaseColor.java y se incluye en un paquete llamado principal (1).
- Como parte del código de la clase, se va a utilizar la clase Profesor que forma parte del paquete clases (2).
- El inicio del programa se lleva a cabo en esta clase, puesto que incluye la función main (5).
- De la clase Profesor se crea un objeto llamado teacher (6).
- Desde la clase ClaseColor se envía un evento a la clase Profesor (a través del objeto teacher), para que esta última ejecute el método preguntacolor (7).
- También utilizamos funciones proporcionadas en las librerías de Java, como println .

Clase Ordenador (Ordenador.java)

```
1 package clases;
2 import java.util.Random;
3 public class Ordenador {
4     public Ordenador() {}
```

```

5   public String color(){
6       Random randomGenerator = new Random();
7       int randomInt = randomGenerator.nextInt(3);
8       if(randomInt == 0)
9           { return "rojo";}
10      else if(randomInt == 1)
11          { return "amarillo";}
12      else
13          { return "verde";}
14  }
15 }
```

Consideraciones de la clase ordenador:

- La clase Ordenador utiliza el método Random para obtener un número aleatorio entre 0 y 2.
- El método color devuelve una de las siguientes cadenas de caracteres (“rojo”, “amarillo”, “verde”), instrucción return (9, 11, 13).

Clase Persona (Persona.java)

```

1 package clases;
2 // Clase utilizada para ser herencia de estudiante y profesor
3 public class Persona {
4     // Métodos de clase. Edad y nombre
5     int i_Edad;
6     String s_Nombre;
7 }
```

Consideraciones de la clase persona:

- La clase Persona no va a ser instanciada directamente en el programa (crear un objeto de la clase). Se trata de una clase padre, que va a ser utilizada por las clases estudiante y profesor para heredar sus características.
- En esta clase se definen las variables de clase int i_Edad y s_Nombre (5, 6).

Clase Estudiante (Estudiante.java)

```

1 package clases;
2 public class Estudiante extends Persona{
3     // Incluye un método de clase que se une a los heredados
4     int i_Curso;
5     public Estudiante() {
6         i_Edad=25;
7         s_Nombre = "Luis";
8         i_Curso = 1;
9     }
10    public void presentarse(){
11        System.out.println("Soy " + s_Nombre + " Alumno de " +
12        i_Curso);
13    }
14    public String preguntacolor(){
15        presentarse();
16        Ordenador mipc = new Ordenador();
17        return mipc.color();
```

```

17   }
18 }
```

Consideraciones de la clase Estudiante:

- Esta clase hereda de la clase Persona, cláusula extends (2).
- En la clase estudiante se pueden utilizar las propiedades y métodos definidos en la clase padre (6, 7, 11).
- Además puede incluir otros métodos y propiedades propias (4).
- Si hubiéramos sobreescrito una de las propiedades o métodos del padre, en la propia clase, ésta utilizaría sus propios métodos y propiedades en lugar de los heredados.

Clase Profesor (Profesor.java)

```

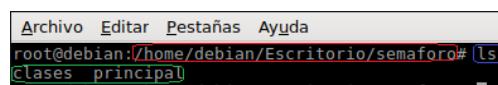
1 package clases;
2
3 public class Profesor extends Persona{
4     Public Profesor() {}
5
6     // Hace la pregunta al estudiante sobre el color
7     Public String preguntacolor(){
8         Estudiante alumno = new Estudiante();
9         String colorRec = alumno.preguntacolor();
10        return colorRec;
11    }
12}
```

Uso del traductor/compilador de Java

Como se indicaba anteriormente, cuando se crea un nuevo proyecto **Java** todos sus recursos "cuelgan" de un determinado directorio en el sistema de archivos ("/home/debian/Escritorio/semaforo" en este ejemplo). Puedes elegir el que quieras en tu equipo.

A partir de aquí, podemos ir distribuyendo los ficheros en diferentes paquetes, que corresponderán a subdirectorios del directorio proyecto ("/home/debian/Escritorio/semaforo"). Así hemos incluido los ficheros Ordenador.java, Persona.java, Profesor.java y Estudiante.java en el paquete clases. Además de distribuir los paquetes en el directorio correspondiente, es necesario indicarlo en el código (ver **instrucción package**).

Para utilizar cada una de estas clases en otros ficheros .java deberemos importarlos (**instrucción import**). Si dos ficheros forman parte del mismo paquete, no es necesario que sean importados.



El fichero de inicio del programa ClaseColor.java (incluye la **función main**), y forma parte del paquete principal.

Para obtener el código **bytecode** del fichero clasecolor.java:

- Acceder al directorio de inicio del proyecto desde un terminal.
- Llamar al compilador (traductor) de Java "javac" indicando la ruta relativa del fichero que queremos compilar. En nuestro caso "javac

principal/ClaseColor.java".

- Como resultado obtenemos el fichero interpretable por Java ClaseColor.class.

Puedes ver el proceso en la siguiente imagen (ejemplo en Linux).

```

Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/semaforo# ls ./principal/
clasecolor.java
root@debian:/home/debian/Escritorio/semaforo# javac ./principal/clasecolor.java
root@debian:/home/debian/Escritorio/semaforo# ls ./principal/
ClaseColor.class
root@debian:/home/debian/Escritorio/semaforo# ls ./clases/
estudiante.class ordenador.class persona.class profesor.class
estudiante.java ordenador.java persona.java profesor.java

```

Para que el compilador funcione tal y como se ha descrito en este punto, será necesario que se hayan metido las variables de entorno tal y como se ha indicado la práctica anterior.

Nota: la compilación del fichero principal "javac principal/ClaseColor.java" puede ya compilar todos los ficheros del proyecto al ser unos dependientes de otros. En caso contrario, compilaremos el resto uno a uno en el siguiente orden: Persona.java, Ordenador.java, Estudiante.java, Profesor.java y ClaseColor.java.

Una vez disponibles todos los ficheros **bytecode**, procedemos a la ejecución del programa desde un terminal, vamos al directorio de inicio del proyecto y ejecutamos el intérprete java: "java principal/ClaseColor".

```

debian@debian: ~
Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/semaforo# java principal/clasecolor
Soy Luis Alumno de 1 y tengo una edad de: 25
La respuesta recibida es:amarillo

```

Empaquetar .class en fichero .jar

Java nos permite empaquetar todos los ficheros (.class, junto a otros recursos utilizados) del proyecto en uno único con extensión **.jar (Java ARchives)**. Además el formato del fichero .jar es comprimido, por lo que ocupa menos espacio que la información original.

A continuación se indica el procedimiento para llevarlo a cabo.

Crear el fichero MANIFEST.MF

Crear un directorio **META-INF** (¡las mayúsculas son importantes!) y dentro un fichero **MANIFEST.MF**. Este fichero indica, entre otras cosas, cuál será la clase principal. A menudo el fichero **MANIFEST.MF** contiene una única línea:

Main-Class: principal.ClaseColor

donde se indica que ClaseColor.class (del paquete principal) es la clase que contiene el método main.

Nota : el fichero **MANIFEST.MF** se puede crear con cualquier editor de texto. Hay que introducir un fin de línea (enter) tras la última línea con texto.

Crear el fichero semaforo.jar

El paquete **JDK** proporciona una serie de ejecutables. Ya se ha trabajado con algunos de ellos como el compilador de java (javac) y con el intérprete (java) en prácticas anteriores. Otro de los programas a nuestra disposición es **jar**, que permite empatequetar todos los recursos de un proyectos **Java** en un fichero con extensión .jar.

Desde consola del sistema, ejecutar el programa jar con los siguientes parámetros:

jar cmf META-INF/MANIFEST.MF semaforo.jar clases/.class principal/*.class*

```

debian@debian: ~
Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/semaforo# jar cmf META-INF/MANIFEST.MN semaforo.jar *.class
root@debian:/home/debian/Escritorio/semaforo# ls
clases META-INF principal semaforo.jar
root@debian:/home/debian/Escritorio/semaforo# java -jar semaforo.jar
Soy Luis Alumno de 1 y tengo una edad de: 25
La respuesta recibida es rojo
  
```

Observa en la imagen una ejecución del proyecto utilizando el fichero .jar obtenido:

`java -jar semaforo.jar`

El programa **jar** también nos permite ver el contenido de proyecto comprimidos .jar:

`jar tf semaforo.jar`



```

debian@debian: ~
Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/semaforo# jar tf semaforo.jar
META-INF/
META-INF/MANIFEST.MF
clases/estudiante.class
clases/ordenador.class
clases/persona.class
clases/profesor.class
principal/clasecolor.class
  
```

Si deseamos descomprimir la información del fichero .jar, para obtener sus fichero originales:

`jar xf semaforo.jar`

```

debian@debian: ~
Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/descomprimir# ls
semaforo.jar
root@debian:/home/debian/Escritorio/descomprimir# jar xf semaforo.jar
root@debian:/home/debian/Escritorio/descomprimir# ls -la
total 24
drwxr-xr-x 5 root root 4096 jun 18 23:49 .
drwxr-xr-x 5 debian debian 4096 jun 18 23:48 ..
drwxr-xr-x 2 root root 4096 jun 18 23:49 clases
drwxr-xr-x 2 root root 4096 jun 18 23:49 META-INF
drwxr-xr-x 2 root root 4096 jun 18 23:49 principal
-rw-r--r-- 1 root root 2877 jun 18 23:48 semaforo.jar
  
```



Script Compilar.sh

Para facilitar la creación de nuestros ficheros interpretables, es útil crear un **script** que realice todo el proceso de una sola vez. En nuestro proyecto podría ser **compilar.sh**, al que se deberá dar permisos de ejecución.

En el módulo de sistemas informáticos se estudia como crear scripts. Se puede dejar este ejercicio para cuando se haya visto como crear scripts en dicho módulo.

Ejemplo en Linux:



```
debian@debian: ~
Archivo Editar Pestañas Ayuda
root@debian:/home/debian/Escritorio/semaforo# more compilar.sh
javac principal/clasecolor.java
jar cmf META-INF/MANIFEST.MN semaforo.jar clases/*.class principal/*.class
root@debian:/home/debian/Escritorio/semaforo# ./compilar.sh
root@debian:/home/debian/Escritorio/semaforo# ls
clases compilar.sh META-INF principal semaforo.jar
root@debian:/home/debian/Escritorio/semaforo# java -jar semaforo.jar
Soy Luis Alumno de 1 y tengo una edad de: 25
La respuesta recibida es:amarillo
```

[« Anterior](#) [Siguiente »](#)

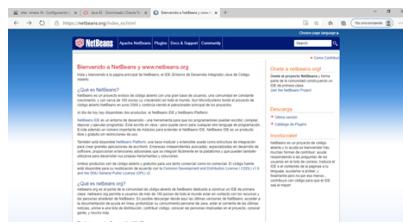
Anexo III.- Instalación JDK y NetBeans en Windows

PASOS

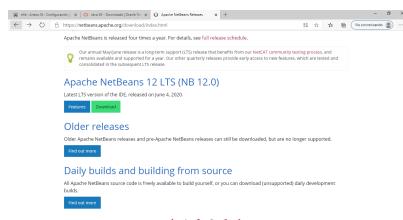
Descargar NetBeans de la siguiente URL:

[Descargar Netbeans.](https://netbeans.apache.org/download.html#stable)

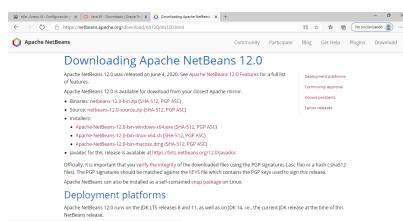
Aparecerá la siguiente ventana:



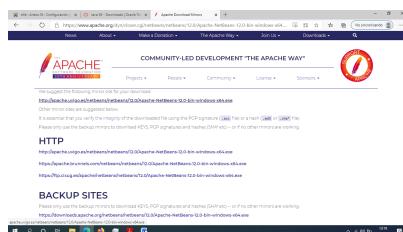
Ves a descarga y pulsa e “última versión”. Te saldrá esta ventana.



Pulsa en “DownLoad”



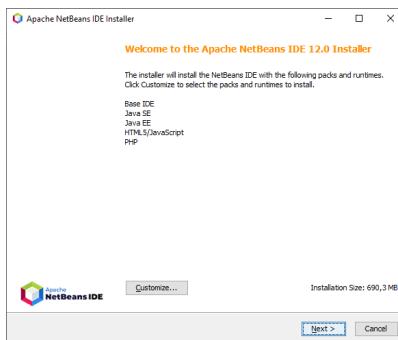
Elige la opción del instalador para Windows. Elige el primer enlace.



Se descargará un fichero con este nombre: Apache-NetBeans-12.0-bin-windows-x64.exe.

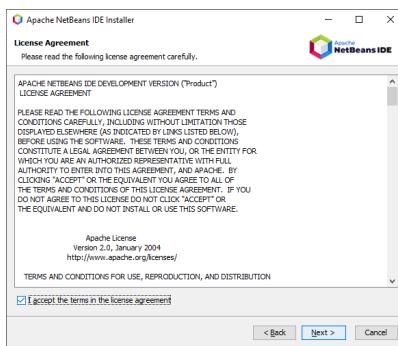
Instálelo.

Saldrá esta ventana.



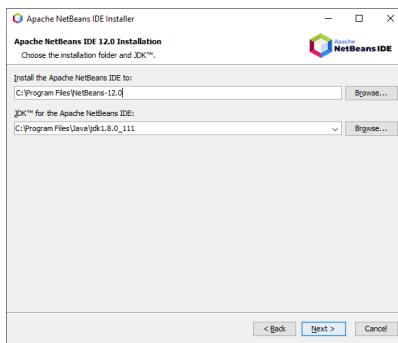
Pulsa “Next”.

Saldrá esta ventana.



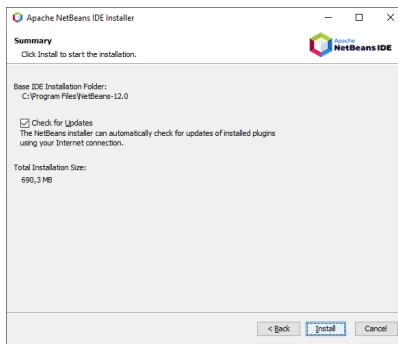
Acepta la licencia y pulsa “Next”.

Te informa que se va instalar NetBeans y el JDK.

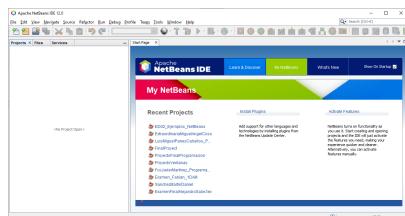


Pulsa “Next”.

Y, por último, pulsa “Install”.



Una vez instalado , ejecútalo y te visualizará esta ventana:

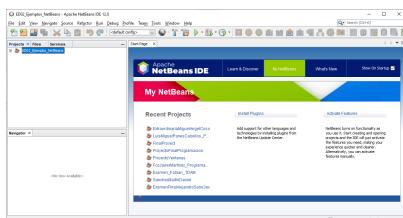


Donde podrás empezar a desarrollar software.

[« Anterior](#) [Siguiente »](#)

Anexo IV.- Configuración y personalización de NetBeans.

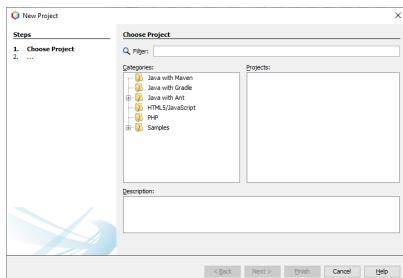
Accedemos a NetBeans y entramos en la página principal de la aplicación.



A la izquierda muestra los proyectos que tengas abiertos.

Para crear un nuevo proyecto, selecciona File/New Project.

Te visualizará esta ventana:



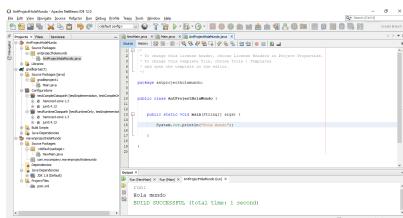
A la hora de crear un proyecto en Java existen tres opciones:

1. **Java with Maven**
2. **Java with Gradle**
3. **Java with Ant**

La diferencia entre ellos lo puedes ver en estos enlaces:

1. **Java with Maven:** <https://es.wikipedia.org/wiki/Maven>
2. **Java with Gradle:** <https://es.wikipedia.org/wiki/Gradle>
3. **Java with Ant:** https://es.wikipedia.org/wiki/Apache_Ant

Aquí se muestra el típico programa que se hace siempre cuando se empieza a manejar un lenguaje y es un simple programa que visualiza "Hola Mundo". Se muestra a continuación un ejemplo de las tres formas para ver la diferencia entre ellos.

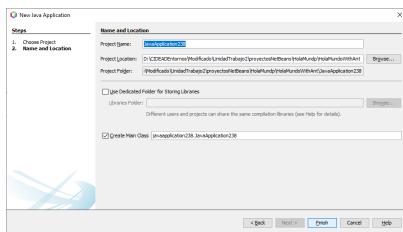


Principalmente mira todo el código que genera según la opción que elijas.

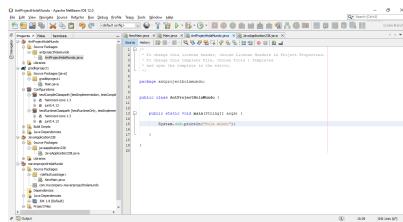
Nosotros, como no estamos iniciando en la programación en Java vamos a trabajar siempre con la opción: Java with Ant.

En esa opción cogéis "Java Application".

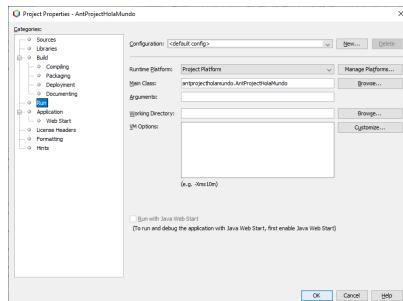
Os pide que indiquéis donde queréis guardar el proyecto y con qué nombre:



Seguidamente os visualizará esta ventana para que empiecéis a programar:



Una vez que tienes el proyecto, sobre el nombre del proyecto pulsa el botón derecho del ratón y elige la opción “Set configuration” y ahí la opción “Customize”:



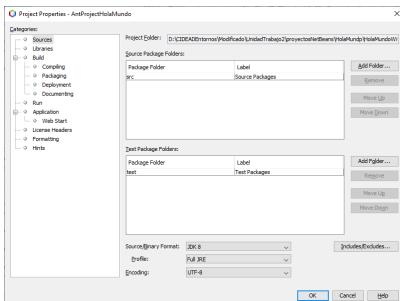
Aquí vemos todo lo que podemos personalizar de la aplicación:

- Fuentes.
- Bibliotecas.
- Generación de código.
- Ejecución de código.
- Opciones de la aplicación.
- Formato del código en el editor de textos.

FUENTES

Podemos modificar:

- La carpeta que contendrá el proyecto.
- La carpeta que almacenará los paquetes fuentes.
- La carpeta que contendrá los paquetes prueba.
- La versión del JDK con la que queremos trabajar.

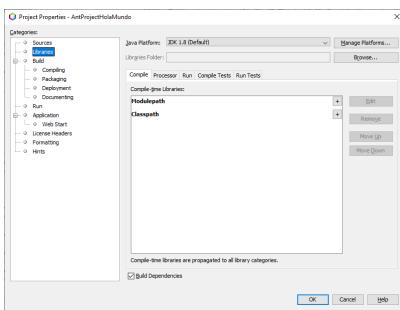


BIBLIOTECAS

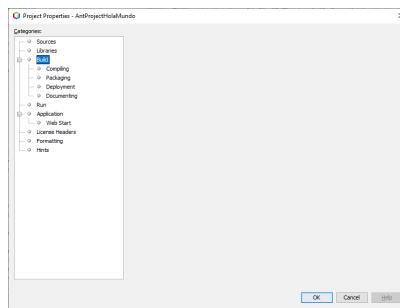
Desde esta ventana podemos elegir la plataforma de la aplicación.

Toma por defecto el JDK, pero se puede cambiar si se quiere, siempre y cuando sea compatible con la versión de NetBeans utilizada.

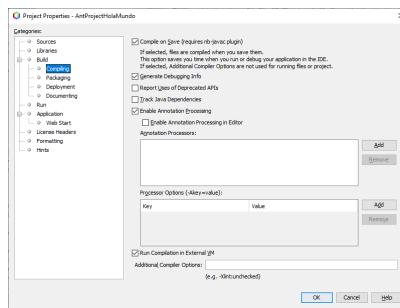
También en esta ventana se puede configurar el paquete de pruebas que se realizará al proyecto.



GENERACIÓN DE CÓDIGO



GENERACIÓN DE CÓDIGO - COMPILANDO

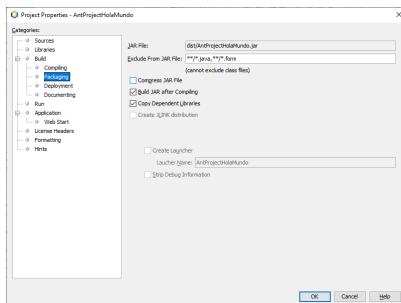


Las opciones que nos permite modificar en cuanto a la compilación del programa son:

- Compilar al guardar: al guardar un archivo se compilará automáticamente.
- Generar información de depuración: para obtener la documentación asociada.
- Enable annotation processing: permitir anotaciones durante el proceso.

También podemos agregar anotaciones concretas para el proceso de compilación y añadir opciones de proceso que, según las características del proyecto, puedan ser de interés para nosotros.

GENERACIÓN DE CÓDIGO - EMPAQUETANDO

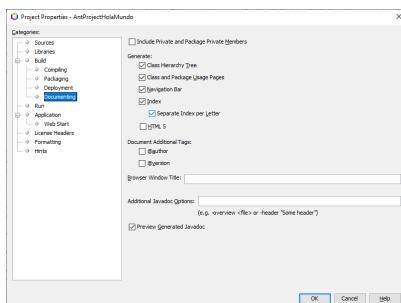


Las aplicaciones resultado de la compilación del código deben ser empaquetadas antes de su distribución, con objeto de tener un único archivo, generalmente comprimido, que contenga en su interior todos los archivos de instalación y configuración necesarios para que la aplicación pueda ser instalada y desarrollada con éxito por el usuario cliente.

Como vemos en la imagen, en esta opción podemos modificar el lugar donde se generará el archivo resultante del empaquetado, así como si deseamos comprimirlo.

También podemos elegir que el archivo empaquetado se construya tras la compilación, que es lo habitual (por eso esta opción aparece como predeterminada).

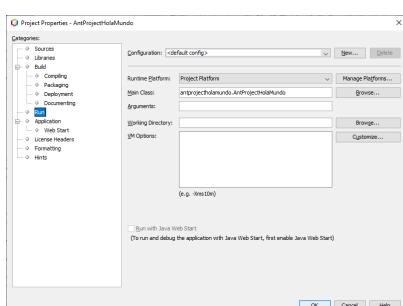
GENERACIÓN DE CÓDIGO – DOCUMENTANDO



Como ya vimos en la unidad anterior, la documentación de aplicaciones es un aspecto clave que no debemos descuidar nunca. NetBeans nos ofrece una ventaja muy considerable al permitirnos obtener documentación de la fase de codificación de los programas de forma automática.

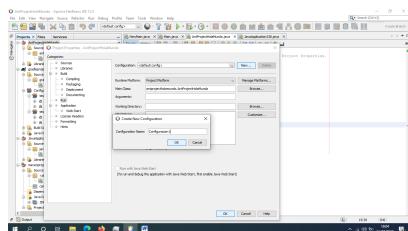
Dentro del documento que se va a generar podemos elegir que se incluyan todas las opciones anteriores. Esto es lo más recomendable, por eso aparecen todas marcadas de forma predeterminada y lo mejor es dejarlo como está.

EJECUTANDO CÓDIGO



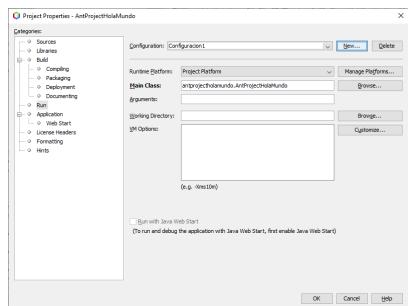
Esta opción nos permite definir una nueva configuración de ejecución de código, elegir la clase principal, las carpetas de trabajo del proyecto y opciones de la máquina virtual.

En la ventana de “Configurar el nombre” escribimos el nombre que tendrá nuestra configuración personalizada.



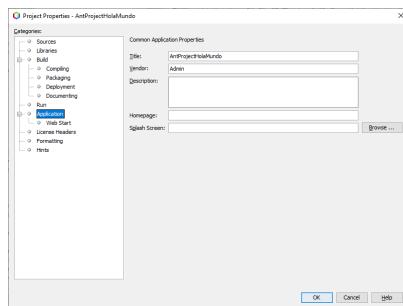
En este caso, escribimos “configuración1” y pulsamos “aceptar”.

A partir de este momento, todas las opciones de configuración que seleccionemos se guardarán en “configuración1”.



Ahora podemos elegir la aplicación sobre la cual queremos aplicar la configuración personalizada de “configuración 1”.

OPCIONES DE LA APLICACIÓN

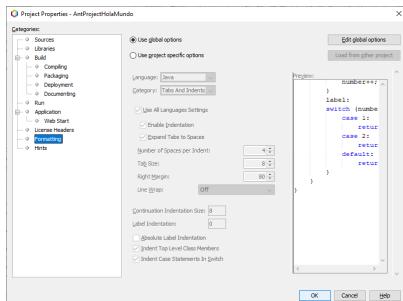


Como vemos, podemos dar una descripción al proyecto, cambiarle el nombre...

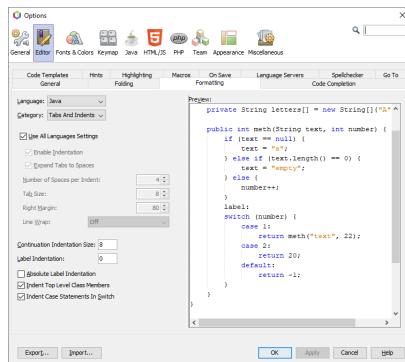
Es conveniente hacerlo, ya que el nombre de los nuevos proyectos se generar automáticamente por NetBeans al inicio de la sesión.

FORMATO

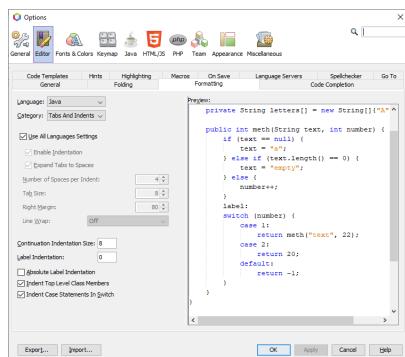
Aquí podemos personalizar aspectos globales del formato del código fuente en la aplicación. Podemos personalizar las opciones sólo para el proyecto actual o bien para todos los proyectos que estén basados en NetBeans a partir de ahora (utilizar opciones globales).



Si seleccionamos “Edit global options” (en opciones globales) nos encontramos con la siguiente ventana, que tiene una barra superior de pestañas para configurar cada apartado del formato de forma independiente.

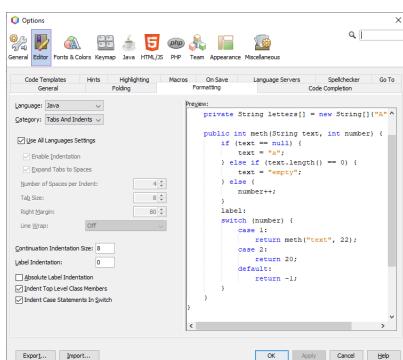


Opción “Editor”:

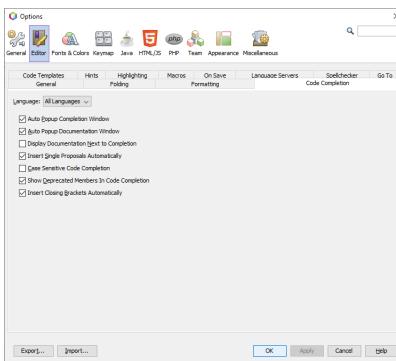


Pestaña Formatting:

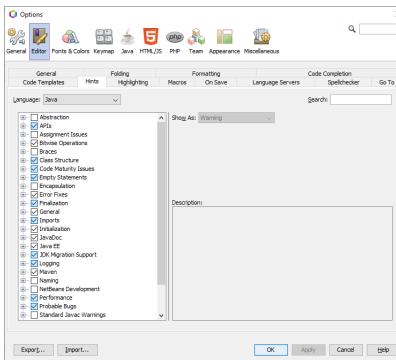
Se puede configurar los tamaños de los espaciados, pestanas...



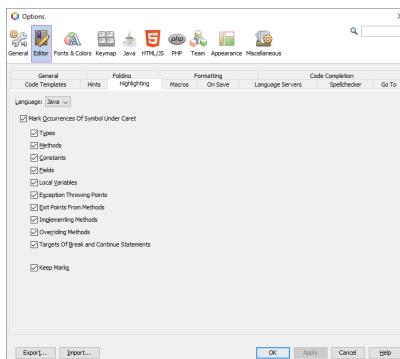
En la pestaña de “Code Completion” podremos cambiar:



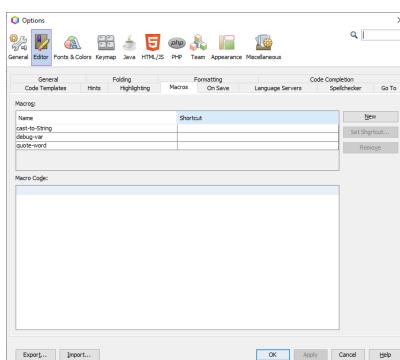
En la pestaña de “Hints”:



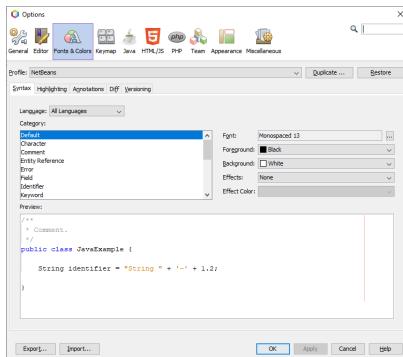
En la pestaña de “HighLighting” (Marcar ocurrencias):



En la pestaña de macros:

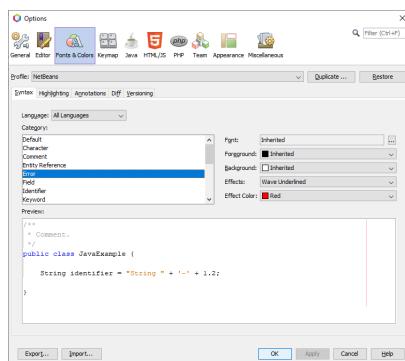


Opción “Fonts & Colors”:



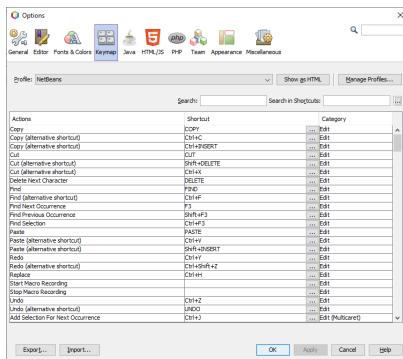
Consiste en elegir el tipo de letra y colores que prefiramos para el texto del código, así como efectos (si es que así lo deseamos).

También podemos configurar el tipo de letra y color de los errores del código (por defecto, de color rojo).



Y lo mismo con los números, espacios en blanco...

Opción “KeyMap”:



En cuanto a los métodos abreviados de teclado (combinación de teclas equivalente a las acciones en NetBeans), podemos modificar aquellas acciones que hagamos con más frecuencia por aquella combinación de teclas que nos sea más fácil recordar.

« Anterior Siguiente »

Anexo V.- Adición de módulos.

Hay muchos programas que se le pueden añadir más funcionalidades instalándoles unos módulos conocidos como plugins.

En este anexo vamos a ver como añadir dichos plugins en NetBeans y en Eclipse.

[« Anterior](#) [Siguiente »](#)

a.- Adición de módulos en NetBeans.

Hay dos formas de añadir módulos y plugins en NetBeans:

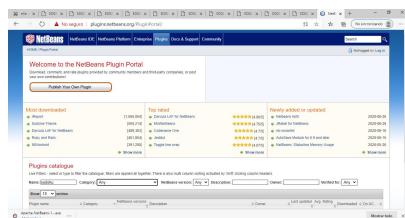
1.- Off-line: Buscar y descargar plugins desde la página web oficial de la plataforma

[Descarga de plugins para NetBeans.](#)

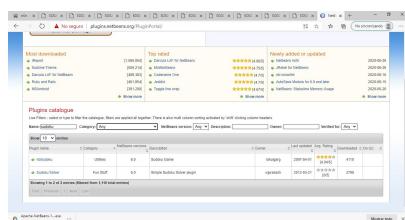
Ejemplo:

Vamos a buscar un plugin para jugar al sudoku desde nuestro IDE. No es muy educativo, pero sirva como ejemplo la manera en que se va a realizar el proceso (será igual en todos los casos):

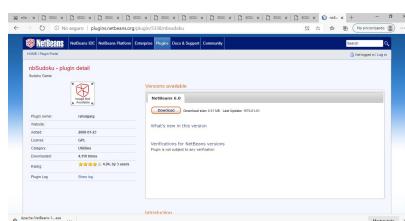
Entramos en la zona de descargas de plugins para NetBeans :



y en la zona del catálogo, escribiremos la palabra sudoku:

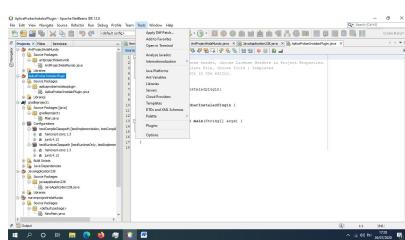


Se nos abre una ventana con las características del plugin y la opción de descargarlo.

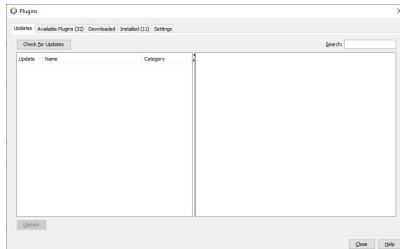


Damos a la opción de descargar

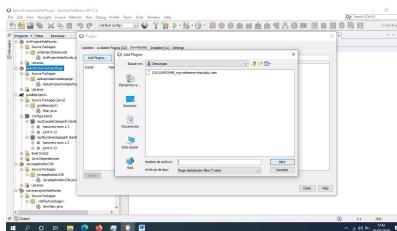
Entramos en NetBeans y pinchamos en Tools:



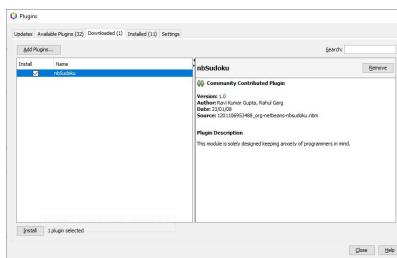
Elegimos la opción Plugins.



En la pestaña "downloaded" (descargado) seleccionamos "Add Plugins" (Aregar Plugins)



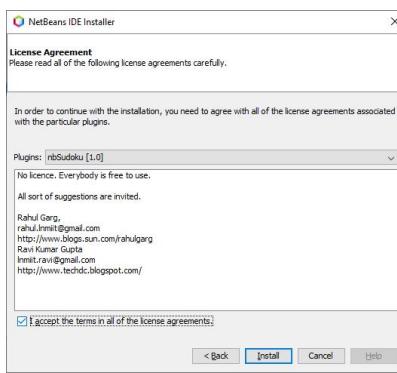
Seleccionamos la carpeta donde habíamos guardado el plugin del sudoku y le damos a "aceptar"



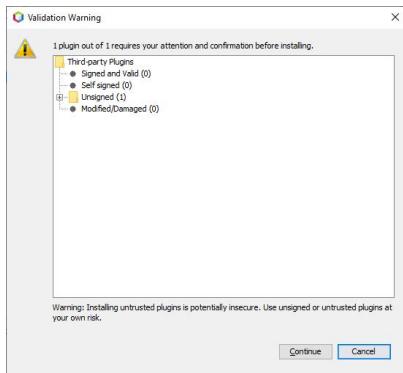
Estando el plugin seleccionado, pulsamos "Install".

Empieza la instalación:

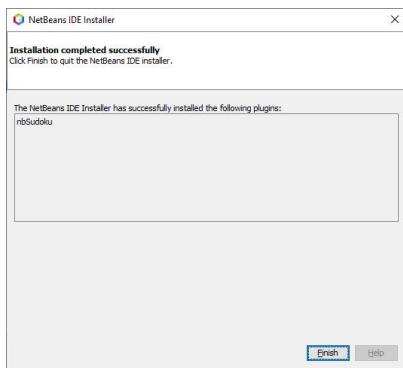
Pulsamos siguiente. Despues, aceptamos la licencia:



Pulsamos "instalar"

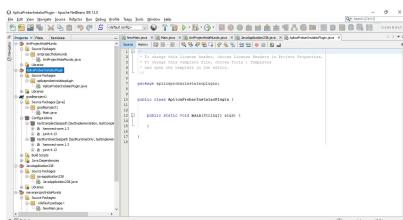


Nos pide una confirmación y se terminará por instalar el plugin.



Seleccionamos "Finish"

Observamos el icono que aparece en la barra de iconos superior del sitio:



Si lo pulsamos, ya podemos jugar un ratito al sudoku para despejarnos:



2.- On-Line: Instalarlos desde el propio entorno de desarrollo:

Se pueden instalar plugins desde NetBeans.

Para ello, vamos a Tools/Plugins. Y pinchamos sobre la pestaña: "Available plugins". Ahí podemos elegir alguno de los plugins que hay e instalarlo.

[« Anterior](#) [Siguiente »](#)

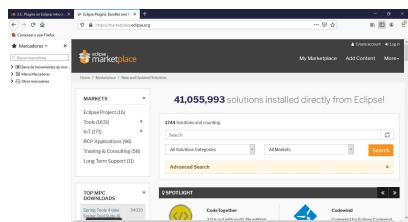
b.- Adición de módulos en Eclipse.

Introducción

Eclipse emplea módulos (**plug-in**) para añadir funcionalidades a las ya existentes, se trata de "un esqueleto" capaz de integrar diferentes paquetes que proporcionarán los servicios que en cada caso el proyecto a desarrollar demande. Por tanto, **Eclipse** se aleja de los diseños monolíticos donde todo queda instalado desde el inicio sea o no necesario.

Existen en la actualidad multitud de módulos disponibles , en este documento veremos dónde encontrar módulos para **Eclipse** que sean de algún interés para nosotros y las distintas formas de instalarlos/desinstalarlos.

El primer lugar a visitar es el "mercado de plugins" de **Eclipse**: <https://marketplace.eclipse.org/>



Instalación de plugins

Existen varias alternativas para instalar un **plugin** en **Eclipse**. Vamos a ver algunas formar de como hacerlo.

Para verlo vamos a instalar tres plugins: Enhanced class decompiler,JBC y UMLet.

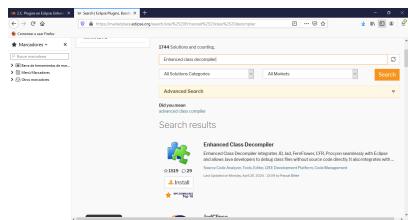
Enhanced class decompiler

El **plugin Enhanced class decompiler** ofrece la posibilidad de descompilar ficheros .class (**bytecode**) para ver su código fuente.

Se encuentra disponible en el **Marketplace de Eclipse** y procederemos a su instalación desde la opción de menú "Ayuda/EclipseMarketPlace del IDE".

Los pasos a seguir son:

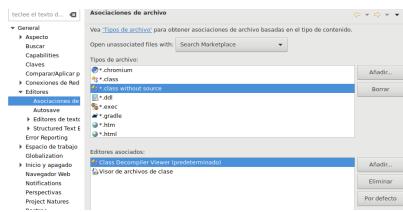
- Indicar el nombre del plugin a instalar y pulsar "Search".
- Una vez localizado, pulsar Install.
- La instalación es guiada y no presenta dificultades. Habrá que aceptar las licencias de instalación y reiniciar Eclipse.



Una vez reiniciado, habrá que seleccionar este **plugin** para mostrar el código descompilado de ficheros con extensión .class.

Para ello aplicaremos la siguiente configuración:

- Mostrar la ventana de preferencias desde la opción de menú "Ventana/Preferencias".
- Seleccionar "General/Editores/Asociaciones de Archivo/*.class without source".
- Marcar el módulo "Class Decompiler Viewer" y pulsar el botón "Por defecto". Finalizar el proceso aplicando los cambios.



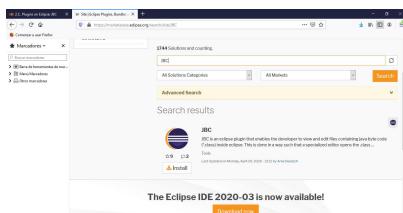
Para comprobar su funcionamiento es necesario acceder a un fichero .class (se obtendrá cuando se compile el código) y abrirlo con el "Class Decompiler Viewer". Cuando compiles tu primer programa obtendrás un fichero .class y podrás probarlo.

Para desinstalar el **plugin**, acudir nuevamente a la opción de menú "Ayuda/EclipseMarketPlace" del **IDE** y seleccionar la pestaña **Installed** donde aparecen los módulos instalados en **Eclipse**. Aquí tendremos la posibilidad de solicitar su cambio-actualización o desinstalación.

JBC

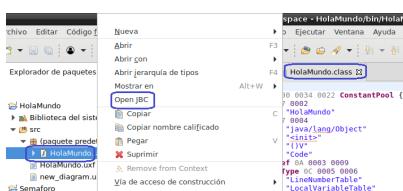
BC (Java ByteCode) permite ver la información compilada de un fichero .class desde el **IDE**.

La instalación se hará nuevamente desde el **MarketPlace de Eclipse**, a través de la opción de menú "Ayuda/EclipseMarketPlace".



Una vez localizado, pulsar el botón **Install**, aceptar los términos de la licencia, proceder con la instalación por defecto y finalmente reiniciar **Eclipse**.

Para comprobar su funcionamiento, pulsar con el botón derecho del ratón sobre un fichero con extensión .java en el "Explorador de paquetes" (vista disponible en la perspectiva Java). Seleccionar la opción **Open JBC** del menú contextual.



Para desinstalar el módulo, acudir nuevamente a la opción de menú "Ayuda/EclipseMarketPlace", pestaña **Installed**, botón **Uninstall**.

UMLet

Durante el curso conoceremos una serie de diagramas de modelado útiles en las fases de análisis y diseño en los desarrollos software. Existe muchas herramientas que nos permite dibujar estos

diagramas, una de ellas es **UMLet**. UMLet está disponible tanto en formato **standalone**, como para ser integrada en otras aplicaciones. Existe un **plugin de UMLet para Eclipse**.

En esta ocasión utilizaremos una técnica distinta para añadir el plugin.

- Descargar de la url <http://www.umlet.com/changes.htm> el **plugin UMLet para Eclipse**.
- Descomprimir y copiar el fichero .jar en la ruta **plugins o dropings de Eclipse**.
- Para iniciar **UMLet**, crea un nuevo proyecto desde la opción de menú "Archivo/Nuevo/Otras/Otro/Umlet diagram".



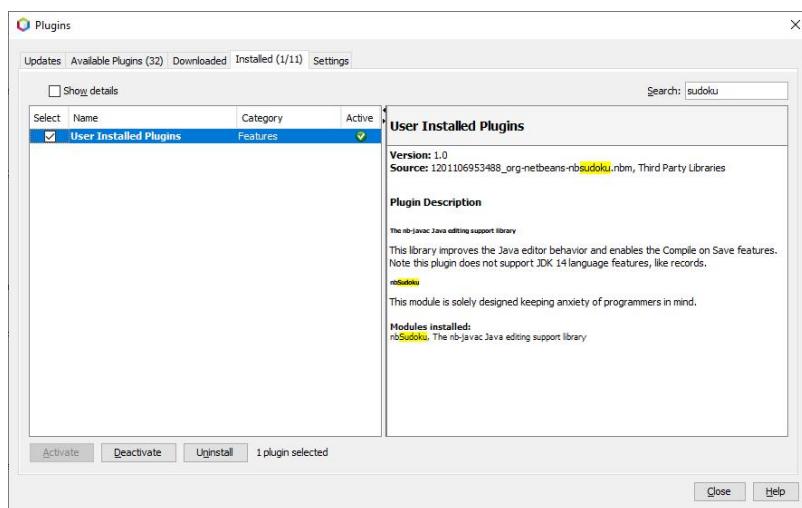
[« Anterior](#) [Siguiente »](#)

Anexo VI.- Eliminar módulos en NetBeans.

Vamos a ver la secuencia de pasos a seguir para eliminar el plugin del juego del sudoku del entorno.

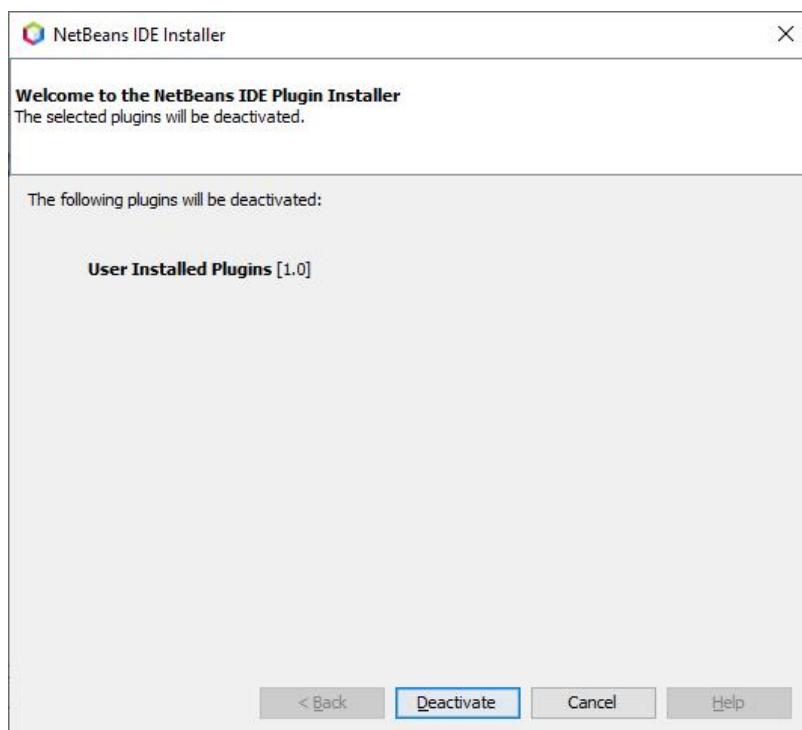
El proceso es muy sencillo: basta con conseguir la lista de complementos instalados (Tools - Plugins). Vamos a la pestaña ("Installed") y localizamos el complemento que queremos eliminar escribiendo su nombre en el lugar destinado para ello y seleccionamos una de entre las dos opciones posibles: desinstalarlo o desactivarlo

En la pestaña de complementos instalados, escribimos el nombre del plugin (sudoku) en la barra de búsqueda:



Cuando lo encuentra, en la ventana aparecen las dos posibilidades de eliminación (desactivarlo o desinstalarlo).

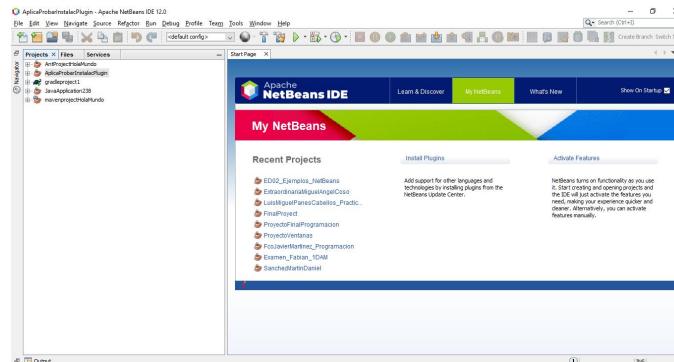
En este caso, hemos optado por desactivarlo.



Anexo VII.- Ejemplo de edición de código.

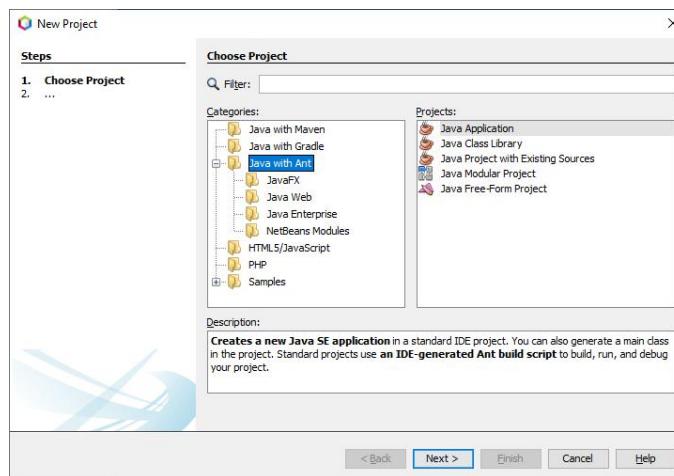
En este documento vamos a introducirnos en la edición de programas en NetBeans a través de un ejemplo sencillo de una aplicación de Java.

Lo primero es iniciar la plataforma:

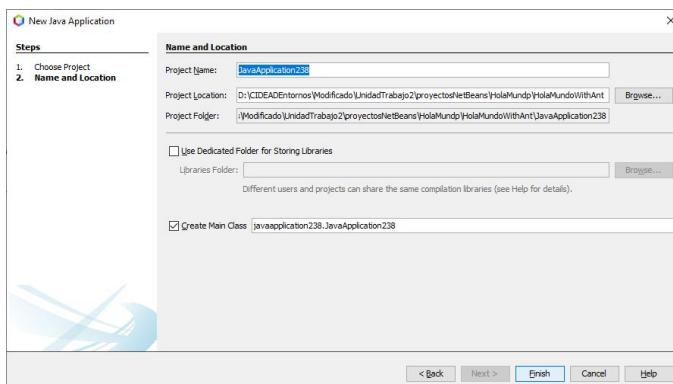


Seleccionamos File- New Project.

Elegimos "Java With Ant" /"Java Application"



Pulsamos sobre siguiente. En la siguiente ventana te pide un nombre (para el proyecto) y su ubicación.



Lo vamos a llamar Ejemplo y ubícalo donde creas conveniente.

Una vez iniciado el proyecto, en la ventana de proyectos (izquierda) vemos cómo se ha cargado el proyecto "Ejemplo". Lo seleccionamos con el ratón y se despliega, mostrando todos sus archivos componentes. Seleccionamos Ejemplo.java (que es el archivo por donde va arrancar el proyecto). Dicho fichero le vamos a editar:

```

 1 /**
 2  * To change this license header, choose License Headers in Project Properties.
 3  * To change this template file, choose Tools | Templates
 4  * and open the template in the editor.
 5  */
 6
 7 package ejemplor;
 8
 9 public class Ejemplo {
10
11     public static void main(String[] args) {
12
13         System.out.println("Hola mundo");
14         System.out.println("Creando mi primer ejemplo");
15     }
16
17 }
  
```

En la ventana de edición (a la derecha) nos aparece el esqueleto de la estructura básica de una aplicación en Java.

Lo que vamos a hacer a lo largo del ejemplo es añadir código.

La primera línea de código que vamos a agregar es una orden sencilla en Java, cuya ejecución posterior dará lugar a la aparición de un mensaje por pantalla.

Añadimos otra línea más con otro mensaje "Creando mi primer ejemplo"

```

 1 /**
 2  * To change this license header, choose License Headers in Project Properties.
 3  * To change this template file, choose Tools | Templates
 4  * and open the template in the editor.
 5  */
 6
 7 package ejemplor;
 8
 9 public class Ejemplo {
10
11     public static void main(String[] args) {
12
13         System.out.println("Hola mundo");
14         System.out.println("Creando mi primer ejemplo");
15     }
16
17 }
  
```

Ahora vamos a modificar la parte de arriba del programa. Añadimos la siguiente línea:

```

 1 /**
 2  * To change this license header, choose License Headers in Project Properties.
 3  * To change this template file, choose Tools | Templates
 4  * and open the template in the editor.
 5  */
 6
 7 package ejemplor;
 8
 9 public class Ejemplo extends JFrame {
10
11     public static void main(String[] args) {
12
13         System.out.println("Hola mundo");
14         System.out.println("Creando mi primer ejemplo");
15     }
16
17 }
  
```

Esta línea nos va a servir para adentrarnos en una de las utilidades más importantes de un entorno de desarrollos.

NetBeans entiende esta orden como un error (aparece subrayada en una línea roja ondulada y con un pequeño icono al lado izquierdo)

Si pulsamos sobre ese ícono con el ratón, NetBeans nos aporta sugerencias para deshacer el error:

The tooltip shows several options related to creating a frame:

- Add Project to Source Packages
- Create class "Frame" in package ejemplor (Source Packages)
- Create Test Class "TestFrame" in Test Packages
- Create Test Class "TestFrame" [link in Test Packages]
- Create Test Class "Frame" [link in Test Packages]
- Create Test Class "Frame" [link in Test Packages]

En este caso, elegimos importar JFrame a la librería.

Y seguimos añadiendo código en el editor:

```

 1  /*
 2   * To change this license header, choose License Headers in Project Properties.
 3   * To change this template file, choose Tools | Templates
 4   * and open the template in the editor.
 5   */
 6
 7  package ejemplo;
 8
 9  import javax.swing.JFrame;
10
11
12  public class Ejemplo extends JFrame {
13
14
15      public static void main(String[] args) {
16
17          System.out.println("Hola mundo");
18          System.out.println("Creando mi primer ejemplo");
19
20          JLabel lblSaludo = new JLabel("Hola Mundo. Creando mi primer ejemplo");
21          add(lblSaludo);
22          this.setSize(600,200);
23          this.setTitle("JFrame");
24          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25
26      }
27
28  }
  
```

Llegados a este punto, ya hemos comprobado que el editor no nos da ningún problema más. En el siguiente punto del tema, veremos cómo ejecutar esto.

Vemos también cómo se han importando con éxito las librerías que nos han hecho falta:

```

 1  /*
 2   * To change this license header, choose License Headers in Project Properties.
 3   * To change this template file, choose Tools | Templates
 4   * and open the template in the editor.
 5   */
 6
 7  package ejemplo;
 8
 9  import javax.swing.JFrame;
10 import javax.swing.JLabel;
11
12
13  public class Ejemplo extends JFrame {
14
15      public Ejemplo() {
16
17          JLabel lblSaludo = new JLabel("Hola Mundo. Creando mi primer ejemplo");
18          add(lblSaludo);
19          this.setSize(600,200);
20          this.setTitle("JFrame");
21          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22          setVisible(true);
23
24      }
25
26      public static void main(String[] args) {
27
28          new Ejemplo();
29
30      }
31  }
  
```

El código completo del ejemplo es el siguiente:

```

package ejemplo;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class Ejemplo extends JFrame{
    public Ejemplo()
    {
        JLabel lblSaludo = new JLabel( "Hola Mundo. Creando mi primer ejemplo");
        add(lblSaludo);
        this.setSize(600,200);
        this.setTitle("JFrame");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);

    }
    public static void main(String[] args) {
        new Ejemplo();
    }
}
  
```

[« Anterior](#) [Siguiente »](#)

Anexo VIII.- Ejecución de un programa en NetBeans.

Continuando con el ejemplo anterior, recuerda que habíamos llegado a este punto:

```

 1 /*
 2  * To change this license header, choose License Headers in Project Properties.
 3  * To change this template file, choose Tools | Templates
 4  * and open the template in the editor.
 5  */
 6 
 7 package ejemplo;
 8 
 9 import javax.swing.JFrame;
10 import javax.swing.JLabel;
11 
12 
13 public class Ejemplo extends JFrame{
14 
15     public Ejemplo()
16     {
17         JLabel lblSaludo = new JLabel("Hola Mundo. Creando mi primer ejemplo");
18         add(lblSaludo);
19         this.setSize(600,200);
20         this.setTitle("Ejemplo");
21         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22         setVisible(true);
23     }
24 
25     public static void main(String[] args) {
26 
27         new Ejemplo();
28     }
29 
30 }
31 
32 
```

Tenemos el programa escrito en el editor libre de errores sintácticos.

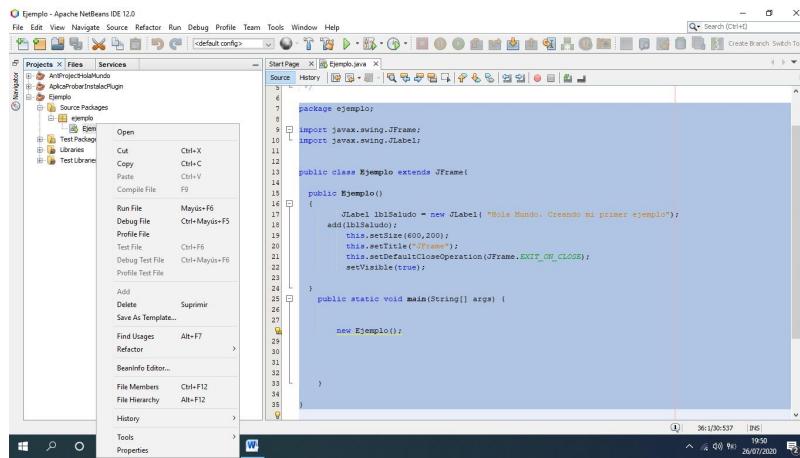
¿Cómo convertir ese programa en ejecutable?

Cabe destacar que, por la sencillez y pequeñez del programa, la ejecución del mismo podría ser directa sin ningún problema.

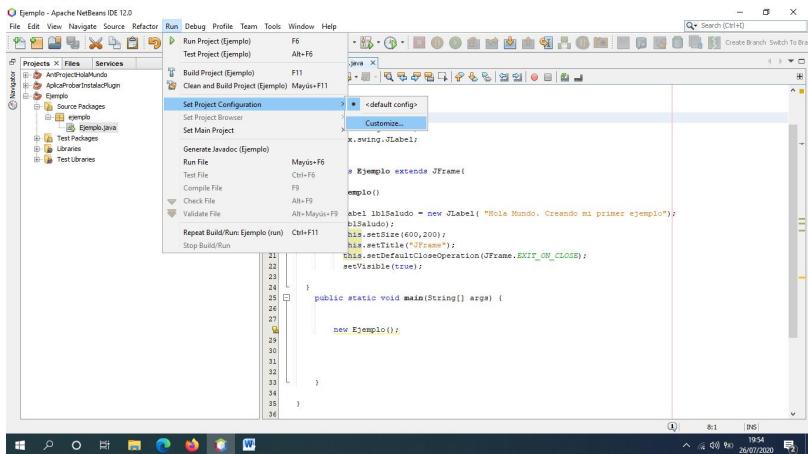
Sin embargo, debemos acostumbrarnos a seguir los pasos adecuados, que son: Editor libre de errores → Compilación → Depuración → Ejecución

Para ejecutar el programa se puede hacer de varias formas:

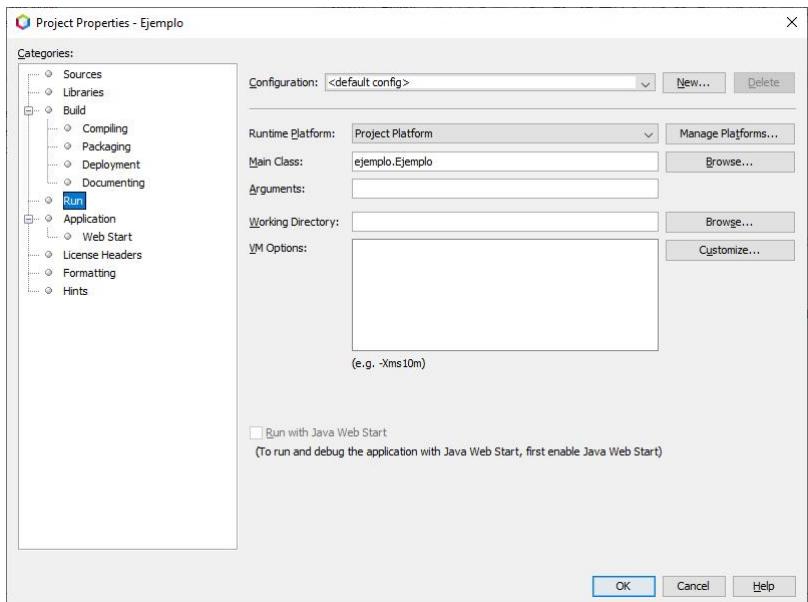
- Sobre el propio fichero, pulsar el botón derecho del ratón y elegir la opción "Run File".



- O pinchar en el menú "Run" y coger la opción "Run project". Si se hace de esta forma, tiene que estar bien configurado el proyecto para indicarle que empiece la ejecución del proyecto desde el fichero que quieres. Esto se hace en: Run/Set project configuration/Customize

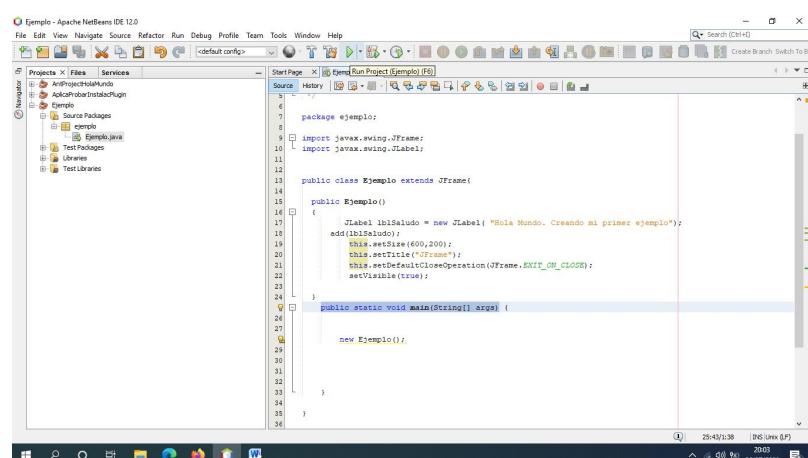


Te visualiza esta ventana:



Donde pone "Main Class" hay que indicar el fichero por el que quieras que empiece la ejecución del programa. Dando a "Browse" podrás elegir el fichero. Sólo te dejaré elegir los ficheros que tengan un método con esta cabecera: "public static void main(String[] args)".

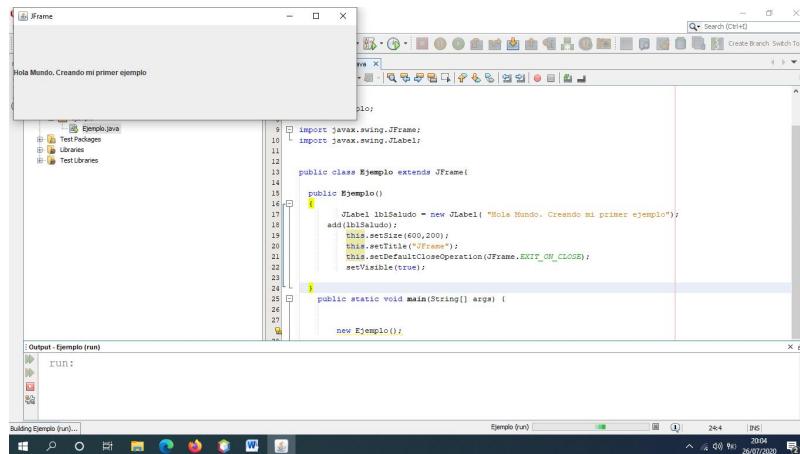
- Mediante el icono de acceso directo en la parte superior de la ventana de edición de código. Es igual que si dieras a Run/Run project. Con lo cual, tiene que estar configurado (como se indicó en su momento) para que en el ejecución del proyecto empiece por el fichero que tú quieras.



- Pulsando F6. Es igual que si dieras a Run/Run project. Con lo cual, tiene que estar configurado (como se indicó en su momento) para que en el ejecución del proyecto empiece por el fichero

que tú quieras.

El resultado que obtenemos (si todo ha ido bien) es:



[« Anterior](#) [Siguiente »](#)

Anexo IX.- Ejemplos completo con Eclipse

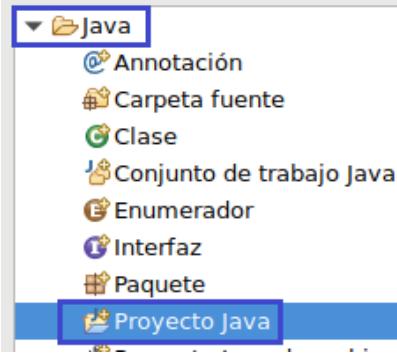
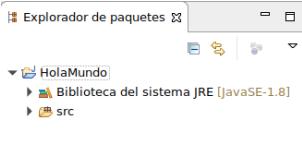
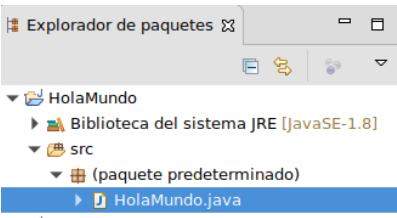
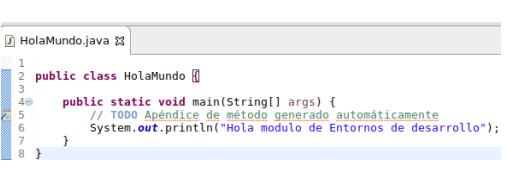
En el módulo **Programación** realizarás gran cantidad de programas, probablemente en **Java** con **Eclipse** o con alguna otra combinación de lenguaje/**IDE**.

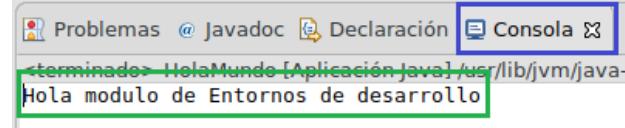
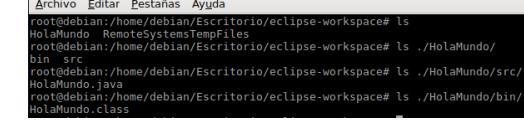
Aquí se presentan un par de sencillos programas para iniciarnos en el manejo de **Eclipse**.

[« Anterior](#) [Siguiente »](#)

a.- Ejemplo "Hola Mundo"

La secuencia para crear un programa Java mediante **Eclipse** es:

<p>Crear un nuevo proyecto Java.</p> <p>Opción de menú "Archivo/Nuevo/Proyecto Java."</p> <p>Si no aparece directamente la opción de proyecto Java probar con "Archivo/Nuevo/Otras/Java/Proyecto Java".</p>	
<p>Introducir el nombre del proyecto.</p> <p>En este caso HolaMundo y pulsar Finalizar.</p>	
<p>En el explorador de paquetes ya aparecerá el nuevo proyecto.</p>	
<p>Crear la clase HolaMundo que incluya el método main de inicio del programa.</p> <p>Botón derecho sobre el proyecto y seleccionar "Nuevo/Clase".</p> <p>Indicar como nombre HolaMundo y seleccionar el check-box para que se cree automáticamente el método main.</p>	
<p>En el apartado anterior se habrá generado el fichero HolaMundo.java en "/src/paquete predeterminado".</p> <p>Pulsa doble click con el ratón para editarlo.</p>	
<p>El fichero ya incluye el método main (anteriormente lo habíamos indicado al crear la clase).</p> <p>Introducir el código que aparece en la línea 6 para mostrar por consola un mensaje de saludo.</p>	 <pre> 1 public class HolaMundo { 2 public static void main(String[] args) { 3 // TODO Apéndice de método generado automáticamente 4 System.out.println("Hola mundo de Entornos de desarrollo"); 5 } 6 } </pre>

<p>La ejecución del programa se puede llevar a cabo de varias formas, una de ellas es desde la opción de menú "Ejecutar/ejecutar". Obtendremos el siguiente resultado.</p>	 <p>The screenshot shows the Eclipse IDE interface with the 'Consola' tab selected. The console window displays the following text: terminados HolaMundo [Aplicación Java] /usr/lib/jvm/java-8-oracle/bin/java -jar /home/debian/Escritorio/eclipse-workspace/HolaMundo.jar Hola modulo de Entornos de desarrollo</p>
<p>Eclipse ha organizado el proyecto como un directorio a partir de la ruta del WorkSpace y una serie de directorios/ficheros colgando de éste.</p>	 <p>The screenshot shows a terminal window with the following command and output: root@debian:/home/debian/Escritorio/eclipse-workspace# ls HolaMundo RemoteSystemsTempFiles root@debian:/home/debian/Escritorio/eclipse-workspace# ls ./HolaMundo/ bin src root@debian:/home/debian/Escritorio/eclipse-workspace# ls ./HolaMundo/src/ HolaMundo.java root@debian:/home/debian/Escritorio/eclipse-workspace# ls ./HolaMundo/bin/ HolaMundo.class</p>



[« Anterior](#) [Siguiente »](#)

b.- Ejemplo "Calculadora"

Ejercicio Propuesto

Esta práctica si se hará usando el entorno de desarrollo "Eclipse".

Tiene que crear un proyecto que se llame "Calculadora"

Crea la clase "Calculadora" y en su método main copia el siguiente código:

```
import java.util.Scanner;
public class Calculadora {
    public static void main(String[] args) {
        Scanner miScan = new Scanner(System.in);
        System.out.println("Calculadora que suma dos numeros enteros");
        System.out.println("Introduce el primer numero");
        String sPrimerNum = miScan.nextLine();
        int iPrimerNum = Integer.parseInt(sPrimerNum);
        System.out.println("Introduce el primer numero");
        String sSegNum = miScan.nextLine();
        int iSegNum = Integer.parseInt(sSegNum);
        int iResul = iPrimerNum + iSegNum;
        System.out.println("El resultado es: " + iResul);
    }
}
```

Compilar el programa y corrige los errores sintácticos. Incluye las librerías que sean necesarias. Ejecuta el programa.

Instalar el plugin ECalculator (calculadora integrada en el IDE), que se encuentra disponible en el Market place de Eclipse. Una vez instalado, aparece disponible como una vista que podrás añadir a tu perspectiva de trabajo.

Comprobar que el programa funciona correctamente haciendo uso del **plugin**. Realizar una suma ejecutando el programa y cotejar su resultado con el plugin.

Comprobar si existen actualizaciones disponibles para el **IDE**. Del listado de actualizaciones pendientes que te aparezca, seleccionar las que consideres oportunas e instalar sus nuevas versiones.

Desinstalar el plugin ECalculator.

[Mostrar retroalimentación](#)

Esta práctica si se hará usando el entorno de desarrollo "Eclipse".

Para ello hay que instalar JDK y Eclipse indicado en el punto 5.1 (instalación del JDK) Y 5.2.2 (instalacion de Eclipse)

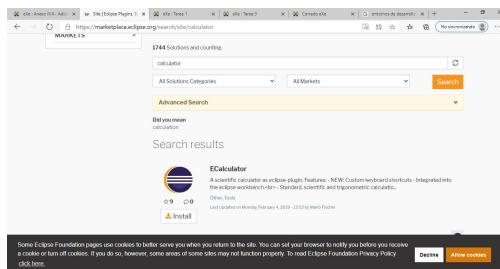
Tiene que crear un proyecto que se llame "Calculadora"

Crea la clase "Calculadora" y en su método main copia el siguiente código:

```
import java.util.Scanner;
public class Calculadora {
    public static void main(String[] args) {
        Scanner miScan = new Scanner(System.in);
        System.out.println("Calculadora que suma dos numeros enteros");
        System.out.println("Introduce el primer numero");
        String sPrimerNum = miScan.nextLine();
        int iPrimerNum = Integer.parseInt(sPrimerNum);
        System.out.println("Introduce el primer numero");
        String sSegNum = miScan.nextLine();
        int iSegNum = Integer.parseInt(sSegNum);
        int iResul = iPrimerNum + iSegNum;
        System.out.println("El resultado es: " + iResul);
    }
}
```

Compilar el programa y corrige los errores sintácticos. Incluye las librerías que sean necesarias. Ejecuta el programa.

Instalar el plugin ECalculator (calculadora integrada en el IDE), que se encuentra disponible en el Market place de Eclipse. Una vez instalado, aparece disponible como una vista que podrás añadir a tu perspectiva de trabajo.



Disponible para instalación en: <https://marketplace.eclipse.org/content/ecalculator>.

Comprobar que el programa funciona correctamente haciendo uso del **plugin**. Realizar una suma ejecutando el programa y cotejar su resultado con el plugin.

Comprobar si existen actualizaciones disponibles para el **IDE**. Del listado de actualizaciones pendientes que te aparezca, seleccionar las que consideres oportunas e instalar sus nuevas versiones.

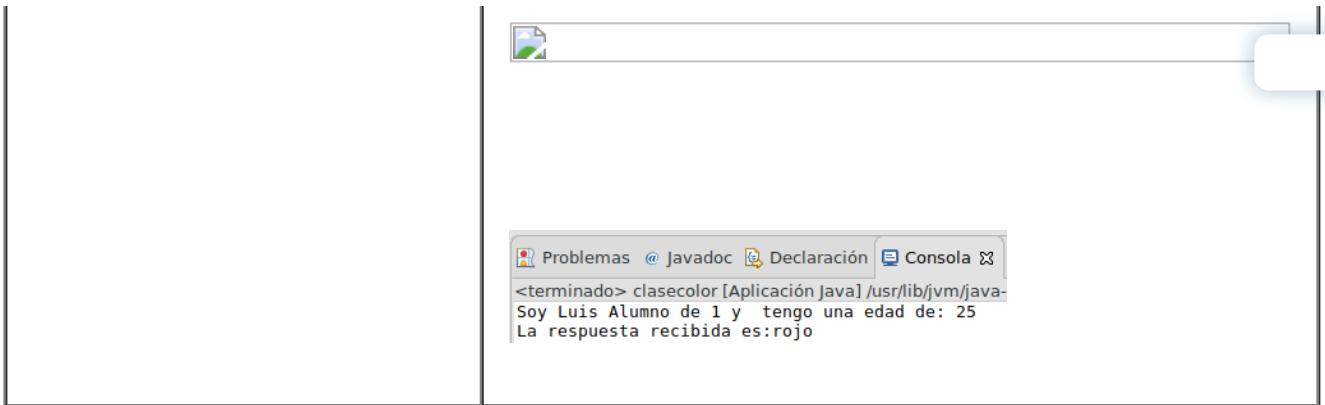
Desinstalar el plugin ECalculator.

« Anterior Siguiente »

c.- Ejemplo "Semáforo"

En el bloque de Actividades y Recursos de esta unidad en el apartado [Recursos disponibles](#) está disponible el fichero semaforo.zip que incluye el código fuente. que está organizado en dos paquetes: principal y clases. En este ejemplo crearemos el proyecto, los dos paquetes e incluiremos el código fuente arrastrando los ficheros java disponibles hasta el **IDE** para integrarlos en el proyecto.

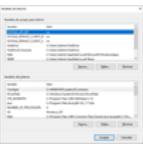
<p>Crear un nuevo proyecto Java Semaforo.</p>	<pre> - Semaforo - src - clases - estudiante.java - ordenador.java - persona.java - profesor.java - principal - clasecolor.java </pre>
<p>Crear los paquetes principal y clases. Sobre el proyecto, botón derecho del ratón y seleccionar "Nueva/Paquete". Una vez por cada paquete.</p>	<pre> - Semaforo - src - clases - principal </pre>
<p>Desde el administrador de archivos, arrastrar los ficheros java a sus respectivos paquetes (principal y clases). Indicando que se haga una copia de los mismos al proyecto.</p>	<pre> - src - clases - estudiante.java - ordenador.java - persona.java - profesor.java - principal - clasecolor.java </pre>
<p>Ejecutamos con la opción de menú "Ejecutar/ejecutar".</p>	

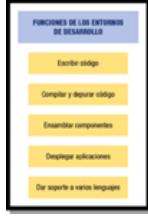


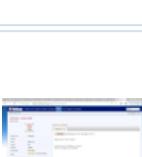
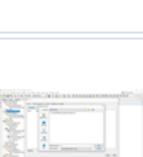
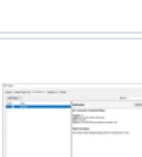
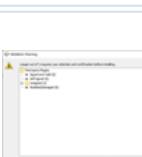
Puede resultar interesante comprobar en el sistema de archivos la estructura de proyecto que se ha generado. Acude a la ruta del proyecto creada en tu **Workspace**.

[« Anterior](#) [Siguiente »](#)

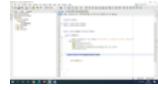
Anexo X.- Licencias de recursos.

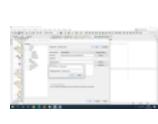
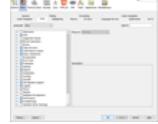
Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: http://netbeans.org		Autoría: jongalloway. Licencia: CC BY-NC-SA 2.0. Procedencia: http://www.flickr.com/photos/jongalloway/2053978954/
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: http://www.eclipse.org/downloads/packages/eclipse-classic-37/indigo Imagen ampliada		Autoría: Adrián Fernández. Licencia: CC-by-nc-sa. Procedencia: http://www.flickr.com/photos/adrian-deejay/442309087/
	Autoría: Francisco Palacios. Licencia: CC by -NC-ND 2.0. Procedencia: http://www.flickr.com/photos/wizard_3303810302/		Autoría: Ubuntu 10.10. Licencia: GNU. Procedencia: http://www.ubuntu.com .
	Autoría: Ubuntu10.10. Licencia: GNU Procedencia: http://www.ubuntu.com .		Autoría: Windows Licencia: GNU Procedencia: http://www.windows.com
	Autoría: Windows Licencia: GNU Procedencia: http://www.windows.com		Autoría: Windows Licencia: GNU Procedencia: http://www.windows.com
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.

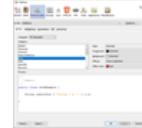
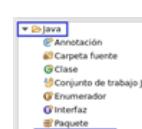
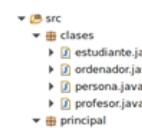
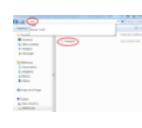
			
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Silveira Neto. Licencia: CC by-sa 2.0. Procedencia: http://www.flickr.com/photos/silveiraneto/2579658422/		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: http://netbeans.org.

			
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbean s. ED02_CONT_R19_JDK-ubuntu 10.pdf Miniatura Comentarios Credenciales del recurso.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>
	<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>		<p>Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.</p>

	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org.		Autoría: netbeans.org.

	Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones.

	Procedencia: Captura de pantalla de Netbeans.		Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones.		Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones.

	Procedencia: Captura de pantalla de Netbeans.		Procedencia: Captura de pantalla de Netbeans.
	Autoría: netbeans.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Netbeans.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org. Licencia: Copyright (cita), se autoriza el uso sin restricciones. Procedencia: Captura de pantalla de Eclipse.
	Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux		Autoría: Windows Licencia: GNU. Procedencia: Captura de pantalla de Windows.
	Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux		Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux
	Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux		Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux

	Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux		Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux
	Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux		Autoría: Linux Licencia: GNU. Procedencia: Captura de pantalla de Linux

[« Anterior](#)

Diseño y realización de pruebas.

Caso práctico



Situación



BK programación se encuentra desarrollando la primera versión de la aplicación de gestión hotelera.

Ada, la supervisora de proyectos de BK Programación, se reúne con Juan y María para empezar a planificar el diseño y realización de pruebas sobre la aplicación,

Ana se va a encargar, de ir probando los distintas partes de la aplicación que se van desarrollando. Para ello usará casos de prueba diseñados por Juan, y evaluará los resultados.

Juan evaluará los resultados de las pruebas realizadas por Ana, y en su caso, realizará las modificaciones necesarias en el proyecto.



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Planificación de las pruebas.

Caso práctico



Todos en la empresa están inmersos en el desarrollo de la aplicación de gestión hotelera. Para garantizar la corrección del desarrollo, Ada propone establecer la planificación de las pruebas. ¿Por qué hay que probar el software? ¿Es necesario seguir un orden concreto en la realización de pruebas? ¿Qué pruebas se realizan?

Durante todo el proceso de desarrollo de software, desde la fase de análisis hasta la implantación en el cliente, es habitual incurrir en errores de varios tipos: incorrecta especificación de los objetivos, errores producidos en el diseño o errores en la fase de desarrollo.

Por lo tanto, se hace necesario hacer un conjunto de **pruebas** que permita comprobar que el producto que se está creando, es correcto y cumple con las especificaciones solicitadas por el usuario.

Las pruebas tratan de **verificar** y **validar** las aplicaciones, entendiendo estos términos como:

- La **verificación** es la comprobación de que un sistema o parte de un sistema, cumple con las condiciones impuestas. Con la verificación se comprueba si la aplicación se está construyendo correctamente.
- La **validación** es el proceso de evaluación del sistema o de uno de sus componentes, para determinar si satisface los requisitos especificados.



Para llevar a cabo el proceso de pruebas, de manera adecuada, se definen estrategias de pruebas.

Siguiendo el **modelo en espiral**, las pruebas empezarán con las **pruebas unitarias** de cada porción de código.

Una vez pasadas estas pruebas con éxito, se seguirá con las **pruebas de integración**, donde se ponen todas las partes del código en común, comprobando que el ensamblado de los bloques de código y sus pruebas atienden a los establecido durante la fase de diseño.

El siguiente paso será la **prueba de validación**, donde se comprueba que el sistema construido cumple con lo establecido en el análisis de requisitos de software.

Finalmente se alcanza la **prueba de sistema** que verifica el funcionamiento total del software y otros elementos del sistema.

Notas:

- El objetivo de las pruebas es conseguir un software libre de errores, por lo tanto la detección de defectos en el software se considera un éxito en esta fase.
- El programador debe evitar probar sus propios programas, ya que aspectos no considerados durante la codificación podrán volver a pasar inadvertidos en las pruebas si son tratados por la misma persona.

2.- Tipos de prueba.

Caso práctico



Juan y María están implementando la mayor parte de la aplicación. ¿Es correcto lo realizado hasta ahora? ¿Cómo se prueba los valores devueltos por una función o método? ¿Es posible seguir la ejecución de un programa, y ver si toma los caminos diseñados?

No existe una clasificación oficial o formal, sobre los diversos tipos de pruebas de software. En la ingeniería del software, nos encontramos con dos enfoques fundamentales:

- **Prueba de la Caja Negra** (Black Box Testing): cuando una aplicación es probada usando su interfaz externa, sin preocuparnos de la implementación de la misma. Aquí lo fundamental es comprobar que los resultados de la ejecución de la aplicación, son los esperados, en función de las entradas que recibe.



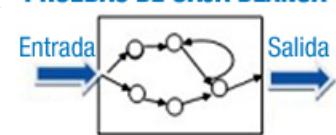
Una prueba de tipo caja negra se lleva a cabo sin tener necesidad de conocer la estructura interna del sistema. Cuando se realiza este tipo de pruebas, sólo se conocen las entradas adecuadas que deberá recibir la aplicación, así como las salidas que les correspondan, pero no se conoce el proceso mediante el cual la aplicación obtiene esos resultados.

En el siguiente ejemplo:

```
if(nota>=5)
    System.out.println("Has suspendido");
else
    System.out.println("Has aprobado");
```

Se detecta un error mediante las pruebas de caja negra. Se entiende que tener una nota superior a 5 debería dar el mensaje de superado y viceversa. Los mensajes están intercambiados.

- **Prueba de la Caja Blanca** (White Box Testing): en este caso, se prueba la aplicación desde dentro, usando su lógica de aplicación.



En contraposición a lo anterior, una prueba de **Caja Blanca**, va a analizar y probar directamente el código de la aplicación, intentando localizar estructuras incorrectas o inefficientes en el código. Como se deriva de lo anterior, para llevar a cabo una prueba de Caja Blanca, es necesario un conocimiento específico del código, para poder analizar los resultados de las pruebas.

A continuación se muestra un código que presenta un error detectable mediante las pruebas de caja blanca:

```
if(nota>=5)
    System.out.println("Has aprobado");
else
    if(nota<5)
        System.out.println("Has suspendido");
    else
        System.out.println("Esta instrucción nunca se ejecutará.");
```

Aunque el programa dará el resultado esperado para cualquier nota (comprobación que se lleva a cabo con las pruebas de caja negra-funcionales), presenta un error en su estructura, el **else** nunca es alcanzado. De este error nos daremos cuenta gracias a las pruebas de caja blanca.

Reflexiona

Resulta habitual, que en una empresa de desarrollo de software se gaste el 40 por ciento del esfuerzo de desarrollo en la prueba ¿Por qué es tan importante la prueba? ¿Qué tipos de errores se intentan solucionar con las pruebas?

[Mostrar retroalimentación](#)

Las pruebas son muy importantes, ya que permiten descubrir errores en un programa, fallos en la implementación, calidad o usabilidad del software, ayudando a garantizar la calidad.

Con las pruebas se intenta verificar que cada componente que se ha diseñado, ya sea un método, función, módulo, etc. realiza la función para la que se ha diseñado. También se intenta comprobar, que existen condiciones en las que todos los caminos de una aplicación llegan a ejecutarse.

2.1.- Funcionales (pruebas de la caja negra)

Estamos ante pruebas de la caja negra. Se trata de probar, si las salidas que devuelve la aplicación, o parte de ella, son las esperadas, en función de los parámetros de entrada que le pasemos. No nos interesa la implementación del software, solo si realiza las funciones que se esperan de él.



Las pruebas funcionales siguen el enfoque de las pruebas de Caja Negra. Comprenderían aquellas actividades cuyo objetivo sea verificar una acción específica o funcional dentro del código de una aplicación. Las pruebas funcionales intentarían responder a las preguntas ¿puede el usuario hacer esto? o ¿funciona esta utilidad de la aplicación?

Su principal cometido, va a consistir, en comprobar el correcto funcionamiento de los componentes de la aplicación informática. Para realizar este tipo de pruebas, se deben analizar las entradas y las salidas de cada componente, verificando que el resultado es el esperado. Solo se van a considerar las entradas y salidas del sistema, sin preocuparnos por la estructura interna del mismo.

Si por ejemplo, estamos implementando una aplicación que realiza un determinado cálculo científico, en el enfoque de las pruebas funcionales, solo nos interesa verificar que ante una determinada entrada a ese programa el resultado de la ejecución del mismo devuelve como resultado los datos esperados. Este tipo de prueba, no consideraría, en ningún caso, el código desarrollado, ni el algoritmo, ni la eficiencia, ni si hay partes del código innecesarias, etc.

Dentro de las pruebas funcionales, podemos indicar tres tipos de pruebas:

Particiones equivalentes: La idea de este tipo de pruebas funcionales, es considerar el menor número posible de casos de pruebas, para ello, cada caso de prueba tiene que abarcar el mayor número posible de entradas diferentes. Lo que se pretende, es crear un conjunto de clases de equivalencia, donde la prueba de un valor representativo de la misma, en cuanto a la verificación de errores, sería extrapolable al que se conseguiría probando cualquier valor de la clase.

Análisis de valores límite: En este caso, a la hora de implementar un caso de prueba, se van a elegir como valores de entrada, aquellos que se encuentra en el límite de las clases de equivalencia.

Pruebas aleatorias: Consiste en generar entradas aleatorias para la aplicación que hay que probar. Se suelen utilizar generadores de prueba, que son capaces de crear un volumen de casos de prueba al azar, con los que será alimentada la aplicación. Esta tipo de pruebas, se suelen utilizar en aplicaciones que no sean interactivas, ya que es muy difícil generar las secuencias de entrada adecuadas de prueba, para entornos interactivos.

Existe otros tipos de pruebas funcionales, aunque todas comparten un mismo objetivo, y es comprobar, solo actuando en la interfaz de la aplicación, que los resultados que produce son los correctos en función de las entradas que se le introducen para probarlos.

2.2.- Pruebas estructurales (pruebas de la caja blanca)

Ya hemos visto que las pruebas funcionales se centran en resultados, en lo que la aplicación hace, pero no en cómo lo hace.

Para ver cómo el programa se va ejecutando, y así comprobar su corrección, se utilizan las pruebas estructurales, que se fijan en los caminos que se pueden recorrer:

Las pruebas estructurales son el conjunto de pruebas de la Caja Blanca. Con este tipo de pruebas, se pretende verificar la estructura interna de cada componente de la aplicación, independientemente de la funcionalidad establecida para el mismo. Este tipo de pruebas, no pretenden comprobar la corrección de los resultados producidos por los distintos componentes, su función es comprobar que se van a ejecutar todas las instrucciones del programa, que no hay código no usado, comprobar que los caminos lógicos del programa se van a recorrer, etc.

Este tipo de pruebas, se basan en unos criterios de cobertura lógica, cuyo cumplimiento determina la mayor o menor seguridad en la detección de errores. Los criterios de cobertura que se siguen son:

Cobertura de sentencias: se han de generar casos de pruebas suficientes para que cada instrucción del programa sea ejecutada, al menos, una vez.

Cobertura de decisiones: se trata de crear los suficientes casos de prueba para que cada opción resultado de una prueba lógica del programa, se evalúe al menos una vez a cierto y otra a falso.

Cobertura de condiciones: se trata de crear los suficientes casos de prueba para que cada elemento de una condición, se evalúe al menos una vez a falso y otra a verdadero.

Cobertura de condiciones y decisiones: consiste en cumplir simultáneamente las dos anteriores.

Cobertura de caminos: es el criterio más importante. Establece que se debe ejecutar al menos una vez cada secuencia de sentencias encadenadas, desde la sentencia inicial del programa, hasta su sentencia final. La ejecución de este conjunto de sentencias, se conoce como camino. Como el número de caminos que puede tener una aplicación, puede ser muy grande, para realizar esta prueba, se reduce el número a lo que se conoce como camino prueba.

Cobertura del camino de prueba: Se pueden realizar dos variantes, una indica que cada bucle se debe ejecutar sólo una vez, ya que hacerlo más veces no aumenta la efectividad de la prueba y otra que recomienda que se pruebe cada bucle tres veces: la primera sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo dos veces.

Autoevaluación

En las pruebas de caja negra:

- Es necesario conocer el código fuente del programa, para realizar las pruebas.
- Se comprueba que todos los caminos del programa, se pueden recorrer, al menos una vez.
- Se comprueba que los resultados de una aplicación, son los esperados para las entradas que se le han proporcionado.
- Es incompatible con la prueba de caja blanca.

Incorrecta, solo interactuamos con la interfaz, no con las instrucciones

No es correcta, esa es una prueba de caja blanca, ya que se comprueba la implementación de los métodos.

Muy bien. Esa es la idea...

No es correcta, ya que cada una tiene una utilidad diferente, y se complementan.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

CUBRIMIENTO.-

Esta tarea la realiza el programador o programadora y consiste en comprobar que los caminos definidos en el código, se pueden llegar a recorrer.

Este tipo de prueba, es de caja blanca, ya que nos vamos a centrar en el código de nuestra aplicación.

Con este tipo de prueba, lo que se pretende, es comprobar que todas las funciones, sentencias, decisiones, y condiciones, se van a ejecutar.

Por ejemplo

Considerando que esta función forma parte de un programa mayor, se considera lo siguiente:

Si durante la ejecución del programa, la función es llamada, al menos una vez, el cubrimiento de la función es satisfactorio.

El cubrimiento de sentencias para esta función, será satisfactorio si es invocada, por ejemplo como prueba(1,1), ya que en esta caso, cada línea de la función se ejecuta, incluida `z=x;`

Si invocamos a la función con prueba(1,1) y prueba(0,1), se satisfará el cubrimiento de decisión. En el primer caso, la if condición va a ser verdadera, se va a ejecutar `z=x`, pero en el segundo caso, no.

El cubrimiento de condición puede satisfacerse si probamos con prueba(1,1), prueba(1,0) y prueba(0,0). En los dos primeros casos (`x>0`) se evalúa a verdad mientras que en el tercero, se evalúa a falso. Al mismo tiempo, el primer caso hace (`y>0`) verdad, mientras el tercero lo hace falso.

```
int prueba (int x, int y)
{
    int z=0;
    if ((x>0) && (y>0))
    {
        z=x;
    }
    return z;
}
```

Existen otra serie de criterios, para comprobar el cubrimiento.

Secuencia lineal de código y salto.

JJ-Path Cubrimiento.

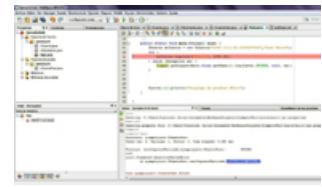
Cubrimiento de entrada y salida.

Existen herramientas comerciales y también de software libre, que permiten realizar la pruebas de cubrimiento, entre ellas, para Java, nos encontramos con Clover.

2.3.- Pruebas de regresión.

Durante el proceso de prueba, tendremos éxito si detectamos un posible fallo o error. La consecuencia directa de ese descubrimiento, supone la modificación del componente donde se ha detectado. Esta modificación, puede generar errores colaterales, que no existían antes. Como consecuencia, la modificación realizada nos obliga a repetir pruebas que hemos realizado con anterioridad.

El objetivo de las pruebas de regresión, es comprobar que los cambios sobre un componente de una aplicación, no introduce un comportamiento no deseado o errores adicionales en otros componentes no modificados.



Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error, como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes.

Normalmente, este tipo de pruebas implica la repetición de las pruebas que ya se hayan realizado previamente, con el fin de asegurar que no se introducen errores que puedan comprometer el funcionamiento de otros componentes que no han sido modificados y confirmar que el sistema funciona correctamente una vez realizados los cambios.

En un contexto más amplio, las pruebas de software con éxito, son aquellas que dan como resultado el descubrimiento de errores. Como consecuencia del descubrimiento de errores, se procede a su corrección, lo que implica la modificación de algún componente del software que se está desarrollando, tanto del programa, de la documentación y de los datos que lo soportan. La prueba de regresión es la que nos ayuda a asegurar que estos cambios no introducen un comportamiento no deseado o errores adicionales. La prueba de regresión se puede hacer manualmente, volviendo a realizar un subconjunto de todos los casos de prueba o utilizando herramientas automáticas.

El conjunto de pruebas de regresión contiene tres clases diferentes de clases de prueba:

Una muestra representativa de pruebas que ejercite todas las funciones del software;

Pruebas adicionales que se centran en las funciones del software que se van a ver probablemente afectadas por el cambio;

Pruebas que se centran en los componentes del software que han cambiado.

Para evitar que el número de pruebas de regresión crezca demasiado, se deben de diseñar para incluir sólo aquellas pruebas que traten una o más clases de errores en cada una de las funciones principales del programa. No es práctico ni eficiente volver a ejecutar cada prueba de cada función del programa después de un cambio.

El diseño de las pruebas de regresión podrá ser una combinación de casos de uso obtenidos desde los enfoques de caja blanca, caja negra y aleatoria.

Autoevaluación

La prueba de regresión:

- Se realiza una vez finalizado cada módulo del sistema a desarrollar.
- Solo utiliza el enfoque de la caja negra.
- Se realiza cuando se produce una modificación, debido a la detección de algún error, en la fase de prueba.
- Es incompatible con la prueba de caja blanca.

Incorrecta, solo si se han producido modificaciones.

Incorrecta. Se trata de volver a realizar las pruebas, y estas pueden ser de cualquier tipo.

Muy bien. Esa es la idea...

No es correcta, ya que cada una tiene una utilidad diferente, y se complementan.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

3.- Procedimientos y casos de prueba.

Caso práctico

En BK, ya tienen el proyecto bastante avanzado. Ahora llega una parte clave: definir las partes del sistema que se van a probar y establecer los casos de prueba para realizarla. Ana va a participar, pero cuando se habla de procedimientos y casos de prueba, se siente perdida. A ella le va a tocar ejecutar los casos de prueba.



Caso práctico



Juan y María prueban cada parte de código que están implementando. Algunos métodos requieren una comprobación de su estructura interna, en otros, valdría con probar los resultados que devuelven. Antonio se pregunta en qué consiste cada prueba, y como se lleva a cabo en la práctica.



La prueba consiste en la ejecución de un programa con el objetivo de encontrar errores. El programa o parte de él, se va a ejecutar bajo unas condiciones previamente especificadas, para una vez observados los resultados, estos sean registrados y evaluados.

Según el IEEE, un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollados para un objetivo particular, como, por ejemplo, ejercitarse un camino concreto de un programa o verificar el cumplimiento de un determinado requisito, incluyendo toda la documentación asociada.

Dada la complejidad de las aplicaciones informáticas, que se desarrollan en la actualidad, es prácticamente imposible, probar todas las combinaciones que se pueden dar dentro de un programa o entre un programa y las aplicaciones que pueden interactuar con él. Por este motivo, en el diseño de los casos de prueba, siempre es necesario asegurar que con ellos se obtiene un nivel aceptable de probabilidad de que se detectarán los errores existentes.

Las pruebas deben buscar un compromiso entre la cantidad de recursos que se consumirán en el proceso de prueba, y la probabilidad obtenida de que se detecten los errores existentes.

Existen varios procedimientos para el diseño de los casos de prueba:

Enfoque funcional o de caja negra. En este tipo de prueba, nos centramos en que el programa, o parte del programa que estamos probando, recibe un entrada de forma adecuada y se produce una salida correcta, así como que la integridad de la información externa se mantiene. La prueba no verifica el proceso, solo los resultados. Este enfoque se centra en las funciones, entradas y salidas. Se aplican los **valores límite** y las **clases de equivalencia**.

Enfoque estructural o caja blanca. En este tipo de pruebas, debemos centrar en la implementación interna del programa. En esta prueba, se deberían de probar todos los caminos que puede seguir la ejecución del programa.

Enfoque aleatorio. A partir de modelos obtenidos estadísticamente, se elaboran casos de prueba que prueben las entradas del programa. Para ello se utilizan generadores automáticos de casos de prueba.

4.- Herramientas de depuración.

Caso práctico



Juan y María tienen muchas líneas de código implementadas. Como programadores de la aplicación, juegan un papel decisivo en la fase de pruebas. Al realizar este proyecto utilizando un entorno de desarrollo integrado (NetBeans), cuenta con una herramienta fundamental, el depurador.

Cada **IDE** incluye herramientas de depuración como: inclusión de puntos de ruptura, ejecución paso a paso de cada instrucción, ejecución por procedimiento, inspección de variables, etc.

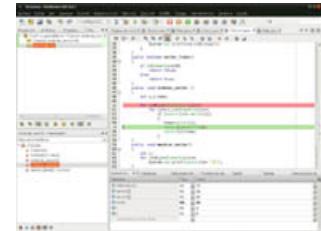
Todo entorno de desarrollo, independientemente de la plataforma, así como del lenguaje de programación utilizado, suministra una serie de herramientas de depuración, que nos permiten verificar el código generado, ayudándonos a realizar pruebas tanto estructurales como funcionales.

Durante el proceso de desarrollo de software, se pueden producir dos tipos de errores: errores de compilación o errores lógicos. Cuando se desarrolla una aplicación en un **IDE**, ya sea Visual Studio, Eclipse o Netbeans, si al escribir una sentencia, olvidamos un ";", hacemos referencia a una variable inexistente o utilizamos una sentencia incorrecta, se produce un **error de codificación**.

Cuando ocurre un error de codificación, el entorno nos proporciona información de donde se produce y como poder solucionarlo. El programa no puede compilarse hasta que el programador o programadora no corrija ese error.

El otro tipo de **errores son lógicos**, comúnmente llamados **bugs**, estos no evitan que el programa se pueda compilar con éxito, ya que no hay errores sintácticos, ni se utilizan variables no declaradas, etc. Sin embargo, los errores lógicos, pueden provocar que el programa devuelva resultados erróneos, que no sean los esperados o pueden provocar que el programa termine antes de tiempo o no termine nunca.

Para solucionar este tipo de problemas, los entornos de desarrollo incorporan una herramienta conocida como **depurador**. El depurador permite supervisar la ejecución de los programas, para localizar y eliminar los errores lógicos. Un programa debe compilarse con éxito para poder utilizarlo en el depurador. El depurador nos permite analizar todo el programa, mientras éste se ejecuta. Permite suspender la ejecución de un programa, examinar y establecer los valores de las variables, comprobar los valores devueltos por un determinado método, el resultado de una comparación lógica o relacional, etc.



Autoevaluación

¿Qué concepto está relacionado con la prueba de caja negra?

- Es la principal herramienta de validación.
- Se pueden comprobar los valores que van tomando las variables
- Se comprueba que todos los caminos del programa, se pueden recorrer, al menos una vez.
- Es incompatible con la prueba de caja blanca.

Correcto. Con las pruebas de caja negra no se depura el programa, se comprueba que devuelve los valores esperados en función de las entradas introducidas.

Incorrecta, solo interactuamos con la interfaz, no con las instrucciones

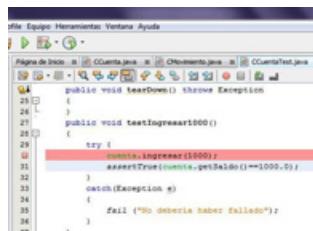
No es correcta, esa es una prueba de caja blanca, ya que se comprueba la implementación de los métodos.

No es correcta, ya que cada una tiene una utilidad diferente, y se complementan.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

4.1.- Puntos de ruptura.



Dentro del menú de depuración, nos encontramos con la opción insertar punto de ruptura (breakpoint). Se selecciona la línea de código donde queremos que el programa se pare, para a partir de ella, inspeccionar variables, o realizar una ejecución paso a paso, para verificar la corrección del código.

Durante la prueba de un programa, puede ser interesante la verificación de determinadas partes del código. No nos interesa probar todo el programa, ya que hemos delimitado el punto concreto donde inspeccionar. Para ello, utilizamos los puntos de ruptura.

Los puntos de ruptura son marcadores que pueden establecerse en cualquier línea de código ejecutable (no sería válido un comentario, o una línea en blanco). Una vez insertado el punto de ruptura, e iniciada la depuración, el programa a evaluar se ejecutaría hasta la línea marcada con el punto de ruptura. En ese momento, se pueden realizar diferentes labores, por un lado, se pueden examinar las variables, y comprobar que los valores que tienen asignados son correctos, o se pueden iniciar una depuración paso a paso, e ir comprobando el camino que toma el programa a partir del punto de ruptura. Una vez realiza la comprobación, podemos abortar el programa, o continuar la ejecución normal del mismo.

Dentro de una aplicación, se pueden insertar varios puntos de ruptura, y se pueden eliminar con la misma facilidad con la que se insertan.

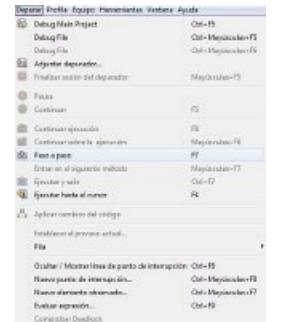
4.2.- Tipos de ejecución.

Para poder depurar un programa, podemos ejecutar el programa de diferentes formas, de manera que en función del problema que queramos solucionar, nos resulte más sencillo un método u otro. Nos encontramos con lo siguientes tipo de ejecución: paso a paso por instrucción, paso a paso por procedimiento, ejecución hasta una instrucción, ejecución de un programa hasta el final del programa,

Algunas veces es necesario ejecutar un programa línea por línea, para buscar y corregir errores lógicos. El **avance paso a paso** a lo largo de una parte del programa puede ayudarnos a verificar que el código de un método se ejecute en forma correcta.

El **paso a paso por procedimientos**, nos permite introducir los parámetro que queremos a un método o función de nuestro programa, pero en vez de ejecutar instrucción por instrucción ese método, nos devuelve su resultado. Es útil, cuando hemos comprobado que un procedimiento funciona correctamente, y no nos interese volver a depurarlo, sólo nos interesa el valor que devuelve.

En la **ejecución hasta una instrucción**, el depurador ejecuta el programa, y se detiene en la instrucción donde se encuentra el cursor, a partir de ese punto, podemos hacer una depuración paso a paso o por procedimiento. En la **ejecución de un programa hasta el final** del programa, ejecutamos las instrucciones de un programa hasta el final, sin detenernos en las instrucciones intermedias.



Los distintos modos de ejecución, se van a ajustar a las necesidades de depuración que tengamos en cada momento. Si hemos probada un método, y sabemos que funciona correctamente, no es necesario realizar una ejecución paso a paso en él.

En el IDE NetBeans, dentro del menú de depuración, podemos seleccionar los modos de ejecución especificados, y algunos más. El objetivo es poder examinar todas la partes que se consideren necesarias, de manera rápida, sencilla y los más clara posible.

4.3.- Examinadores de variables.

Durante el proceso de implementación y prueba de software, una de las maneras más comunes de comprobar que la aplicación funciona de manera adecuada, es comprobar que las variables vayan tomando los valores adecuados en cada momento.

Los examinadores de variables, forman uno de los elementos más importantes del proceso de depuración de un programa. Iniciado el proceso de depuración, normalmente con la ejecución paso a paso, el programa avanza instrucción por instrucción. Al mismo tiempo, las distintas variables, van tomando diferentes valores. Con los examinadores de variables, podemos comprobar los distintos valores que adquiere las variables, así como su tipo. Esta herramienta es de gran utilidad para la detección de errores.

En el caso del entorno de desarrollo NetBeans, nos encontramos con un panel llamado Ventana de Inspección. En la ventana de inspección, se pueden ir agregando todas aquellas variables de las que tengamos interés en inspeccionar su valor. Conforme el programa se vaya ejecutando, NetBeans irá mostrando los valores que toman las variables en al ventana de inspección.

Como podemos apreciar, en una ejecución paso a paso, el programa llega a una función de nombre potencia. Esta función tiene definida tres variables. A lo largo de la ejecución del bucle, vemos como la variable **result**, van cambiando de valor. Si con valores de entrada para los que conocemos el resultado, la función no devuelve el valor esperado, "Examinando las variables" podremos encontrar la instrucción incorrecta.

Nombre	Punto de interrup.	Tipo	Valor
<Escribe el nuevo religio...		Clases	
this		Clases	#46
base		doble	2.0
exponente		doble	3.0

Autoevaluación

¿Qué afirmación sobre depuración es incorrecta?

- En la depuración, podemos inspeccionar las instrucciones que va ejecutando el programa.
- No es posible conocer los valores que toman las variables definidas dentro de un método.
- Solo podemos insertar un punto de ruptura en la depuración

Incorrecta. Podemos ir paso a paso por instrucción, o por procedimiento.

No es correcta. Podemos inspeccionar todas la variables del proyecto, cuando la depuración llegue al método en cuestión, podemos inspeccionar los valores tomados por las variables.

Muy bien. En depuración podemos insertar tantos puntos de ruptura como queremos, y que la depuración vaya saltando de uno a otro.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

5.- Validaciones.

Caso práctico



Durante todo el proceso de desarrollo, existe una permanente comunicación entre Ada y su equipo, con representantes de la empresa a la que va destinado el proyecto. Ana y Juan van a asistir a la siguiente reunión, donde se va a mostrar a los representantes de la empresa, las fases de proyectos ya implementadas. Será la primera reunión de validación del proyecto.

En el proceso de validación, interviene de manera decisiva el cliente. Hay que tener en cuenta, que estamos desarrollando una aplicación para terceros, y que son estos los que deciden si la aplicación se ajusta a los requerimientos establecidos en el análisis.



En la validación intentan descubrir errores, pero desde el punto de vista de los requisitos.

La validación del software se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos.

Un plan de prueba traza la clase de pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos funcionales, que se alcanzan todos los requisitos de rendimiento, que las documentaciones son correctas e inteligible y que se alcanzan otros requisitos, como portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento etc.

Autoevaluación

Durante la validación:

- Procedemos a depurar el programa.
- Sometemos el código a pruebas de cubrimiento.
- Comprobamos que la aplicación cumple los requerimientos del cliente.

Incorrecta.

No es correcta.

En esta prueba, es el cliente, junto con el equipo de desarrollo, quienes comprueban que lo desarrollado cumple las especificaciones establecidas.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

6.- Normas de calidad.

Caso práctico

Las aplicaciones que desarrolla BK Programación, se caracterizan por cumplir todos los estándares de calidad de la industria. Como no podía ser de otro modo, a la hora de realizar las pruebas, también van a seguir los estándares más actuales del mercado. Ada se va a encargar de supervisar el cumplimiento de los estándares más actuales.



Los estándares que se han venido utilizando en la fase de prueba de software son::

Metodología Métrica v3.

Estándares BSI

BS 7925-1, Pruebas de software. Parte 1. Vocabulario.

BS 7925-2, Pruebas de software. Parte 2. Pruebas de los componentes software.

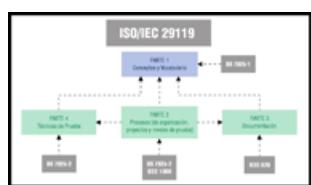
Estándares IEEE de pruebas de software.:

IEEE estándar 829, Documentación de la prueba de software.

IEEE estándar 1008, Pruebas de unidad

Otros estándares ISO / IEC 12207, 15289

Otros estándares sectoriales



Sin embargo, estos estándares no cubren determinadas facetas de la fase de pruebas, como son la organización el proceso y gestión de las pruebas, presentan pocas pruebas funcionales y no funcionales etc. Ante esta problemática, la industria ha desarrollado la norma ISO/IEC 29119.

La norma ISO/IEC 29119 de prueba de software, pretende unificar en una única norma, todos los estándares, de forma que proporcione vocabulario, procesos, documentación y técnicas para cubrir todo el ciclo de vida del software. Desde estrategias de prueba para la organización y políticas de prueba, prueba de proyecto al análisis de casos de prueba, diseño, ejecución e informe. Con este estándar, se podrá realizar cualquier prueba para cualquier proyecto de desarrollo o mantenimiento de software.

La norma ISO/IEC 29119 se compone de las siguientes partes:

- **Parte 1.** Conceptos y vocabulario.
 - Introducción a la prueba.
 - Pruebas basadas en riesgo.
 - Fases de prueba (unidad, integración, sistema, validación) y tipos de prueba (estática, dinámica, no funcional, ...).
 - Prueba en diferentes ciclos de vida del software.
 - Roles y responsabilidades en la prueba.
 - Métricas y medidas.
- **Parte 2.** Procesos de prueba.
 - Política de la organización.
 - Gestión del proyecto de prueba.
 - Procesos de prueba estática.
 - Procesos de prueba dinámica.
- **Parte 3.** Documentación.
 - Contenido.
 - Plantilla.
- **Parte 4.** Técnicas de prueba.
 - Descripción y ejemplos.
 - Estáticas: revisiones, inspecciones, etc.
 - Dinámicas: caja negra, caja blanca, técnicas de prueba no funcional (seguridad, rendimiento, usabilidad, etc).

Autoevaluación

¿Qué norma de calidad intenta unificar los estándares para pruebas de software?

- BS 7925-1.
- IEEE 1008.
- ISO/IEC 29119.

Incorrecta.

No es correcta.

Muy bien.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

7.- Pruebas unitarias.

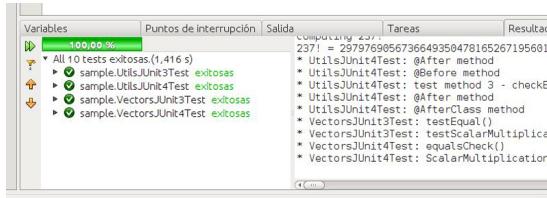
Caso práctico



Antonio está un poco confuso. La aplicación que están diseñando Juan y María es ya de cierta envergadura y se pregunta por la labor ingente que queda, solo para probar todos los componentes de la aplicación. María le tranquiliza y le comenta que los Entornos de Desarrollo actuales, incorporan herramientas que realizan la pruebas de cada método, de forma automática.

Una unidad es la parte de la aplicación más pequeña que se puede probar. En programación procedural, un unidad puede ser una función o procedimiento. En programación orientada a objetos, una unidad es normalmente un método.

Las pruebas unitarias, o prueba de la unidad, tienen por objetivo probar el correcto funcionamiento de un módulo de código. El fin que se persigue, es que cada módulo funciona correctamente por separado, es decir, que cada caso de prueba sea independiente del resto.



Las pruebas de software son parte esencial del ciclo de desarrollo. La elaboración y mantenimiento de unidad, pueden ayudarnos a asegurar que los los métodos individuales de nuestro código, funcionan correctamente. Los entorno de desarrollo, integran frameworks, que permiten automatizar las pruebas.

Posteriormente, con la prueba de integración, se podrá asegurar el correcto funcionamiento del sistema.

En el diseño de los casos de pruebas unitarias, habrá que tener en cuenta los siguientes requisitos:

Automatizable: no debería requerirse una intervención manual.

Completas: deben cubrir la mayor cantidad de código.

Repetibles o Reutilizables: no se deben crear pruebas que sólo puedan ser ejecutadas una sola vez.

Independientes: la ejecución de una prueba no debe afectar a la ejecución de otra.

Profesionales: las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

El objetivo de las pruebas unitarias es aislar cada parte del programa y demostrar que las partes individuales son correctas. Las pruebas individuales nos proporcionan cinco ventajas básicas:

- Fomentan el cambio:** Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- Simplifica la integración:** Puesto que permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.
- Documenta el código:** Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- Separación de la interfaz y la implementación:** Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.
- Los errores están más acotados y son más fáciles de localizar:** dado que tenemos pruebas unitarias que pueden desenmascararlos.

7.1...- Herramientas para Java.

Entre la herramientas que nos podemos encontrar en el mercado, para poder realizar las pruebas, las más destacadas serían:

Jtiger:

- Framework de pruebas unitarias para Java (1.5).
- Es de código abierto.
- Capacidad para exportar informes en **HTML**, **XML** o texto plano.
- Es posible ejecutar casos de prueba de Junit mediante un plugin.
- Posee una completa variedad de aserciones como la comprobación de cumplimiento del contrato en un método.
- Los **metadatos** de los casos de prueba son especificados como anotaciones del lenguaje Java.
- Incluye una tarea de Ant para automatizar las pruebas.
- Documentación muy completa en JavaDoc, y una pagina web con toda la información necesaria para comprender su uso, y utilizarlo con IDE como Eclipse.
- El **Framework** incluye pruebas unitarias sobre sí mismo.



TestNG:

- Esta inspirado en JUnit y NUnit.
- Está diseñado para cubrir todo tipo de pruebas, no solo las unitarias, sino también las funcionales, las de integración ...
- Utiliza las anotaciones de Java 1.5 (desde mucho antes que Junit).
- Es compatible con pruebas de Junit.
- Soporte para el paso de parámetros a los métodos de pruebas.
- Permite la distribución de pruebas en maquinas esclavas.
- Soportado por gran variedad de plug-ins (Eclipse, NetBeans, IDEA ...)
- Los clases de pruebas no necesitan implementar ninguna interfaz ni extender ninguna otra clase.
- Una vez compiladas la pruebas, estas se pueden invocar desde la linea de comandos con una tarea de Ant o con un fichero XML.
- Los métodos de prueba se organizan en grupos (un método puede pertenecer a uno o varios grupos).

Junit:

En el caso de entornos de desarrollo para Java, como NetBeans y Eclipse, nos encontramos con el framework JUnit. JUnit es una herramienta de automatización de pruebas que nos permite de manera rápida y sencilla, elaborar pruebas. La herramienta nos permite diseñar clases de prueba, para cada clase diseñada en nuestra aplicación. Una vez creada las clases de prueba, establecemos los métodos que queremos probar, y para ello diseñamos casos de prueba. Los criterios de creación de casos de prueba, pueden ser muy diversos, y dependerán de lo que queramos probar.

Una vez diseñados los casos de prueba, pasamos a probar la aplicación. La herramienta de automatización, en este caso Junit, nos presentará un informe con los resultados de la prueba (imagen anterior). En función de los resultados, deberemos o no, modificar el código.

Sus características principales son:

- Framework de pruebas unitarias creado por Erich Gamma y Kent Beck.
- Es una herramienta de código abierto.
- Multitud de documentación y ejemplos en la web.
- Se ha convertido en el estándar de hecho para las pruebas unitarias en Java.
- Soportado por la mayoría de los IDE como eclipse o Netbeans.
- Es una implementación de la arquitectura xUnit para los frameworks de pruebas unitarias.
- Posee una comunidad mucho mayor que el resto de los frameworks de pruebas en Java.
- Soporta múltiples tipos de aserciones.
- Desde la versión 4 utiliza las anotaciones del JDK 1.5 de Java.
- Posibilidad de crear informes en HTML.
- Organización de las pruebas en Suites de pruebas.
- Es la herramienta de pruebas más extendida para el lenguaje Java.
- Los entornos de desarrollo para Java, NetBeans y Eclipse, incorporan un plugin para Junit.

Caso práctico

Juan está realizando pruebas de la unidad, es decir, comprueba el correcto funcionamiento de los métodos que ha implantado. Para ello, utiliza las herramienta de prueba incorporadas en el entorno de

desarrollo. En su caso, ya que está utilizando NetBeans, se decanta por Junit. Ana está muy interesada en conocer esta herramienta, que ayuda notablemente en el proceso de pruebas.



Para saber más

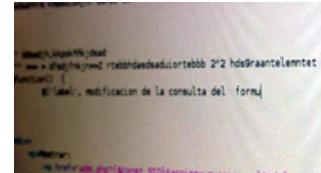
En el siguiente enlace nos encontramos con un ejemplo completo de prueba de la unidad con NetBeans

[Creación de Casos de Prueba en NetBeans con Junit](#)

7.2.- Herramientas para otros lenguajes.

En la actualidad, nos encontramos con un amplio conjunto de herramientas destinadas a la automatización del prueba, para la mayoría de los lenguajes de programación más extendidos en la actualidad. Existen herramientas para C++, para PHP, FoxPro, etc.

Cabe destacar las siguientes herramientas:



CppUnit:

Framework de pruebas unitarias para el lenguaje C++.

Es una herramienta libre.

Existe diversos entornos gráficos para la ejecución de pruebas como QtTestRunner.

Es posible integrarlo con múltiples entornos de desarrollo como Eclipse.

Basado en el diseño de xUnit.

Nunit:

Framework de pruebas unitarias para la plataforma .NET.

Es una herramienta de código abierto.

También está basado en xUnit.

Dispone de diversas expansiones como Nunit.Forms o Nunit.ASP. Junit

SimpleTest: Entorno de pruebas para aplicaciones realizadas en PHP.

PHPUnit: framework para realizar pruebas unitarias en PHP.

FoxUnit: framework OpenSource de pruebas unitarias para Microsoft Visual FoxPro

MOQ: Framework para la creación dinámica de objetos simuladores (mocks).

Autoevaluación

Las herramientas de automatización de pruebas más extendida para Java es:

- Junit.
- FoxUnit.
- Simple Test.

Muy bien. Hay otras herramientas para Java, pero esta es la más popular.

No es correcta. Esta diseñada para Microsoft Visual FoxProFoxUnit

No es correcta. Esta diseñada para PHP

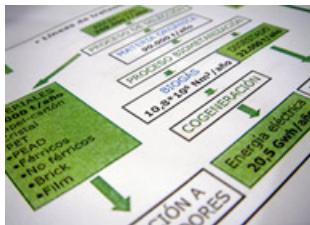
Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

8.- Documentación de la prueba.

Caso práctico

BK Programación, al igual que en todas la fase de diseño de un sistema, en la fase de prueba realiza la documentación. Ada, como coordinadora, le pide a Ana que ayuda a realizar la documentación de la prueba, y le pide que se repase la metodología Métrica v.3 y ayude a María y a Juan en la labor de documentación.



Como en otras etapas y tareas del desarrollo de aplicaciones, la documentación de las pruebas es un requisito indispensable para su correcta realización. Unas pruebas bien documentadas podrán también servir como base de conocimiento para futuras tareas de comprobación.

Las metodologías actuales, como **Métrica v.3**, proponen que la documentación de la fase de pruebas se basen en los estándares ANSI / IEEE sobre verificación y validación de software.

En propósito de los estándares ANSI/IEEE es describir un conjunto de documentos para las pruebas de software. Un documento de pruebas estándar puede facilitar la comunicación entre desarrolladores al suministrar un marco de referencia común. La definición de un documento estándar de prueba puede servir para comprobar que se ha desarrollado todo el proceso de prueba de software.

Los documentos que se van a generar son:

Plan de Pruebas: Al principio se desarrollará una planificación general, que quedará reflejada en el "Plan de Pruebas". El plan de pruebas se inicia el proceso de Análisis del Sistema.

Especificación del diseño de pruebas. De la ampliación y detalle del plan de pruebas, surge el documento "Especificación del diseño de pruebas".

Especificación de un caso de prueba. Los casos de prueba se concretan a partir de la especificación del diseño de pruebas.

Especificación de procedimiento de prueba. Una vez especificado un caso de prueba, será preciso detallar el modo en que van a ser ejecutados cada uno de los casos de prueba, siendo recogido en el documento "Especificación del procedimiento de prueba".

Registro de pruebas. En el "Registro de pruebas" se registrarán los sucesos que tengan lugar durante las pruebas.

Informe de incidente de pruebas. Para cada incidente, defecto detectado, solicitud de mejora, etc, se elaborará un "informe de incidente de pruebas".

Informe sumario de pruebas. Finalmente un "Informe sumario de pruebas" resumirá las actividades de prueba vinculadas a uno o más especificaciones de diseño de pruebas.

Para saber más

En el siguiente enlace podrás visitar la página de Ministerio de Política Territorial y Administración Pública, dedicada a Métrica v.3

[Métrica v.3](#)

Autoevaluación

La documentación de la prueba:

- Es una labor voluntaria que se puede realizar al final del proceso de pruebas
- Cada equipo de pruebas decide qué documenta y cómo.
- En España se usa Métrica v.3

Incorrecta. Hay que documentar cada paso que se dé en el proceso de pruebas.

No es correcta. Hay que seguir alguna metodología de la industria.

Muy bien. Es la metodología más extendida en la actualidad.

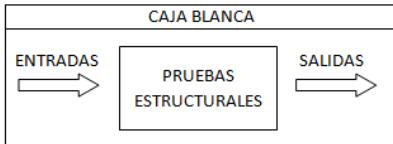
Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

Anexo I.- Pruebas de la caja blanca

Las **pruebas estructurales** son el conjunto de **pruebas de la caja blanca**. Consiste en hacer diferentes pruebas del software con el fin de pasar por todas las líneas de ejecución del programa.

Se trata de plantear casos de prueba viendo el código interno.



Las pruebas estructurales no pretenden asegurar la corrección de los resultados producidos, su función es comprobar que se van a ejecutar todas las instrucciones del programa, que no hay código no usado, comprobar que los caminos lógicos del programa se van a recorrer, etc.

Este tipo de pruebas, se basan en unos criterios de cobertura lógica, cuyo cumplimiento determina la mayor o menor seguridad en la detección de errores.

Los criterios de cobertura son:

- **Cobertura de sentencias:** se han de generar casos de prueba suficientes para que cada instrucción del programa sea ejecutada al menos una vez.
- **Cobertura de decisiones:** se trata de crear los suficientes casos de prueba para que cada resultado de una comprobación lógica del programa, se evalúe al menos una vez a cierto y otra a falso. En la decisión **MIENTRAS (A and B)**, habrá casos de prueba donde (A and B) sea verdadero y donde (A and B) sea falso.
- **Cobertura de condiciones:** se trata de crear los suficientes casos de prueba para que cada elemento de una condición, se evalúe al menos una vez a falso y otra a verdadero. En la decisión **MIENTRAS (A and B)**, habrá casos de prueba donde A sea falso, A sea verdadero, B sea falso y B sea verdadero.
- **Cobertura de condiciones y decisiones:** consiste en cumplir simultáneamente las dos anteriores.
- **Cobertura del camino de prueba:** se pueden realizar dos variantes, una indica que cada bucle se debe ejecutar sólo una vez, ya que hacerlo más veces no aumenta la efectividad de la prueba y otra que recomienda que se pruebe cada bucle tres veces: la primera sin entrar en su interior, otra ejecutándolo una vez y otra más ejecutándolo al menos dos veces.



a.- Pasos a seguir

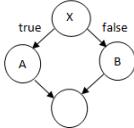
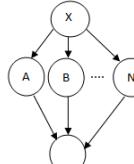
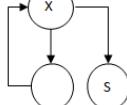
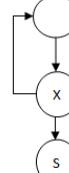
La técnica para determinar los **casos de prueba de caja blanca** que garantiza cobertura de sentencias, decisiones/condiciones y de caminos.

Se realiza completando los siguientes pasos:

- **Crear un grafo** que represente el código a probar.
- Calcular la **complejidad ciclomática o de McCabe** del grafo obtenido.
- Determinar tantos **caminos** (recorridos del grafo) como la complejidad ciclomática calculada.
- Generar un **caso de uso** por cada camino, determinando sus datos de entrada y los resultados esperados.
- Lanzar una **ejecución del programa** por cada caso de uso y **comparar los resultados obtenidos con los esperados** para comprobar la corrección del código.

a.1.- Creación del grafo

Se trata de **crear un grafo** en base al tipo de instrucciones que vayamos encontrando en nuestro código. Los tipos de estructuras principales que aparecen en los programas son secuencias de instrucciones, condiciones e iteraciones. Éstas se representan como sigue en el grafo.

Estructuras básicas	
Secuencia	 <pre>String sNota = "10"; System.out.println("Tu nota es: " + sNota);</pre>
Condición	 <pre>int iNota = 3; if(iNota>= 5) { System.out.println("Enhorabuena. Superado."); } else if (iNota < 5) { System.out.println("La proxima vez sera"); }</pre>
Selección múltiple	 <pre>switch (iNota) { case 1: case 2: case 3: case 4: { System.out.println("La proxima vez sera"); break; } default: { System.out.println("Enhorabuena. Superado."); }</pre>
Iteración	 <pre>int iNumSal = 2; while(iNumSal > 0) { System.out.println("Hola !!!!!"); iNumSal--; }</pre>
Do Iteración	 <pre>int iNumSal = 2; do { System.out.println("Hola !!!!!"); iNumSal--; } while(iNumSal > 0);</pre>

Los **grafos se construyen a partir de nodos y aristas**. Los nodos representan secuencias de instrucciones consecutivas donde no hay alternativas en la ejecución o condiciones a evaluar, que en función del resultado hará que la ejecución siga una dirección u otra.

Las aristas son las encargadas de unir los nodos.

En el caso de que las decisiones tengan múltiples condiciones, habrá que separar cada condición en un nodo como sigue:

Estructuras de decisión compuestas	
And (&&)	

	<pre> int iNota = 6; if (iNota == 5 iNota > 5) { System.out.println("Enhorabuena. Superado."); } else { System.out.println("La proxima vez sera"); } </pre>
Or ()	<pre> int iNota = 6; if (iNota == 5 iNota > 5) { System.out.println("Enhorabuena. Superado."); } else { System.out.println("La proxima vez sera"); } </pre>

Algunos **consejos útiles** al crear grafos son:

- Separar todas las condiciones.
- Agrupar sentencias 'simples' en bloques.
- Numerar todos los bloques de sentencias y también las condiciones.

a.2.- Complejidad de McCabe o ciclomática

A partir del grafo se determina su **complejidad ciclomática**. Es posible hacerlo por tres métodos diferentes, pero todos ellos han de dar el mismo resultado.

- $V(G) = a - n + 2$, siendo a el número de arcos o aristas del grafo y n el número de nodos.
- $V(G) = r$, siendo r el número de regiones cerradas del grafo (incluida la externa).
- $V(G) = c + 1$, siendo c el número de nodos de condición.

a.3.- Caminos de prueba

El número de **caminos de prueba** debe ser igual a la complejidad calculada. Consiste en hacer recorridos desde el inicio hasta el final del método con el que se esté trabajando. Se irán registrando los nodos por los que va pasando la ejecución del camino.

Cada nuevo camino deberá aportar el paso por nuevas aristas/nodos del grafo. La definición de caminos se hará desde los más sencillos a los más complicados.

a.4.- Casos de uso, resultados esperados y análisis

Ahora toca definir los datos de entrada al programa que nos permitan recorrer los caminos definidos en el apartado anterior. El conjunto de entradas al programa utilizados en la ejecución de cada camino se denomina **caso de uso**.

Además habrá que definir los **resultados previstos**. Cuando posteriormente se lance la ejecución del programa para cada caso de uso, los resultados obtenidos serán comparados con los esperados y así determinar la corrección del código.

b.- Ejemplo 1. Fibonacci

El siguiente programa **java**:

```

1 package seriefibonaccicajablanca;
2 import java.util.Scanner;
3 public class SerieFibonacciCajaBlanca {
4     public static void main(String[] args) {
5         SerieFibonacciCajaBlanca misCal = new SerieFibonacciCajaBlanca();
6         misCal.Fibonacci();
7     }
8     public void Fibonacci() {
9         Scanner miScan = new Scanner(System.in);
10        System.out.print("¿Quiere salir del programa?: ");
11        String sSalir = miScan.nextLine();
12        int iValor = 0;
13        String sResultado;
14        String sAux;
15        while (!(sSalir.equals("S") || sSalir.equals("s"))) {
16            sResultado = "";
17            System.out.print("\n\t\tCuantos numeros de la serie deseas mostrar?: ");
18            sAux = miScan.nextLine();
19            iValor = Integer.parseInt(sAux);
20            switch (iValor) {
21                case 3:
22                    sResultado = " 1";
23                case 2:
24                    sResultado = " 1" + sResultado;
25                case 1:
26                    sResultado = " 0" + sResultado;
27                    System.out.println("\t\tLos " + iValor + " primeros numeros de la serie de Fibonacci son: " + sResultado);
28                    break;
29                default:
30                    System.out.println("\t\tNúmero no permitido. Tiene que estar entre 1 y 3.");
31            }
32            System.out.print("\n\t\t¿Quiere salir del programa?: ");
33            sSalir = miScan.nextLine();
34        }
35    }
36 }
```

Hace el cálculo de la **serie Fibonacci** y muestra el resultado por pantalla. El programa visualizará tantos dígitos de la serie como se indique por el teclado, siendo tres el número más alto que se puede indicar.

La serie de Fibonacci, comienza por el cero, sigue por el uno, y los siguientes números se van calculando como la suma de los dos anteriores, es decir: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.

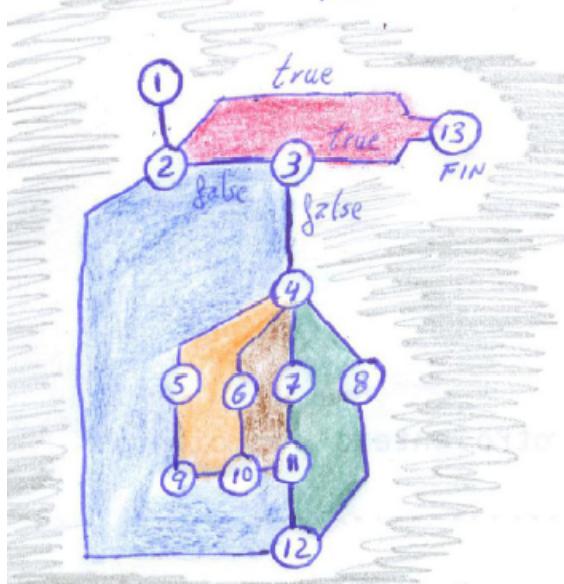
Con lo cual:

- Si el usuario inserta un 1, se visualizará: "0"
- Si el usuario inserta un 2, se visualizará: "0 1"
- Si el usuario inserta un 3, se visualizará: "0 1 1"

Una vez visualizada la serie, podrá insertar otro número el usuario hasta que inserte una "S" o una "s" indicando que quiere salir del programa.

Pulsa [aquí](#) para descargarte este proyecto y poder hacer pruebas ("SerieFibonacciCajaBlanca.zip")..

b.1.- Creación del grafo

Grafo	Nodo	Línea - Condición
	1	Lin: 9-14.
	2	Lin: 15. Cond. "==" S"
	3	Lin: 15. Cond. "==" s"
	4	Lin 16-20. Sentencia Switch
	5	Lin 21. iValor == 3
	6	Lin 23. iValor == 2
	7	Lin 25. iValor == 1
	8	Lin 29-30. iValor != anteriores. "Número no permitido. Tiene que estar entre 1 y 3."
	9	Lin 22.
	10	Lin 24.
	11	Lin 26.
	12	Lin 32-33.
	13	Lin 36.

Nota: los nodos 5 y 9 podrían haberse agrupado en uno sólo. Tanto juntos como por separado, el número de caminos, casos de uso y complejidad ciclomática es el mismo.

Lo mismo sucede con los nodos 6-10 y 7-11.

b.2.- Complejidad de McCabe o ciclomática

Calculada por los tres métodos posibles. Todos ellos deben dar el mismo resultado.

Método de cálculo	Complejidad	Comentarios
Nº de regiones	6	Hay que considerar la región exterior.
Nº de aristas - Nº nodos + 2	$16 - 12 + 2 = 6$	
Nº de condiciones + 1	$5 + 1 = 6$	Nodos 2, 5, 6, 7 y 8

b.3.- Caminos de prueba

El número de caminos de prueba debe ser igual a la complejidad calculada. Cada nuevo camino deberá aportar el paso por nuevas aristas/nodos del grafo. La definición de caminos se hará desde los más sencillos a los más complicados.

- Camino 1: **1 - 2 - 13.**
- Camino 2: **1 - 2 - 3 - 13.**
- Camino 3: **1 - 2 - 3 - 4 - 5 - 9 - 10 - 11 - 12 - 2 - 13.**
- Camino 4: **1 - 2 - 3 - 4 - 6 - 10 - 11 - 12 - 2 - 13.**
- Camino 5: **1 - 2 - 3 - 4 - 7 - 11 - 12 - 2 - 13.**
- Camino 6: **1 - 2 - 3 - 4 - 8 - 12 - 2 - 13.**

b.4.- Casos de uso, resultados esperados y análisis

Caminos/ Casos de uso	Datos		Salidas
	sSalir	sAux	
1	"s"	Indistinto	Fin del programa.
2	"S"	Indistinto	Fin del programa.
3	Cualquiera diferente de 'S' y 's'	3	"Los 3 primeros número de la serie de Fibonacci son: 0 1 1"
4	Cualquiera diferente de 'S' y 's'	2	"Los 2 primeros número de la serie de Fibonacci son: 0 1"
5	Cualquiera diferente de 'S' y 's'	1	"Los 1 primeros número de la serie de Fibonacci son: 0"
6	Cualquiera diferente de 'S' y 's'	4	Número no permitido. Tiene que estar entre 1 y 3.

c.- Ejemplo 2. "Tablero para jugar al bingo"

El siguiente programa java:

```

1 | public class Tablero
2 |
3 |     String tab[][];
4 |     String sNombre;
5 |
6 |     public String PintTab(char cTipo, int iFil, int iCols, String sNomb)
7 |     {
8 |         tab = new String[iFil][iCols];
9 |         sNombre = "";
10 |        int iNumPos = 0;
11 |
12 |        if (cTipo == 'T') {
13 |            sNombre = sNomb;
14 |        } else if (cTipo == 'D') {
15 |            sNombre = "CARTON";
16 |        } else if (cTipo == 'B') {
17 |            sNombre = "";
18 |        }
19 |        for (int i=0; i < iFil; i++)
20 |        {
21 |            for (int j=0; j<iCols; j++)
22 |            {
23 |                iNumPos++;
24 |                tab[i][j] = "";
25 |            }
26 |        }
27 |
28 |        return sNombre + " tiene " + iNumPos + " posiciones";
29 |    }
30 |

```

Para poder probar el método PintTab, será necesario escribir un método main que haga uso de él. Se proporciona el siguiente código de ejemplo para poder probar el método PintTab:

```

public class Test {

    public static void main(String[] args) {

        Tablero tablero=new Tablero();

        System.out.println(tablero.PintTab('T', 4, 5, "Mario"));

    }

}

```

Se crearía una clase llamada Test en el mismo paquete y se usaría esta clase para probar el método PintTab. En este ejemplo se prueba con los valores 'T',4,5 y "Mario"

Este programa crea un tablero/cartón para jugar al bingo. Además cada cartón tendrá un nombre que dependiendo del valor cTipo será:

- 'D' - "CARTON".
- 'T' - El nombre será el valor del parámetro sNomb,
- 'B' - "" (vacío).

Y devuelve el número de posiciones que tiene el tablero y su nombre.

Pulsa [aquí](#) para descargarte dicho proyecto ("TableroBingoCajaBlanca.zip")..

c.1.- Creación del grafo

Grafo	Nodo	Línea - Condición
	1	Lin. 3 - 11.
	2	Lin. 12. Cond. cTipo == "T"
	3	Lin. 14. Cond. cTipo == "D"
	4	Lin. 16. Cond. cTipo == "D"
	5	Lin. 13
	6	Lin. 15
	7	Lin. 17
	8	Lin. 18. Toda condición if termina ahí.
	9	Lin. 19. Cond. i < iFil
	10	Lin. 21. Cond. < iCols
	11	Lin. 23-24
	12	Lin. 25
	13	Lin. 28-29

Notas:

- La vuelta a los for (nodos 9 y 10), se hace desde los nodos 12 y 11 respectivamente. Por otra parte, cuando la condición de los bucles no se cumple, la salida se realiza por las arista 9-13 y 10-12 en cada caso.
- El paso de los nodos 4 a 8 no aparece reflejado de forma explícita en el código

c.2.- Complejidad de McCabe o ciclomática

Calculada por los tres métodos posibles. Todos ellos deben dar el mismo resultado.

Método de cálculo	Complejidad	Comentarios
Nº de regiones	6	Hay que considerar la región exterior.
Nº de aristas - Nº nodos + 2	$17 - 13 + 2 = 6$	
Nº de condiciones + 1	$5 + 1 = 6$	Nodos 2, 3, 4, 9 y 10.

c.3.- Caminos de prueba

El número de caminos de prueba debe ser igual a la complejidad calculada. Cada nuevo camino deberá aportar el paso por nuevas aristas/nodos del grafo. La definición de caminos se hará desde los más sencillos a los más complicados.

- Camino 1: **1 - 2 - 5 - 8 - 9 - 13.**
- Camino 2: **1 - 2 - 3 - 6 - 8 - 9 - 13.**
- Camino 3: **1 - 2 - 3 - 4 - 7 - 8 - 9 - 13.**
- Camino 4: **1 - 2 - 3 - 4 - 8 - 9 - 13. Pasa por arista 4-8.**
- Camino 5: **1 - 2 - 5 - 8 - 9 - 10 - 12 - 9 - 13.**
- Camino 6: **1 - 2 - 5 - 8 - 9 - 10 - 11 - 10 - 12 - 9 - 13.**

c.4.- Casos de uso, resultados esperados y análisis

Caminos/ Casos de uso	Datos				Salidas
	cTipo	iFilas	iColumn	sNombre	
1	T	0	0	Valeria	Valeria tiene 0 posiciones
2	D	0	0	Noa	CARTON tiene 0 posiciones
3	B	0	0	Mila	tienen 0 posiciones.
4	S	0	0	Miguel	tienen 0 posiciones.
5	T	1	0	Agus	Agus tiene 0 posiciones.
6	T	1	1	Raquel	Raquel tiene 1 posiciones

Anexo II.- Pruebas de la caja negra

El propósito de las **pruebas de caja negra o funcionales** es comprobar si las salidas que devuelve la aplicación son las esperadas en función de los parámetros de entrada.



Este tipo de prueba, no consideraría, en ningún caso, el código desarrollado, ni el algoritmo, ni la eficiencia, ni si hay partes del código innecesarias, etc. Estos aspectos son comprobados en las pruebas de caja blanca.

Cómo ya se explicó anteriormente, dentro de las pruebas funcionales, podemos indicar los siguientes **tipos**:

- **Particiones equivalentes:** la idea de este tipo de pruebas funcionales, es considerar el menor número posible de casos de pruebas, para ello, cada caso de prueba tiene que abarcar el mayor número posible de entradas diferentes. Lo que se pretende, es crear un conjunto de clases de equivalencia, donde la prueba de un valor representativo de la misma, en cuanto a la verificación de errores, sería extrapolable al que se conseguiría probando cualquier valor de la clase.
- **Análisis de valores límite:** en este caso, a la hora de implementar un caso de prueba, se van a elegir como valores de entrada aquellos que se encuentra en el límite de las clases de equivalencia.
- **Pruebas aleatorias:** consiste en generar entradas aleatorias para la aplicación que hay que probar. Se suelen utilizar generadores de prueba, que son capaces de crear un volumen de casos de prueba al azar, con los que será alimentada la aplicación.
- **Conjetura de errores:** trata de generar casos de prueba que la experiencia ha demostrado generan típicamente errores. En valores numéricos, un buen ejemplo es comprobar si funciona correctamente con el valor 0, ya que si es utilizado como denominador en alguna división podría generar un error en nuestro programa.

a.- Pasos a seguir

La técnica para determinar los casos de prueba de caja negra se realiza completando los siguientes pasos:

- Determinar **las clases de equivalencia**.
- Determinar un **análisis de valores límite**.
- **Conjetura de errores**.
- Generar los **caso de uso** necesarios para probar las clases válidas y no válidas. Establecer los datos de entrada y los resultados esperados.
- Lanzar una **ejecución del programa** por cada caso de uso y **comparar los resultados obtenidos con los esperados** para determinar la corrección del código.

a.1.- Determinar las clases de equivalencia

La técnica de **clases de equivalencia** es un tipo de prueba funcional, donde en cada caso de prueba se agrupa el mayor número de entradas posibles. A partir de aquí, se asume que la prueba de un valor representativo de cada clase, permite suponer que el resultado que se obtiene con él, será el mismo que con cualquier otro valor de la clase.

Los **pasos** a seguir para identificar las clases de equivalencia son:

- **Identificar las condiciones de las entradas del programa**, es decir, restricciones de formato o contenido de los datos de entrada.
- A partir de ellas, **identificar clases de equivalencia** que pueden ser:
 - De datos válidos.
 - De datos no válidos o erróneos.

Existen algunas **reglas** que ayudan a identificar las clases:

Tipo de dato	Ejemplo	Clases equivalencia
Rango de valores de entrada. Crear una clase válida y dos clases no válidas.	La edad de acceso a un evento está comprendida entre 18 y 100 años.	<p>Clase válida:</p> <p>Valor entre 18 - 100</p> <p>Clases no válidas:</p> <p>Menor de 18.</p> <p>Mayor de 100.</p>
Número finito y consecutivo de valores. Creará una clase válida y dos no válidas.	Una encuesta puede ser valorada con los valores 0, 1, 2, 3.	<p>Clase válida:</p> <p>Cualquiera de los valores 0,1,2,3</p> <p>Clases no válidas:</p> <p>Menor de 0.</p> <p>Mayor de 3.</p>
Condición verdadero/falso .	Una persona tiene la condición de ser mayor de edad.	<p>Clase válida:</p> <p>Edad ≥ 18</p> <p>Clase no válida:</p> <p>Edad < 18</p>
Conjunto de valores admitidos. Se identifica una clase válida por cada valor y una no válida.	Una opción de menú puede aceptar los valores 'A' para altas, 'B' para bajas y 'S' para salir del programa.	<p>Clases validas:</p> <p>Opción 'A'</p> <p>Opción 'B'</p> <p>Opción 'S'</p> <p>Clase no válida:</p> <p>Opción 'J'</p>

En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores.

a.2.- Análisis de valores límite

La experiencia indica que los casos de prueba que exploran las condiciones límite de un programa producen un mejor resultado para detectar defectos.

El **AVL (Análisis de valores límite)** es una técnica de diseño de casos de prueba que complementa a la de particiones de equivalencia.

La principal diferencia se encuentra en el tratamiento que tienen las clases de equivalencia de rango de valores y de número finito y consecutivo de valores. Ahora la prueba se realizará sobre los valores límite de los rangos.

Ejemplo	Clase de equivalencia	Valores límite
La edad de acceso a un evento está comprendida entre 18 y 100 años.	Clase valida:	Clases válidas:
	Valor entre 18 - 100. Caso único. P.e 30	Caso 1: 18 Caso 2: 100
	Clases no válidas:	Clases no válidas:
	Menor de 18. P.e 15 Mayor de 100. P.e. 110	Menor de 18. 17 Mayor de 100. 101
Una encuesta puede ser valorada con los valores 0, 1, 2, 3.	Clase valida:	Clases válidas:
	Cualquier de los valores 0,1,2,3 Caso único. P.e 2	Caso 1: 0 Caso 2: 3
	Clases no válidas:	Clases no válidas:
	Menor de 0. P.e -10 Mayor de 3. P.e 7	Valor -1 Valor 4

En las pruebas AVL también habría que generar casos de prueba atendiendo a clases de equivalencia de los datos de salida.

```
public double funcion1 (double x)
{
    if (x > 5)
        return x;
    else
        return -1;
}

public int funcion2 (int x)
{
    if (x > 5)
        return x;
    else
        return -1;
}
```

En el código Java adjunto, aparecen dos funciones que reciben el parámetro x. En la **función1**, el parámetro es de tipo real y en la **función2**, el parámetro es de tipo entero.

Como se aprecia, el código de las dos funciones es el mismo, sin embargo, los casos de prueba con valores límite va a ser diferente.

La experiencia ha demostrado que los casos de prueba que obtienen una mayor probabilidad de éxito, son aquellos que trabajan con valores límite.

Esta técnica, se suele utilizar como complementaria de las particiones equivalentes, pero se diferencia, en que se suelen seleccionar, no un conjunto de valores, sino unos pocos, en el límite del rango de valores aceptado por el componente a probar.

Cuando hay que seleccionar una valor para realizar una prueba, se escoge aquellos que están situados justo en el límite de los valores admitidos.

Por ejemplo, supongamos que queremos probar el resultado de la ejecución de una función, que recibe un parámetro x:

Si el parámetro x de entrada tiene que ser mayor estricto que 5, y el valor es real, los valores límite pueden ser 4,99 y 5,01.

Si el parámetro de entrada x está comprendido entre -4 y +4, suponiendo que son valores enteros, los valores límite serán -5, -4 , -3,3, 4 y 5.

Autoevaluación

Si en un bucle while la condición es while ($x > 5 \&\& x < 10$), siendo x un valor single, sería valores límite

- 4 y 11
- 4,99 y 11
- 4,99 y 9,99.

Incorrecta. El valor de la variable es real, el 4 y el 11 no son valores límite

No es correcta. El 4,99 si es valor límite para la primera parte de la condición, pero 11 no.

Muy bien. En este caso, estos dos valores, pertenecen al conjunto de valores límite.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

a.3.- Conjetura de errores

La experiencia en la fase de pruebas indica que existen ciertos valores de entrada que típicamente son generadores de errores, y que en ocasiones, pasan desapercibidos en las técnicas de clases de equivalencia y de valores límite.

La **conjetura de errores** considera esos datos y define nuevos casos de prueba a los que someter a los programas. Se trata de una técnica menos metodica que las anteriores, tiene mucho más que ver con la **intuición y experiencia** del programador.

Una prueba típica en conjetura de errores es probar el valor de entrada 0 para datos numéricos por si pudiera participar como denominador en alguna división durante la ejecución del programa.

a.4.- Casos de uso, resultados esperados y análisis.

Ahora toca definir los datos de entrada al programa. Al conjunto de entradas al programa utilizados para cada ejecución se le denomina **caso de uso**. Los casos de uso se generarán a partir de las clases de equivalencia, valores límite y conjetas de errores obtenidos en los apartados anteriores.

Este proceso consta de las siguientes **fases**:

- Numerar las clase de equivalencia.
- Crear casos de uso que cubran todas las clases de equivalencia válidas. Se intentará agrupar en cada caso de uso tantas clases de equivalencia como sea posible.
- Crear un caso de uso para cada clase de equivalencia no válida.

Además toca definir los **resultados previstos** en cada ejecución. Cuando posteriormente se lance la ejecución del programa para cada caso de uso, los resultados obtenidos serán comparados con los esperados y así determinar la corrección del código.

b.- Ejemplo práctico

Indica:

- Las clases válidas y las clases no válidas.
- Todos los casos de prueba.

Siguiendo los criterios de las pruebas funcionales o caja negra.

De un programa que pide que indique una de estas dos palabras: "amigos", "visDivisores". Si no inserta ninguna de estas dos cosas, se cerrará el programa, visualizando este mensaje:

"Lo siento, no se ha indicado la operación adecuada. Cerramos el programa". (Mensaje 1)

Si indica una de estas dos cosas ("amigos" ó "visDivisores") se pedirá un primer número.

Dicho número tiene que ser par y positivo y, por supuesto, que no inserte letras.

Si dicho número:

- No es par y positivo, se visualizará un mensaje como este:

"El primer número no es par y positivo. Cerramos el programa". (Mensaje 2)

Y se cerrará el programa.

- Si se inserta letras, se visualizará un mensaje como este:

"Ha insertado letras en la indicación del primer número. Cerramos el programa". (Mensaje 3)

Y se cerrará el programa.

Si el número es correcto, se pedirá un segundo número que tiene que estar entre 3000 y 5000 (incluido el 3000 pero no el 5000) y, por supuesto, que no inserte letras (en la inserción de este segundo número).

Si este segundo número:

- No está entre 3000 y 5000, se visualizará un mensaje como este:

"El segundo número no está entre 3000 y 5000. Cerramos el programa". (Mensaje 4)

Y se cerrará el programa.

- Si inserta letras, en la indicación del segundo número, se visualizará un mensaje como este;

"Ha insertado letras en la indicación del segundo número. Cerramos el programa". (Mensaje 5)

Y se cerrará el programa.

Si indica bien el segundo número se visualizará:

- Si estos dos números son amigos

Dos números son amigos si tienen el mismo número de divisores, sin contar el 1 y el propio número.

O

- Los divisores de cada número.

Según se haya indicado al principio

b.1.- Determinar las clases de equivalencia y analizar los valores límite

Condición de entrada	Clases de equivalencia	Clases válidas	COD	Clases no válidas	COD
Operación	Conjunto de valores admitidos. Se identifica una clase válida por cada valor y una no válida.	“Amigos” “Divisores”	C1B1 C1B2	Algo distinto de “Amigos” o “Divisores”	C1E
Primer número	Condición verdadero/falso	Un dato que sea par y positivo	C2B	Un dato que no sea par y positivo Inserte letras	C2E1 C2E2
Segundo número	Rango de valores de entrada.	Un dato entre 3000 y 5000 Valor límite de 3000(es válido)	C3B1 C3B2	Un dato menor de 3000 Valor límite de 5000(no es válido) Un dato mayor de 5000 Inserte letras	C3E1 C3E2 C3E3 C3E4

b.2.- Casos de uso, resultados esperados y análisis

Casos prueba	de Clases equivalencia de	Condiciones de entrada			Resultado esperado
		Operación	Primer número	Segundo número	
CP1	C1B1,C2B,C3B1	"Amigos"	18	3000	Indicar si son amigos el 18 y el 3000
CP2	C1B1,C2B,C3B2	"Amigos"	18	4000	Indicar si son amigos el 18 y el 4000
CP3	C1B2,C2B,C3B1	"Divisores"	18	3000	Visual. Los divis.de 18 y 3000
CP4	C1B2,C2B,C3B2	"Divisores"	18	4000	Visual. Los divis.de 18 y 4000
CP5	C1E	"Paula"	---	---	Mensaje 1
CP6	C1B1,C2E1	"Amigos"	-15	---	Mensaje 2
CP7	C1B1,C2E2	"Amigos"	"Hola"	---	Mensaje 3
CP8	C1B2,C2E1	"Divisores"	-15	---	Mensaje 2
CP9	C1B2,C2E2	"Divisores"	"Hola"	---	Mensaje 3
CP10	C1B1,C2B,C3E1	"Amigos"	18	2000	Mensaje 4
CP11	C1B1,C2B,C3E2	"Amigos"	18	5000	Mensaje 4
CP12	C1B1,C2B,C3E3	"Amigos"	18	6000	Mensaje 4
CP13	C1B1,C2B,C3E4	"Amigos"	18	"Adiós"	Mensaje 5
CP14	C1B2,C2B,C3E1	"Divisores"	18	2000	Mensaje 4
CP15	C1B2,C2B,C3E2	"Divisores"	18	5000	Mensaje 4
CP16	C1B2,C2B,C3E3	"Divisores"	18	6000	Mensaje 4
CP17	C1B2,C2B,C3E4	"Divisores"	18	"Adiós"	Mensaje 5

Habría que insertar números que no son amigos entre si o que no tengan divisores, pero eso no lo establece la caja negra. Siempre se puede dar cuenta el testeador y tenerlo en cuenta,

Anexo III. JUnit

En los anexos anteriores se veía que entre los diferentes tipos de pruebas a los que someter los desarrollos software están las **pruebas unitarias**.

JUnit es una herramienta que ayuda a pasar de forma automática estas pruebas a los métodos y clases de los programas **Java**.

Para ello, habrá que identificar los casos de uso, datos de entrada y resultados esperados con las técnicas de caja blanca y negra ya conocidos. Una vez esté disponible esta información, toca lanzar con **JUnit** las ejecuciones programadas y valorar si los resultados obtenidos son exitosos (de acuerdo a lo previsto).

a.- Probando JUnit

Este manual describe de forma introductoria conceptos relacionados con **JUnit** en su **versión 5**, tales como:

- Etiquetas disponibles para configurar pruebas en Java (**anotaciones**).
- Llamadas a métodos de prueba para comprobar el funcionamiento del código (**assertions**).
- Aprender a utilizar el **plugging JUnit** disponible en Eclipse.

En las siguientes **urls** podrás encontrar información útil referida a **JUnit** y su uso en el **IDE Eclipse**:

Propósito	URL
Guía de uso de JUnit 5	https://junit.org/junit5/docs/current/user-guide/
Uso de JUnit 5 en Eclipse	https://www.eclipse.org/eclipse/news/4.7.1a/#junit-5-support

a.1.- Código propuesto

Vamos a usar JUnit con NetBeans. Desde la versión 7, ya lo tiene incorporado.

Para probar JUnit, vamos a crear un nuevo proyecto (en NetBeans) con este código:

```
package claseunojunit_;

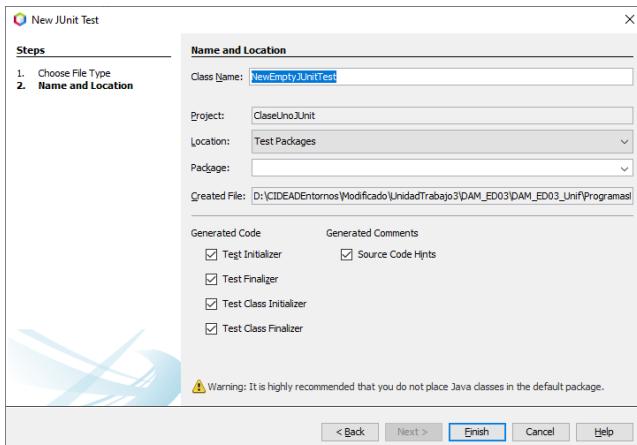
public class ClaseUnoJUnit_ {
    int dato1,dato2;
    public ClaseUnoJUnit_(int dato1, int dato2) {
        this.dato1=dato1;
        this.dato2=dato2;
    }
    public int suma()
    {
        return dato1+dato2;
    }
    public int resta()
    {
        return dato1-dato2;
    }

    public int dividir()
    {
        return dato1%dato2;
    }
}
```

Pulsa [aquí](#) para bajarte dicho proyecto ("ClaseUnoJUnit_.zip").

Posteriormente, sobre el nombre del proyecto. En el menú contextual, que aparece, elegir la opción new/JUnit Test.

Se visualizará esta ventana:



Os aconsejo poner el nombre de la clase (a probar) acabado en Test.

Vais a ver que os creado un fichero java para hacer las pruebas, en el paquete "Test packages":

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a Java project named "ClassmateJunitTest".
- Java Editor:** Displays the file "ClassmateJunitTest.java" containing JUnit test code.
- Toolbars:** Standard Eclipse toolbars for file operations, search, and navigation.
- MenuBar:** Includes "File", "Edit", "View", "Source", "Refactor", "Run", "Debug", "Profile", "Tools", "Help", and "Window".
- Search Bar:** Located at the top right.

```
1 package com.classmate.junit;
2
3 import org.junit.After;
4 import org.junit.Before;
5 import org.junit.Test;
6 import static org.junit.Assert.assertEquals;
7
8 /**
9  * To change this template file, choose Tools | Templates
10  * and open the template in the editor.
11 */
12
13 public class ClassmateJunitTest {
14
15     /**
16      * Author: Amith
17     */
18     public void testAddition() {
19
20         // Test case 1
21         assertEquals(4, add(2, 2));
22
23         // Test case 2
24         @BeforeClass
25         public static void setUpClass() {
26
27             @AfterClass
28             public static void tearDownClass() {
29                 // Clean up
30             }
31
32             @Before
33             public void setUp() {
34
35             }
36
37             @Test
38             public void testMultiplication() {
39
40                 assertEquals(12, multiply(3, 4));
41             }
42
43         }
44     }
45 }
```

a.2.- Etiquetas JUnit

.En esta sección se describen las utilizadas con mayor frecuencia.

Estas serían las etiquetas más utilizadas:

- **@BeforeClass:** Sólo puede haber un método con este marcador, es invocado una vez al principio de todas los test. Se suele usar para inicializar atributos.
- **@AfterClass:** Sólo puede haber un método con este marcador y se invoca al finalizar todas los test.
- **@Before:** Se ejecuta antes de cada test.
- **@After:** Se ejecuta después de cada test.
- **@Ignore:** Los métodos marcados con esta anotación no serán ejecutados.
- **@Test:** Identifica los métodos test que deben ser probados

La que mas usaremos sera **BeforeClass** y **Test**, pero siempre nos puede venir bien usar alguna de las anotaciones anteriores.

Unas anotaciones a destacar son @beforeclass y @afterclass, tienen algunas diferencias respecto a las anteriores:

- **@beforeclass:** solo puede haber un método con esta etiqueta. El método marcado con esta anotación es invocado una vez al principio del lanzamiento de todas las pruebas. Se suele utilizar para inicializar atributos comunes a todas las pruebas o para realizar acciones que tardan un tiempo considerable en ejecutarse.
- **@afterclass:** solo puede haber un método con esta anotación. Este método será invocado una sola vez cuando finalicen todas las pruebas.

Este es el código que te ha generado NetBeans:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Admin
 */
public class ClaseUnoJUnitTest_ {

    public ClaseUnoJUnitTest_() {

    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
}
```

```
// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
}
```

En dicho código he incluido una serie de mensajes para ver en qué orden se ejecutan los métodos. Este sería el código:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Admin
 */
public class ClaseUnoJUnitTest_ {

    public ClaseUnoJUnitTest_() {
        System.out.println("Llamando a la función constructora: ClaseUnoJUnitTest");
    }

    @BeforeClass
    public static void setUpClass() {
        System.out.println("Llamando a setUpClass, con la etiqueta @BeforeClass");
    }

    @AfterClass
    public static void tearDownClass() {
        System.out.println("Llamando a tearDownClass con la etiqueta @AfterClass");
    }

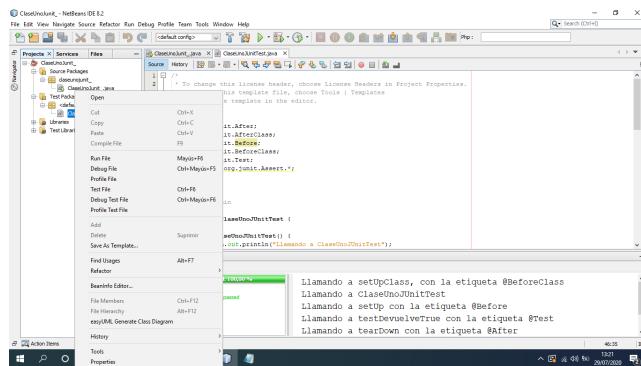
    @Before
    public void setUp() {
        System.out.println("Llamando a setUp con la etiqueta @Before");
    }

    @After
    public void tearDown() {
        System.out.println("Llamando a tearDown con la etiqueta @After");
    }

    // TODO add test methods here.
    // The methods must be annotated with annotation @Test. For example:
    //
    // @Test
    // public void hello() {}
    @Test
    public void testDevuelveTrue() {

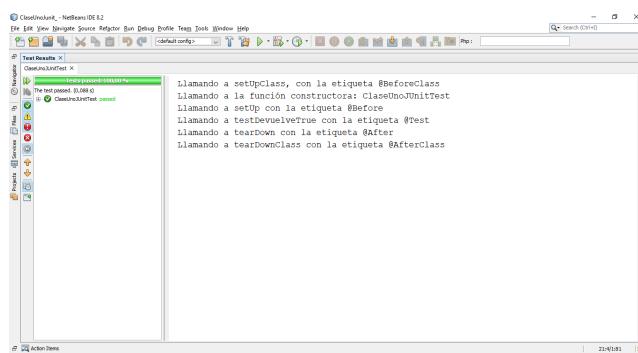
        System.out.println("Llamando a testDevuelveTrue con la etiqueta @Test");
    }
}
```

Para ejecutarlo, tienes que dar al botón derecho del ratón:



Y eliges la opción "Run File".

La salida sería esta:



Así puedes ver en qué orden se ejecutan los métodos.

a.3.- Assertions (afirmaciones).

En el apartado anterior se indica que las llamadas a los métodos de prueba son configurables en diferentes aspectos, tales como: qué hacer antes/después de cada prueba, cómo pasar parámetros a las llamadas o cómo repetir de forma automática un determinado test. Pero en realidad, no hemos lanzado prueba alguna a los métodos del código del proyecto implementado en la clase ClaseUnoJUnit.

JUnit utiliza la **clase Assertions** para lanzar los **test**, que básicamente está compuesta por una serie de métodos, que una vez llamados ejecutan los métodos a probar y analizan su comportamiento comparándolos con los resultados que se espera de ellos.

Así, hay métodos que nos permiten comprobar si dos valores son o no iguales, si el valor del parámetro pasado se puede resolver como true o false, si el tiempo consumido en ejecutar un método supera el previsto, etc.

Además, los métodos están sobrecargados, permitiendo en algunos casos indicar el mensaje que ha de devolver si la comprobación no resulta exitosa, o definir un margen que valide dos números como iguales si su diferencia es inferior a dicha tolerancia .

Ahora vamos a ver los métodos que usaremos, disponibles en JUnit, todos se invocan con la clase **Assert** de forma estática. Los mas usados son:

- **assertEquals(resultado esperado, resultado actual)**: le pasamos el resultado que nosotros esperamos y invocamos la función que estamos testando.
- **assertNull(objeto)**: si un objeto es null el test sera exitoso.
- **assertNotNull(objeto)**: al contrario que el anterior.
- **assertTrue(condición)**: si la condición pasada (puede ser una función que devuelva un booleano) es verdadera el test sera exitoso.
- **assertFalse(condición)**: si la condición pasada (puede ser una función que devuelva un booleano) es falsa el test sera exitoso.
- **assertSame(Objeto1, objeto2)**: compara las referencias de los objetos.
- **assertNotSame(Objeto1, objeto2)**: al contrario que el anterior.

Para entender mejor los **métodos assert**, se va a modificar la clase ClaseUnoJUnitTest nuevamente, sustituyendo los mensajes por pantalla que se mostraban en la sección anterior por llamadas assert que permitan probar los métodos de la clase ClaseUnoJUnit_.

Nuevo código de la ClaseUnoJUnitTest.java:

```
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
import claseunojunit_.ClaseUnoJUnit_;

/**
 *
 * @author Admin
 */
public class ClaseUnoJUnitTest {

    ClaseUnoJUnit_ calc;

    public ClaseUnoJUnitTest() {
        System.out.println("Llamando a la función constructora: ClaseUnoJUnitTest");
    }

    @BeforeClass
    public static void setUpClass() {
        System.out.println("Llamando a setUpClass, con la etiqueta @BeforeClass");
    }

    @AfterClass
    public static void tearDownClass() {
        System.out.println("Llamando a tearDownClass con la etiqueta @AfterClass");
    }
}
```

```

@Before
public void setUp() {

    System.out.println("Llamando a setUp con la etiqueta @Before");

    calc=new ClaseUnoJUnit_(4, 67);
}

@After
public void tearDown() {
    System.out.println("Llamando a tearDown con la etiqueta @After");
}

// TODO add test methods here.
// The methods must be annotated with annotation @Test. For example:
//
// @Test
// public void hello() {}
@Test
public void testDevuelveTrue() {

    System.out.println("Llamando a testDevuelveTrue con la etiqueta @Test");
}

public void testSuma() {

    assertEquals(71, calc.suma());

}

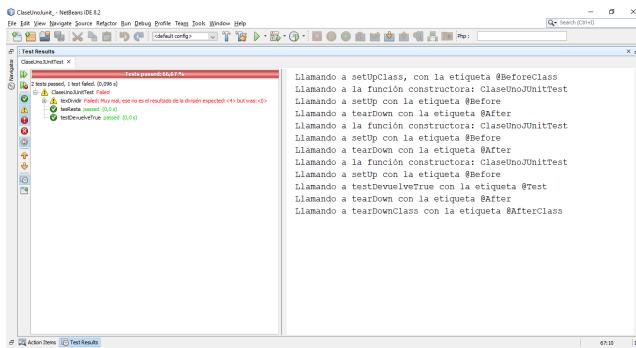
@Test
public void tesResta()
{

    assertEquals(-63, calc.resta());
}

@Test
public void texDividir()
{
    ClaseUnoJUnit_ calc=new ClaseUnoJUnit_(100, 25);
    // assertTrue(calc.dividir()==1);
    assertEquals("Muy mal, ese no es el resultado de la división",4, calc.dividir());//Lanza ese mensaje si no cuadra
}
}

```

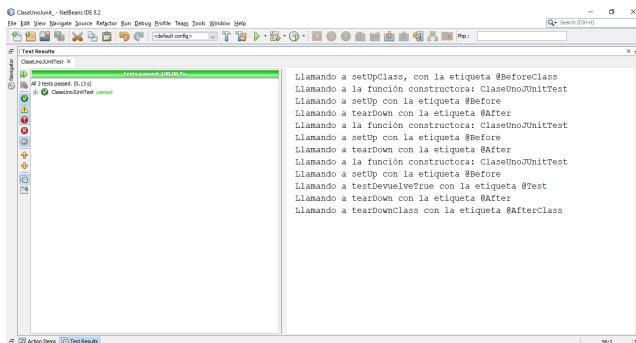
Ejecutamos dicha clase y vemos que el resultado de dicha ejecución es:



Donde nos da información de que ha ejecutado los métodos que se le ha pedido. Y te confirma que el resultado era el que esperabas menos el de la división. Pusimos que tenía que devolver un 4 y devuelve un 0.

Al ver ese error, vamos a dicho método y comprobamos que efectivamente el método dividir está mal. Pusimos el operador "%" que da el resto de una división y había que haber puesto "/" que es el operador en Java que devuelve el cociente de una división.

Si corregimos ese fallo y volvemos a ejecutar el Test veremos que el resultado es este:



Anexo IV.- Depuración con Eclipse

Para poder comprobar algunas de las **funciones de depuración** que nos ofrece **Eclipse**, vamos a crear un proyecto en **Java** con el siguiente código:

Clase Test

```
public class Test {
    public static void main(String[] args) {
        Contador contador = new Contador();
        contador.contar();
        System.out.println("Cuenta: " + contador.getResultado());
    }
}
```

Clase Contador

```
public class Contador {
    private int resultado = 0;
    public int getResultado()
    {
        return resultado;
    }

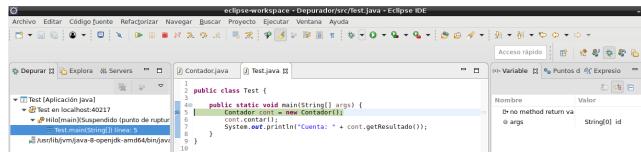
    public void contar()
    {
        for(int i=0; i < 100 ; i++)
        {
            resultado = resultado + i + 1;
        }
    }
}
```

Pulse [aquí](#) para descargarte dicho proyecto ("Depuracion-sw.rar").

Existen varias alternativas para lanzar la ejecución de un programa en modo debug. Una de ellas es pulsando con el botón derecho del ratón sobre la clase de inicio del proyecto (implementa el método main), y seleccionar en el menú contextual que aparece "**Depurar como => Aplicación Java**".



En **modo depuración**, **Eclipse** da la opción de trabajar con la perspectiva depurar, que ofrece una serie de vistas muy interesantes para este tipo de ejecución, tales como: la vista de visualización y cambio de variables, la vista de puntos de parada establecidos o la pila de llamadas entre otras.



a.- Puntos de ruptura.

Al solicitar una ejecución en modo debug, si no hemos establecido puntos de parada en el código, éste ejecutará hasta el final del mismo modo que lo haría en una ejecución normal.

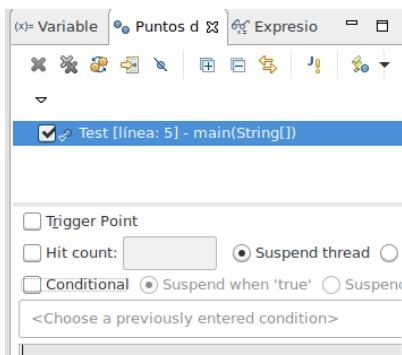
Para establecer un **punto de parada**, basta con hacer doble clic en el margen izquierdo de la línea de código donde se va a establecer.

```

1 Contador.java 2 Test.java ☰
2
3 public class Test {
4
5     public static void main(String[] args) {
6         Contador contador = new Contador();
7         contador.contar();
8         System.out.println("Cuenta: " + contador.getResultado());
9     }
10

```

El punto de parada es dado de alta y ya aparece en la **vista de puntos de parada**.



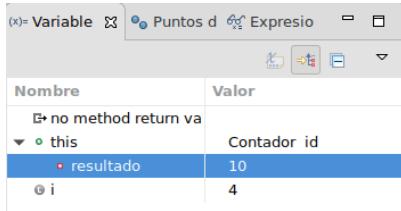
Eclipse permite definir puntos de parada condicionales para personalizar cuándo o porqué se para el programa. Si por ejemplo, seleccionamos en un punto de parada la **opción Hit count** y determinamos un valor, el programa parará cuando haya pasado por este punto el número de veces indicado.

Es posible crear condiciones más elaboradas dependientes de otras variables disponibles en el contexto de la ejecución. El programa sólo parará cuando la condición definida se cumpla y la ejecución pase por esa línea de código.

En el momento que tenemos detenido el programa, se pueden realizar diferentes labores: por un lado, se pueden examinar las variables, y comprobar que los valores que tienen asignados son correctos, o se pueden iniciar una depuración paso a paso e ir comprobando el camino que toma el programa a partir del punto de ruptura. Una vez realizada la comprobación, podemos abortar el programa, o continuar la ejecución normal del mismo.

b.- Examinadores de variables

Durante el proceso de implementación y prueba de software, una de las maneras más comunes de comprobar que la aplicación funciona de manera adecuada, es chequear que las **variables** vayan tomando los valores adecuados en cada momento. **Eclipse** nos proporciona la **vista variables** donde podemos ir comprobando el valor que van tomando las variables activas en la zona de código donde está el programa parado.



The screenshot shows the Eclipse IDE's "Variables" view. The title bar has tabs for "Variable", "Puntos d", "Expresio", and a close button. Below the tabs are icons for copy, cut, paste, and refresh. A toolbar with a magnifying glass and other buttons is at the top. The main area is a table with columns "Nombre" and "Valor". It lists several variables:

Nombre	Valor
↳ no method return va	
↳ this	Contador id
↳ resultado	10
↳ i	4

Además, pulsando sobre el valor de cualquiera de estas variables es posible modificarlas. Esto nos permite evaluar nuevos escenarios de prueba con datos diferentes.

c.- Botones de depuración

Cuando estamos en el proceso de depuración de un programa con la **perspectiva Depurar**, **Eclipse** nos ofrece una serie de botones en su **barra de herramientas** que pasamos a describir a continuación.

	Ctrl + Alt + B	Desactiva temporalmente todos los puntos de parada del código.
	F8	Continua la ejecución del programa. Se detendrá en el siguiente punto de parada.
		Pausa/detiene la ejecución del programa en el punto de código donde se encuentre al ser pulsado.
	Ctrl + F2	Finaliza la ejecución del programa.
		Función no considerada en este manual. Consultar manual de Eclipse.
	F5	Si el programa se encuentra detenido en la llamada a un método, al pulsar este botón la ejecución pasa a la primera línea del mismo.
	F6	Si el programa se encuentra detenido en la llamada a un método, al pulsar este botón la ejecución del método se hace por completo, sin depurar su implementación.
	F7	Avanza la ejecución del programa hasta que nos salimos del método actual y vamos hasta el sitio donde fue llamado.

Anexo V.- Enlaces de interés

Algunos enlaces de interés.

Metodología métrica.

https://administracionelectronica.gob.es/pae/Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html#W3u0j7i-nIU

Portal de Administración Electrónica Ministerio de Política Territorial y Función Pública Secretaría General de Administración Digital.

Metodologías ágiles o DevOps (Desarrollo y operación).

Se tratan de metodologías más actuales que buscan minimizar los tiempos desde que se adquiere el compromiso en un desarrollo hasta que se encuentra disponible en operación.

<https://es.wikipedia.org/wiki/DevOps>

Esta obra contiene una traducción parcial derivada de *DevOps* de Wikipedia en inglés, publicada por sus editores bajo la Licencia de documentación libre de GNU y la Licencia Creative Commons Atribución-CompartirIgual 3.0 Unported.

Caja blanca.

<https://www.youtube.com/watch?v=GVegCwwfBZ0>

Canal de Youtube de Juan V. Carrillo publicado con Licencia Atribución de Creative Commons.

<https://www.youtube.com/watch?v=9N5vPeSWRfQ>

Canal de Youtube de Juan V. Carrillo publicado con Licencia Atribución de Creative Commons.

<https://www.youtube.com/watch?v=iLLI-n57IEs>

Canal de Youtube de Juan V. Carrillo publicado con Licencia Atribución de Creative Commons.

Caja negra.

<https://www.youtube.com/watch?v=PmdFMDZVmM>

Canal de Youtube de Juan V. Carrillo publicado con Licencia Atribución de Creative Commons.

Anexo VI.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría:Ebnz Licencia:Creative Commons. Genérica de Atribución/Compartir-Igual 3,0 Procedencia:Montaje sobre http://es.wikipedia.org/		Autoría: Oracle Corporation Licencia:Copyright cita Procedencia:Captura de pantalla de Netbeans
	Autoría:Oracle Corporation Licencia: Copyright cita Procedencia: Captura de pantalla de Netbeans		Autoría:Oracle Corporation Licencia:Copyright cita Procedencia:Captura de pantalla de Netbeans
	Autoría: Oracle Corporation. Licencia: Copyright cita Procedencia: Captura de pantalla de Netbeans		Autoría: Francisco Javier Cabrerizo Membrilla. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Francisco Javier Cabrerizo Membrilla. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Francisco Javier Cabrerizo Membrilla. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría:Oracle Corporation Licencia:Copyright cita Procedencia:Captura de pantalla de Netbeans		Autoría:Scott Schram. Licencia:CC by dominio público. Procedencia: www.flickr.com
	Autoría:Oracle Corporation Licencia:Copyright cita Procedencia:Captura de pantalla de Netbeans		Autoría:JaulaDeArdilla Licencia:CC by-nc-nd Procedencia: http://www.flickr.com/photos/jauladeardilla/2285620559/
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de		Autoría: Ministerio de Educación.

	<p>Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.</p>

	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia

	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		

Optimización y documentación.

Caso práctico

Situación

BK programación se encuentra desarrollando la aplicación de gestión hotelera.

Como parte del proceso de desarrollo surgen una serie de problemas: hay un código que presenta algunas implementaciones que se pueden mejorar aplicando refactorización, nos encontramos que los miembros del equipo de desarrollo están modificando constantemente métodos o clases, creando diferentes versiones de las mismas y falta una documentación clara y precisa del código.

Ada propone a **Juan** y a **María** (programadores principales de la aplicación) que usen los patrones generales de refactorización para conseguir un código de mejor calidad, más fácil de entender, más fácil de probar y con menor probabilidad de generar errores. **Ana** va a intentar ayudar a **Juan**, y **Carlos** a **María**, pero no conocen nada de refactorización, ni entiende la necesidad de realizar este trabajo "extra".



Como todos los miembros de BK Programación trabajan sobre los mismo proyectos, **Ada** debe coordinar el trabajo de todos ellos, por lo que propone que cada uno de ellos utilice un cliente de control de versiones, de forma que ella, de manera centralizada, pueda gestionar la configuración del software.

Los miembros con menor experiencia, **Carlos** y **Ana**, van a ir generando, utilizando herramientas de documentación automatizadas, la documentación de las clases y del código generado por **Juan** y **María**.

Ada va a encargarse de la Gestión de Configuraciones del Software, ya que es una labor fundamental para cualquier jefe de proyecto. Asimismo, debe garantizar que el código generado por **Juan** y **María** esté refactorizado.



[Ministerio de Educación y Formación Profesional](#). (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Refactorización.

Caso práctico



Gran parte de las clases, métodos y módulos que forman parte de la aplicación de Gestión Hotelera han sido implementados, surge ahora una pregunta. ¿Podemos mejorar la estructura del código y que sea de mayor calidad, sin que cambie su comportamiento? ¿Cómo hacerlo? ¿Qué patrones hay que seguir?

¿Podemos mejorar la estructura del código y que sea de mayor calidad, sin que cambie su comportamiento? ¿Cómo hacerlo?
¿Qué patrones hay que seguir?.

La **refactorización** es una técnica, que consiste en realizar pequeñas transformaciones en el código de un programa, para mejorar la estructura sin que cambie el comportamiento ni funcionalidad del mismo. Su objetivo es mejorar la estructura, la legibilidad o la eficiencia del código.

Con la refactorización se mejora el diseño del software, hace que el software sea más fácil de entender, hace que el mantenimiento del software sea más sencillo, la refactorización ayuda a que el programa sea más rápido.

La idea de refactorización de código, se basa en el concepto matemático de factorización de polinomios. Así, resulta que $(x + 1)(x - 1)$ se puede expresar como $x^2 - 1$ sin que se altere su sentido.

Algunas pistas que nos pueden indicar la necesidad de refactorizar un programa son:

- Código duplicado.
- Métodos demasiado largos.
- Clases muy grandes o con demasiados métodos.
- Métodos más interesados en los datos de otra clase que en los de la propia.
- Grupos de datos que suelen aparecer juntos y parecen más una clase que datos sueltos.
- Clases con pocas llamadas o que se usan muy poco.
- Exceso de comentarios explicando el código.

Hay que resaltar que la refactorización no cambia el comportamiento observable del software. El software sigue cumpliendo la misma función que hacía antes. Ningún usuario, ya sea usuario final u otro programador, podrá determinar qué cosas han cambiado.

Para saber más

Puedes visitar la siguiente página web (en inglés), donde se presenta el proceso de refactorización de aplicaciones Java con Netbeans.

[Refactorización en Netbeans.](#)

1.1.- Concepto.

Caso práctico



Juan va a empezar a refactorizar parte del código que ha generado. Ana no sabe que es refactorizar, así que Juan le va a explicar las bases del proceso de refactorización.

El concepto de refactorización de código, se basa en el concepto matemático de factorización de polinomios.

Podemos definir el concepto de refactorización de dos formas:

FACTORIZACIÓN DE POLINOMIOS

$$X^2 - 1 = (X + 1)(X - 1)$$

Refactorización: Cambio hecho en la estructura interna del software para hacerlo más fácil de entender y fácil de modificar sin modificar su comportamiento.

Ejemplos de refactorización es "Extraer Método" y "Encapsular Campos". La refactorización es normalmente un cambio pequeño en el software que mejora su mantenimiento.

Campos encapsulados: Se aconseja crear métodos `getter` y `setter`, (de asignación y de consulta) para cada campo que se defina en una clase. Cuando sea necesario acceder o modificar el valor de un campo, basta con invocar al método `getter` o `setter` según convenga.

Refactorizar: Reestructurar el software aplicando una serie de refactorizaciones sin cambiar su comportamiento.

El propósito de la refactorización es hacer el software más fácil de entender y de modificar. Se pueden hacer muchos cambios en el software que pueden hacer algún pequeño cambio en el comportamiento observable. Solo los cambios hechos para hacer el software más fácil de entender son refactorizaciones.

Hay que diferenciar la refactorización de la optimización. En ambos procesos, se pretende mejorar la estructura interna de una aplicación o componente, sin modificar su comportamiento. Sin embargo, cuando se optimiza, se persigue una mejora del rendimiento, por ejemplo mejorar la velocidad de ejecución, pero esto puede hacer un código más difícil de entender.

Hay que resaltar que la refactorización no cambia el comportamiento observable del software. El software sigue cumpliendo la misma función que hacia antes. Ningún usuario, ya sea usuario final u otro programador, podrá determinar que cosas han cambiado.

Reflexiona

Con la refactorización de código, estamos modificando un código que funciona correctamente, ¿merece la pena el esfuerzo de refactorizar un código ya implementado?

[Mostrar retroalimentación](#)

Merece la pena, porque el código que se consigue suele ser más fácil de entender por otros programadores y programadoras, y por tanto, más fácil de mantener. Además, en muchos casos, se mejora la eficiencia del programa.

1.2.- Limitaciones.

Caso práctico

Ana se muestra muy interesada por conocer estas técnicas avanzadas de programación, sin embargo Juan le pone los pies en el suelo, explicándole que son técnicas con muchas limitaciones y poca documentación en la que basarse.



La refactorización es una técnica lo suficientemente novedosa para conocer cuáles son los beneficios que aporta, pero falta experiencia para conocer el alcance total de sus limitaciones. Se ha constatado que la refactorización presenta problemas en algunos aspectos del desarrollo.

Un área problemática de la refactorización son las bases de datos. Una base de datos presenta muchas dificultades para poder ser modificada, dado la gran cantidad de interdependencias que soporta. Cualquier modificación que se requiera de las bases de datos, incluyendo modificación de esquema y migración de datos, puede ser una tarea muy costosa. Es por ello que la refactorización de una aplicación asociada a una base de datos, siempre será limitada, ya que la aplicación dependerá del diseño de la base de datos.

Otra limitación, es cuando cambiamos interfaces. Cuando refactorizamos, estamos modificando la estructura interna de un programa o de un método. El cambio interno no afecta al comportamiento ni a la interfaz. Sin embargo, si renombramos un método, hay que cambiar todas las referencias que se hacen a él. Siempre que se hace esto se genera un problema si es una interfaz pública. Una solución es mantener las dos interfaces, la nueva y la vieja, ya que si es utilizada por otro clase o parte del proyecto, no podrá referenciarla.

Hay determinados cambios en el diseño que son difíciles de refactorizar. Es muy difícil refactorizar cuando hay un error de diseño o no es recomendable refactorizar, cuando la estructura a modificar es de vital importancia en el diseño de la aplicación.

Hay ocasiones en las que no debería refactorizar en absoluto. Nos podemos encontrar con un código que, aunque se puede refactorizar, sería más fácil reescribirlo desde el principio. Si un código no funciona, no se refactoriza, se reescribe.

Autoevaluación

La refactorización:

- Se utiliza como técnica complementaria de realización de pruebas.
- Genera un código más difícil de entender y de mantener.
- Utiliza una serie de patrones, de aplicación sobre el código fuente.
- Es una técnica de programación no recogida por los entornos de desarrollo.

Incorrecta, la refactorización no cambia el comportamiento.

No es la respuesta correcta, todo lo contrario.

Muy bien. Esa es la idea.

No es correcta, la mayoría de los IDE actuales, incluyen herramientas de refactorización.

Solución

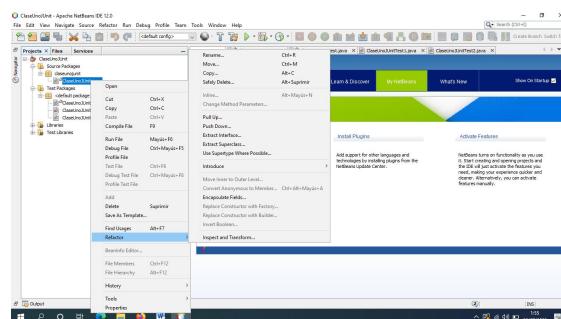
1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

1.3.- Patrones de refactorización más habituales.

Caso práctico



A pesar de la poca documentación y lo novedoso de la técnica, **Juan** le enseña a **Ana** algunos de los patrones más habituales de refactorización, que vienen ya integrados en la mayoría de los entornos de desarrollos más extendidos en el mercado.



Algunos de los **patrones más habituales de refactorización**, que vienen ya integrados en la mayoría de los entornos de desarrollos, son los siguientes:

- **Renombrar.** Cambiar el nombre de un paquete, clase, método o campo, por un nombre más significativo.
- **Encapsular campos.** Crear métodos de asignación y de consulta (**getters y setters**) para los campos de la clase, que permitan un control sobre el acceso de estos campos, debiendo hacerse siempre mediante el uso de estos métodos.
- **Sustituir bloques de código por un método.** En ocasiones se observa que un bloque de código puede constituir el cuerpo de un método, dado que implementa una función por si mismo o aparece repetido en múltiples sitios. De esta forma, cada vez que queramos acceder a ese bloque de código, bastaría con invocar al método.
- **Modificar la extensión del código.** Hacer un código más extenso si se gana en claridad o menos extenso sólo si con eso se gana eficiencia.
- **Reorganizar código condicional complejo.** Patrón aplicable cuando existen varios if o condiciones anidadas o complejas.
- **Crear código común** (en una clase o método) para evitar el código repetido.
- **Mover la clase.** Mover una clase de un paquete a otro, o de un proyecto a otro. Esto implica la actualización en todo el código fuente de las referencias a la clase en su nueva localización.
- **Borrado seguro.** Garantizar que cuando un elemento del código ya no es necesario, se borran todas las referencias a él que había en cualquier parte del proyecto.
- **Cambiar los parámetros del método.** Permite añadir/modificar/eliminar los parámetros en un método y cambiar los modificadores de acceso.
- **Extraer la interfaz.** Crea un nueva interfaz de los métodos public non-static seleccionados en una clase o interfaz.

Autoevaluación

¿Cuál no es un patrón de refactorización?

- Eliminar parámetros de un método.
- Renombrado.
- Sustitución de un bloque de sentencias por un método.
- Mover clase.

Correcta. Al eliminar parámetros de un método, podría cambiar el comportamiento del método.

Incorrecta.

No es correcta.

No es cierto.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

1.4.- Refactorización en Eclipse

Caso práctico

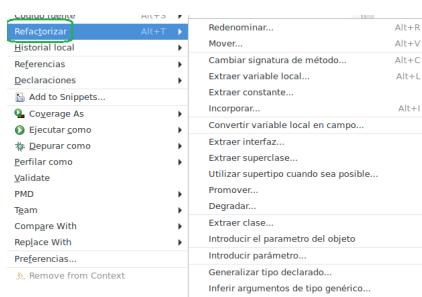


Ana ya conoce los principios y patrones básicos de refactorización, ahora Juan le va a enseñar las herramientas de Eclipse para refactorizar de forma automática y sencilla, código Java.

Los entornos de desarrollo actuales, nos proveen de una serie de herramientas que nos facilitan la labor de refactorizar nuestro código. En puntos anteriores, hemos indicado algunos de los patrones que se utilizan para refactorizar el código. Esta labor se puede realizar de forma manual, pero supone una pérdida de tiempo, y podemos inducir a redundancias o a errores en el código que modificamos.

En el Entorno de Desarrollo Eclipse, la refactorización está integrada como una función más, de las utilidades que incorpora.

A continuación se muestra el menú contextual disponible al hacer *clic* con el botón secundario sobre un fragmento de código en algunas versiones de **Eclipse** y escoger la opción **Refactorizar**. Aparece un menú con muchas opciones, de las que estudiaremos algunas.



Nota: el menú mostrado es contextual, por lo que la opción **Refactorizar**, podrá mostrar algunas opciones diferentes en función de la porción de código sobre la que sea llamado.

La mayor parte de estas funciones permanecen disponibles en las versiones más actuales de **Eclipse**.

A continuación se van a explorar las funciones de refactorización

1.4.1.- Renombrar

Es la opción más común, modifica el nombre a cualquier elemento (variable, atributo, método, clase...) y hace los cambios necesarios en las referencias que haya a dicho elemento en todo el proyecto.

Ejemplo: sustituir en el método `main` el nombre de la variable local `teacher` por `tch`. El cambio se realiza en todas sus apariciones.

```
clasecolor.java
1 package principal;
2 import clases.profesor;
3 // Clase color, el profesor pregunta
4 public class clasecolor {
5@     public static void main(String[] args) {
6         profesor teacher = new profesor();
7         String color = teacher.getColor();
8         System.out.println("La color es " + color);
9     }
}
```

```
clasecolor.java
1 package principal;
2 import clases.profesor;
3 // Clase color, el profesor pregunta
4 public class clasecolor {
5@     public static void main(String[] args) {
6         profesor tch = new profesor();
7         String color = tch.getColor();
8         System.out.println("La color es " + color);
9     }
}
```

1.4.2.- Mover

Cambia una clase de un paquete a otro, afectando a su declaración "package" y a su localización en el disco.
Ejemplo: mover la clase ordenador del paquete clases al paquete principal.



1.4.3.- Cambiar firma del método

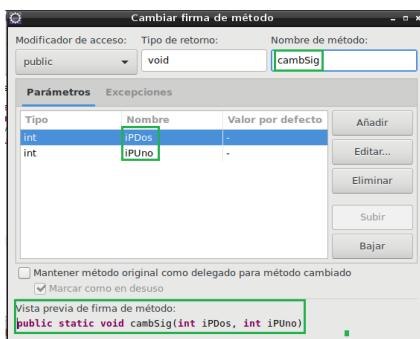
Modifica la “firma” o cabecera del método. Si el método ha sido ya usado, cambiar el número o tipo de parámetros (así como el tipo de valor devuelto) provocará fallos de compilación.

Es útil para cambiar el nombre de los parámetros, o su orden (**Eclipse** modificará también el orden de entrada de los parámetros en todas las llamadas al método).

Ejemplo: cambiar el nombre del método y de los parámetros al método `public void CambSignatura(int iParamUno, int iParamDos):`

```

1 package principal;
2 import clases.profesor;
3 // Clase color, el profesor pregunta a un alumno por un color ent
4 public class clasecolor {
5     public static void main(String[] args) {
6         profesor tch = new profesor();
7         String color = tch.preguntacolor();
8         System.out.println("La respuesta recibida es:" + color);
9         CambSignatura(5,10);
10    }
11 }
12
13@ public static void CambSignatura(int iParamUno, int iParamDos)
14 {
15     System.out.println("Primer Parametro" + iParamUno);
16     System.out.println("Segundo Parametro" + iParamDos);
17 }
18 }
```



```

clasecolor.java 33
1 package principal;
2 import clases.profesor;
3 // Clase color, el profesor pregunta a un alumno por un color ent
4 public class clasecolor {
5@     public static void main(String[] args) {
6         profesor tch = new profesor();
7         String color = tch.preguntacolor();
8         System.out.println("La respuesta recibida es:" + color);
9         CambSig(10,5);
10    }
11 }
12
13@ public static void CambSig(int iPUno, int iPDos)
14 {
15     System.out.println("Primer Parametro" + iPUno);
16     System.out.println("Segundo Parametro" + iPDos);
17 }
18 }
```

1.4.4.- Extraer variable local.

Crear una variable local inicializada con el valor de un literal (número, String...). Las referencias a esa expresión se modifican por una referencia a la variable.

```
5. public static void main(String[] args) {  
6.     profesor tch = new profesor();  
7.     System.out.println(tch.preguntarPregunta());  
8.     System.out.println("La respuesta recibida es: " + color);  
9.  
10.    cambiaColor(10,3);  
11}
```

↓

Extrair variable local

Nombre de variable: **sResp**

Sustituir todas las apariciones de la expresión seleccionada por referencias a la variable local

Declarar la variable local como 'final'

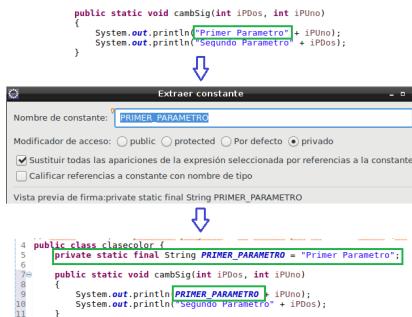
→

```
public static void main(String[] args) {  
    profesor tch = new profesor();  
    System.out.println(tch.preguntarPregunta());  
    String sResp = "La respuesta recibida es: ";  
    System.out.println(sResp + color);  
}
```

1.4.5. Extraer constante

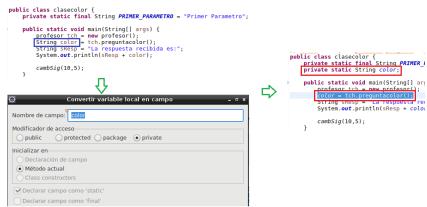
Exactamente igual que el anterior, pero genera una constante con la expresión seleccionada.

Ejemplo: sobre el método cambSig anteriormente creado, convertir el literal "Primer parametro" en una constante en el ámbito de la clase.



1.4.6. Convertir variable local en atributo.

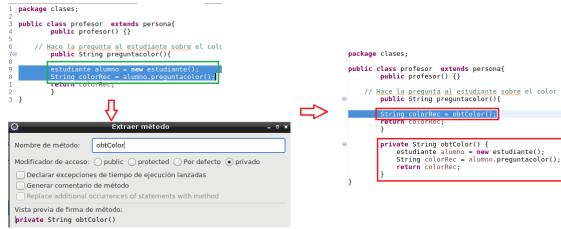
A veces se definen variables locales dentro de un método que luego resultan ser relevantes en el ámbito de la clase, por tanto debe ser considerada un atributo de la clase.



1.4.7. Extraer método.

Convierte el código seleccionado en un método, útil en código que es reutilizado en varios sitios del programa. También puede servir para aligerar un método que es demasiado largo.

Eclipse solo solicita el nombre del método pero descubre automáticamente los parámetros y tipo de retorno necesarios.



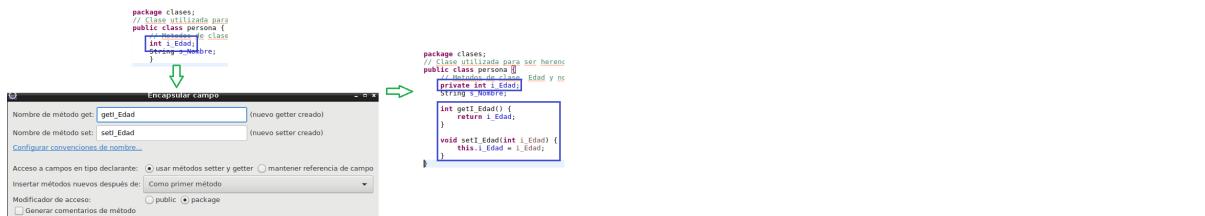
1.4.8. Incorporar

Hace lo contrario que los “*Extract*”: elimina una declaración de variable, método o constante y coloca su valor (en el caso de variables y constantes) o su código (en el caso de métodos) en aquellos lugares en que se referenciaba a esa variable, método o constante que ya no existirán.

Es muy útil cuando se comprueba que el contenido de una variable o constante se va a usar una sola vez y por tanto no merece la pena almacenarlo, sino que queda más limpio el código en una línea. También cuando se observa que un método sólo se llama una o dos veces, por lo que no merece la pena aislar ese código en un método.

1.4.9. Autoencapsular atributo.

Convierte una variable de clase en privada y genera los métodos *Get* y *Set* públicos para acceder a la misma. Opción también disponible desde el menú *código fuente* que se verá a continuación.



1.5.- Analizadores de código.

Caso práctico



María se va a centrar el uso de analizadores de código con la ayuda de **Carlos**. Carlos no sabe lo que son los analizadores de código ni para qué sirven.

Cada IDE incluye herramientas de refactorización y analizadores de código. En el caso se software libre, existen analizadores de código que se pueden añadir como complementos a los entornos de desarrollo.

El análisis estático de código, es un proceso que tiene como objetivo, evaluar el software, sin llegar a ejecutarlo. Esta técnica se va a aplicar directamente sobre el código fuente, para poder obtener información que nos permita mejorar la base de código. Si el analizador considera que nuestro código fuente tiene una estructura mejorable, nos lo indicará y también nos comunicará la mejora a realizar. Por lo tanto, el uso de analizadores de código proporciona información sobre algunos aspectos a considerar en la refactorización de los programas.

Las principales funciones de los analizadores es encontrar partes del código que puedan reducir el rendimiento, provocar errores en el software, tener una excesiva complejidad, complicar el flujo de datos, crear problemas de seguridad.

El análisis se realiza siguiendo una serie de reglas predefinidas.

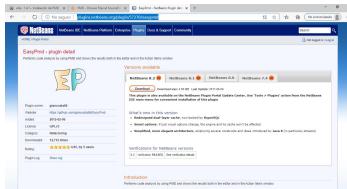
Un ejemplo es **PMD**, una herramienta para **Java** que basa su funcionamiento en detectar patrones, que son posibles errores en tiempo de ejecución, código que no se puede ejecutar nunca porque no se puede llegar a él, código que puede ser optimizado, expresiones lógicas que pueden ser simplificadas, malos usos del lenguaje, etc.

Otro analizador de código disponible en el mercado es **Sonarcube**, herramienta Open-Source de análisis de calidad del código disponible para gran cantidad de lenguajes de programación.

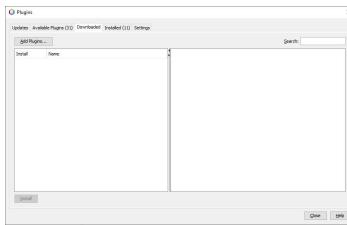
1.5.1.- Instalación de PMD

Los analizadores de código estático, se suelen integrar en los Entornos de Desarrollo, aunque en algunos casos, hay que instalarlos como plug-in, tras instalar el IDE. En el caso de Netbeans vamos a instalar el plug-in para PMD. Para ello lo descargamos de este enlace:

<http://plugins.netbeans.org/plugin/57270/easypmd>



Cómo ya vimos en la unidad anterior, para instalar dicho plugin, en NetBeans vamos a Tools/plugins/Downloaded.

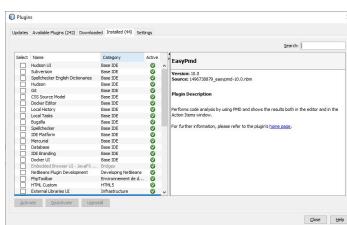


Pulsamos sobre "Add plugins"

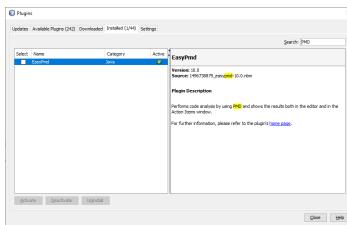
Y buscamos el fichero que nos hemos descargado, lo abrimos y lo instalamos. Ya vimos en la unidad 3 como se hacía.

Y reiniciamos NetBeans.

Vamos a Plugins/Installed:



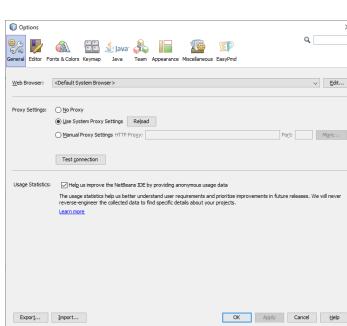
y en "Search" ponemos "PMD":



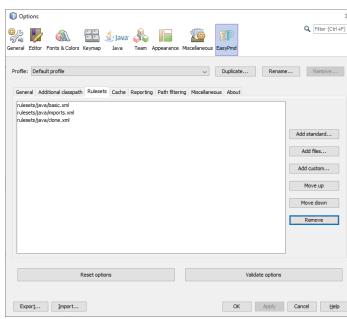
Y nos aseguramos que ha quedado activado. Si no es así, lo hacemos.

1.5.2.- Configuración.

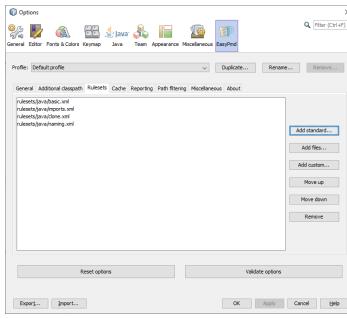
Como se indicó en el apartado anterior, PMD es un analizador de código estático, capaz de detectar automáticamente un amplio rango de defectos y de inseguridades en el código. PMD se centra en detectar defectos de forma preventiva. Una vez que tenemos desarrollado nuestro código, si queremos analizarlo con PMD, y obtener el informe del análisis, primero hay que pulsar en el menú Tools/Options:



y en "EasyPMD", en la pestaña "Rulesets":

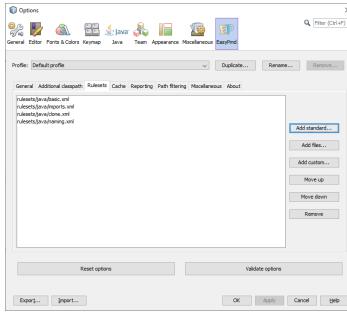


Se indican las reglas que están activadas. Puedes activas más reglas. Para ello, pinchas en "Add standard" y añades las reglas que creas conveniente. Por ejemplo, vamos a añadir la regla "Naming". La seleccionamos y aceptar.



Y veremos que esta regla se queda incluida.

Esta regla analiza, por ejemplo, que el nombre de un método no empieza por la primera letra en mayúscula, variables que no se usan o nombres muy cortos.



Mientras se analiza el código, el plug-in mostrará una barra de progreso en la esquina inferior derecha. El informe producido contiene la localización, el nombre de la regla que no se cumple y la recomendación de cómo se resuelve el problema.

El informe PMD, nos permite navegar por el fichero de clases y en la línea donde se ha detectado el problema. En el número de línea, veremos una marca PMD. Si se posiciona el ratón encima de ella, veremos un tooltip con la descripción del error.

Para probarlo, crea un proyecto en NetBeans con este código (pulsa [aquí](#) si quieres descargar dicho proyecto):

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package practicapmd;

public class PracticaPMD {

    public static void main(String[] args) {
        int a;

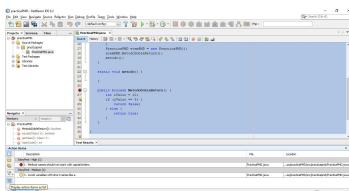
        int altura = 89;
        if (altura > 89) {
            System.out.println("Eres muy alto");
        }
        PracticaPMD ejemPMD = new PracticaPMD();
        ejemPMD.MetodoDobleReturn();
        metodo();
    }

    static void metodo() {

    }

    public boolean MetodoDobleReturn() {
        int iValor = 10;
        if (iValor == 5)
            return false;
        else {
            return true;
        }
    }
}
```

Pulsa sobre el menú "Windows" y sobre "Action Items"

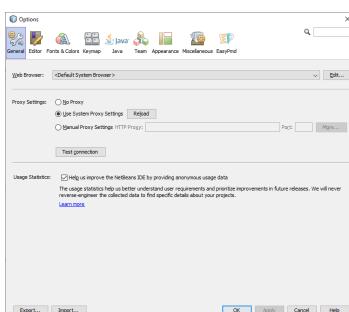


Y te da un informe de las reglas que no cumple tú código. En el ejemplo te avisa de:

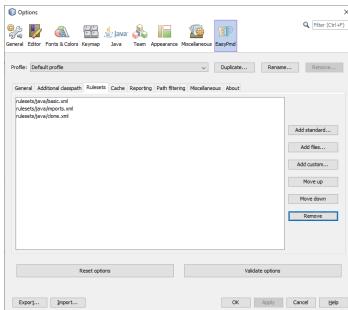
- Un error muy grave y es poner a un método un nombre que empieza por una letra mayúscula.
 - Un error leve que es el nombre de la variable "a". Un nombre muy corto.

Así podemos ir añadiendo más reglas. Cómo, por ejemplo, que me avise si alguna estructura de control no tiene llaves, aunque tenga una sola instrucción. Para ello, hay que añadir la regla: "braces".

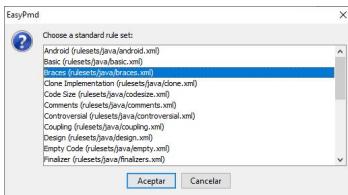
Para ello, pulsamos en el menú Tools/Options:



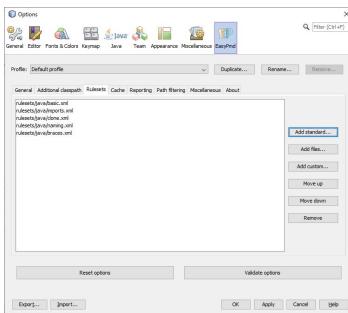
y en "EasyPMD", en la pestaña "Rulesets":



Pulsamos sobre "Add Standard" y añadimos la regla:

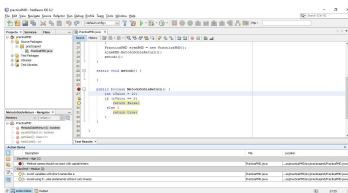


Aceptamos y tendremos esa regla añadida:

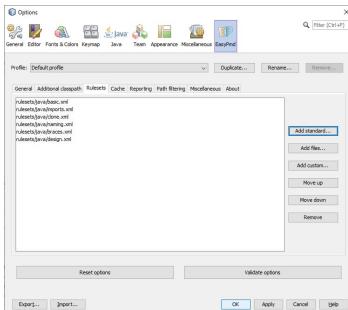


Damos a Ok.

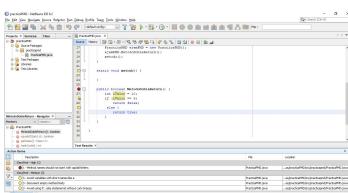
Y veremos como me avisa que hay un if que no tiene llaves:



Ahora añade la regla "Design" y veras como te avisa en el informe que tienes un método vacío.



Información del Action Items:



2.- Control de versiones.

Caso práctico



Juan y María están implementando la mayor parte de la aplicación. Continuamente está modificando el código generado, bien para mejorar algunos aspecto, o por qué han refactorizado. Hay diferentes versiones de una misma clase, de un mismo método. ¿Qué ocurre si se borra accidentalmente algún fichero? ¿Qué pasa si María modifica una clase que necesita Juan?

Cuando estamos desarrollando software, el código fuente está cambiando continuamente, bien por el propio desarrollo o por el mantenimiento a que se ve sometido. Además los proyectos en ocasiones se desarrollan por fases o se hacen diferentes entregas al cliente. Todos estos factores hacen necesario un **sistema de control de versiones**.

Las ventajas de utilizar un sistema de control de versiones son múltiples. Un sistema de control de versiones bien diseñado **facilita** al equipo de **desarrollo** su labor, permitiendo que varios programadores trabajen en el mismo proyecto (incluso sobre los mismo archivos) de forma simultánea, permitiendo gestionar los conflictos que se puedan producir por actualizaciones simultáneas sobre el mismo código. Las herramientas de control de versiones proveen de un sitio central donde almacenar el código fuente de la aplicación, así como el historial de cambios realizados a lo largo de la vida del proyecto.

También permite a los desarrolladores volver a versiones estables previas del código fuente si es necesario.

Una **versión**, desde el punto de vista de la evolución, se define como la forma particular de un objeto en un instante o contexto dado. Se denomina **revisión**, cuando se refiere a la evolución en el tiempo.

Pueden coexistir varias versiones alternativas en un instante dado y hay que disponer de un método, para designar las diferentes versiones de manera organizada.

Existen distintas alternativas de control de versiones de código abierto, **GIT**, **CVS**, **Subversion**, **Mercurial**... En ocasiones podrán ser motivo de gestión proyectos software en desarrollo o simplemente conjuntos de archivos de algún otro uso.

Un repositorio interesante para la gestión de proyectos es **Bitbucket**, está basado en Git y dispone de una versión gratuita para equipos de 5 personas. Ideal para gestionar el proyecto fin de ciclo.

Para gestionar las distintas versiones que se van generando durante el desarrollo de una aplicación, los **IDE**, proporcionan herramientas de Control de Versiones y facilitan el desarrollo en equipo de aplicaciones.

2.1.- Tipos de herramientas de control de versiones.

En acuerdo al modo de organizar la información, se pueden clasificar las herramientas de control de versiones como:

- **Sistemas locales:** se trata del control de versiones donde la información se guarda en diferentes directorios en función de sus versiones. Toda la gestión recae sobre el responsable del proyecto y no se dispone de herramientas que automatizan el proceso. Es viable para pequeños proyectos donde el trabajo es desarrollado por un único programador.
- **Sistemas centralizados:** responden a una arquitectura cliente-servidor. Un único equipo tiene todos los archivos en sus diferentes versiones, y los clientes replican esta información en sus entornos de trabajo locales. El principal inconveniente es que el servidor es un dispositivo crítico para el sistema ante posibles fallos.
- **Sistemas distribuidos:** en este modelo cada sistema hace con una copia completa de los ficheros de trabajo y de todas sus versiones. El rol de todos los equipos es de igual a igual y los cambios se pueden sincronizar entre cada par de copias disponibles. Aunque técnicamente todos los repositorios tienen la posibilidad de actuar como punto de referencia; habitualmente funcionan siendo uno el repositorio principal y el resto asumiendo un papel de clientes sincronizando sus cambios con éste.

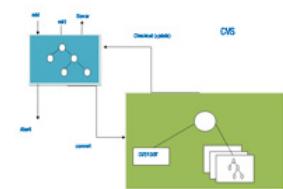


2.2.- Estructura de herramientas de control de versiones.

Caso práctico



Para Ana el concepto de control de versiones le resulta muy abstracto. Juan va a intentar explicarle como funciona el mecanismo de control de versiones, tomando como ejemplo una herramienta que él conoce: CVS.



Las herramientas de control de versiones, suelen estar formadas por un conjunto de elementos, sobre los cuales, se pueden ejecutar órdenes e intercambiar datos entre ellos. Como ejemplo, vamos a analizar la herramienta CVS.

Una herramienta de control de versiones, como CVS, es un sistema de mantenimiento de código fuente (grupos de archivos en general) extraordinariamente útil para grupos de desarrolladores que trabajan cooperativamente usando alguna clase de red. CVS permite a un grupo de desarrolladores trabajar y modificar concurrentemente ficheros organizados en proyectos. Esto significa que dos o más personas pueden modificar un mismo fichero sin que se pierdan los trabajos de ninguna. Además, las operaciones más habituales son muy sencillas de usar.

CVS utiliza una arquitectura cliente-servidor: un servidor guarda la versión actual del proyecto y su historia, y los clientes conectan al servidor para sacar una copia completa del proyecto, trabajar en esa copia y entonces ingresar sus cambios. Típicamente, cliente y servidor conectan utilizando Internet, pero cliente y servidor pueden estar en la misma máquina. El servidor normalmente utiliza un sistema operativo similar a Unix, mientras que los clientes CVS pueden funcionar en cualquier de los sistemas operativos más difundidos.

Los clientes pueden también comparar diferentes versiones de ficheros, solicitar una historia completa de los cambios, o sacar una "foto" histórica del proyecto tal como se encontraba en una fecha determinada o en un número de revisión determinado. Muchos proyectos de código abierto permiten el "acceso de lectura anónimo", significando que los clientes pueden sacar y comparar versiones sin necesidad de teclear una contraseña; solamente el ingreso de cambios requiere una contraseña en estos escenarios. Los clientes también pueden utilizar el comando de actualización con el fin de tener sus copias al día con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.

El sistema de control de versiones está formado por un conjunto de componentes:

Repositorio: es el lugar de almacenamiento de los datos de los proyectos. Suele ser un directorio en algún ordenador.

Módulo: en un directorio específico del repositorio. Puede identificar una parte del proyecto o ser el proyecto por completo.

Revisión: es cada una de las versiones parciales o cambios en los archivos o repositorio completo. La evolución del sistema se mide en revisiones. Cada cambio se considera incremental.

Etiqueta: información textual que se añade a un conjunto de archivos o a un módulo completo para indicar alguna información importante.

Rama: revisiones paralelas de un módulo para efectuar cambios sin tocar la evolución principal. Se suele emplear para pruebas o para mantener los cambios en versiones antiguas.

Algunos de los **servicios** que típicamente proporcionan son:

- **Creación de repositorios.** Creación del esqueleto de un repositorio sin información inicial del proyecto.
- **Clonación de repositorios.** La clonación crea un nuevo repositorio y vuela la información de algún otro repositorio ya existente. Crea una réplica.
- **Descarga de la información del repositorio principal al local.** Sincroniza la copia local con la información disponible en el repositorio principal.
- **Carga de información al repositorio principal desde el local.** Actualiza los cambios realizados en la copia local en el repositorio principal.
- **Gestión de conflictos.** En ocasiones, los cambios que se desean consolidar en el repositorio principal entran en conflicto con otros cambios que hayan sido subidos por algún otro desarrollador. Cuando se da esta situación, las herramientas de control de versiones tratan de combinar automáticamente todos los cambios. Si no es posible sin pérdida de información, muestra al programador los conflictos acontecidos para que sea éste el que tome la decisión de cómo combinarlos.
- **Gestión de ramas.** Creación, eliminación, integración de diferencias entre ramas, selección de la rama de trabajo.
- **Información sobre registro de actualizaciones.**
- **Comparativa de versiones.** Genera información sobre las diferencias entre versiones del proyecto.

Las órdenes que se pueden ejecutar son:

checkout: obtiene una copia del trabajo para poder trabajar con ella.

Update: actualiza la copia con cambios recientes en el repositorio.

Commit: almacena la copia modificada en el repositorio.

Abort: abandona los cambios en la copia de trabajo.

2.2.1.- Repositorio.

Caso práctico



Juan le ha explicado a Ana la los fundamentos del control de versiones, pero quiere profundizar más en el concepto de repositorio, ya que para él es la parte fundamental del control de versiones.

El repositorio es la parte fundamental de un sistema de control de versiones. Almacena toda la información y datos de un proyecto.

El repositorio es un almacén general de versiones. En la mayoría de las herramientas de control de versiones, suele ser un directorio.

El repositorio centraliza todos los componentes de un mismo sistema, incluyendo las distintas versiones de cada componente. Con el repositorio, se va a conseguir un ahorro de espacio de almacenamiento, ya que estamos evitando guardar por duplicado, los elementos que son comunes a varias versiones. El repositorio nos va a facilitar el almacenaje de la información de la evolución del sistema, ya que, aparte de los datos en sí mismo, también almacena información sobre las versiones, temporización, etc.

El entorno de desarrollo integrado Netbeans usa como sistema de control de versiones CVS. Este sistema, tiene un componente principal, que es el repositorio. En el repositorio se deberán almacenar todos los ficheros de los proyectos, que puedan ser accedidos de forma simultánea por varios desarrolladores.

Cuando usamos una sistema de control de versiones, trabajamos de forma local, sincronizándonos con el repositorio, haciendo los cambios en nuestra copia local, realizado el cambio, se acomete el cambio en el repositorio. Para realizar la sincronización, en el entorno Netbeans, lo realizamos de varias formas:

Abriendo un proyecto CVS en el IDE.

Comprobando los archivos de un repositorio.

Importando los archivos hacia un repositorio.

Si tenemos un proyecto CVS versionado, con el que hemos trabajado, podemos abrirlo en el IDE y podremos acceder a las características de versionado. El IDE escanea nuestro proyectos abiertos y si contienen directorios CVS, el estado del archivo y la ayuda-contextual se activan automáticamente para los proyectos de versiones CVS.

Autoevaluación

¿Qué afirmación sobre control de versiones es correcta?

- Solo puede existir una única versión de una clase.
- El almacenamiento de versiones es local a cada máquina.
- El repositorio centraliza el almacenamiento de los datos.

Incorrecta. Podemos tantas versiones como se deseé.

No es correcta. El repositorio es un lugar común y único a todas las máquinas y colaboradores de un proyecto.

Muy bien. El repositorio es la parte fundamental en el control de versiones, ya que almacena todos los datos de los proyectos.

Solución

1. Incorrecto
2. Incorrecto

3. Opción correcta

2.2.2.- Gestión de versiones y entregas.

Caso práctico



María ha desarrollado varias versiones del módulo de reservas de la aplicación de Gestión Hotelera. Le explica a **Carlos** como ha sido la evolución de cada versión del módulo, desde la primera versión, hasta la versión actual, que ella cree definitiva.



Las versiones hacen referencia a la evolución de un único elemento, dentro de un sistema software. La evolución puede representarse en forma de grafo, donde los nodos son las versiones y los arcos corresponden a la creación de una versión a partir de otra ya existente.

Grafo de evolución simple: las revisiones sucesivas de un componente dan lugar a una simple secuencia lineal. Esta evolución no presenta problemas en la organización del repositorio y las versiones se designan mediante números correlativos.

Variantes: en este caso, existen varias versiones del componente. El grafo ya no es una secuencia lineal, si no que adopta la forma de un árbol. La numeración de las versiones requerirá dos niveles. El primer número designa la variante (línea de evolución) y el segundo la versión particular (revisión) a lo largo de dicha variante.

La terminología que se usa para referirse a los elementos del grafo son:

Tronco (trunk): es la variante principal.

Cabeza (head): es la última versión del tronco.

Ramas (branches): son las variantes secundarias.

Delta: es el cambio de una revisión respecto a la anterior.

Propagación de cambios: cuando se tienen variantes que se desarrollan en paralelo, suele ser necesario aplicar un mismo cambio a varias variantes.

Fusión de variantes: en determinados momentos puede dejar de ser necesario mantener una rama independiente. En este caso se puede fundir con otra (MERGE).

Técnicas de almacenamiento: como en la mayoría de los casos, las distintas versiones tienen en común gran parte de su contenido, se organiza el almacenamiento para que no se desaproveche espacio repitiendo los datos en común de varias versiones.

Deltas directos: se almacena la primera versión completa, y luego los cambios mínimos necesarios para reconstruir cada nueva versión a partir de la anterior.

Deltas inversos: se almacena completa la última versión del tronco y los cambios necesarios para reconstruir cada versión anterior a partir de la siguiente. En las ramas se mantiene el uso de los deltas directos.

Marcado selectivo: se almacena el texto refundido de todas las versiones como una secuencia lineal, marcando cada sección del conjunto con los números de versiones que corresponde.

En cuanto a la **gestión de entregas**, en primer lugar definimos el concepto de entrega como una instancia de un sistema que se distribuye a los usuarios externos al equipo de desarrollo.

La planificación de la entrega se ocupa de cuándo emitir una versión del sistema como una entrega. La entrega está compuesta por el conjunto de programas ejecutables, los archivos de configuración que definen como se configura la entrega para una instalación particular, los archivos de datos que se necesitan para el funcionamiento del sistema, un programa de instalación para instalar el sistema en el hardware de destino, documentación electrónica y en papel, y, el embalaje y publicidad asociados, diseñados para esta entrega. Actualmente los sistemas se entregan en discos ópticos (CD o DVD) o como archivos de instalación descargables desde la red.

2.3.- Herramientas de control de versiones.

Caso práctico



María quiere que **Carlos** conozca las herramientas de control de versiones que integra Netbeans, ya que es el Entorno que utilizan para su desarrollo. Dado que estamos utilizando un Entorno de Desarrollo Integrado, **Carlos** debe conocer las herramientas que incorpora Netbeans.

Durante el proceso de desarrollo de software, donde todo un equipo de programadores están colaborando en el desarrollo de un proyecto software, los cambios son continuos. Es por ello necesario que existan en todos los lenguajes de programación y en todos los entornos de programación, herramientas que gestionen el control de cambios.

Si nos centramos en Java, actualmente destacan dos herramientas de control de cambios: CVS y Subversion. CVS es una herramienta de código abierto ampliamente utilizada en numerosas organizaciones. Subversion es el sucesor natural de CVS, está rápidamente integrándose en los nuevos proyectos Java, gracias a sus características que lo hacen adaptarse mejor a las modernas prácticas de programación Java. Estas dos herramientas de control de cambios, se integran perfectamente en los entornos de desarrollado para Java, como Netbeans y Eclipse.

Otras herramientas de amplia difusión son:

SourceSafe: es una herramienta que forma parte del entorno de desarrollo Microsoft Visual Studio.

Visual Studio Team Foundation Server: es el sustituto de Source Safe. Es un productor que ofrece control de código fuente, recolección de datos, informes y seguimiento de proyectos, y está destinado a proyectos de colaboración de desarrollo de software.

Darcs: es un sistema de gestión de versiones distribuido. Algunas de sus características son: la posibilidad de hacer commits locales (sin conexión), cada repositorio es una rama en sí misma, independencia de un servidor central, posibilidad de renombrar ficheros, varios métodos de acceso como local, ssh, http y ftp, etc.

Git: esta herramienta de control de versiones, diseñada por Linus Torvalds.

Mercurial: esta herramienta funciona en Linux, Windows y Mac OS X. Es un programa de línea de comandos. Es una herramienta que permite que el desarrollo se haga distribuido, gestionando de forma robusta archivos de texto y binarios. Tiene capacidades avanzadas de ramificación e integración. Es una herramienta que incluye una interfaz web para su configuración y uso.



Autoevaluación

¿Qué herramienta no es una herramienta de Control de Versiones?

- Subversion.
- CVS.
- Mercurial.
- PMD.

Incorrecta.

No es cierto.

No es correcta.

Correcta. PMD es un analizador de código.

Solución

1. Incorrecto
2. Incorrecto

- 3. Incorrecto
- 4. Opción correcta

Debes conocer

También debes visitar el siguiente enlace donde se puede ver la guía de uso de subversión en Netbeans. Está en inglés pero es conveniente que le eches un vistazo para conocer cómo funciona Subversion.

[Guía de uso de Subversion en Netbeans.](#)

Autoevaluación

¿Qué cliente de Gestión de Versiones no incorpora Netbeans?

- VS Team Foundation.
- CVS.
- Mercurial.

Muy bien. Visual Studio Team Foundation, está diseñada para el trabajo colaborativo y control de versiones en Microsoft Visual Studio.

No es correcta.

Incorrecta.

Solución

- 1. Opción correcta
- 2. Incorrecto
- 3. Incorrecto

Caso práctico



Juan le va a enseñar a Ana los clientes de control de versiones que hay en Netbeans, para que aprenda a utilizarlos e integrarlos en los proyectos que realice de ahora en adelante.

2.4.- Planificación de la gestión de configuraciones.

Caso práctico

El equipo de desarrollo de BK Programación decide reunirse para planificar la gestión de configuraciones, ya que la aplicación de Gestión Hotelera es amplia y compleja, y continuamente se están diseñando nuevos módulos, clases o métodos.



La Gestión de Configuraciones del software (GCS) es un conjunto de actividades desarrolladas para gestionar los cambios a lo largo del ciclo de vida.

La planificación de las Gestión de Configuraciones del software, está regulado en el estándar IEEE 828.

Cuando se habla de la gestión de configuraciones, se está haciendo referencia a la evolución de todo un conjunto de elementos. Una configuración es una combinación de versiones particulares de los componentes que forman un sistema consistente. Desde el punto de vista de la evolución en el tiempo, es el conjunto de las versiones de los objetos componentes en un instante dado.

Una configuración puede cambiar porque se añaden, eliminan o se modifican elementos. También puede cambiar, debido a la reorganización de los componentes, sin que estos cambien.

Como consecuencia de lo expuesto, es necesario disponer de un método, que nos permita designar las diferentes configuraciones de manera sistemática y planificada. De esta forma se facilita el desarrollo de software de manera evolutiva, mediante cambios sucesivos aplicados a partir de una configuración inicial hasta llegar a una versión final aceptable del producto.

La Gestión de Configuraciones de Software se va a componer de cuatro tareas básicas:

1. **Identificación.** Se trata de establecer estándares de documentación y un esquema de identificación de documentos.
2. **Control de cambios.** Consiste en la evaluación y registro de todos los cambios que se hagan de la configuración software.
3. **Auditorías de configuraciones.** Sirven, junto con las revisiones técnicas formales para garantizar que el cambio se ha implementado correctamente.
4. **Generación de informes.** El sistema software está compuesto por un conjunto de elementos, que evolucionan de manera individual, por consiguiente, se debe garantizar la consistencia del conjunto del sistema.

2.5.- Gestión del cambio.

Caso práctico



Para gestionar el control de versiones de forma centralizada, **Ada** va a supervisar los cambios de versión que el equipo de **Juan** y de **María** están efectuando de forma independiente. **Ada** va a realizar una gestión del cambio centralizada y organizada.

Las herramientas de control de versiones no garantizan un desarrollo razonable, si cualquier componente del equipo de desarrollo de una aplicación puede realizar cambios e integrarlos en el repositorio sin ningún tipo de control. Para garantizar que siempre disponemos de una línea base para continuar el desarrollo, es necesario aplicar controles al desarrollo e integración de los cambios. El control de cambios es un mecanismo que sirve para la evaluación y aprobación de los cambios hechos a los elementos de configuración del software.

Pueden establecerse distintos tipos de control:

1. Control individual, antes de aprobarse un nuevo elemento.

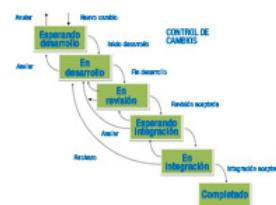
Cuando un elemento de la configuración está bajo control individual, el programador responsable cambia la documentación cuando se requiere. El cambio se puede registrar de manera informal, pero no genera ningún documento formal.

2. Control de gestión u organizado, conduce a la aprobación de un nuevo elemento.

Implica un procedimiento de revisión y aprobación para cada cambio propuesto en la configuración. Como en el control individual, el control a nivel de proyecto ocurre durante el proceso de desarrollo pero es usado después de que haya sido aprobado un elemento de la configuración software. El cambio es registrado formalmente y es visible para la gestión.

3. Control formal, se realiza durante el mantenimiento.

Ocurre durante la fase de mantenimiento del ciclo de vida software. El impacto de cada tarea de mantenimiento se evalúa por un Comité de Control de Cambios, el cuál aprueba la modificación de la configuración software.



Autoevaluación

¿Cuál de las siguientes no es una tarea básica de la Gestión de Configuraciones del Software?

- Control de cambios.
- Generación de informes.
- Auditorías de configuraciones.
- Gestión del repositorio.

Incorrecta. Hay que controlar cualquier cambio en la configuración.

No es cierto. Los informes son necesarios en la Gestión de Configuraciones de Software.

No es correcta. Garantizan que el cambio se ha realizado correctamente.

Muy bien. El repositorio es un elemento del control de versiones.

Solución

1. Incorrecto

- 2. Incorrecto
- 3. Incorrecto
- 4. Opción correcta

3.- Documentación.

Caso práctico



Juan y María saben de la importancia de tener documentado el código y todo el proceso de desarrollo de software. Al mismo tiempo que codifican y refactorizan, dejan constancia documental de lo que van haciendo. Como están desarrollando con Netbeans, y con el objetivo de facilitar su trabajo de documentación, van a utilizar JavaDoc. **Ana y Antonio** consiguen entender el funcionamiento de la aplicación, y la función de cada método, gracias a los comentarios que insertan en el código los dos programadores principales.

El proceso de documentación de código, es uno de los aspectos más importantes de la labor de un programador. Documentar el código nos sirve para explicar su funcionamiento, punto por punto, de forma que cualquier persona que lea el comentario, puede entender la finalidad del código.

En la actualidad, el desarrollo rápido de aplicaciones, en muchos casos, va en detrimento de una buena documentación del código. Si el código no está documentado, puede resultar bastante difícil de entender, y por tanto de solucionar errores y de mantenerlo.

La labor de documentación es fundamental para la detección de errores y para su mantenimiento posterior, que en muchos casos, es realizado por personas diferentes a las que intervinieron en su creación. Hay que tener en cuenta que todos los programas tienen errores y todos los programas sufren modificaciones a lo largo de su vida.

La documentación añade explicaciones de la función del código, de las características de un método, etc. Debe tratar de explicar todo lo que no resulta evidente. Su objetivo no es repetir lo que hace el código, sino explicar por qué se hace.

La documentación explicará cuál es la finalidad de un clase, de un paquete, qué hace un método, para qué sirve una variable, qué se espera del uso de una variable, qué algoritmo se usa, por qué hemos implementado de una manera y no de otra, qué se podría mejorar en el futuro, etc.

Para saber más

El siguiente enlace nos muestra el estilo de programación a seguir en Java, así como la forma de documentar y realizar comentarios de un código. (En inglés)

[Documentación y comentarios en Java](#)

3.1.- Uso de comentarios.

Caso práctico



Carlos está aprendiendo muchas cosas con **María**. Sin embargo hay algunos métodos y clases que ha implementado **María**, y que no logra entender. **María** le comenta que va a incluir comentarios en su código, y le va a enseñar la forma correcta de hacerlo.

Caso práctico



Juan le explica a **María**, que para documentar el software, existen diferentes formas de hacerlo y distintas herramientas en el mercado que automatizan la tarea de documentación.

Uno de los elementos básicos para documentar código, es el uso de comentarios. Un comentario es una anotación que se realiza en el código, pero que el compilador va a ignorar, sirve para indicar a los desarrolladores de código diferentes aspectos del código que pueden ser útiles.

Es la primera alternativa que surge para documentar código. Con los comentarios, documentamos la funcionalidad de una línea de código, de un método o el comportamiento de una determinada clase.

En principio, los comentarios tienen dos propósitos diferentes:

Explicar el objetivo de las sentencias. De forma que el programador o programadora, sepa en todo momento la función de esa sentencia, tanto si lo diseñaron como si son otros los que quieren entenderlo o modificarlo.

Explicar qué realiza un método, o clase, no cómo lo realiza. En este caso, se trata de explicar los valores que va a devolver un método, pero no se trata de explicar cómo se ha diseñado.

En el caso del lenguaje Java, C# y C, los comentarios, se implementan de forma similar. Cuando se trata de un comentario de una sola línea, se usan los caracteres // seguidos del comentario. Para comentarios multilínea, los caracteres a utilizar son /* y */, quedaría: /* comentario-multilínea */.

No es obligatorio, pero en muchas situaciones es conveniente, poner los comentarios al principio de un fragmento de código que no resulta lo suficientemente claro, a la largo de bucles, o si hay alguna línea de código que no resulta evidente y pueda llevarnos a confusión.

Insertando comentario en el código más difícil de entender, y utilizando la documentación generada por alguna de las herramientas citadas anteriormente, se genera la suficiente información para ayudar a cualquier nuevo programador o programadora.

Hay que tener en cuenta, que si el código es modificado, también se deberán modificar los comentarios.

```
/* Método para ingresar cantidad en la cuenta. Modifica el saldo.
 * Esta método va a ser probado con JUnit.
 */
// Comentarios estilo javadoc
public void ingresar(int cantidad) {
    if (cantidad < 0) {
        throw new Exception("No se puede ingresar una cantidad negativa");
    }
    saldo += cantidad;
    System.out.println("Ingresó de una cantidad de dinero.");
}
```

3.2.- Documentación de clases.

Caso práctico



Juan va a utilizar JavaDoc para documentar las clases que ha desarrollado. Ana se da cuenta de la ayuda tan importante que ofrece, tanto para el futuro mantenimiento de la aplicación como para entender su funcionamiento, tener documentadas las clases.

Existen diferentes herramientas que permiten automatizar, completar y enriquecer nuestra documentación. Podemos citar JavaDoc, SchemeSpy y Doxygen, que producen una documentación actualizada, precisa y utilizable en línea, incluyendo además, con SchemeSpy y Doxygen, modelos de bases de datos gráficos y diagramas.

Las clases que se implementan en un aplicación, deben de incluir comentarios. Al utilizar un entorno de programación para la implementación de la clase, debemos seguir una serie de pautas, muchas de las cuales las realiza el IDE de forma transparente, en el diseño y documentación del código.

Los entornos de programación que implementa Java, como Eclipse o Netbeans, incluyen una herramienta que va a generar páginas HTML de documentación a partir de los comentarios incluidos en el código fuente. La herramienta ya se ha indicado en los puntos anteriores, y es JavaDoc.

Para que JavaDoc pueda generar las páginas HTML es necesario seguir una serie de normas de documentación en el código fuente, estas son:

Los comentarios son obligatorios con JavaDoc, y se deben incorporar al principio de cada clase, al principio de cada método y al principio de cada variable de clase. Se escriben empezando por `/**` y terminando con `*/`, estos comentarios pueden ocupar varias líneas. Todos los comentarios hechos con `//` y `/*comentario*/` no se incluirán en la documentación de la clase.

Los comentarios pueden ser a nivel de clase, a nivel de variable y a nivel de método.

La documentación se genera para métodos public y protected.

Se puede usar `param` tag para documentar diferentes aspectos determinados del código, como parámetros.

Con el uso de los entornos de desarrollo, las etiquetas se añaden de forma automática, estableciendo el `@author` y la `@version` de la clase de forma transparente al programador-programadora. También se suele añadir la etiqueta `@see`, que se utiliza para referenciar a otras clases y métodos.

Dentro de la la clase, también se documentan los constructores y los métodos.

Existe una serie de etiquetas que fijan como se presentará la información en la documentación resultante JavaDoc. En la url <https://en.wikipedia.org/wiki/Javadoc> aparece una colección de palabras reservadas (**etiquetas**) definidas en Javadoc. A continuación se muestran algunas de las más utilizadas.

Etiqueta y parámetros	Uso	Asociada a
<code>@author nombre</code>	Nombre del autor (programador)	Clase, interfaz
<code>@version numero-version</code>	Comentario con datos indicativos del número de versión.	Clase, interfaz
<code>@since numero-version</code>	Fecha desde la que está presente la clase.	Clase, interfaz, campo, método.
<code>@see referencia</code>	Permite crear una referencia a la documentación de otra clase o método.	Clase, interfaz, campo, método.
<code>@param</code> seguido del nombre del parámetro	Describe un parámetro de un método.	Método.
<code>@return descripción</code>	Describe el valor devuelto de un método.	Método.
<code>@exception clase descripción</code> <code>@throws clase descripción</code>	Comentario sobre las excepciones que lanza.	Método.
<code>@deprecated descripción</code>	Describe un método obsoleto.	Clase, interfaz, campo, método.

Los campos de una clase, también pueden incluir comentarios, aunque no existen etiquetas obligatorias en JavaDoc.

Caso práctico



Para documentar el código, el equipo de desarrollo de BK Programación, ha decidido utilizar JavaDoc, por lo que todos los componentes del equipo de desarrollo, debe familiarizarse con la herramienta.

Autoevaluación

Un comentario en formato JavaDoc.

- Utiliza los caracteres //
- Comienzan con /* y termina por */
- Comienza por /** y terminan por */

Incorrecta. Serían para los comentarios en línea.

No es correcta.

Muy bien. Es la manera de comenzar y termina un comentario con JavaDoc.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

Anexo I.- Repositorio CVS.

Conectar con un repositorio.

Si queremos conectar con un repositorio remoto desde el IDE, entonces chequearemos los ficheros e inmediatamente comenzará a trabajar con ellos. Se hace de la siguiente forma:

1. En el IDE NetBeans elegimos Herramientas - CVS - Extraer. El asistente para extraer el modulo se nos abrirá.
2. En el primer panel del asistente, se introduce la localización del repositorio definido con cvsroot. El IDE soporta diferentes formatos de cvsroot, dependiendo de si el repositorio es local o remoto, y del método utilizado para conectarnos a él.

Métodos de conexión a cvsroot

Método	Descripción	Ejemplo
<u>pserver</u>	Contraseña de servidor remoto.	:pserver:username@hostname:/repository_path
<u>ext</u>	Acceso usando Remote Shell (RSH) o Secure Shell (SSH).	:ext:username@hostname:/repository_path
<u>local</u>	Acceso a un repositorio local.	:local:/repository_path (requiere un <u>CVS</u> externo ejecutable)
<u>fork</u>	Acceso a un repositorio local usando un protocolo remoto.	:fork:/repository_path (requiere un <u>CVS</u> externo ejecutable)

3. En el panel de Módulos a extraer del asistente, especificamos el módulo que queremos extraer, en el campo **Módulo**. Si no sabemos el nombre del módulo, podemos extraerlo haciendo clic en el botón Examinar. Si lo que queremos es conectar a un repositorio remoto desde el IDE, debemos extraer los ficheros e inmediatamente trabajar con ellos, se hace de la siguiente forma:



4. En el campo de texto, ponemos el nombre del Módulo que queremos extraer. Podemos usar el botón **Examinar**, para buscar aquellos módulos disponibles.
5. En el campo **Carpeta local**, pondremos la ruta de nuestro ordenador donde queremos extraer los archivos. Pulsaremos el botón **Terminar** para que el proceso comience.

Importar archivos hacia un repositorio.

Alternativamente, podemos importar un proyecto en el que estemos trabajando en el IDE hacia un repositorio remoto, seguiremos trabajando en el IDE después de que haya sido versionado con el repositorio CVS.

Para importar un proyecto a un repositorio:

1. Desde la ventana de proyectos, seleccionamos un proyecto que no este versionado, y elegimos Equipo - CVS - Importar al depósito. Se abrirá el asistente de importación CVS.
2. En el panel Raíz CVS del asistente, se especifica la localización del repositorio definido como cvsroot. Dependiendo del método usado, necesitaremos utilizar más información, como la contraseña o configuración del proxy, para conectarnos a un repositorio remoto.
3. En **Carpeta a importar**, se especifica el directorio local donde queremos colocar el repositorio.

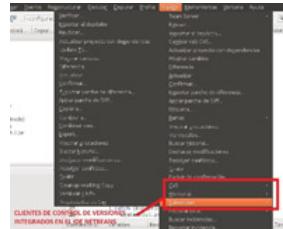


4. En el área de texto **Importar mensaje**, pondremos una descripción del proyecto que estamos importando.
5. Se especifica la localización del repositorio donde queremos importar el proyecto, escribiendo la ruta en **Carpeta del depósito**. De forma alternativa, se puede hacer clic en el botón **Examinar** para ir a una localización específica en el repositorio. Al hacer clic en el botón **Terminar** se inicia la importación. El IDE copiará los ficheros del proyecto en el repositorio.

Anexo II.- Clientes de Control de Versiones en Netbeans.

El entorno de desarrollo integrado NetBeans 6.9.1., incorpora tres clientes de control de versiones. Estos tres clientes son CVS, Subversion y Mercurial.

Cuando estemos desarrollando una aplicación, si queremos gestionar el control de versiones, podemos utilizar cualquiera de los tres.



CVS

En el caso del IDE NetBeans, es importante su uso para mantener de forma segura, tanto los programas como las pruebas, en un repositorio de código. La configuración de CVS puede suponer unos cinco minutos, que dentro del tiempo de desarrollo de una aplicación, es un tiempo despreciable. Sin embargo, el uso de CVS en NetBeans, nos va a gestionar las distintas versiones del código que se desarrollen y nos va a evitar pérdidas de datos y de código.

Para utilizar CVS, es necesario la instalación de un cliente CVS, sin embargo, NetBeans nos va a evitar esta instalación, ya que incorpora un cliente CVS en Java.

CVS se puede utilizar con NetBeans de tres formas:

1. Desde la línea de comando CVS, donde se escribirán los comandos CVS y de esta forma se interactuará con el repositorio.
2. Si CVS está incorporado a NetBeans, dispondremos de un conjunto de clases Java que imitan los comandos clásicos de CVS.
3. Con fórmulas de línea de comandos CVS genéricas basadas en plantillas proporcionadas por el usuario, que son pasadas al shell del sistema operativo.

En los casos anteriores, donde se utilizan plantillas, las plantillas están parametrizadas, con la lógica NetBeans, que sustituye los operadores CVS por variables. Esto quiere decir, que NetBeans se va a encargar de decidir el nombre de los ficheros, que datos se importan o exportan del repositorio, sin que el usuario deba conocer toda la lógica interna de CVS. Nos evitamos tener que conocer todos los parámetros y opciones necesarias para realizar las operaciones con CVS.

SUBVERSION

Subversion es un sistema de control de versiones de software libre, que se ha convertido en el sustituto natural de CVS. A diferencia de CVS, los archivos que se versionan no tienen un número de versión independiente, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado.

Subversion puede acceder al repositorio a través de redes. Esto implica que varias personas pueden acceder, modificar y administrar el mismo conjunto de datos, con lo que se va a fomentar la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no se compromete la calidad del software que se desarrolla.

Recuerda visitar la guía de Subversion que te sugerimos en la unidad de trabajo.

MERCURIAL

Mercurial es un sistema de control de versiones que utiliza sobre todo la línea de comandos. Todas las operaciones de Mercurial se invocan como opciones dadas a su motor hg. Las principales metas de Mercurial incluyen un gran rendimiento y escalabilidad, desarrollo completamente distribuido, sin necesidad de un servidor, gestión robusta de archivos de texto y binarios, y capacidades avanzadas de ramificación e integración.

Anexo III.- GIT

En un proyecto de trabajo colaborativo, cada miembro del equipo realiza las tareas que le han sido asignadas, pero al terminar cada uno de ellos, habrá que preguntarse:

- ¿Cómo se integran todas las partes?.
- Si hay errores o cambios, ¿cómo se actualizan los nuevos cambios?.
- Si hay una nueva versión, ¿cómo se gestionan los conjuntos de módulos compatibles con cierta versión?, ¿cómo se recuperan versiones anteriores?, ¿cómo se fusionan versiones? ...

En el punto 2.3 hemos nombrado las diferentes herramientas de control de versiones. Vamos a ver en este anexo, en detalle, la herramienta "GIT".

Algunas de las características de **GIT** son:

- Gratis, de código abierto.
- Muy popular, disponible en múltiples plataformas.
- Distribuido. Las diferentes réplicas de los repositorios tienen una relación de igual a igual. Esta es una consideración técnica, en la práctica una de ellas suele adquirir el rol de repositorio principal.
- Basado en changesets. Si varios componentes del proyecto han cambiado respecto a la última versión, todos ellos son volcados al repositorio de una vez, esta entrega se considera atómica.
- No bloqueante. Es posible que varios repositorios locales estén trabajando de forma simultánea sobre los mismos componentes, posteriormente habrá que integrar las modificaciones efectuadas en todos ellos para obtener un versionado común.

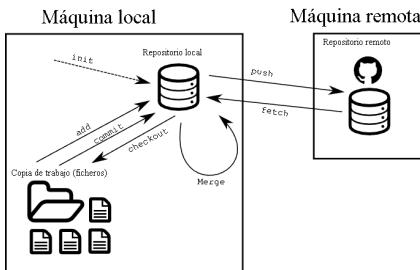
a.- Terminología

Algunos de los términos de uso habitual en **GIT** son:

- **Repositorio:** base de datos con las sucesivas versiones. Un repositorio podrá estar compuesta de diversas líneas de desarrollo o ramas. Típicamente habrá un repositorio común y tantos repositorios locales como participantes en la actividad colaborativa.
- **Rama master o trunk:** rama principal del desarrollo.
- **Rama (Branch):** línea de desarrollo paralela a la rama master. Podría recoger trabajos para diferentes clientes.
- **Área de trabajo:** ficheros sobre los que trabaja el programador.
- **Commit:** confirmación de una nueva versión (de la copia de trabajo al repositorio local).
- **Checkout:** (re)generación de la copia de trabajo a partir de la información del repositorio local.
- **Fusión (Merge):** acumular cambios en una misma versión . Permite combinar diferentes branches en el repositorio local.
- **Comentario:** cada commit deberá documentar sus efectos y motivaciones.
- **Push:** actualización del repositorio remoto con la información del repositorio local.
- **Fetch:** actualización del repositorio local con la información del repositorio remoto.
- **Pull:** actualización del repositorio local y del área de trabajo con la información del repositorio remoto.

b.- Escenario de trabajo con GIT.

En un **proyecto colaborativo** coordinado con **GIT**, un escenario típico de trabajo sería el siguiente.



En la figura se puede ver una **máquina remota** donde se encuentra el repositorio principal o remoto, sitio donde se pone en común todo el trabajo del conjunto de integrantes del equipo. Cada desarrollador tendrá su propio espacio de trabajo, en la figura se muestra una única **máquina local**.

Inicialmente, el entorno local tendrá un repositorio que será una réplica idéntica del repositorio remoto. Además el programador dispondrá de su área de trabajo, donde irá actualizando el código. Cuando unos cambios cobran suficiente entidad como para constituir una versión, éstos son guardados en el repositorio local. Finalmente, si el programador desea poner en común con los otros miembros del proyecto sus cambios, enviará sus actualizaciones desde el repositorio local al común.

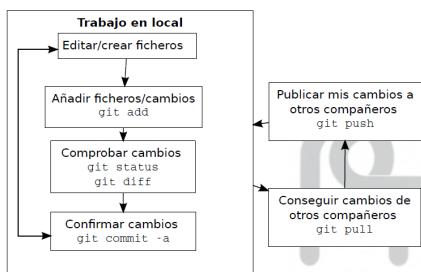
c.- Flujo de trabajo.

El **flujo de trabajo** normal del programador consistirá en las siguientes actuaciones:

- **Modificar el área de trabajo.** Actualizando el contenido de los ficheros del proyecto o creando/eliminando algunos de ellos.
- **Seleccionar aquellos ficheros** que se desea formen parte de la nueva versión del proyecto. Los ficheros que no sean seleccionados, aunque hayan sido cambiados no serán guardados en el repositorio.
- **Comprobar cambios.** Habrá que cerciorarse que los cambios introducidos en el proyecto son los deseados.
- **Salvar las modificaciones** introducidas en el área de trabajo al repositorio local.

Los cuatro puntos anteriores se podrán repetir tantas veces como sea necesario. A continuación habrá que:

- **Conseguir los cambios de otros compañeros.** Es muy probable que en el repositorio común existan actualizaciones que haya introducido algún otro colaborador, por lo que se hace necesario combinar esos cambios con los que queremos compartir nosotros.
- **Actualizar el repositorio común desde el local** para compartir nuestros cambios.



d.- Resumen de comandos GIT.

A continuación se muestran algunos de los **comandos más utilizados en GIT**.

- o [init](#)

Crea un nuevo repositorio en el directorio inicialmente vacío. Se almacena en el directorio oculto .git.

```
alvaro@debian8-64:~$ alvarogonzalez:~/aplicacion-web$ git init
Initialized empty Git repository in /home/alvaro/aplicacion-web/.git/
```

- o [clone](#)

Crea un nuevo repositorio, copia de uno ya existente. El repositorio replicado se indica mediante su *URL*. La copia de trabajo inicial será la de la rama *MASTER*

```
alvaro@debian8-64:~$ alvarogonzalez:~/aplicacion-web$ git clone https://github.com/alvarogonzalezestillo/aplicacion-php.git
Cloning into 'aplicacion-php'...
remote: Counting objects: 36, done.
remote: Compressing objects: 100% (36/36), done.
remote: Writing objects: 100% (36/36), done.
Unpacking objects: 100% (36/36), done.
Checking connectivity... done.
```

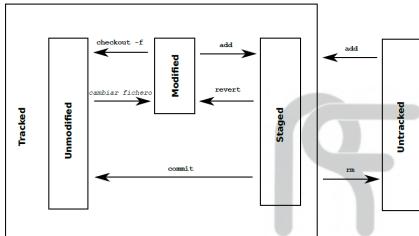
- o [add, rm, mv](#)

Permite identificar los ficheros del área de trabajo que van a ser incluidos en el repositorio. Para seleccionar ficheros se usa el comando *add*, para deseleccionarlos el comando a utilizar será *rm*. El borrado de ficheros no tiene carácter retroactivo sobre las versiones anteriores.

- o [status](#)

Imprime un resumen del estado de los ficheros de la copia de trabajo. Los ficheros se pueden encontrar en los siguientes estados:

- **Untracked:** no guardados en el repositorio. GIT ignora el fichero.
- **Unmodified:** igual que en la rama activa del repositorio.
- **Modified:** con cambios respecto al repositorio, pero que no está previsto que sean guardados al hacer *commit*.
- **Staged:** el fichero está en el *index*. Será parte del próximo *changeset* que se añadirá al repositorio.



- [commit](#)

Crea una nueva versión en el repositorio local, incluyendo los cambios en estado *Staged* procedentes del área de trabajo.

Los ficheros pasan a estar seleccionados como parte de las nuevas versiones con el comando *add*. El comando *git commit -a* incluye en el *index* todos los ficheros del área de trabajo diferentes a los disponibles en el repositorio local, a continuación aplica un *commit* creando una nueva versión.

```
alvaro@alvaro-vain:~/aplicacion-web$ git commit -a
[master 906fa20] Cambios varios permiso de ejecución
 5 files changed, 411 insertions(+), 2 deletions(-)
 create mode 100755 bin/eliminar-archivos.sh
 mode change 100755 => 100644 borrar-temporales-latex.sh
 mode change 100755 => 100644 generar-pdf.sh
 mode change 100755 => 100644 publicar-pdf.sh
 mode change 100755 => 100644 publicar-pdfs.sh
```

- [push](#)

Sube las versiones del repositorio local a un repositorio remoto. El repositorio remoto puede establecerse con:

- o *git clone* (en este caso se denomina *origin*).
- o *git remote add*

El repositorio debe tener todas las versiones del repositorio remoto, en otro caso, deberá realizarse un *git pull* previo. Aquí tienes una referencia al comando [git remote](#).

```
alvaro@debian8-64:~$ alvarogonzalez:~/aplicacion-web$ git push
Password: 
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), done.
To https://alvarogonzalezestillo@github.com/alvarogonzalezestillo/aplicacion-web.git
 ! [rejected]        master -> master (fetch first)
error: failed to push to ref 'master', it already exists.
hint: You can push an update to 'master' with -f or --force .
hint: If you want to force the update, use -f or --force .
hint: If you want to prevent this error from happening in the future,
hint: set up a 'receive.denyDeletes' hook in your repository.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

- o Observar en la figura que el comando *push* ha dado error, el motivo es que el repositorio remoto tiene recogidos cambios de otros colaboradores que no han sido volcados al repositorio local. Si no se combinan ambos, podría producirse pérdida de información, por lo que sugiere hacer un *pull* previo al *push*.

- **checkout**

Extrae versiones del repositorio local a la copia de trabajo, la acción se puede realizar sobre diferentes elementos del repositorio:

- **Un fichero:** git checkout fichero.
- **Una versión:** git checkout hash_de_versión.
- **Un branch:** git checkout branch.

También permite crear nuevas ramas, con *git checkout -b nombre_rama*.

- **branch**

Lista, crea o elimina ramas en un repositorio.

- **Otros comandos de interés son:**

- **[git log](#):** historia de cambios de un repositorio.
- **[git diff](#):** Muestra los cambios de los ficheros en estado modified.
- **[git merge](#):** fusión de dos ramas.
- **[git gui](#):** interfaz gráfica para utilizar GIT.

e.- Gitk

Gitk es un visor del repositorio local. Permite revisar ramas, commits y merges entre otros.

Es una aplicación útil para:

- Recuperar versiones antiguas de un fichero.
- Visualizar las ramas de un repositorio.
- Hacer un seguimiento de la actividad en el repositorio.



Anexo IV.- Práctica de refactorización en Eclipse

A continuación se propone un ejercicio en el que se ponen en práctica algunas de las funciones disponibles en las opciones de menú **Código fuente** y **Refactorizar** en **Eclipse**.

a.- Enunciado

Los pasos a seguir son los siguientes:

Crear el proyecto:

- Crear un proyecto Eclipse llamado “refactor”.
- Crear un paquete “utilidades” y dentro crear la clase Circulo.
- Crear un paquete “figuras” y dentro crear la clase Test.

Nota: puedes ver el código más adelante. Pulsa [aquí](#) siquieres descargarlo.

Utilizando las opciones del menú Código fuente, resuelve los siguientes pasos:

- Generar métodos get y set para la clase Circulo.
- Corregir la tabulación del código.
- Dar formato al código.

Utilizando las opciones del menú Refactorizar, resuelve los siguientes pasos:

- Mover la clase Circulo al paquete figuras.
- Renombrar la clase Circulo por Circunferencia. Observar si el cambio afecta a otras clases (en este caso Test).
- Renombrar el atributo "rad" por "radio". ¿Cómo afecta al método get?.
- Convertir la variable local "color" del método imprimir en un atributo, inicializando su valor en el mismo método imprimir.
- En imprimir, en lugar de calcular y escribir el diámetro directamente en el println, extraer a una variable local "d" e imprimir dicha variable.
- Hacer que 3.1416 sea una constante llamada PI.
- Extraer el cálculo del área a un método llamado calcularArea. No recibirá parámetros y devolverá un double.
- Cambiar la firma o cabecera del método esIgual, invirtiendo el orden de los parámetros y cambiando el nombre de conDecimales por considerarDecimales. ¿Cómo afecta el cambio a la clase Test, en la que se usaba este método?.
- Ahora se propone usar "inline" para deshacer algunos cambios, es decir, hacer el código más concreto. Seleccionar la variable "d" (diámetro) y hacer que su valor se use en línea, desapareciendo por tanto la variable.
- Seleccionar la llamada al método calcularArea y hacer que su código se incorpore en la misma línea, desapareciendo la necesidad de usar el método (se puede borrar el método después).
- Seleccionar la constante PI y hacer que su valor se incorpore a las líneas en que se usa, desapareciendo por tanto la constante.
- Se propone repetir el paso anterior con el atributo "color". ¿Es posible eliminar este atributo y utilizar su valor en línea?.

Código de las dos clases:

Clase Circulo (paquete utilidades)

```

1 package utilidades;
2 public class Circulo {
3     private double rad;
4     public Circulo(double radio)
5     {
6         this.rad = radio;
7     }
8     public void imprimir()
9     {
10        String color = "rojo";
11        System.out.println("Diámetro: " + 2*rad);
12        System.out.println("Color: " + color);
13        double area = 2 * 3.1416 * rad * rad;
14        System.out.println(area);
15    }
16    public boolean esIgual (Circulo otro, boolean conDecimales)
17    {
18        double radio1 = this.rad;
19        double radio2 = otro.getRad();
20        if (conDecimales)
21        {
22            if (radio1 == radio2)
23                return true;
24            else
25                return false;
26        }
27        else
28        {
29            if (Math.abs(radio1-radio2)<1)
30                return true;
31            else
32                return false;
33        }
34    }
}

```

Clase Test (paquete figuras)

```
1 package figuras;
2 public class Test {
3     public static void main (String[] args)
4     {
5         Circulo c1 = new Circulo(5.5);
6         Circulo c2 = new Circulo(10.1);
7         Circulo c3 = new Circulo(10.9);
8
9         if (c2.esIgual(c3, false))
10        {
11            System.out.println("c2 y c3: iguales sin considerar decimales");
12        }
13
14         if (c2.esIgual(c3, true))
15        {
16            System.out.println("c2 y c3: iguales considerando decimales");
17        }
18    }
19 }
```

b.- Solución de la práctica.

A continuación se muestran algunas indicaciones sobre como resolver la actividad.

	Opción a probar	Solución propuesta
1	Crear un proyecto Eclipse llamado "refactor".	Menú principal: Archivo/Nuevo/Proyecto/Proyecto Java.
2	Crear un paquete "utilidades" y, dentro, una clase Circulo.	Pulsar el botón derecho del ratón sobre el proyecto refactor: Nuevo/paquete. Pulsar el botón derecho del ratón sobre el paquete utilidades: Nuevo/clase.
3	Crear un paquete "figuras" y, dentro, una clase Test.	Idem anterior. Con el código inicialmente ofrecido, el programa no compila.
4	Mediante las utilidades de "Código fuente": <ul style="list-style-type: none">• Generar métodos get y set para la clase Circulo.• Corregir la tabulación del código.• Dar formato al código.	Generar métodos de obtención y establecimiento. Sangrado correcto. Formatear.
5	Mover la clase Circulo al paquete figuras.	Sobre el nombre de la clase, pulsar el botón derecho del ratón: refactorizar/mover. Continuar a pesar de la advertencia.
6	Renombrar la clase Circulo por Circunferencia. Observar si el cambio afecta a otras clases (en este caso Test).	Sobre el nombre, pulsar el botón derecho del ratón: refactorizar/redenominar.
7	Renombrar el atributo "rad" por "radio". ¿Cómo afecta al método get?.	Actualiza el parámetro devuelto, pero no modifica la firma del método.
8	Convertir la variable local "color" del método imprimir en un atributo, inicializando su valor en el mismo método imprimir.	Sobre la variable local, pulsar el botón derecho del ratón: refactorizar/convertir variable local en campo.
9	En imprimir, en lugar de calcular y escribir el diámetro directamente en el println, extraer a una variable local "d" e imprimir dicha variable.	Sobre la variable area incluida en la instrucción println, pulsar el botón derecho del ratón: refactorizar/extraer variable local.
10	Hacer que 3.1416 sea una constante llamada PI.	Sobre 3.1416, pulsar el botón derecho del ratón: refactorizar/extraer constante.
11	Extraer el cálculo del área a un método llamado calcularArea. No recibirá parámetros y devolverá un double.	Sobre la instrucción que realiza el cálculo , pulsar el botón derecho del ratón: refactorizar/extraer método
12	Cambiar la firma o cabecera del método eslgual, invirtiendo el orden de los parámetros y cambiando el nombre de conDecimales por considerarDecimales. ¿Cómo afecta el cambio a la clase Test, en la que se usaba este método?.	Sobre la cabecera del método, pulsar el botón derecho del ratón: refactorizar/cambiar firma del método. En los sitios donde es usado ha cambiado el orden de los parámetros automáticamente.
13	Ahora usaremos "inline" para deshacer algunos cambios, es decir, hacer el código más concreto. Seleccionar la variable "d" (diámetro) y hacer que su valor se use en línea, desapareciendo por tanto la variable.	Sobre la variable d, pulsar el botón derecho del ratón: refactorizar/Incorporar.
14	Seleccionar la llamada al método calcularArea y hacer que su código se incorpore en la misma línea, desapareciendo la necesidad de usar el método (puedes borrar el método después).	Sobre la llamada al método, pulsar el botón derecho del ratón: refactorizar/Incorporar.
15	Seleccionar la constante PI y hacer que su valor se incorpore a las líneas en que se usa, desapareciendo por tanto la constante.	Sobre la aparición de PI en la función, pulsar el botón derecho del ratón: refactorizar/Incorporar.
16	Intenta hacer la misma operación con el atributo "color". ¿Es posible eliminar un atributo y utilizar su valor en línea?	No, porque se trata de un valor variable que puede cambiar en otras partes del código, al contrario que los casos anteriores.

Anexo V.- Práctica con GIT

GIT es un software de control de versiones que dispone de infinidad de posibilidades, en este documento vamos a hacer un pequeño repaso de aquellas que nos pueden resultar más útiles en nuestro día a día.

En la web de GIT se dispone de GIT para diversas plataformas:

<https://git-scm.com/downloads>

En este ejemplo se va a trabajar con la versión de Linux **Debian** y el **IDE Eclipse**.

Considérese el siguiente escenario:

- **Repositorio principal**, donde diferentes usuarios ponen la información en común.
- **Repositorio local-consola**, se trata de una zona de trabajo para un usuario que actuará sobre la información desde un terminal. Asociado al repositorio principal.
- **Repositorio local-eclipse**, se trata de una zona de trabajo para un usuario que irá modificando la información desde el **IDE Eclipse**. Asociado al repositorio principal.

<pre>root@debian:/home/debian/Escritorio/GitEjer# pwd /home/debian/Escritorio/GitEjer root@debian:/home/debian/Escritorio/GitEjer# ls GitEclipse GitLocal GitRepo</pre>	GitRepo - Repositorio principal.
	GitLocal - Repositorio local-consola.
	GitEclipse - Repositorio local-eclipse.

Para no tener problemas de permisos durante el desarrollo de la práctica, se asignarán permisos completos al árbol de directorios que cuelgan de `/home/debian/Escritorio/GitEjer`, para ello desde un terminal con permisos de administrador:

- cd /home/debian/Escritorio
- chmod -R 777 ./GitEjer/

Nota: en un entorno real, en lugar de dar permisos de acceso completos, habría que aplicar sólo aquellos estrictamente necesarios.

Dispondremos de GIT en nuestro equipo **Debian** mediante su instalación desde repositorio. Para ello, con permisos de administrador, ejecuta desde consola los comandos.

- apt-get update.
- apt-get install git.

a.- Puesta en marcha.

Crear el repositorio inicial.

Mediante el uso del comando git init se crea el repositorio principal vacío.

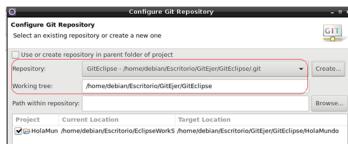
```
root@debian:/home/debian/Escritorio/GitEjer# cd GitRepo/
root@debian:/home/debian/Escritorio/GitEjer/GitRepo# git init
Initialized empty Git repository in /home/debian/Escritorio/GitEjer/GitRepo/.git/
/
root@debian:/home/debian/Escritorio/GitEjer/GitRepo# ls -la
total 12
drwxr-xr-x 3 root root 4096 mar 22 00:12 .
drwxrwxrwx 3 root root 4096 mar 22 00:07 .
drwxr-xr-x 7 root root 4096 mar 22 00:12 git
```

Crear proyecto HolaMundo en Eclipse.

Crear en Eclipse un proyecto Java **HolaMundo.java** que implemente el método main.

Crear repositorio local-Eclipse.

Pulsar el botón derecho sobre el *proyecto* => *Team/Compartir proyecto*/

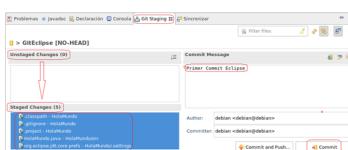


Hacer el primer commit.

Volvado de la información del área de trabajo al repositorio local-Eclipse. Opción *Team/Commit*.

Previo al commit, habrá que seleccionar los ficheros del proyecto candidatos a formar parte del repositorio (Unstaged => Staged): seleccionarlos + Botón derecho + *Add to Index*.

Finalmente, introducir un mensaje y pulsar el botón *Commit*.



Desde consola podemos ver que el commit se ha realizado de forma satisfactoria.

```
root@debian:/home/debian/Escritorio/GitEjer/GitEclipse# git log
commit 13b76102faf65fc0d006361c19f955e330219
Author: debian <debian@debian>
Date:   Thu Mar 22 00:21:41 2018 +0100
    Primer Commit Eclipse
```

Enlazar el repositorio local-Eclipse al repositorio Principal.

Haciendo un primer push nos pedirá información del repositorio Principal que todavía no tiene asociado. */Team/push branch 'master'*.

Una vez introducida la información en la ventana, habrá que ir comprobando cierta información que nos va mostrando **Eclipse (botón Preview)** y finalmente pulsar el botón *Push*.

Descargar el repositorio Principal al repositorio local-consola.

Volvado del repositorio principal al área de trabajo local de consola con *git clone*.

b.- Trabajando en el repositorio local - consola.

En este apartado se va a modificar información en el repositorio local-consola, para posteriormente hacer un volcado de la información al repositorio principal.

Los pasos a realizar son:

- Modificar información en el área de trabajo local. Se realiza con el editor linux **nano** para modificar el fichero **HolaMundo.java** ubicado en la ruta **/home/debian/Escritorio/GitEjer/GitLocal/GitRepo/HolaMundo/src**.
- Seleccionar los ficheros modificados como candidatos para ser volcados al repositorio local. Comando **add**.
- Identificar el usuario que opera en esta copia local mediante el comando **git config --global**.
- Volcado de la información desde el área de trabajo al repositorio local. Comando **commit**.

Para poder subir los cambios al repositorio principal desde consola, éste no debe tener como rama activa la destinataria (master en este caso).

La siguiente imagen muestra que la única rama existente en el repositorio principal al principio es la master. Se creará una segunda rama (Rama 1) y se marcará como activa (checkout).

Ahora ya es posible enviar los cambios del repositorio local-consola al repositorio principal mediante el comando push.

c.- Trabajando en el repositorio local - Eclipse.

En este apartado se va a modificar información en el repositorio local-eclipse. Posteriormente se hace un volcado de la información al repositorio principal.

En primer lugar habrá que ponerse al día con el repositorio Principal. */Team/Pull*, puesto que éste ha sido modificado en el apartado anterior desde el repositorio local de consola.

Equivale a:

- **Fetch**. Descarga de la versión del repositorio principal al repositorio local.
- **Merge**. Mezcla del repositorio local con el área de trabajo. Si no hay conflictos, se realiza automáticamente. En caso de haberlos, será el programador quién habrá de tomar la decisión de resolverlos. Ver siguientes apartados.

Modificar el código en el área de trabajo **Eclipse** sin hacer **commit** al repositorio local. Observa que **Eclipse** nos advierte de esta situación marcando el proyecto con el símbolo ">". El área de trabajo no está sincronizada con el repositorio local. No obstante, si hacemos un Pull, nos advierte que todo está al día; en este caso compara los repositorios Principal y Local-Eclipse.

Sincronizar el repositorio local-eclipse con el principal. Team/push branch master.

Sincronizar el repositorio local -consola desde el repositorio principal.

Finalmente, actualizamos el repositorio local-consola para que considere los últimos cambios volcados en el repositorio principal.

- Observa que **git status** no advierte de discrepancias de código puesto que el área de trabajo local está sincronizada con el repositorio local - consola. Se puede ver que los cambios no están presentes en el fichero HolaMundo.java visualizado con el comando more.
- El comando git pull actualiza el repositorio-local y el área de trabajo desde el repositorio principal.

d.- Gestión de conflictos - 1

Modificar ambos repositorios locales a partir de una versión común y combinar todos los cambios para tener una versión final en el repositorio Principal con modificaciones conjuntas.

Subir en primer lugar en Eclipse y consolidar desde consola.

La siguiente figura muestra cambios introducidos en eclipse y su commit al repositorio local-eclipse.

A continuación se muestran los cambios del proyecto en la versión local de consola y su **commit**.

Subir los cambios del repositorio-eclipse al repositorio principal (*Team/push branch master*.). Antes de lanzar el **push** siempre es mejor hacer un **pull** para ver si trabajamos sobre la última versión del repositorio principal.

En este caso, no hay conflictos porque los últimos cambios/commit se han hecho sobre la última versión que se subió al repositorio principal. Team/Push to branch master.

Ahora toca subir los cambios realizados en el repositorio local-consola al principal. Previo al **push** (subida de cambios) hacemos un **pull** para combinar las modificaciones que anteriormente hemos subido desde **Eclipse**.

El comando **pull** hace un **fetch** con lo que el repositorio local-consola y principal quedan en la misma versión y un **merge** que combina el repositorio local-consola con el área de trabajo. Ahora el nuevo fichero combinado considera los cambios locales y los procedentes del pull. Como no es posible resolver automáticamente las discrepancias, el programador será quien tenga que decidir sobre cómo quedará el código final combinado.

Finalmente habrá que hacer un **commit** y un **push** del fichero que ya combinará las actualizaciones realizadas desde ambos ámbitos locales de trabajo.

Solo faltaría hacer un **pull** en **Eclipse** para que quede también actualizado.

e.- Gestión de conflictos - 2.

Modificar ambos repositorios locales a partir de una versión común y combinar los cambios de ambos para tener una versión final en el repositorio Principal con modificaciones conjuntas.

Subir en primer lugar en consola y consolidar desde Eclipse.

En primer lugar se modificarán las zonas de trabajo locales y se actualizan sendos repositorios locales.

Subir los cambios del repositorio-consola al repositorio principal (**push**). Antes de lanzar el **push** siempre es mejor hacer un **pull** para ver si trabajamos sobre la última versión del repositorio principal.

Antes de hacer un **push** en **Eclipse**, habrá que comprobar si el repositorio principal está en la misma versión que el local (no que el área de trabajo), y si no lo está, solicitar un **pull** (**fetch + merge**) para poner en común los cambios en ambos repositorios. *Team/Pull*.

GIT no ha sido capaz de decidir como sincronizar los cambios y nos muestra una combinación de ambos (véase la figura anterior) para que sea el programador quien tome la decisión sobre como hacerlo. Una vez cambiados, se hará un **commit** al repositorio **local-Eclipse** y un **push** a continuación para subir la versión que recoge todas las modificaciones al repositorio principal.

Si ahora se solicita un **pull** desde **Eclipse**, avisa de que no hay información que actualizar. Los dos repositorios **Principal** y **local-Eclipse** están sincronizados.

El que ha quedado desincronizado es el repositorio local-consola respecto al repositorio principal.

Como no ha habido cambios en local - ningún **commit** desde la última vez que estuvo sincronizado (último **pull**). Un nuevo **pull** deberá actualizar el repo local sin necesidad de hacer sincronización de cambios manuales.

f.- Comandos Git, Status, Log.

git status. Estado de los archivos en el directorio de trabajo y en el staging area.

git log. Mostrar la historia de commits en un branch.

git show. Información sobre el último *commit*, y los cambios que produjo.

Anexo VI.- Práctica con GitHub

GibHub es un gestor de versiones **GIT** disponible en la nube. Está accesible en la url: <https://github.com/>.

a.- Alta de cuenta en GitHub.

Para crear una nueva cuenta, habrá que pulsar en el enlace "*Sign in*", que muestra la ventana de acceso para usuarios **GIT**, y que además ofrece la opción de crear una nueva cuenta mediante el enlace "*Create an account*".

A partir de este punto, un asistente irá guiando la creación de la cuenta.

b.- Crear un repositorio.

Para ilustrar las actividades que se van a llevar a cabo en este documento, se va a dar de alta el repositorio **HolaMundoGit**. A continuación, se volcará la información del repositorio **Eclipse local** con el que se ha trabajado en el anexo V.

Una vez creado, el repositorio está disponible en una url del tipo:
https://github.com/usuario/repositorio

Para este ejemplo será:
https://github.com/josemanuelmoreno1/HolaMundoGit

El repositorio está inicialmente vacío.

c.- Carga de información en el repositorio HolaMundoGit de GitHub.

En este apartado se sincronizará el repositorio recién creado con los repositorios utilizados en el anexo V (en particular con **Eclipse local**).

Pulsar el botón derecho del ratón sobre el proyecto y seleccionar la opción de menú Push: */Team/Push branch master*

En la figura superior, se puede comprobar que el repositorio remoto seleccionado es el disponible en **GitHub**.

Pedirá el **usuario/contraseña** que tenemos en la cuenta de **GitHub**.

Para finalizar, confirmar el **Push**.

Se puede observar que el proyecto **HolaMundoGit** ya está disponible en el repositorio **Github**.

d.- Descarga del repositorio HolaMundoGit de GitHub a un repositorio local de consola.

Para crear desde consola un nuevo repositorio local con la información disponible en el repositorio de **GitHub**, desde un terminal de consola con permisos de administrador:

- Crear un nuevo repositorio vacío: **git init**.
- Seleccionar el repositorio remoto con el que estará sincronizado: **git remote add**.
- Descargar el proyecto: **git pull**.

Se puede observar que tras la descarga, el proyecto ya está disponible en el nuevo repositorio local.

Anexo VII.- Práctica: JavaDoc con Eclipse.

En el punto 3 (Documentación) se describen algunas de las funcionalidades proporcionadas por la herramienta **JavaDoc**. Para poner en práctica su uso utilizaremos el **IDE Eclipse** y el proyecto Semaforo ya utilizado en otras prácticas durante el curso. Para descargar dicho proyecto, pulsa [aquí](#).

Partiendo del código del proyecto Semaforo, se van a incluir una serie de comentarios y etiquetas **JavaDoc** distribuidos por el proyecto. El código, una vez modificado, está disponible en este enlace: [Proyecto semáforo con comentarios](#).

En la siguiente figura se puede ver el fichero *ordenador.java* una vez incluidos los comentarios **JavaDoc**.

Los pasos a seguir son:

- Sobre el código fuente del proyecto, incluir los comentarios **JavaDoc** que se consideren oportunos.
 - Seleccionando el proyecto semaforo, acceder a la función "Generar Javadoc" en la opción de menú *Proyecto* de la barra de herramientas.
-
- En la ventana "Generar Javadoc" que aparece, seleccionar como "Mandato javadoc" el **ejecutable Javadoc** que viene integrado en el paquete **jdk** e indicar la ruta donde queremos que se guarde la documentación generada.

Una vez generada la documentación, estará disponible en la ruta seleccionada como destino en la ventana anterior. El resultado es una estructura web cuyo fichero de inicio es *index.html*. Si pulsas [aquí](#) te puedes descargar el *javaDoc* generado.

En la siguiente figura se muestra el aspecto que tiene la documentación generada. En particular, se presenta la clase *ordenador* del proyecto.

Anexo VIII.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Gnome.org. Licencia: GNU. Procedencia: Captura de pantalla de gnome.org.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial.

	Procedencia: Elaboración propia.		Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com

	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com

	Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com		Autoría: GIT. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com		Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com		Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com		Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com
	Autoría: GitHub. Licencia: Copyright cita. Procedencia: github.com		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.		Autoría: Oracle Corporation. Licencia: Copyright cita. Procedencia: Captura de pantalla de Netbeans.
	Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.		Autoría: eclipse.org Licencia: Copyright cita. Procedencia: Captura de pantalla de Eclipse.

Diseño orientado a objetos. Elaboración de diagramas estructurales.

Caso práctico



En la empresa siguen trabajando en diferentes aplicaciones con un nivel alto de complejidad, se desarrolla para diferentes plataformas, en entornos de ventanas, para la web, dispositivos móviles, etc. **Ada** lleva un tiempo observando a su equipo, y a pesar de que ya han hablado de las diferentes fases de desarrollo del software, y que están descubriendo nuevos entornos de programación que han facilitado su trabajo enormemente, se ha dado cuenta de que todavía hay una asignatura pendiente, sus empleados no utilizan herramientas ni crean documentos en las fases previas del desarrollo de una aplicación, a pesar de ser algo tan importante como el resto de fases del proceso de elaboración de software. Tampoco construyen modelos que ayuden a hacerse una idea de como resultará el proyecto. Estos documentos y modelos son muy útiles para que todo el mundo se ponga de acuerdo en lo que hay que hacer, y cómo van a hacerlo.

Como **Ada** muy bien conoce, un proyecto de software tendrá éxito sólo si produce un software de calidad, consistente y sobre todo que satisfaga las necesidades de los usuarios que van a utilizar el producto resultante.

Para desarrollar software de calidad duradera, hay que idear una sólida base arquitectónica que sea flexible al cambio.

Incluso para producir software de sistemas pequeños sería bueno hacer análisis y modelado ya que redonda en la calidad, pero lo que si es cierto, es que cuanto más grande y complejos son los sistemas más importante es hacer un buen modelado ya que nos ayudará a entender el comportamiento del sistema en su totalidad. Y cuando se trata de sistemas complejos el modelado nos dará una idea de los recursos necesarios (tanto humanos como materiales) para abordar el proyecto. También nos dará una visión más amplia de cómo abordar el problema para darle la mejor solución.

Ada se da cuenta de que el equipo necesita conocer procedimientos de análisis y diseño de software, así como alguna herramienta que permita generar los modelos y la documentación asociada, así que decide reunir a su equipo para empezar a tratar este tema...



[Ministerio de Educación y Formación Profesional.](#) (Dominio público)

**Materiales formativos de FP Online propiedad del Ministerio de
Educación y Formación Profesional.**

[Aviso Legal](#)

1.- Programación orientada a objetos

Caso práctico



Ya en la sala de reuniones...

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

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

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

El enfoque estructurado.

Sin embargo, cómo se hace es algo que ha ido evolucionando con el tiempo, en un principio se tomaba el problema de partida y se iba sometiendo a un proceso de división en subproblemas más pequeños reiteradas veces, hasta que se llegaba a problemas elementales que se podía resolver utilizando una función. Luego las funciones se hilaban y entrelazan hasta formar una solución global al problema de partida. Era, pues, un proceso centrado en los procedimientos, se codificaban mediante funciones que actuaban sobre estructuras de datos, por eso a este tipo de programación se le llama programación estructurada. Sigue una filosofía en la que se intenta aproximar qué hay que hacer, para así resolver un problema.

Enfoque orientado a objetos.

La orientación a objetos ha roto con esta forma de hacer las cosas. Con este nuevo paradigma el proceso se centra en simular los elementos de la realidad asociada al problema de la forma más cercana posible. La abstracción que permite representar estos elementos se denomina objeto, y tiene las siguientes características:

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

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

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

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

Para saber más

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

[Grupo de gestión de objetos.](#)

1.1.- Conceptos de orientación a objetos.

Caso práctico

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



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

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

- ✓ **Abstracción:** Permite capturar las características y comportamientos similares de un conjunto de objetos con el objetivo de darles una descripción formal. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad, o el problema que se quiere atacar.
- ✓ **Encapsulación:** Organiza los datos y métodos de una clase, evitando el acceso a datos por cualquier otro medio distinto a los definidos. El estado de los objetos sólo debería poder ser modificado desde métodos de la propia clase. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.
- ✓ **Modularidad:** Propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. En orientación a objetos es algo consustancial, ya que los objetos se pueden considerar los módulos más básicos del sistema.
- ✓ **Principio de ocultación:** La implementación de una clase sólo es conocida por los responsables de su desarrollo. Gracias a la ocultación, ésta podrá ser modificada para mejorar su algoritmo de implementación sin tener repercusión en el resto del programa. Principalmente se oculta las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas. Reduce la propagación de efectos colaterales cuando se producen cambios.
- ✓ **Polimorfismo:** Consiste en reunir bajo el mismo nombre comportamientos diferentes. La selección de uno u otro depende del objeto que lo ejecute.
- ✓ **Herencia:** Relación que se establece entre objetos en los que unos utilizan las propiedades y comportamientos de otros formando una jerarquía. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
- ✓ **Recolección de basura:** Técnica por la cual el entorno de objetos se encarga de destruir automáticamente los objetos, y por tanto desvincular su memoria asociada, que hayan quedado sin ninguna referencia a ellos.

1.2.- Ventajas de la orientación a objetos.

Caso práctico

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



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

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

Autoevaluación

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

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

No es así. La orientación a objetos tiene la ventaja de que una vez definido un módulo (objeto) puede utilizarse en cualquier aplicación en la pueda ser útil.

No es así, los principios de encapsulación y modularidad permiten la construcción de abstracciones con un estado y un comportamiento propio que suele ser sencillo de ampliar gracias a la independencia del resto de objetos del sistema.

No es cierto, el mantenimiento de estas aplicaciones suele ser más sencillo porque afecta a un solo módulo de forma que el efecto que puede producir sobre el resto de la aplicación es nulo o muy pequeño.

Así es, gracias a que cumple con las propiedades de modularidad, encapsulación, reutilización y escalabilidad.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

1.3.- Clases, atributos y métodos.

Caso práctico

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

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



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

Una clase se describe por su nombre (identifica cada clase del programa) y por un conjunto de elementos que se denominan **miembros**. Estos miembros son:

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

Por ejemplo, si consideramos un objeto *ícono* en una aplicación gráfica; tendrá como atributos el tamaño, o la imagen que muestra; y su protocolo puede constar de mensajes producidos al pulsar el botón sobre el objeto. De esta forma los mensajes son el único conducto que conectan al objeto con el mundo exterior.

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

Autoevaluación

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

- Verdadero.**
- Falso.**

Así es, el objeto tiene un estado formado por los valores concretos que toman los atributos de la clase a la que pertenece, además para modificar estos valores tenemos que hacerlo utilizando los métodos de la clase a través del paso de mensajes.

No es cierto, el objeto si es una concreción de los atributos de una clase.

Solución

1. Opción correcta
2. Incorrecto

1.4.- Visibilidad.

Caso práctico

—Pues creo que ya lo tenemos todo...

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



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

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

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

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

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

- ✓ El estado debe ser privado. Los atributos de una clase se deben modificar mediante métodos de la clase creados a tal efecto.
- ✓ Las operaciones que definen la funcionalidad de la clase deben ser públicas.
- ✓ Las operaciones que ayudan a implementar parte de la funcionalidad deben ser privadas (si no se utilizan desde clases derivadas) o protegidas (si se utilizan desde clases derivadas).

Autoevaluación

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

- Desde cualquier zona de la aplicación.

- Desde la clase y sus clases derivadas.
- Solo desde los métodos de la clase.

Falso, el estado no se define como público.

Falso, el estado no se define normalmente como protegido.

Así es, ya que, por regla general, el estado se define como privado.

Solución

1. Incorrecto
2. Incorrecto
3. Opción correcta

1.5.- Objetos. Instanciación.

Caso práctico

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

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



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

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

Un objeto se define por:

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

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

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

Citas para pensar

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

Grady Booch

Ejemplo de objetos:

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

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

2.- UML.

Caso práctico

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

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



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

Citas para pensar

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

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

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

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

¿Porqué es útil modelar?

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

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

2.1.- Familiarizándonos con algunos conceptos UML.

A continuación se describen algunos de los términos típicamente usados en **UML**. Se acompaña de un ejemplo relacionado con la creación de una melodía musical.

2.1.1.- Notación.

Una **notación** es un conjunto de **símbolos y técnicas para combinarlos**, que en el contexto **UML**, permiten crear **diagramas** normalizados. Estas representaciones posibilitan al analista o desarrollador describir el comportamiento del sistema (análisis) y los detalles de una arquitectura (diseño) de forma no ambigua.

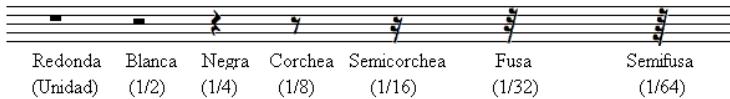
Que una notación sea detallada no significa que todos sus aspectos deban ser utilizados en todas las ocasiones. Utilizar UML debe facilitar el desarrollo/entendimiento de todos los participantes en el proyecto y no complicarlo. ¡Si se crean todos los diagramas al máximo nivel de detalle podría liar más que aclarar!

Las notaciones UML deben ser independientes de los lenguajes de programación.

Un **Diagrama** es una representación gráfica de una colección de elementos de modelado (**modelo**), a menudo dibujada como un grafo con vértices conectados por arcos (**notación**).

- **Notación musical-1.** La notación musical formal considera términos como pentagrama; notas redonda, blanca, negra, corchea, semicorchea, fusa, semifusa; clave de sol, fa, do ...

Tipos de silencios empleados en música



- **Notación musical-2.** Otra alternativa, útil para inexpertos, podría ser escribir en una/varias líneas la secuencia de notas que forman una melodía. **DO - RE - MI - FA - SOL - LA - SI.**

2.1.2.- Modelos y herramientas

Un **modelo** captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.

Volviendo al ejemplo musical, la melodía puede mostrarse desde diferentes vistas.

Vista - 1. Usando la primera notación del apartado anterior.



Vista - 2. Usando la segunda notación del apartado anterior.

DO - DO - RE - DO - FA - MI - DO - DO - RE - DO - SOL - FA - FA - LA - DO - LA - FA - MI - RE

Vista - 3. La interpretación de la melodía sería otra vista distinta. Incluso se pueden considerar diferentes vistas en función del compás, instrumento ...



En **UML** existen una serie de modelos (**diagramas**) que nos proporcionan vistas en las fases de análisis y diseño.

Cada modelo es completo desde su punto de vista del sistema, sin embargo, existen relaciones de trazabilidad entre los diferentes modelos.

Una **herramienta** es el soporte automático de una notación. En nuestro ejemplo hablaríamos de un lápiz, un cuaderno musical, un programa de ordenador, un órgano

Para el desarrollo de diagramas **UML** en este curso se usará la aplicación/herramienta **UMLet**.

2.1.3.- Métodos

Un **método** es un proceso disciplinado para generar uno/varios modelos que describen aspectos de un sistema de software en desarrollo, utilizando alguna notación bien definida.

2.2.- Tipos de diagramas UML.

Caso práctico

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



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

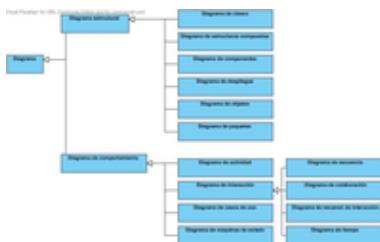
—Utilizaremos diagramas, que son unos grafos en los que los nodos definen los elementos del diagrama, y los arcos las relaciones entre ellos.

UML define un sistema como una **colección de modelos** que describen sus diferentes perspectivas. Los modelos se implementan en una serie de diagramas que son representaciones gráficas de una colección de elementos de modelado, a menudo dibujado como un grafo conexo de arcos (relaciones) y vértices (otros elementos del modelo).

Los diagramas **UML** se clasifican en:

- **Diagramas estructurales.** Representan la visión **estática** del sistema. Especifican clases y objetos y como se distribuyen físicamente en el sistema.
- **Diagramas de comportamiento.** Muestran la **conducta en tiempo** de ejecución del sistema, tanto desde el punto de vista del sistema completo como de las instancias u objetos que lo integran.

En la siguiente imagen aparecen todos los diagramas organizados según su categoría:



En total se describen trece diagramas para modelar diferentes aspectos de un sistema, sin embargo no es necesario usarlos todos, dependerá del tipo de aplicación a generar y del sistema, es decir, se debe generar un diagrama solo cuando sea necesario.

Diagramas estructurales.

- **Diagrama de clases.** Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones.
- **Diagrama de objetos.** Muestra los elementos del modelo estático en un momento concreto, habitualmente en casos especiales de un diagrama de clases o de comunicaciones, y está formado por un conjunto de objetos y sus relaciones.

- **Diagrama de componentes.** Especifica la organización lógica de la implementación de una aplicación, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos.
- **Diagrama de despliegue.** Representa la configuración del sistema en tiempo de ejecución. Aparecen los nodos de procesamiento y sus componentes. Exhibe la ejecución de la arquitectura del sistema. Incluye nodos, ambientes operativos tanto de hardware como de software, así como las interfaces que las conectan, es decir, muestra como los componentes de un sistema se distribuyen entre los ordenadores que los ejecutan. Se utiliza cuando tenemos sistemas distribuidos.
- **Diagrama de estructuras compuestas.** Muestra la estructura interna de una clase, e incluye los puntos de interacción de esta clase con otras partes del sistema.
- **Diagrama de paquetes.** Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. Suele ser útil para la gestión de sistemas de mediano o gran tamaño.

Diagramas de comportamiento.

- **Diagrama de casos de uso.** Representa las acciones a realizar en el sistema desde el punto de vista de los usuarios. En él se representan las acciones, los usuarios y las relaciones entre ellos. Sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.
- **Diagrama de estado de la máquina.** Describe el comportamiento de un sistema dirigido por eventos. En el que aparecen los estados que puede tener un objeto o interacción, así como las transiciones entre dichos estados. También se denomina diagrama de estado, diagrama de estados y transiciones o diagrama de cambio de estados.
- **Diagrama de actividades.** Muestra el orden en el que se van realizando tareas dentro de un sistema. En él aparecen los procesos de alto nivel de la organización. Incluye flujo de datos, o un modelo de la lógica compleja dentro del sistema.
- **Diagramas de interacción.**
 - **Diagrama de secuencia.** Representa la ordenación temporal en el paso de mensajes. Modela la secuencia lógica, a través del tiempo, de los mensajes entre las instancias.
 - **Diagrama de comunicación/colaboración.** Resalta la organización estructural de los objetos que se pasan mensajes. Ofrece las instancias de las clases, sus interrelaciones, y el flujo de mensajes entre ellas. Comúnmente enfoca la organización estructural de los objetos que reciben y envían mensajes.
 - **Diagrama de interacción.** Muestra un conjunto de objetos y sus relaciones junto con los mensajes que se envían entre ellos. Cada nodo de actividad dentro del diagrama puede representar otro diagrama de interacción.
 - **Diagrama de tiempos.** Muestra el cambio en un estado o una condición de una instancia o un rol a través del tiempo. Se usa normalmente para exhibir el cambio en el estado de un objeto en el tiempo, en respuesta a eventos externos.

En la imagen aparecen todos los diagramas organizados según su categoría. En total se describen trece diagramas para modelar diferentes aspectos de un sistema, sin embargo no es necesario usarlos todos, dependerá del tipo de aplicación a generar y del sistema.

Citas para pensar

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

2.3.- Herramientas para la elaboración de diagramas UML.

Caso práctico

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



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

-¡Guau, eso sí es facilitar el trabajo!

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

Podemos encontrar, entre otras, las siguientes herramientas:

- ✓ **Rational Systems Developer de IBM:** Herramienta propietaria que permite el desarrollo de proyectos software basados en la metodología UML. Desarrollada en origen por los creadores de UML ha sido recientemente absorbida por IBM. Ofrece versiones de prueba, y software libre para el desarrollo de diagramas UML.

Para saber más

Si sientes curiosidad puedes seguir este enlace a la página oficial de [Rational Systems Developer](#).

- ✓ **Visual Paradigm for UML (VP-UML):** Incluye una versión para uso no comercial que se distribuye libremente sin más que registrarse para obtener un archivo de licencia (bajo licencia LGPL).

- ➡ Incluye diferentes módulos para realizar desarrollo UML, diseñar bases de datos, realizar actividades de ingeniería inversa y diseñar con Agile.
- ➡ Compatible con UML 2.0.
- ➡ Admite la generación de informes en formatos PDF, HTML y otros.
- ➡ Es compatible con los IDE de Eclipse, Visual Studio .net, IntelliJDEA y NetBeans.
- ➡ Multiplataforma.
- ➡ Incluye instaladores para Windows y Linux.



Para saber más

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

- ✓ **ArgoUML:** se distribuye bajo licencia Eclipse. Soporta los diagramas de UML 1.4, y genera código para java y C++. Para poder ejecutarlo se necesita la plataforma java. Admite ingeniería directa e inversa.

Para saber más

Aquí tienes el enlace a [ArgoUML](#).

- **UMLet:** herramienta UML de código abierto y libre distribución. Dispone de un interfaz de usuario sencillo de utilizar

Para saber más

Si sientes curiosidad puedes seguir este enlace a la página oficial de [UMLet](#).

Autoevaluación

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

- Verdadero.
- Falso.

No es así, estas herramientas se utilizan para muchas más cosas, no solo para la elaboración de los diagramas.

Así es, además de permitir la creación y edición de los diagramas tienen funcionalidad para la documentación, la generación automática de código y operaciones de ingeniería inversa, que generan diagramas a partir de código fuente o incluso binario.

Solución

1. Incorrecto
2. Opción correcta

2.3.1.- Generación de la documentación.

Caso práctico

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

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

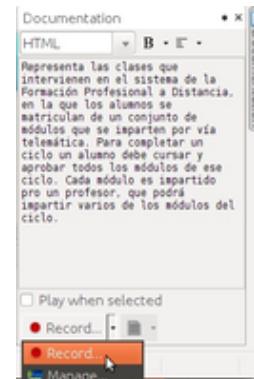


—No solo eso, además de generar documentación exhaustiva, la herramienta permite crear informes con ella, pasándola a un formato más cómodo de leer e interpretar en papel por el equipo de desarrollo.

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

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

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



Generar informes

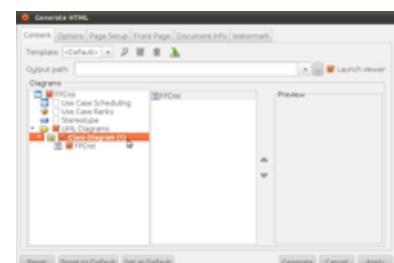
Cuando los modelos están completos podemos generar un informe en varios formatos diferentes (HTML, PDF o Word) con la documentación que hemos escrito. Para generar un informe hacemos:

Desde VP-UML accedemos a Tools >> Reports >> Report writer y seleccionamos el tipo de informe que queremos.

Desde el SDE para NetBeans seleccionamos Modelin >> Reports >> Report writer.

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

- ✓ Qué diagramas queremos que intervengan y donde se almacenará el informe.
- ✓ La pestaña opciones (Options) permite configurar los elementos que se añadirán al informe, como tablas de contenidos, títulos, etc.
- ✓ Las propiedades de la página.
- ✓ Si se va a añadir una marca de agua.



El resultado es un archivo (.html, .pdf o .doc) en el directorio de salida que hayamos indicado con la documentación de los diagramas seleccionados.

2.3.2.- UMLet.

Para los **diagramas UML** que se van a realizar a lo largo del curso se va a usar la herramienta **UMLet**. Algunas de sus características son:

- Es un software gratuito.
- Es multiplataforma.
- Incluye un módulo para integrarse con Eclipse.
- Dispone de la versión web **UMLetino** <http://www.umletino.com/umletino.html>, que no precisa instalación.

La instalación de **UMLet** como herramienta de escritorio es muy sencilla:

- Descargar **UMLet standalone** del sitio oficial de descargas <https://www.umlet.com/changes.htm>, actualmente se encuentra en versión 14.3. El fichero descargado es *umlet-standalone-14.3.0.zip*.
- Descomprimir el fichero, el paquete proporciona un conjunto de ficheros bajo el directorio *uml* con ejecutables para diferentes plataformas. En particular, usaremos el archivo comprimido **Java umlet.jar** para lanzarlo desde nuestro entorno de trabajo.

 custom_elements	08/11/2018 0:45	Carpeta de archivos
 img	08/11/2018 0:45	Carpeta de archivos
 lib	08/11/2018 0:45	Carpeta de archivos
 palettes	08/11/2018 0:45	Carpeta de archivos
 LICENCE	03/04/2016 15:14	Documento de tex
 umlet	16/11/2017 10:39	Archivo DESKTOP
 Umlet	28/05/2018 19:20	Aplicación
 umlet	05/08/2018 11:26	Executable Jar File
 umlet.sh	06/04/2018 9:59	Archivo SH

- Podemos copiar la carpeta descomprimida en alguna ruta del sistema de archivos.

Utilizando **UMLet** dibujaremos diferentes diagramas a lo largo de este tema y del siguiente.

Nota: Si tienes problemas para ejecutar **UMLet** no dudes en utilizar **UMLetino**.

En el Anexo II se va hacer una breve explicación de las diversas posibilidades que ofrece **UMLet**.

2.4.- Ingeniería inversa.

Caso práctico

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



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

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

La ingeniería inversa puede ser de varios tipos:

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

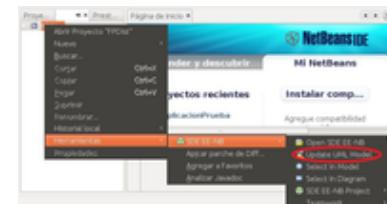
Ejercicio Propuesto

Parte de un proyecto en Java que tengas ya hecho y obtén el el diagrama de clases correspondiente.

[Mostrar retroalimentación](#)

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

1. Abrimos el SDE.
2. Seleccionamos el proyecto en el panel de proyectos y abrimos la herramienta SDE-CE NB.
3. Abrir un proyecto nuevo VP-UML en el proyecto de NetBeans.
4. Desde el nodo del proyecto seleccionamos en el menú contextual la opción "**Update UML Model**" lo que iniciará el proceso de ingeniería inversa.



Desde VP-UML

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

Para saber más

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

[Ingeniería Inversa.](#) (0.53 MB)

3.- Diagrama de clases

Caso práctico

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

—Empecemos por los diagramas estructurales, entre ellos el más importante es el diagrama de clases, fíjate, representa la estructura estática del sistema y las relaciones entre las clases.



El **diagrama de clases** puede considerarse el más importante de todos los existentes en **UML**, encuadrado dentro de los diagramas estructurales, representa los elementos estáticos del sistema, sus atributos y comportamientos, y como se relacionan entre ellos. Contiene las clases del dominio del problema, y a partir de éste se obtendrán las clases que formarán después el programa informático que dará solución al problema.

En un **diagrama de clases** podemos encontrar los siguientes elementos:

- **Clases**: agrupan conjuntos de objetos con características comunes, que llamaremos atributos, y su comportamiento, que serán métodos. Los atributos y métodos tendrán una visibilidad que determinará quién puede acceder al atributo o método. Por ejemplo, una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará.
- **Relaciones**: en el diagrama se representan relaciones reales entre los elementos del sistema a los que hacen referencia las clases. Pueden ser de asociación, agregación, composición y generalización. Por ejemplo, si tenemos las clases persona y coche, se puede establecer la relación conduce entre ambas. O una clase alumno puede tener una relación de generalización respecto a la clase persona.
- **Notas**: se representan como un cuadro donde podemos escribir comentarios que ayuden al entendimiento del diagrama.
- **Elementos de agrupación**: Se utilizan cuando hay que modelar un sistema grande, entonces las clases y sus relaciones se agrupan en paquetes, que a su vez se relacionan entre sí.

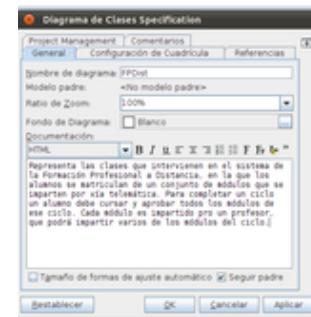
Ejercicio Propuesto

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

[Mostrar retroalimentación](#)

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

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



3.1.- Creación de clases.

Nombre Clase
-lista de atributos
+lista de métodos()

Una clase se representa en el diagrama como un rectángulo dividido en tres filas: arriba aparece el nombre de la clase, a continuación los atributos con su visibilidad y después los métodos con su visibilidad.

Citas para pensar

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

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

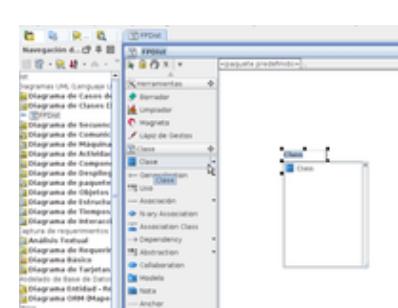
[Booch G., 1994]

Ejercicio Propuesto

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

[Mostrar retroalimentación](#)

Cuando generamos un diagrama nuevo aparece un panel con los elementos que podemos añadir al diagrama. Si hacemos clic sobre el ícono que genera una clase nueva y a continuación sobre el lienzo aparecerá un cuadro para definir el nombre de la nueva clase. Posteriormente, cuando la clase esté creada si hacemos clic con el botón secundario sobre la clase y seleccionamos "Abrir Especificación" podremos añadir atributos y métodos.



Autoevaluación

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

- Verdadero.
- Falso.

Lo habitual es que una clase tenga estos tres elementos, pero obligatoriamente sólo hay que ponerle un nombre (si no se le pone se le deja el nombre por defecto). Los atributos y métodos dependen del objeto que se abstrae.

Así es, si algún elemento no tiene sentido para la clase no hay que ponerlo por fuerza.

Solución

1. Incorrecto
2. Opción correcta

3.2.- Atributos.

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

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

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

- ✓ **Público (+)**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
- ✓ **Privado(-)**: Sólo se pueden acceder desde operaciones de la clase.
- ✓ **Protegido(#)**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
- ✓ **Paquete(~)**: se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación tiene esta característica como es el caso de Java.

Ejercicio Propuesto

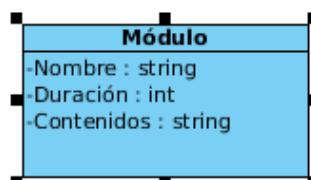
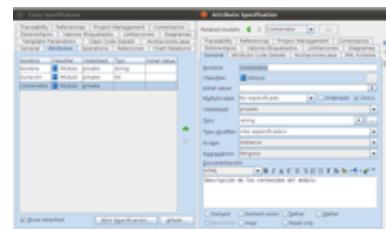
Crear una clase de nombre "Módulo" y que tenga tres atributos:

- ✓ Nombre, de tipo string.
- ✓ Duración de tipo Int.
- ✓ Contenidos de tipo string.

Mostrar retroalimentación

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

Así queda la representación de la clase, los guiones al lado del atributo significan visibilidad privada.



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

Autoevaluación

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

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

Falso, el símbolo más “+” significaría que es público.

Falso, el símbolo almohadilla “#” significaría que es protegido.

Falso, el símbolo “~” significaría que pertenece a un paquete de clases java.

Efectivamente, cuando se establece un atributo o método como privado va acompañado del símbolo menos “-”.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

3.3.- Métodos.

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

Existe dos métodos particulares:

- El **constructor** de la clase, que tiene como característica que no devuelve ningún valor. El constructor tiene el mismo nombre de la clase y se usa para ejecutar las acciones necesarias cuando se instancia un nuevo objeto.
- El **destructor** de la clase. Cuando no se vaya a utilizar más el objeto, se podrá utilizar un método destructor que libere los recursos del sistema que tenía asignados. La destrucción del objetos es tratada de formas diferentes en función del lenguaje de programación que se esté utilizando.

Ejercicio Propuesto

Añadir a la clase creada anteriormente los métodos:

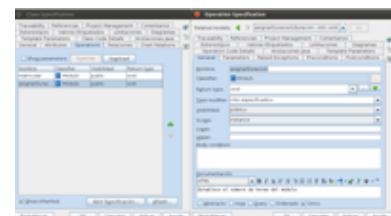
- ✓ matricular(alumno : Alumno) : void
- ✓ asignarDuración(duracion : int) : void

[Mostrar retroalimentación](#)

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

```
+nombre(<lista_parámetros>)      :
    tipo_devuelto
```

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



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

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



Autoevaluación

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

- El constructor.
- Todos los métodos devuelven algo, aunque sea void.
- ~<nombre_clase>

Efectivamente, como sabes, es un método especial que coincide en nombre con la clase y no devuelve ningún tipo de dato. Se usa para realizar las operaciones de inicialización pertinentes cuando se instancia un objeto.

Falso, existe un método especial que no devuelve nada, ni siquiera void.

Falso, el símbolo ~ se utiliza para designar el destructor en el lenguaje de programación C++, y solo se usará en UML cuando C++ sea el lenguaje final.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto

3.4.- Relaciones entre clases.

Caso práctico

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



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

Una **relación** es una conexión entre dos clases que incluimos en el diagrama.

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

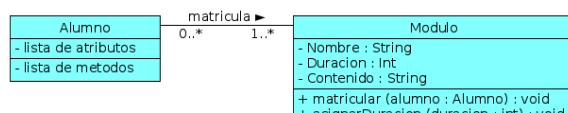
Las relaciones se caracterizan por su **cardinalidad**, que representa cuantos objetos de una clase se pueden involucrar en la relación

Ejercicio Propuesto

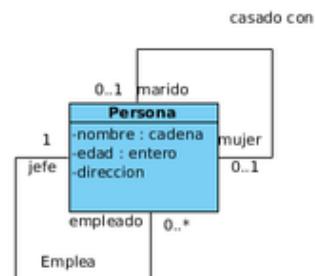
Crea una clase nueva llamada Alumno y establece una relación de asociación con el nombre “matrícula” entre ésta y la clase Módulo.

[Mostrar retroalimentación](#)

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



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



Otros tipos de relaciones que se verán más adelante son:

- ✓ De herencia.
- ✓ De composición.
- ✓ De agregación.

Autoevaluación

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

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

Falso, existen relaciones, como las de herencia que se infieren o, a veces, se crean en el diagrama para facilitar la labor al analista.

Cierto, es normal que se creen otro tipo de relaciones como las de herencia que no suelen aparecer específicamente en la especificación de requisitos.

Solución

1. Incorrecto
2. Opción correcta

3.4.1.- Cardinalidad o multiplicidad de la relación

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

Significado de las cardinalidades.

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

Por ejemplo, si tengo la siguiente relación:



quiere decir que la clase Alumno se relaciona con la clase Módulo debido a que los alumnos se matriculan en diferentes módulos y en un módulo puede estar matriculado alumnos. La cardinalidad indicada quiere decir que todo alumno está matriculado en al menos un módulo y puede estar matriculado en varios y que en un módulo puede haber varios alumnos matriculados y puede ser que en un módulo no haya nadie matriculado.

O esta otra:



quiere decir que la clase Profesor relaciona con la clase Módulo debido a que los profesores imparten diferentes módulos y un módulo es impartido por un profesor. La cardinalidad indicada quiere decir que todo profesor imparte al menos un módulo pudiendo impartir varios y, todo módulo es impartido por un profesor y sólo uno.

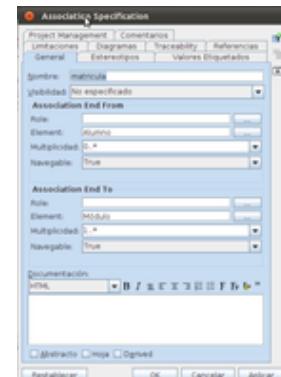
Ejercicio Propuesto

Establece la cardinalidad de la relación que has creado en el punto anterior para indicar que un alumno debe estar matriculado en al menos un módulo, o varios y

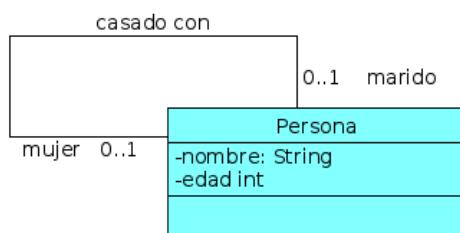
que para cada módulo se puede tener ningún alumno, uno o varios.

[Mostrar retroalimentación](#)

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



Y este ejemplo:



Indica que una persona se relaciona con otra persona por la relación "casado con". La cardinalidad indica que una mujer puede estar soltera o casada con una persona y los maridos igual.

3.4.2.- Relación de herencia (Generalización)

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

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

Tipos:

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

Representación:

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

Ejercicio Propuesto

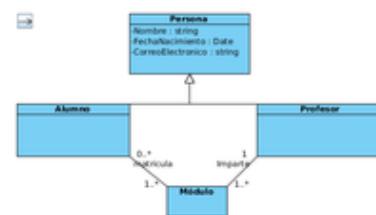
En nuestro diagrama tenemos Alumnos y Profesores. Aún no hemos hablado de su definición y estructura, pero en nuestro sistema tanto un alumno como un profesor tienen unas características comunes como el nombre, la fecha de nacimiento o el correo electrónico por el hecho de ser personas:



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

[Mostrar retroalimentación](#)

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



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

Autoevaluación

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

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

No nos interesa que sea público porque se va a usar en las clases derivadas para calcular el sueldo.

No puede ser privada porque entonces nadie podría usar el método fuera de la clase.

Correcto. Así puede usarse libremente en las clases derivadas, pero como no es necesario que se use fuera queda restringida para otras clases.

Incorrecto. Como solo se usa en las clases derivadas no hace falta que se vea en otras clases del paquete Java.

Solución

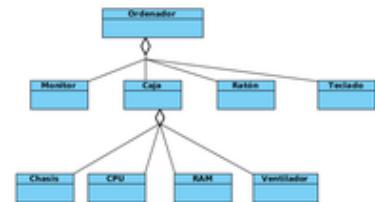
1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

3.4.3.- Agregación y composición.

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

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

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



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

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



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

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

3.4.4.- Atributos de enlace.

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

Ejercicio Propuesto

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

[Mostrar retroalimentación](#)

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



Autoevaluación

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

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

Así es, puesto que este método puede ser útil desde cualquier ámbito fuera de la clase.

No puede ser privada porque entonces no se podría usar el método fuera de la clase.

No tiene sentido puesto que no existen clases derivadas.

Podría usarse, pero es mejor ponerlo público para que pueda usarse libremente si la aplicación crece lo suficiente como para tener más de un paquete.

Solución

1. Opción correcta
2. Incorrecto
3. Incorrecto
4. Incorrecto

3.4.5.- Restricciones

En ocasiones la relación entre dos clases está condicionada al cumplimiento de algún requisito, o un parámetro de una clase tiene un valor constante ...

Cuando se precisa reflejar una condición que aparece en el enunciado y no disponemos de una notación particular para que quede reflejada en el diagrama de clases, es posible mostrarla mediante una **restricción**.

Las restricciones se incluyen mediante una descripción textual encerrada entre llaves.

Ejemplo-1. Supongamos un ejercicio donde el socio de un club de fútbol desea acceder al palco del estadio. Si esta posibilidad sólo está disponible para antiguos jugadores del equipo, junto a la línea que relaciona las clases socio y palco, se puede incluir la restricción {sólo para ex-jugadores}.

Ejemplo-2. Supongamos que la clase producto de un establecimiento tiene un IVA fijo del 21%, sea cual sea el tipo de producto que pone a la venta. Dado que este valor es fijo, podría considerarse una constante. En el diagrama de clases podría indicarse con una restricción del tipo {IVA constante 21%}.

3.5.- Pautas para crear diagramas de clase.

Caso práctico

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

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



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

A la hora de crear diagramas de clases, la clave está en hacer una buena elección de las clases que sugiere el problema.

Para identificar las clases candidatas a formar parte del diagrama, es recomendable subrayar cada nombre o sintagma nominal que aparece en el enunciado.

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

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

La clase se considera si cumple todos (o casi todos) los criterios.

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

3.5.1.- Obtención de atributos y operaciones.

Atributos

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



Operaciones

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

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

Se obtienen analizando verbos en el enunciado del problema.

Relaciones

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

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

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

Caso práctico

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

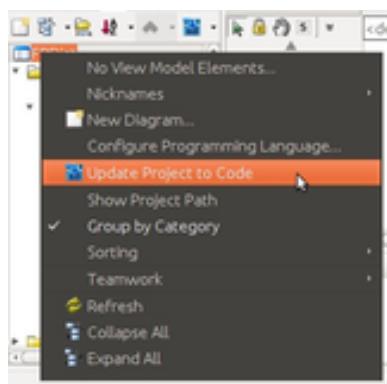


—Pues sí, pero aún no lo sabes todo, el diagrama de clases aún te va a dar más facilidades...

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

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

Utilizando el SDE integrado de VP-UML en NetBeans:



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

Se puede hacer de dos formas:

- ✓ **Sincronizar con el código:** El código fuente eliminado no se recuperará. Solo se actualizará el código existente.
- ✓ **Forzar sincronizado a código:** Se actualizará todo el código que pueda partir del modelo, incluido el de código eliminado del proyecto NetBeans.

Para generar todas las clases y paquetes de un proyecto VP-UML en NetBeans abrimos el proyecto CE-NB y desplegamos el menú contextual y seleccionamos **Update Project to Code**.

También existe la posibilidad de hacerlo directamente desde una clase en particular.

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

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

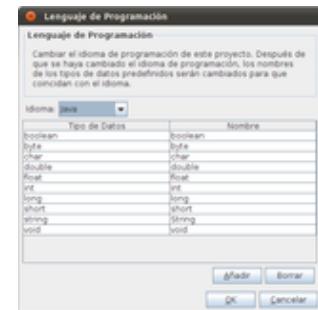
Desde VP-UML

Para generar el código java de un diagrama de clases, utilizamos el menú **Herramientas >> Generación instantánea >> Java...**. Se muestra una ventana en la que podemos configurar el idioma, las clases a generar, y otras características básicas relacionadas con la nomenclatura de atributos y métodos. También permite seleccionar la forma en que se va a implementar la asociación de composición, en nuestro caso hemos elegido la opción por defecto que es a través de un vector.

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

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

Podemos definir el lenguaje de programación final desde el menú **Herramientas >> Configurar lenguaje de programación**. Si seleccionamos Java automáticamente cambiará los nombres de los tipos de datos al lenguaje escogido.



Anexo I.- Descarga e instalación de Visual Paradigm.

Descarga e instalación de Visual Paradigm

Obtenemos los archivos desde:

[Página de Visual Paradigm](#)

Ofrece dos versiones:

- ✓ **Visual Paradigm for UML (VP-UML)**, versión de prueba de 10 días, ampliable a 30 días mediante registro.
- ✓ **Versión Community-Edition**, para uso no comercial (gratuito).

En cualquier caso necesitamos un código de activación que conseguiremos registrándonos. Se envía al correo electrónico que se indique en el registro.

La versión **Community-Edition** incluye algunas de las funcionalidades de la versión completa, entre las que no se encuentra la generación de código ni la ingeniería inversa, que se verán al final de la unidad por lo que se recomienda empezar por la versión completa de prueba por 30 días, para los que se necesita un código de tipo, conseguiremos el código de activación, que es un archivo de tipo **.zvpl**, en este caso llamado **vpsuite.zvpl**.

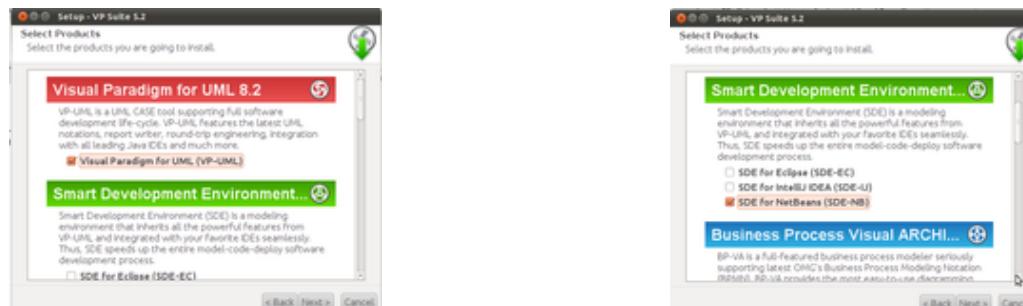
Para la siguiente unidad también usaremos VP-UML, de modo que si fuera necesario tendríamos que conseguir un nuevo código de activación, esta vez de tipo **Community-Edition**.

Proceso de instalación

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

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

Durante la instalación tendremos que indicar qué módulos queremos instalar, seleccionaremos Visual Paradigm for UML y el **SDE** (Smart Development Environment o Entorno de Desarrollo Inteligente), de NetBeans que es el que vamos a usar.



A continuación tendremos que indicar que vamos a utilizar la versión **Enterprise** de ambas herramientas y en que directorio está NetBeans:

Es importante destacar que la instalación debe hacerse sobre una instalación limpia de NetBeans, es decir, que solo podremos instalarlo en el directorio que indicamos una vez.

A continuación se pide un archivo con la **licencia** de la herramienta. Al iniciar la descarga nos pedirá que nos registremos, tras hacerlo podremos solicitar este archivo. Lo insertamos ahora, como hemos instalado dos herramientas nos pedirá dos archivos, pero podemos usar la opción de archivo de licencia combinado, de modo que nos sirva para los dos casos. Si nos lo pide, tendremos que volver a añadirlo después al iniciar Visual Paradigm, con una copia nueva del archivo de clave.

Por último indicamos dónde queremos que ponga los archivos con los proyectos y finalizamos la instalación indicando que no queremos que abra ninguna aplicación.

Iniciar Visual Paradigm

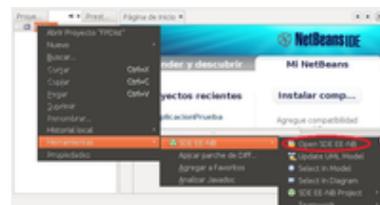
Una vez realizada la instalación tendremos una entrada en el menú **Aplicaciones** llamada **Otras**, si trabajamos con Linux o bien una entrada de menú en el botón Inicio para Visual Paradigm, si es que trabajamos en Windows. En cualquiera de los casos para abrir la herramienta buscamos la opción **Visual Paradigm for UML**, que se abrevia como **VP-UML**. Al hacer clic se abrirá el programa, y nos preguntará cual es el directorio por defecto para guardar los proyectos, podemos dejar la opción por defecto o seleccionar nuestro propio directorio.

Iniciar VP-UML desde NetBeans

Al hacer la instalación hemos indicado, marcado, que se instale también el SDE para NetBeans, por lo que también tenemos la opción de iniciar la herramienta para usarla integrada con NetBeans. Para abrirlo buscamos dentro del menú de Visual Paradigm la opción **SDE for NetBeans**.

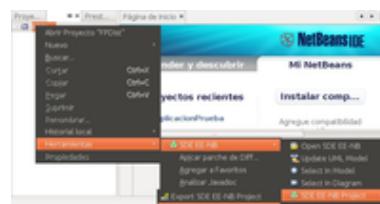
Esto abre la aplicación NetBeans, a la que se ha incorporado una pequeña diferencia, y es que podemos añadir a un proyecto en desarrollo existente un proyecto VP-UML. ¿Cómo lo hacemos?

Estando en la ventana de **Proyectos**, si hacemos clic con el botón secundario sobre un proyecto vemos una serie de opciones, como compilar o construir, ahora, además, abrir el SDE desde **Open SDE EE-NB**, que abre el SDE. La primera vez nos pedirá que importemos un archivo de clave, que podremos obtener con el botón Request Key desde la página oficial. Para ello necesitaremos el correo de registro que hemos utilizado al hacer la instalación.

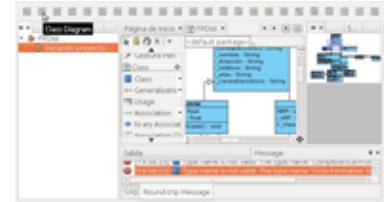


Los proyectos de Visual Paradigm se podrán almacenar en el directorio por defecto, que se denomina **vpproject** y cuelga del directorio principal del proyecto NetBeans, o en otra ubicación. Nosotros nos quedaremos con la opción por defecto.

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



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

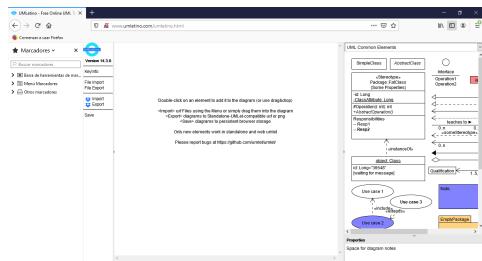


Anexo II.- Introducción a UMLet

En este anexo se va hacer una breve explicación de las diversas posibilidades que ofrece UMLet. Para ello vamos a utilizar su versión en línea UMLetino disponible en la dirección <http://www.umletino.com/umletino.html>.

a.- Pantalla principal.

La primera pantalla a la que accedemos es:



Se ven claramente diferenciadas cuatro zonas:

- zona izquierda con un menú que dispone de algunas opciones generales
- zona central donde se mostrará el diagrama que estemos diseñando
- zona superior derecha que muestra distintos símbolos de UML
- zona inferior derecha que será la zona de edición de las propiedades de cada elemento

b.- Opciones.

En la zona izquierda se visualizan las siguientes opciones:

- **Keyinfo:** muestra los atajos de teclado
- **File Import:** importa un proyecto UML en el formato propio de UMLet (.uxf)
- **File Export:** exporta un proyecto UML de tipo UMLet (.uxf)
- **Import:** importa un proyecto UML en el formato propio de UMLet (.uxf) desde Dropbox
- **Export:** exporta un proyecto UML de tipo UMLet (.uxf) a Dropbox
- **Save:** salva el proyecto en el sistema de almacenamiento persistente del propio navegador

b.1.- Atajos de teclado.

Si seleccionas la opción **Keyinfo** aparecerá la siguiente pantalla:

KEYBOARD SHORTCUTS	
DIAGRAM	
DELETE	delete the currently selected elements
BACKSPACE	
Ctrl+A	select all elements
Ctrl+Shift+A	deselect all elements
Ctrl+D	
Ctrl+C	copy selected elements
Ctrl+X	cut selected elements
Ctrl+V	paste cut or copied elements
Ctrl+S	save current diagram in browser storage
SHIFT	hold to disable sticking of elements
Cursor ↑	moves selected element(s) up
Cursor ↓	moves selected element(s) down
Cursor ←	moves selected element(s) left
Cursor →	moves selected element(s) right
BROWSER (only if browser supports them)	
F11	switch to fullscreen
Ctrl+PLUS	zoom diagram in
Ctrl-MINUS	zoom diagram out
Ctrl+0	reset diagram zoom
PROPERTIES PANEL	
Ctrl+SPACE	Show all autocompletion suggestions

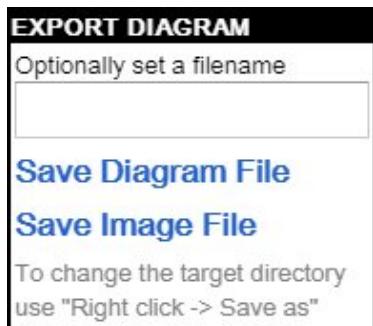
b.2.- Importación de diagramas.

Pulsando **File Import** se abre una ventana de selección de un fichero de extensión **.uxf** para cargarlo.

Si el fichero se encuentra en Dropbox se debe utilizar la opción **Import**

b.3.- Exportación de diagramas.

Pulsando **File Export** se abre una ventana para guardar un fichero de extensión **.uxf** que almacena la información del diagrama.

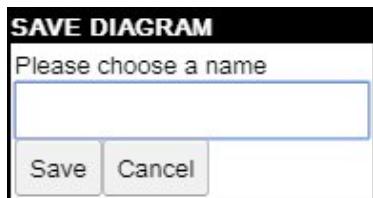


Podemos llenar el nombre de fichero a guardar y elegir la opción **Save Diagram File**. La opción **Save Image File** únicamente guarda una imagen del diagrama.

Si el fichero se encuentra en Dropbox se debe utilizar la opción **Export**.

b.4.- Salvar diagramas.

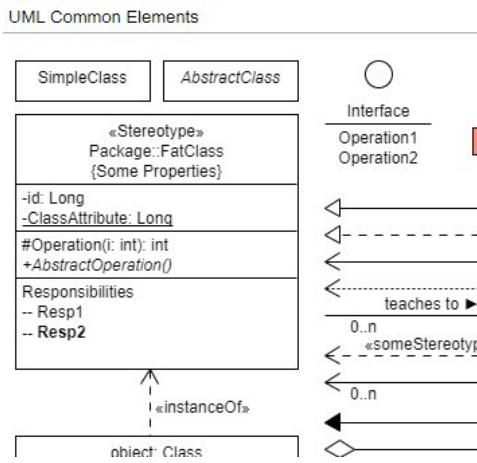
En la versión web disponemos de la posibilidad de almacenar el diagrama en el propio navegador. Para ello seleccionaremos la opción **Save** apareciendo la siguiente pantalla:



Escribimos el nombre que demos al diagrama y veremos su aparición debajo de la opción **Save**.

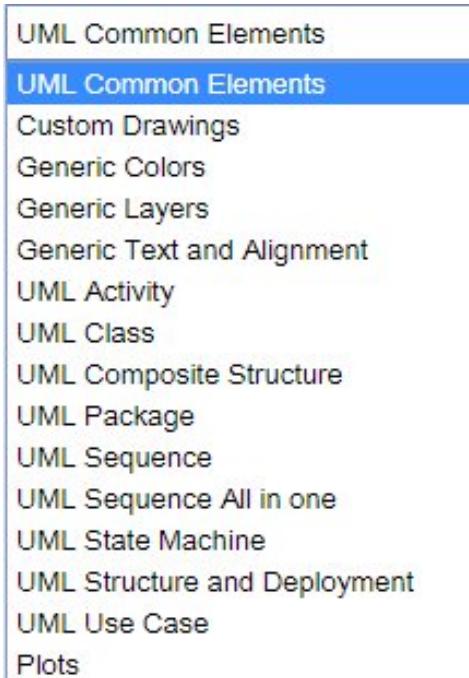
c.- Elementos del diagrama.

En la zona superior derecha se pueden ver distintos símbolos propios de UML.



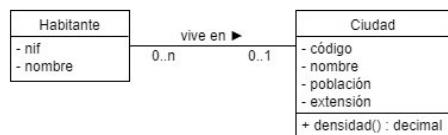
Para utilizar cualquier elemento haremos doble clic sobre éste o lo arrastraremos a la zona central.

En la parte superior aparece un menú desplegable para seleccionar distintos tipos de diagrama:

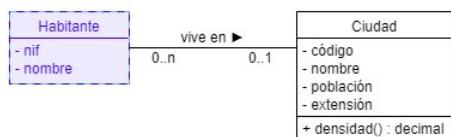


d.- El diagrama.

En la zona central se muestra el diagrama que se está diseñando.



Si pulsamos cualquiera de los elementos incluidos en el diagrama éste cambiará de color,



y aparecerá su información en la zona inferior derecha.

Properties

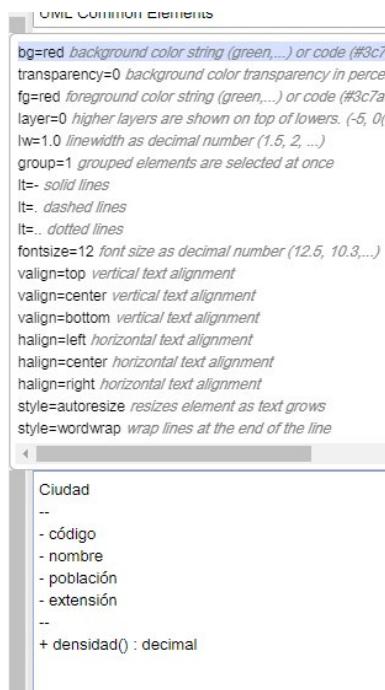
Habitante
--
- nif
- nombre

e.- Edición de las propiedades.

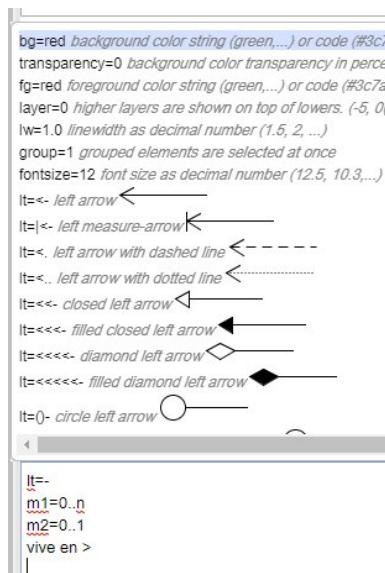
Al seleccionar cualquier elemento del diagrama sus propiedades aparecen en la ventana inferior derecha.

Para saber las opciones disponibles dependiendo del elemento que estemos editando debemos teclear **Ctrl+Barra de espacio**.

En el caso de una clase veremos las siguientes opciones:



En el caso de una relación:



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

Descripción del problema

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

La aplicación tiene que gestionar la información de los alumnos y profesores. De todas estas personas nos interesa: nombre, dirección y teléfono. Además, tanto los alumnos y profesores se identifican con un alias en el sistema y se comunican a través de correo electrónico.

Cuando un alumno finaliza el ciclo se emite un certificado de competencias a su nombre donde aparece la descripción de las competencias que forman el ciclo y la nota media obtenida. Para los profesores, además, se debe conocer su número de registro personal (NRP).

Interesa también gestionar la información de los módulos en los que se matriculan los alumnos en diferentes ciclos formativos a distancia. De cada módulo formativo nos interesa: nombre, número de horas (en el curso) y contenido.

Interesa saber de los alumnos que están matriculados en cada módulo. En cada módulo habrá, al menos, un alumno matriculado. Y también queremos saber en qué módulos está matriculado un alumno. Puede ser que no esté matriculado en ningún módulo.

También interesa saber cuales son de los módulos que imparte un profesor que pondrán los contenidos del módulo a disposición de los alumnos. Un profesor impartirá, al menos, un módulo y queremos tener información del profesor que imparte un módulo, el cual sólo puede ser impartido por un único profesor.

La aplicación tiene que tener controlada la información de todos los ciclos formativos que se imparten. Nos interesa, de cada uno de ellos: su nombre, descripción (del módulo) y horas totales.

De cada ciclo formativo interesa saber cuales son sus competencias profesionales. De cada competencia nos interesa una sola cosa: su descripción. Sabemos que un ciclo formativo está formado por una o varias competencias profesionales. Si un ciclo desaparece, desaparecen dichas competencias profesionales.

Interesa tener registrado los módulos que acompañan a cada competencia profesional. Cada competencia profesional tiene, al menos, un módulo. Un módulo pertenece a una única competencia profesional y si desaparece dicha competencia profesional, desaparece dicho módulo.

Para superar un módulo hay que hacer una tarea y un examen que se calificarán de uno a diez, y sacar en ambos casos una puntuación superior a cinco. Por ello, queremos tener información, para cada módulo, de la tarea que conlleva este módulo. Cada módulo conlleva una única tarea. Una tarea pertenece a un único módulo y si desaparece el módulo, desaparece la tarea. De cada tarea nos interesa un único dato que es: su descripción.

También, queremos tener registrado todos los exámenes, que se hacen para cada módulo. Cada examen pertenece a un único módulo y, para cada módulo, hay un único examen. Si el módulo desaparece, desaparece el examen.

Y, hay una batería de preguntas que se usan para hacer los diferentes exámenes. Estas preguntas puede ser que no sean usadas en ningón examen o que sean usadas en varias. Y un

examen siempre está formado por 30 preguntas. Si el examen desaparece, eso no implica la desaparición de ninguna pregunta. De cada pregunta nos interesa: enunciado, posibles respuestas y el número de la respuesta (qué es válida).

a.- Extracción de los sustantivos de la descripción del problema.

Primero subrayamos los sustantivos de la descripción del problema (sin repeticiones) y los pasamos a una tabla:

La aplicación tiene que gestionar la información de los alumnos y profesores. De todas estas personas nos interesa: nombre, dirección y teléfono. Además, tanto los alumnos y profesores se identifican con un alias en el sistema y se comunican a través de correo electrónico.

Cuando un alumno finaliza el ciclo se emite un certificado de competencias a su nombre donde aparece la descripción de las competencias que forman el ciclo y la nota media obtenida. Para los profesores, además, se debe conocer su número de registro personal (NRP).

Interesa también gestionar la información de los módulos en los que se matriculan los alumnos en diferentes ciclos formativos a distancia. De cada módulo formativo nos interesa: nombre, número de horas (en el curso) y contenido.

Interesa saber de los alumnos que están matriculados en cada módulo. En cada módulo habrá, al menos, un alumno matriculado. Y también queremos saber en qué módulos está matriculado un alumno. Puede ser que no esté matriculado en ningún módulo.

También interesa saber cuales son de los módulos que imparte un profesor que pondrán los contenidos del módulo a disposición de los alumnos. Un profesor impartirá, al menos, un módulo y queremos tener información del profesor que imparte un módulo, el cual sólo puede ser impartido por un único profesor.

La aplicación tiene que tener controlado la información de todos los ciclos formativos que se imparten. Nos interesa, de cada uno de ellos: su nombre, descripción (del módulo) y horas totales.

De cada ciclo formativo interesa saber cuales son sus competencias profesionales. De cada competencia nos interesa una sola cosa: su descripción. Sabemos que un ciclo formativo está formado por una o varias competencias profesionales. Si un ciclo desaparece, desaparecen dichas competencias profesionales.

Interesa tener registrado los módulos que acompañan a cada competencia profesional. Cada competencia profesional tiene, al menos, un módulo. Un módulo pertenece a una única competencia profesional y si desaparece dicha competencia profesional, desaparece dicho módulo.

Para superar un módulo hay que hacer una tarea y un examen que se calificarán de uno a diez, y sacar en ambos casos una puntuación superior a cinco. Por ello, queremos tener información, para cada módulo, de la tarea que conlleva este módulo. Cada módulo conlleva una única tarea. Una tarea pertenece a un único módulo y si desaparece el módulo, desaparece la tarea. De cada tarea nos interesa un único dato que es: su descripción.

También, queremos tener registrado todos los exámenes, que se hacen para cada módulo. Cada examen pertenece a un único módulo y, para cada módulo, hay un único examen. Si el módulo desaparece, desaparece el examen.

Y, hay una batería de preguntas que se usan para hacer los diferentes exámenes. Estas preguntas puede ser que no sean usadas en ningún examen o que sean usadas en varias. Y un examen siempre está formado por 30 preguntas. Si el examen desaparece, eso no implica la desaparición de ninguna pregunta. De cada pregunta nos interesa: enunciado, posibles respuestas y el número de la respuesta (qué es válida).

Tabla de sustantivos

Clase/objeto potencial	Categoría
Alumno	Entidad externa o rol
Ciclo Formativo a Distancia	Unidad organizacional
Modulo Formativo	Unidad organizacional
Año	Atributo
Profesor	Entidad externa o rol
Contenidos	Atributo
Tarea	Cosa
Examen	Cosa
Uno	Atributo
Diez	Atributo
Pregunta	Cosa
Enunciado	Atributo
Respuesta	Atributo
Competencia Profesional	Unidad organizacional
Descripción	Atributo
Horas	Atributo
Nombre	Atributo
Nota media	Atributo
Alias	Atributo
Nombre	Atributo
Dirección	Atributo
Teléfono	Atributo
Persona	Rol o entidad externa
Contenidos	Atributo
Número de registro personal	Atributo

b.- Selección de sustantivos como objetos/clases del sistema.

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

Cómo se indicó en el punto 3.5., estos son los criterios que hay que seguir para determinar si los sustantivos, que aparecen en el enunciado, son clases:

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

La clase se considera si cumple todos (o casi todos) los criterios.

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

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

Obtención de los atributos de los objetos.

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

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

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

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

d.- Obtención de los métodos.

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

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

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

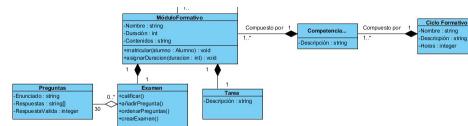
e.- Obtener relaciones.

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

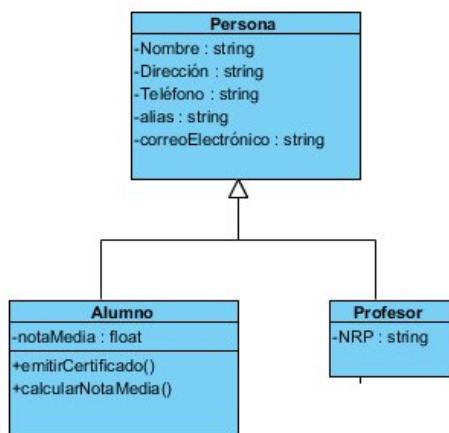
Comenzaremos por las clases que hacen referencia a la estructura de los Ciclos, cada Ciclo se compone de una o más competencias profesionales, que no tienen la capacidad de existir por si mismas, es decir, la competencia no tiene sentido sin su ciclo, por lo que vamos a crear una relación entre ambas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



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



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

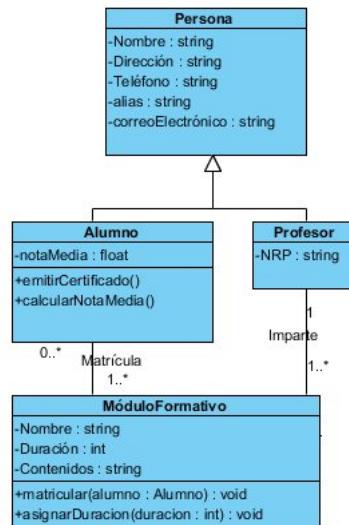


Por último queda relacionar a alumnos y profesores con los módulos formativos. Un alumno se matricula de un conjunto de módulos formativos, y un profesor puede impartir uno o varios módulos formativos.

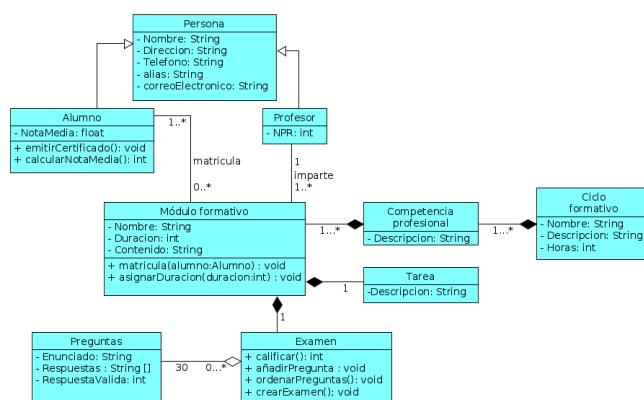
Más concretamente, de cara a la cardinalidad, un alumno puede estar matriculado en uno o varios módulos, mientras que un módulo puede tener, uno o varios alumnos matriculados. Por su

parte un profesor puede impartir uno o varios módulos, aunque un módulo es impartido por un profesor.

Éste análisis da como resultado lo siguiente:



Este sería el diagrama de clases completo:



Añadir Getters, Setters y constructores

Hay que evitar introducir información cuyo aporte funcional no sea muy relevante. Por ejemplo, en los programas aparecerán métodos constructores, getters, setters ..., todos ellos importantes en el código durante la programación, pero en ocasiones poco relevantes desde una perspectiva de definición de funciones de la clase.

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

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

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

f.- Documentación adicional.

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

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

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

Ejercicio Propuesto

A raíz de ese diagrama de clases se puede generar el siguiente código (usando Java como lenguaje de programación):

[Mostrar retroalimentación](#)

```
class Persona

{
    String nombre;
    String dirección;
    String teléfono;
    String alias;
    String correoElectrónico;
}

class Alumno extends Persona

{
    float notaMedia;
    MóduloFormativo infModFormaMatric[];
}

class Profesor extends Persona

{
    int NPR;
    MóduloFormativo infModForImparteClases[];
}

class MóduloFormativo
```

```
String nombre;

int duración;

String contenido;

Alumno infAlumMatriculados[];

Profesor profesorImparte;

Examen examen;

Tarea tarea;

}

class CompetenciaProfesional

{

String descripción;

MóduloFormativo infModuFormat[];

}

class Tarea

{

String descripción;

}

class CicloFormativo

{

String nombre;

String descripción;

int horas;

CompetenciaProfesional infCompeProfesionales[];

}

class Pregunta

{

String enunciado;

String respuestas[];

int respuestaValida;

Examen examenesDondeSeUsa[];
```

```
}

class Examen

{

Pregunta preguntas[] = new Pregunta[30];

}
```

Anexo IV.- Generación del diagrama de clases de otro problema dado.

Descripción del problema

La empresa Truan dedicada a recambios de coches necesita una aplicación para gestionar sus ventas. La empresa está compuesta de una serie de sedes repartidas en diferentes provincias (Id de la sede y provincia). Por su parte la empresa informará del número total de empleados que tiene.

Cada sede lleva a cabo su actividad con autónomos que tendrán datos como el nombre, CIF, teléfono y dirección; y serán del tipo proveedores (que además informarán de su código) o talleres (que informarán de su especialización chapa o mecánica).

Cada sede compra las piezas a distintos proveedores para después venderlas a los talleres.

Cada pieza de recambio tiene un código que la identifica y un único proveedor que puede proporcionarla. También tiene un precio de compra al proveedor y un precio de venta a los talleres. Las operaciones que se quieren hacer sobre la pieza son modificar los dos precios.

Cada sede quiere poder dar de alta nuevos proveedores. Además, las sedes quieren mantener la información de los talleres con los que trabaja.

Por motivos de aranceles, si la sede es Canarias no se podrá trabajar con el proveedor con código 666.

a.- Extracción de los sustantivos de la descripción del problema.

La **empresa Truan** dedicada a recambios de coches necesita una aplicación para gestionar sus ventas. La empresa está compuesta de una serie de **sedes** repartidas en diferentes **provincias** (**Id de la sede y provincia**). Por su parte la empresa informará del número total de **empleados** que tiene.

Cada sede lleva a cabo su actividad con **autónomos** que tendrán datos como el **nombre**, **CIF**, **teléfono** y **dirección**; y serán del tipo **proveedores** (que además informarán de su **código**) o **talleres** (que informarán de su **especialización** chapa o mecánica).

Cada sede compra las **piezas** a distintos proveedores para después venderlas a los **talleres**.

Cada pieza de recambio tiene un **código** que la identifica y un único proveedor que puede proporcionarla. También tiene un **precio de compra** al proveedor y un **precio de venta** a los talleres. Las operaciones que se quieren hacer sobre la pieza son modificar los dos precios.

Cada sede quiere poder dar de alta nuevos proveedores. Además, las sedes quieren mantener la información de los talleres con los que trabaja.

Por motivos de aranceles, si la sede es Canarias no se podrá trabajar con el proveedor con código 666.

b.- Selección de sustantivos como objetos/clases del sistema.

Ahora aplicamos los criterios de selección de clases. En este apartado es necesario destacar que aunque algunos de los sustantivos que tenemos en el enunciado podrían llegar a convertirse en clases y objetos, el proceso de creación de diagramas no es inmediato, sino que está sujeto a revisiones, cambios y adaptaciones hasta tener un resultado final completo.

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

Clase/objeto potencial	Atributos
Empresa Truan.	Empleados.
Autónomo.	Nombre, CIF, Teléfono, Dirección.
Proveedor.	Código.
Taller.	Tipo.
Sede.	Id_Sede, Nombre.
Pieza.	CódigoPieza, PrecioCompra, PrecioVenta.



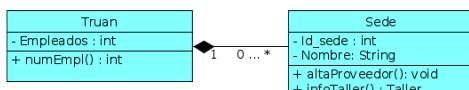
d.- Obtención de los métodos.

Clase/objeto potencial	Atributos
Empresa Truan.	+ numEmpl(): int
Autónomo.	
Proveedor.	+ codigoProv(): int
Taller.	
Sede.	+ altaProveedor(): void + infoTaller(): Taller
Pieza.	+ ponerPrecioCompra(): void + ponerPrecioVenta(): void

e.- Obtener relaciones.

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

Del enunciado se puede suponer que la empresa Truan es una composición de sedes y que no tiene sentido la existencia de sedes en caso de desaparecer la empresa. En la relación de composición la cardinalidad en el lado de la clase más general siempre es 1. En este caso particular, se considera que la empresa puede tener varias sedes o todavía no tener ninguna adherida.

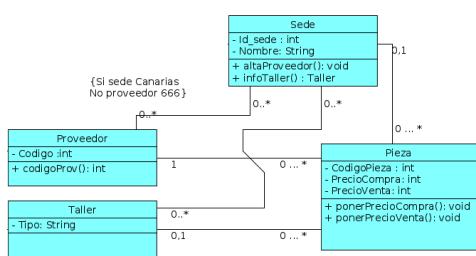


Por otro lado, tanto los proveedores como los talleres actúan como autónomos, compartiendo ciertas características comunes, por lo que resultará útil definir una relación de generalización-herencia.



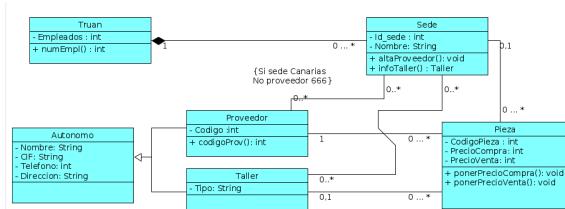
Cada pieza habrá sido distribuida por un proveedor y podrá haber sido adquirida por una sede y entregada a un taller.

Una sede que acaba de ser dada de alta, no dispondrá de proveedores ni talleres con los que trabajar inicialmente, ni dispondrá de piezas para suministrar. No obstante, con el tiempo, la sede podrá tener una relación con varios objetos de cada una de estas tres clases.



Para terminar, el enunciado advierte que el proveedor con código 666 no puede operar en Canarias; este aspecto quedará reflejado en el diagrama mediante una restricción, indicando tal circunstancia entre llaves.

Este sería el diagrama de clases completo:



f.- Documentación adicional.

El diagrama de clases, deberá ir acompañado de una descripción en texto referente a cada clase, sus atributos y métodos.

Clase/objeto potencial	Descripción
Empresa Truan.	Las empresas se construyen a partir de sedes e informan del número de empleados que tienen.
Autónomo.	Generalización para agrupar las características comunes de proveedores y talleres. Informa del nombre, CIF, teléfono y dirección.
Proveedor.	Es un tipo de autónomo. Suministra piezas a las sedes. Un proveedor puede trabajar para varias sedes, con excepción del proveedor con código 666, que no podrá suministrar piezas a la sede de Canarias. Informa de su código de proveedor.
Taller.	Es un tipo de autónomo. Receptor de las piezas disponibles en las sedes. Un taller puede ser destinatario de piezas cuyo origen sea sedes distintas. Informa del tipo de actividad de taller.
Sede.	La agregación de sedes construyen la empresa. Las sedes reciben piezas de los proveedores y las distribuyen a los talleres. Informa de su nombre e id.
Pieza.	Cada pieza habrá sido desarrollada por un proveedor y puede haber sido entregada a un taller. Las sedes son las intermedias entre proveedores y talleres en la distribución de piezas. Informa de su identificador y de los precios de compra y venta.

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

Ejercicio Propuesto

A raíz del diagrama resultante de este ejercicio, indica las clases del proyecto y los datos que tendrían dichas clases (usando Java como lenguaje de programación):

Mostrar retroalimentación

```
class Truan

{
    int empleados;

    int numEmpl();

}

class Sede

{
    int Id_sede;

    String nombre;

    Proveedor inforProveedores[];

    Taller inforTalleres[];

    Pieza inforPiezas[];

    void altaProveedor();

    Taller infoTaller();

}

Class Autonomo

{
    String nombre;

    String dirección;

    String CIF;
```

```
        int telefono;

    }

    class Proveedor extends Autonomo

    {

        int código;

        Sede inforSedes[];

        Pieza inforPiezas[];

        int códigoProv();

    }

    class Taller extends Autonomo

    {

        String tipo;

        Sede inforSedes[];

        Pieza inforPiezas[];

    }

    class Pieza

    {

        int códigoPieza;

        int precioCompra;

        int precioVenta;

        Proveedor proveedor;

        Taller taller;

        Sede sede;

        void ponerPrecioCompra();

        void ponerPrecioVenta();

    }

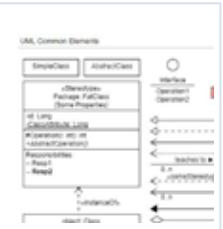
}
```

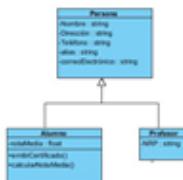
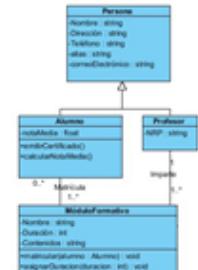
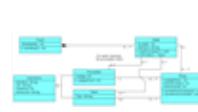
Anexo V.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: Visual Paradigm International. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm for UML.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.		Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.

A diagram showing two classes: 'Categoría' and 'Categoría'. 'Categoría' has attributes 'Nombre' (String) and 'Descripción' (String). It has a relationship 'pertenece a' with multiplicity '0..1' to 'Categoría'.	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.	A screenshot of the Visual Paradigm interface showing the 'Diagrama de Clases' (Class Diagram) tool.	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
A screenshot of the Visual Paradigm interface showing the main menu and project list.	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.	A screenshot of the Visual Paradigm interface showing the 'Diagrama de Integración' (Integration Diagram) tool.	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
A screenshot of the Visual Paradigm interface showing the 'Documentación' (Documentation) tool.	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.	A screenshot of the Visual Paradigm interface showing the 'Diagrama de Integración' (Integration Diagram) tool.	Autoría: Visual Paradigm. Licencia: Copyright cita. Procedencia: Captura de pantalla de Visual Paradigm.
A screenshot of the NetBeans IDE interface showing the code editor and toolbars.	Autoría: NetBeans. Licencia: Copyright cita. Procedencia: Captura de pantalla de NetBeans.	A photo of a person sitting at a desk, working on a computer.	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
A diagram showing a class 'Persona' with attributes 'correo_electronico' (String), 'edad' (Int), and 'nombre' (String). It has relationships 'pertenece a' with multiplicity '0..1' to 'Categoría' and 'pertenece a' with multiplicity '0..1' to 'Categoría'.	Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm.	A screenshot of the Visual Paradigm interface showing the 'Diagrama de Clases' (Class Diagram) tool.	Autoría: Visual Paradigm International Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm for UML
A diagram showing a class 'Persona' with attributes 'correo_electronico' (String), 'edad' (Int), and 'nombre' (String). It has relationships 'pertenece a' with multiplicity '0..1' to 'Categoría' and 'pertenece a' with multiplicity '0..1' to 'Categoría'.	Autoría: María José Navascués González. Licencia: Uso Educativo no comercial. Procedencia: Elaboración Propia.	A photo of a piano keyboard.	Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
A photo of a musical score page.	Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.	A photo of a piano keyboard.	Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
A photo of a woman.	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.	A photo of a man.	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso Educativo no comercial. Procedencia: Elaboración propia.

	<p>Procedencia: Elaboración propia.</p>		<p>Procedencia: Elaboración propia.</p>
	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm.</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm.</p>
	<p>Autoría: NetBeans. Licencia: Copyright cita Procedencia: Captura de pantalla de NetBeans.</p>		<p>Autoría: NetBeans. Licencia: Copyright cita Procedencia: Captura de pantalla de NetBeans.</p>
	<p>Autoría: NetBeans. Licencia: Copyright cita Procedencia: Captura de pantalla de NetBeans.</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>		<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>

			Paradigm
<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
 <pre> classDiagram class Alumno { Nombre string Apellido string Nota string Promedio string } class Profesor { Nombre string Apellido string Nota string Promedio string } class Materia { Nombre string Créditos int Nota string Promedio string } Alumno < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	 <pre> classDiagram class Persona { Nombre string Direccion string Telefono string Email string valor string correoElectronico string } class Alumno { matricula string nombreCertificado string matriculaAlumno() } class Profesor { NIFP string } Persona < -- Alumno Persona < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular Materia "*" *-- "1..n" Alumno : MatricularAlumno Alumno >--> Materia : MatricularAlumno </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
 <pre> classDiagram class Alumno { Nombre string Apellido string Nota string Promedio string } class Profesor { Nombre string Apellido string Nota string Promedio string } class Materia { Nombre string Créditos int Nota string Promedio string } Alumno < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	 <pre> classDiagram class Persona { Nombre string Direccion string Telefono string Email string valor string correoElectronico string } class Alumno { matricula string nombreCertificado string matriculaAlumno() } class Profesor { NIFP string } Persona < -- Alumno Persona < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular Materia "*" *-- "1..n" Alumno : MatricularAlumno Alumno >--> Materia : MatricularAlumno </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
 <pre> classDiagram class Alumno { Nombre string Apellido string Nota string Promedio string } class Profesor { Nombre string Apellido string Nota string Promedio string } class Materia { Nombre string Créditos int Nota string Promedio string } Alumno < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	 <pre> classDiagram class Persona { Nombre string Direccion string Telefono string Email string valor string correoElectronico string } class Alumno { matricula string nombreCertificado string matriculaAlumno() } class Profesor { NIFP string } Persona < -- Alumno Persona < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular Materia "*" *-- "1..n" Alumno : MatricularAlumno Alumno >--> Materia : MatricularAlumno </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>
 <pre> classDiagram class Alumno { Nombre string Apellido string Nota string Promedio string } class Profesor { Nombre string Apellido string Nota string Promedio string } class Materia { Nombre string Créditos int Nota string Promedio string } Alumno < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>	 <pre> classDiagram class Persona { Nombre string Direccion string Telefono string Email string valor string correoElectronico string } class Alumno { matricula string nombreCertificado string matriculaAlumno() } class Profesor { NIFP string } Persona < -- Alumno Persona < -- Profesor Alumno "0..1" *-- "1..n" Materia : Matricular Materia "*" *-- "1..n" Alumno : MatricularAlumno Alumno >--> Materia : MatricularAlumno </pre>	<p>Autoría: Visual Paradigm Licencia: Copyright cita Procedencia: Captura de pantalla de Visual Paradigm</p>

Diseño orientado a objetos. Elaboración de diagramas de comportamiento.

Caso práctico

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



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

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



[Ministerio de Educación y Formación Profesional.](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Introducción.

Caso práctico

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



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

- ✓ Diagramas de casos de uso.
- ✓ Diagramas de actividad.
- ✓ Diagramas de estados.
- ✓ Diagramas de interacción.
 - ◆ Diagramas de secuencia.
 - ◆ Diagramas de comunicación/collaboración.
 - ◆ Diagramas de interacción.
 - ◆ Diagramas de tiempo.

Para saber más

En el siguiente enlace tienes una descripción y algunos ejemplos de todos los diagramas UML, puedes usarlo como complemento a lo que vamos a ver en la unidad. No obstante te animo a que busques en la web información y ejemplos diferentes que te ayuden a seguir los contenidos.

[Introducción a UML.](#)

2.- Diagramas de casos de uso.

Caso práctico

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

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



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

Lo que Ada necesita son los diagramas de casos de uso.

Al construir software es esencial saber cuáles son los **requerimientos** del sistema que se desea crear, y se precisa alguna herramienta que ayude a especificarlos de una manera clara, sistemática y que los clientes puedan entender fácilmente.

Pero, ¿no bastaría con hacer una lista de requerimientos y describirlos exhaustivamente?. No, una descripción textual puede inducir a errores de interpretación y suele dejar cabos sueltos. La solución puede ser los **diagramas de casos de uso**.

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

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

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

Reflexiona

Los diagramas de casos de uso se crean en las primera etapa de desarrollo del software, y se enmarcan en el proceso de análisis, para definir de forma detallada la funcionalidad que se

espera cumpla el software, y que, además, se pueda comunicar fácilmente al usuario, pero, ¿termina aquí su función?

[Mostrar retroalimentación](#)

En absoluto, de los diagramas de casos de uso se desprenden otros (normalmente se realiza antes que el diagrama de clases) que describen tanto la estructura del sistema como su comportamiento, lo que influye directamente en la implementación del sistema y en su arquitectura final. Por otra parte al describir específicamente qué se espera del software también se usa en la fase de prueba, para verificar que el sistema cumple con los requisitos funcionales, creándose muchos de los casos de prueba (pruebas de caja negra) directamente a partir de los casos de uso.

2.1.- Elementos del diagrama de casos de uso

Los elementos de un diagrama de casos de uso son los siguientes:

- Actores.
- Casos de uso.
- Relaciones

2.1.1.- Actores.

Caso práctico

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



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

Los **actores** representan los tipos de **usuario que interactúan con el sistema** (un ser humano, un PC, una empresa ...) . Es importante entender la diferencia entre actores y los usuarios, por ejemplo, un usuario puede interpretar diferentes roles según la operación que esté ejecutando, cada uno de estos roles representará un actor diferente. Por otro lado, cada actor puede ser interpretado por diferentes usuarios.

Por ejemplo, el dueño de una panadería podrá aparecer en un diagrama de casos de uso con los roles de administrador y de cocinero, a su vez, puede tener otro usuario contratado, de forma que el actor cocinero podrá ser "interpretado" tanto por el dueño como por el empleado.

Tipos de actores:

- **Primarios:** interaccionan con el sistema para explotar su funcionalidad. Trabajan directa y frecuentemente con el software.
- **Secundarios:** soporte del sistema para que los primarios puedan trabajar. Son precisos para alcanzar algún objetivo.
- **Iniciadores:** es posible que haya casos de uso que no sean iniciados por ningún usuario, en ese caso se podrá considerar un actor "tiempo" o "sistema" que asuma el arranque del caso.

Los actores se representan mediante la siguiente figura:



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

Autoevaluación

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

- Verdadero.
- Falso.

Efectivamente, un actor no tiene porqué ser una persona física, es cualquier entidad que interaccione con el sistema.

No es así, un actor puede ser cualquier entidad externa que interactúe con el sistema, sea persona, empresa o software.

Solución

1. Opción correcta
2. Incorrecto

2.1.2.- Casos de uso.

Caso práctico

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



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

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

El conjunto de casos de uso forma el "**comportamiento requerido**" de un sistema. El objetivo principal de elaborar un diagrama de casos de uso no es crear el diagrama en sí, sino la descripción que de cada caso se debe realizar, ya que esto es lo que ayuda al equipo de desarrollo a crear el sistema a posteriori. Junto al diagrama, por cada caso de uso se crea una tabla con una descripción textual, en la que se deben incluir, al menos, los siguientes datos (a los que se denomina **contrato**).

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



User Case	Actor	Date
Nombre		28-agosto-2011 13:56:56
Brief Description		
Pre-conditions		
Post-conditions		
Flow of Events	Actor Input	System Response

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

Autoevaluación

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

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

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

Falso, no es una acción atómica que pueda llevarse a cabo junto con otras para realizar la acción del caso de uso.

Incorrecto, no es una acción atómica que pueda llevarse a cabo junto con otras para realizar la acción del caso de uso en un caso particular del mismo.

No parece que sea una condición que deba cumplirse para poder ejecutar un caso de uso.

Correcto, es una condición que se cumple una vez que el caso de uso se ha ejecutado correctamente.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

2.1.3.- Relaciones.

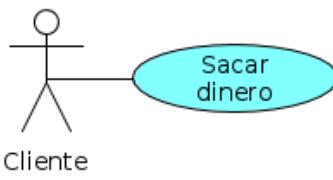
Caso práctico

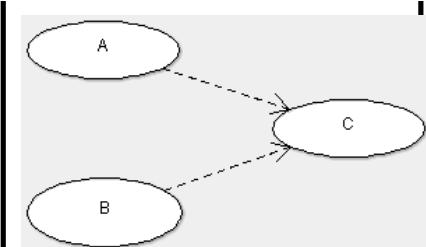
-De acuerdo, y ahora ¿Cómo asociamos a los actores con los casos de uso que pueden realizar?



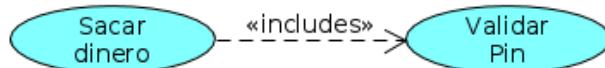
Los diagramas de casos de uso son grafos no conexos en los que los nodos son **actores** y **casos de uso**, y las aristas son las **relaciones** que existen entre ellos. Las relaciones representan qué actores realizan las tareas descritas en los casos de uso, en concreto qué actores inician un caso de uso. Pero además existen otros tipos de relaciones que se utilizan para especificar relaciones más complejas, como uso o herencia entre casos de uso o actores.

Existen diferentes tipos de relaciones entre elementos:

Relación	Descripción/ Ejemplo
Asociación	
	<p>Representa la relación entre el actor y un caso de uso en el que participa.</p> <p>Ejemplo: Relación entre el caso de uso <i>sacar dinero</i> y el <i>cliente</i> de un banco.</p> 
Inclusión (include - use)	
 Include	<p>Se trata de una relación entre casos de uso. La ejecución de un caso de uso implica necesariamente la ejecución del segundo.</p>
	<p>Esta relación es muy útil cuando se desea especificar algún comportamiento común en dos o más casos de uso, aunque es frecuente cometer el error de utilizar esta técnica para hacer subdivisión de funciones, por lo que se debe tener mucho cuidado cuando se utilice.</p>

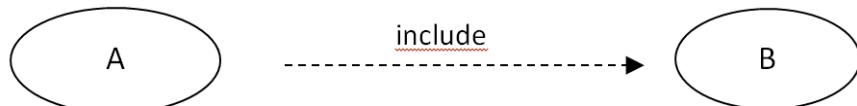
**Ejemplo 1:**

Al ejecutar el caso de uso *sacar dinero*, **obligatoriamente** se ejecuta el caso de uso *validar pin* de la tarjeta de crédito.

**Ejemplo 2:**

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

En el siguiente gráfico se representa que A usa B, es decir, que **A siempre ejecuta B**.

**Extensión (extend)**

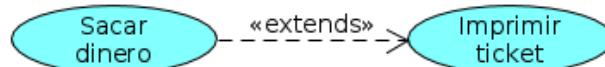
Se trata de una relación entre casos de uso. La ejecución de un caso de uso **puede** provocar la ejecución del segundo

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

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

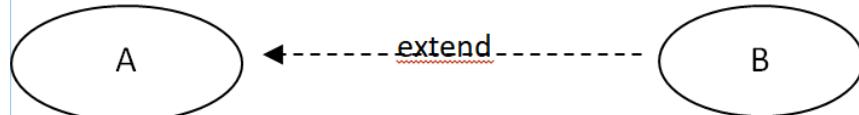
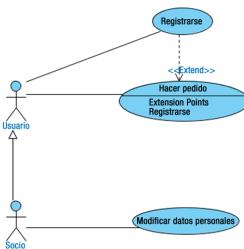
Ejemplo 1:

Imprimir ticket es consecuencia del caso de uso *sacar dinero*, pero su ejecución es opcional a que sea requerida por el cliente.



Ejemplo 2:

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



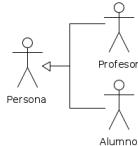
A opcionalmente ejecuta B.

Generalización

Se utiliza para representar relaciones de herencia entre casos de uso o actores. No se contemplan generalizaciones combinadas entre actores y casos de uso.

Se utiliza cuando se tiene uno o más casos de uso que son especificaciones de un caso más general.

Por ejemplo, entre actores: tanto *profesor* como *alumno* son casos particulares del actor *persona*.



Ejemplos, entre casos de uso:

Un ejemplo de generalización de casos de uso sería la compra de artículos en un comercio, pudiendo considerarse la compra de alimentos o de bebidas. Ambos tipos de compras tendrán las características heredadas del caso de uso compra general, más las particulares definidas para cada caso.

Autoevaluación

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

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

- Asociación.
- Generalización.
- extends.
- Include.



Incorrecto, no se establecen relaciones de asociación entre dos casos de uso, solo entre actores y casos de uso.

Falso, Consultar stock no se puede considerar un caso particular de Servir pedido ni lo amplía.

No es adecuado utilizar extends porque la comprobación del stock debe hacerse necesariamente, no en algún caso particular.

Correcto. Así, la consulta de existencias debe realizarse necesariamente y, además, tiene entidad suficiente como para ser un caso de uso por sí mismo, que puede usarse en otros casos.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

2.2.- Elaboración de casos de uso.

Caso práctico

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

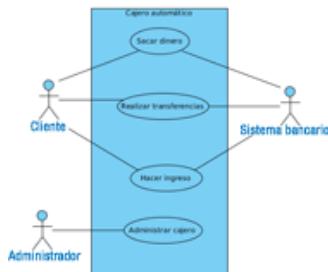


En los **diagramas de casos** se hace una **abstracción de la realidad** en la que representamos **qué cosas** pueden hacerse en nuestro sistema y **quién** las va a hacer.

Necesitamos diagramas cuya información permita al equipo de desarrollo la toma de decisiones adecuadas en la fase de análisis y diseño (cumpliendo los requisitos), así como que sean útiles en la fase de implementación en un lenguaje orientado a objetos.

Partiremos de una descripción lo más detallada posible del problema a resolver y trataremos de detectar aspectos como:

- Usuarios que interactúan con el sistema, para obtener los actores.
- Tareas que realizan estos actores para determinar los casos de uso más genéricos.
- Refinar el diagrama analizando los casos de uso más generales para detectar casos relacionados por inclusión, extensión y generalización.



2.3.- Escenarios.

Caso práctico

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



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

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

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

Un posible escenario podría ser:

Realizar pedido de unos zapatos y unas botas.

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

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

2.4.- Ejercicio resuelto 1 ("ZAPATERÍA TACÓN DE ORO") (Elaboración de un diagrama de casos de uso).

Ejercicio Propuesto

Descripción del problema: "El tacón de oro".

La zapatería Tacón de oro ha decidido crear un espacio web para ampliar su línea de negocio, así sus usuarios podrán adquirir los artículos: zapatos, bolsos y complementos que se venden en la tienda.

Los usuarios del sistema navegarán por la web para ver los artículos, zapatos, bolsos y complementos que se venden en la tienda. De los artículos nos interesa su nombre, descripción, material, color, precio y stock. De los zapatos nos interesa su número y el tipo. De los bolsos nos interesa su tipo (bandolera, mochila, fiesta). De los complementos (cinturones y guantes) su talla.

Los artículos se organizan por campañas para cada temporada (primavera/verano y otoño/invierno) de cada año.

Los artículos son de fabricación propia, pero, opcionalmente, pueden venderse artículos de otras firmas. De las firmas nos interesa saber su nombre, CIF y domicilio fiscal. La venta de artículos de firma se realiza a través de proveedores, de forma que un proveedor puede llevar varios artículos de diferentes firmas, y una firma puede ser suministrada por más de un proveedor. Los artículos pertenecen a una firma solamente. De los proveedores debemos conocer su nombre, CIF, y domicilio fiscal.

Los usuarios pueden registrarse en el sitio web para hacerse socios. Cuando un usuario se hace socio debe proporcionar los siguientes datos: nombre completo, correo electrónico y dirección.

Los socios pueden hacer pedidos de los artículos. Los usuarios pueden consultar todos los productos que tienen a su disposición, pero para realizar compras han de registrarse como socios.

Para comprar productos, se generan pedidos. Un pedido está formado por un conjunto de detalles de pedido que son parejas formadas por artículo y la cantidad. De los pedidos interesa saber la fecha en la que se realizó y cuánto debe pagar el socio en total. El pago se hace a través de tarjeta bancaria, cuando se va a pagar una entidad bancaria comprueba la validez de la tarjeta. De la tarjeta interesa conocer el número.

Las campañas son gestionadas por el administrativo de la tienda que se encargará de dar de baja la campaña anterior y dar de alta la nueva siempre que no haya ningún pedido pendiente de cumplimentar.

Existe un empleado de almacén que revisa los pedidos a diario y los cumplimenta. Esto consiste en recopilar los artículos que aparecen en el pedido y empaquetarlos. Cuando el paquete está listo se pasa al almacén a la espera de ser repartido. Del reparto se encarga una empresa de transportes que tiene varias rutas preestablecidas. Según el destino del paquete (la dirección del socio) se asigna a una u otra ruta. De la empresa de transportes se debe conocer su nombre, CIF y domicilio fiscal. Las rutas tienen un área de influencia que determina los destinos, y unos días de reparto asignados. Se debe conocer la fecha en la que se reparte el pedido. Si se produce alguna incidencia durante el reparto de algún pedido se almacena la fecha en la que se ha producido y una descripción.

Los socios pueden visualizar sus pedidos y una vez comprobados, puede cancelarlos (siempre y cuando no hayan sido cumplimentados por el empleado de almacén) o confirmar la

compra. Las compras deberán ser abonadas a través de una entidad bancaria. Así mismo los socios pueden modificar sus datos personales.

Se pide:

- Diagrama de casos de uso. Identifica los actores y casos de uso, incluye relaciones de asociación, identifica generalizaciones (de actores y de casos de uso). Si consideras alguna relación tipo include o extend, justifica su uso.
- Del diagrama que hayas obtenido en el apartado anterior, agrupa todos los casos de uso que hayas considerado en el proceso de gestión de pedidos en uno sólo. Para este apartado, se entiende que el pedido ya ha sido dado de alta y que el proceso normal de un pedido es que termine siendo comprado. Desarrolla su notación escrita (tabla del caso de uso).

Mostrar retroalimentación

Los diagramas de casos de uso quedan encuadrados en la fase de análisis de los proyectos, se entienden como una puesta en común entre el cliente y el analista sobre como entender requisitos funcionales.

Al utilizarse UML, se entiende que ambos interlocutores conocen la notación propia de estos diagramas y las reglas sobre como combinar los diferentes símbolos, de forma que no hay ambigüedades que sí podrían darse mediante una descripción escrita.

Completar un diagrama de casos de uso supone versionar el diagrama tantas veces como sean necesarias hasta que la vista del proyecto que se presenta al cliente garantiza que el requisito funcional ha sido entendido.

En nuestros ejercicios, sólo hay una descripción que no permite ir refinando el diagrama mediante sucesivas consultas con el cliente, por lo tanto, las soluciones propuestas podrán ser diversas, todas ellas válidas siempre que reflejen razonablemente lo propuesto en el enunciado.

A la hora de valorar el ejercicio, lo que no es interpretable, es que el uso de la notación sea el correcto y que se muestre una variedad de los símbolos que conocemos para estos diagramas.

Con estas consideraciones, se propone la siguiente solución:

Actores que participan en el problema.

Zapatería (web), usuario (que podrá ser socio o no socio), productos/artículos (que podrán ser zapatos o complementos), y pedidos.

Generalizaciones.

Se observan dos posibles generalizaciones, que en el desarrollo del diagrama podrán ser expresadas en relación a actores o en relación a los casos de uso en los que participan. En la solución propuesta se ha decidido hacer una generalización de actores -- usuario socio o no socio -- y de casos de uso -- comprar zapatos o complementos (que podría haber sido una generalización de productos) --.

Relaciones extends.

De la ejecución del caso de uso visualizar pedido se pueden derivar los casos de uso cancelar pedido y comprar artículo. Se trata de casos de uso que no tienen porque siempre llevarse a cabo.

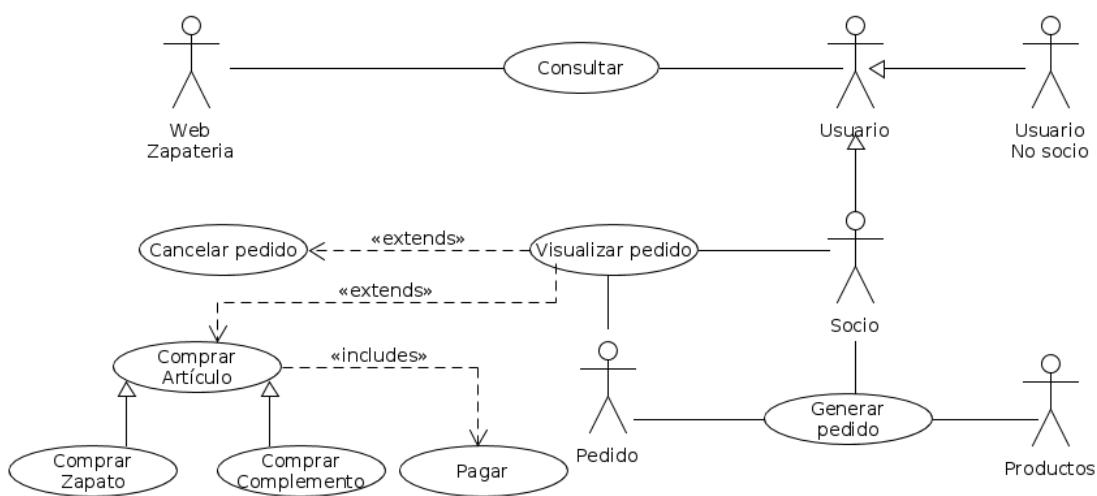
Relaciones includes/use.

Cuando se realiza la compra de un artículo, siempre habrá que pagar.

Notas:

- Puesto que todos los usuarios pueden consultar la web de la zapatería, la relación consultar web la hace el actor usuario (padre de la relación de generalización), en cambio, generar pedido sólo puede llevarse a cabo por los socios, así que es éste el que participa en este caso de uso.
- También podría haberse considerado el actor entidad bancaria, partícipe del caso de uso pagar.

Quedando el siguiente diagrama.



Como se indicaba en los contenidos del apartado 2.2, lo más importante en la elaboración de un diagrama de casos de uso, no es el diagrama en sí, sino la documentación de los casos de uso que es lo que permitirá desarrollar otros diagramas que ayuden en la codificación del sistema, y la elaboración de los casos de prueba de caja negra.

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

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

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

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

- ✓ En la pestaña "**General**" rellenamos el nombre "**Hacer pedido**", y tenemos un espacio para escribir una breve descripción del caso de uso, por ejemplo:

"El cliente visualiza los productos que están a la venta, que se pueden seleccionar para añadirlos al pedido. Puede añadir tantos artículos como desee, cada artículo añadido modifica el total a pagar según su precio y la cantidad seleccionada.

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

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

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

Al finalizar un pedido se añade al sistema con el estado pendiente."
- ✓ En la pestaña "**Valores etiquetados**" encontramos un conjunto de campos predefinidos, entre los que se encuentran el autor, precondiciones y postcondiciones, que podemos llenar de la siguiente manera:

Autor: usuario.

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

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

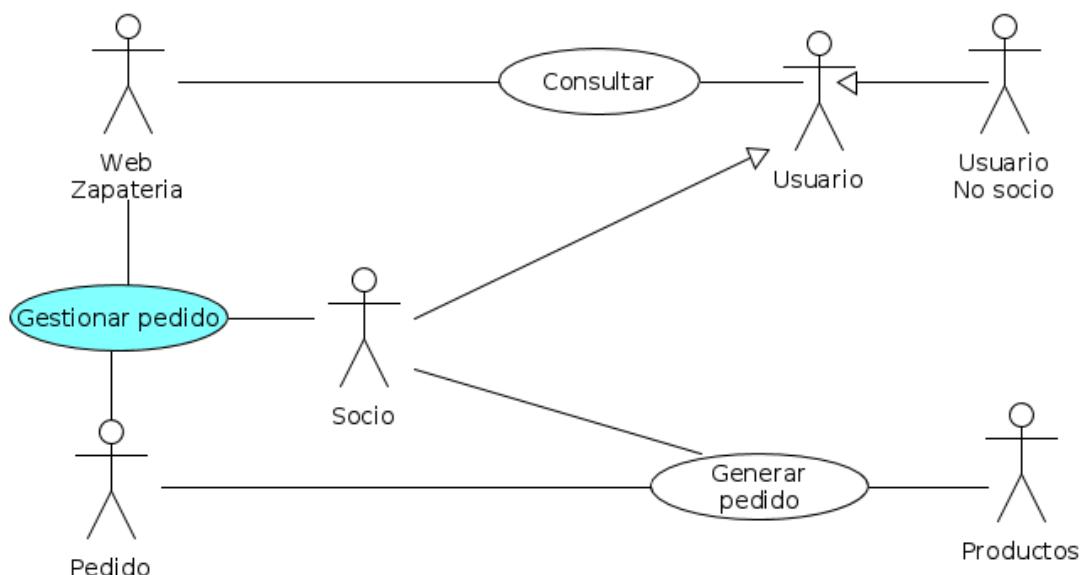
También se pueden incluir otros datos como la complejidad, el estado de realización del caso de uso la complejidad que no hemos visto en esta unidad.

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

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

Tabla gestionar pedido.

Una vez adaptado el diagrama para obtener el caso de uso gestionar pedidos, podría quedar como sigue:



La tabla resultante sería:

Caso de uso	Gestionar pedidos.		
Actores	Principal: usuario-socio. Secundarios: pedido, zapatería.		
Propósito	Gestión de los pedidos ya dados de alta en el sistema.		
Pre-condiciones	El pedido ha sido generado y está en modo activo. El usuario debe ser socio.		
Flujo normal	El flujo normal finaliza con la compra del pedido.	Visualizado del pedido. Compra del pedido. Pago del pedido. Fin de flujo.	
Flujos alternativos	Cancelación del pedido.	Visualizado del pedido. Cancelación del pedido. El pedido desaparece de la lista de pedidos activos.	

		Fin de flujo.
	Imposibilidad de efectuar el pago.	Visualizado del pedido.
		Compra del pedido.
		Error en la operación de pago. La compra del pedido no se completa.
		Fin de flujo.
Post-condiciones		Las post-condiciones son estados en los que ha de quedar el sistema siempre, sea cual sea el flujo ejecutado (normal o alternativo). Para este enunciado no se identifica ninguna post-condición en particular.
Requerimientos trazados		Los que correspondan en el documento de requisitos. Recuerda que el tipo de requisitos que se consideran en los diagramas de casos de uso son siempre de tipo funcional.
Puntos de inclusión	de	Ninguno. Este diagrama no dispone de otros casos de uso con los que establecer estas relaciones.
Puntos extensión	de	Ninguno. Este diagrama no dispone de otros casos de uso con los que establecer estas relaciones.

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

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

Use Case	Hacer pedido		
Author	usuario		
Date	26-agosto-2011 13:56:56		
Brief Description	El usuario selecciona un conjunto de artículos, junto con la cantidad de los mismos, para crear el pedido. Cuando se formaliza se comprueba que el usuario sea socio. A continuación se comprueban los datos bancarios, se realiza el cobro y se crea el pedido.		
Preconditions	Existe un catálogo de productos disponibles para pedir. El usuario está registrado. Los datos bancarios son correctos.		
Post-conditions	Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.		
Flow of Events		Actor Input	System Response
	1	Inicia el pedido.	
	2		Se crea un pedido en estado "en construcción".
	3	Selecciona un artículo.	
	4	Selecciona una cantidad.	

5		Recupera la información del artículo para obtener el precio y modifica el precio total del pedido.
6	El proceso se repite hasta completar la lista de artículos.	
7	Se acepta el pedido.	
8		Se comprueba si el usuario es socio, si no lo es se le muestra un aviso para que se registre en el sitio.
9		Se comprueban los datos bancarios con una entidad externa.
10		Se genera: calcula el total, sumando los gastos de envío.
11		Se realiza el pago a través de la entidad externa.
12		Se almacena la información del pedido con el estado "pendiente".

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

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

Author	usuario	
Date	26-agosto-2011 17:56:35	
Brief Description	Cuando el usuario hace un pedido si no está registrado se abre la opción de registro para que se dé de alta.	
Preconditions	El usuario no está registrado. Existe un catálogo de artículos para hacer pedido. Los datos bancarios son correctos.	
Post-conditions	El usuario queda registrado. Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.	
Flow of Events	Actor Input	System Response
1	Inicia el pedido.	
2		Se crea un pedido en estado "en construcción".
3	Selecciona un artículo.	
4	Selecciona la cantidad.	
5		Recupera la información del artículo para obtener su precio y modifica el precio total a pagar por el pedido.

6		Añade la información al pedido en creación.
7	EL proceso se repite hasta completar la lista de artículos del pedido.	
8	Acepta el pedido.	
9		Se comprueba si el usuario es socio.
10		Se invoca el registro de usuario.
11		Se comprueban los datos bancarios con una entidad externa.
12		Se genera: calcula el total, sumando los gastos de envío.
13		Se realiza el pago a través de la entidad externa.
14		El pedido queda almacenado con el estado "pendiente".

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

Author	usuario																			
Date	26-agosto-2011 18:14:35																			
Brief Description	Una vez que se han seleccionado los artículos y se han introducido los datos del socio, al hacer la comprobación de los datos bancarios con la entidad externa se produce algún error, se da la posibilidad al usuario de modificar los datos o de cancelar el pedido.																			
Preconditions	Existe un catálogo de productos disponibles para pedir. El usuario está registrado. Los datos bancarios no son correctos.																			
Post-conditions	Se crea un pedido con los datos del usuario que lo realiza y los artículos solicitados.																			
Flow of Events	<table border="1"> <thead> <tr> <th></th> <th>Actor Input</th> <th>System Response</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Inicia el pedido.</td> <td></td> </tr> <tr> <td>2</td> <td></td> <td>Se crea un pedido en estado "en construcción".</td> </tr> <tr> <td>3</td> <td>Selecciona un artículo.</td> <td></td> </tr> <tr> <td>4</td> <td>Selecciona una cantidad.</td> <td></td> </tr> <tr> <td>5</td> <td></td> <td>Recupera la información del artículo para obtener el precio y modifica el</td> </tr> </tbody> </table>			Actor Input	System Response	1	Inicia el pedido.		2		Se crea un pedido en estado "en construcción".	3	Selecciona un artículo.		4	Selecciona una cantidad.		5		Recupera la información del artículo para obtener el precio y modifica el
	Actor Input	System Response																		
1	Inicia el pedido.																			
2		Se crea un pedido en estado "en construcción".																		
3	Selecciona un artículo.																			
4	Selecciona una cantidad.																			
5		Recupera la información del artículo para obtener el precio y modifica el																		

		precio total del pedido.
6	El proceso se repite hasta completar la lista de artículos.	
7	Se acepta el pedido.	
8	Acepta el pedido.	
9		Se comprueban los datos bancarios con una entidad externa, fallando la comprobación.
10		Se solicitan los datos de nuevo.
11	Introduce los datos de nuevo.	
12		Se repite el proceso hasta que se acepten los datos bancarios o se cancele la operación.
13		Se genera: calcula el total, sumando los gastos de envío.
14		Se realiza el pago a través de la entidad externa.
15		Se almacena la información del pedido con el estado "pendiente".

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

Debes conocer

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

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

Primeros pasos

- Antes de elaborar el diagrama tienes que leer con detenimiento el documento con la especificación del problema a resolver y asegurarte de que entiendes la idea central del problema, crear una tienda virtual en la que se puedan realizar pedidos de los productos a la venta (zapatos). El proceso se centra en el pedido, desde poner a disposición del cliente los artículos en venta, pasando por la selección de artículos a pedir, la cumplimentación de toda la información necesaria para el pedido, pago, confección del pedido, envío y reajuste del stock en almacén, todo ello, a través de la web.



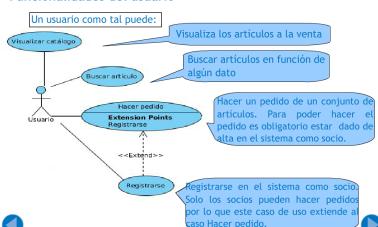
Identificar funcionalidades

- Para facilitar la creación del diagrama vamos a ir sacando funcionalidades para cada usuario.
- Debemos recordar que una caso de uso representa una interacción de un actor con el sistema, que está relacionado con los requisitos funcionales de la aplicación final y que, en definitiva representa tareas que llevará a cabo el sistema.

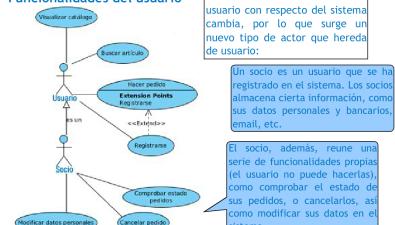
Funcionalidades del usuario

- Cuando una persona se conecta al sistema lo primero que podrá hacer será visualizar el catálogo de la temporada.
- También puede hacer un pedido con uno o varios artículos del catálogo, para ello visualizará los artículos de forma que pueda seleccionar algunos de ellos e indicar la cantidad que quiere comprar.
- También puede hacer búsquedas por datos concretos de artículos.
- Cuquier persona que acceda al sistema puede darse de alta para ser socio.
- Así mismo, si es socio, podrá comprobar el estado de sus pedidos y cancelarlos.

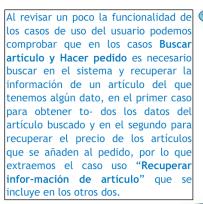
Funcionalidades del usuario



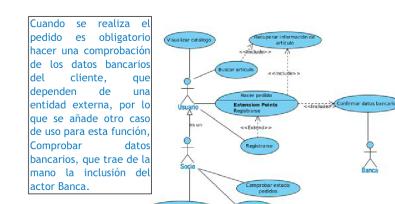
Funcionalidades del usuario



Funcionalidades del usuario



Funcionalidades del usuario



Funcionalidades del administrador

- El objetivo del administrador web es gestionar los contenidos de la web, en concreto de las diferentes campañas, ya que cada temporada se debe cerrar la campaña antigua, retirando los artículos de la temporada anterior y abrir la temporada nueva, añadiendo sus artículos. Para que se pueda cerrar una temporada es necesario que en el almacén se hayan gestionado todos sus pedidos, por lo que es obligatorio comprobarlo, antes de cerrar.

Funcionalidades del administrador



Funcionalidades del responsable de almacén

- Es el encargado de leer los pedidos de los usuarios y cumplimentarlos, está será su única función, si bien, es una función complicada, ya que implica realizar una serie de tareas:

 - Seleccionar el pedido más antiguo.
 - Buscar los artículos a servir.
 - Empaquetarlos junto con un albarán para el socio.
 - Colocarlos en su ruta de envío.

Funcionalidades del responsable de almacén



2.5.- Ejercicio resuelto 2 ("QUIJOTE")(Elaboración de un diagrama de casos de uso).

Ejercicio Propuesto

La empresa Quijote se dedica a la venta de material informático puerta a puerta, ofrece sus productos a los clientes en sus propios domicilios. Sus empleados se organizan en dos grandes grupos: vendedores y publicitarios. Los publicitarios tratan de facilitar el acceso de los vendedores a los clientes para que éstos les hagan llegar los catálogos de productos y realicen las operaciones de ventas.

Los publicitarios anualmente encargan a la consultora Sancho un estudio de sus resultados, y en función de los datos que desprenda pueden realizar un análisis de mercado. Gracias a la información obtenida en el análisis se hacen campañas publicitarias en radio y televisión.

La política de la empresa Quijote se basa en tener grandes profesionales en sus filas, por lo que todos sus empleados reciben formación periódicamente.

Se pide:

- Diagrama de casos de uso. Identifica los actores y casos de uso, incluye relaciones de asociación, identifica generalizaciones. Si consideras alguna relación tipo include o extend, justifica su uso.
- Si agrupamos todos los casos de uso que hayas considerado en el proceso relacionado con los empleados de marketing/publicitarios en uno sólo. Desarrolla la notación escrita del caso de uso.

[Mostrar retroalimentación](#)

Los diagramas de casos quedan encuadrados en la fase de análisis de los proyectos, se entienden como una puesta en común entre el cliente y el analista sobre como entender requisitos funcionales.

Al utilizarse UML, se entiende que ambos interlocutores conocen la notación propia de estos diagramas y las reglas sobre como combinar los diferentes símbolos, de forma que no hay ambigüedades que sí podrían darse mediante una descripción escrita.

Completar un diagrama de casos de uso supone versionar el diagrama tantas veces como sean necesarias hasta que la vista del proyecto que se presenta al cliente garantiza que el requisito funcional ha sido entendido.

En nuestros ejercicios, sólo hay una descripción que no permite ir refinando el diagrama mediante sucesivas consultas con el cliente, por lo tanto, las soluciones propuestas podrán ser diversas, todas ellas válidas siempre que reflejen razonablemente lo propuesto en el enunciado.

A la hora de valorar el ejercicio, lo que no es interpretable, es que el uso de la notación sea el correcto y que se muestre una variedad de los símbolos que conocemos para estos diagramas.

Con estas consideraciones, se propone la siguiente solución:

Actores que participan en el problema.

Empleado (que podrá ser ventas o publicitario), cliente y consultora.

Generalizaciones.

Se observan dos posibles generalizaciones, que en el desarrollo del diagrama podrán ser expresadas en relación a actores o en relación a los casos de uso en los que participan. En la solución propuesta se ha decidido hacer una generalización de actores -- empleado de ventas y empleado publicitario -- y de casos de uso -- lanzar campaña por radio o por televisión --.

Relaciones extends.

De la ejecución del caso de uso comprobar resultados se puede derivar el caso de uso analizar mercado.

Relaciones includes/use. Cuando se realiza un análisis de mercado, siempre se lanza una campaña publicitaria.

Quedando el siguiente diagrama.

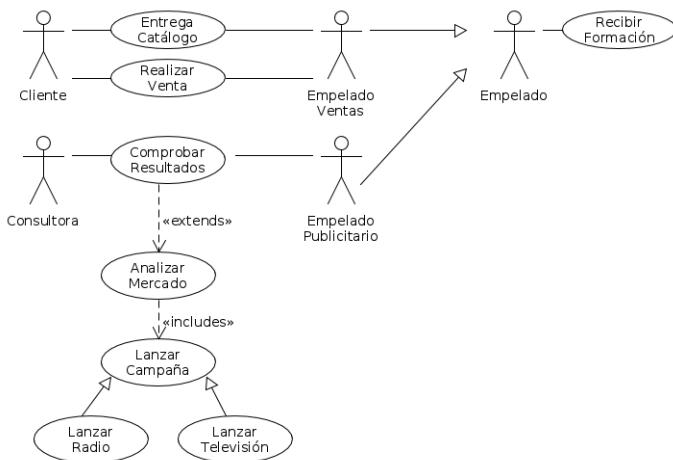


Tabla promocionar Quijote.

Una vez adaptado el diagrama para obtener el caso de uso promocionar Quijote, podría quedar como sigue:



La tabla resultante sería:

Caso de uso	Promocionar Quijote.	
Actores	Empleado marketing (principal), consultor (secundario).	
Propósito	Promocionar la empresa para ganar nuevos clientes.	
Pre-condiciones	Seleccionar consultora con experiencia en el sector.	
Flujo normal	<p>Como flujo normal vamos a suponer que del estudio se genera un análisis de mercado, sería igualmente válida la decisión contraria.</p>	<p>El empleado solicita comprobar los resultados del año en curso a la consultora.</p> <p>La consultora concluye que se precisa un análisis de mercado de acuerdo al estado actual de la empresa.</p> <p>Se lanza una campaña publicitaria en radio.</p> <p>Se lanza una campaña publicitaria en televisión.</p> <p>Fin de flujo.</p>
Flujos alternativos	<p>El análisis de mercado no muestra potenciales clientes.</p>	<p>El empleado solicita comprobar los resultados del año en curso a la consultora.</p> <p>La consultora concluye que se precisa un análisis de mercado de</p>

		acuerdo al estado actual de la empresa, pero no se detectan clientes futuros.
		Fin de flujo.
	El presupuesto para las campañas publicitarias disponible no es suficiente	El empleado solicita comprobar los resultados del año en curso a la consultora.
		La consultora concluye que se precisa un análisis de mercado de acuerdo al estado actual de la empresa.
		Se lanza una campaña publicitaria en radio.
		Se suspende la campaña publicitaria en televisión por falta de fondos.
		Fin de flujo.
Post-condiciones	No se identifican en este ejercicio. Serían aquellas condiciones que se deben cumplir <u>siempre</u> al finalizar el caso de uso sea cual sea el flujo ejecutado.	
Requisitos trazados	Los que correspondan en el documento de requisitos. Recuerda que el tipo de requisitos que se consideran en los diagramas de casos de uso son siempre de tipo funcional.	
Puntos de inclusión	Ninguno. Este diagrama no dispone de otros casos de uso con los que establecer estas relaciones.	
Puntos extensión	Ninguno. Este diagrama no dispone de otros casos de uso con los que establecer estas relaciones	

2.6.- Ejercicio resuelto 3 ("ALQUILER DE PISOS Y LOCALES")(Elaboración de un diagrama de casos de usos).

Ejercicio Propuesto

Una empresa de alquiler de pisos y locales desea diseñar un sistema que cumpla los siguientes requisitos:

- Los propietarios previa identificación en el sistema, podrán dar de alta o baja un piso o un local. También podrán modificar los datos de ese piso o local.
- Los futuros inquilinos también deben identificarse antes de poder usar el sistema. Al acceder se les presenta un menú donde pueden elegir la acción a realizar:
 1. listar pisos y locales disponibles.
 2. solicitar el alquiler de un piso o local.

Para alquilar un local se le pedirá su email y para alquilar un piso su número de teléfono

Diagrama de casos de uso

1. Identificar los actores.
2. Identificar los casos de uso.
3. Implementar con UMLet el diagrama de casos de uso.

[Mostrar retroalimentación](#)

1.-Identificar los actores.

Los actores identificados son : Propietario e Inquilino.

PropietarioEl propietario será el responsable de la gestión de los pisos o locales mediante el alta, la baja y las modificaciones de los mismos. Una vez identificado en el sistema podrá gestionar pisos o locales.

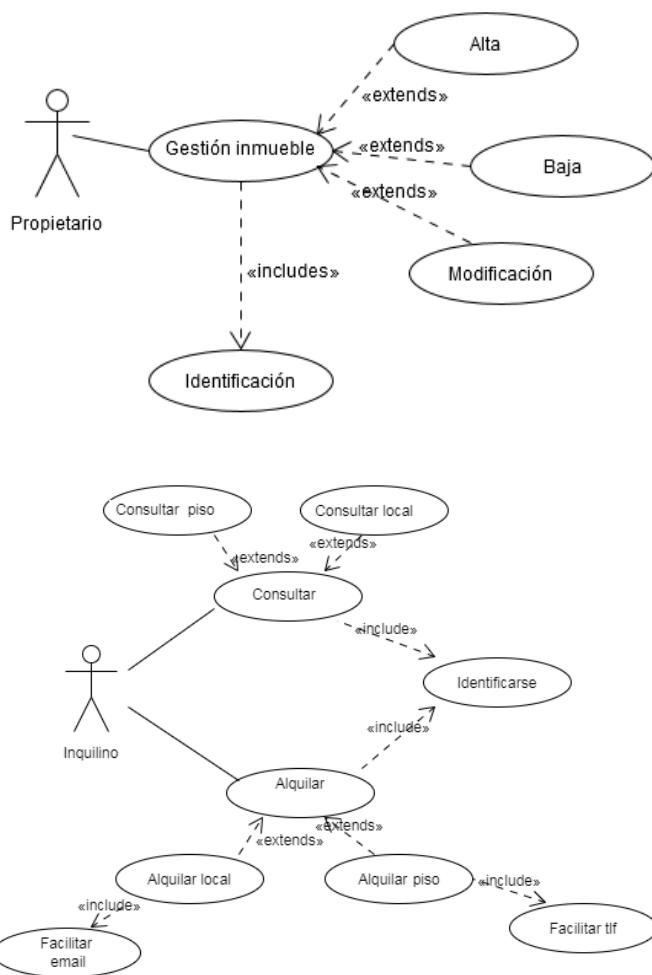
InquilinoEl inquilino consultará los pisos o locales y también podrá solicitar el alquiler. Debe identificarse en el sistema.

2.-Identificar los casos de uso.

El propietario podrá dar de alta, baja dar de baja o modificar un inmueble. Si se ofrece un menú podemos usar extends con estos casos de uso y uno llamado gestión inmueble. El inquilino siempre que quiera gestionar un inmueble se identifica.

El inquilino podrá consultar o alquilar. En base a lo que elija podrá ser local o piso. Cuando alquile deberá facilitar teléfono o email según proceda. También debe identificarse para realizar las acciones.

3.-Implementar con UMLet el diagrama de casos de uso.



3.-Diagrama de interacción

Una vez conocidos los diagramas de casos de uso, se hace necesario buscar la forma de representar como circula la información, los objetos que participan en los casos de uso, los mensajes que envían, y en el momento en que se producen. Disponer de esta información ayudará con posterioridad en el desarrollo de los diagramas de clases.

Los **diagramas de interacción** son vistas del sistema que muestran como grupos de objetos interactúan para un cierto comportamiento. Captan la ejecución de los casos de uso, representando a los actores que participan y los mensajes que se pasan.

Hay dos tipos de diagramas de interacción: **diagramas de secuencia** y **diagramas de colaboración**.

El diagrama de colaboración contiene la misma información que un diagrama de secuencia, pero la anotación es diferente.

3.1.- Diagramas de secuencia.

Caso práctico

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

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

Como siempre, Ada tiene una solución.

-Tendremos que investigar los diagramas de secuencia.



En los **diagramas de secuencia**, los objetos/actores que forman parte del escenario de un caso de uso se representan mediante rectángulos distribuidos horizontalmente en la zona superior del diagrama, a los que se asocia una línea temporal vertical (una para cada actor) de las que salen, en orden, los diferentes mensajes que se pasan entre ellos.

Con esto el equipo de desarrollo puede hacerse una idea de las diferentes operaciones que deben ocurrir al ejecutarse una determinada tarea y el orden en que deben realizarse.

Reflexiona

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

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

Caso práctico

Ada, orienta a su equipo:

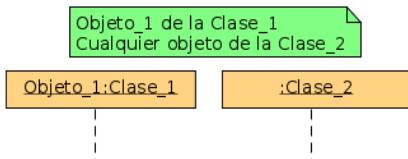
-Bien, ¿qué nos haría falta para poder representar la interacción de los objetos que participan en el caso de uso a lo largo del tiempo?

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



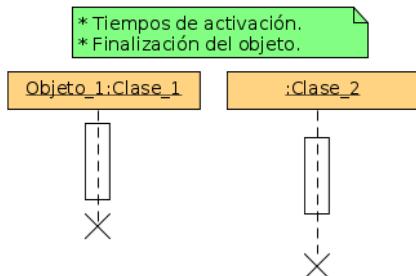
Representación de objetos y linea de vida.

En un **diagrama de secuencia**, los **objetos** se dibujan mediante rectángulos y se distribuyen horizontalmente en la parte superior del diagrama. Por cada objeto se identifica su nombre, seguido del símbolo de dos puntos y a continuación el nombre de la clase a la que pertenece. Si no se indica el nombre del objeto, se considera que para el propósito del diagrama es válido cualquier objeto de la clase.



De cada rectángulo sale una línea punteada que representa el **paso del tiempo**, se denomina **línea de vida**. La línea de vida se prolonga mientras el objeto es relevante en el diagrama, una vez deja de serlo se indica mediante una cruz "X", dejando por tanto de existir a partir de ese momento.

Cuando el objeto toma protagonismo en el intercambio de mensajes, se dice que está **activo** y se indica mediante un recuadro sobre su línea de vida.

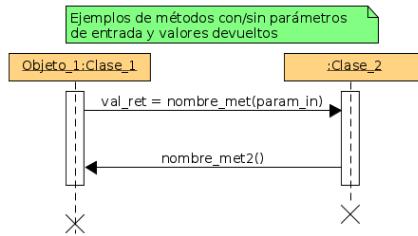


Una línea de vida puede estar encabezada por otro tipo de instancias como el sistema o un actor que aparecerán con su propio nombre. Usaremos el sistema para representar solicitudes al mismo, como por ejemplo pulsar un botón para abrir una ventana o una llamada a una subrutina

Paso de mensajes (Invocación de métodos).

Los **mensajes**, que significan la invocación de métodos, se representan como flechas horizontales que van de una línea de vida a otra, indicando con la flecha la dirección del mensaje. Los mensajes se dibujan desde el objeto que envía el mensaje al que lo recibe, pudiendo ser el mismo objeto emisor y receptor de un mensaje. El orden en el tiempo viene determinado por su posición vertical, un mensaje que se dibuja debajo

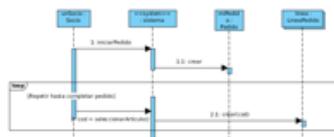
de otro indica que se envía después, por lo que no se hace necesario un número de secuencia. Los **mensajes** tendrán un **nombre** y pueden incluir **argumentos de entrada**, **valores devueltos** e **información de control** (condición o iteración).



Una notación alternativa para recoger valores devueltos por los métodos es dibujar una línea de puntos finalizada en flecha, que irá desde el objeto destinatario del mensaje al que lo ha generado, acompañado del texto del valor devuelto.

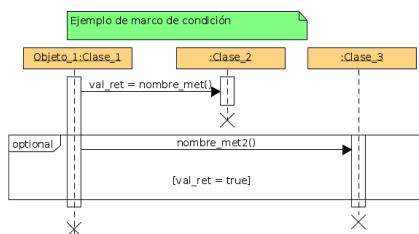
Condicionales e iteraciones.

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



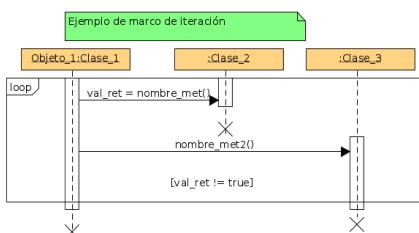
Las secuencias de control; tanto condicionales, como iterativas, se pueden representar usando **marcos**, normalmente se nombra el marco con el tipo de bucle a ejecutar y la condición de parada. También se pueden representar flujos de mensajes condicionales en función de un valor determinado.

La expresión a evaluar para la condición o iteración se representa entre corchetes.



Combinando varios marcos opcionales es posible representar diferentes alternativas en la ejecución de un diagrama de secuencia.

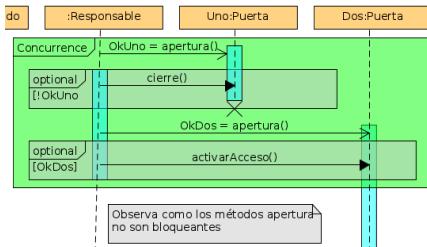
Para el caso de una iteración, tenemos el siguiente ejemplo.



Por defecto los métodos son **bloqueantes**, se entiende que el proceso del diagrama de secuencia completa cada método antes de continuar con el siguiente, es una secuencia de métodos en el tiempo. Pero en ocasiones se producen situaciones en las que se desea mostrar varios procesos en paralelo (**conurrencia**), se puede reflejar mediante el uso de marcos con la etiqueta **concurrence**.

Junto a los marcos de concurrencia, se hace necesario el uso de **métodos no bloqueantes (asíncronos)**, que permitan en paralelo activar diferentes procesos. La notación utilizada para los métodos asíncronos es

una línea finalizada con media cabeza de flecha o en UMLet una línea cuya punta flecha no está rellena.



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

Autoevaluación

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

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

Incorrecto, pueden intervenir porque desencadenan acciones que producen la invocación de mensajes.

No es correcto, son una parte esencial de un diagrama de secuencia.

Falso, son útiles para representar acciones iterativas.

Así es, los eventos no tienen representación en el diagrama de secuencia que se centra más en el envío de mensajes.

Solución

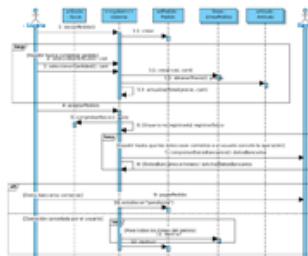
1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

3.1.2.- Ejercicio resuelto 1 ("Generar pedido") (Elaboración de un diagrama de secuencias).

Diagrama de interacción: Diagrama de secuencia

Vamos a generar el diagrama de secuencia que representa el caso de uso "**Generar pedido**" del diagrama de casos de uso del ejercicio resuelto 1 "ZAPATERÍA TACÓN DE ORO" (en el punto 2 de "Los diagramas de casos de uso"). En dicho diagrama se establece la secuencia de operaciones que se llevarán a cabo entre los diferentes objetos que intervienen en el caso de uso.

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



[Resumen textual alternativo](#)

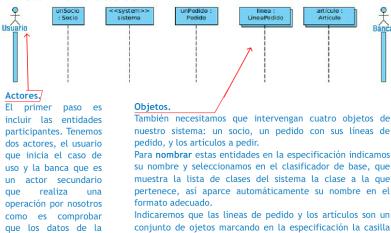
Debes conocer

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

Crear el diagrama de secuencia

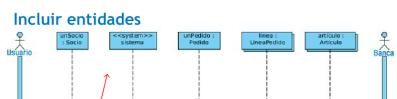
- El diagrama de secuencia que vamos a tratar es el del caso de uso Hacer pedido, que hemos descrito en el apartado anterior. Si no recuerdas bien cuál era la descripción del caso te invito a que la repases en el punto 2.4 de los contenidos de la unidad.
- Este es un diagrama muy completo que incluye bastantes elementos de este diagrama, como bucles o condicionantes, veamos como incluirlos con la herramienta Visual Paradigm, como siempre, debes saber que puedes investigar y utilizar otras herramientas que sirvan para este mismo propósito.

Incluir entidades



Mensajes

Los mensajes pueden devolver un valor, que podemos escribir en la especificación del mismo en la opción return value.

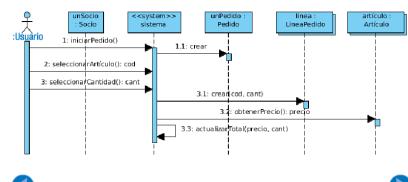


Sistema.
Por último también interviene el sistema en si mismo, que utilizaremos para las operaciones relacionadas con la interfaz gráfica y las que se lanzan directamente para crear objetos, recuperar información del sistema como los datos de un artículo o comunicarse con entidades externas como la banca.

Se añade como una línea de vida más, a la que en su especificación indicamos que su nombre es sistema, y en la pestaña Estereotipos seleccionamos System.

Mensajes

En esta imagen vemos los mensajes correspondientes al apartado de añadir los datos del pedido. Se selecciona el artículo y la cantidad, se crea una nueva línea de pedido y se recuperan los datos del artículo para obtener suprecio y actualizar el total. Puesto que este proceso lo podemos repetir en más de una ocasión lo meteremos en un bucle.



Mensajes

Para añadir un bucle seleccionamos la opción Loop Combined Fragment.

Mensajes

Añadiremos condiciones de guarda cuando queramos indicar que un mensaje se enviará sólo si se cumple cierta condición, por ejemplo, solo registraremos a un usuario si no es socio ya.



Mensajes

También se pueden incluir condiciones más elaboradas que implique una bifurcación en el flujo de eventos, como es el caso de la comprobación de la tarjeta, si todo marcha bien se finalizará la creación del pedido, si no, el usuario puede cancelar la operación y terminar sin guardar nada.

3.1.3.- Ejercicio resuelto 2 ("ESTADIO") (Elaboración de un diagrama de secuencia).

Ejercicio Propuesto

Se pretende desarrollar un programa que dé respuesta a las necesidades de un estadio de fútbol en uno de sus partidos. Tras varias entrevistas con el responsable, se ha llegado al acuerdo de que los requisitos funcionales se pueden recoger en el caso de uso Gestión del estadio donde hay que considerar actividades como:

- **Control de puntos de acceso.** Se dispone de 2 puertas, a las que el responsable del estadio dará orden de apertura. Cada puerta hace un test interno y devuelve al responsable Ok o Ko. Si una puerta no abre, queda inactiva para el resto del partido. El proceso de apertura se realiza de forma simultánea en ambas puertas. Al finalizar el partido, el responsable del estadio dará orden de cierre de las puertas que se han usado durante el partido.
 - **Control de acceso de aficionados.** Si la puerta está disponible, el responsable da orden de iniciar el acceso al estadio. Mientras haya aficionados, el operario de la puerta valida la entrada, si es correcta le da acceso al campo, en caso contrario avisa al responsable del estadio que ha habido un intento de acceso con entrada falsa.

Nota:

- Considera en este diagrama que la puerta uno está averiada y la puerta dos operativa.

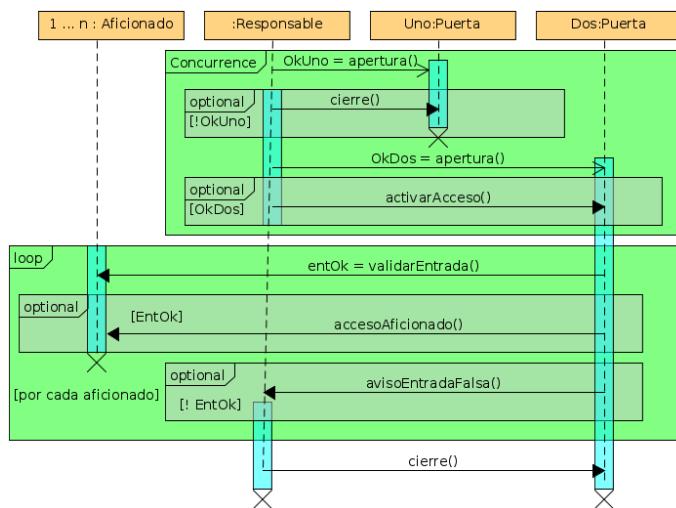
Se pide:

Desarrollar el diagrama de secuencia resultante del caso de uso planteado.

Atiende a aspectos como:

- Identificación de los objetos involucrados en el diagrama de secuencia.
 - Que el diagrama recoja la secuencia de mensajes intercambiados, tomando en consideración las funcionalidades descritas en el enunciado.
 - Uso correcto de la notación vista para este tipo de diagramas.
 - Que los diagramas generados sean visualmente útiles. Un diagrama de secuencia debe dar una idea clara/rápida de la secuencia de acciones que se derivan de la ejecución del caso de uso que representa.

[Mostrar retroalimentación](#)



3.1.4.- Ejercicio resuelto 3 ("ROPERO") (Elaboración de un diagrama de secuencia).

Ejercicio Propuesto

Se pretende crear un programa para la gestión de ciertas funciones de una discoteca. En particular, el depósito de los abrigos en el ropero.

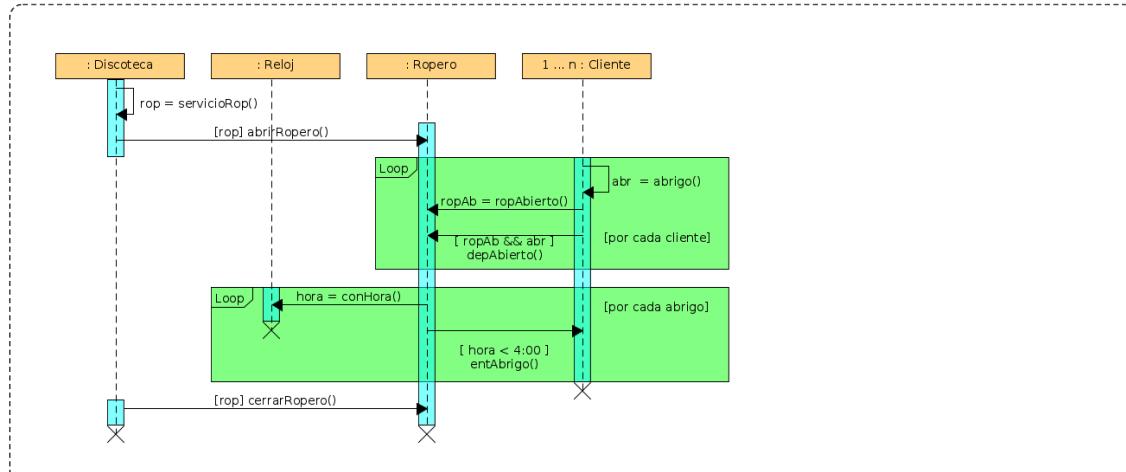
Algunas consideraciones:

- El responsable de la discoteca cada día decide si abre el vestíbulo, y es el encargado de su apertura y cierre.
- El propio ropero informa a los clientes si proporciona o no servicio de guardarropa.
- Si está abierto, a primera hora recoge los abrigos de los clientes que tengan abrigo.
- Al finalizar la jornada, mientras haya abrigos en el ropero se devuelven siempre que sea antes de las 4 de la mañana.
- Se pide desarrollar el diagrama de secuencia resultante del caso de uso planteado.

Considera aspectos como:

- Identificación de los objetos involucrados.
- Que el diagrama recoja la secuencia de mensajes intercambiados, tomando en consideración las funcionalidades descritas en el enunciado.
- Uso correcto de la notación vista para este tipo de diagramas.

Mostrar retroalimentación



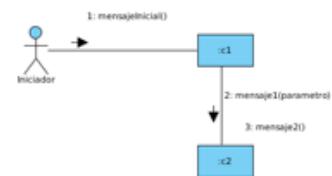
3.2.- Diagramas de colaboración.

Caso práctico

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



Al igual que los diagramas de secuencia, **los diagramas de colaboración** muestran una secuencia de ejecución de uno o varios casos de uso. La notación utilizada es muy similar y la principal diferencia radica en el modo de mostrar el orden de mensajes intercambiados entre objetos. Mientras el diagrama de secuencia establece el orden de los mensajes en el tiempo según su posición de arriba-abajo, el diagrama de colaboración lo hace mediante el etiquetado de los mensajes. Las interacciones entre los objetos se describen en forma de grafo en el que los nodos son objetos y las aristas son enlaces entre objetos a través de los cuales se pueden enviar mensajes entre ellos.



Los **diagramas de colaboración permiten una mejor organización visual de los objetos** al no ser obligada su representación en la parte superior del diagrama, en cambio la secuencia temporal suele ser más complicada de seguir.

Los diagramas de colaboración tienen forma de grafo en el que los nodos son objetos y las aristas son los mensajes que intercambian.

UMLet no dispone de herramientas para la elaboración de diagramas de colaboración directamente. No obstante, no resulta complicado generarlos a partir de los símbolos disponibles para otros diagramas: representación de objetos mediante cajas, paso de mensajes mediante líneas, información de los métodos mediante descripciones textuales; todos ellos disponibles en los diagramas de secuencia de UMLet.

Reflexiona

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

[Mostrar retroalimentación](#)

Es cierto, ambos diagramas representan la misma información, representan entidades del sistema y los mensajes que circulan entre ellas, además en ambos casos es fácil detectar la secuencialidad bien por el uso de líneas de vida, bien por los números de secuencia. De hecho en algunas aplicaciones para el desarrollo de estos diagramas es posible crear un diagrama a partir del otro.

3.2.1.- Representación de objetos.

Caso práctico

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



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

Clase

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

:objeto

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

:Clase

objeto:clase

3.2.2.- Paso de mensajes.

Caso práctico

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



Para que sea posible el paso de mensajes es necesario que exista una asociación entre los objetos, que se podrá mostrar mediante una línea que los une y una flecha que indique la dirección.

Al igual que sucedía en los diagramas de secuencia, es posible incluir parámetros en los mensajes, valores devueltos, mensajes enviados sólo si se cumple una determinada condición, o mensajes que se ejecutan varias veces (iteraciones).

La **sintaxis de un mensaje** es la siguiente:

[Secuencia] [*] [Condición] {valorDevuelto} : mensaje (argumentos de entrada) o
[Secuencia] [*] [Condición] mensaje (argumentos de entrada) : {valorDevuelto}

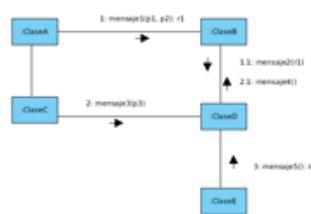
Donde:

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

La enumeración de los mensajes se puede hacer de dos formas:

- Numeración simple: empieza en 1, se va incrementando en 1 y no hay ningún nivel de anidamiento.
- Numeración decimal: se muestran varios niveles de subíndices para indicar anidamiento de operaciones. Por ejemplo, 1 es el primer mensaje; 1.1 es el primer mensaje anidado en el mensaje 1, 1.2 es el segundo mensaje anidado en el mensaje 1; y así sucesivamente.

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



Los mensajes 1, 1.1 y 2 tienen parámetros y los mensajes 1 y 3 devuelven un resultado.

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

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

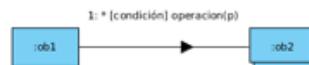
Cuando tenemos una condición se repite el número de secuencia y se añaden las condiciones necesarias, como vemos en la imagen según la condición se enviará el mensaje 1 o el 2, pero no ambos, por lo que coinciden en número de secuencia.

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

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

Autoevaluación

Indica qué afirmación no es correcta para el siguiente diagrama:



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

Esta afirmación es correcta, ya que el icono doble indica que así es.

Así es, según la dirección de la flecha.

Efectivamente, esta afirmación no es correcta, ya que la operación solo se ejecuta si se cumple la condición de guarda.

Esta afirmación es cierta, el asterisco que la precede así lo indica.

Solución

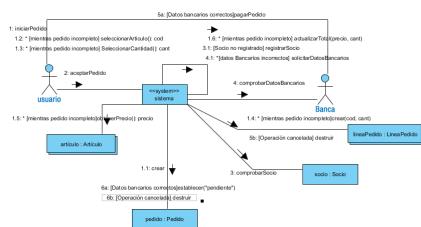
1. Incorrecto
2. Incorrecto
3. Opción correcta
4. Incorrecto

3.2.3.- Ejemplo de un diagrama de colaboración.

Diagrama de interacción: Diagrama de colaboración

A continuación se muestra un diagrama de colaboración de ejemplo.

Este es el diagrama de colaboración que representa el caso de uso "**Generar pedido**" del diagrama de casos de uso del ejercicio resuelto 1 "ZAPATERÍA TACÓN DE ORO" (en el punto 2 de "Los diagramas de casos de uso"). Se ha creado siguiendo el diagrama de secuencia, por lo que no te debe ser muy difícil seguirlo, de hecho algunas aplicaciones para la creación de estos diagramas permiten la obtención de uno a partir de otro. Debes tener en cuenta que la aplicación modifica un poco la signatura de los mensajes, el valor devuelto se representa al final precedido de dos puntos.



Resumen textual alternativo

Los aspectos más destacados son los siguientes:

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

4.- Diagramas de estados.

Caso práctico

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



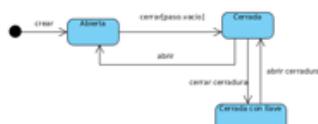
Los **diagramas de estados** permiten analizar como va evolucionando el **estado de un objeto** a lo largo del tiempo, es decir, representa su comportamiento transitando por una serie de estados.

Modelan el **comportamiento dinámico** de los objetos en respuesta a determinados **eventos**.

En relación con el diagrama de estados se cumple que:

- Un **objeto** está en un **estado concreto** en un cierto momento, que principalmente viene determinado, por los **valores de sus atributos**.
- La **transición de un estado** a otro es momentánea y se produce cuando ocurre un determinado **evento**.

Por ejemplo, aquí tenemos el **diagrama de estados de una puerta**.



Autoevaluación

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

- Verdadero.
- Falso.

No ya que no hay una transición directa desde cerrada con llave a abierta, para poder abrirla necesitamos abrir la cerradura primero.

Cierto, antes hay que abrir la cerradura.

Solución

1. Incorrecto
2. Opción correcta

4.1.- Estados y eventos.

Caso práctico

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



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

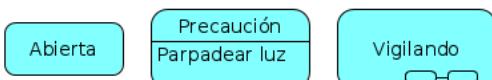
Existen **tres tipos de estado** en los que se puede encontrar un objeto:

- **Estado inicial.** Punto de partida por defecto del diagrama de estados. Corresponde a los valores de los atributos de una clase en el momento de instanciar un nuevo objeto.
- **Estado final.** Estado en el que se encuentra el objeto una vez finalizada la secuencia de eventos que pueden proporcionar transiciones entre estados.
- **Estado intermedio.** Cualquiera de los estados intermedios entre los dos anteriores.



Los estados se representan mediante una caja y admite algunas variantes. La información que se muestra en los estados suele ser:

- **Nombre del estado.** Por ejemplo Abierta.
- **Nombre del estado y acción/actividad asociada al objeto** mientras se encuentra en ese estado. En un semáforo en estado de precaución, se produce la actividad de parpadeo de la luz.
- **Estado con subestados.** En el ejemplo se indica que el estado vigilando tiene asociado una serie de subestados, si se trata de un vigilante de seguridad, el estado "vigilando" podría tener relacionados los subestados de ruta a pie y/o de visionado de cámaras.



Un **evento** es un acontecimiento que dispara una transición entre dos estados del objeto. Existen eventos externos y eventos internos según el agente que los produzca.

Tipos de eventos:

- **Señales (excepciones):** la recepción de una señal, producida por una **situación excepcional en el sistema**. Puede ser origen de una transición entre estados.
- **Llamadas:** la recepción de una petición para **invocar una operación**. Normalmente un evento de llamada es **manejado por un método del objeto**.
- **Paso de tiempo:** el evento se genera como consecuencia del cumplimiento de un temporizador.
- **Cambio de estado:** evento generado por un cambio en el estado o el cumplimiento de una condición.

4.2.- Transiciones.

Caso práctico

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



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

La notación de una transición tiene tres partes, todas ellas optativas:

Evento(argumentos) [Condición] / Acción.

Elementos de una transición:

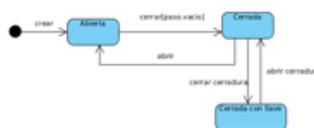
- **Evento:** cuando se produce un evento, afecta a todas las transiciones que lo contienen en su etiqueta.
- **Condición:** expresión evaluable como verdadera o falsa. Si es falsa, la transición no se dispara.
- **Acción:** conjunto de actuaciones que lleva asociada la transición. Puede incluir llamadas a operaciones de objetos, creación o destrucción de objetos ...

Ejemplo: Vamos a ver el **diagrama de estados para un semáforo**. Recoge ejemplos de los tres elementos descritos para las transiciones.



Autoevaluación

Recordemos el diagrama de estado de la puerta:



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

- Que cuando cerremos la puerta el paso quedará vacío.
- Que para cerrar la puerta el paso debe estar vacío.

- Que cuando se está cerrado la puerta se vacía el paso.

No es así, los corchetes no representan una acción a la salida.

Así es, los corchetes en un diagrama UML significan una condición que se debe cumplir. No se podrá cerrar la puerta hasta que el paso no esté vacío.

No es cierto, la acción a realizar se representa con un /.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto

4.3.- Ejercicio resuelto 1 ("Generar pedido") (Elaboración de un diagrama de estados).

Diagrama de estados:

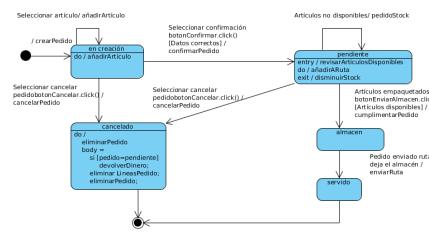
Para exemplificar la creación de un diagrama de estados vamos a ver el que representa el caso de uso "**Generar pedido**" del diagrama de casos de uso del ejercicio resuelto 1 "ZAPATERÍA TACÓN DE ORO" (en el punto 2 de "Los diagramas de casos de uso") que cumple con las condiciones que hemos visto al principio, tiene un comportamiento significativo en tiempo real, ya que su situación tanto física, como el sistema, va evolucionando conforme pasa el tiempo, y participa en varios casos de uso (como Hacer pedido y Cumplimentar pedido).

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

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

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

El diagrama quedaría de la siguiente manera:



Resumen textual alternativo

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

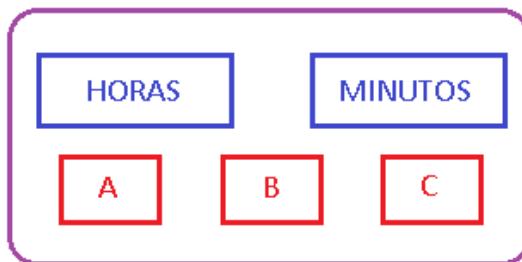
A los **estados** se les ha añadido la acción a realizar, apartado do/ y en algunos casos la acción de entrada, por ejemplo en el caso del estado pendiente, se debe revisar que los artículos a enviar tengan disponibilidad y la de salida, en el ejemplo disminuir el stock.

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

4.4.- Ejercicio resuelto 2 ("RELOJ")(Elaboración de un diagrama de estados).

Ejercicio Propuesto

La siguiente figura muestra un **reloj digital** cuyo comportamiento se describe a continuación:



El reloj se enciende y está visualizando las horas y minutos.

Funciones de reloj:

Pulsado de A durante tres segundos: parpadea la hora. Para evitar cambios de hora involuntarios, si el tiempo de pulsado es inferior a tres segundos no se activa la función.

El botón B no funciona, si no se ha pulsado antes el botón A durante 3 segundos.

De tal forma que si el reloj está en el estado en el que la hora está parpadeando:

1. Si se pulsa el botón B incrementa la hora en una unidad.
2. Si se pulsa el botón A, pasará al estado de poder cambiar los minutos. Los minutos parpadearán. No se precisa mantener pulsado el botón porque se entiende que se está modificando la hora de forma voluntaria.

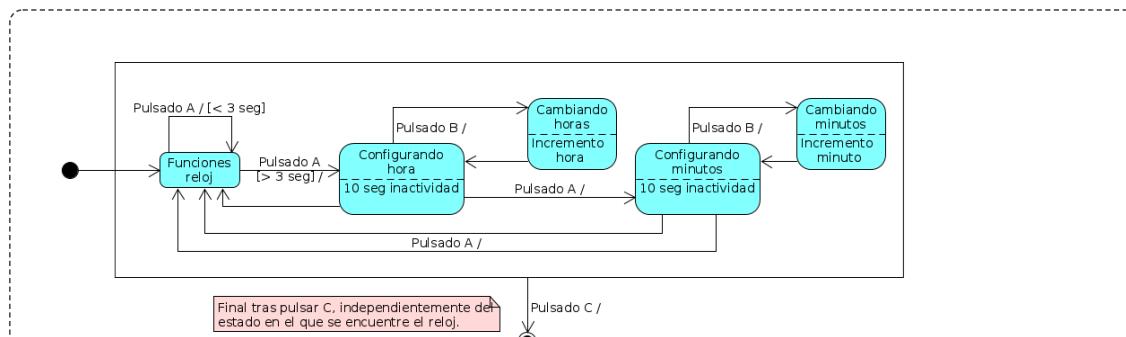
Si está el reloj en el estado de poder cambiar los minutos:

1. Pulsado del botón B: cada pulsación del botón B incrementa los minutos en una unidad
2. Si pulsamos el botón A, finaliza el modo configuración y vuelve a mostrar la hora.

Pulsado de C: apaga del reloj sin tener en consideración el estado en el que se encuentre.

Cuando el reloj está en modo configuración de horas o minutos, tras 10 segundos de inactividad abandona la configuración y pasa a modo funciones de reloj.

[Mostrar retroalimentación](#)



Notas:

- Las transiciones sin eventos, aunque pueden tener condiciones, se producen al finalizar la actividad del estado.
- Es posible salir de un estado tanto por un evento externo, como por el fin de la actividad del estado.
- El evento pulsado C genera el apagado del reloj con independencia del estado en el que se encuentre.

4.5.- Ejercicio resuelto 3 ("VIDA LABORAL") (Elaboración de un diagrama de estados).

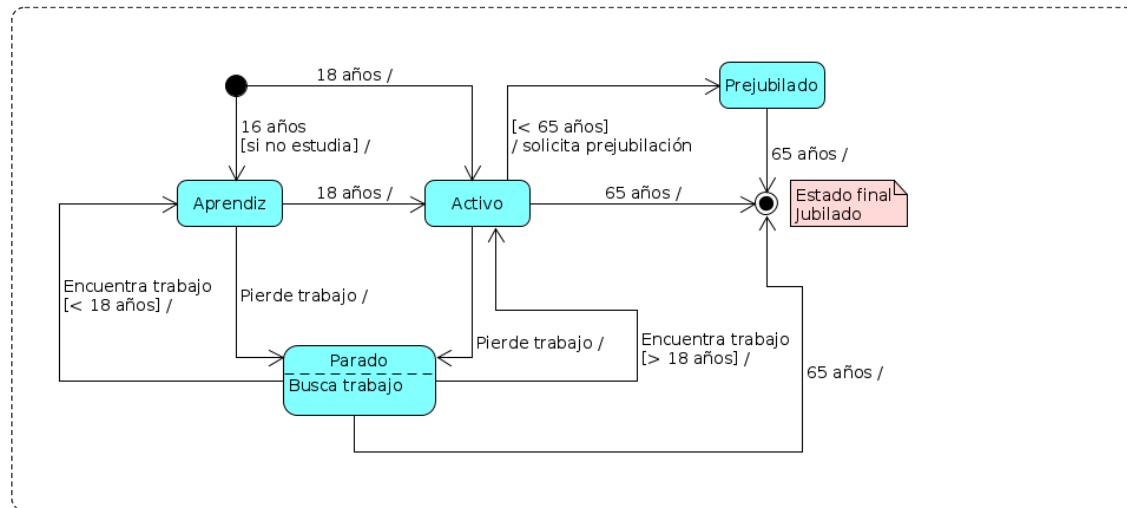
Ejercicio Propuesto

Crea un diagrama de estados que muestre la **evolución de un empleado** a lo largo de su vida laboral.

Se considera que el proceso transcurre entre los 16 y 65 años de edad y se plantean los siguientes **estados**:

- **Preempleado.** Anterior a los 16 años. Se considera el estado inicial.
- **Aprendiz.** Es el periodo comprendido entre los 16 y 18 años para aquellas personas que han decidido no continuar sus estudios.
- **Activo.** El trabajador se encuentra en activo y con contrato en vigor.
- **Parado.** El trabajador ha perdido el empleo, su tarea principal es la búsqueda de un nuevo trabajo.
- **Prejubilado.** El trabajador solicita dejar de estar activo, pero no ha alcanzado la edad de 65 años. Desde el estado de parado no se considera la opción de solicitar la prejubilación.
- **Jubilado.** El trabajador ha cumplido los 65 años y pasa a disfrutar de un merecido descanso. Se considera el estado final.

[Mostrar retroalimentación](#)



5.- Diagramas de actividad.

Caso práctico

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

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

-Pasaría que tendrías que hacer un diagrama de actividad.



El **diagrama de actividad** es una especialización del diagrama de estados, organizado en torno a las **acciones** en lugar de los objetos, que se compone de una serie de actividades y representa como se pasa de unas a otras. **Las actividades se enlazan por transiciones automáticas**, es decir, cuando una actividad termina se desencadena el paso a la siguiente.

El diagrama de actividades resulta útil cuando se quiere representar sólo las acciones que tienen lugar, prescindiendo de quien las genera. ¿Qué pasa primero, qué ocurre después y qué cosas pueden hacerse al mismo tiempo?·

Se utilizan fundamentalmente para modelar el flujo de control entre actividades en el que se puede distinguir cuales ocurren **secuencialmente** a lo largo del tiempo y cuales se pueden llevar a cabo **concurrentemente**. Permite visualizar la dinámica del sistema desde otro punto de vista que complementa al resto de diagramas.

Un diagrama de actividades es un grafo conexo en el que los nodos son **estados**, que pueden ser de **actividad** o de **acción** y los arcos son **transiciones** entre estados.

Reflexiona

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

[Mostrar retroalimentación](#)

En los anteriores diagramas, tratábamos la interacción entre objetos y entidades, cual es el flujo de mensajes, en qué orden y bajo qué condiciones se envían, en los diagramas de actividades se incluye el factor de la concurrencia, y trata sobre todo de expresar el flujo de las actividades, su elemento fundamental, y como se pasa de unas a otras. Además también representa como se influye en los objetos.

5.1.- Elementos del diagrama de actividad.

Caso práctico

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



Normalmente los diagramas de actividades contienen:

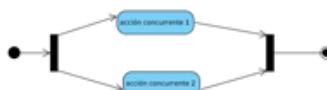
- ✓ **Estados** de actividad y estados de acción.
 - ◆ **Estado de actividad**: Elemento compuesto cuyo flujo de control se compone de otros estados de actividad y de acción.
 - ◆ **Estado de acción**: Estado que representa la ejecución de una acción atómica, que no se puede descomponer ni interrumpir,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



- ✓ **Transiciones**: Relación entre dos estados que indica que un objeto en el primer estado realizará ciertas acciones y pasará al segundo estado cuando ocurra un evento específico y satisfaga ciertas condiciones. Se representa mediante una línea dirigida del estado inicial al siguiente. Podemos encontrar diferentes tipos de transacciones:
 - ◆ **Secuencial o sin disparadores**: Al completar la acción del estado origen se ejecuta la acción de salida y, sin ningún retraso, el control sigue por la transición y pasa al siguiente estado.
 - ◆ **Bifurcación(Decision node)**: Especifica caminos alternativos, elegidos según el valor de alguna expresión booleana. Las condiciones de salida no deben solaparse y deben cubrir todas las posibilidades (puede utilizarse la palabra clave `else`). Pueden utilizarse para lograr el efecto de las iteraciones.



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



- ◆ **Unión (Join node)**: Por definición, en la unión los flujos entrantes se sincronizan, es decir, cada uno espera hasta que todos los flujos de entrada han alcanzado la unión.
- ✓ **Objetos**: Manifestación concreta de una abstracción o instancia de una clase. Cuando interviene un objeto no se utilizan los flujos de eventos habituales sino flujos de objetos (se representan con una flecha de igual manera) que permiten mostrar los objetos que participan dentro del flujo de control

asociado a un diagrama de actividades. Junto a ello se puede indicar cómo cambian los valores de sus atributos, su estado o sus roles.



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

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

Autoevaluación

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

- Verdadero.
- Falso.

Pues así es. Podemos representar acciones concurrentes utilizando las herramientas de fusión y unión, ya que se considera que todas aquellas acciones que queden entre ambas son concurrentes.

No es cierto, podemos representar concurrencia utilizando las herramienta de fusión y unión.

Solución

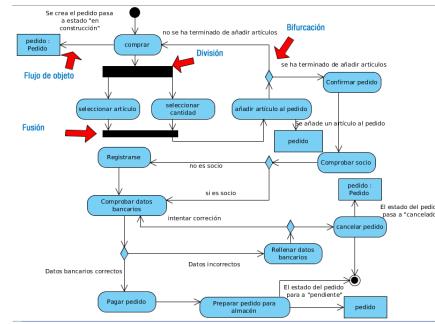
1. Opción correcta
2. Incorrecto

5.2.- Ejemplo de un diagrama de actividad.

Diagrama de actividad:

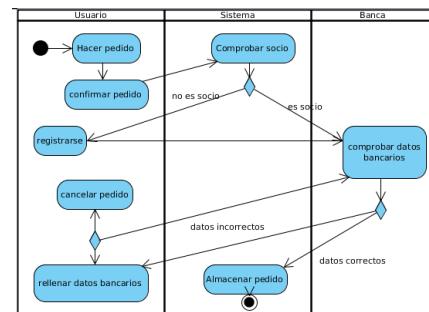
El siguiente diagrama de actividad representa el caso de uso "**Generar pedido**" del diagrama de casos de uso del ejercicio resuelto 1 "ZAPATERÍA TACÓN DE ORO" (en el punto 2 de "Los diagramas de casos de uso"), en el aparecen los elementos que hemos visto en las secciones anteriores.

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



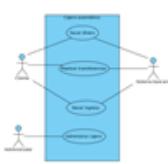
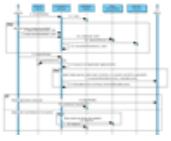
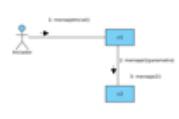
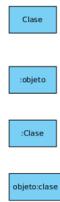
Resumen textual alternativo

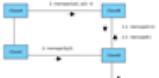
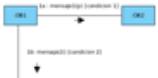
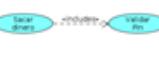
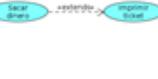
En este otro diagrama se simplifican las acciones a realizar y se eliminan los objetos para facilitar la inclusión de calles que indican quien realiza cada acción:



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

Anexo I.- Licencias de recursos.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso Educativo no comercial. Procedencia: Elaboración Propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.

	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Elena Pérez Nebreda. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Elena Pérez Nebreda. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Elena Pérez Nebreda. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: Elena Pérez Nebreda. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.		Autoría: Elena Pérez Nebreda. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.
	Autoría: María José Navascués González. Licencia: Uso educativo no comercial.		Autoría: Elena Pérez Nebreda. Licencia: Uso educativo no comercial.

 Abierta	<p>Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación. Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
	<p>Autoría: Ministerio de Educación</p>		<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial.</p>

 <p>Mensajes: Los mensajes que aparecen en este cuadro pertenecen al diseño del sistema de gestión de notas.</p>	<p>Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>	 <p>Mensajes: Los mensajes que aparecen en este cuadro corresponden al diseño de clase de datos de tipo 'Alumno' que se incluye en el sistema.</p>	<p>Procedencia: Elaboración propia.</p>
 <p>Mensajes: Los mensajes que aparecen en este cuadro pertenecen al diseño del sistema de gestión de notas.</p>	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>	 <p>Mensajes: Los mensajes que aparecen en este cuadro pertenecen al diseño de clase de datos de tipo 'Alumno' que se incluye en el sistema.</p>	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
 <p>Mensajes: Los mensajes que aparecen en este cuadro pertenecen al diseño del sistema de gestión de notas.</p>	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>	 <p>Mensajes: Los mensajes que aparecen en este cuadro pertenecen al diseño de clase de datos de tipo 'Alumno' que se incluye en el sistema.</p>	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>
 <p>Mensajes: Los mensajes que aparecen en este cuadro pertenecen al diseño del sistema de gestión de notas.</p>	<p>Autoría: Ministerio de Educación Licencia: Uso educativo no comercial. Procedencia: Elaboración propia.</p>		