Tabla de contenidos

Entorno de desarrollo integrado (IDE) NetBeans 4.1

1. Elementos que integran la interfaz del IDE Netbeans

Elementos que integran la interfaz del IDE Netbeans

*Dedicarse actualmente a la programación requiere, además de un profundo conocimiento del lenguaje elegido, dominar el entorno de programación sobre el que vamos a desarrollar las aplicaciones. En una empresa como **SI Andalucía**, que acaba de iniciar su andadura en el mundo profesional, el coste de las herramientas puede ser un importante obstáculo en sus comienzos, pero gracias a los productos de software libre han podido conseguir el IDE (Entorno de Desarrollo Integrado) NetBeans de una forma muy fácil a través de Internet. Al principio estuvieron haciendo pruebas con este entorno y **José**, que pasa por ser el más exigente en cuanto al producto final que saca la empresa, ha dado su consentimiento y le ha parecido de muy buena calidad.*
*Cuando en **SI Andalucía**, se incorpora un nuevo programador, lo que debe conocer en primer lugar y manejar con gran destreza, es precisamente NetBeans. En esta situación está **Víctor**, que en breve será uno de los programadores de la empresa y le han pasado unos documentos que debe estudiar para familiarizarse con los principales aspectos del entorno de programación: Menús, Ventanas y Barras de herramientas.*

En esta unidad pretendemos que conozcas un poco mejor el funcionamiento del entorno, para que puedas sacarle un mayor rendimiento. Debes tener en cuenta que más que una unidad que debas memorizar, se trata de que dispongas de **una guía** y un apoyo para que el tiempo de aprendizaje sea menor, y así consigas familiarizarte más rápidamente con el uso de esta herramienta. Ten en cuenta que en el mundo laboral, tendrás que trabajar con **Entornos Integrados de Desarrollo**, que serán con toda seguridad muy similares a éste. De esta forma, no dudes que si aprendes a aprovechar sus principales funcionalidades, te resultará muy útil aunque en el futuro tengas que trabajar con otro entorno.

Al igual que en otras herramientas gráficas que se utilizan para el desarrollo de aplicaciones, en el IDE de Netbeans encontramos una serie de normas comunes a todas las existentes. De esta forma encontraremos en el entorno elementos comunes a otras herramientas gráficas con la misma funcionalidad y otros que serán específicos de la funcionalidad de la propia herramienta. A continuación vamos a describir qué elementos son los que componen el IDE de Netbeans.

1.1. La barra de Menús

La barra de Menús

Al igual que en otras herramientas gráficas, **en la barra de menús encontramos los menús que engloba la herramienta**. **Algunos de estos menús serán conocidos ofreciendo las mismas funcionalidades que en otros entornos gráficos y otros serán propios de la herramienta con funcionalidades propias diseñadas para el fin al que va destinado su uso**. Recordemos que **los menús son meras agrupaciones de funcionalidades (agrupadas en comandos) de que dispone la herramienta.** La agrupación de estas funcionalidades en uno u otro menú vendrá dada por la relación existente entre ellas. De esta forma se hará más fácil el aprendizaje, la localización de funcionalidades, etc.

Los menús disponibles son:

1. Menú Archivo (**`File`**)
2. Menú Editar (**`Edit`**)
3. Menú Ver (**`View`**)
4. Menú Construir (**`Build`**)
5. Menú Ejecutar o Lanzar (**`Run`**)
6. Menú herramientas (**`Tools`**)
7. Menú **`Refactor`**
8. Menú **`Version`**
9. Menú Ventana (**`Window`**)
10. Menú Ayuda (**`Help`**)

Asociado a cada comando u opción de menú podemos encontrar los siguientes elementos:

1. **Un icono representativo de la función que realiza**. El objetivo es identificar este comando en la barra de herramientas por medio de su icono, de forma que se asocie la funcionalidad al icono, con independencia del lugar en el que se muestre dentro de la aplicación (menú, o barra de herramientas).
2. **El nombre del comando u opción**, que intenta ser lo más representativo posible de la función que desempeña.
3. **Una combinación de teclas**, con la que podemos acceder a un determinado comando u opción del menú.

El que aparezcan uno u otro o todos, depende de la forma en la que podemos acceder al comando. Por ejemplo si aparece asociado a un comando de menú un icono en su parte izquierda esto quiere decir que a este comando se puede acceder por medio de una barra de herramientas pulsando dicho icono. Por tanto para acceder a una opción concreta podemos hacerlo accediendo a su icono de la barra de herramientas correspondiente, mediante combinación de teclas o mediante su menú.

Dentro de los menús **podemos encontrar también a su vez submenús. Estos vienen representados con un nombre de submenú seguido de un icono en su parte derecha que simboliza una flecha de dirección, que nos indica que no es un comando sino el nombre de un submenú**. Posicionándonos sobre el nombre se desplegará este nuevo menú.

A continuación vamos a ver qué comandos presenta cada uno de estos menús. Para ello nos vamos a centrar en los comandos propios que ofrece la herramienta, es decir, no vamos a ver comandos comunes a otras herramientas como cortar, pegar, etc., ya que su uso y función se supone trivial y conocida.

DEMO: Visualiza las opciones del menú de NetBeans

Autoevaluación

1.1.1. Menú Archivo

Menú Archivo

Los comandos que encontramos en el menú archivo son los siguientes:

Comando	Función
(Proyecto Nuevo)	Este comando permite la creación de un nuevo proyecto. Su elección activa el asistente de la aplicación que permite mediante pasos la selección de las características del nuevo proyecto.
(Archivo nuevo)	Crea de un nuevo archivo. Éste se adjunta al proyecto que esté activo en ese momento. Al igual que el anterior la elección de este comando lanzará el asistente permitiendo seleccionar el tipo de archivo nuevo que queremos crear.
(Abrir proyecto)	Abre un proyecto existente. Al seleccionar el comando, la herramienta abrirá el explorador en el directorio de trabajo configurado.

(Abrir archivo)	Permite abrir un archivo existente.
(Abrir un proyecto reciente)	Abre un proyecto con el que se ha estado trabajando recientemente. El número de proyectos recientes que la herramienta permite seleccionar es configurable.
(Cerrar un proyecto, "Nombre del Proyecto")	Cierra el proyecto que tengamos activo en ese momento.
(Establecer proyecto principal)	Submenú. Permite establecer cuál de los proyectos de la lista es el proyecto principal.
(Propiedades del proyecto)	Nos lleva a las propiedades del proyecto activo.
(Guardar)	Guarda los cambios. Esta opción afecta al archivo que se tenga activo.
(Guardar Todo)	Igual que el anterior pero afecta a todos los archivos modificados desde su última actualización.
(Refrescar todos los archivos)	Refresca los contenidos de todos los archivos en pantalla de manera que si alguno de ellos ha sido modificado veamos sus cambios si no han sido cargados.
(Configurar página)	Permite configurar las propiedades de las páginas sobre las que se trabajan. Las páginas en la herramienta son los archivos de código.
(Imprimir)	Envía a imprimir el archivo activo.
(Imprimir a HTML)	Permite imprimir el archivo activo como una página Web. Esto permite visualizar el contenido del archivo en una página Web y poder ser representado por un navegador Web.
(Salir)	Salir de la herramienta.

Autoevaluación

1.1.2. Menu Edición

Menu Edición

En este menú encontramos los siguientes comandos:

Comando	Función
(Buscar en proyectos)	Realiza búsquedas sobre los proyectos que estén abiertos en ese momento en la herramienta. Al ejecutar el comando se abre un formulario donde podremos especificar las características de la búsqueda.
(Ir al código)	Permite desplazarnos al código fuente del elemento sobre el que está posicionado el punto de inserción.
(Ir a la declaración)	Nos lleva a la declaración del método o campo sobre el que está situado el punto de inserción.
(Ir a la línea)	Permite desplazarnos a una línea concreta. Esta línea la introduciremos en el formulario que aparece una vez seleccionado este comando.
(Ir a la clase)	Busca una clase. Cuando seleccionamos este comando aparecerá un formulario donde introducimos el nombre de la clase a buscar. También se podrá configurar algunas características sobre este proceso de búsqueda.

Autoevaluacion

1.1.3. Menú Ver

Menú Ver

La funcionalidad que ofrece este menú se refiere, como su nombre indica, a opciones de visibilidad del IDE.

Estas opciones de visibilidad abarcan a elementos tales como editores que podemos visualizar u ocultar, formas de visualizar el código, barra de herramientas, etc.

> **Con este menú podemos configurar cómo queremos que se presente el aspecto de trabajo de nuestro entorno, determinando qué elementos y qué aspectos de éstos queremos que se visualicen.**

Por ejemplo, si estamos trabajando en el entorno con el editor de código y queremos que para las líneas de código se visualice el número de orden que ocupan dentro del archivo en el que están situadas, podemos indicarlo dentro de este menú.

Al igual que pasa con muchas otras cosas dentro del entorno, estas opciones pueden seleccionarse desde fuera de este menú siguiendo otros pasos como son:

1. Mediante selección de sus funciones a través de barras de herramientas
2. Por medio del uso de **combinaciones de teclas**, como son.
   - Atajos de teclado (**`Alt + Letra subrayada en el menú`**, como por ejemplo, **`Alt + F + w`** abre la opción **`New Project`** del menú File)
   - Aceleradores de teclado (combinación de teclas escrita a la derecha de la opción en el menú, como por ejemplo **`Ctrl.+ Mayúsculas + N`** para abrir directamente la opción **`New Project`**)

Algunos de los comandos que incluye el menú **`View`** relacionados con la edición de código son los siguientes:

Comando	Función
(Navegador Web)	Abre el navegador Web que esté establecido por defecto en el equipo en que está instalada la herramienta.

(Ver números de línea)	Permite visualizar las líneas de código fuente numeradas dentro del archivo. La numeración de éstas aparece en la barra lateral izquierda del editor de código.
(Ver barras de herramientas del editor)	Permite visualizar u ocultar la barra de herramientas de la ventana editor de código.

Autoevaluación

Los submenús que encontramos dentro del menú ver son:

`Editors` (Editores): **permite acceder a los diferentes editores con los que podemos trabajar en la herramienta, como por ejemplo el editor de código fuente o el editor de diseño (editor gráfico o diseñador con el que trabajaremos a través de objetos gráficos y no líneas de código, llamado `Form Editor` en NetBeans)**.

Hay que tener en cuenta que los editores sobre los que podemos trabajar en un momento determinado, dependerán de nuestro contexto de trabajo actual. Así, el editor gráfico, por ejemplo, no estará disponible si en la aplicación en la que se trabaja no existen hasta ese momento elementos gráficos creados.

La siguiente imagen muestra el aspecto del Editor de Código.

A continuación podemos ver el aspecto del Editor Gráfico.

Autoevaluación

`Code Folds` (Plegar código): Permite configurar diferentes aspectos sobre como se presentará el contenido de los archivos de código fuente. Antes de seguir con este punto debemos aclarar que el texto que aparece en el editor de código, se puede agrupar en código fuente (líneas de código creadas por el usuario basándose en un lenguaje de programación, en este caso lenguaje Java) o comentarios (texto asociado al lenguaje Java que comenta características del mismo). Tanto uno como otro se encuentra agrupado en bloques. Los bloques viene determinados por:

1. El inicio o fin de un método o clase Java
2. El inicio o fin de un comentario.
3. El inicio o fin para cada uno de los casos se establece de manera diferente.

Las opciones que encontramos dentro de este submenú son las siguientes:

Comando	Función
	Permite plegar el bloque de código o comentario sobre el que esté posicionado el punto de inserción.
	Permite desplegar el bloque de código o comentario sobre el que este posicionado el punto de inserción.
	Comprime todos los bloques de código y comentarios a la vez.
	Despliega todos los bloques de código y comentarios a la vez.
	Comprime todos los **`Javadoc`** del documento.
	Expande todos los **`Javadoc`** del documento.
	Comprime todo el código Java del documento.
	Expande todo el código Java del documento.

Autoevaluación

`Toolbars` (barra de herramientas): Permite acceder a las barras de herramientas que posee la herramienta con el fin de ponerlas ocultas o visibles según se necesite.

Las barras de herramientas que presenten una marca en su lado izquierdo indican que están visibles. Cuando lleguemos al punto de este capítulo donde se habla de éstas se verá las formas de visualizarlas u ocultarlas que existen.

1.1.4. Menú Build

Menú Build

El menú construir (**`Build`**) va a presentar comandos o funciones muy utilizadas en la herramienta. Para entender este menú vamos a plantearnos las siguientes cuestiones:

¿Que significa construir un proyecto?

Construir un proyecto significa que el IDE empaqueta todos los elementos de los que hace uso para una aplicación concreta (las clases que pertenecen a nuestro proyecto, clases que se usen de otros proyectos, librerías que usa, etc.) y genera los ficheros de salida pertinentes, normalmente con extensión .jar o .war. Antes de realizar este empaquetado (agrupación) el IDE compila los archivos fuente generados en nuestro proyecto para verificar que no existen errores de compilación.

Los pasos que se realizan durante este proceso se pueden ir observando en la ventana de salida del IDE (Ventana Output). Si durante el proceso se detectaran errores de compilación también se observarían en esta ventana, cuyo aspecto puedes ver en la imagen siguiente.

Las opciones que encontramos en el **menú `Build`** son las que se comentan a continuación:

Comando	Función
(Construir el proyecto principal)	El IDE compila el archivo. Si hemos seleccionado una carpeta el IDE compila todos los archivos de esa carpeta sin tener en cuenta los cambios producidos y que no se han guardado desde la última compilación.
(Limpiar y construir el proyecto principal)	Realiza el mismo proceso que la opción anterior limpiando los ficheros a los que afecta el proceso.
(Generar los Javadoc para el proyecto "nombre proyecto")	Genera los Javadoc para el proyecto sobre el que estemos en este momento trabajando. Este proyecto es al que hace referencia "nombre proyecto"
(Compilar "nombre.java")	Compila el fichero que contiene la clase sobre la que estamos actualmente trabajando.
(Parar la construcción)	Para la construcción del proyecto. Solamente accesible durante el proceso de construcción.

(Próximo Error)	Cuando se ha terminado el proceso de construcción puede o no existir errores en los archivos de código fuente. En el caso de existir errores, este comando se posiciona en el próximo error de la lista.
(Error anterior)	Cuando se ha terminado el proceso de construcción puede o no existir errores en los archivos de código fuente. En el caso de existir errores, este comando se posiciona en el error anterior de la lista.

Autoevaluación

1.1.5. Menú Run

Menú Run

Este menú agrupa **las acciones que podemos realizar tanto antes como durante la ejecución de nuestro programa escrito en lenguaje Java**.

Por tanto, al igual que sucede con las acciones disponibles por el menú **`Build`** (construir), serán muy utilizadas e importantes.

Algunas de las opciones comentadas en este punto están estrechamente ligadas con el punto **Herramientas de depuración de programas**, que se explicarán más adelante en esta misma unidad.

A continuación se describen cuáles son estas acciones que encontramos en este menú:

Comando	Función
(Ejecutar programa principal)	Lanza la ejecución del proyecto a partir de su clase principal. Recuerda que la clase principal de un proyecto es aquella que contiene el método **main().** Para que la ejecución pueda ser lanzada el código del proyecto tiene que estar libre de errores de compilación.
(Depurar Programa principal)	Al igual que la anterior lanza la ejecución del proyecto. La diferencia que encontramos es que ahora esta ejecución se podrá parar en el punto del código que establezcamos. Esto es útil para depurar errores de ejecución, es decir, errores que no son de compilación.
(Terminar sesión de depuración)	Cuando se está produciendo la depuración del proyecto, esta opción termina con dicho proceso.
(Pausar)	Para el proceso de depuración del proyecto. Posteriormente este proceso puede continuarse a partir del punto en el que se paró.
(Continuar)	Continúa con el proyecto de depuración. Disponible cuando se comenzó y por algún motivo este proceso se paró.
	Ejecuta línea a línea el código. Si la línea del código se refiere a un método éste es ejecutado entero como si de una única línea se tratase.
	Ejecuta línea a línea el código. Si la línea de código se refiere a un método, las líneas de código que lo componen son ejecutadas una a una.
	Ejecuta una línea de código. Si la línea de código es una llamada a un método lo ejecuta entero y después se para la ejecución.
(Ejecutar hasta el cursor)	Ejecuta el código hasta llegar a la posición del cursor.
(Aplicar cambios al código)	Aplica los cambios que se han realizado en el código, es decir, los guarda.
(Borrar punto de ruptura)	Elimina el punto de ruptura sobre el que está situado el punto de inserción.
(Nuevo punto de ruptura)	Inserta un nuevo punto de ruptura en la línea que está situado el puntero de inserción.
(Nueva variable a observar)	Añade una nueva variable a la ventana de variables para poder evaluar su contenido.

Autoevaluación

1.1.6. Menú Window

Menú Window

Dentro del menú **`Window`** (menú Ventana) **encontramos las diferentes ventanas con las que podemos trabajar en la herramienta**. También encontramos submenús que agrupan ventanas que utilizaremos durante diferentes procesos como el proceso de depuración del programa.

> **La utilidad de este menú será la de visualizar en un momento determinado aquellas ventanas que resulten necesarias dependiendo de nuestra situación de trabajo.**

De esta forma si en algún momento alguna ventana no se encuentra disponible, solamente tendremos que acceder a este menú y activarla seleccionando la misma.

A continuación se muestra una tabla donde se describen las ventanas más importantes y utilizadas de la herramienta.

Ventanas	Función
(Proyectos)	La ventana proyectos será el punto de entrada para las fuentes que forman parte de nuestro proyecto. En esta ventana se mostrarán los paquetes, clases, métodos y demás elementos que forman los distintos proyectos abiertos en el IDE. Pulsando sobre alguno de estos elementos con el botón derecho del ratón tendremos acceso a su menú contextual.

(Archivos)	La ventana archivos muestra una vista del directorio en el que se encuentra el proyecto activo. En esta ventana se mostrarán también los archivos y carpetas para el proyecto que no se incluyen en la ventana **`Projects`**. Desde esta ventana se pueden abrir los archivos del proyecto, para corregirse o modificarse.
(Favoritos)	Desde la ventana "**`Favorites`**", es posible tener acceso a cualquier carpeta de nuestro ordenador.
(Salida)	Ventana de salida. Esta ventana es donde el IDE Mostrará sus mensajes. **Aparece de forma automática** siempre que realizamos el proceso de compilación, generación de los **`Javadocs`**, etc.
(Navegador)	Esta ventana muestra información acerca de los archivos fuente Java del proyecto abierto actualmente en la herramienta. Seleccionando uno u otro archivo la ventana navegador mostrará la información sobre dicho archivo.
(Propiedades)	Esta ventana muestra las propiedades para el elemento seleccionado. Los elementos pueden ser objetos visuales (como un botón, campo de texto, etc.), archivos fuentes, etc. Desde esta ventana podremos cambiar o establecer dichas propiedades
(En ejecución)	Esta ventana muestra los recursos de los que disponemos en tiempo de ejecución que están registrados en el IDE. Algunos de estos recursos son bases de datos, servicios Web, etc.

Desde este menú no podemos visualizar la ventana que muestra los **editores de la herramienta**. Recuerda que podemos trabajar con dos editores:

- el de código fuente y
- el de diseño.

La ventana que contiene los editores (según corresponda el de diseño o código fuente) se mostrará automáticamente cuando carguemos en la herramienta algún archivo que los use. Así por ejemplo, si creamos o cargamos una clase Java, se abrirá automáticamente el editor de código de fuente conteniendo dicha clase. La siguiente imagen muestra el editor de código fuente que contiene el código para la clase **`AltaPelicula`**.

Una vez que la ventana del editor esté activa, podremos cambiar entre uno u otro desde la barra de herramientas de dicha ventana o a través del menú `view` + `editors` + `editor deseado` (`Source` o `Design`)

1.2. La barra de herramientas

La barra de herramientas

La barra de herramientas que encontramos en el entorno, al igual que pasa con los menús, se asemeja a otras herramientas que manejamos (como procesadores de texto, hojas de cálculo, navegadores, etc.) en que **ponen a nuestra disposición las acciones más comunes o usadas de la herramienta**. De la misma manera que los anteriores, en NeatBeans encontramos diferentes barras de herramientas las cuales **pueden estar visibles u ocultas**.

¿Porqué podemos ocultarlas o ponerlas visibles y no están establecidas de manera fija?

La respuesta es bien sencilla: dependiendo de las barras de herramientas que tengamos visibles en nuestro entorno dispondremos de **menos espacio útil** para trabajar, por esto NetBeans permite esta posibilidad. De esta forma nosotros podremos establecer cuáles estarán visibles dependiendo de su necesidad.

DEMO: Mira las opciones del menú de herramientas de NetBeans

Para poner una barra de herramientas visible u oculta podremos hacerlo de una de las dos formas siguientes:

1. **Desde el menú `view` + `toolbars` + `barra de herramienta elegida` para ocultar o poner visible**.
2. **Pulsando con el botón derecho sobre el espacio que ocupan las barras de herramientas en la ventana de NetBeans. Del menú resultante elegimos la que queremos poner visible u oculta**.

Además de ocultar o mostrar las barras de herramientas también podemos establecer la forma en que aparecerán los iconos que acompañan a los comandos, de forma reducida o normal.

Ahora que conocemos para qué sirven y cómo manejar las barras de herramientas, vamos a comentar **cuáles son las que encontramos en el IDE y qué funcionalidades ofrecen**. Las barras de herramientas que presenta el IDE de NetBeans son:

1. **`Build`** (Construir)
2. **`Debug`** (Depurar)
3. **`Edit`** (Editar)
4. **`File`** (Archivo)
5. **`Memory`** (Memoria)
6. **`Versioning`**

Las acciones representadas por iconos a las que podemos tener acceso desde las barras de herramientas las vamos a encontrar dentro de los menús en forma de comandos, es decir, a una misma función podemos tener acceso desde varios sitios. Recuerda que para cada acción que podemos realizar en el entorno podemos encontrar un icono, nombre y su abreviación de teclas.

Autoevaluación

1.3. Principales Ventanas del IDE

Principales Ventanas del IDE

Las ventanas de NetBeans nos van a mostrar contenidos importantes agrupados por temas. Así, para un mismo proyecto, podremos acceder a través de ventanas a:

- Sus archivos,
- Sus clases,
- Sus variables,
- Su código,
- Ver los mensajes que el IDE nos comunica cuando realicemos alguna operación,
- etc.

Las ventanas de las que dispone la aplicación al igual que sucede con las barras de herramientas, pueden estar **visibles u ocultas.** También podemos encontrarlas

desplegadas sobre el área de trabajo o bien minimizadas como botones sobre una barra del IDE.

Las ventanas más comunes y utilizadas dentro del IDE las describimos cuando vimos el menú Window. A continuación vamos a **repasar** algunas de ellas. En puntos sucesivos aparecerán ventanas nuevas que no han sido aún comentadas.

1.3.1. Elementos y operaciones comunes de las ventanas del IDE

Elementos y operaciones comunes de las ventanas del IDE.

Lo que se pretende en este punto es describir qué **funciones** podemos realizar con cualquier ventana de IDE de NetBeans:

Observando esta figura observamos que sobre la barra de título de la ventana encontramos los siguientes elementos distribuidos como se enumeran a continuación:

1. Una **zona** a partir de la que podremos ubicar la ventana donde queramos. Para realizar esta operación nos situaremos con el ratón sobre dicha zona pulsaremos el botón izquierdo y sin soltar desplazamos la ventana hasta la zona deseada.
2. El **nombre** de la ventana. Este nombre como es de suponer indica de qué ventana se trata.
3. Un **icono** para minimizar, restaurar la ventana.
4. Otro con el que cerraremos la ventana.

Algunas de las ventanas que podemos encontrar en el IDE de NetBeans son las siguientes:

- La ventana de proyectos
- La ventana de salida
- La ventana en tiempo de ejecución
- La ventana de edición

1.3.2. La ventana de proyectos (Projects)

La ventana de proyectos (Projects)

En la ventana de proyectos vamos a poder ver el proyecto o proyectos sobre los que estamos trabajando. También se mostrarán en esta ventana los elementos que forman parte de cada proyecto. Para acceder a esta ventana podemos realizar los siguientes pasos:

1. **Desde el menú** `view + projects`.
2. **con la combinación de teclas** `CTRL+1`.
3. **pulsando sobre su pestaña dentro del entorno**.

Esta ventana será el equivalente al explorador de nuestro sistema operativo, pero en este caso, en vez de navegar por unidades de disco, directorios, etc. navegaremos sobre paquetes, clases y otros elementos propios de la programación.

Algunos iconos que podemos ver en esta ventana se comentan a continuación en una tabla junto a su descripción.

Icono	Descripción
	Proyecto estándar de Java
	Grupo de Paquetes del proyecto
	Paquete Java
	Clase del proyecto
	Paquete Java privado
	Paquete Java público
	Clase principal o ejecutable
	Paquete vacío Java

1.3.3. La ventana de salida (Output)

La ventana de salida (Output)

La ventana de salida del IDE (`Output`), será utilizada para mostrar los mensajes que genere el entorno cuando realice alguna operación como compilación, generación de Javadocs, Etc. Por tanto ésta será la vía de comunicación que tendrá el IDE para indicarnos sus mensajes referidos a la operación que está realizando. En la imagen podemos ver la ventana `Output` del IDE realizando el proceso de debug de una aplicación.

1.3.4. La ventana en tiempo de ejecución (Runtime)

La ventana en tiempo de ejecución (Runtime)

Como hemos indicado anteriormente, la ventana `Runtime` mostrará los **recursos** a los que podemos acceder en tiempo de ejecución desde el IDE. Para que estos recursos se muestren tienen que estar previamente integrados en el mismo. Algunos de estos recursos son los que se muestran en la siguiente tabla en la que se mostrará su representación gráfica y una descripción del mismo.

Representación gráfica del recurso	Descripción
	Lista de los servidores que actualmente están registrados en el IDE.
	Lista de procesos que actualmente se están ejecutando en el IDE. **Desde aquí podemos terminar la ejecución una aplicación o programa pulsando sobre él con el botón derecho del ratón y seleccionando** `Terminate Process`.
	Lista de los controladores de las bases de datos que han sido añadidas al IDE.
	Lista de los servicios Web integrados en el IDE. Desde aquí podemos chequear dichos servicios a través del botón derecho del ratón.

1.3.5. La ventana de edición

La ventana de edición

En esta ventana va a ser donde se situarán los editores disponibles en la herramienta (el de código y diseño).

El editor de código será el que nos permitirá, entre otras cosas, escribir y cambiar los archivos fuente de nuestro proyecto.

Hay que tener claro que cuando nos referimos a un arhivo fuente, nos referimos en realidad a una clase Java. En el editor de código podremos:

1. **Escribir una nueva clase Java**. Este proceso comenzaría una vez que la hayamos creado desde el menú Archivo. Cuando nos referimos a escribir en realidad lo que queremos decir es generar todo el código (métodos, funciones, implementación de algoritmos, etc)
2. **Modificar una clase Java existente**.
3. En el caso de haber varias clases cargadas, **cambiar de una a otra pulsando sobre su pestaña correspondiente**.

La siguiente imagen muestra el aspecto del editor de código.

El editor de diseño lo utilizaremos cuando incorporemos elementos que proporcionan las librerías gráficas del entorno, como AWT y Swing, a nuestro proyecto para poder crear interfaces gráficas de usuario. Para ello primeramente tendremos que tener creado un formulario sobre el que depositaremos estos elementos.

La siguiente imagen muestra el aspecto del editor de diseño, o editor gráfico, o sencillamente diseñador. (`Form Editor`)

2. Herramientas de depuración de programas

Herramientas de depuración de programas

*Conocer las herramientas de NetBeans que aparecen a simple vista, no es difícil, sólo hay que probarlas con algunos de los ejemplos que le han pasado sus compañeros. Pero **Víctor** necesita ayuda, básicamente porque nunca se ha dedicado a la programación como parte integrante de un equipo profesional en una empresa, donde, sencillamente no se pueden hacer muchos experimentos, sino más bien ir a lo práctico y terminar en los plazos indicados los pedidos de los clientes. Evidentemente para conocer a fondo el IDE necesitó la ayuda de **Carmen**, que es quien realmente le enseñó a utilizar las herramientas de compilación y depuración de las aplicaciones para localizar errores. Pero lo que más le ha gustado a **Víctor** ha sido la ejecución paso a paso del programa con el fin de detectar funcionamientos incoherentes. Esto le ocurrió en una ocasión en la que cometió el fallo de utilizar una condición inadecuada para controlar un bucle, mediante la que se ejecutaba una vez más de lo debido y era prácticamente imposible localizar el error, porque por mucho que miraba el código, éste parecía correcto. **Carmen** le enseñó a utilizar estas técnicas que utilizan ventanas características, puntos de ruptura, evaluación de variables, modos de ejecución...*

Cuando hemos escrito un programa tenemos que compilarlo y depurar los posibles errores que nos indique el compilador antes de poder ejecutarlo, hasta que los hayamos corregido todos. ¿Quiere decir eso que nuestra aplicación ya está libre de errores?

Desafortunadamente no. Pueden existir errores en tiempo de ejecución, que generen un malfuncionamiento de la aplicación, que hagan que ésta aborte, o sencillamente que no haga lo que esperábamos que hiciera.

Las herramientas de depuración de las que dispone el IDE de NetBeans van a permitir la depuración de errores de ejecución, es decir, errores que no hacen referencia a las reglas de escritura del código y que están ligadas al lenguaje que se utiliza.

Un **ejemplo** de error de ejecución sería:

Supongamos que queremos sumar el contenido de dos variables: A + B. Si damos los siguientes valores para las variables A=3 y B=2 y obtenemos como resultado 1 evidentemente esto no funciona correctamente. Para detectar la causa de ese mal funcionamiento una buena solución sería poder situarse encima de la expresión y poder evaluar qué valores tienen las variables antes de ser ejecutada dicha expresión. De esta manera podemos asegurar que los valores que contenían eran 3, 2 y no otros valores que nos den como resultado 1. También podemos ver si la expresión es A + B o A - B. Esto sería un error aunque el compilador no lo detecta como tal ya que la expresión está bien construida aunque no haga en principio lo que se deseaba. Todo este tipo de errores son los que vamos a detectar y corregir utilizando las herramientas y consejos que se dan en este apartado.

Una sesión de depuración de errores puede comenzarse de las siguientes formas:

1. **Desde el menú `Run + Debug Main Project`.**
2. **Pulsando la tecla `F5` que corresponde a la ejecución de dicho comando.**
3. **Pulsando el botón `Debug Main Project` en la barra de herramientas ()**

También podemos iniciar una sesión para un determinado archivo fuente y no para el conjunto de archivos del programa. Para realizar esto lo primero que debemos hacer es, desde la ventana projects, seleccionar el archivo o clase que queremos depurar. Una vez realizado esto:

1. Podemos acceder de nuevo al menú **`Run + Debug`** nombre de la clase.
2. Pulsando **`CTRL + Mayusculas + F5`** que corresponde con el comando anterior.
3. A estas opciones también podemos acceder pulsando sobre el nombre de la clase con el botón derecho del ratón y del menú contextual que aparece seleccionando el comando **`Debug File`**.

Cuando comenzamos una sesión de eliminación de errores (Depuración, Debug), todas las ventanas que intervienen en este proceso aparecen automáticamente en la pantalla (**ventana `Output`, ventana `Debugger console`, etc.**). Sobre estas ventanas podremos entre otras cosas ir evaluando el proceso de la sesión de depuración, ver errores que detecta el compilador, etc.

Existen **otras ventanas** muy utilizadas durante el proceso de depuración que no aparecen de manera automática de modo que tendremos que visualizarlas y ocultarlas cuando se necesiten. Estas ventanas son las que se comentan en el punto siguiente.

La imagen siguiente muestra la ventana output, con algunas pestañas del proceso de depuración abiertas.

2.1. Ventanas que podemos usar durante el proceso de depuración

Ventanas que podemos usar durante el proceso de depuración

En este punto **vamos a ver:**

- **Las diferentes ventanas que podemos usar en el proceso de depuración**
- **Cómo podemos cambiar de unas a otras**
- **Cómo podemos visualizarlas mediante una combinación de teclas**.

Las **ventanas de la depuración** se abren automáticamente siempre que comencemos una sesión de depuración para eliminar errores y se cierran cuando terminamos dicha sesión. Pero existen otras ventanas que el IDE no abre de manera automática a menos que en sesiones anteriores se lo hayamos indicado, de la misma forma tenemos que indicar nosotros que no aparezcan en futuras sesiones de depuración si así lo queremos. Estas ventanas van a ser muy útiles dependiendo de nuestra situación de trabajo. Existen ventanas donde podremos ver los puntos de parada de ejecución, deshabilitarlos, volverlos a habilitar, ventanas donde veremos el contenido de variables de nuestro programa etc.

Estas ventanas son las que se describen en la siguiente tabla. Para ello indicamos su nombre, combinación de teclas con las que se visualizan y una descripción de su función.

Nombre de la ventana	Combinación de teclas	Descripción
Local variables (Variables Locales)	`Alt + Shift +1`	**Enumera las variables locales que están dentro de la llamada actual, es decir, muestra las variables del método que se está ejecutando actualmente.**
Watches (Vigilancia)	`Alt + Shift + 2`	**Enumera todas las variables y las expresiones que hemos elegido para evaluar su contenido durante el proceso de ejecución actual. Dependiendo de su situación y por donde se vaya ejecutando la aplicación estarán visibles o no.**
Call Stack (Pila de llamadas)	`Alt + Shift + 3`	**Enumera la secuencia de las llamadas hechas durante la ejecución del proceso actual.**
Classes (Clases)	`Alt + Shift + 4`	**Exhibe la jerarquía de todas las clases que han sido cargadas por el proceso que se está depurando actualmente.**
Breakpoints (puntos de ruptura de ejecución)	`Alt + Shift +5`	**Enumera los puntos que establecemos para que la ejecución de nuestra aplicación se detenga.**
Sessions (Sesiones)	`Alt + Shift + 6`	**Enumera las sesiones de depuración que se están realizando en el IDE. Desde esta ventana se puede parar una sesión de depuración.**
Threads (Procesos o hilos en ejecución)	`Alt + Shift + 7`	**Enumera los grupos de procesos en ejecución para el programa en ejecución actual.**
Source (Fuentes)	`Alt + Shift + 8`	**Enumera los directorios de fuente en su classpath del proyecto.**

DEMO: Visualiza cómo se utilizan los puntos de ruptura para depurar

Para ocultar o establecer que en próximas sesiones algunas de estas ventanas no estén disponibles, basta con pulsar sobre la **X** que acompaña a su nombre en su lado derecho con el botón izquierdo del ratón.

La siguiente imagen muestra la ventana que contiene variables locales.

Autoevaluación

2.2. Estableciendo puntos de parada de ejecución o Breakpoints

Estableciendo puntos de parada de ejecución o Breakpoints

Es muy posible (por no decir seguro) que dependiendo de las características de nuestro programa necesitemos o bien queramos que la ejecución del mismo se **detenga** en un punto concreto.

Para establecer un punto de parada de ejecución vamos a utilizar lo que se denomina Breakpoints. Con ellos podremos detener la ejecución de nuestro programa en el punto donde los establezcamos. Ésta será una herramienta muy utilizada cuando queramos por ejemplo evaluar expresiones, contenidos de variables en un momento determinado de la ejecución etc. **Para establecer puntos de parada o Breakpoints en nuestro código realizamos los siguientes pasos**:

1. **Tenemos que poner visible en el editor de código la clase sobre la que queremos establecer un punto de parada o Breakpoint**.
2. Una vez hecho esto **localizamos en el código el punto exacto donde queremos que la ejecución se detenga y nos posicionamos sobre dicha línea**.
3. **Ahora podemos crear un nuevo punto de parada o Breakpoint siguiendo uno de estos métodos**:
    1. **Pulsamos a la altura de dicha línea y sobre la barra izquierda del editor de código el botón izquierdo del ratón**.
    2. **Accedemos al menú Run + comando new Breakpoint**
    3. De esta forma donde se ha establecido el punto de parada debe aparecer un **cuadrado de color rosa** sobre la barra izquierda del editor de código y la línea donde se ha establecido también aparece de color rosa o salmón, tal y como muestra la imagen siguiente.

Ahora si lanzamos de nuevo la ejecución de nuestro programa, cuando ésta pase sobre este punto (donde existe un punto de parada o breakpoint) se detendrá. Veamos una simulación de cómo se inserta un punto de ruptura.

DEMO: Mira cómo añadir puntos de parada de ejecución a un programa

Para eliminar puntos de parada existentes realizamos los siguientes pasos:

1. En la ventana del editor de código, localizamos el punto de parada o `Breakpoints` que queremos eliminar. Pulsamos sobre el botón izquierdo del ratón.
2. Desde la ventana `Breakpoints` lo desactivamos pulsando sobre su recuadro.

La diferencia de hacerlo desde el editor de código o desde la ventana Breakpoints es que desde el editor de código se elimina el punto de parada y desde la ventana lo que hacemos es deshabilitarlo pero no eliminarlo, de esta forma podemos volver a habilitarlo en futuras sesiones de depuración.

También desde la ventana de Breakpoints y pulsando con el botón derecho del ratón podemos mediante el menú contextual acceder a otras operaciones sobre Breakpoints, tal y como se aprecia en la imagen que acompaña al texto. Esas operaciones sobre Breakpoints son:

1. `Enable` (Habilitarlo).
2. `Disable` (Deshabilitarlo)
3. `New Breakpoint...` (Añadir uno nuevo)
4. `Delete` (Borrarlo)

5. **`Enable All`** (Habilitarlos todos)
6. **`Delete All`** (Borrarlos todos)
7. **`Disable All`** (Deshabilitarlos todos)

Autoevaluación

2.3. Evaluando variables de nuestro programa

Evaluando variables de nuestro programa

Otra operación muy común que realizaremos durante una sesión de depuración o eliminación de errores será **evaluar el contenido de variables que utilicemos en el mismo**.

Esto es así debido a que cuando escribimos un programa es prácticamente imposible (debido a su complejidad) asegurar que todas las operaciones que éste debe realizar estén indicadas de forma correcta.

Pensemos en el **ejemplo** de la suma de variables A y B (un ejemplo muy sencillo considerando a las situaciones que todo programador se tendrá que enfrentar) cuyo resultado no coincide con lo que se esperaba. ¿Por qué puede suceder esto? Por muchos factores, bien porque llevamos muchas horas trabajando y escribimos cosas que no pensamos correctamente, bien porque cuando se diseñó la solución no fue la adecuada, o un sin fin de causas normales en el mundo de la programación.

Para ello una herramienta muy útil será poder **evaluar** en determinados momentos qué hace nuestro código y verificar el valor de las variables que trata. Esto se puede hacer mediante el uso de dos ventanas comentadas anteriormente, la ventana variables locales (**`Local Variables`**) y la ventana de evaluación de variables o expresiones (**`Watches`**).

La diferencia que existen entre ambas es que la **primera (variables locales)**, hace referencia a las variables que están definidas dentro del método que se está ejecutando y que por tanto es visible. El contenido de esta ventana va variando dependiendo de la situación de ejecución. De esta forma, cuando estemos en un método determinado, mostrará las variables de dicho método (variables locales) y su valor conforme lo vayan tomando. Cuando se termine de ejecutar este método y se salga de él, esta ventana dejará de mostrarlas. Por tanto esta ventana la vamos a utilizar a la hora de ver o evaluar las variables declaradas para un método determinado.

Otra ventana de las comentadas es **Watches**. Su función es similar a la anterior pero permite que nosotros añadamos (lo tenemos que hacer ya que si no, no aparece) **qué variables o expresiones son las que queremos evaluar** o ver. Al igual que la anterior estas variables estarán disponibles siempre y cuando estén visibles, la **visibilidad** de una variable va a depender de su declaración. Por ejemplo una variable declarada dentro de un método estará visible a la herramienta cuando estemos dentro de ese método, si por el contrario está declarada como variable de una clase ésta será visible mientras exista esa clase.

Una de las operaciones que podemos realizar, además de la de evaluar el contenido de una variable o expresión, es la de transformar su valor en tiempo de ejecución.

Para ello solamente deberemos pulsar sobre el botón de puntos suspensivos que aparece a la derecha del valor y en la ventana resultante, cambiarlo. Esto es útil si queremos que la variable en cuestión tenga un valor determinado en el momento de su evaluación y comprobar que con esto se cumple lo que esperamos.

¿Cómo podemos añadir variables a la ventana Watches para su evaluación? Siguiendo este procedimiento:

1. Accedemos al menú **`Run + comando New Watch`**.
2. Pulsando sobre el editor de código con el botón derecho del ratón y del menú contextual resultante seleccionamos comando **`New Watch`**.
3. En ambos casos después habrá que escribir el nombre de la variable o expresión a evaluar. Una vez hecho esto la variable o expresión aparece en la ventana **`Watches`** .
4. Otra forma sería primero seleccionar en el editor de código la variable o expresión y después acceder al comando **`New Watch`** de una de las dos formas vistas. La diferencia ahora radica en que no tendremos que introducir mediante teclado el nombre ya que lo toma de la selección.

¿Cómo podemos **eliminar variables** fijadas en la ventana **`Watches`**? Siguiendo este procedimiento:

1. Seleccionamos la variable a eliminar dentro de la ventana
2. Pulsamos con el botón derecho del ratón y del menú contextual resultante seleccionamos la opción eliminar.

Al igual que pasaba con la ventana Breakpoints, este menú contextual permite otras operaciones adicionales como:

1. **`New Watch`** (Añadir una nueva variable a la ventana)
2. **`Delete`** (Borrarla)
3. **`Delete All`** (Borrar todas las variables)

DEMO: Aquí puedes ver cómo añadir variables a la ventana Watches

Autoevaluación

2.4. Formas de ejecución del código en modo depuración

Formas de ejecución del código en modo depuración

A continuación vamos a ver cómo podemos hacer que se ejecute el código de nuestro programa durante una sesión de depuración o eliminación de errores. Estas formas las podremos seleccionar mediante la pulsación de las siguientes teclas:

1. **Ejecución de bloques (F8).** Ejecuta una línea fuente. Si la línea fuente contiene una llamada, ejecuta la rutina entera sin entrar dentro de ella.
2. **Ejecución línea a línea (F7).** Ejecuta una línea fuente. Si la línea fuente contiene una llamada, la ejecución entrará en dicha llamada y ejecutará una a una las líneas que pertenecen a la misma, para ello deberemos ir pulsando F7 para cada línea.
3. **Paso Hacia fuera (Alt-Shift-F7).** Ejecuta una línea fuente. Si la línea fuente es parte de una rutina, ejecuta las líneas restantes de la rutina y devuelve control al llamador de la rutina.
4. **Pausa.** Se detiene brevemente la ejecución.
5. **Continuar (Ctrl-F5).** Continúa la ejecución. La ejecución se detendrá (en caso de existir) en el siguiente punto de ruptura o breakpoint.

Recuerda que las acciones que realizan estas teclas las podemos encontrar en forma de comandos dentro del menú Run del IDE.

Una simulación en la que se salta la ejecución de un método.

DEMO: Visualiza cómo saltar la ejecución de un método al depurar

Una simulación en la que se entra en un método para depurar su código.

DEMO: Y aquí, mira cómo continúa la depuración dentro de un método

Una simulación en la que se evalúan todas las variables locales de los métodos por los que pasa la depuración del programa.

DEMO: Para finalizar, mira cómo se evalúan las variables de los métodos al depurar

3. Javadocs

Javadocs

__Víctor__ va entregando los módulos de programa terminados a __José__ para que le vaya dando el visto bueno. Quiere hacer un buen trabajo y los ha revisado concienzudamente, incluso les ha añadido algunos comentarios, lo que no es muy habitual en él. Todo esto le ha llevado más tiempo del que esperaba, pero se siente satisfecho y espera que __José__ le felicite por su trabajo.
Pero esto no ocurre. __José__ se muestra indiferente ante el trabajo de __Víctor__, le dice que está bien y le dice que debe incorporar la documentación. Ante esta situación __Víctor__ se siente decepcionado y a la vez molesto por tener que dedicar ahora más tiempo a la documentación. Pero __José__, en tono conciliador, le explica que esa documentación la genera automáticamente Java, aprovechando los comentarios que él ha ido incorporando al código. Ante esto, __Víctor__ se alegra de no tener que hacer un trabajo extra y porque el resultado obtenido es espectacular. Aunque sí que tiene que hacer algunas pequeñas modificaciones en la sintaxis de esos comentarios.

JavaDoc es una herramienta integrada en el SDK de Java, que permite documentar de manera rápida y sencilla las clases y los métodos que componen los programas Java. Dicha herramienta genera de forma semiautomática un grupo de archivos HTML en base a una sintaxis de comentarios insertados en el código fuente.

De esta forma podremos indicar información importante para otros programadores y nosotros mismos sobre el código de que se componen los programas que creamos, ya que con el tiempo olvidaremos qué hacía un método, una clase, etc. Y la documentación que acompaña a dicho código será muy útil para recordarlo o bien para conocerlo según sea el caso.

Al generar el JavaDoc documentos HTML, podemos utilizar las etiquetas de dicho lenguaje dentro de los comentarios ya que la herramienta lo permite interpretando éstas como etiquetas HTML y no como simple texto descriptivo. Vamos a proceder a continuación a tratar diferentes aspectos importantes sobre los Javadocs.

DEMO: Visualiza cómo se genera la documentación del programa

3.1. Sintaxis de los comentarios

Sintaxis de los comentarios

Los comentarios que formarán parte del código se mostrarán en forma de marca de agua para distinguir a simple vista qué forma parte de un comentario y qué forma parte del código Java propiamente dicho. Los comentarios para su creación utilizan o pueden utilizar (dependiendo del caso) los siguientes elementos de sintaxis:

- **Delimitadores /** , */ :** los símbolos indican el comienzo y final de un comentario. Para entenderlo mejor observamos el siguiente ejemplo de la imagen de un comentario Javadoc, en el que se observan los delimitadores de inicio y final del comentario :

En este ejemplo se pueden observar los comentarios asociados al método `AltaPelicula`. Estos comentarios indican qué realiza dicho método. Observamos que dicho comentario tiene un inicio y un final establecido por los delimitadores antes comentados. El resultado final, una vez generados los Javadocs del proyecto es el que se observa en la siguiente figura:

Visualización del comentario en su documento HTML generado por la herramienta Javadoc

- **Código HTML**: como hemos comentado los comentarios pueden usar etiquetas HTML, siendo éstas interpretadas como tal. Supongamos que en un comentario queremos introducir un enlace donde indicamos información adicional para el método que acompaña al usuario que lee dicha documentación. Esto sería como muestra el siguiente ejemplo:

En este ejemplo incorporamos etiquetas HTML para insertar un enlace a una página Web (con una dirección hipotética) dentro de dicho comentario.

- **Tags (etiquetas) comenzando con @:** Las etiquetas que utilizamos en los comentarios son palabras reservadas que están destinadas para la creación de los mismos, es decir, sucede exactamente igual como las palabras reservadas del lenguaje Java las cuales solamente las podemos utilizar para desarrollar código Java y no otra cosa. Para indicar al IDE que son etiquetas propias para comentarios comenzaremos la línea con el carácter @. Algunas de estas etiquetas y sus funciones son las que se indican a continuación:

Un ejemplo que ilustra el uso de etiquetas especiales sería el siguiente:

Autoevaluación

3.2. Generando los Javadocs para un proyecto

Generando los Javadocs para un proyecto

Una vez que ya hemos visto lo que son los Javadocs y qué sintaxis se utiliza para su creación vamos ahora a ver cómo podemos generar los mismos desde el IDE Netbeans. Como es de suponer al igual que para otras operaciones esto se podrá realizar de diversas formas, algunas de ellas son las que se comentan a continuación:

1. **Accediendo al menú `Build` del IDE y seleccionando el comando `Generate Javadoc For nombreproyecto`**. Si en la ventana proyectos hubiese más de un proyecto cargado, previamente debemos seleccionar en la misma el proyecto sobre el que queremos generar los Javadocs.
2. **Accediendo a la ventana de proyectos, pulsando con el botón derecho del ratón sobre el proyecto que se quiere generar y seleccionando del menú contextual resultante el comando `Generate Javadoc for project`.**

Ambas opciones realizan la misma función. El utilizar una u otra dependerá de la que nos resulte más cómoda.

Una vez que **la documentación se ha generado estará disponible dentro de la carpeta dist** de dicho proyecto. Para cada proyecto existirá su documentación generada.

Como curiosidad, para que veas la potencia de esta característica de Java, decirte que toda la documentación de la API de Java está hecha automáticamente a partir de los comentarios Javadocs.

4. Elementos importantes para el desarrollo de aplicaciones Java

Elementos importantes para el desarrollo de aplicaciones Java

*Tras varias prácticas con NetBeans, **Víctor** ha llegado a sentirse cómodo con este entorno de desarrollo y construye aplicaciones con relativa facilidad. Es posible que esta destreza la haya adquirido por sí mismo, cuando tuvo el acierto de preguntar a **Carmen** qué aspectos y elementos debe tener en cuenta a la hora de trabajar con proyectos en este IDE. Al principio a **Carmen** le pareció una tontería que tomara nota de los pasos que iba siguiendo en la primera de las prácticas, pero a **Víctor** le sirvió para ordenar las ideas y adquirir soltura para comenzar las aplicaciones, que sin duda es el momento de mayor incertidumbre.*
__Víctor__ elaboró varios borradores con los pasos a seguir al desarrollar un proyecto. Cada uno de estos borradores consistía en una corrección del anterior mejorando algunas cosas y finalmente consiguió asentar las ideas de modo que ya no necesitaba el borrador porque había memorizado todos los pasos y los aspectos importantes de NetBeans ante un proyecto. Realmente ha constatado que debe ser ordenado y tener muy claro el motivo de optar por las diferentes alternativas que se ofrecen en cada momento para fijar las propiedades que después determinarán el comportamiento del proyecto y ajustar el SDK a las necesidades del cliente.

En este último punto vamos a ver cuales son las **propiedades** que se pueden establecer para un proyecto que se desarrolle a través del IDE Netbeans. Para ello comenzaremos con un repaso de conocimientos sobre la programación en JAVA. Esto nos va a servir para adquirir o consolidar conocimientos fundamentales a la hora de programar en JAVA, ya sea utilizando un IDE o no. Bien, ¡comencemos!

4.1. Entendiendo el SDK

Entendiendo el SDK

Lo primero que vamos a recordar es: **¿Qué es un IDE? Un IDE ("Integrated Development Environment", o Entorno Integrado de Desarrollo) es una herramienta que usamos para desarrollar programas, en este caso, programas Java, ya que el IDE utilizado es NetBeans**. Éste nos facilita un conjunto de herramientas para que el desarrollo de programas sea fácil y productivo.

La **productividad** del programador está directamente relacionada con el coste de desarrollo y mantenimiento del software. Cualquier empresa (o persona) dedicada al desarrollo del software debe procurar disponer de herramientas adecuadas para que el tiempo de desarrollo, y por tanto el coste de desarrollo, sea mínimo. Es la única forma de que sus productos (aplicaciones desarrolladas) sean competitivos. Y justamente una de esas herramientas que facilita todo el trabajo, reduciendo el tiempo necesario para el desarrollo de la aplicación es el IDE. Podemos programar cualquier aplicación Java, incluso las más elaboradas, con interfaz gráfica incluida sin más que el **Kit Estándar de Desarrollo de Software** (**SDK**, que incluye el **JDK** u el **JRE**, como vimos en unidades anteriores), pero sería mucho más complejo, requeriría un mejor dominio del lenguaje, y el tiempo necesario sería mucho mayor. ¡Disponer de un IDE adecuado lo hace todo más fácil!

Ahora bien**, cuando instalamos NetBeans, ¿qué sucede? Lo que ocurre es que se instala previamente el SDK ("Software Development Kit", Kit de Desarrollo de Software), necesario para que nuestro entorno pueda realizar funciones de compilación, ejecución, generación de documentación, etc**. El SDK se divide básicamente en dos partes:

1. **JDK ("Java Development Kit", Kit de Desarrollo Java): Que es el que nos permite compilar y ejecutar programas Java mediante el conjunto de herramientas que integra**.
2. **Una versión reducida del JDK que es el JRE ("Java Runtime Environment", Entorno de Ejecución Java), destinada únicamente a ejecutar código Java (no permite compilar) y que es la que aporta la JVM (Máquina Virtual Java).** Como es sabido, Java es un lenguaje multiplataforma, es decir, puede ser ejecutado en cualquier plataforma (Windows, Linux, Solaris, etc.) ya que cuando generamos los archivos ejecutables estos no son generados para una plataforma concreta como ocurre con otros lenguajes que no son multiplataforma, sino que son generados como bytecodes para la JVM de Java y ésta es estándar para todas las plataformas.

Estas dos herramientas se pueden observar dentro del directorio JAVA que se creó cuando se instaló el SDK. **Según lo comentado anteriormente podemos afirmar dos cosas importantes**:

1. **Para ejecutar una aplicación Java en un ordenador debemos tener instalado en el mismo el JRE, que es el que nos aporta la Máquina Virtual de Java**.
2. **Para poder desarrollar una aplicación Java ya sea con un editor de texto simple o un IDE que use el lenguaje, necesitamos previamente tener instalado el JDK**.

Autoevaluación

4.2. Tipo de ficheros que generamos con el IDE

Tipo de ficheros que generamos con el IDE

Vamos a seguir aclarando conceptos. **Cuando estamos utilizando un editor ya sea un editor de textos simple o el que viene integrado en el IDE NetBeans, para escribir un archivo Java**, lo que estamos creando es un fichero con la extensión .java. Este fichero es un fichero de texto, claro que el texto es escrito siguiendo la sintaxis del lenguaje Java. Bien, cuando compilamos el fichero `.java` lo que se genera es un fichero `.class` que es un fichero en binario y es el que se utiliza cuando ejecutamos.

Bien, vamos a suponer que tenemos escrito un fichero `.java`, como puede ser `Altapelicula.java`. **Cuando compilamos con el IDE este fichero lo que ocurre es que éste llama a la aplicación `javac.exe` que se encuentra dentro del JKD en su carpeta BIN, de esta forma se genera el fichero `Altapelicula.class` que será el que utilice la aplicación `java.exe` cuando le indicamos al IDE que ejecute el fichero**.

El entorno sólo nos está ahorrando el trabajo de abrir y cerrar el editor, tipo bloc de notas, guardar el archivo del programa con las modificaciones, abrir la consola de MSDOS, establecer el **path** y el **classpath** adecuado, y ejecutar el fichero `javac.exe` para compilar y el fichero `java.exe` para ejecutar la clase compilada, tal y como hacíamos con nuestros primeros programas de ejemplo.

De lo comentado anteriormente deducimos que:

1. Dentro de la carpeta BIN del directorio JDK se encuentran los programas que el entorno utiliza para ejecutar, compilar, etc.
2. Los ficheros escritos en lenguaje Java son los `.java`
3. Los ficheros que son ejecutados son los `.class`

4.3. Ejecutando programas Java desde cualquier carpeta

Ejecutando programas Java desde cualquier carpeta

En este apartado vamos a revisar lo aprendido sobre la variable de entorno **classpath**.

Para poder ejecutar programas Java desde cualquier carpeta, la carpeta (BIN) con los archivos `javac.exe` y `java.exe` debe estar incluida en el PATH del sistema. Tecleando PATH en una ventana de MSDOS o del Intérprete de comandos, se muestran las rutas que en ese momento forman parte del PATH.

Java utiliza además una nueva variable de entorno denominada **classpath**, la cual determina dónde buscar tanto las clases o librerías de Java (el API de Java) como otras clases de usuario. La variable **classpath** puede incluir la ruta de directorios o ficheros `*.zip` o `*.jar` en los que se encuentren los ficheros `*.class`. En el caso de los

ficheros **`*.zip`** hay que observar que los ficheros en él incluidos no deben estar comprimidos.

Para acceder a la variable **`CLASSPATH:`**

1. Desde Windows 95-98-Millenium: en el fichero **`Autoexec.bat`** debe aparecer una línea como ésta:
   `set CLASSPATH=.\;%JAVAPATH%\lib\classes.zip;%CLASSPATH%`
2. Desde Windows NT-2000-XP: en Propiedades del Sistema + Opciones avanzadas + Variables de entorno hay que crear una nueva variable llamada CLASSPATH, en la que escribir simplemente un punto:
   `CLASSPATH=.`

4.4. Estableciendo propiedades del proyecto con NetBeans.

Estableciendo propiedades del proyecto con NetBeans.

En este punto vamos a ver las **propiedades** que se pueden establecer para un proyecto desde el IDE. Algunas de estas propiedades se corresponden con los elementos que hemos comentado en los puntos anteriores. Lo primero que a vamos hacer es ver cómo podemos acceder a las propiedades de un proyecto desde el IDE y posteriormente comentar qué son cada una de ellas.

Para acceder a las propiedades de un proyecto podemos hacerlo de la siguiente forma:

Pulsamos con el botón derecho del ratón sobre el proyecto y del menú contextual resultante seleccionamos el comando **`properties`**.

De esta forma aparecerá la ventana siguiente, en la que **agrupados por categorías encontramos los siguientes elementos que podemos configurar**:

1. **`Sources`**: Esta categoría muestra los archivos fuente que pertenecen a nuestro proyecto, es decir, los archivos que nosotros creamos cuando programamos. Teniendo seleccionada esta categoría podemos ver en la parte derecha de la ventana cuál es el directorio de trabajo para el proyecto actual y qué paquetes lo integran.
2. **`Libraries`**: **Esta categoría hace referencia al JDK que nuestro IDE está usando para realizar las operaciones de compilación, ejecución, generación de Javadocs, etc.** Si nuestra aplicación necesita usar alguna otra librería, debe aparecer indicado aquí. Veremos un ejemplo de esto en la unidad 19, cuando hablemos de generación de ficheros PDF e impresión en Java, que necesitan la inclusión de librerías adicionales.

Si observamos la parte derecha **de la ventana para esta categoría, podemos ver la versión del JDK que está utilizando nuestro IDE. También para esta versión del JDK, qué elementos del mismo utiliza para compilar, ejecutar, etc.**

Desde esta ventana podemos fijar qué JDK va a utilizar nuestro IDE, pulsando sobre el botón **`Manager Plataforms...`**, de esta manera accederemos a la ventana **`Java Platform Manager`** desde donde se podrá realizar dicha operación.

Cuando establecemos un JDK para el IDE de esta manera, estamos indicando el valor de la variable de entorno Classpath para el IDE, es decir, lo que hacemos en realidad es indicar al IDE, dónde se encuentran las herramientas que va a necesitar para compilar, ejecutar, generar Javadocs, etc.

3. **`Build`**: Esta categoría hace referencia a aspectos de construcción de aplicaciones Java relacionados con la compilación, el uso de paquetes y documentación.
4. **`Run`**: Esta última categoría es muy importante ya que a través de la misma indicamos algunas opciones que se van a tener en cuenta para la ejecución de nuestro programa JAVA. Estas opciones son las que se enumeran a continuación.

Los elementos que podemos configurar desde esta categoría los describimos por su importancia a continuación:

- **`Main Class`**: **Indica al IDE cuál va a ser nuestra clase principal. Si por ejemplo creamos un formulario y sobre éste establecemos un menú con opciones (una aplicación gráfica y este formulario va a ser la interfaz de nuestra aplicación), nosotros podemos indicar a través de aquí que éste va a ser la clase principal, de manera que cuando se ejecute nuestro programa este formulario será el que se presente como interfaz del programa**. En definitiva, debe ser una clase de nuestro proyecto que contenga el método **`main()`**, por el que empieza la ejecución. Puede haber varias clases con método **`main()`**, y desde aquí decidimos por cuál de ellas comenzamos a ejecutar la aplicación.
- **`Arguments`**: Si nuestro programa o aplicación Java usa argumentos (variables que se inicializan en tiempo de ejecución) estos aparecen o se pueden establecer desde aquí, tal y como se vio en unidades anteriores, con el ejemplo de **`CrearDirectorios`**.
- **`Working Directory`**: Directorio de trabajo, es decir, dónde se encuentran los archivos que vamos generando con el IDE. Por defecto, éste es el que se estableció cuando se creó el proyecto.
- **`VM Options`**: Esta opción permite especificar opciones para la Máquina Virtual de Java (JVM). De esta forma podemos establecer argumentos que la JVM va a tener en cuenta cuando ejecute un programa.

Esos argumentos que establecemos indican a la JVM acciones a realizar en procesos como depuración, ejecución etc. de aplicaciones.

Autoevaluación

PARA SABER MÁS.
Desde este enlace podrás profundizar sobre cualquiera de los puntos aquí descritos y ampliar otros que te interesen sobre el IDE.
Web Oficial de NetBeans.