Unidad 3

Lenguajes de programación y entornos de desarrollo integrado

Programación

3

Índice

Introducción

Hoy en día, todo lo que nos rodea cuenta con sistemas informáticos y programas que se ejecutan sobre estos. Una gran cantidad de tareas se han automatizado gracias a ellos en muy poco tiempo. Pero no solo ha revolucionado métodos ya existentes, sino que también se han abierto camino en nuevos ámbitos. Todo esto gracias a la mejora y abaratamiento de los equipos informáticos, así como a las mejoras en infraestructuras de telecomunicaciones.

Para llegar a este punto, se ha recorrido un largo camino donde muchas tecnologías han ido evolucionando en función de las necesidades existentes. Gracias a esta evolución, ha cambiado nuestra forma de abordar los problemas y ofrecer soluciones, por lo tanto, los lenguajes de programación también se han actualizado.

Lo que empezó con el código máquina, una secuencia de unos y ceros que eran interpretados por un sistema para lograr un resultado, ha evolucionado hasta el uso de palabras con un orden lógico que facilitan el desarrollo de los programas. Por lo que podemos entender que hay distintas generaciones entre los lenguajes de programación.

Al finalizar esta unidad

- Conoceremos los diferentes lenguajes de programación.
- Podremos describir las características de la tecnología Java desde línea de comandos.
- Manejaremos las herramientas de la tecnología Java desde línea de comandos.
- Utilizaremos los entornos integrados de desarrollo.

3.1. Lenguajes de programación

Como ya hemos comentado, entre los lenguajes de programación hay una gran variedad, y muchos de ellos han llevado a cabo su propia evolución en el tiempo. Como los lenguajes fueron creado con distintas características y propósitos, se les puede ordenar en función de distintos aspectos:

- **Según generación:** podemos distinguir entre lenguajes de primera, segunda, tercera, cuarta y quinta generación.
- **Según su tipo de ejecución:** pueden ser lenguajes compilados y lenguajes interpretados.
- **Según su propósito:** existen de propósito específico y de propósito general.

3.2. Java

En este apartado hablaremos en profundidad sobre el lenguaje de programación Java, y de por qué fue elegido como el lenguaje ideal para aprender programación.

3.2.1. Descripción de la tecnología

Según el Índice TIOBE, Java es uno de los lenguajes de programación con mayor demanda de mercado, además, se puede utilizar de forma libre y es de los más populares. Este lenguaje nos da la posibilidad de programar una gran variedad de aplicaciones (aplicaciones web, de escritorio, juegos, servicios, etc.) y lo componen diferentes tecnologías

El 23 de mayo de 1995 se anunció Java, sin embargo, la primera versión del JDK (JDK 1.0) fue publicada por Sun Microsystems, la cual fue adquirida por Oracle hace más de una década, el 23 de enero de 1996, pretendiendo que el desarrollo de programas fuera más fácil, y que no hubiera que tener en cuenta ni el sistema operativo ni el hardware usado. Gracias a esto, se podía generar un código intermedio a partir de programas desarrollados, el cual era capaz de ejecutarse en cualquier sistema, siempre que contase con el entorno de ejecución de Java (JRE). Esto facilitó la portabilidad, redujo costes, minimizó tiempos de implantación y ya no hacía falta el desarrollo de otras versiones de los programas para adaptarlas a otras plataformas.

Hay una extensa variedad de distribuciones, que han evolucionado para cubrir las necesidades que han ido surgiendo. Esto es debido a la alta variedad de aplicaciones que se pueden desarrollar con esta herramienta, y encontramos:

- **J2ME (Java 2 Micro Edition)**, esta distribución surgió para desarrollar software que se utilizaría en dispositivos con menos recursos (impresoras, móviles, electrodomésticos, etc.) Gracias al gran crecimiento de los juegos en móviles esta distribución fue muy exitosa, pero su uso se ha reducido significativamente.
- **J2SE (Java 2 Standard Edition)**, esta distribución estaba a cargo del desarrollo de aplicaciones de escritorio, tanto para servidores como para ordenadores personales, también se encargaba de las conexiones con bases de datos (JDBC) y de aplicaciones cliente-servidor. Esta distribución utiliza una gran cantidad de librerías contenidas en J2ME, pero no lo engloba en su totalidad.
- **J2EE (Java 2 Enterprise Edition)**, esta distribución es la más completa de todas, se centra en desarrollar e implementar aplicaciones empresariales basadas en tecnología web (Servlets, JSP, WebServices, etc.).

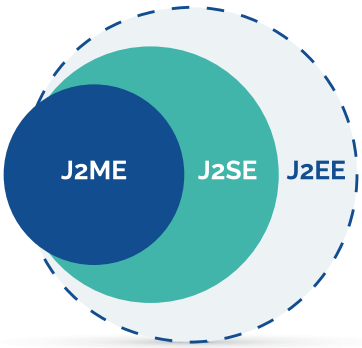

Imagen 1. Distribuciones de Java.

3.2.2. Entorno de desarrollo JDK

Oracle suministra entornos de desarrollo que nos serán necesarios para el desarrollo de aplicaciones que se basen en Java. Estos entornos cuentan con herramientas que nos permitirán construir y ejecutar casi cualquier tipo de aplicación.

El JDK (Java Development Kit) será el entorno de desarrollo que utilizaremos. Se recomienda el uso de la última versión disponible que sea estable, aunque puede suceder, que dicha versión cuente con alguna clase de incompatibilidad con algunas herramientas que pueda ser necesario instalar posteriormente (IDE).

Oracle suministra dos tipos de JDK, los cuales son muy parecidos y ofrecen una funcionalidad similar, pero cuentan con licencias distintas:

- Binary Code License Agreement for the Java SE Platform Products and JavaFX es la licencia con la que cuenta el JDK.
- GNU General Public License, version 2 es la licencia con la que cuenta el OpenJDK

Las aplicaciones que desarrollemos en el JDK necesitarán del JRE (Java Runtime Environment) para ser ejecutadas. El JRE viene suministrado junto al JDK.

3.2.3. Entorno de ejecución JRE

A la hora de escribir programas, los lenguajes de alto nivel proporcionan, en relación a los de bajo nivel, un nivel superior de abstracción sobre la arquitectura física. Utilizar compiladores se hace necesario para la obtención de programas partiendo de ellos.

En un principio, los compiladores traducían los programas desde un lenguaje similar al de las personas, a un lenguaje que pudiesen comprender el sistema operativo que lo iba a ejecutar. C, Pascal o Delphi, por ejemplo, trabajaban de esta manera.

En cambio, en el mundo del desarrollo existen otro tipo de tecnologías muy importantes que, mediante compiladores, traducen a un código intermedio esos programas. Este código intermedio no se puede ejecutar directamente en el sistema operativo, por lo que se requiere de un elemento especial, el cual se ejecuta sobre el sistema operativo y que se encarga de traducir ese código intermedio a un lenguaje que el este pueda comprender.

Este código intermedio se conoce como bytecode en el mundo de Java, y el JRE es el elemento especial que intermedia entre sistema operativo y programa. Al JRE lo componen bibliotecas con código Java, una JVM (Java Virtual Machine) y algunos elementos adicionales. El código se puede ejecutar, sin necesidad de modificarlo, en diferentes sistemas. Esto es gracias a que existen versiones para la mayoría de los sistemas operativos.

CIL o MSIL es como se le denominaría al código intermedio en el mundo .NET y la máquina virtual que lo traduciría sería CLR (Common Language Runtime).

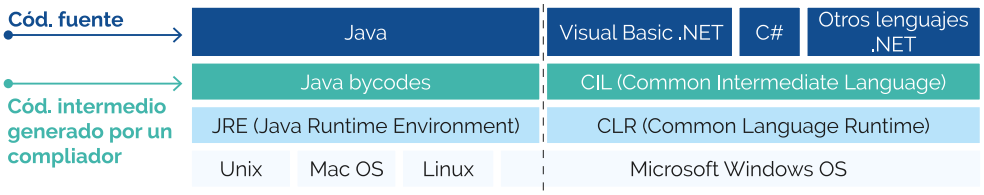

Imagen 2. Java vs .NET

Para evitar que el código intermedio se traduzca una y otra vez, se incluyó un compilador JIT (Just In Time) en el JRE. El compilador, entre otras acciones, utiliza código que ha compilado anteriormente, evitando así tener que interpretarlo de nuevo, esto reduce considerablemente los tiempos de ejecución y mejora el rendimiento.

3.2.4. Variables de entorno PATH y CLASSPATH

Para que se produzca la compilación, generación y ejecución de paquetes jar de una forma cómoda en Java, debemos configurar correctamente los valores de dos variables de entorno:

- **PATH:** variable que usa el sistema operativo para la búsqueda de los ejecutables de las distintas aplicaciones desde el terminal de Linux o el intérprete de comandos de MS-DOS.
- **CLASSPATH:** esta variable la utiliza el entorno de ejecución de Java (JRE) para realizar la búsqueda de clases y paquetes definidos por el usuario.

Es muy importante que configuremos correctamente estas variables si queremos trabajar desde las líneas de comandos o terminal.

3.2.5. Tipos de archivos empleados

A continuación, se describen los distintos tipos de archivos que se van a utilizar y sus respectivas extensiones:

- Cuando utilizamos Java, el código fuente debe ser almacenado en un archivo de texto plano y cuya extensión sea .java.
- Al compilar con éxito un programa que haya sido escrito en un archivo .java, por lo menos nos devolverá un archivo que contenga código intermedio (bytecode) cuya extensión será .class
- Otros lenguajes, al compilar correctamente, generan archivos ejecutables, sin embargo, Java crea por cada clase del proyecto, un archivo independiente de extensión .class. En los programas en los que haya una gran cantidad de clases, eso puede generar molestias a la hora de su manejo, por eso Java posibilita la opción de empaquetar todos esos archivos y los recursos utilizados por ellos (archivos de texto plano, imágenes, etc.) en un solo archivo de extensión .jar (Java Archive). Para construir estos archivos se utiliza el algoritmo de compresión Zip.
- Continuando con esta lógica y para que implantar las aplicaciones en los servidores de aplicaciones, Java pone a nuestra disposición los siguientes archivos:
  - .war (Web Application Archive): Contienen aquellos archivos que conforman una aplicación web que se haya desarrollado con Java (Servlets, JSP, etc.) y, además, facilitan su despliegue.
  - .ear (Enterprise Archive): sirven para empaquetar uno o varios módulos en un único archivo, de manera que se facilita su despliegue en un solo paso sobre un servidor de aplicaciones de acuerdo con la estructura que se encuentra definida en el archivo que contiene META-INF/applicatoin.xml. Por lo que le es posible contener archivos de cualquier tipo (también .war y .jar).

3.2.6. Recolector de basura

El recolector de basura (garbage collector) es un elemento muy importante de Java, ya que el programador puede despreocuparse de tareas de reserva y liberación de memoria. Esto es debido que el garbage collector abraza una filosofía contraria a la impuesta por la gestión manual de memoria.

Esta característica puede ser percibida como una ventaja o un inconveniente por los programadores que vengan de tecnologías como Delphi o C:

- **Ventaja:** Se facilita el desarrollo y se reducen los errores de los programadores al gestionar la memoria. Esto permite que se escriba código sin preocuparse de que los recursos del sistema se consuman antes de que el programa pueda finalizar.

- **Inconveniente:** Este elemento consume recursos, lo cual afectará al rendimiento del programa. Por este motivo, no se lleva a cabo el desarrollo de sistemas de tiempo real mediante este tipo de herramientas. Con respecto al recolector de basura, hay distintas posturas:
  - Ejecutar el recolector cuando no quede memoria libre.
  - Poner un límite de memoria ocupada, para que se ejecute el recolector al alcanzarlo.
  - Ejecutarlo a intervalos regulares.
  - Ejecutarlo antes de cada reserva de memoria.
  - Permitir que el programador pueda invocar el recolector cuando considere conveniente.

El recolector de basura es utilizado por varias tecnologías, y cada uno lleva a cabo su implementación de diferente manera. Java crea una subrutina que se ejecuta paralelamente a los demás programas y que detecta los objetos que están siendo usados y los que no. El funcionamiento es el siguiente:

- **Marcado:** primero se identifican en el Heap aquellas zonas de memoria de objetos que no cuenten con su referencia en el Stack. Identificando así las zonas de memoria que no se están usando para liberarlas posteriormente.
- **Borrado de objetos no referenciados:** se liberan aquellas zonas en las que se encontraban objetos sin su correspondiente referencia.
- **Compactación del espacio usado:** al liberar espacio, se crearán huecos en la memoria, por lo que, para optimizar el guardado de nuevos objetos en la memoria, el recolector se encargará de compactar ese espacio.

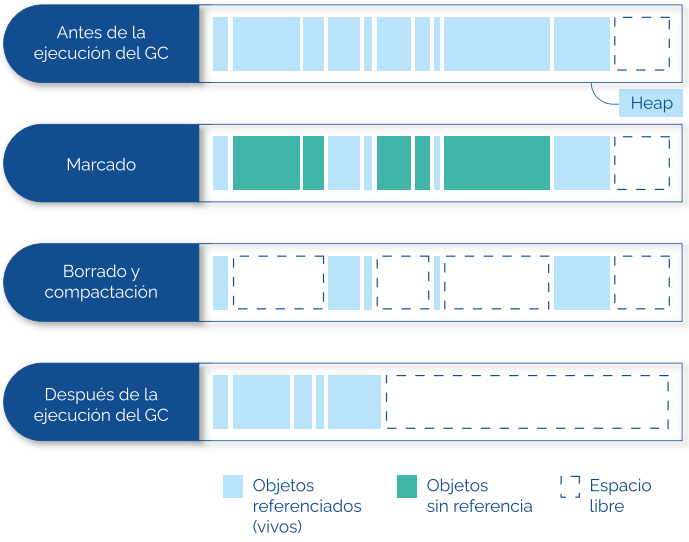

Imagen 3. Funcionamiento del recolector de basura (Java).

3.3. Programación sin IDE

En apartados anteriores hemos visto las características de los entornos de desarrollo y ejecución JDK y JRE, los cuales, junto a un editor de texto plano, resultan necesarios para llevar a cabo el desarrollo de un proyecto. Se manejarán tanto desde línea de comandos como desde un IDE con interfaz gráfica.

Manejar Java desde la línea de comandos no supone ningún reto. Se mostrará solo el uso de uno de ellos, ya que los comandos son equivalentes para todos los sistemas operativos.

3.3.1. Compilar un programa

Para convertir a código intermedio el código fuente que previamente hayamos escrito en un archivo de texto plano, se utilizará el comando Javac. La sintaxis de este comando es:

```
Javac <opciones> <ficheros>
```

3.3.2. Crear un jar

Habiendo generado los archivos de código intermedio y disponiendo de los recursos que estos van a utilizar, ya podemos generar un archivo .jar para comprimirlos, de manera que se facilita su almacenaje, transporte e inclusión en proyectos futuros.

```
jar {ctxui}[vfm0Me] [fichero jar] [fichero manifest]
[punto de entrada a la aplicación] [-C dir] ficheros ...
```

Las opciones -cvf, con las que podremos construir estos archivos, nos permiten indicar que queremos crear un nuevo archivo con un determinado nombre. Se podrán construir como:

- **Simples contenedores de recursos los cuales se usarán en otros proyectos**; sería suficiente con general el archivo .jar.
- **Ejecutables comprimidos.** En estos casos deberemos indicar la clase principal además de añadir la opción -e.

Sobre los archivos .jar se puede añadir o consultar contenido (opción -u y opción -t respectivamente), pero para eliminar elementos contenidos en ellos, se deben utilizar otro tipo de herramientas como (WinRAR, WinZip, 7zip, etc.)

Al construir estos archivos, se creará un directorio META-INF dentro del archivo contenedor, en el cual podremos encontrar un archivo de texto plano MANIFEST.MF. En este archivo podemos encontrar información importante sobre el archivo .jar, como, por ejemplo, la clase principal de un ejecutable.

3.3.3. Ejecutar un programa

Habiendo generado el código intermedio y disponiendo de un archivo .jar ejecutable o un .class, podremos lanzarlo sobre la JRE para su interpretación y ejecución. La sintaxis es la siguiente:

```
Java [opciones] <clase principal> [argumentos…]
```

3.4. Programación con IDE

Anteriormente hemos visto las principales características de los entornos de desarrollo y ejecución JDK y JRE, y hemos mostrado como utilizar este tipo de herramientas desde la línea de comandos.

Pero desarrollar grandes proyectos utilizando solo la línea de comandos puede ser complicado, por lo que, para solucionar este problema, se crearon los entornos de desarrollo integrado.

3.4.1. Características y elementos

Estas aplicaciones, conocidas como IDE (Integrated Development Environment), constituyen una aglomeración de herramientas, las cuales fueron generadas para que se facilitase el desarrollo de programas. Estas herramientas se suministran conjuntamente.

Normalmente, los entornos de desarrollo ofrecidos por este tipo de paquetes están basados en aplicaciones gráficas, que facilitan el acceso a las herramientas, las cuales, en su mayoría, no se encuentras disponibles en un simple JDK.

Algunas de las herramientas que generalmente podemos encontrar en un IDE son un depurador, un editor, un ensamblador, un compilador, etc.

3.4.2. Alternativas

En el mercado podemos encontrar varias alternativas tanto libres como de pago. Algunas están orientadas al desarrollo en varias tecnologías, mientras que otras se centran en un único lenguaje.

Caben destacar NetBeans y Eclipse. Son gratuitas, soportan distintos tipos de lenguaje, y pueden ser instaladas en sistemas como Windows y Linux (entre otros).

Otras alternativas serían:

- **IntelliJ IDEA:** Desarrollado en Java y con versiones gratuitas y de pago. Disponible para Windows y Linux. Puede tener soporte para distintos lenguajes.
- **JCreator:** Solo está disponible en Windows, con una versión de pago y otra libre. Desarrollado en C++, por lo cual, se recomienda su uso en máquinas que limitados recursos.

3.4.3. Configuración de NetBeans

Podemos realizar configuraciones sobre el IDE de tipo específico, lo cual solo afecta a un proyecto, o de tipo general, las cuales afectan a todos los proyectos. Más adelante se abordará con detenimiento qué es un proyecto, ahora mismo es suficiente con saber que se trata de un conjunto de recursos por los cuales se va a representar el programa a construir, y que se organizan en una carpeta.

Configuraciones generales

Las configuraciones generales se encuentran en Tools > Options.

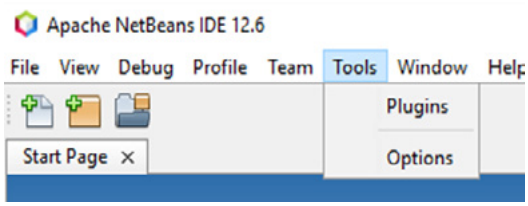

Imagen 4. Acceso a opciones generales en NetBeans.

Lo primero será la activación de las características Java, de manera que el entorno esté preparado para emplear esta tecnología. Lo que haremos será habilitar el soporte para Java desde la opción Java (en la pestaña GUI Builder o Java Debugger).

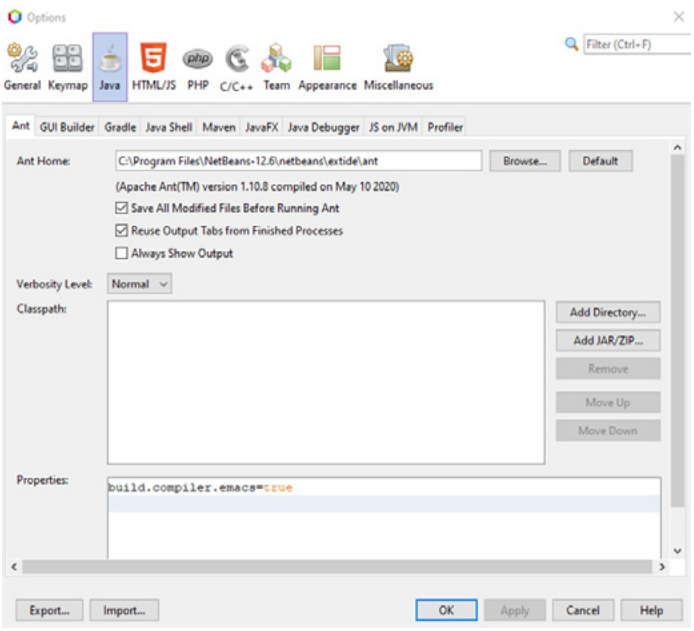

Imagen 5. Activación de características.

De forma automática se habilitarán todas las herramientas del IDE que necesitaremos para desarrollar el proyecto, así como nuevas configuraciones generales que estaban ocultas.

Estas son las configuraciones generales más importantes:

- **Editor:** establece cómo se corrige y optimiza el código (On Save), cómo funciona el autocompletado (Code Completion), como se realizarán las tabulaciones en el código (Formatting), etc.
- **Fonts & Colors:** desde aquí podremos elegir el tipo de simbología y la tipografía que utiliza el editor.
- **Keymap:** principalmente se utiliza para establecer shortcuts.
- **Java:** nos da la posibilidad de configurar como se comportará el depurador (Java Debugger) y el constructor (GUI Builder)
- **Appearance:** permite configurar el aspecto que tender el IDE (ventanas, separadores de documentos y aspecto).

Se recomienda también que las ventanas de Projects y Files (proyectos y ficheros) estén visibles, para poder acceder a la estructura del proyecto desde el IDE.

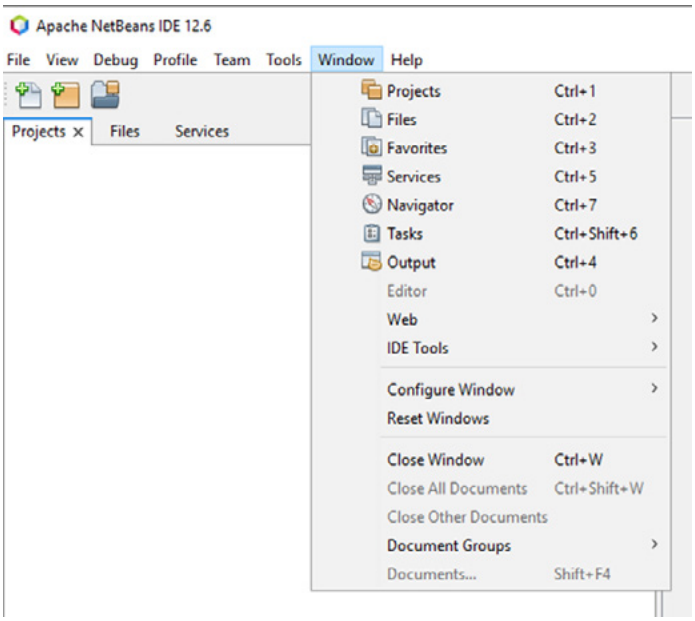

Imagen 6. Acceso a ventanas Projects y Files.

Configuraciones específicas de cada proyecto

Nada más crearlos, los proyectos seguirán la configuración general establecida en el IDE. No obstante, podemos hacer cambios en algunas configuraciones sin afectar a las demás.

Haciendo clic derecho en proyecto y seleccionando Properties, se abrirá una ventana desde la que podremos configurar las propiedades de nuestro proyecto (generales y específicas del proyecto).

3.4.4. Manejo del IDE

Crear un proyecto

Después instalar y configurar el IDE, ya podemos empezar a escribir nuestro primer programa. NetBeans funciona por proyectos, por lo que, aun cuando ofrece la posibilidad de crear archivos independientes de lenguajes como JavaScript o HTML, deberemos crear dentro de un proyecto nuestro primer programa.

Las acciones más habituales se encuentran en la barra de tareas de NetBeans. Por lo que crearemos el proyecto haciendo clic en el icono de la carpeta.

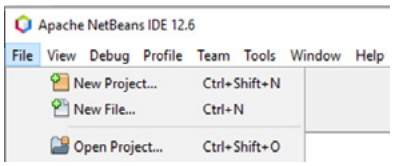

Imagen 7. Creación de proyectos.

De manera automática, para guiarnos en el proceso, se abrirá un asistente. Deberemos seleccionar Java Application, y nos aparecerá una ventana en la que designaremos el directorio y el nombre del proyecto. Si marcamos el checkbox de Create Main Class, podremos crear de manera automática la clase principal del proyecto.

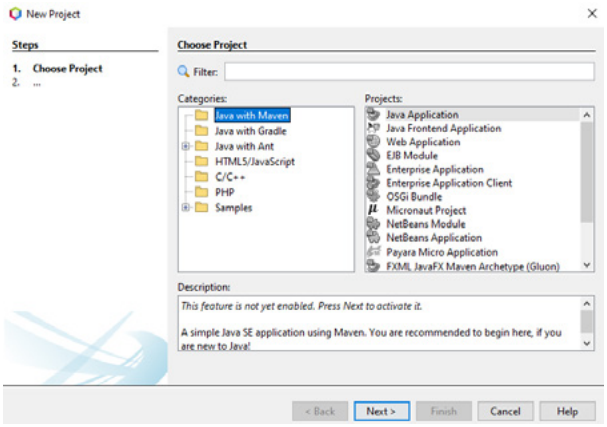

Imagen 8. Selección de tipo de proyecto.

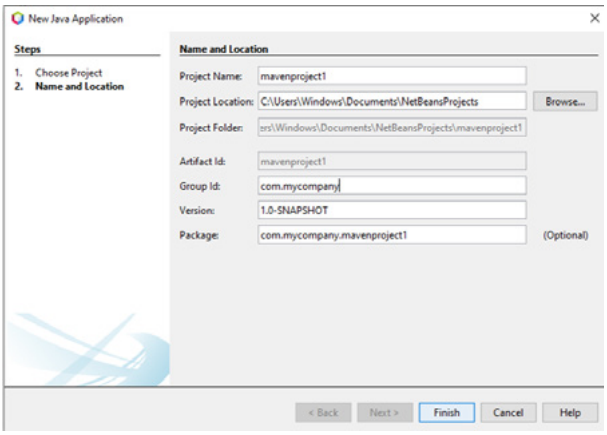

Imagen 9. Establecimiento de las propiedades del proyecto.

Como resultado de este paso se habrá generado el proyecto en el directorio indicado y la estructura del programa que contendrá la clase principal.

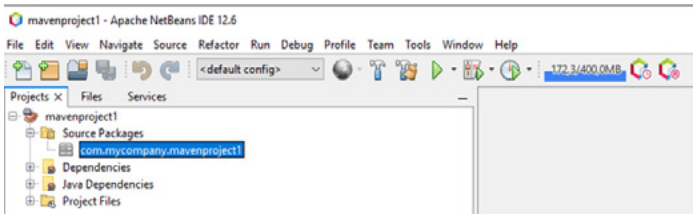

Imagen 10. Nuevo proyecto.

Construir el proyecto

Una vez se haya generado el proyecto y escrito el código del programa, se deberá generar el código intermedio que será almacenado en los archivos .class. Para ello el entorno ofrece dos alternativas:

- **Build Project:** construye el proyecto, compilando únicamente los archivos de código fuente que sea necesario y modificando el jar con dicho contenido.
- **Clean and Build Project:** elimina todo resultado de la construcción anterior, compila todos los archivos de código fuente, enlaza los contenidos y construye un archivo .jar que contendrá los archivos compilados y recursos que utilicen.

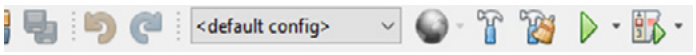

Imagen 11. Herramientas para la construcción de proyectos.

La primera vez que se construya el proyecto ambas alternativas provocarán el mismo resultado. Desde la ventana Files se podrá tener acceso a la estructura de directorios generada para el proyecto.

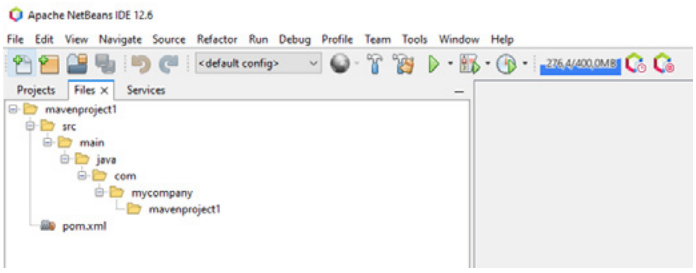

Imagen 12. Estructura de directorios del proyecto.

Ejecutar un programa

El proyecto libre de errores habrá dado como resultado un código intermedio que estará listo para ser ejecutado. Para ejecutar el código y visualizar el resultado existe la opción Run Project. El resultado del programa podrá observarse en la ventana Output.

Depurar un programa

De la misma forma que se puede ejecutar el código, también se podrá depurar para corregir posibles errores producidos en tiempo de ejecución o estudiar su comportamiento. En la opción Debug del menú de opciones se encuentran los ítems de acceso a las diferentes opciones que permiten realizar esta tarea. Entre estas se encuentran:

- **Ctrl + F5:** comenzar el Debug.
- **Mayús + F5:** detener el Debug.
- **F5:** continuar con la ejecución hasta el siguiente punto de ruptura o hasta el final del programa.
- **F4:** ejecutar el programa hasta la posición actual del cursor.
- **F7:** ejecutar la siguiente línea del programa. Si la siguiente instrucción es la llamada aun método, continuará con la ejecución línea a línea dentro del método.
- **F8:** ejecutar la siguiente línea del programa. Si la siguiente instrucción es la llamada aun método, ejecutará el método como si de una instrucción simple se tratase.
- **Ctrl + F8:** establecer o eliminar un punto de ruptura sobre la línea actual del cursor.
- **Ctrl + Mayús + F7:** añadir una nueva ventana de inspección (Watch).

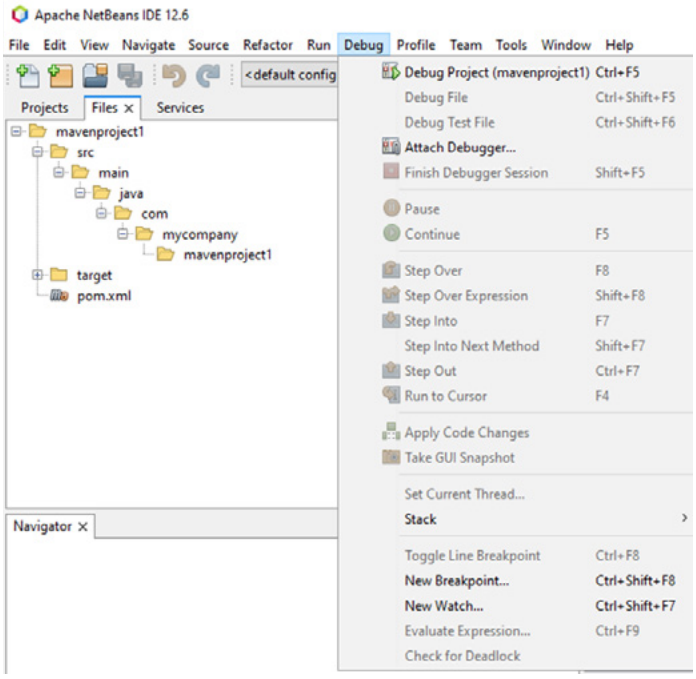

Imagen 13. Herramientas para la depuración de programas.

Al igual que en la ejecución, los resultados de la depuración podrán observarse en la ventana Output, mientras que el valor de las variables manejadas en el programa podrá visualizarse en la ventana Variables.

También será posible consultar el valor de ciertas variables situando el cursor del ratón sobre ellas cuando el programa haya quedado detenido en algún punto de ruptura.

www.universae.com