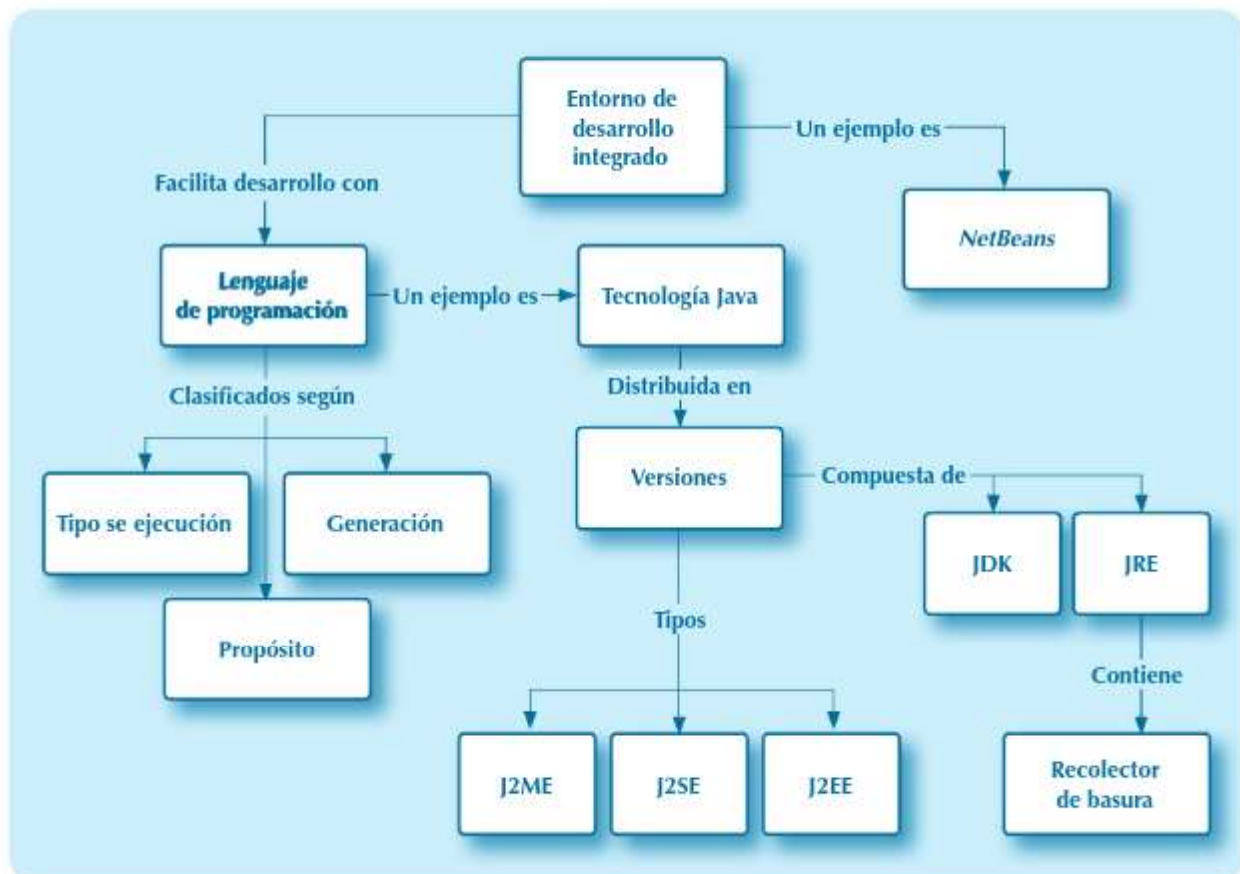


Lenguajes de programación y entornos de desarrollo integrado

Objetivos

- ✓ Conocer y clasificar los diferentes lenguajes de programación.
- ✓ Describir las características de la tecnología Java.
- ✓ Manejar las herramientas de la tecnología Java desde línea de comandos.
- ✓ Utilizar los entornos integrados de desarrollo.

Mapa conceptual



Glosario

Algoritmo. Conjunto de instrucciones que, expresadas según determinada lógica y orden, permiten obtener la solución a un problema determinado.

Código fuente. Programa escrito en un lenguaje de alto nivel, más cercano al lenguaje que entiende un humano que al que entiende un microprocesador, que finalmente es compilado para generar un código máquina que pueda ser ejecutado o interpretado.

Código máquina o código objeto. Conjunto de códigos que son interpretados directamente por un microprocesador para que sea ejecutado y que son dependientes de este último.

Compilador. Complejo programa encargado de la traducción de código fuente a código objeto o código máquina. El proceso de compilación se divide en varias fases: análisis léxico, análisis sintáctico, análisis semántico, generación del código intermedio, optimización del código, detección de errores, generación de código... Finalmente también hace uso de otros programas que son el enlazador y el ensamblador.

CORBA. Siglas de *Common Object Request Broker Architecture*. Estándar que permite a programas que se ejecutan en distintas ubicaciones físicas interactuar entre sí incluso cuando están desarrollados con diferentes lenguajes de programación.

Depurador. También conocido como *debugger*, es un módulo empleado para la detección de errores y la optimización del código fuente.

Enlazador. También conocido como *linker*, se ocupa de juntar el código objeto generado por el compilador, los recursos referenciados desde el código fuente y las bibliotecas utilizadas, eliminando los recursos no utilizados.

Ensamblador. Programa empleado para traducir código ensamblador en código máquina.

Heap. Zona de memoria dinámica en la que se almacenan los objetos que se van creando cuyas referencias son almacenadas en el STACK.

IDE. Entorno de desarrollo integrado que facilita la creación, mantenimiento y depuración de programas.

Índice TIOBE. Informe mensual que elabora y publica la empresa TIOBE Software BV sobre la cuota de mercado de los diferentes lenguajes de programación.

JSP. Acrónimo de *JavaServer Pages*. Tecnología Java empleada para el desarrollo de páginas web dinámicas cuyo contenido se genera mediante la inclusión de etiquetas con código en el código HTML.

JVM. Acrónimo de *Java Virtual Machine* (máquina virtual de Java).

Lenguaje de programación. Conjunto de signos que, ordenados conforme a determinadas reglas, permiten la escritura de programas. Aunque existen lenguajes de programación que, por sus características, pueden ser empleados para la creación de programas de muy diverso ámbito, otros están claramente dirigidos a la ejecución de tareas muy concretas.

Open source. Término traducido como *código abierto o fuente abierta*, que se refiere a aquello que puede ser modificado y compartido porque su contenido y diseño es de acceso público.

RMI. Acrónimo de *Java Remote Method Invocation*. Mecanismo que permite invocar a través de la red métodos pertenecientes a objetos que están ejecutándose en otra ubicación.

Servlet. Pequeño programa Java que se ejecuta en un servidor con la finalidad de atender peticiones y devolver respuestas de contenido dinámico.

Shell. Intérprete de comandos que sirve de interfaz entre el sistema operativo y el usuario.

Shortcut. Combinación de teclas que representa un atajo de teclado.

Stack. Zona de memoria estática en la que se almacenan las referencias a los objetos que se van creando y que se almacenan en el *Heap*.

Webservice. Conjunto de protocolos y tecnologías diseñadas para el intercambio de información entre diferentes máquinas a través de una red de telecomunicaciones.

Zip. Formato y algoritmo de compresión sin pérdida de información.

3.1. Introducción

Hoy en día los sistemas informáticos y los programas que sobre estos se ejecutan están presentes en todo lo que nos rodea. En muy poco tiempo se ha pasado de realizar una gran cantidad de tareas de forma manual a realizarse sobre programas informáticos. Pero no solo ha supuesto un cambio en los procesos ya existentes, sino que también han encontrado su sitio en nuevos escenarios. Todo ello promovido por las mejoras en las infraestructuras de telecomunicaciones, las innovaciones y el abaratamiento en los componentes de equipos informáticos, así como la mejora de las prestaciones de estos últimos.

Antes de llegar a este punto, ha sido necesario recorrer un largo camino en el que muchas tecnologías han sido desarrolladas y han evolucionado conforme a las necesidades. Como consecuencia de esta evolución se ha cambiado el enfoque con el que se analizan los problemas, se han mejorado las técnicas con las que se proponen sus soluciones y, por consiguiente, se han actualizado los lenguajes de programación a utilizar.

Lo que comenzó siendo una secuencia de ceros y unos, interpretados por un sistema para generar un resultado y que fue denominado *código máquina*, fue evolucionando hacia secuencias de palabras siguiendo cierto orden lógico que generaban el mismo resultado y que facilitaban el desarrollo de dichos programas. Podríamos hablar por tanto de distintas generaciones de lenguajes de programación.

3.2. Lenguajes de programación

Como ya se ha mencionado en el apartado introductorio, existe multitud de lenguajes de programación y la gran mayoría ha sufrido su propia evolución a lo largo del tiempo. No todos los lenguajes tienen las mismas características ni tampoco han sido creados para el mismo propósito. Es por ello por lo que se les puede clasificar en función de varios aspectos en distintas categorías:

- *Según generación*: distinguiéndose entre lenguajes de primera generación, segunda generación, tercera generación, cuarta generación y quinta generación.
- *Según su tipo de ejecución*: se pueden encontrar lenguajes interpretados y lenguajes compilados.
- *Según su propósito*: diferenciándose entre lenguajes de propósito general y de propósito específico.

www

Recurso digital

En el anexo web 3.1, disponible en www.sintesis.com y accesible con el código indicado en la primera página del libro, encontrarás información sobre la clasificación de los lenguajes de programación.

3.3. Java

Aunque no es la primera vez que se habla de Java en este libro, sí será en este apartado en el que se realizará una descripción detallada de qué es, en qué consiste y de por qué se ha elegido como tecnología vehicular sobre la cual aprender programación.

3.3.1. Descripción de la tecnología

Java es uno de los lenguajes de programación más populares, que se puede utilizar libremente y que cuenta con la mayor demanda en el mercado según el Índice TIOBE. Este potente lenguaje permite programar casi cualquier tipo de aplicación (juegos, aplicaciones web, aplicaciones de escritorio, servicios...) y está compuesto por varias tecnologías.



SABÍAS QUE...

Java fue diseñado inicialmente para utilizarse en electrodomésticos y, aunque no tuvo el éxito esperado, debido a su buen diseño y características, se decidió utilizarlo en el desarrollo de aplicaciones para ordenadores.

Tal fue el impacto de la tecnología Java en el mundo de la programación, que rápidamente desencadenó el nacimiento de nuevas tecnologías basadas en ella. *Microsoft Visual J++*, *J#* o *Processing* son algunas de las tecnologías que han surgido a partir de Java.

No debe confundirse Java con *JavaScript*, puesto que la única relación que guardan es la de su sintaxis.

Java fue anunciado públicamente el 23 de mayo de 1995, pero no fue hasta el 23 de enero del 1996 cuando se publicó la primera versión del JDK (*JDK 1.0*) de la mano de *Sun Microsystems*, adquirida hace casi una década por *Oracle*, con la intención de facilitar el desarrollo de programas sin tener en cuenta el hardware final ni el sistema operativo sobre el cual se ejecutaría. De esta manera, a partir de los programas desarrollados, se generaba un *código intermedio* que podía ejecutarse indistintamente en un sistema u otro que dispusiera del entorno de ejecución de Java (JRE). Todo ello facilitaba enormemente la portabilidad de programas reduciendo costes, minimizando el tiempo de implantación y eliminando la necesidad de desarrollar nuevas versiones de los programas para cada nueva plataforma.

Como ya se ha mencionado anteriormente, Java puede ser empleado para el desarrollo de aplicaciones de muy diverso tipo. Debido a ello hay una gran variedad de distribuciones disponibles, las cuales han ido evolucionando para adaptarse a las nuevas necesidades:

- ✓ *J2ME (Java 2 Micro Edition)* es la distribución destinada al desarrollo de software para dispositivos de recursos más limitados (electrodomésticos, TDT, impresoras, teléfonos móviles, PDA...). Tuvo mucho éxito debido al auge de los juegos en teléfonos móviles, aunque su uso ha decrecido notablemente.
- ✓ *J2SE (Java 2 Standard Edition)* es la distribución destinada al desarrollo de aplicaciones de escritorio para ordenadores personales o servidores, así como al desarrollo de apli-

caciones cliente-servidor y conexiones con bases de datos (JDBC). Aunque no engloba completamente a J2ME, sí cuenta con una gran parte de las librerías en él contenidas.

- ✓ **J2EE** (*Java 2 Enterprise Edition*) es la distribución más completa de las tres y expande la funcionalidad de J2SE con soporte y API para el desarrollo de aplicaciones basadas en tecnologías web (*JSP*, *Servlets*, *WebServices...*), invocación de métodos remotos (*CORBA*, *RMI*)...

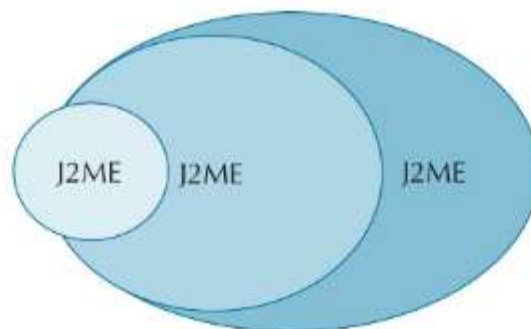


Figura 3.1
Distribuciones Java.

Actividades propuestas



- 3.1.** Clasifica la tecnología Java en las diferentes categorías expuestas en el apartado 3.2. Lenguajes de programación.
- 3.2.** Uno de los puntos fuertes de Java es que se trata de una tecnología que se puede utilizar de forma gratuita, pero no siempre siguió fielmente la filosofía *open source*. Esta situación fue bautizada como “la trampa de Java”.

Lea el artículo “Libre pero encadenado. La trampa de Java”, de Richard Stallman, publicado en GNU.org (accesible a partir del código QR adjunto) y busca información sobre cuál fue la situación inicial de Java con respecto al software libre y cuáles fueron los argumentos de peso que hicieron que la máquina virtual de Java no viniese incluida por defecto en las distribuciones GNU Linux.



3.3.2. Entorno de desarrollo JDK

Para poder desarrollar aplicaciones basadas en Java será necesario utilizar uno de los entornos de desarrollo suministrados por *Oracle*. Este entorno de desarrollo ofrecerá las herramientas necesarias para construir y ejecutar aplicaciones de casi cualquier tipo.

El entorno de desarrollo a emplear será el *JDK* (*Java Development Kit*). Es recomendable elegir la última versión estable disponible, pero es probable que dicha versión presente algún tipo de incompatibilidad con herramientas que se necesiten instalar posteriormente (*IDE*).

Es importante destacar que existen dos tipos de *JDK* suministrados por *Oracle*, que, aun siendo similares y ofreciendo la misma funcionalidad, tienen distinta licencia:

- El primero de ellos es el JDK suministrado bajo licencia *Binary Code License Agreement for the Java SE Platform Products and JavaFX*.
- El segundo es el *OpenJDK* que es suministrado bajo licencia *GNU General Public License, version 2*.

Para ejecutar las aplicaciones desarrolladas con el JDK, independientemente de la variante usada, también será necesario el JRE (*Java Runtime Environment*). Este vendrá suministrado directamente junto con el JDK.



PARA SABER MÁS

La primera versión del *Java Development Kit (JDK 1.0)* fue liberada por Sun Microsystems el 23 de enero de 1996 con el lema “Escribe una vez, ejecuta en todas partes”.

La última versión estable en el momento en que se está redactando este texto es la *JDK 10* y *OpenJDK 10* pero ambas son incompatibles con las últimas versiones estables de los IDE que se emplearán en el curso.

3.3.3. Entorno de ejecución JRE

Los lenguajes de programación de alto nivel ofrecen respecto a los lenguajes de bajo nivel una mayor abstracción sobre la arquitectura física a la hora de escribir programas. Para que se puedan obtener programas a partir de ellos es imprescindible la utilización de compiladores.

Originalmente los compiladores llevaban a cabo la traducción de los programas escritos, en un lenguaje de programación cercano al ser humano, a un código escrito en un lenguaje entendible por el sistema operativo sobre el cual se pretendía ejecutar. Algunos de los lenguajes que trabajan de esta manera son C, Pascal o Delphi.

Sin embargo, otras tecnologías muy relevantes en el mundo del desarrollo utilizan compiladores que traducen esos programas a un código intermedio que, por sí mismo, no puede ser ejecutado directamente por el sistema operativo. Ese código intermedio debe ser manejado por un elemento especial que se ejecuta sobre el sistema operativo y cuya función es traducir el código intermedio a instrucciones nativas entendibles por el sistema operativo.

En el mundo Java ese código intermedio se denomina *bytecode* y el elemento que hace de intermediario para que pueda ser ejecutado se denomina JRE (*Java Runtime Environment*). Este está formado por una JVM (*Java Virtual Machine*), bibliotecas con código Java y algunos componentes extra. Existen versiones para la mayor parte de los sistemas operativos del mercado, lo cual garantiza que el mismo código pueda ser ejecutado en diferentes sistemas sin necesidad de hacer modificaciones.

De forma análoga en el mundo .NET el “código intermedio” se denomina *CIL* o *MSIL* y la máquina virtual que lo traduce CLR (*Common Language Runtime*).

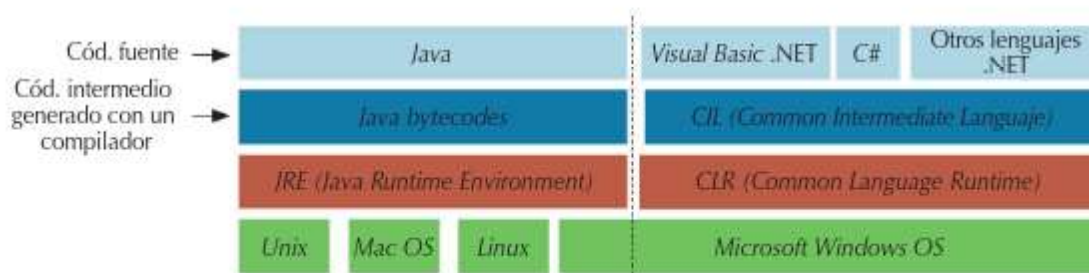


Figura 3.2
Java versus .NET.

Dado que la ejecución del código intermedio implicaría su traducción una y otra vez a lo largo de la vida del programa y esto ralentizaría en gran medida el tiempo de ejecución, se decidió incluir en el JRE un compilador JIT (*Just In Time*) que mejora notablemente el rendimiento de las aplicaciones y minimiza este retardo. Este compilador hace uso del código compilado con anterioridad para evitar interpretarlo nuevamente, además de otras acciones llevadas a cabo en pro de la mejora del rendimiento.



SABÍAS QUE...

Aunque Java fue diseñado para ejecutar sus programas sobre una máquina virtual, se han desarrollado compiladores que permiten generar ejecutables nativos que se pueden lanzar sobre el sistema operativo sin necesidad del JRE. Un ejemplo es GCJ (*GNU Compiler for Java*).

3.3.4. Variables de entorno PATH y CLASSPATH

A la hora de trabajar con Java es importante tener presente que hay dos variables de entorno cuyos valores tienen que estar correctamente configurados para permitir llevar a cabo la compilación, ejecución y generación de paquetes *jar* de una forma mucho más cómoda:

- **PATH:** es la variable del sistema utilizada por el sistema operativo para buscar los ejecutables de las diferentes aplicaciones desde el intérprete de comandos de MS-DOS o el terminal de Linux.
- **CLASSPATH:** es la variable del sistema utilizada por el entorno de ejecución de Java (JRE) para buscar los paquetes y clases definidas por el usuario.

Si se desea trabajar desde la línea de comandos o terminal, es de vital importancia configurar correctamente ambas variables para facilitar el trabajo.

Actividad propuesta 3.3



Accede a la página de descargas de *Oracle*, descarga la versión para su sistema operativo del *JDK 8* e instálalo. Por último, lleva a cabo la configuración de las variables de entorno *PATH* y *CLASSPATH*.

Recurso digital

www

En los anexos web 3.2 y 3.3, disponibles en www.sintesis.com y accesibles con el código indicado en la primera página del libro, encontrarás información sobre la instalación y configuración el entorno.

3.3.5. Tipos de archivos empleados

Antes de comenzar a programar en Java, es necesario conocer los diferentes tipos de archivos que se van a manejar, así como las extensiones asociadas a estos:

- Cuando se escribe *código fuente* en el lenguaje de programación Java, este debe almacenarse en un archivo de texto plano y con extensión *.java*.
- Como resultado de compilar exitosamente un programa escrito en un archivo *.java* obtendremos, como mínimo, un archivo contenedor de código intermedio (*bytecode*) con extensión *.class*.
- A diferencia de otras tecnologías en las que el resultado de una correcta compilación es un archivo ejecutable, Java genera un archivo independiente por cada clase del proyecto con extensión *.class*. Esto puede hacer muy incómodo el manejo de los programas en aquellos proyectos que contengan un gran número de clases. Por este motivo Java ofrece la posibilidad de empaquetar ese conjunto de archivos *.class* y los recursos que estos utilizan (imágenes, archivos de texto plano...) en un único archivo con extensión *.jar* (*Java Archive*). Para la construcción de este tipo de archivos se emplea el algoritmo de compresión *Zip*.
- Siguiendo con la misma lógica y para facilitar la implantación de las aplicaciones en los servidores de aplicaciones, Java ofrece los archivos:
 - a) *.war* (*Web Application Archive*): destinados a contener todos los archivos que forman una aplicación web desarrollada con tecnología Java (*JSP*, *Servlets*...) y facilitar su despliegue.
 - b) *.ear* (*Enterprise Archive*): empaquetan uno o varios módulos en un archivo que facilita su despliegue en un único paso sobre un servidor de aplicaciones conforme a la estructura definida en el archivo contenido *META-INF/application.xml*. Puede, por tanto, contener archivos de cualquier tipo (incluidos *.jar* y *.war*).

3.3.6. Recolector de basura

Otro de los elementos importantes y diferenciadores de la tecnología Java es su recolector de basura (*garbage collector*). Este sigue una filosofía opuesta a la que impone la gestión manual de memoria, de manera que exime al programador de las tareas inherentes a su reserva y liberación.

Aquellos programadores provenientes de tecnologías como C o Delphi pueden ver esta característica como una ventaja o como un inconveniente:

- Ventaja porque facilita el desarrollo y evita errores humanos a la hora de gestionar la memoria. Gracias a esta liberación, es posible escribir código con la tranquilidad de que los programas desarrollados no terminarán consumiendo los recursos del sistema antes de que el programa finalice.
- Inconveniente si tenemos en cuenta que este elemento no supone un coste cero y afectará al rendimiento del programa, ya que consume recursos. Este es el motivo principal por el que se hace impensable el desarrollo de sistemas de tiempo real utilizando este tipo de tecnologías. Existen diferentes políticas de manejo del recolector de basura:
 - a) Esperar a que no quede memoria libre y en ese momento ejecutar el recolector de basura.
 - b) Fijar un umbral de ocupación de la memoria libre y, una vez alcanzado, ejecutar el recolector de basura.
 - c) Ejecutar el recolector de basura a intervalos regulares.
 - d) Ejecutar el recolector de basura justo antes de cada reserva de memoria.
 - e) Permitir al programador que invoque explícitamente al recolector de basura cuando quiera.

Son varias las tecnologías que hacen uso del recolector de basura y cada una lo implementa de una forma particular. En el caso de Java, su funcionamiento se basa en la creación de una subrutina que se está ejecutando de forma paralela a los programas y que está controlando la memoria para detectar qué objetos están en uso y cuáles no. Su funcionamiento básico es el siguiente:

- *Marcado*: el recolector de basura identificará las zonas de memoria que ya no están en uso para su posterior liberación. Para ello identificará en el *Heap* las zonas de memoria de objetos que ya no tienen su correspondiente referencia en el *Stack*.
- *Borrado de objetos no referenciados*: el recolector de basura liberará las zonas de memoria en las que anteriormente había objetos que han quedado sin referencias que los apunten.
- *Compactación del espacio usado*: una vez liberado el espacio quedarán huecos en la memoria utilizada. El recolector de basura compactará el espacio usado para optimizar la ubicación en memoria de nuevos objetos que se vayan creando a lo largo de la ejecución.



Figura 3.3
Funcionamiento
del recolector
de basura (Java).

Recurso web

www

En el apartado 3.3 se ha explicado de forma muy básica cómo trabaja el recolector de basura de Java, pero este mecanismo es mucho más complejo. Para conocer con más precisión su funcionamiento, escanea el código QR adjunto.



3.4. Programación sin IDE

En los apartados 3.3.2 y 3.3.3 se han presentado las características de los entornos de desarrollo y ejecución para Java (JDK y JRE respectivamente), que suponen junto con un editor de texto plano los requisitos mínimos con los que afrontar el desarrollo de un proyecto. Estos podrán manejarse tanto desde un IDE con interfaz gráfica como desde línea de comandos.

El manejo de Java desde línea de comandos no entraña ninguna dificultad. Dado que los comandos serán equivalentes para cualquier sistema operativo, únicamente se mostrará su uso en uno de ellos y, por tanto, no se entrará en detalle acerca de cómo realizar las mismas tareas desde otro sistema.

3.4.1. Compilar un programa

Una vez se haya escrito el código fuente en un archivo de texto plano, se utilizará el comando *javac* para convertirlo a código intermedio. La sintaxis de uso del comando es la siguiente:

```
javac <opciones> <ficheros>
```

En el ejemplo mostrado en la figura 3.4 se pueden ver instrucciones equivalentes que generan el mismo resultado.



Figura 3.4
Compilación de programas desde línea de comandos.

En este caso se han creado cinco archivos con extensión *.class* como resultado del proceso de compilación. El nombre de cada archivo *.class* generado será el de la clase contenida dentro de cada archivo de código fuente compilado exitosamente. El código que contendrá deberá ser interpretado por la JRE.

Actividad propuesta 3.4



Ejecuta el comando `javac -help` desde un intérprete de comandos de MS-DOS o desde un terminal de Linux y estudia su sintaxis de uso.

3.4.2. Crear un jar

Una vez se hayan generado los archivos de código intermedio y se disponga de los recursos que estos utilizarán, se podrá generar un archivo `.jar` que los comprima para facilitar su transporte, almacenaje o inclusión en nuevos proyectos.

La sintaxis para emplear para el uso del comando `jar` es compleja, pues dispone de varias opciones:

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

Estos archivos podrán construirse con las opciones `-cvf` para indicar que se quiere crear un archivo nuevo con un nombre determinado. Podrán construirse como:

- ✓ *Simple contenedores* de recursos que serán utilizados en otros proyectos; en este caso bastaría con generar el archivo `.jar`.

```
C:\Windows\system32\cmd.exe
C:\programas>jar -cvf jarClasesSimple.jar Clase3.class Clase4.class Clase5.class
manifesto agregado
agregando: Clase3.class(entrada = 186) (salida = 155)(desinflado 16%)
agregando: Clase4.class(entrada = 186) (salida = 155)(desinflado 16%)
agregando: Clase5.class(entrada = 186) (salida = 155)(desinflado 16%)
C:\programas>
```

Figura 3.5

Creación de `jar` contenedor.

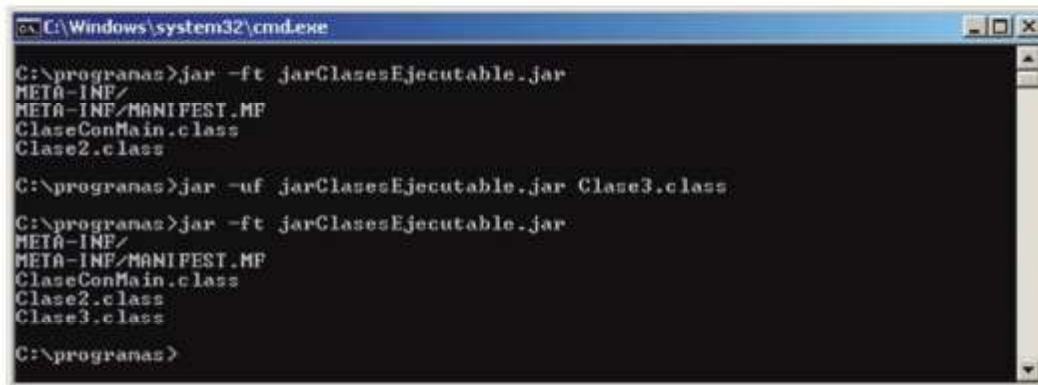
- ✓ *Ejecutables comprimidos*. En este caso sería necesario añadir la opción `-e` además de indicar qué clase será la principal.

```
C:\Windows\system32\cmd.exe
C:\programas>jar -cvfe jarClasesEjecutable.jar ClaseConMain ClaseConMain.class C
lase2.class
manifesto agregado
agregando: ClaseConMain.class(entrada = 449) (salida = 300)(desinflado 33%)
agregando: Clase2.class(entrada = 186) (salida = 155)(desinflado 16%)
C:\programas>
```

Figura 3.6

Creación de `jar` ejecutable.

Sobre cualquier archivo *.jar* se podrá añadir (opción *-u*) o consultar contenido (opción *-t*), aunque no existen opciones para eliminar elementos contenidos en él y esta tarea debe realizarse con otras herramientas (WinZip, WinRAR, 7zip...).



```

C:\Windows\system32\cmd.exe

C:\programas>jar -ft jarClasesEjecutable.jar
META-INF/
META-INF/MANIFEST.MF
ClaseConMain.class
Clase2.class

C:\programas>jar -uf jarClasesEjecutable.jar Clase3.class

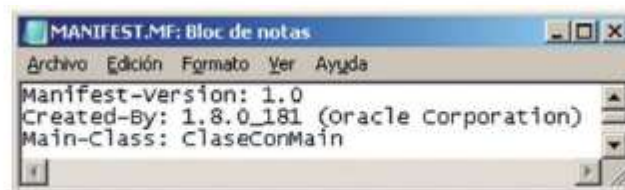
C:\programas>jar -ft jarClasesEjecutable.jar
META-INF/
META-INF/MANIFEST.MF
ClaseConMain.class
Clase2.class
Clase3.class

C:\programas>
    
```

Figura 3.7

Consulta y modificación de contenidos de un *jar*.

Cuando estos archivos se construyen, dentro del archivo contenedor se creará un directorio *META-INF* en el que se podrá encontrar el archivo de texto plano *MANIFEST.MF*. Este archivo contiene cabeceras con información importante relativa al archivo *jar* (por ejemplo, la clase que se haya especificado principal en un ejecutable).



```

MANIFEST.MF: Bloc de notas
Archivo Edición Formato Ver Ayuda
Manifest-Version: 1.0
Created-By: 1.8.0_181 (Oracle Corporation)
Main-Class: ClaseConMain
    
```

Figura 3.8

Contenido del fichero *MANIFEST.MF* de un *jar*.



Actividad propuesta 3.5

Ejecuta el comando *jar* desde un intérprete de comandos de MS-DOS, o desde un terminal de Linux, y estudia su sintaxis de uso.

3.4.3. Ejecutar un programa

Una vez se haya generado el código intermedio y se disponga de un archivo *.class* o de un *jar* ejecutable, podrá lanzarse sobre la JRE para que esta lo interprete y ejecute. La sintaxis de uso del comando es la siguiente:

```
java [opciones] <clase principal> [argumentos...]
```



```
CA: C:\Windows\system32\cmd.exe
C:\programas>javac HolaMundo.java
C:\programas>java HolaMundo
Hola Mundo !
C:\programas>java -jar jarClasesEjecutable.jar
Ejecutando desde ClaseConMain !
C:\programas>
```

Figura 3.9
Ejecución de programas.

www

Recurso web

En la URL <https://compiler.javatpoint.com/opr/test.jsp> puedes encontrar una herramienta online que permite compilar y ejecutar programas sin necesidad de instalar ninguna aplicación.

Actividad propuesta 3.6



Ejecuta el comando `java -help` desde un intérprete de comandos de MS-DOS o desde un terminal de Linux y estudia el resultado.



INVESTIGA

Averigua cómo ejecutar programas Java ubicados en el directorio `C:/programas` sobre un sistema operativo Microsoft Windows, o `/programas` si trabaja en Linux, sin necesidad de crear la variable de entorno `CLASSPATH`.

3.5. Programación con IDE

A lo largo de los apartados anteriores se han descrito las características principales de los entornos de desarrollo y ejecución (JDK y JRE, respectivamente), mostrándose cómo hacer uso de estas herramientas desde línea de comandos.

Como seguramente se ha podido comprobar, llevar a cabo el desarrollo de proyectos de gran envergadura utilizando únicamente la línea de comandos puede ser una tarea ardua y propensa a cometer errores. Por este motivo surgieron los entornos de desarrollo integrado.

3.5.1. Características y elementos

Este tipo de aplicaciones, también conocidas como IDE (*Integrated Development Environment*), son un conjunto de herramientas que han sido generadas para facilitar el desarrollo de programas y que se suministran de forma conjunta.

Generalmente este tipo de paquetes ofrecen un ambiente de desarrollo basado en aplicaciones gráficas desde las que es muy fácil acceder a todas las herramientas, la mayor parte de ellas no disponibles en un simple JDK.

Las herramientas que, de forma general, se pueden encontrar en los IDE son un editor, un ensamblador, un compilador, un enlazador, un depurador, asistentes que facilitan el desarrollo, un gestor de conexiones a diferentes orígenes de datos...

3.5.2. Alternativas

Existen varias alternativas en el mercado tanto de pago como libres. Algunas están orientadas al desarrollo en un único lenguaje de programación mientras que otras se adaptan al desarrollo en varias tecnologías.

Entre todas ellas hay que destacar *Eclipse* y *NetBeans*. Ambos pueden ser instalados en sistemas operativos Microsoft Windows y Linux (entre otros), se pueden descargar libremente, tienen soporte para desarrollar en varios lenguajes...

Otras alternativas son:

- *IntelliJ IDEA*: disponible para sistemas operativos Microsoft Windows y Linux. Puede incorporar soporte para varios lenguajes de programación. Está desarrollado en Java y dispone de versiones gratuitas y de pago.
- *JCreator*: únicamente hay versiones para el sistema operativo Microsoft Windows. Está desarrollado en C++, lo que lo hace muy recomendable para máquinas con escasos recursos. Tiene una versión gratuita y otra de pago.

Recurso digital

www

En el anexo web 3.4, disponible en www.sintesis.com y accesible con el código indicado en la primera página del libro, encontrarás información sobre la instalación de *NetBeans*.



Actividad propuesta 3.7

Accede a la página de descargas de *NetBeans*, descarga la versión correspondiente a su sistema operativo e instálalo.



INVESTIGA

Aunque este libro propone la instalación de *NetBeans* como IDE de trabajo, posiblemente consideres que otra de las alternativas propuestas se adecúa mejor a tus necesidades. Investiga y profundiza sobre las características de los IDE presentados, así como de otros que consideres oportuno, y debate en clase con tus compañeros sobre cuál es el más apropiado.

3.5.3. Configuración de NetBeans

Las configuraciones para realizar sobre el IDE pueden ser de tipo general, afectando a todos los proyectos, o de tipo específico, afectando únicamente a un proyecto. Aunque en el capítulo siguiente se explicará en detalle qué es un proyecto, en este momento se puede entender como un conjunto de recursos que representan al programa que se va a construir y que están organizados en una carpeta.

A) Configuraciones generales

Para hacer configuraciones generales hay que acceder a *Tools > Options*.



Figura 3.10
Acceso a opciones generales en *NetBeans*.

El primer paso será activar las características Java para que el entorno se quede preparado para trabajar con esta tecnología. Para ello activar el soporte para Java desde la opción Java (pestaña *GUI Builder* o *Java Debugger*).



Figura 3.11
Activación de características.

Automáticamente se activarán todas las herramientas del IDE necesarias para el desarrollo de nuestros proyectos y se habrán habilitado también nuevas configuraciones generales hasta ahora ocultas.



Figura 3.12
Nuevos accesos a herramientas.



Figura 3.13
Nuevas configuraciones habilitadas.

RECUERDA

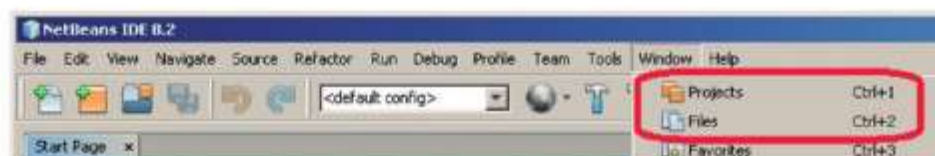
- ✓ Las configuraciones establecidas por defecto generalmente son buenas para casi cualquier desarrollador y proyecto, por lo que es recomendable no modificarlas.

A continuación, se describen las configuraciones generales más interesantes:

- ✓ *Editor*: establecerá cómo realizar las tabulaciones en el código (*Formatting*), cómo trabajará el autocompletado (*Code Completion*), cómo se optimizará y corregirá el código (*On Save*), los tipos de mensajes de error o advertencia mostrados por el entorno (*Hints*)...
- ✓ *Fonts & Colors*: desde esta opción es posible configurar la tipografía y la simbología empleada por el editor.
- ✓ *Keymap*: usado principalmente para establecer *shortcuts*.
- ✓ *Java*: permite configurar el comportamiento del constructor (*GUI Builder*) y del depurador (*Java Debugger*).
- ✓ *Appearance*: desde esta opción será posible configurar el aspecto del IDE (separadores de documentos, ventanas y aspecto).

También es recomendable mostrar las ventanas *Projects* y *Files* (proyectos y ficheros, respectivamente) para tener acceso desde el IDE a la estructura del proyecto.

Figura 3.14
Acceso a ventanas
Projects y *Files*.



Actividad propuesta 3.8



Configura el entorno de trabajo para que muestre las ventanas *Projects* y *Files*.

B) Configuraciones específicas de cada proyecto

Todos los proyectos recién creados atenderán a las configuraciones que se hayan establecido de forma general en el *IDE*. Sin embargo, es posible matizar estas configuraciones para realizar los cambios oportunos sobre algunas de ellas dejando intactas las demás.

Para acceder a las configuraciones específicas de un proyecto, será necesario hacer clic derecho sobre el proyecto y seleccionar la opción *Properties*. Aparecerá una nueva ventana desde la que podrán configurarse las propiedades del proyecto (algunas de ellas similares a las configuradas de forma general y otras totalmente específicas).

Dado que aún no se ha descrito cómo crear un proyecto, se recomienda retroceder a este apartado una vez se haya completado el apartado 3.5.4.

3.5.4. Manejo del IDE

A) Crear un proyecto

Una vez se ha realizado la instalación y configuración del IDE, ya se puede comenzar a escribir el primer programa. *NetBeans* está diseñado para trabajar por proyectos y, por tanto, aunque permite crear archivos independientes de algunas tecnologías como *HTML* o *JavaScript*, el primer programa deberá ser creado dentro de un proyecto.

Como se puede observar, *NetBeans* ofrece en su barra de tareas iconos para las acciones más habituales. Para crear el proyecto se debe hacer clic en el icono de la carpeta. También se puede hacer uso del atajo de teclado correspondiente.

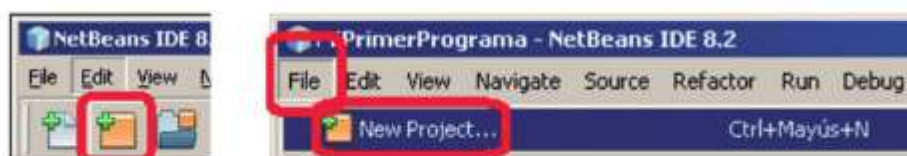


Figura 3.15
Creación de proyectos.

Automáticamente se abrirá un asistente que guiará el proceso. Se debe seleccionar *Java Application* y, en la ventana siguiente, establecer el nombre y el directorio del proyecto. Será posible crear la clase principal automáticamente dentro del proyecto si se dejó el checkbox *Create Main Class* marcado.



Figura 3.16
Selección del tipo de proyecto.

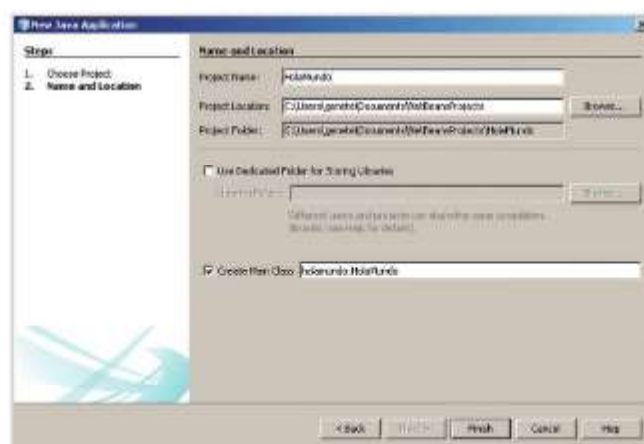


Figura 3.17
Establecimiento de las propiedades del proyecto.

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

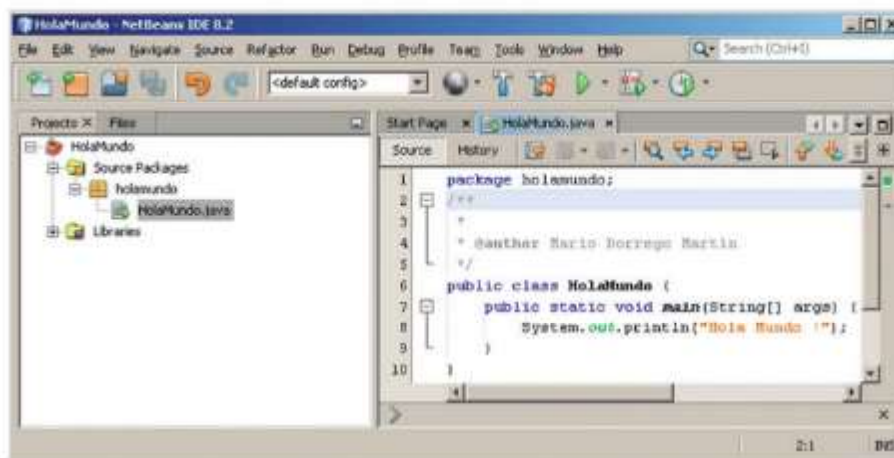


Figura 3.18
Proyecto HolaMundo.

B) 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.



Figura 3.19
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.

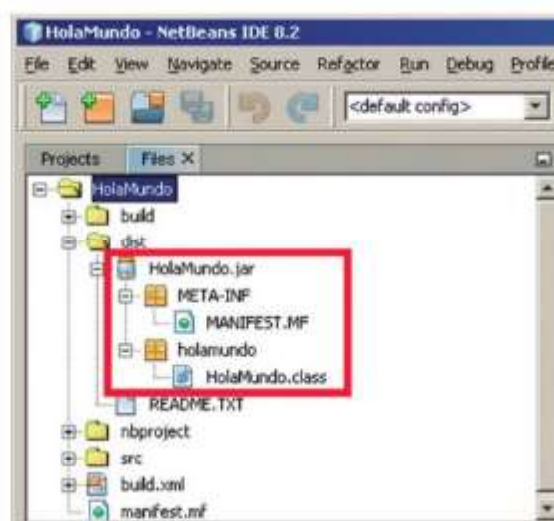


Figura 3.20
Estructura de directorios del proyecto.

C) 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*.

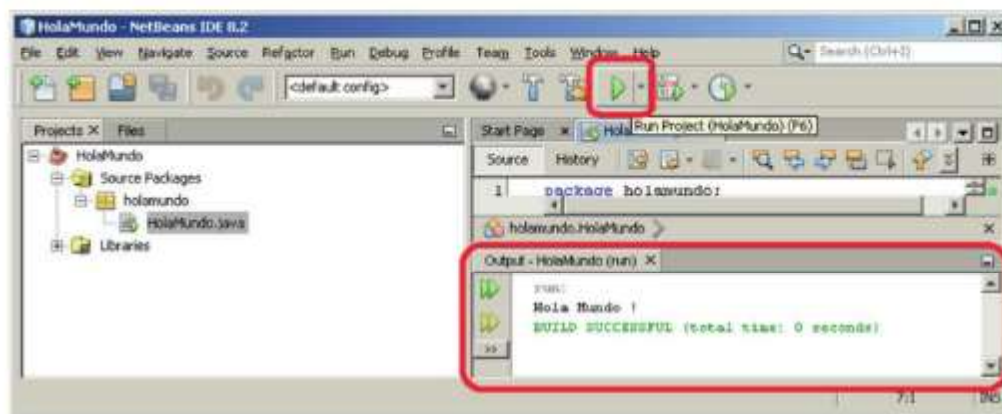


Figura 3.21
Ejecución de proyectos.

D) 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:

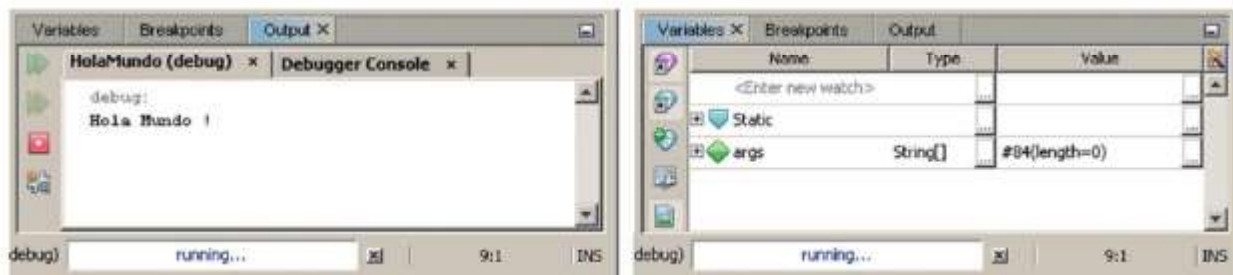
- *Ctrl + F5*: comenzar el *Debug*.
- *Mayús + F5*: detener el *Debug*.
- *F5*: continuar con la ejecución hasta el siguiente punto de ruptura o hasta el final del programa.
- *F4*: ejecutar el programa hasta la posición actual del cursor.
- *F7*: ejecutar la siguiente línea del programa. Si la siguiente instrucción es la llamada a un 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 a un 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*).

Otra vía de acceso a las mismas acciones es la barra de tareas con iconos (figura 3.22).



Figura 3.22
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*.

**Figura 3.23**

Ventanas resultado de la depuración e inspección de 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.

Resumen

- Hoy en día existe una gran variedad de lenguajes y tecnologías con los que afrontar el desarrollo de un proyecto.
- Los lenguajes de programación pueden clasificarse según varios aspectos.
- Algunos lenguajes no se ejecutan directamente sobre el sistema operativo, sino que requieren de un elemento intermedio o máquina virtual.
- Java es uno de los lenguajes más utilizados en la actualidad y dispone de recursos para el desarrollo de aplicaciones de casi cualquier tipo.
- Para trabajar en Java es necesario el JDK y el JRE, así como unas configuraciones mínimas del sistema.
- La utilización de un IDE facilita el desarrollo de programas y minimiza los errores.



Ejercicios propuestos

1. Aunque la primera versión del JDK fue liberada en 1996, los orígenes de Java se remontan a un periodo anterior. Investiga cuál fue el punto de partida del proyecto e indica los diferentes nombres que fue tomando hasta bautizarse como Java.
2. Ordena los lenguajes de programación que han sido citados en el capítulo por orden cronológico.
3. ¿Utilizarías Java para el desarrollo de un sistema de tiempo real? Justifica tu respuesta.
4. Describe las diferencias existentes entre la arquitectura Java y .NET.
5. Investiga las diferencias existentes en la forma de trabajar de un compilador y un intérprete de código.
6. Enumera y describe tres características de Java que hacen que se trate de una buena alternativa a la hora de afrontar un proyecto.
7. Averigua cómo reestablecer la disposición de las ventanas del IDE *NetBeans* a su posición por defecto.