APÉNDICE

A Plataformas para Java

Este apéndice ilustra los pasos requeridos para compilar un programa escrito en Java usando tres entornos diferentes:

- *JDK* de Sun.
- *Symantec Café*.
- *Microsoft Visual J++*.

Supondremos que ya se ha realizado una instalación básica.

A.1 Estableciendo el entorno

Independientemente del sistema de desarrollo que utilicemos, necesitamos asegurarnos de que se les ha asignado un valor adecuado a las siguientes dos variables de entorno:

- `PATH`: especifica un conjunto de directorios donde se buscan los ejecutables (como *javac*, *java* y *javadoc*).
- `CLASSPATH`: especifica un conjunto de directorios donde se buscan los paquetes (como `java.lang` y `EstructurasDatos`).

Si se asigna a `PATH` un valor inadecuado, podría no encontrarse alguno de los ejecutables, como *javac*. Si se asigna una valor inadecuado a `CLASSPATH`, entonces fallarán las directivas `import` y otros usos de los paquetes.

Algunos de los productos descritos en esta sección asignan automáticamente valores a las variables de entorno al ser instalados. Aun así, para usar los programas del libro, necesitará modificar `CLASSPATH` para incluir los paquetes aquí descritos.

Las instrucciones para Unix y Windows95 son básicamente idénticas, excepto en lo que se refiere a su sintaxis concreta.

A.1.1 Instrucciones para Unix

Las siguientes instrucciones Unix suponen que estamos usando una C-shell.

1. Para comprobar si la variable `PATH` tiene un valor adecuado se usa el mandato

   ```
   which javac
   ```

   Si el sistema responde con un nombre (por ejemplo, `/bin/javac`), entonces tiene un valor adecuado, pero si responde con un mensaje diciendo

que no lo encuentra (proporcionando después la lista de lugares donde lo buscó), entonces debe ser actualizada.

2. Pregunte a su administrador de sistema dónde se encuentra situado el compilador de Java. Supongamos por ejemplo que se encuentra en el directorio `/home/jdk/bin`. Editamos su fichero `.cshrc` y añadimos una línea para actualizar la variable. En nuestro ejemplo, escribiríamos

   ```
   set path= ( $path /home/jdk/bin )
   ```

3. A continuación actualizamos la variable `CLASSPATH`. Si ya tenemos una variable `CLASSPATH`, probablemente tendrá correctamente asignado el valor necesario para acceder a los paquetes del sistema. Se puede ver el valor actual escribiendo

   ```
   echo $CLASSPATH
   ```

   Incluso aunque tenga un valor adecuado, tendremos que actualizarla para usar los paquetes del libro. Si no posee una variable `CLASSPATH`, pregunte al administrador del sistema dónde se encuentran situados los ficheros de librerías del sistema. Entonces podríamos escribir por ejemplo

   ```
   setenv CLASSPATH .:/jdk/lib/classes.zip
   ```

   Observe que ordenamos buscar en dos sitios: el directorio actual (.) y en un fichero comprimido que contiene las clases de Java.

4. Añadimos al final de `CLASSPATH` el directorio que contiene los paquetes vistos en este libro. Por ejemplo

   ```
   setenv CLASSPATH .:/jdk/lib/classes.zip:$HOME/libro
   ```

   El nombre del directorio real variará en cada instalación. Observe que los cambios realizados en el fichero `.cshrc` no se hacen efectivos hasta la siguiente vez que entramos en el sistema. Sin embargo, podemos hacer que los cambios se hagan efectivos inmediatamente ejecutando el mandato

   ```
   source .cshrc
   ```

A.1.2 Instrucciones para Windows95/NT

WindowsNT se comporta de la misma forma que Windows95, excepto por el hecho de que el proceso se puede simplificar editando las variables de entorno en el Panel de Control. (Seleccione *Panel de Control* y luego *Sistema*.) Las instrucciones para Windows95 se muestran a continuación.

1. Comprobamos que la variable `PATH` tiene un valor correcto. El mandato

   ```
   javac
   ```

   intenta invocar al compilador de Java. Si el sistema responde con un mensaje de error que detalla la sintaxis correcta del mandato *javac* (por ejemplo, especificando una lista de opciones), entonces la variable tiene un valor correcto. Si el sistema responde con un mensaje diciendo que no encuentra *javac*, se debe actualizar la variable `PATH`.

2. Para actualizarla, necesitamos saber dónde se encuentra situado el compilador de Java (puede usar las utilidades de búsqueda para saberlo). Supongamos, por ejemplo, que se encuentra en el directorio `C:\jdk\bin`. Entonces, editamos el fichero `autoexec.bat` (usando el *Bloc de Notas*) y añadimos una línea para actualizar la variable, como la siguiente

   ```
   SET PATH = %PATH%;C:\jdk\bin
   ```

3. A continuación actualizamos la variable `CLASSPATH`. Si ya disponemos de una, probablemente tendrá el valor adecuado para encontrar los paquetes del sistema. Podemos ver los valores de todas las variables de entorno ejecutando el mandato `set` sin parámetros.

   Incluso aunque la variable `CLASSPATH` tenga el valor correcto, tendremos que actualizarla para poder usar los paquetes de este libro. Si no disponemos de una variable `CLASSPATH`, necesitamos saber dónde están situados los ficheros de las clases de las librerías. Una asignación típica sería

   ```
   set CLASSPATH = .:C:\jdk\lib\classes.zip
   ```

   Observe que con ello mandamos buscar en dos lugares: el directorio actual (.) y un fichero comprimido que contiene las clases de Java.

4. Añadimos al final de `CLASSPATH` el directorio que contenga los paquetes de este libro. Por ejemplo, añadiendo una segunda línea

   ```
   set CLASSPATH = %CLASSPATH%;C:\codigolibro
   ```

   El directorio real variará en cada instalación. Observe que los cambios realizados en el fichero `autoexec.bat` no se hacen efectivos hasta la siguiente inicialización del sistema (o en el caso de WindowsNT hasta la siguiente vez que se entra en el sistema).

A.2 JDK de Sun

JDK es un sistema sin florituras. Hemos de introducir nuestro código en ficheros Java usando un editor estándar del sistema. Algunos editores de Unix son *vi*, *pico* y *emacs*; y de Windows, *Bloc de Notas* y *WordPad*.

Para compilar un fichero fuente de Java, usamos el mandato *javac*. Para ejecutar la función `main` de la clase, usamos el mandato *java*. Por ejemplo, supongamos que `main` está en la clase `Ejemplo`. Entonces los mandatos son

```
javac Ejemplo.java
java Ejemplo
```

Para cada clase definida en cada fichero fuente compilado por *javac*, el compilador almacena el código-j resultante en el fichero de clase correspondiente (con el sufijo `.class`). Para cada clase referenciada en los ficheros fuente, el compilador busca en la variable `CLASSPATH` tanto el fichero fuente como el fichero compilado, y recompila el fichero fuente (regenerando por tanto el fichero `.class`) si éste se ha modificado. En otras palabras, *javac* calcula cuál es el menor conjunto de ficheros que hace falta recompilar y lo hace de forma automática.

El mandato *javac* tiene varias opciones. En la documentación para JDK se listan con detalle. En la Figura A.1 se muestran algunas de ellas.

Opción	Resultado
`-O`	Optimiza el código compilado desplegando los métodos estáticos, finales y privados.
`-verbose`	Hace que el compilador y el enlazador impriman mensajes sobre los ficheros que se están compilando y los ficheros ya compilados (o ficheros de clase) que se están cargando.
`-depend`	Hace que el compilador recompile ficheros de clases referenciados desde otros ficheros de clase. Normalmente sólo recompila ficheros de clase obsoletos a los que se hace referencia desde el código fuente.

Figura A.1 Opciones de *javac*.

El mandato *java* también tiene opciones. Se proporciona una lista completa de ellas en la documentación para JDK.

A.3 Entornos de desarrollo visual

Symantec Café se distinguió por ser el primer sistema del mercado con un entorno de desarrollo moderno. *Microsoft Visual J++* se distingue por ser un producto Microsoft. Su apariencia y fundamentos son idénticos al popular Visual C++ de Microsoft.

Ambos sistemas incluyen, entre otras cosas,

- un editor visual que ilumina en diferentes colores las palabras clave, los comentarios y los tokens;
- botones para compilar y ejecutar programas;
- ventanas que proporcionan simultáneamente varias vistas del proyecto, como la salida del compilador, el código fuente y la lista de ficheros del proyecto;
- herramientas de depuración; y
- ayuda durante su uso.

La integración de estas vistas en un único producto implica, por ejemplo, que si se pulsa dos veces sobre un mensaje de error del compilador, aparece iluminada la línea de código donde se ha detectado el error. Una vez haya disfrutado de un sistema visual, le resultará difícil volver a un sistema basado en texto. Una característica importante es el editor de recursos, que permite al programador diseñar un entorno gráfico usando un sistema CAD. El editor de recursos generará código Java que describe el entorno. Esto es útil para el diseño de Interfaces Gráficas de Usuario.

Los sistemas visuales tienen algunas desventajas. En primer lugar, a diferencia del JDK de Sun, no son gratuitos. En segundo lugar, se debe hacer el esfuerzo de crear un proyecto para cada programa, lo cual puede resultar tedioso para programas pequeños que solamente utilizan un fichero fuente. En tercer lugar, la versión más actualizada de Java siempre aparecerá disponible antes para el sistema JDK de Sun. Las actualizaciones aparecerán en los sistemas comerciales poco después, y no serán gratuitas. Por ejemplo, cuando se escribió la versión original de este libro, el único compilador que soportaba Java 1.1 era el JDK 1.1[1].

[1] En este apéndice se habla de los sistemas que estaban disponibles en el momento en que se escribió la versión original de este libro. Las nuevas versiones serán probablemente similares, pero no completamente idénticas.

Este apéndice describe, para ambos entornos, cómo crear un proyecto para la clase `TestPila`. El código fuente de la versión original, `TestStack.java`, se encuentra en Internet en el directorio **Chapter06**[2]. Se supone que ya hemos cargado el código de los programas de este libro y que se ha asignado el valor adecuado a la variable `CLASSPATH`, como se ha descrito en la Sección A.1.2. Necesitaremos crear un directorio para el proyecto, y copiar la versión original, `TestStack.java`, en dicho directorio.

A.3.1 Symantec Café

1. Comenzamos por inicializar Café. Veremos brevemente un logotipo coloreado. A continuación, nos encontraremos con la peor característica de Café: la pantalla es prácticamente transparente, por lo que su entorno de sistema será aún visible. Si es la primera vez que utilizamos Café, nos encontraremos con algunos botones esparcidos por la pantalla. Se pueden mover a la parte superior para obtener algo similar a la Figura A.2. Nótese que en la figura solamente se muestra la parte superior de la pantalla.
2. Ahora crearemos un nuevo proyecto para `TestStack`, al que por simplicidad le llamaremos `TestStack`. Pulse el menú *Project* y seleccione *New*, como se muestra en la Figura A.3.
3. En este punto, inicializamos la herramienta *ProjectExpress*, que hace aparecer una secuencia de cuatro cuadros de diálogo. Es probable que sólo necesitemos los tres primeros.

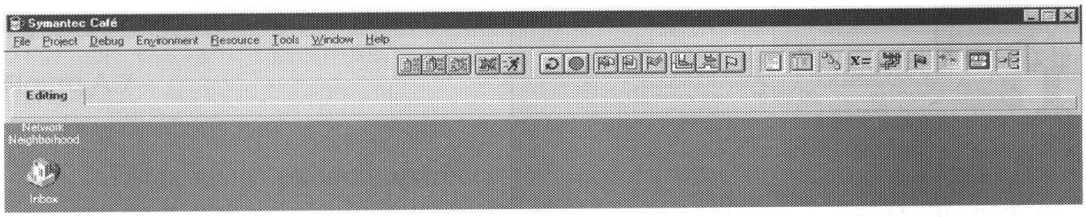

Figura A.2

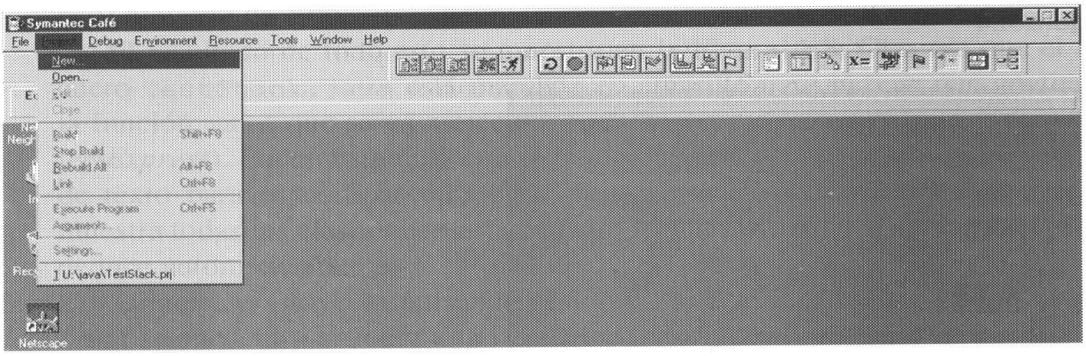

Figura A.3

[2] *N. del T.:* En el Capítulo 6 definimos una clase `TestPila` cuya versión original (en inglés) se llama `TestStack`. Las figuras en este apéndice reflejan la creación de un proyecto para la versión original de la clase, por lo que, de ahora en adelante todas las referencias se harán a la versión original, para mantener la coherencia con las figuras.

a) Seleccionamos en el primer cuadro un dispositivo de disco. Seleccionamos también el directorio en el que deseamos guardar el proyecto. Entonces escribimos el nombre del proyecto. Asegúrese de que termina con el sufijo `.prj`. Pulse *Next*. *Véase* la Figura A.4.

b) En el segundo cuadro, seleccionamos *Application* en *Target Type*. Tras ello escribimos `TestStack` como nombre de la clase principal. Pulse *Next*. *Véase* la Figura A.5.

c) En el tercer cuadro de diálogo, añadimos los ficheros al proyecto. El fichero de la clase principal debe estar ya en el mismo directorio que el fichero del proyecto. Si no es así, no se preocupe, podemos añadir ficheros al proyecto más adelante. Alternativamente, podemos copiar el fichero en este momento (desde fuera de Café) y teclear su nombre. Esto se muestra en la Figura A.6. Si `TestStack.java` estuviera ya presente en el directorio, habría aparecido en la lista de posibles ficheros a añadir. Si hay varios ficheros a añadir, seguimos pulsando *Add* o pulsando dos veces sobre el nombre del fichero en la lista de ficheros. Cuando hayamos añadido los ficheros, pulsamos *Finish*. No necesitamos ir al paso 4 ofrecido por *ProjectExpress*.

4. Una vez que se ha creado el proyecto, establecemos algunas vistas. En el menú *Window*, seleccionamos *Goto View*. Tras ello añadimos las vistas *Output* y *Project*. *Véase* la Figura A.7.

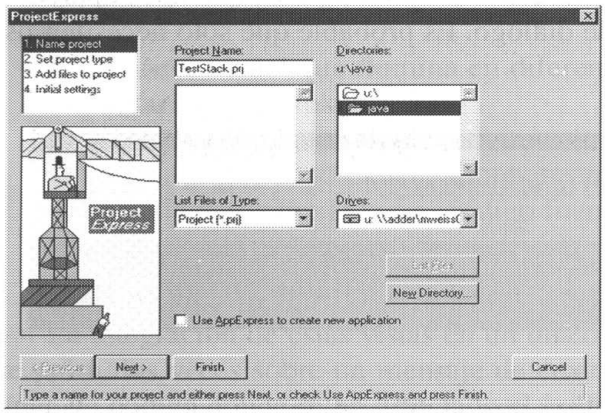

Figura A.4

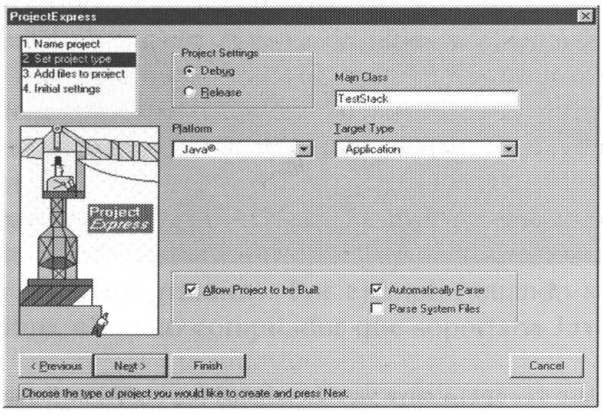

Figura A.5

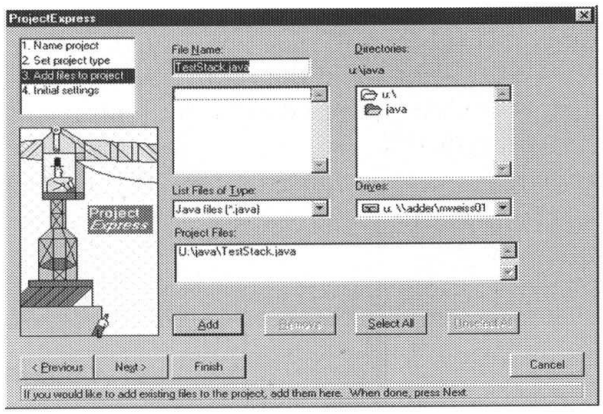

Figura A.6

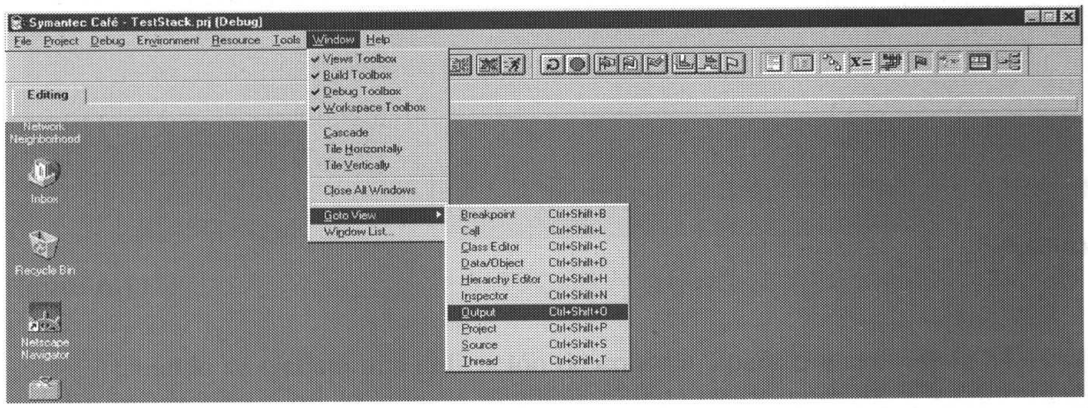

Figura A.7

Estas vistas pueden aparecer con un tamaño o en lugares que no nos gusten, pero podemos cambiar el tamaño de las ventanas, y moverlas a otros sitios. Desde la vista del proyecto, podemos pulsar dos veces sobre un fichero para obtener una ventana *Source*. Podemos cambiar el tamaño de dicha ventana y moverla de sitio.

El resultado de todo esto se muestra en la Figura A.8. Observe que el fichero `TestStack.java` está marcado en amarillo para indicar que contiene la función `main` que se va a ejecutar (esto se especificó en el paso 2 de *ProjectExpress*). Podemos pulsar con el botón derecho del ratón sobre otro objeto de texto para especificar otra clase como principal. La vista del proyecto muestra todas las clases necesarias que no forman parte del sistema, lo cual se deduce automáticamente.

5. Compilamos el proyecto pulsando el botón *Build*. (En la Figura A.9, el botón *Build* se encuentra arriba y a la izquierda de la etiqueta *Build*.)

   Solamente se compilan aquellos ficheros obsoletos. (Alternativamente podemos solicitar una compilación completa.) Cualquier error aparecerá en la ventana de salida. Si no hay errores, se presentará un mensaje al respecto. En este momento, podemos ejecutar el programa pulsando el botón *Run*. (Este botón muestra una persona corriendo, así que no podemos equivocarnos.)

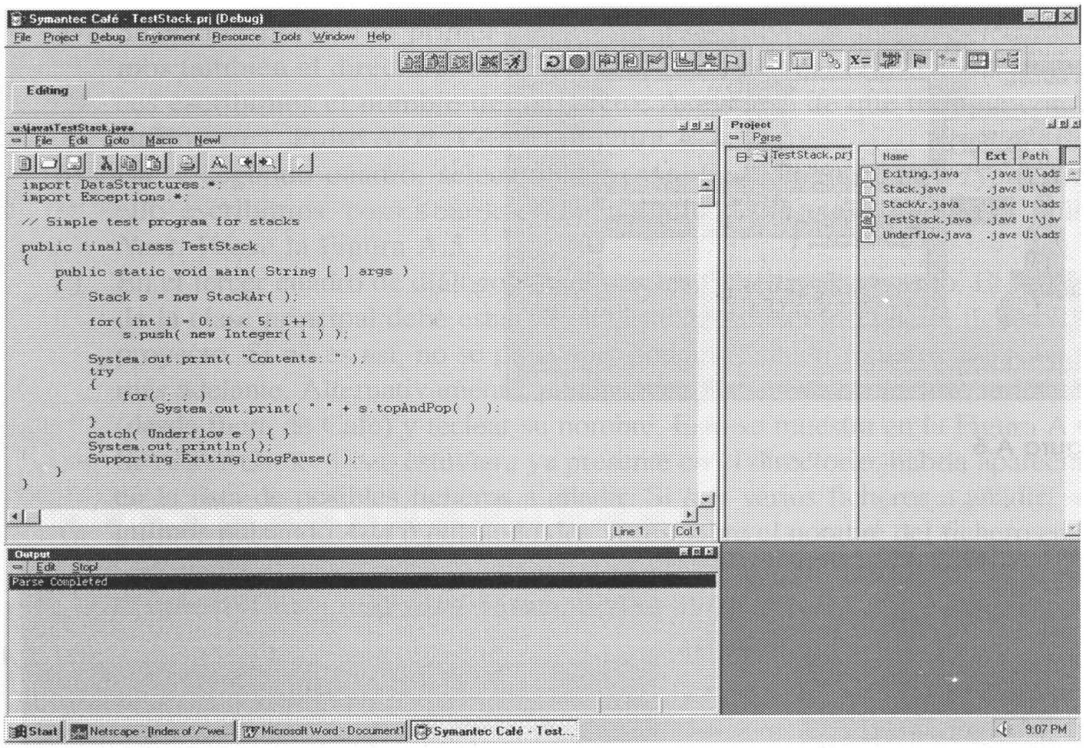

Figura A.8

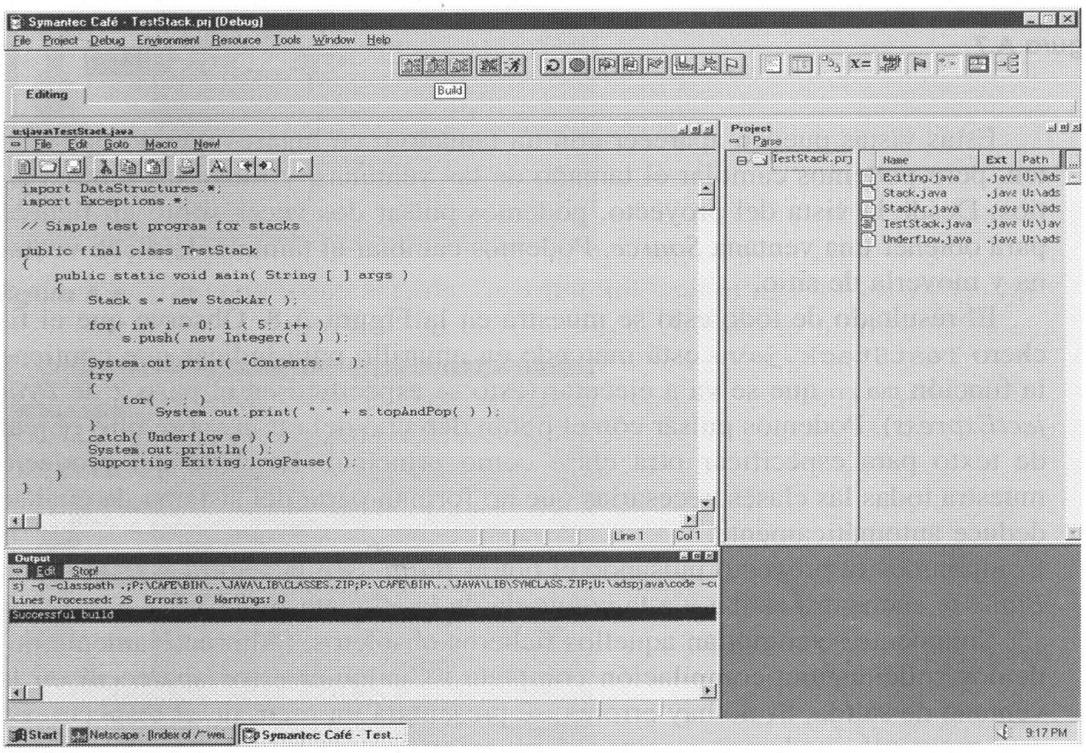

Figura A.9

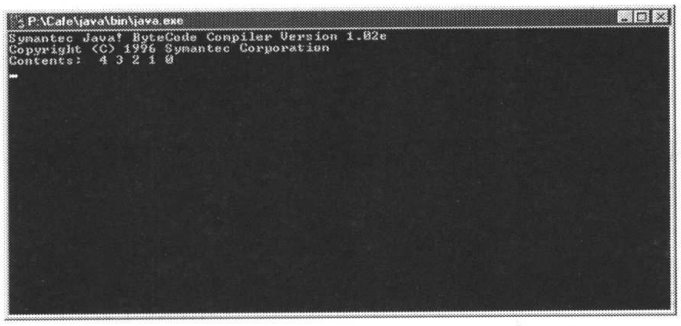

Figura A.10

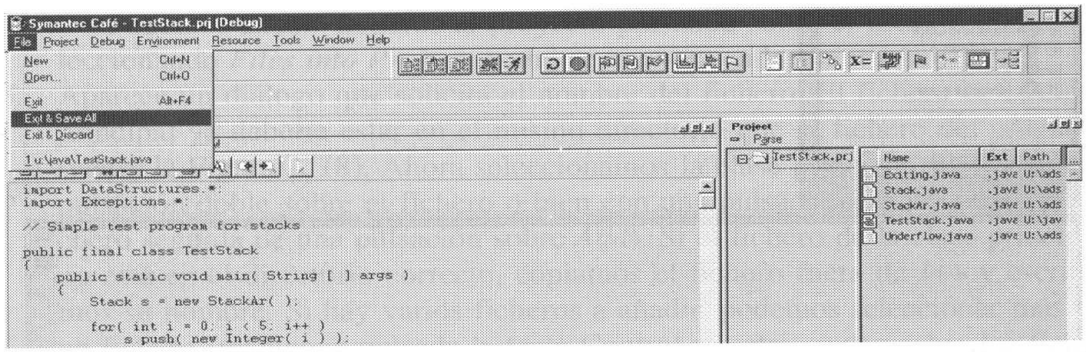

Figura A.11

Cuando se ejecuta una aplicación de consola, aparece una ventana MS-DOS. Es interactiva, con lo que se aceptan mandatos de entrada. Desafortunadamente, la ventana desaparece cuando el programa termina, lo que hace difícil ver la salida. Para evitar esto, podemos hacer una de las siguientes cosas:

I. Añadir una llamada a `Soporte.Saliendo.pausaLarga` o a `Soporte.Salir.apuntoDeSalir`, para evitar con ello que el programa termine.
II. Ejecutar el mandato *java* desde una ventana MS-DOS separada de Café.

Observemos que la opción I es inaceptable si el programa aborta por culpa de una excepción sin tratar. Sin embargo, en este caso, podemos ejecutar el depurador (mirar el menú *Debug*) para ver que es lo que ha ido mal. La opción I es también inaceptable si se está leyendo hasta el final de la entrada. La Figura A.10 muestra la ventana MS-DOS mencionada en la opción II.

6. Cuando hayamos terminado, salimos seleccionando el menú *Fichero*. Las opciones incluyen (*véase* la Figura A.11) salvar todo el trabajo (normalmente aconsejable) o salir sin salvar (aconsejable más a menudo de lo que nos gustaría).

A.3.2 Microsoft Visual J++

1. Comenzamos inicializando J++. Veremos brevemente un logotipo coloreado para Microsoft Studio, tras el cual aparecerá una pantalla. Si es la primera vez que usamos J++, nos encontraremos con varios objetos esparcidos, como una

barra de consejos y una ayuda (que contiene *Microsoft Visual J++ Books Online*). La barra de consejos se puede hacer desaparecer y la ayuda se puede colocar en la parte superior para obtener algo similar a la Figura A.12.

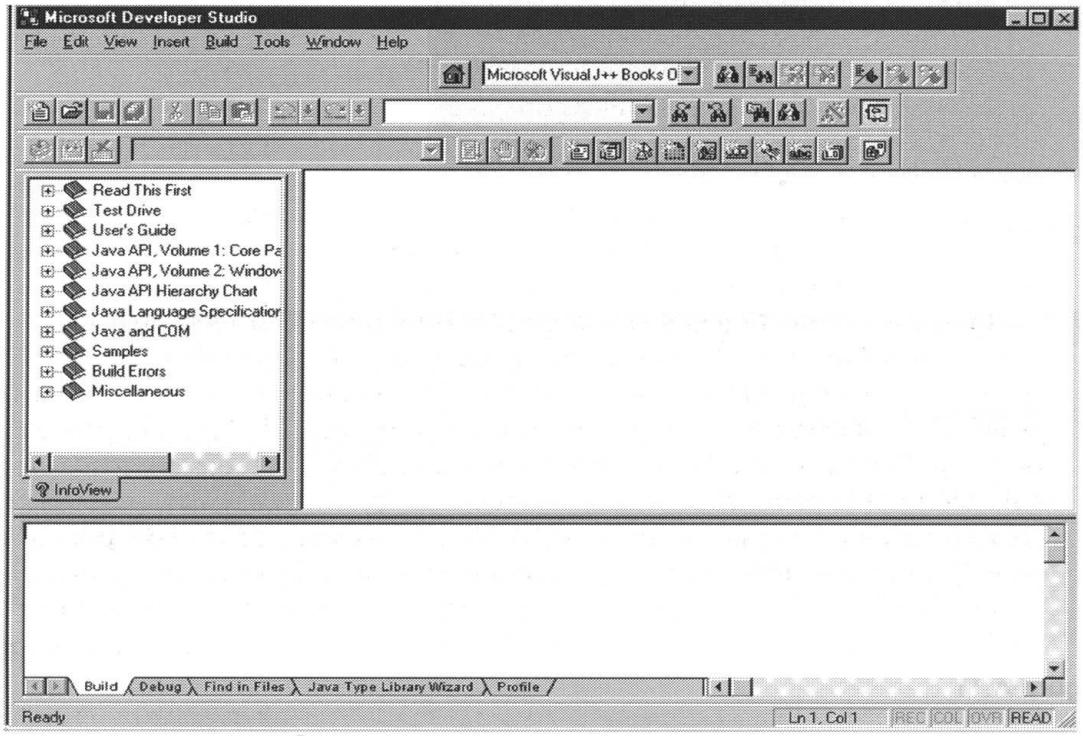

Figura A.12

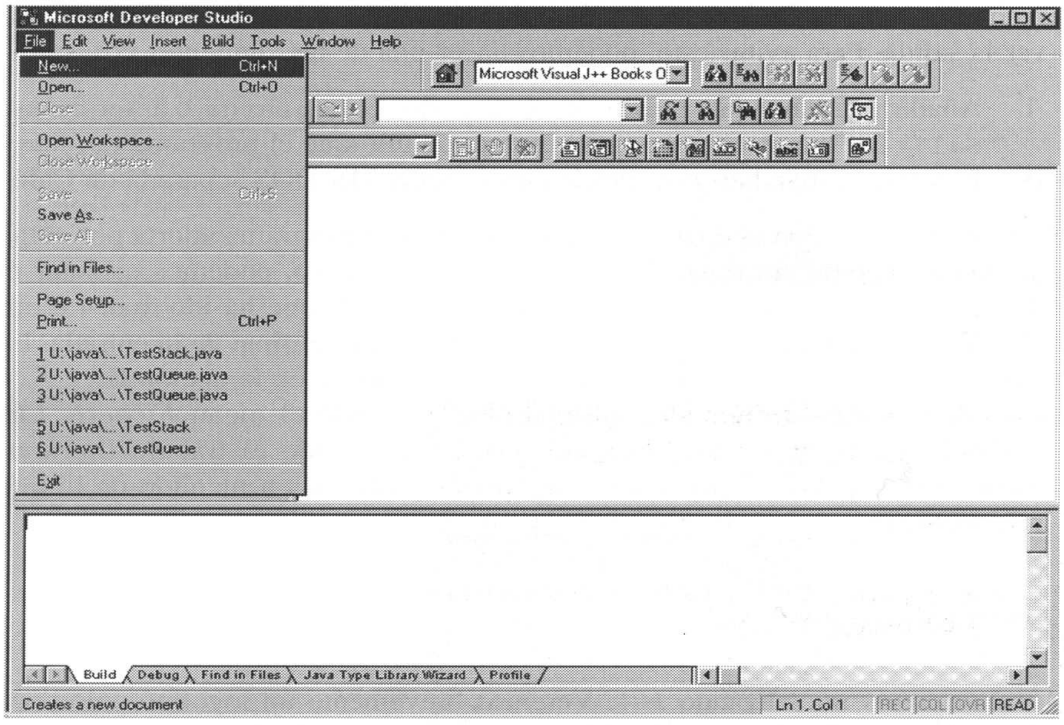

Figura A.13

2. Creamos un nuevo proyecto para `TestStack`. Pulsamos sobre el menú *File* y seleccionamos *New*, como se muestra en la Figura A.13. Esto genera un cuadro de diálogo que nos pide que especifiquemos lo que deseamos hacer. Pulsamos dos veces sobre *Project Workspace*, como se muestra en la Figura A.14.
3. A continuación aparece un cuadro de diálogo que nos solicita datos sobre el proyecto. Seleccionamos *Java Workspace* en *Type*; en *Location* indicamos el directorio donde se va a guardar el proyecto, y escribimos como nombre de proyecto `TestStack`. Todo ello se muestra en la Figura A.15. Después pulsamos *Create*.
4. En este momento, el cuadro de ayuda se sustituye por una vista de la clase, como se muestra en la Figura A.16. Si abrimos la carpeta de *TestStack classes*, veremos el fichero que constituye el proyecto. De momento el proyecto está vacío.

   Podemos añadir ficheros al proyecto pulsando sobre el menú *Insert* y seleccionando *Files into Project*, como se muestra en la Figura A.17.
5. Aparece un diálogo que solicita el nombre del fichero. El fichero de la clase principal ya debería estar en el mismo directorio que el fichero del proyecto (*véase* la Figura A.18). Ahora seleccionamos la clase principal (bien con una pulsación doble sobre el fichero o bien con una pulsación simple sobre el fichero seguida de una pulsación sobre *Add*). Si el fichero de la clase principal no está en el directorio correcto, copiamos el fichero fuera de J++ y escribimos su nombre. Si hay varios ficheros a añadir, podemos seleccionar más de un fichero (mantenga pulsada la tecla Control mientras pulse sobre los ficheros de la lista).
6. Pulse dos veces sobre *TestStack files* en la vista de la clase para ver que se ha añadido `TestStack.java` al proyecto. Pulse dos veces sobre *TestStack.java* para abrir el fichero en la ventana de código fuente.

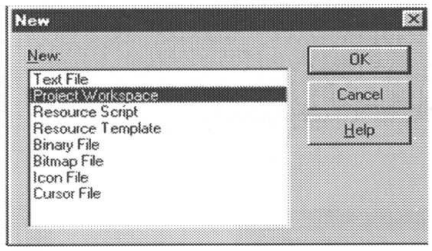

Figura A.14

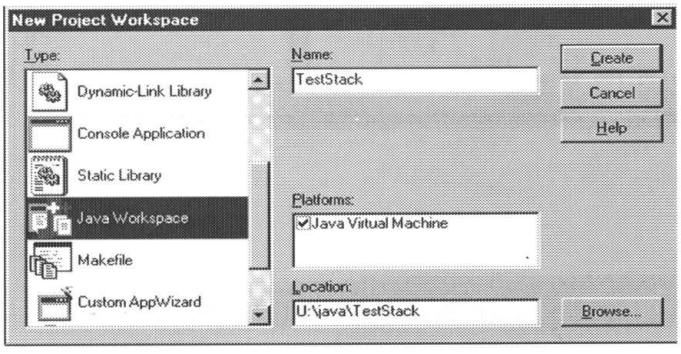

Figura A.15

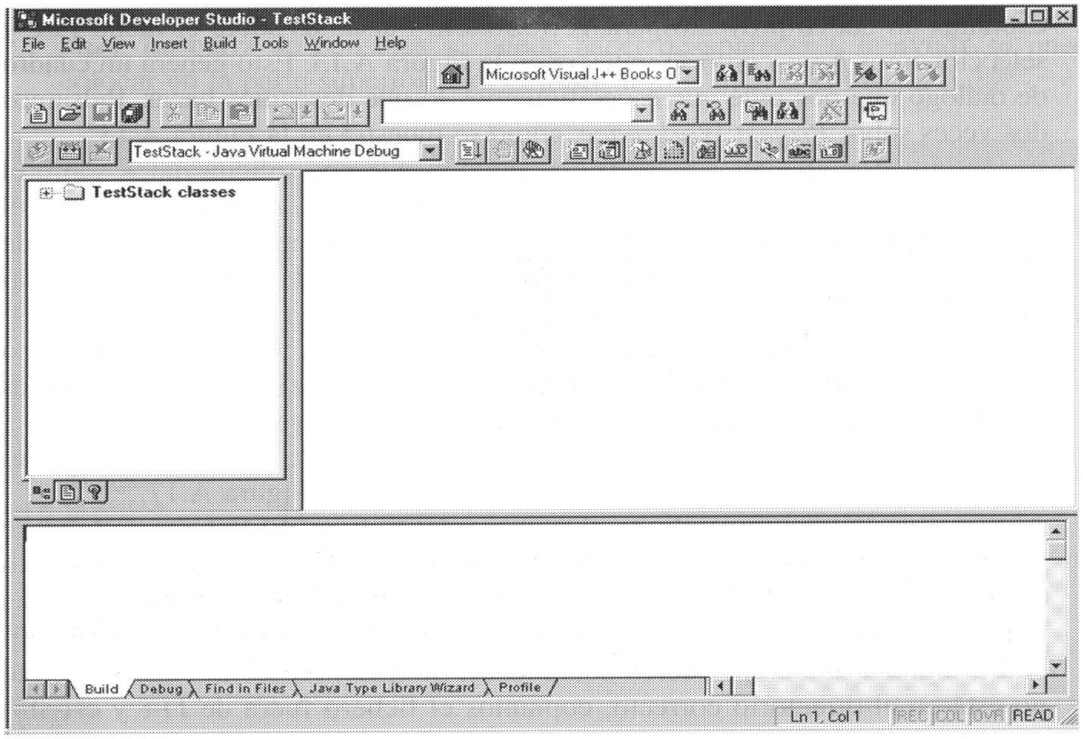

Figura A.16

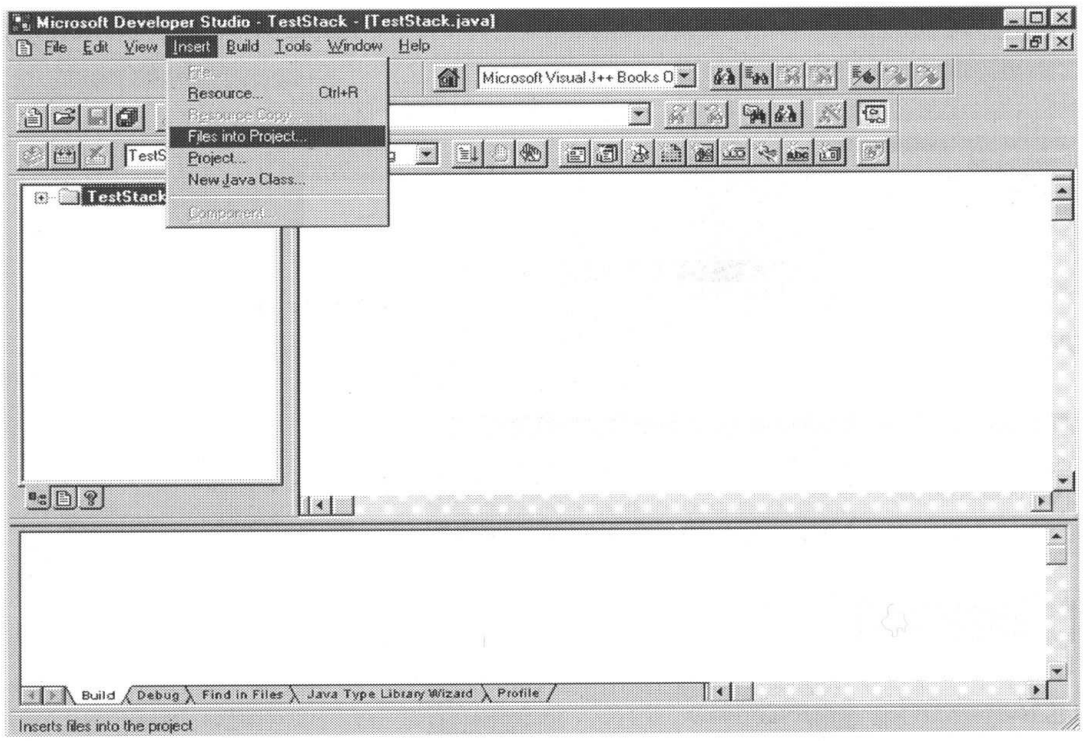

Figura A.17

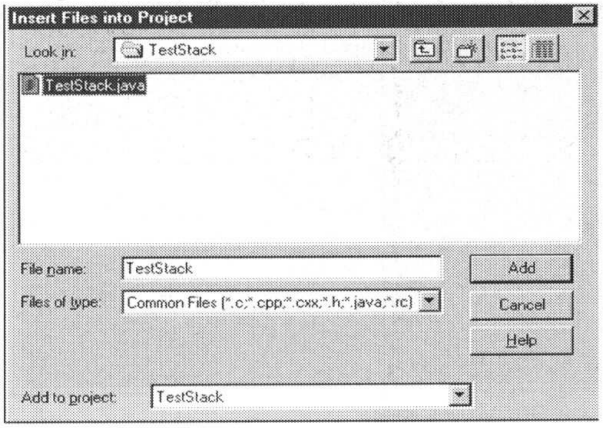

Figura A.18

7. A continuación, podemos cambiar algunas características del proyecto. En el menú *Build*, seleccionamos *Settings*, como se muestra en la Figura A.19.
8. Aparece entonces el cuadro de diálogo *Project Settings*. En él seleccionamos *Debug*, escribimos el nombre de la clase a depurar/ejecutar en *Class for debugging/executing*, y seleccionamos la opción *Stand-alone interpreter*, como se muestra en la Figura A.20.
9. Ahora está todo listo para compilar el proyecto y ejecutarlo. Los botones para hacer esto son los siguientes: el botón para compilar es el botón del medio situado directamente sobre la vista de la clase y el botón para ejecutar es el primero sobre la vista del código fuente. Estas posiciones dependen de la confi-

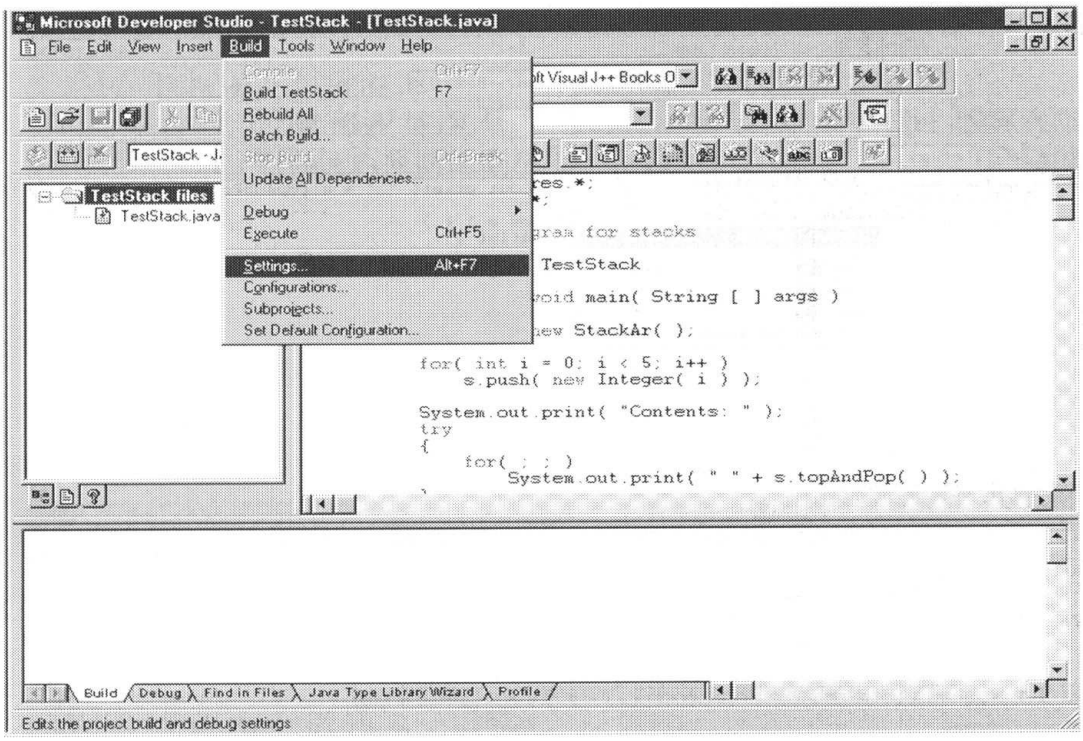

Figura A.19

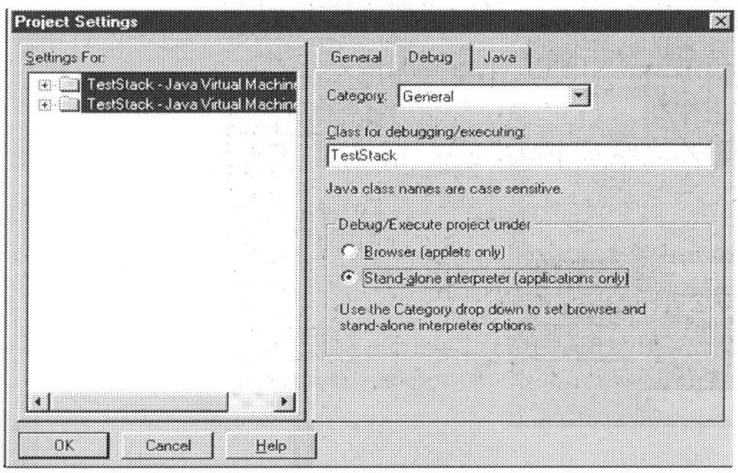

Figura A.20

guración local. Los botones podrían no ser visibles; en tal caso, se debe usar el menú *Build*. La Figura A.21 muestra el menú *Build*. Tiene una opción para compilar y otra para ejecutar. La ejecución compila automáticamente si el fichero de clase no existe o está obsoleto. La opción de compilación solamente compila los ficheros obsoletos. (Se puede solicitar también una compilación completa). Cualquier error aparecerá en la ventana de salida. Si no hay errores, aparecerá un mensaje al respecto.

Cuando ejecutamos una aplicación de consola, aparece una ventana MS-DOS, como en la Figura A.22. La ventana es interactiva, por lo que se

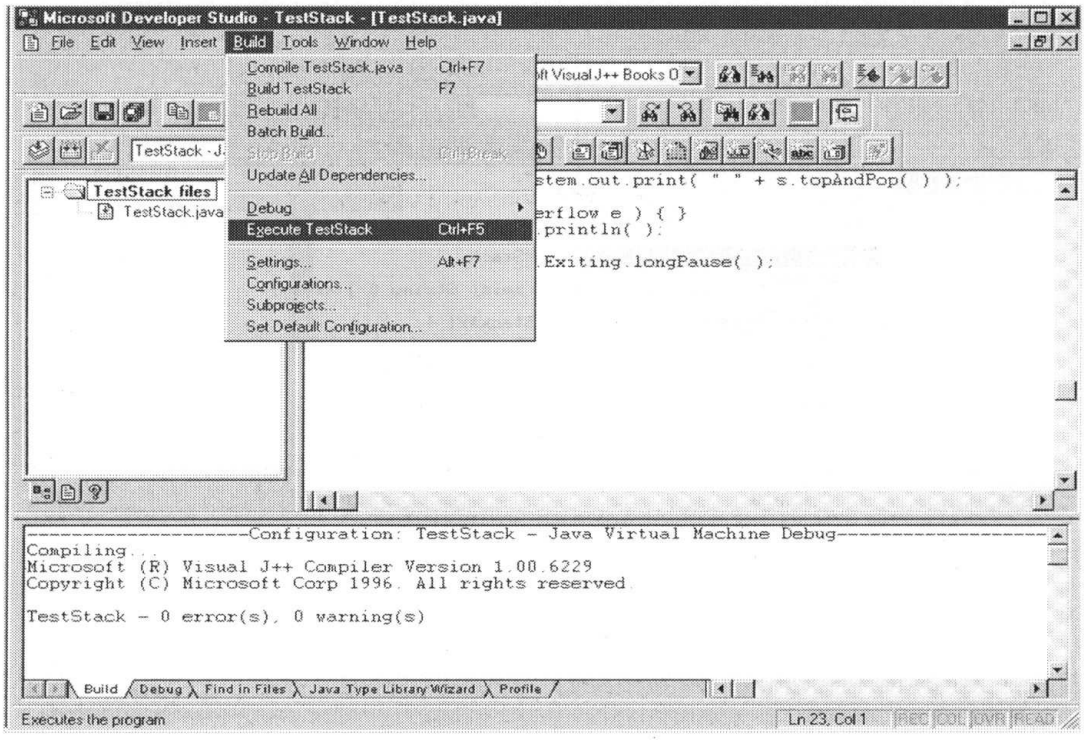

Figura A.21

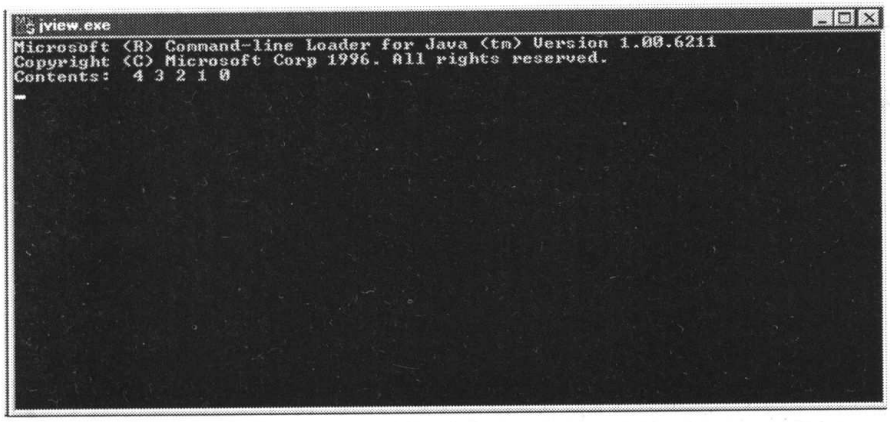

Figura A.22

pueden usar mandatos de entrada. En J++ 1.0, la ventana desaparece cuando el programa termina. Esto hace difícil ver la salida. Confiamos en que este problema desaparezca cuando esté disponible la versión de J++ acorde con Java 1.1. Hasta entonces, deberíamos utilizar las ideas del punto 5 de la Sección A.3.1 para evitar que el programa termine, impidiendo así que la ventana desaparezca.