El "tutorial" de BlueJ

Versión 2.0.1

para BlueJ versión 2.0.x

Michael Kölling
Mærsk Institute
University of Southern Denmark

Traducido al español por Germán Bordel
Universidad del País Vasco / Euskal Herriko Univertsitatea (UPV/EHU)
Lejona - Vizcaya
Spain
febrero 2005

Contenidos

1 Prefacio 3

1.1 Sobre BlueJ ... 3
1.2 Ámbito y audiencia ... 3
1.3 Copyright, licencia y redistribución. ... 3
1.4 "FeedBack" ... 4

2 Instalación 5

2.1 Instalación en Windows ... 5
2.2 Instalación en Macintosh ... 5
2.3 Instalación en Linux/Unix y otros sistemas. ... 6
2.4 Problemas de instalación. ... 6

3 Comenzando - editar /compilar /ejecutar. 7

3.1 Arrancando el BlueJ. ... 7
3.2 Abriendo un proyecto. ... 8
3.3 Creando objetos. ... 8
3.4 Ejecución. ... 10
3.5 Editando una clase. ... 12
3.6 Compilación. ... 13
3.7 Ayuda con los errores de compilación. ... 13

4 Haciendo un poco más ... 15

4.1 Inspección. ... 15
4.2 Pasando objetos como parámetros. ... 17

5 Creando un nuevo proyecto. 19

5.1 Creando el directorio del proyecto. ... 19
5.2 Creando clases ... 19
5.3 Creando dependencias ... 19
5.4 Eliminando elementos ... 20

6 Utilizando la zona de código. 21

6.1 Mostrando la zona de código ... 21
6.2 Evaluación de expresiones simples ... 22
6.3 Recibiendo objetos ... 22
6.4 Inspeccionando objetos ... 23
6.5 Ejecutando sentencias ... 23
6.6 Sentencias multilínea y secuencias de sentencias ... 23
6.7 Trabajando con variables ... 24
6.8 Historial de mandos ... 24

7 Depuración [Debugging]. 25

7.1 Estableciendo puntos de ruptura [Breakpoints]. ... 25
7.2 Avanzando paso a paso por el código. ... 27
7.3 Inspeccionando variables. ... 27
7.4 Detener y terminar. ... 28

8 Creando aplicaciones autónomas ("stand-alone"). 29

9 Creando applets. 31

9.1 Poniendo en marcha un applet. ... 31
9.2 Creando un applet. ... 32
9.3 Comprobando el applet. ... 32

10 Otras operaciones. 33

10.1 Abriendo paquetes no-BlueJ con BlueJ. ... 33
10.2 Añadiendo clases existentes al proyecto. ... 33
10.3 Llamando a "main" y a otros métodos estáticos. ... 33
10.4 Generando documentación. ... 34
10.5 Trabajando con bibliotecas. ... 34
10.6 Creando objetos a partir de clases de biblioteca. ... 34

11 Sólo los sumarios. 36

1 Prefacio

1.1 Sobre BlueJ

Este "tutorial" es una introducción al uso del entorno de programación BlueJ. BlueJ es un entorno de desarrollo Java™ diseñado específicamente para la enseñanza a un nivel introductorio. Ha sido diseñado e implementado por el equipo BlueJ en la Deakin University, Melbourne, Australia, y la University of Kent, en Canterbury, UK.

En http://www.bluej.org se encuentra disponible más información sobre BlueJ.

1.2 Ámbito y audiencia

Este "tutorial" esta orientado a aquella gente que quiera familiarizarse con las capacidades del entorno. No explica decisiones de diseño sobre las cuales se ha construido el entorno o las cuestiones de investigación que se encuentran tras él.

Este "tutorial" no está pensado para enseñar Java. Se aconseja a los principiantes en programación con Java el estudio de un libro de texto introductorio o seguir un curso de Java.

Este no es un manual de referencia completo del entorno. Se han dejado fuera muchos detalles - el énfasis se pone en dar una introducción breve y concisa en lugar de una completa cobertura de sus capacidades. Para obtener una referencia más detallada, véase el *"The BlueJ Environment Reference Manual"*, disponible en el sitio Web de BlueJ (www.bluj.org).

Cada sección comienza con un sumario en una frase de una línea. Esto permite a los usuarios ya familiarizados con partes del sistema decidir si leer o no cada apartado. La sección 11 simplemente repite estas líneas de sumario a modo de referencia rápida.

1.3 Copyright, licencia y redistribución.

1.4 “FeedBack”

Los comentarios, cuestiones, correcciones, críticas y cualquier otra clase de “feedback” concerniente al sistema BlueJ o este "tutorial" son bienvenidos y se anima a ello activamente. Por favor envíen el mail a Michael Kölling (mik@mip.sdu.dk).*

* En lo referente a esta traducción puede contactarse con german@we.lc.ehu.es

2 Instalación

BlueJ se distribuye en tres formatos diferentes: uno para sistemas Windows, otro para MacOs, y otro para todos los demás sistemas. La instalación es prácticamente directa.

Prerrequisitos

Debe tener J2SE v1.4 (a.k.a. JDK 1.4) o posterior instalado en el sistema para utilizar BlueJ. En general, es recomendable actualizar a la última versión estable (no beta) de Java. Si no tiene el JDK instalado puede descargarlo del sitio Web de Sun en http://java.sun.com/j2se/. En MacOS X se encuentra preinstalada una versión reciente de JDK - no necesita instalarla usted mismo. Si encuentra usted una página de descarga que ofrece "JRE" (Java Runtime Environment) y "SDK" (Software Development Kit), debe descargar "SDK" - el JRE no es suficiente.

2.1 Instalación en Windows

El fichero de distribución para el sistema Windows se llama *bluejsetup.xxx.exe*, donde xxx es un número de versión. Por ejemplo, la distribución de BlueJ 2.0.0 se llama bluejsetup-200.exe. Debe tener este fichero en el disco, o puede descargarlo del sitio Web de BlueJ en http://www.bluej.org. Ejecute este instalador.

El instalador le permite seleccionar el directorio donde realizar la instalación. También ofrece la opción de instalar un acceso en el menú de inicio y en el escritorio.

Tras la instalación, encontrará el programa *bluej.exe* en el directorio de instalación de BlueJ.

La primera vez que lance BlueJ, buscará el sistema Java (JDK). Si encuentra más de un sistema Java adecuado (p. ej. tiene usted instalados JDK 1.4.2 y JDK 1.5.0), un dialogo le permitirá seleccionar cual usar. Si no encuentra ninguno, se le pedirá que lo localice usted mismo (esto puede suceder cuando se ha instalado un sistema JDK, pero las correspondientes entradas en el registro se han borrado).

El instalador de BlueJ también instala un programa llamado *vmselect.exe*. Utilizando este programa usted puede cambiar posteriormente la versión de Java que usa BlueJ. Ejecute *vmselect* para arrancar BlueJ con una versión diferente de java.

La elección de JDK se almacena para cada versión de BlueJ. Si tiene diferentes versiones de BlueJ instaladas, usted puede usar una versión de BlueJ con JDK 1.4.2 y otra versión de BlueJ con JDK 1.5. El cambio de la versión de Java para BlueJ realizará este cambio para todas las instalaciones de BlueJ de la misma versión para el mismo usuario.

2.2 Instalación en Macintosh

Por favor, tenga en cuenta que BlueJ solo corre sobre MacOS X.

El fichero de distribución para MacOS se llama *BlueJ-xxx.zip*, donde xxx es un número de versión. Por ejemplo, la distribución de BlueJ versión 2.0.0 se llama *BlueJ-200.zip*. Debe tener este fichero en el disco, o puede descargarlo del sitio Web de BlueJ en http://www.bluej.org.

MacOS normalmente descomprimirá este archivo automáticamente después de la descarga. Si no es así, haga doble clic para descomprimirlo.

Después de descomprimir, tendrá usted una carpeta llamada *BlueJ-xxx*. Mueva esta carpeta a su carpeta Aplicaciones (o donde quiera que usted desee mantenerla). No es necesaria más instalación.

2.3 Instalación en Linux/Unix y otros sistemas.

El fichero general de distribución es un fichero ejecutable jar. Se llama *bluej-xxx.jar*, donde xxx es un número de versión. Por ejemplo, la distribución de BlueJ 2.0.0 se llama *bluej-200.jar*. Debe tener este fichero en el disco, o puede descargarlo del sitio Web de BlueJ en http://www.bluej.org.

Ponga en marcha el instalador ejecutando el siguiente mando. NOTA: para este ejemplo uso el fichero de distribución *bluej-200.jar* — es necesario usar el nombre de fichero que usted tenga (con el número de versión correcto).

```
<j2se-path>/bin/java -jar bluej-200.jar
```

`<j2se-path>` es el directorio donde se instaló el JDK.

Se muestra una ventana, permitiéndole elegir el directorio de instalación de BlueJ y la versión de JDK a usar para hacer funcionar el BlueJ.

Pulse *Install*. Una vez terminado, BlueJ debería estar instalado.

2.4 Problemas de instalación.

Si tiene cualquier problema, consulte las "*Frequently Asked Questions* (FAQ)" en el sitio Web de BlueJ (http://www.bluej.org/help/faq.html) y lea la sección "How To Ask For Help" (http://www.bluej.org/help/ask-help.html)

3 Comenzando - editar /compilar /ejecutar.

3.1 Arrancando el BlueJ.

En Windows y MacOS, se ha instalado un programa llamado *BlueJ*. Póngalo en marcha.

En los sistemas Unix, el instalador sitúa un "script" llamado *bluej* en el directorio de instalación. Desde un interfaz GUI, simplemente haga doble-clic en el fichero. Desde una línea de mando puede iniciar BlueJ con un proyecto como argumento o sin el:

```
$ bluej
```

o

```
$ bluej examples/people
```

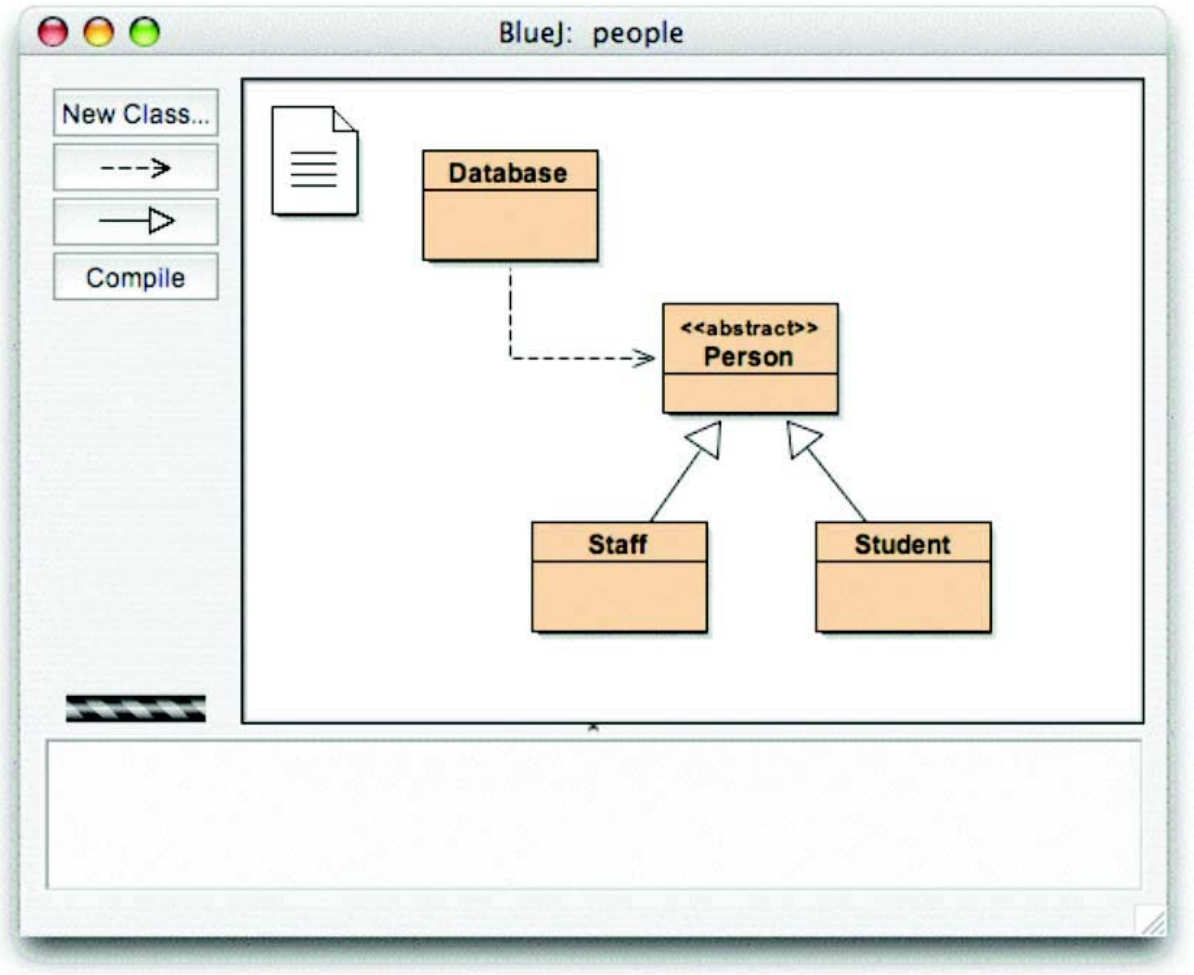

Figura 1: La ventana principal de BlueJ.

3.2 Abriendo un proyecto.

Sumario: Para abrir un proyecto, seleccione Open *del menú* Project

Los proyectos BlueJ, como los paquetes estándar de Java, son directorios conteniendo los ficheros incluidos en el proyecto.

Después de instalar BlueJ, use el mando de menú Project-Open... para seleccionar y abrir un proyecto.

En el directorio *examples* se incluyen varios proyectos de ejemplo con la distribución estándar de BlueJ.

Para esta sección del "tutorial", abra el proyecto *people*, que está incluido en este directorio. Puede encontrar el directorio *examples* en el directorio base (home) de BlueJ. Después de abrir el proyecto, debería usted ver algo similar a la ventana mostrada en la Figura 1. La ventana podría no mostrarse exactamente igual en su sistema, pero las diferencias deberían ser menores.

3.3 Creando objetos.

Sumario: Para crear un objeto, seleccione un constructor del menú emergente de la clase.

Una de las características fundamentales de BlueJ es que no sólo se puede ejecutar una aplicación completa, sino que se puede también interactuar con objetos aislados de cualquier clase y ejecutar sus métodos públicos. Una ejecución en BlueJ se realiza habitualmente creando un objeto e invocando entonces a uno de sus métodos. Esto es de gran ayuda durante el desarrollo de una aplicación — pueden comprobarse las clases individualmente tan pronto como han sido escritas. No es necesario escribir la aplicación completa primero.

> Nota "marginal": los métodos estáticos pueden ser ejecutados directamente sin crear primero un objeto. Uno de los métodos estáticos puede ser "main", por lo que podemos hacer lo mismo que sucede normalmente en las aplicaciones Java — comenzar una aplicación simplemente ejecutando un método estático principal ("main"). Volveremos a esto más tarde. Primero haremos algunas otras cosas, más interesantes, que no pueden hacerse normalmente en entornos Java.

Los cuadrados que ve en la parte central de la ventana principal (etiquetados *Database*, *Persona*, *Staff* y *Student*) son iconos representando las clases involucradas en esta aplicación. Puede obtener un menú con operaciones aplicables a una clase pulsando el botón derecho del ratón sobre el icono de la clase (Macintosh: ctrl.+ pulsar ratón[1]) (Figura 2). Las operaciones mostradas son operaciones *new* con cada uno de los constructores definidos para esta clase (primero) seguidas por algunas operaciones proporcionadas por el entorno.

[1] Cuando quiera que se mencione "pulsar botón derecho" en este "tutorial", los usuarios de Macintosh deben entenderlo como "ctrl.+ pulsar ratón"

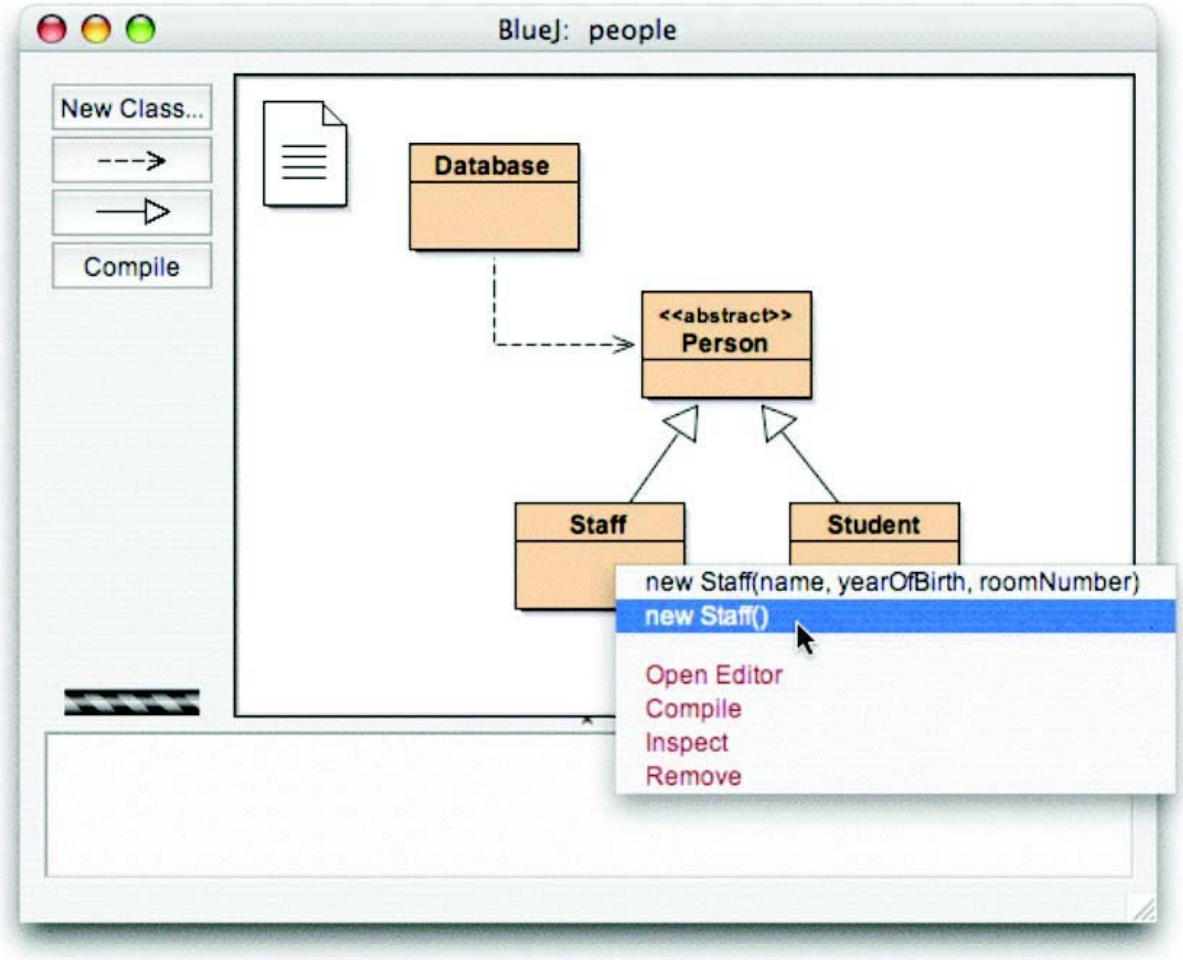

Figura 2. Operaciones de clase (menú emergente (popup))

Queremos crear un objeto *Staff*, por lo tanto debemos hacer "click-derecho" en el icono *Staff* (que hace emerger el menú mostrado en la Figura 2). El menú muestra dos constructores para crear un objeto *Staff*, uno con parámetros y otro sin ellos. Primero, seleccione el constructor sin parámetros. Aparece el dialogo mostrado en la Figura 3.

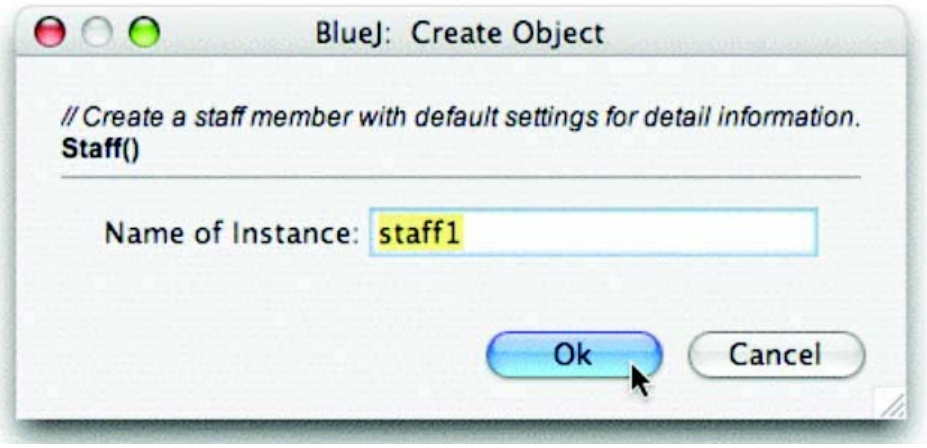

Figura 3:Creación de un objeto sin parámetros.

Este dialogo le pide un nombre para el objeto a crear. Al mismo tiempo, es sugerido un nombre por defecto (*staff_1*). Este nombre por defecto sirve por el momento, así que simplemente pulse OK. Se creará un objeto *Staff*.

Una vez que se ha creado el objeto, es situado en el banco de objetos (Figura 4). Esto es todo en cuanto a creación de objetos: seleccionar un constructor del menú de la clase, ejecutarlo y ya tenemos el objeto en el banco de objetos.

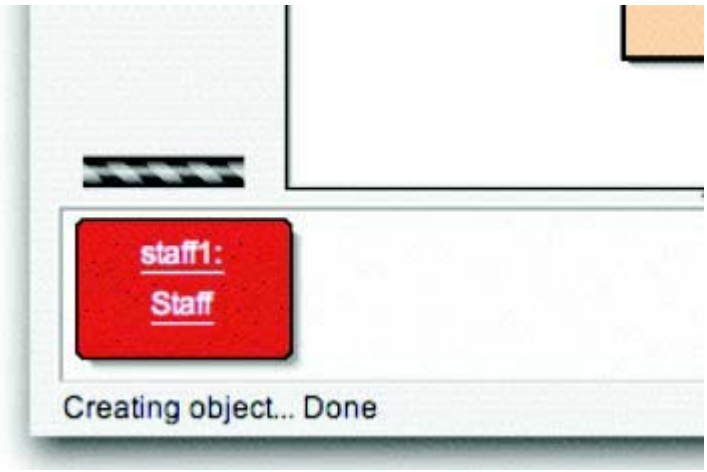

Figura 4: Un objeto en el banco de objetos.

Puede que haya observado que la clase *Person* esta etiquetada como <<abstract>> (es una clase abstracta). Notará (si lo prueba) que no puede crear objetos de clases abstractas (como lo define la especificación del lenguaje Java).

3.4 Ejecución.

Sumario: Para ejecutar un método, selecciónelo en el menú emergente del objeto.

Ahora que ha creado un objeto, puede ejecutar sus operaciones públicas. (Java llama a las operaciones *métodos*). Pulse el botón derecho del ratón sobre el objeto y aparecerá un menú con operaciones del objeto (Figura 5). El menú muestra los métodos disponibles para ese objeto y dos operaciones especiales proporcionadas por el entorno (*Inspect* (inspeccionar) y *Remove* (eliminar)). Analizaremos estas más tarde. Primero, concentrémonos en los métodos.

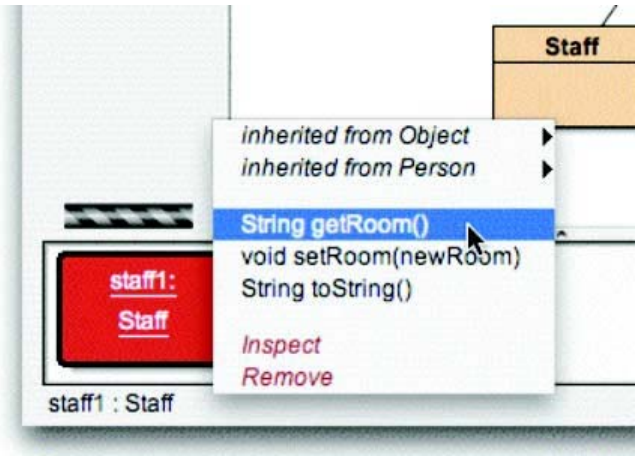

Figura 5: El menú de objeto.

Vemos que hay métodos *setRoom* y *getRoom* que establecen y devuelven el número de local de este empleado [miembro del staff]. Pruebe a llamar a *getRoom*. Sencillamente selecciónelo en el menú del objeto y será ejecutado. Aparece un dialogo mostrando el resultado de la llamada (Figura 6). En este caso el nombre reza "(unknown room)" (local desconocido) porque no especificamos previamente un local para esta persona.

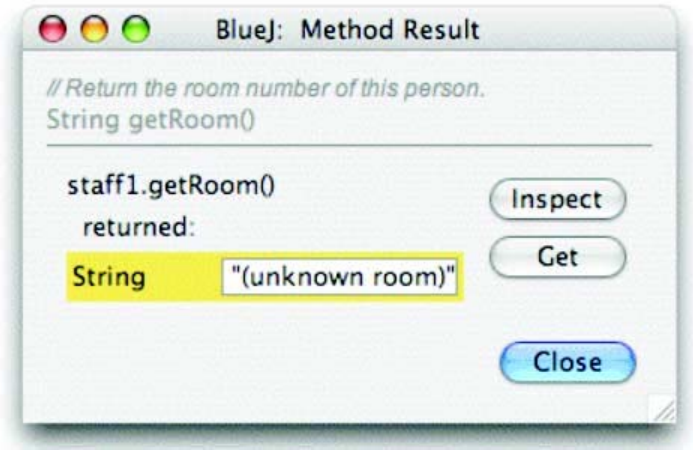

Figura 6: Presentación del resultado de una función.

Los métodos heredados de una superclase son accesibles a través de un submenú. En la parte alta del menú emergente del objeto hay dos submenús, uno para los métodos heredados de Object y uno para los de *Person* (Figura 5). Puede llamar a los métodos de *Person* (como por ejemplo *getName*) seleccionándolos en el submenú. Pruébelo. Notará que la respuesta es igualmente vaga: responde "(unknown name)" (nombre desconocido), porque no hemos dado un nombre a esta persona.

Ahora probemos a especificar un número de local. Esto mostrará cómo hacer una llamada que tiene parámetros. (Las llamadas a *getRoom* y *getName* tenían valores de retorno, pero no parámetros). Llame a la función *setRoom* seleccionándola en el menú. Aparece un diálogo requiriéndole que introduzca un parámetro (Figura 7).

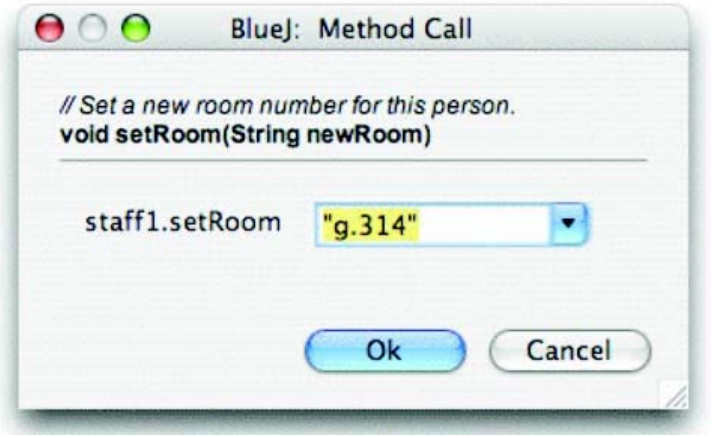

Figura 7: Dialogo de llamada a una función con parámetros.

En lo más alto, este diálogo muestra el interfaz del método a llamar (incluyendo comentario y prototipo [signature]). Debajo de esto se encuentra un campo de entrada de texto donde puede introducir el parámetro. La firma en la parte alta nos indica que se espera un parámetro de tipo String. Introduzca el nuevo local como una String (incluyendo las comillas) en el campo de texto y pulse OK.

Esto es todo — como este método no devuelve un valor, no hay dialogo de resultado. Llame a *getRoom* de nuevo para comprobar que el local ha cambiado.

Juegue un momento con la creación de objetos y las llamadas a métodos. Pruebe a llamar a un constructor con argumentos y llame a algunos otros métodos hasta que le resulten familiares estas operaciones.

3.5 Editando una clase.

Sumario: Para editar el código fuente de una clase, haga doble-clic en su icono.

Hasta el momento hemos tratado sólo con el interfaz de un objeto. Ahora es el momento de mirar dentro. Puede ver la implementación de una clase seleccionando *Open Editor* de entre las operaciones de clase. (Recordatorio: haciendo clic-derecho en el icono de clase se muestran las operaciones de clase.) Hacer doble-clic en el icono de clase es un atajo (shortcut) para la misma funcionalidad. En este "tutorial" no se describe el editor con mucho detalle, pero le debe de resultar muy sencillo de usar. Algunos detalles sobre el editor se describirán más tarde separadamente. Por ahora, obra la implementación de la clase *Staff*. Localice la implementación del método *getRoom*. Éste devuelve, como sugiere su nombre, el local del miembro del personal. Cambiemos el método añadiendo el prefijo "room" al resultado de la función (de modo que el método devolverá, digamos, "room M.3.18" en lugar de sólo "M.3.18"). Podemos hacer esto cambiando la línea

```
return room
```

por

```
return "room " + room;
```

BlueJ soporta por completo todo el Java no modificado, de modo que no hay nada especial a tener en cuenta respecto a cómo implementar las clases.

3.6 Compilación.

Sumario: Para compilar una clase, pulse el botón Compile *del editor. Para compilar un proyecto, pulse el botón* Compile *en la ventana del proyecto.*

Después de insertar el texto (antes de que haga ninguna otra cosa), compruebe la vista general del proyecto (la ventana principal). Notará que el icono de la clase *Staff* ha cambiado: ahora esta rayado. El aspecto rayado marca las clases que no han sido compiladas desde el último cambio. Volvamos al editor.

> Nota "marginal": Se preguntará por qué los iconos de clase no estaban rayados cuando abrió el proyecto. Esto se debe a que las clases en el proyecto people se distribuyen ya compiladas. A menudo los proyectos en BlueJ se distribuyen sin compilar, de modo que puede esperar ver la mayoría de los iconos rayados cuando abra por primera vez un proyecto en adelante.

En la barra de herramientas que se encuentra en la parte alta del editor hay varios botones con las funciones usadas frecuentemente. Una de ellas es *Compile*. Esta función le permite compilar la clase directamente desde dentro del editor. Si no ha cometido ningún error, debe aparecer un mensaje en el área de información que se encuentra abajo en el editor notificándole que la clase ha sido compilada. Si ha cometido alguna equivocación que conduce a un error de sintaxis, la línea con el error es resaltada y se muestra un mensaje de error en el área de información. (En el caso de que la compilación haya funcionado a la primera, pruebe a introducir un error sintáctico ahora —como por ejemplo eliminar un punto y coma— y compile de nuevo únicamente para comprobar cómo se muestra).

Una vez que haya compilado con éxito la clase, cierre el editor.

> Nota "marginal": no es necesario salvar de modo explícito el código fuente de la clase. Los fuentes se salvan automáticamente cuando es necesario (por ejemplo cuando se cierra el editor o antes de compilar la clase). Puede salvar explícitamente si lo desea (hay una función en el menú *Class* del editor) pero esto sólo es necesario si su sistema es realmente inestable y se viene abajo frecuentemente y le preocupa perder su trabajo.

La barra de herramientas de la ventana del proyecto también tiene un botón *Compile*. Esta operación de compilación afecta a todo el proyecto. (De hecho determina qué clases necesitan recompilación y entonces lo hace en el orden adecuado.) Compruebe esto cambiando dos o más clases (de modo que dos o más clases aparecerán rayadas en el diagrama de clases) y entonces pulse el botón *Compile*. Si se detecta un error en una de las clases compiladas, se abrirá el editor y serán mostrados el mensaje de error y su posición.

Podrá comprobar que el banco de objetos estará de nuevo vacío. Los objetos son eliminados cada vez que se cambia la implementación.

3.7 Ayuda con los errores de compilación.

Sumario: Para obtener ayuda para un mensaje de error de compilación, pulse la interrogación que está próxima al mensaje de error.

Muy frecuentemente, los estudiantes principiantes tienen dificultades para entender los mensajes de error del compilador. Nosotros intentamos proporcionar alguna ayuda.

Abra el editor de nuevo, introduzca un error en el fichero fuente, y compile. Debe aparecer un mensaje en el área de información del editor. A la derecha del área de información aparece una

interrogación que se puede pulsar para obtener algo más de información sobre ese tipo de error (Figura 8).

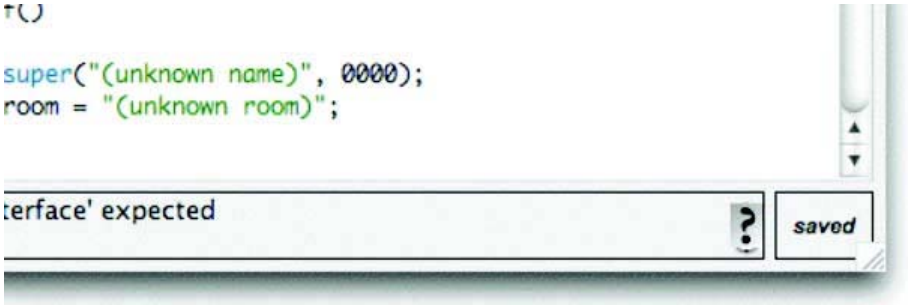

Figura 8: Un error de compilación y el botón de ayuda.

En este momento, no hay textos de ayuda para todos los mensajes de error. Todavía falta escribir algunos textos de ayuda. De todos modos merece la pena probarlo — muchos errores ya están explicados. Los que faltan serán escritos e incluidos en una distribución futura de BlueJ.

4 Haciendo un poco más ...

En esta sección, veremos algunas cosas más que pueden realizarse con el entorno. Cosas que no son esenciales, pero sí comúnmente muy usadas.

4.1 Inspección.

Sumario: La inspección de objetos permite realizar una depuración sencilla al mostrar el estado interno de un objeto.

Cuando ejecutó un método de un objeto, pudo haber notado que, además de los métodos definidos por el usuario, se encuentra disponible para los objetos la operación *Inspection* [inspección] (Figura 5). Esta operación permite comprobar el estado de las variables de instancia ("campos") de los objetos. Pruebe a crear un objeto con algunos valores definidos (por ejemplo, un objeto *Staff* con el constructor que recibe parámetros). A continuación seleccione *Inspection* en el menú del objeto. Aparece un cuadro de diálogo mostrando los campos del objeto, sus tipos y sus valores (Figura 9).

Figura 9: Dialogo Inspección.

La inspección es útil para comprobar rápidamente si una operación "mutadora" (una operación que cambia el estado del objeto) se ha ejecutado correctamente. Por tanto, la inspección es una herramienta sencilla de depuración.

En el ejemplo *Staff*, todos los campos son de tipo simple (no hay objetos ni strings). El valor de estos tipos puede mostrarse directamente. Usted puede ver inmediatamente que el constructor ha hecho las asignaciones correctas.

En casos más complejos, los valores de los campos pueden ser referencias a objetos definidos por el usuario. Para ver un ejemplo de ello usaremos otro proyecto. Abra el proyecto *people2*, que también se encuentra en la distribución estándar de BlueJ. En la Figura 10 se muestra la ventana principal de people2. Como puede ver, este segundo ejemplo tiene una clase *Address* además de las clases vistas anteriormente. Uno de los campos en la clase *Person* es del tipo definido por el usuario *Address*.

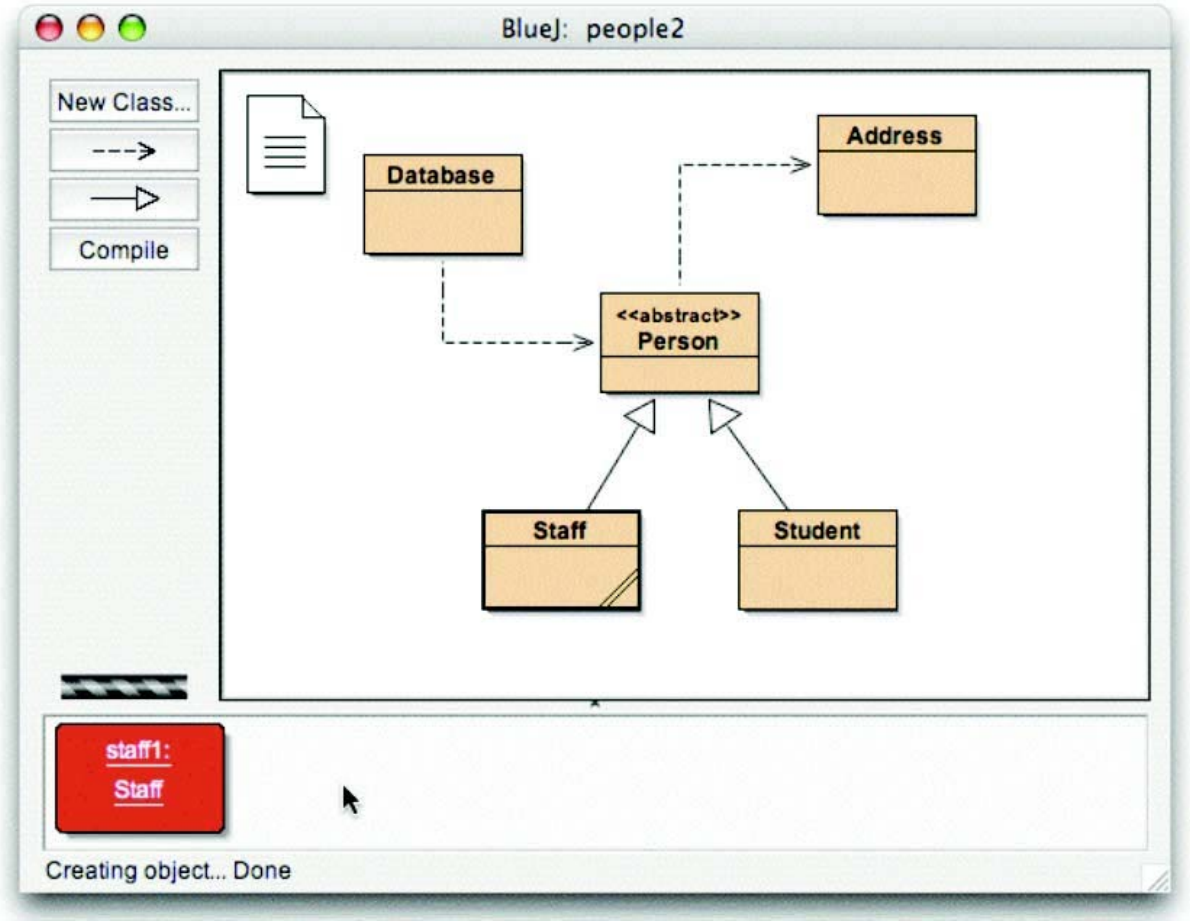

Figura 10: La ventana del proyecto *people2*.

Para lo próximo que queremos probar —inspección de campos objeto— cree un objeto Staff y llame al método *setAddress* de este objeto (lo encontrará en el submenú de *Person*). Introduzca una dirección. Internamente, el código de *Staff* crea un objeto de clase *Address* y lo almacena en el campo *address*.

Ahora, inspeccione el objeto *Staff*. El cuadro de diálogo de inspección resultante se muestra en la Figura 11. Los campos en el objeto *Staff* incluyen ahora *address*. Como puede ver, su valor se muestra como una flecha, que representa una referencia a otro objeto. Dado que es un objeto complejo, definido por el usuario, su valor no puede ser mostrado directamente en esa lista. Para examinar la dirección [address] más a fondo, seleccione el campo *address* en la lista y pulse el botón *Inspect* del cuadro de diálogo. (También puede hacer doble clic en el campo *address*). Se abre otra ventana de inspección, mostrando los detalles del objeto *Address* (Figura 12).

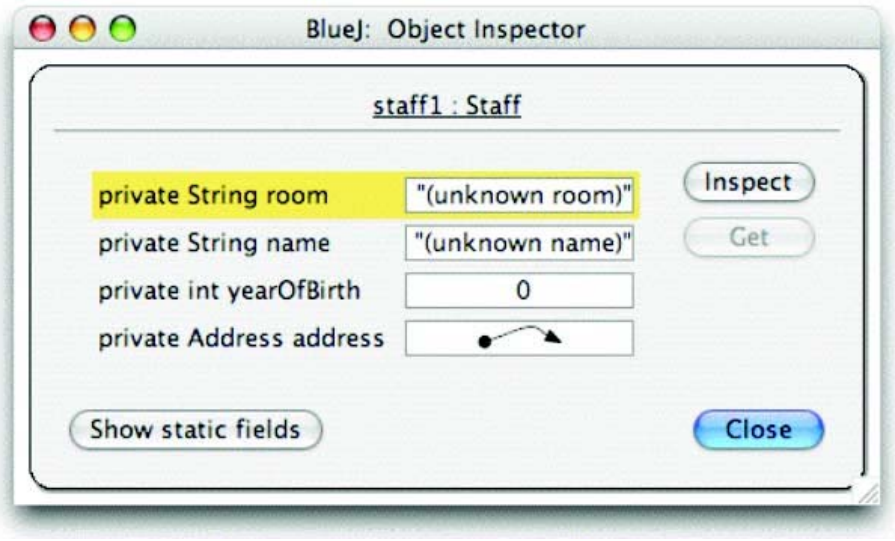

Figura 11: Inspección con referencia a objeto.

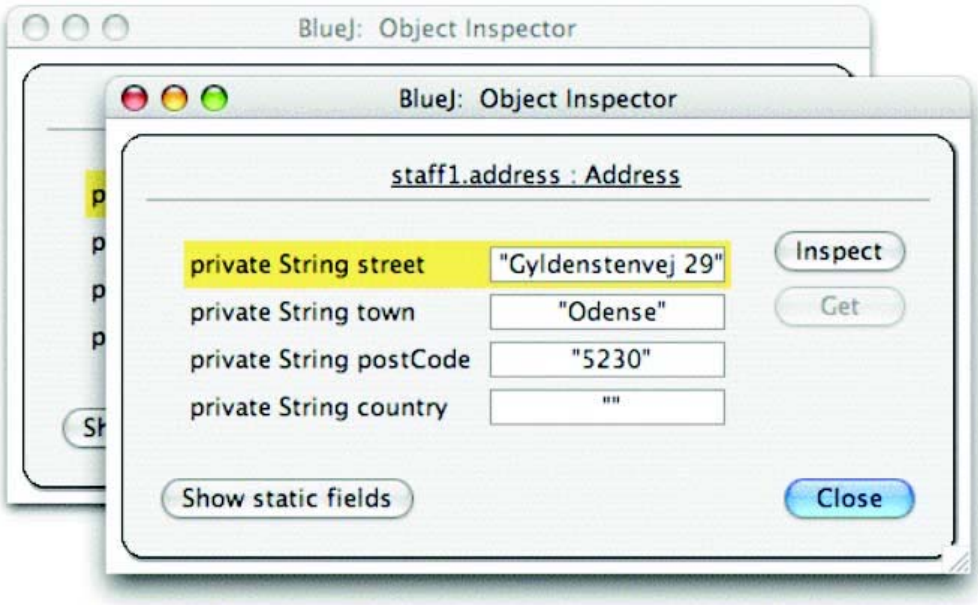

Figura 12: Inspección de objeto interno.

Si el campo seleccionado es público entonces, en lugar de pulsar *Inspect*, puede también seleccionar el campo *address* y pulsar el botón *Get*. Esta operación sitúa el objeto seleccionado en el banco de objetos. Ahí, usted puede examinarlo más a fondo haciendo llamadas a sus métodos.

4.2 Pasando objetos como parámetros.

Sumario: *Se puede pasar un objeto como parámetro a un método pulsando en el icono del objeto.*

Los objetos pueden ser pasados como parámetros a métodos de otros objetos. Probemos un ejemplo. Cree un objeto de clase *Database*. (Observará que la clase *Database* sólo tiene un constructor sin parámetros, de modo que la construcción de un objeto es directa). El objeto *Database* tiene la habilidad de almacenar una lista de personas. Tiene métodos para añadir objetos persona y mostrar

todas las personas almacenadas en un momento dado. (¡Llamarle *Database* es ciertamente un poco exagerado!).

Si no tiene aún un objeto *Staff* o *Student* en el banco de objetos, cree también uno de ellos. Para lo que sigue necesita un objeto *Database* y un objeto *Staff* o *Student* en el banco de objetos a la vez.

Ahora llame al método *addPerson* del objeto *Database*. El prototipo le indica que se necesita un parámetro de tipo *Person*. (Recuerde: la clase *Person* es abstracta, por lo que no hay objetos que sean directamente de tipo *Person*. Pero, debido a la "subclasificación", objetos *Student* y *Staff* pueden ser sustituidos por objetos *Person*. Por tanto es legal pasar un objeto *Student* o *Staff* sonde se espera un *Person*). Para pasar el objeto que tiene en su banco de objetos como parámetro a la llamada que esta llevando a cabo, puede introducir su nombre en el campo del parámetro o, a modo de atajo, simplemente hacer doble clic en el objeto. Esto introduce su nombre en el campo del cuadro de diálogo de llamada al método. Pulse OK y la llamada se realiza. Como no hay valor de retorno para este método, no vemos inmediatamente un resultado. Puede llamar a *ListAll* en el objeto *Database* para comprobar que la operación efectivamente se ha realizado. La operación *ListAll* escribe la información de la persona en la salida estándar. Observará que se abre automáticamente un terminal de texto para mostrarlo.

Pruebe esto de nuevo con más de una persona introducida en la "base de datos".

5 Creando un nuevo proyecto.

Este capítulo le lleva a echar un vistazo rápido al modo de establecer de un nuevo proyecto.

5.1 Creando el directorio del proyecto.

Sumario: Para crear un proyecto seleccione New... *del menú* Project.

Para crear un nuevo proyecto seleccione *Project - New...* en el menú. Se abre un cuadro de dialogo de selección de ficheros que le permite especificar un nombre y una localización para el nuevo proyecto. Pruébelo ahora. Puede elegir cualquier nombre para su proyecto. Después de que pulse OK, se creará un directorio con el nombre que especificó usted, y la ventana principal muestra el nuevo proyecto vacío.

5.2 Creando clases

Sumario: Para crear una clase, pulse el botón New Class *y especifique el nombre de la clase.*

Puede crear ahora sus clases pulsando en el botón *New Class* en la barra de herramientas del proyecto. Se le pedirá un nombre para la clase - este nombre debe ser un identificador válido en Java.

También puede usted elegir entre cuatro tipos de clases: abstracta, interfaz, applet o "estándar". Esta elección determina el esqueleto del código que se crea inicialmente para su clase. Puede cambiar el tipo de clase posteriormente editando el código fuente (por ejemplo, añadiendo la palabra clave "abstract" en el código).

Después de crear la clase, queda representada por un icono en el diagrama. Si no es una clase estándar, el tipo (interfaz, abstracta o applet) se indica en el icono de clase. Cuando abre el editor para una nueva clase observará que se ha creado el esqueleto por defecto - esto debería facilitar la iniciación. El código por defecto es sintácticamente correcto. Puede ser compilado (pero no hace gran cosa). Pruebe a crear unas pocas clases y compilarlas.

5.3 Creando dependencias

Sumario: Para crear una flecha, pulse en el botón de flecha y arrástrela en el diagrama, o simplemente escriba el código en el editor.

El diagrama de clases muestra dependencias entre clases a modo de flechas. Las relaciones de herencia ("extends" o "implements") se muestran como flechas con punta hueca; las relaciones de tipo "usa" se muestran como flechas discontinuas con punta abierta.

Puede añadir dependencias tanto gráficamente (directamente en el diagrama) como textualmente en el código fuente. Si añade una flecha gráficamente, el código fuente es actualizado automáticamente; si añade la dependencia en el fuente, se actualiza el diagrama.

Para añadir una flecha gráficamente, pulse el botón de flecha apropiado (flecha hueca para "extender / implementar", o flecha discontinua para "usar") y arrastre la flecha de una clase a la otra.

Añadir una flecha de herencia inserta la definición "extends" o "implements" en el código fuente de la clase (dependiendo de si el objetivo es una clase o un interfaz).

Añadir una flecha de uso, no cambia inmediatamente el fuente (a no ser que el objetivo sea una clase de otro paquete. En ese caso se genera una sentencia "import", pero no hemos visto esto aún en nuestros ejemplos). Tener una flecha de uso en el diagrama apuntando a una clase que no se usa en el código generará posteriormente un aviso [warning] haciendo ver que la relación de uso de una clase se ha declarado pero dicha clase no se utiliza.

Añadir las flechas textualmente es sencillo: simplemente escriba el código como lo haría normalmente. Tan pronto como la clase es guardada, el diagrama se actualiza. (Y recuerde: cerrar el editor guarda automáticamente).

5.4 Eliminando elementos

Sumario: Para eliminar una clase o una flecha, seleccione la función Remove *de su menú emergente.*

Para eliminar una clase del diagrama, selecciónela y ejecute *Remove Class* en el menú *Edit*. También puede seleccionar *Remove* en el menú emergente de la clase. Ambas opciones funcionan también para las flechas: puede seleccionar la flecha primero y después seleccionar *Remove* del menú, o puede usarse el menú emergente de la flecha.

6 Utilizando la zona de código.

La zona de código de BlueJ permite la evaluación rápida y sencilla de “snippets” de Java (expresiones y sentencias). Por tanto la zona de código puede ser utilizada para investigar los detalles de la semántica de Java y para ilustrar y experimentar con la su sintaxis.

6.1 Mostrando la zona de código

Sumario: Para comenzar a utilizar la zona de código seleccione Show Code Pad *en el menú* View.

La zona de código no se muestra por defecto. Para mostrarla utilice el elemento *Show Code Pad* en el menú *View*. La ventana principal incluirá ahora la zona de código en la esquina inferior derecha, junto al banco de objetos (Figura 13). Los separadores verticales y horizontales de la zona de código y del banco de objetos pueden ser ajustados para cambiar sus dimensiones.

La zona de código podrá ser ya utilizada para introducir expresiones o sentencias. Pulsando Enter, cada línea será evaluada y podrá mostrarse un resultado.

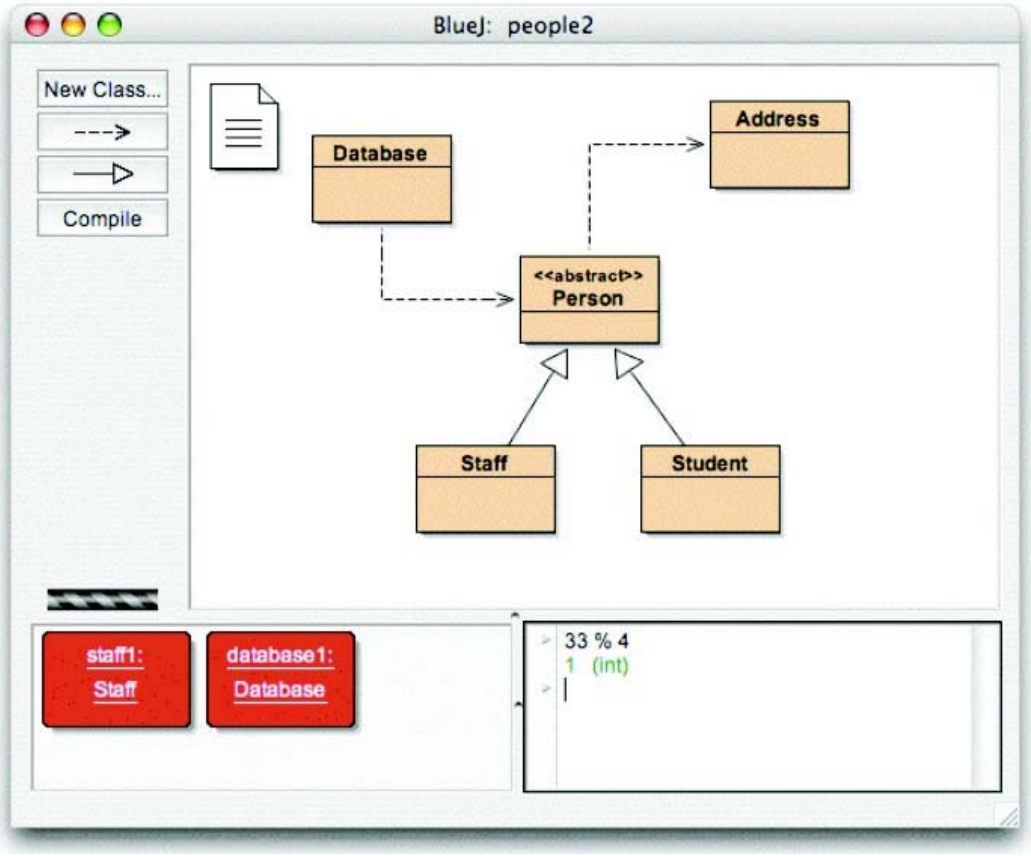

Figura 13: La ventana principal mostrando la zona de código.

6.2 Evaluación de expresiones simples

Sumario: Para evaluar expresiones Java, simplemente escríbalas en la zona de código.

La zona de código puede ser utilizada para evaluar expresiones simples. Pruebe a introducir, por ejemplo:

```
4 + 45
"hola".length()
Math.max(33, 4)
(int) 33.7
javax.swing.JOptionPane.showInputDialog(null, "Nombre:")
```

Las expresiones pueden referirse a objetos y valores estándar de Java, así como a clases del proyecto en curso. La zona de código mostrará el valor resultante, seguido por su tipo (entre paréntesis), o un mensaje de error si la expresión es incorrecta.

Puede utilizar también objetos que tenga en el banco de objetos. Pruebe lo siguiente: sitúe un objeto de la clase Student en el banco de objetos (utilizando el menú emergente de la clase como se ha descrito anteriormente). Llámelo *estudiante1*.

En la zona de código, ahora puede teclear:

```
estudiante1.getName()
```

Similarmente, puede referirse a todos los métodos disponibles de las clases de sus proyectos.

6.3 Recibiendo objetos

Sumario: Para transferir objetos de la zona de código al banco de objetos, arrastre el pequeño icono del objeto.

Algunos resultados de expresiones son objetos en lugar de valores simples. En este caso el objeto se muestra como una referencia a objeto <*object reference*>, seguida por el tipo del objeto y se muestra un pequeño icono representativo del objeto junto a la línea de resultado (figura 14).

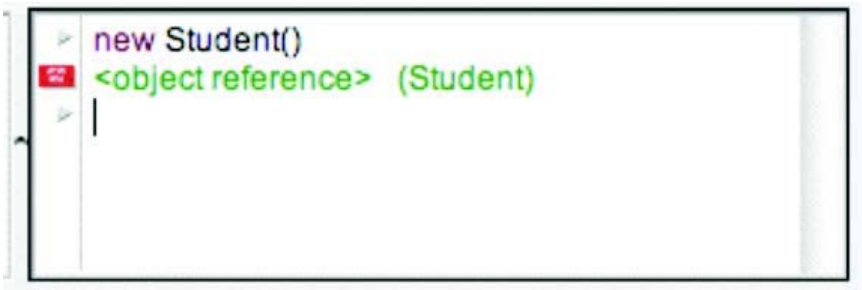

Figura 14: Un objeto como resultado de una expresión en la zona de código.

Si el resultado es una String, el valor de la String es presentado como resultado, pero también podrá ver el pequeño icono (ya que las Strings son objetos).

Las siguientes son algunas expresiones que puede probar para crear objetos:

```
new Student()
```

```
"mermelada".substring(3,8)
new java.util.Random()
"hola" + "mundo"
```

El pequeño icono del objeto puede utilizarse ahora para continuar trabajando con el objeto resultante. Se puede apuntar al icono y arrastrarlo al banco de objetos (Figura 15). Esto situará el objeto en el banco, donde estará disponible para futuras llamadas a sus métodos, bien vía su menú emergente o bien vía zona de código.

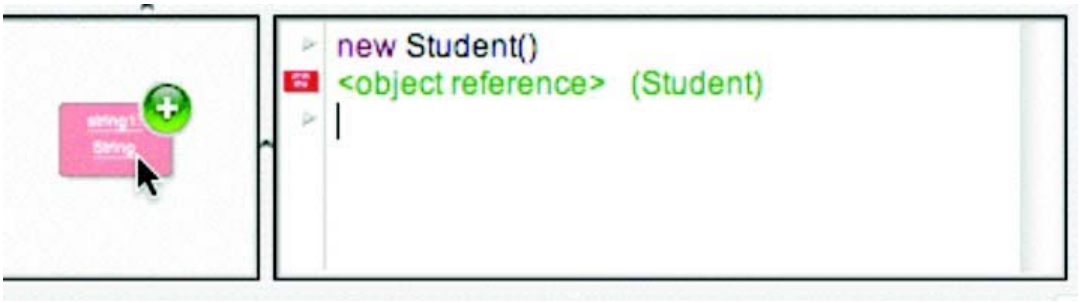

Figura 15: Arrastrando el objeto al banco de objetos.

6.4 Inspeccionando objetos

Sumario: Para inspeccionar objetos de resultado en la zona de código, hacer doble clic en el pequeño icono de objeto.

Si quiere inspeccionar un objeto obtenido como resultado de una expresión en la zona de código, puede hacerlo sin situar el objeto en el banco de objetos: puede simplemente hacer doble clic en el icono del objeto para abrir el inspector de objetos habitual.

6.5 Ejecutando sentencias

Sumario: Las sentencias que se introducen en la zona de código, son ejecutadas.

Puede también utilizar la zona de código para ejecutar sentencias (es decir instrucciones Java que no retornan un valor). Pruebe por ejemplo esto:

```
System.out.println("Ensalada de pepino");
System.out.println(new java.util.Random().nextInt(10));
```

Las sentencias se evalúan y ejecutan correctamente sin necesidad de poner punto y coma al final.

6.6 Sentencias multilínea y secuencias de sentencias

Sumario: Para introducir sentencias multilínea use shift-enter *al final de cada línea.*

Puede introducir secuencias de sentencias o sentencias que se extienden a lo largo de varias líneas utilizando *shift-Enter* al final de la línea introducida. Utilizando el cursor se moverá al principio de la siguiente línea, pero no será (todavía) ejecutada la entrada. Al final de la última línea pulse Enter para evaluar todas las líneas a la vez. Intente, por ejemplo, un ciclo *for*;

```
For (int i=0; i<5; i++ {
```

```
        System.out.println(“número: “ + i);
    }
```

6.7 Trabajando con variables

Sumario: pueden usarse variables locales en entradas multilínea únicas, los nombres de objetos en el banco de objetos pueden ser utilizados como campos de instancias.

En la zona de código puede trabajarse de modo restringido con variables –campos de la instancia y variables locales-.

Puede declarar variables locales en la zona de código, pero esto es sólo útil como parte de secuencias de sentencias multilínea, ya que las variables son eliminadas entre distintas entradas. Por ejemplo: puede introducir el siguiente bloque como una única entrada multilínea, y funcionará como es de esperar.

```
int sum;
sum = 0;
for (int i=0; i<100; i++) {
    sum += i;
}
System.out.println("La suma es: " + sum);
```

Sin embargo, introducir la misma secuencia como líneas de entrada separadas fallará, ya que la variable local *sum* no es recordada entre entradas.

Puede pensar en la entrada que se introduce como el texto de la definición de un método. Todo el código que es legal en el cuerpo de un método Java es también legal en la zona de código. Sin embargo, cada texto de entrada introducido forma parte de un método *diferente*, por lo que no puede hacerse referencia desde una línea de entrada a una variable declarada en otra.

Puede pensar en los objetos del banco de objetos como campos de una instancia. No puede definir ningún campo de instancia dentro del cuerpo de un método (o dentro de la zona de código).Pero puede referirse a los campos de instancia y hacer llamadas a los objetos que contienen.

Puede crear un nuevo campo de instancia arrastrando un objeto de la zona de código al banco de objetos.

6.8 Historial de mandos

Sumario: Para hacer uso del historial de mandos, utilice las teclas de flecha arriba y abajo.

La zona de código mantiene un historial de las entradas que se le han proporcionado con anterioridad. Utilizando las teclas de flecha arriba y flecha abajo, pueden rescatarse las líneas previas, que pueden reeditarse antes de ser reutilizadas.

7 Depuración [Debugging].

Esta sección presenta los aspectos más importantes de la funcionalidad de depuración en BlueJ. Hablando con profesores de computación, hemos oído muy a menudo el comentario de que sería interesante utilizar un depurador en el primer año de enseñanza, pero simplemente no hay tiempo para presentarlo. Los estudiantes se pelean con el editor, el compilador y la ejecución; no queda más tiempo para presentar otra complicada herramienta.

Por esto hemos decidido hacer el depurador lo más sencillo posible. El objetivo es tener un depurador que pueda explicarse en 15 minutos, y que los estudiantes lo puedan usar de ahí en adelante sin ninguna instrucción adicional. Veamos si hemos tenido éxito.

Antes de nada, hemos reducido la funcionalidad de los depuradores tradicionales a tres tareas:

- Poner puntos de ruptura
- Avanzar paso a paso por el código
- Inspeccionar variables.

Como resultado, cada una de las tres tareas es muy sencilla. Ahora probaremos cada una de ellas.

Para comenzar, abra el proyecto *debugdemo*, que se incluye en el directorio *examples* de la distribución. Este proyecto contiene unas pocas clases con el único propósito de demostrar la funcionalidad del depurador – no tienen demasiado sentido por otro lado.

7.1 Estableciendo puntos de ruptura [Breakpoints].

Sumario: Para poner un punto de ruptura, pulsar en el área de puntos de ruptura que está a la izquierda del texto en el editor.

Poner un punto de ruptura le permite interrumpir la ejecución en un determinado punto del código. Cuando la ejecución es interrumpida, puede investigar el estado de sus objetos. A menudo esto ayuda a entender que es lo que está sucediendo con su código.

El área de puntos de ruptura está en el editor, a la izquierda del texto (Figura 13). Puede establecer un punto de ruptura pulsando en ella. Aparecerá un pequeño signo de stop para marcar el punto de ruptura. Pruebe esto ahora. Abra la clase *Demo*, busque el método *loop*, y ponga un punto de ruptura en algún punto del bucle *for*. En su editor debe aparecer el signo de stop

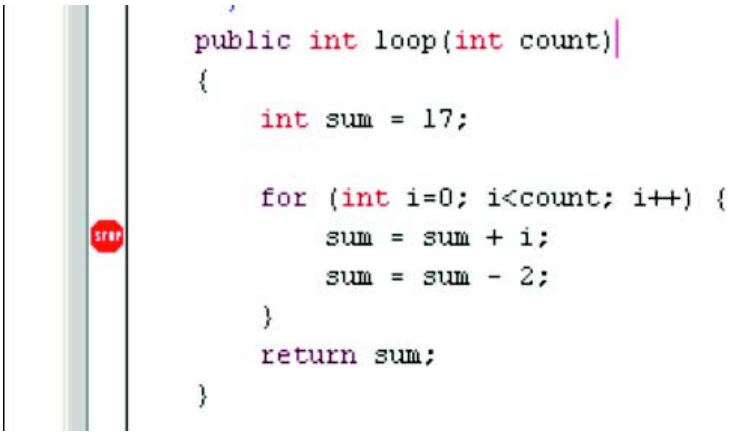

Figura 16: Un punto de ruptura.

Cuando se alcanza la línea de código que tiene asociado el punto de ruptura, la ejecución se interrumpirá. Probemos esto ahora.

Cree un objeto de clase *Demo* y llame al método *loop* con un parámetro de, digamos, 10. Tan pronto como se alcanza el punto de ruptura, emerge la ventana del editor, mostrando la línea de código actual y emerge una ventana de depurador. Esto tiene un aspecto semejante a la Figura 17.

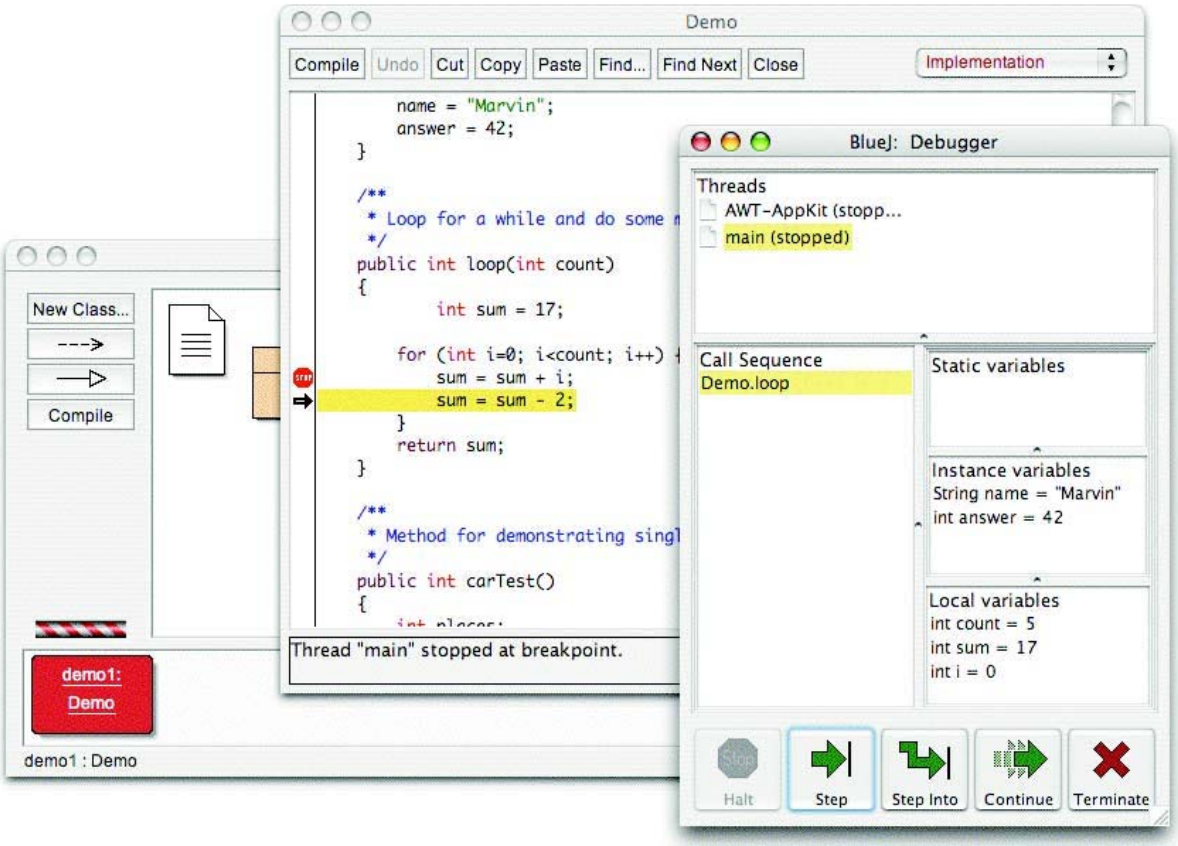

Figura 17: La ventana del depurador.

El resaltado en el editor muestra la línea que será ejecutada a continuación. (La ejecución se para antes de que esta línea sea ejecutada).

7.2 Avanzando paso a paso por el código.

Sumario: Para dar un paso sobre el código, use los botones Step *y* Step Into *en el depurador.*

Ahora que hemos parado la ejecución (lo que nos asegura de que el método realmente se esta ejecutando y que este punto en el código efectivamente se ha alcanzado), podemos avanzar paso a paso por el código y ver como progresa la ejecución. Para hacer esto, pulse repetidas veces en el botón *Step* en la ventana del depurador. Usted debe ver cambiar la línea fuente en el editor (el resaltado se mueve con la línea que se está ejecutando). Cada vez que usted pulsa el botón *Step*, se ejecuta una única línea de código y la ejecución se para de nuevo. Observe también que los valores de las variables mostrados en la ventana del depurador cambian (por ejemplo el valor de sum). Por lo tanto usted puede ejecutar paso a paso y observar qué sucede. Cuando se canse de esto, puede pulsar en el punto de ruptura de nuevo y eliminarlo, y entonces pulsar el botón *Continue* en el depurador para rearrancar la ejecución y continuar normalmente.

Probemos esto de nuevo con otro método. Ponga un punto de ruptura en la clase *Demo*, método *carTest ()*, en la línea que dice

```
Places = myCard.seats();
```

Llame al método. Cuando el punto de ruptura se alcanza, usted esta a punto de ejecutar una línea que contiene una llamada al método *seats()* en la clase *Car*. Pulsando *Step* daría un paso sobre la línea completa. Probemos *Step Into* esta vez. Si usted da un paso dentro [step into] de una llamada a un método, entra en el método y lo ejecuta línea a línea (no todo él como un único paso). En este caso usted es llevado dentro del método *seats()* en la clase *Car*. Felizmente usted puede dar pasos por este método hasta que llegue al final y volver al método que lo ha llamado. Observe como el depurador muestra cambios.

Step y *Step Into* se comportan de modo idéntico si la línea actual no contiene una llamada a un método.

7.3 Inspeccionando variables.

Sumario: Inspeccionar variables es fácil – son mostradas automáticamente en el depurador.

Cuando depura su código, es importante tener la capacidad de inspeccionar el estado de sus objetos (variables locales y variables de instancia).

Hacerlo es trivial – la mayor parte de esto ya ha sido visto. No necesita mandos especiales para inspeccionar variables; las variables estáticas, variables de instancia del objeto actual y variables locales del método actual son siempre mostradas y actualizadas automáticamente.

Puede seleccionar métodos en la secuencia de llamadas para ver variables de otros objetos y métodos actualmente activos. Pruebe, por ejemplo, un punto de ruptura en el método *carTest()* de nuevo. En la parte izquierda de la ventana del depurador, ve usted la secuencia de llamadas. Actualmente muestra

```
Car.seats
Demo.carTest
```

Esto indica que `Car.seats` ha sido llamado por `Demo.carTest`. Puede usted seleccionar `Demo.carTest` en esta lista para inspeccionar el código fuente y los valores actuales de las variables en este método.

Si da un paso rebasando la línea que contiene la instrucción `newCar(...)`, puede observar que el valor de la variable local *mycar* se muestra como <*objet reference*>. Todos los valores de tipos objeto

(excepto para las Strings) se muestran de esta manera. Puede inspeccionar esta variable haciendo doble clic sobre ella. Esto abrirá una ventana de inspección de objeto idéntica a las descritas anteriormente (sección 4.1). No hay diferencia entre inspeccionar objetos aquí e inspeccionar objetos en el banco de objetos.

7.4 Detener y terminar.

Sumario: Para parar una ejecución temporal o permanentemente puede usarse Halt *y* Terminate.

En ocasiones un programa funciona durante un largo periodo de tiempo y usted se pregunta si todo estará bien. Puede ser que haya un bucle infinito, puede que simplemente necesite todo ese tiempo. Bien, podemos comprobarlo. Llame al método *longloop()* de la clase *Demo*. Este tarda un cierto tiempo.

Ahora queremos saber qué es lo que está pasando. Si no se encuentra ya en la pantalla, muestre la ventana del depurador. (A propósito, pulsar en el símbolo que gira indicando que la máquina esta funcionando durante la ejecución es un atajo para mostrar el depurador).

Ahora pulse el botón *Halt*. La ejecución se interrumpe del mismo modo que si hubiésemos alcanzado un punto de ruptura. Ahora puede dar un par de pasos, observar las variables y ver que todo esta bien – simplemente necesita un poco más de tiempo para terminar. Puede pulsar *Continue* y *Halt* varias veces para ver cuan rápido está contando. Si no quiere usted continuar (por ejemplo, descubre que realmente está en un ciclo infinito) simplemente pulse *Terminate* para terminar por completo la ejecución. *Terminate* no debería usarse muy frecuentemente – puede dejar objetos que están perfectamente bien escritos en un estado inconsistente al terminar la ejecución de la máquina, por lo que es recomendable usarlo sólo como un mecanismo de emergencia.

8 Creando aplicaciones autónomas ("stand-alone").

Sumario: Para crear una aplicación autónoma use Project – Create Jar File...

BlueJ puede crear ficheros ejecutables jar. Los ficheros ejecutables jar pueden ser ejecutados en algunos sistemas haciendo doble clic en el fichero (por ejemplo en Windows y MacOS X), o mediante el mando `java –jar <nobre-del-fichero>.jar` (Unix o DOS).

Probaremos esto con el proyecto ejemplo *hello*. Ábralo (está en el directorio de *examples*). Asegúrese de que el proyecto está compilado. Seleccione la función *Create Jar File...* del menú *Project*.

Se abre un cuadro de diálogo que le permite elegir el modo de almacenamiento (Figura 15). Elija "*jar file*" para crear un fichero jar ejecutable. Para hacer que el fichero jar sea ejecutable, debe especificar también una clase principal. Esta clase debe tener definido un método *main* válido (con el prototipo `public static void main(String[] args)`).

Figura 18: El dialogo "Export"

En nuestro ejemplo, elegir la clase principal es fácil: sólo hay una clase. Seleccione *Hello* en el menú emergente. Si tiene otro proyecto, seleccione la clase que contenga el método "main" que quiere que sea ejecutado.

Normalmente, no incluirá usted los ficheros fuentes en el fichero jar. Pero puede hacerlo si quiere distribuir también sus fuentes. (Puede usar el formato jar para, por ejemplo, enviar el proyecto completo a alguien vía e-mail en un único fichero).

Si ha configurado BlueJ para utilizar bibliotecas de usuario (bien vía el parámetro [setting] *Preferences/Libraries*, o bien utilizando el directorio *lib/userlib*), verá un área titulada *Include user libraries* en el centro del cuadro de diálogo. (Si no esta utilizando ninguna biblioteca, esta área estará ausente). Debe seleccionar todas las bibliotecas que su proyecto actual utilice.

Pulse *Continue*. A continuación verá un cuadro de diálogo de selección de ficheros que le permite especificar un nombre para el fichero jar a crear. Teclée *hello* y pulse *Create*.

Si no tiene ninguna biblioteca que incluir, el fichero *hello.jar* será creado en este momento. Si tiene bibliotecas, se creará in directorio llamado *hello*, y en su interior se encontrará el fichero jar *hello.jar*.

El directorio también contiene todas las bibliotecas necesarias. Su fichero jar espera encontrar todas las bibliotecas referenciadas en su mismo directorio – por tanto asegúrese de mantener todos estos ficheros jar juntos cuando los cambie de lugar.

Puede hacer doble clic en el fichero jar sólo si la aplicación tiene un interfaz gráfico (GUI). Nuestro ejemplo usa entrada/salida de texto, por lo que debemos arrancarlo en un terminal de texto. Probemos ahora a ejecutar el fichero jar.

Abra un terminal o ventana de DOS. Una vez hecho esto, vaya al directorio donde salvó el fichero jar (debe ver el fichero *hello.jar*). Asumiendo que Java está instalado correctamente en su sistema, debe usted poder escribir

```
java –jar hello.jar
```

para ejecutar el fichero.

9 Creando applets.

9.1 Poniendo en marcha un applet.

Sumario: Para poner en marcha un applet, seleccione Run Applet *en el menú emergente del applet.*

BlueJ permite crear y ejecutar applets además de aplicaciones. Hemos incluido un applet en el directorio de ejemplos de la distribución. Primero, queremos probar la ejecución de un applet. Abra el proyecto *appletdemo* de los ejemplos.

Verá que este ejemplo tiene sólo una clase; se llama *CaseConverter*[*]. El icono de la clase está marcado (con el distintivo `<<applet>>`) como un applet. Después de compilar, seleccione el mando *Run Applet* del menú emergente de la clase.

Aparece un cuadro de diálogo que le permite realizar varias elecciones (Figura 19).

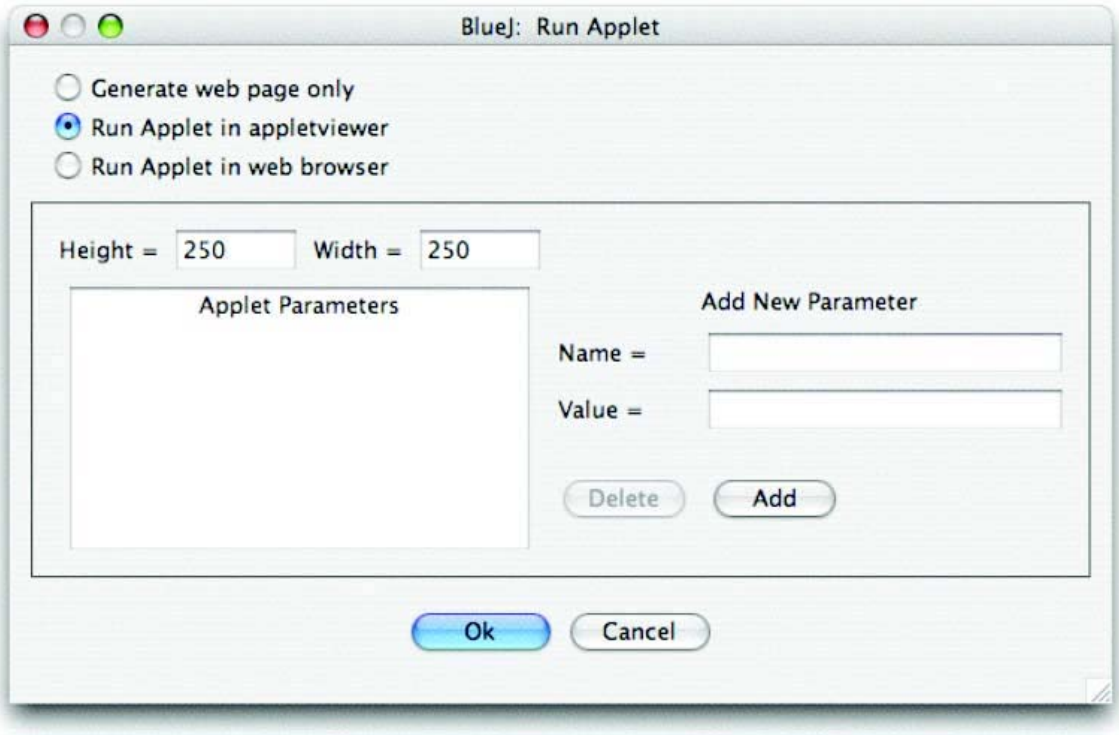

Figura 19: El dialogo "Run Applet"

Puede ver que tiene la opción de ejecutar el applet en un navegador [web browser] o en un visor de applets [appletviewer] (o simplemente generar la página web sin ponerla en marcha). Deje los valores por defecto y pulse OK. Tras unos escasos segundos, aparecerá un visor de applets mostrando el applet de conversión de capitalización.

El visor de applets se instala conjuntamente con el J2SE SDK (su instalación de Java), por lo que está garantizada la coincidencia en su versión con el compilador de Java. En general causa menos problemas de lo que hacen los navegadores. Puede que su navegador maneje una versión distinta de

[*] N.T.-*conversor de capitalización de letras [mayúsculas/minúsculas]*

Java y, dependiendo de qué versión de qué navegador use, puede causar problemas. De todos modos debería funcionar bien con los navegadores más normales.

En los sistemas Microsoft Windows y MacOS, BlueJ utiliza el navegador establecido por defecto. En los sistemas Unix, el navegador se define en los parámetros [settings] de BlueJ.

9.2 Creando un applet.

Sumario: Para crear un applet, pulse el botón New Class *y seleccione* Applet *como el tipo de clase.*

Después de ver como ejecutar un applet, queremos crear uno nosotros mismos.

Cree una nueva clase con *Applet* como tipo de clase (puede seleccionar el tipo en el cuadro de diálogo *New Class*). Compile, y entonces ejecute el applet. ¡Ya está!. No ha sido demasiado difícil, ¿no?

Los applets (como otras clases) se generan con un esqueleto por defecto que contiene algo de código válido. Para los applets, este código muestra un applet sencillo con dos líneas de texto. Ahora puede abrir el editor y editar el applet para insertar su propio código.

Verá que todos los métodos comunes de los applets están ahí, cada uno con un comentario explicando su propósito. Todo el código del ejemplo está dentro del método *paint*.

9.3 Comprobando el applet.

En algunas ocasiones puede ser útil crear un objeto applet en el banco de objetos (como para las clases normales). Puede hacerlo – en el menú emergente del applet se muestra el constructor. No puede ejecutar el applet completo desde el banco de objetos, pero puede llamar a algunos de sus métodos. Esto puede ser útil para comprobar aisladamente métodos que ha escrito como parte de la implementación de su applet.

Si pone puntos de ruptura en un applet, no tendrán efecto cuando corran en un visor de applets o en un navegador Web. Esto es debido a que ambos, el visor de applets y el navegador Web, utilizan sus propias máquinas virtuales para ejecutar el applet, que no saben nada de los puntos de ruptura de BlueJ.

Si quier utilizar puntos de ruptura y avanzar paso a paso en un applet, puede utilizar la clase *AppletWindow*, escrita por Michael Trigoboff. Esta clase proporciona un marco de aplicación [frame] que le permite correr el applet directamente dentro de BlueJ, de manera que los métodos de depuración normales funcionan. Puede encontrar esta clase en la sección *Resources* del sitio web de BlueJ.

10 Otras operaciones.

10.1 Abriendo paquetes no-BlueJ con BlueJ.

Sumario: Pueden abrirse paquetes no-BlueJ con el mando Project: Open Non BlueJ...

BlueJ le permite abrir paquetes existentes creados sin BlueJ. Para hacer esto, seleccione *Project - Open Non BlueJ*... en el menú. Seleccione el directorio que contiene los ficheros fuente Java, y entonces pulse el botón Open in BlueJ. El sistema le pedirá confirmación de que quiere abrir ese directorio.

10.2 Añadiendo clases existentes al proyecto.

Sumario: Pueden copiarse clases a un proyecto desde fuera utilizando el mando Add Class from File....

A menudo, usted puede querer utilizar en su proyecto BlueJ una clase que ha obtenido de algún sitio. Por ejemplo, un profesor puede dar una clase Java para que los estudiantes la usen en un proyecto. Puede incorporar fácilmente una clase existente su proyecto seleccionando *Edit - Add Class From File* en el menú. Esto le permitirá seleccionar un fichero fuente Java (con nombre terminado en .java) para ser importado.

Cuando se importa la clase al proyecto, se hace una copia que se almacena en el directorio actual del proyecto. El efecto es exactamente el mismo que si usted hubiese creado una clase nueva y escrito todo su código.

Una alternativa es añadir el fichero fuente de la clase al directorio desde fuera del BlueJ. La próxima vez que se abra el proyecto, la clase será incluida en el diagrama del proyecto.

10.3 Llamando a "main" y a otros métodos estáticos.

Sumario: Los métodos estáticos pueden ser llamados desde el menú emergente de la clase.

Abra el proyecto *hello* del directorio *examples*. La única clase en el proyecto (la clase *Hello*) define un método principal estándar.

Pulse el botón derecho sobre la clase y verá que el menú incluye no sólo el constructor de la clase, sino que también el método estático *main*. Puede llamar ahora directamente al *main* desde este menú (sin crear primero un objeto).

Todos los métodos estáticos pueden ser llamados de este modo. El método principal estándar espera recibir un array de Strings como argumento. Puede pasar un array de Strings usando la sintaxis estándar de Java para los arrays de constantes. Por ejemplo, puede pasar

```
{“uno”, “dos”, ”tres”}
```

(incluyendo las llaves) al método. ¡Pruébelo!

Nota “marginal”: En Java estándar, los arrays de constantes no pueden ser usados como argumentos en llamadas a métodos. Sólo pueden ser utilizados como inicializadores. En BlueJ, para permitir llamadas interactivas a los métodos principales estándar, permitimos el paso de arrays de constantes como parámetros.

10.4 Generando documentación.

Sumario: Para generar documentación de un proyecto, seleccione Project Documentation *del menú* Tools*.*

Puede generar documentación para su proyecto en el formato estándar de javadoc desde dentro de BlueJ. Para hacer esto, seleccione *Tools – Project Documentation* en el menú. Esta función generará la documentación para todas las clases en un proyecto a partir del código fuente de las mismas y abrirá el navegador para mostrarla.

También puede generar y ver la documentación para una sola clase en el editor de BlueJ. Para hacer esto, abra el editor y use el menú emergente en su barra de herramientas. Cambie la selección de *Implementation* a *Interface*. Esto mostrará la documentación al estilo de javadoc (el interfaz de la clase) en el editor.

10.5 Trabajando con bibliotecas.

Sumario: La API de clases estándar de Java puede consultarse seleccionando Help - Java Standard Libraries*.*

Frecuentemente, cuando se escribe un programa en Java, se necesita hacer referencia a la bibliotecas estándar de Java. Puede abrir un navegador mostrando la documentación de la API de JDK seleccionando *Help - Java Standard Libraries* en el menú (si se encuentra conectado a la red).

También se puede instalar y utilizar la documentación del JDK localmente (offline). Los detalles para ello están explicados en la sección de ayuda del sitio de BlueJ en la red[2].

10.6 Creando objetos a partir de clases de biblioteca.

Sumario: Para crear objetos a partir de clases de biblioteca, use Tools – Use Library Class*.*

BlueJ también ofrece una función para crear objetos a partir de clases que no son parte de su proyecto, sino que se encuentran definidas en la biblioteca. Puede, por ejemplo, crear objetos de clase String o ArrayList. Esto puede ser muy útil para experimentar de un modo rápido con estos objetos de biblioteca.

---

[2] NOTA DEL TRADUCTOR: No es precisamente inmediato encontrar esta información en la red, y como realmente no puede ser más sencillo, pues ahí va:

Evidentemente debe disponer de una copia local de la documentación. Si no es así, puede descargarla de http://java.sun.com/j2se. Seleccione *Preferences* en el menú *Tools*. Seleccione la solapa “Miscellaneous”. Verá un campo etiquetado como "JDK documentación URL". Debe poner ahí la dirección URL de su copia local de la documentación (p. ej. -en una instalación particular para Windows- C:\jdk1.5.0\docs\api\index.html).

Puede crear objetos de biblioteca seleccionando *Tools – Use Library Class* del menú. Surgirá un cuadro de diálogo que le solicitará que introduzca un nombre de clase que la determine completamente, como por ejemplo java.lang.String. (Observe que debe introducir el nombre completamente determinado, esto es el nombre incluyendo los nombres de paquetes que contienen a la clase).

El campo de introducción de texto tiene asociado un menú emergente que muestra las clases utilizadas recientemente. Una vez que se ha introducido un nombre de una clase, pulsar la tecla Enter mostrará todos los constructores y métodos estáticos de la clase en una lista en el cuadro de diálogo. Seleccionándolos en la lista, pueden ser invocados cualquiera de estos constructores o métodos estáticos.

La invocación procede como cualquier otro constructor o llamada a método.

11 Sólo los sumarios.

Comenzando

1. Para abrir un proyecto, seleccione *Open* del menú Project
2. Para crear un objeto, seleccione un constructor del menú emergente de la clase.
3. Para ejecutar un método, selecciónelo en el menú emergente del objeto.
4. Para editar el código fuente de una clase, haga doble-clic en su icono.
5. para compilar una clase, pulse el botón *Compile* del editor. Para compilar un proyecto, pulse el botón *Compile* en la ventana del proyecto.
6. Para obtener ayuda para un mensaje de error de compilación, pulse la interrogación que está próxima al mensaje de error.

Haciendo un poco más

7. La inspección de objetos permite realizar una depuración sencilla al mostrar el estado interno de un objeto.
8. Se puede pasar un objeto como parámetro a un método pulsando en el icono del objeto.

Creando un nuevo proyecto

9. Para crear un proyecto seleccione *New...* del menú *Project*.
10. Para crear una clase, pulse el botón *New Class* y especifique el nombre de la clase.
11. Para crear una flecha, pulse en el botón de flecha y arrástrela en el diagrama, o simplemente escriba el código en el editor.
12. Para eliminar una clase o una flecha, seleccione la función *Remove* de su menú emergente.

Utilizando la zona de código.

13. Para comenzar a utilizar la zona de código seleccione *Show Code Pad* en el menú *View*.
14. Para evaluar expresiones Java, simplemente escríbalas en la zona de código.
15. Para transferir objetos de la zona de código al banco de objetos, arrastre el pequeño icono del objeto.
16. Para inspeccionar objetos de resultado en la zona de código, hacer doble clic en el pequeño icono de objeto.
17. Las sentencias que se introducen en la zona de código, son ejecutadas.
18. Para introducir sentencias multilínea use *shift-enter* al final de cada línea.
19. pueden usarse variables locales en entradas multilínea únicas, los nombres de objetos en el banco de objetos pueden ser utilizados como campos de instancias.
20. Para hacer uso del historial de mandos, utilice las teclas de flecha arriba y abajo.

Depuración [Debugging].

21. Para poner un punto de ruptura, pulsar en el área de puntos de ruptura que está a la izquierda del texto en el editor.
22. Para dar un paso sobre el código, use los botones *Step* y *Step Into* en el depurador.
23. Inspeccionar variables es fácil – son mostradas automáticamente en el depurador.
24. Para parar una ejecución temporal o permanentemente puede usarse *Halt* y *Terminate*.

Creando aplicaciones autónomas [stand-alone].

25. Para crear una aplicación autónoma use *Project – Create Jar File...*

Creando applets.

26. Para poner en marcha un applet, seleccione *Run Applet* en el menú emergente del applet.
27. Para crear un applet, pulse el botón *New Class* y seleccione *Applet* como el tipo de clase.

Otras operaciones.

28. Pueden abrirse paquetes no-BlueJ con el mando *Project: Open Non BlueJ...*
29. Pueden copiarse clases a un proyecto desde fuera utilizando el mando *Add Class from File....*
30. Los métodos estáticos pueden ser llamados desde el menú emergente de la clase.
31. Para generar documentación de un proyecto, seleccione *Project Documentation* del menú *Tools*.
32. La API de clases estándar de Java puede consultarse seleccionando *Help - Java Standard Libraries*.
33. Para crear objetos a partir de clases de biblioteca, use *Tools – Use Library Class*.

