There are no translations available.

El rey actual de los lenguajes de programación, Java, se está convirtiendo por méritos propios en el imprescindible protagonista de todos los cursos, asignaturas y módulos profesionales que versan sobre iniciación a la programación. Pero Java es un lenguaje orientado a objetos, y esto supone un desafío para todos aquellos que, incluso teniendo gran experiencia en lenguajes estructurados como Pascal o C, quieren aprenderlo o enseñarlo. Las peculiaridades de este tipo de lenguajes han motivado la aparición de entornos de programación específicamente adaptados para su enseñanza: este monográfico te presenta dos de los más utilizados (BlueJ y jGrasp) confrontándolos con un entorno de desarrollo profesional (NetBeans).

En muy poco tiempo aprenderás a ponerlos en funcionamiento de forma sencilla, y conocerás sus funcionalidades más interesantes de cara a la enseñanza y el aprendizaje de Java. Al acabar de leer estos artículos estarás en condiciones de comenzar a trabajar y, lo más importante, tendrás las claves para escoger el entorno de desarrollo que más se ajuste a tus necesidades.

1. Introducción

Hasta hace no demasiados años, los compiladores se utilizaban introduciendo órdenes escritas en la línea de comandos. El proceso no era sencillo: para empezar, el programador tenía que preparar el código fuente usando alguno de los rudimentarios editores de texto existentes; a continuación abandonaba el editor para ejecutar la llamada al compilador y obtener el código compilado. Después solía ser necesario utilizar otra herramienta para el ensamblado del programa. Finalmente, podía probar (también desde la línea de comandos) el archivo ejecutable: pero ante cualquier error detectado, debía volver a empezar este proceso.

No es de extrañar, por tanto, que poco a poco fuera desarrollándose el concepto de lo que se ha terminado llamando IDE (Integrated Development Environment o entorno de desarrollo integrado). Como su nombre indica, su objetivo es integrar en una misma aplicación las herramientas necesarias para realizar todo el proceso de programación: edición, compilación, revisión de errores, depuración, ejecución.... El resultado es que utilizando un IDE, compilar y ejecutar un programa requiere únicamente pulsar un botón.

Existen muchísimos entornos para cada lenguaje de programación, y cada uno de ellos ofrece al usuario unas determinadas funcionalidades. Escoger uno u otro dependerá tanto de nuestras necesidades como de lo intuitivo y agradable que resulte trabajar en uno u otro entorno.

Normalmente el programador va probando entornos hasta que encuentra aquel con el que se encuentra más cómodo, pero en el caso que nos ocupa (la enseñanza de Java) nos interesa ir sobre seguro y escoger un entorno que ofrezca al alumno sencillez de uso, facilidad para comprender las ideas subyacentes en el proceso de programación, y estimulación para animarse a hacer programas por sí mismo. Cualquiera de los tres entornos que se presentan en este monográfico será una buena elección de cara a la enseñanza del lenguaje.

Antes de empezar, ten en cuenta que un entorno de desarrollo no es más que una fachada para el proceso de compilación y ejecución de un programa. Eso quiere decir que no basta con tener un IDE instalado en el ordenador para empezar a trabajar: previamente debemos instalar el compilador, un proceso que en Java suele generar bastante confusión.

En esta primera entrega nos centraremos precisamente en eso: la instalación del compilador en nuestro equipo. Además, se explicarán las particularidades de los lenguajes orientados a objetos que tendremos en cuenta a la hora de escoger nuestro entorno.

2. A vueltas con las siglas: ¿JRE ó JDK?

Si has intentado alguna vez abordar la programación en Java quizá te hayas desanimado antes de empezar, y es que la instalación de Java resulta algo compleja: veamos por qué.

Sabemos que existen lenguajes compilados y lenguajes interpretados:

- Los lenguajes compilados traducen el programa fuente a código que puede ejecutar el ordenador, y que por tanto es dependiente de la máquina en la que se va a ejecutar o, como mínimo, de su sistema operativo. Por ejemplo, si compilo mi programa en un equipo con Linux no podré utilizar el archivo ejecutable resultante en Windows.
- Los lenguajes interpretados, por el contrario, se van traduciendo sobre la marcha, por lo que no se necesita compilador pero sí un intérprete que también será diferente en función de la

máquina en la que se instale. La interpretación requiere tiempo: por eso la ejecución de los lenguajes interpretados es más lenta que la de sus homólogos compilados.

¿Y qué es Java? ¿Compilado o interpretado? Pues ninguna de las dos, o mejor dicho las dos a la vez. Java compila el programa a un lenguaje intermedio *predigerido* llamado bytecode (archivos con extensión *.class*) común a todas las plataformas. A partir de ahí funciona como un lenguaje interpretado gracias al entorno de ejecución de Java, un intérprete diferente para cada plataforma y que deberá estar instalado en nuestro ordenador si queremos ejecutar programas escritos en Java. Las continuas mejoras en esta máquina virtual realizadas por Sun, la empresa creadora del lenguaje, hacen que las diferencias de velocidad de ejecución entre Java y un lenguaje totalmente compilado (como C) sean cada vez menores.

El resultado es el exhibido por el lema de Java: *compile o-nce, run anywhere*. Es decir, sólo necesitas compilar el programa una vez y podrá ejecutar en múltiples y diversas máquinas, siempre que éstas tengan instalado el entorno de ejecución.

Ahora podemos comprender mejor los dos paquetes que nos ofrece Sun para instalar Java:

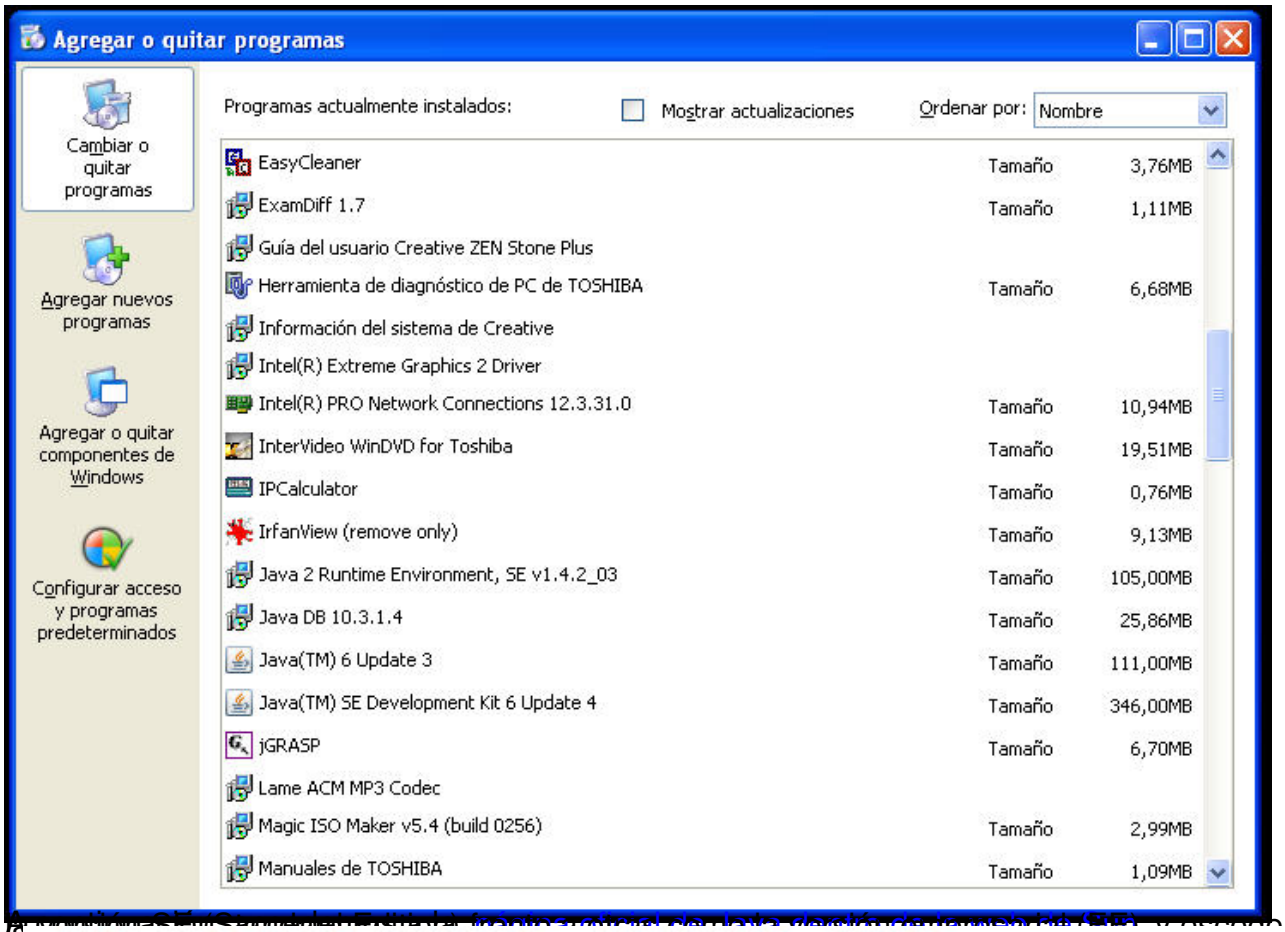
- **JRE (Java Runtime Environment)** es el entorno de ejecución necesario para interpretar y ejecutar archivos *.class*. Es casi seguro que cualquier ordenador que encuentres en el centro de trabajo tendrá instalada una versión (a veces varias) de este complemento.

- **JDK (Java Development Kit)** es el kit de desarrollo, es decir, el compilador para poder crear programas en Java. Esta será, por tanto, la opción que tendremos que escoger en nuestro caso. Ten en cuenta que la instalación del JDK ya incluye el entorno de ejecución, por tanto no tendrás que hacer ninguna instalación adicional.

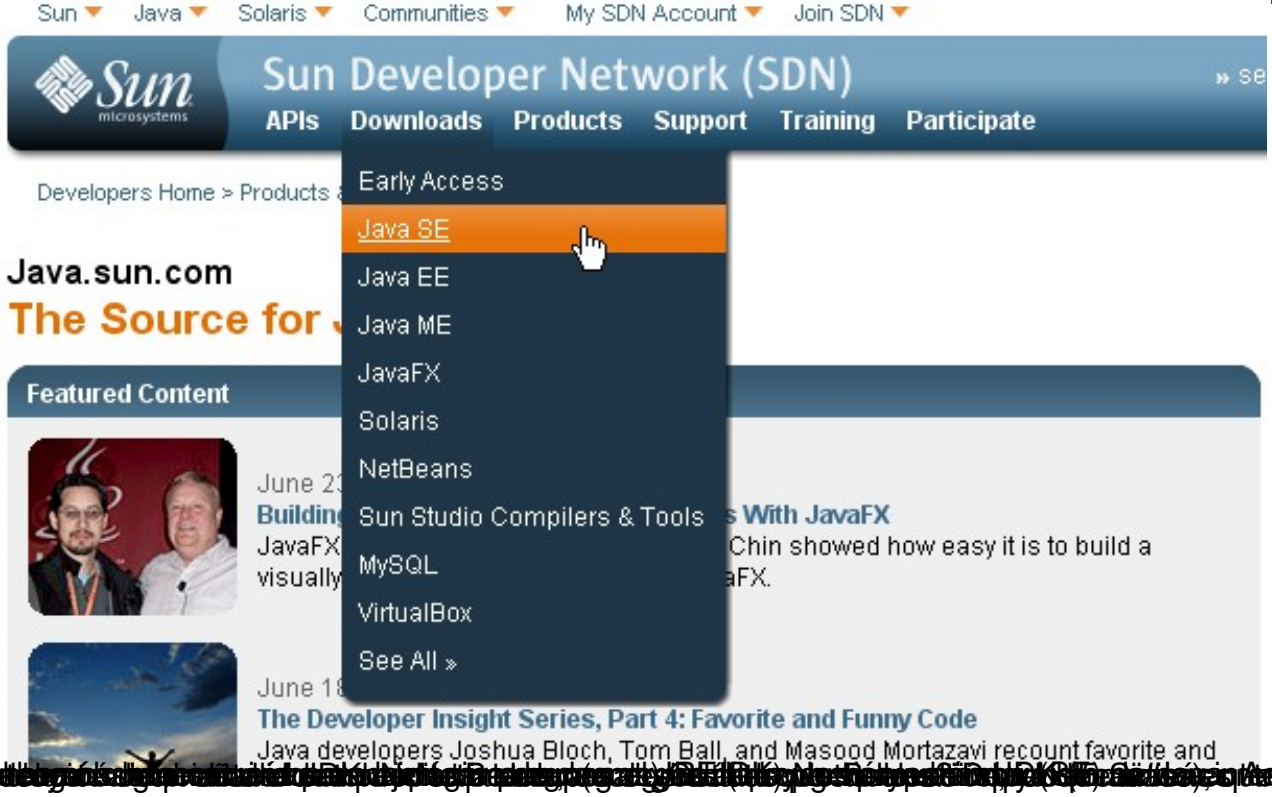
Hemos dicho que los entornos de desarrollo requieren tener instalado el compilador. Por ejemplo, si intentas ejecutar BlueJ sin haber instalado previamente el JDK, obtendrás este mensaje:

Algo parecido ocurre en jGrasp:

Entonces, ¿qué debemos hacer? Si contamos con el consentimiento del resto de usuarios del equipo, lo mejor es desinstalar **todas** las versiones existentes de Java y reinstalar únicamente el JDK más actual. Por ejemplo, en la siguiente figura conviven dos entornos de ejecución, uno antiguo (versión 1.4) y otro más moderno (versión 6 actualización 3), así como un JDK (versión 6, actualización 4). No te preocupes demasiado por Java DB: se trata de un añadido para bases de datos que se instala junto con el JDK. Por tanto, en este caso desinstalaríamos los cuatro elementos que empiezan por *Java:*


Agregar o quitar programas
Cambiar o quitar programas
Agregar nuevos programas
Agregar o quitar componentes de Windows
Configurar acceso y programas predeterminados
Programas actualmente instalados:
Mostrar actualizaciones
Ordenar por: Nombre
EasyCleaner Tamaño 3,76MB
ExamDiff 1.7 Tamaño 1,11MB
Guía del usuario Creative ZEN Stone Plus
Herramienta de diagnóstico de PC de TOSHIBA Tamaño 6,68MB
Información del sistema de Creative
Intel(R) Extreme Graphics 2 Driver
Intel(R) PRO Network Connections 12.3.31.0 Tamaño 10,94MB
InterVideo WinDVD for Toshiba Tamaño 19,51MB
IPCalculator Tamaño 0,76MB
IrfanView (remove only) Tamaño 9,13MB
Java 2 Runtime Environment, SE v1.4.2_03 Tamaño 105,00MB
Java DB 10.3.1.4 Tamaño 25,86MB
Java(TM) 6 Update 3 Tamaño 111,00MB
Java(TM) SE Development Kit 6 Update 4 Tamaño 346,00MB
jGRASP Tamaño 6,70MB
Lame ACM MP3 Codec
Magic ISO Maker v5.4 (build 0256) Tamaño 2,99MB
Manuales de TOSHIBA Tamaño 1,09MB

la [illegible] mos


Sun Java Solaris Communities My SDN Account Join SDN
Sun Developer Network (SDN)
APIs Downloads Products Support Training Participate
Early Access
Java SE
Java EE
Java ME
JavaFX
Solaris
NetBeans
Sun Studio Compilers & Tools
MySQL
VirtualBox
See All »
Developers Home > Products
Java.sun.com
The Source for
Featured Content
June 18
The Developer Insight Series, Part 4: Favorite and Funny Code
Java developers Joshua Bloch, Tom Ball, and Masood Mortazavi recount favorite and

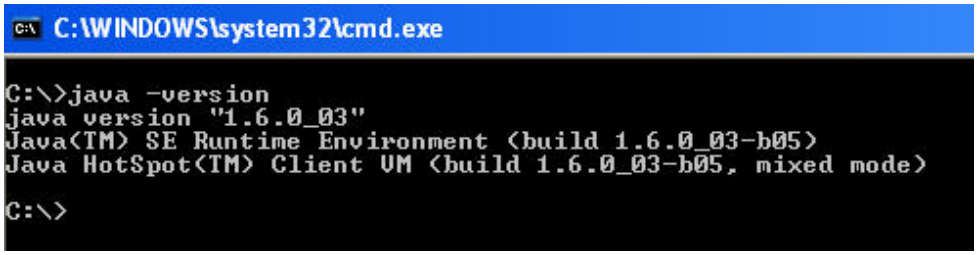
[illegible] a,

3. El problema de las versiones

El proceso de instalación del JDK es trivial, y no reviste ninguna dificultad. Sin embargo, cuando empieces a trabajar es posible que te enfrentes a ciertos problemas relativos a las distintas versiones existentes en los equipos tanto del JDK como del JRE. Este apartado te previene para que no desesperes si te ocurren y sepas reaccionar, pero si lo deseas puedes posponer su lectura y regresar sólo si encuentras dificultades.

Supongamos que hemos estado desarrollando un programa en el ordenador del centro de trabajo, y nos lo llevamos a casa. Es posible que al intentar ejecutarlo en tu equipo encuentres este críptico mensaje de error: Unsupported major.minor version. ¿Cómo es posible, si en el centro ejecutaba perfectamente? Se trata de un error muy común, y la explicación es la siguiente: con un JRE moderno puedes ejecutar programas compilados con un JDK antiguo, pero no sucede lo mismo al revés. Un JRE antiguo no sabrá ejecutar programas compilados con un JDK moderno. En nuestro caso de ejemplo, en el equipo de casa teníamos una versión de Java más antigua, y eso ha motivado el problema. ¿La solución? Lo ideal será instalar en casa la última versión de Java, pero existe una alternativa más rápida: elimina los archivos compilados (.class) del proyecto para forzar a tu equipo a recompilar las clases usando el compilador más antiguo.

Este problema afecta también con frecuencia a quienes prefieren trabajar a la antigua usanza, es decir, con la línea de comandos. El caso típico es el siguiente: tenemos instalada la última versión del JDK, compilamos el programa pero no conseguimos hacerlo ejecutar. El motivo es el mismo: tratamos de ejecutar un programa con un JRE obsoleto, seguramente sin darnos cuenta. El diagnóstico es sencillo: ejecuta el comando java -version y sabrás qué versión del JRE estás utilizando:



Si al hacerlo ves una versión antigua (por ejemplo 1.4), ya has detectado el problema. Pero ¿cómo es posible, si instalaste la última versión de Java? Ten en cuenta que algunas aplicaciones (por ejemplo ciertos paquetes del gestor de bases de datos Oracle) pueden llevar incluido un entorno de ejecución sin que a veces seamos conscientes de ello, y por llevar instalados más tiempo en nuestro equipo, su ruta aparece antes en la variable de sistema PATH. La solución es sencilla: edita esta variable y sitúa al principio del todo la ruta a nuestra última versión de Java.

Volviendo a la actualización, seguramente estés pensando: ¿puedo realmente actualizar el JRE y el JDK sin perjudicar a otras personas que utilicen el mismo equipo? La respuesta, siempre que actualicemos a versiones más modernas, es sí; pero estrictamente hablando debemos matizar esta afirmación. Como todos los lenguajes Java va evolucionando, y algunas características se convierten en obsoletas (*deprecated*). Es posible que, al cabo de algunas versiones, al intentar compilar un programa muy antiguo haya alguna sentencia que con la nueva versión provoque un fallo de compilación. Sin embargo, es prácticamente imposible que eso ocurra en el nivel en que nos encontramos y además, incluso en ese caso, lo correcto sería corregir las escasas líneas que dieran problemas para actualizarlas según la sintaxis actual del lenguaje. Así que no temas actualizar el JDK: siempre será una buena idea.

4. Las peculiaridades de enseñar Java

Cuando una persona aprende a programar en Pascal, Modula 2 ó C, no está simplemente aprendiendo esos lenguajes: está comprendiendo también el paradigma de programación estructurada, y por tanto sabrá pasar de un lenguaje a otro con gran facilidad, tan pronto como se acostumbre a las peculiaridades sintácticas de cada lenguaje. De igual forma, cuando aprendemos Java debemos desentrañar el paradigma de programación orientada a objetos, que difiere bastante del anterior. Por tanto debemos tener en cuenta que, incluso si nuestros alumnos tienen nociones de programación en lenguajes estructurados - o nosotros mismos como docentes las tenemos - será necesario cierto esfuerzo para cambiar el enfoque sobre la forma de diseñar y abordar nuestros programas.

Explicar las características de Java excede el propósito de este monográfico, pero es interesante destacar algunos aspectos que afectarán a la decisión de qué entorno de trabajo escoger. Y es que si en los lenguajes clásicos nos enfrentábamos a un problema pensando **cómo** hay que resolverlo, en Java pensaremos **qué** necesitamos para resolverlo. En efecto, un programa en Java es como una obra de teatro en la que debemos preparar primero cada personaje, definir qué características tiene y qué va a saber hacer: una vez ese trabajo previo esté realizado, la obra se desarrollará sacando personajes a escena y haciéndoles interactuar.

Cuando hablamos de preparar personajes, nos referimos a programar *clases*. Igual que tenemos un tipo de datos entero para representar números y un tipo String para representar cadenas de texto, ¿no sería perfecto disponer de un tipo Cuadrado, Persona, Avión, Factura, ListaDeClientes, etc. según las necesidades de nuestro problema? Pues eso es precisamente lo que haremos en Java: crearemos una *clase*

Rectángulo y definiremos sus atributos (la base, la altura, el color...) y sus métodos (lo que sabe hacer: calcular su área, dibujarse en la pantalla...). Una vez esté hecha, en mi programa principal podré crear todos los objetos que quiera de esa clase y personalizarlos con sus distintos atributos. Así, las clases y los objetos son el equivalente a tipos y variables: igual que declaro una variable entera
x
del tipo entero, creo un objeto
juan
de la clase Cliente. La diferencia es que el objeto
juan
no es una simple variable: incluye todos sus datos personales, y es capaz de hacer cosas como calcular su edad - con el código juan.calcularEdad()-, imprimir sus ventas - juan.imprimirDatosVentas() -, y todo aquello para lo que le hayamos preparado.

Pero si lo que definimos son únicamente tipos, ¿dónde está el auténtico programa principal? Debe existir un método especial, llamado *main*, que sí nos recordará a los lenguajes estructurados; sólo que en este caso su misión consistirá únicamente en crear objetos a imagen de la clase (sacarlos a escena) y pedirles que hagan cosas. Es aconsejable incluir este método especial en una clase aparte con un nombre significativo como
Test
o
Principal
, para que el alumno entienda que esa clase no representa ningún concepto (una figura, una persona) sino que simplemente alberga el método principal.

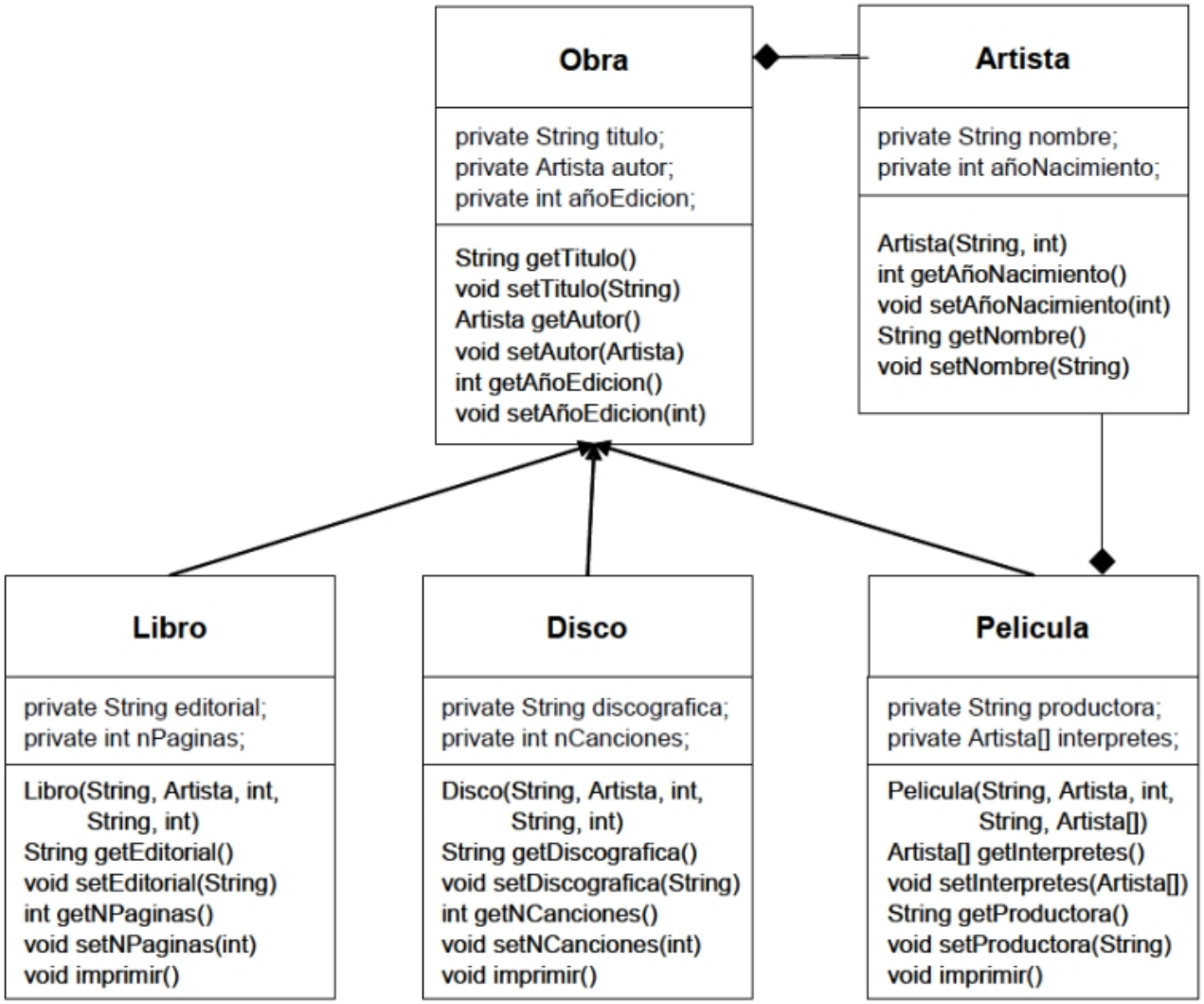
Una de las grandes ventajas de Java es que las empresas pueden repartir el trabajo de programación, encargando el desarrollo de diferentes clases a diferentes personas. La integración del trabajo es sencilla dado que las clases son *cajas negras*: sólo necesitas saber qué atributos tiene la clase Cuadrado y qué métodos posee. Pero por complicado que fuese para quien lo programó, no necesitas saber cómo resolvió el método
calcularArea
de una figura: te basta con saber que está disponible.

¿Y en qué afecta todo esto a nuestra labor docente y al entorno? Veamos algunas conclusiones interesantes:

- La fase previa a la programación en Java es muy creativa: se trata de diseñar las clases que voy a necesitar, y modelarlas según el problema que quiero resolver. ¿Qué es para mí un cuadrado? Si mi problema es trigonométrico,su atributo será el lado. Pero en otros contextos

quizá necesite conocer el material de que está hecho, o su color. Algo parecido pasa con los métodos. ¿Qué voy a querer pedirle a un cuadrado? Quizá que calcule su área, o quizá que se dibuje en la pantalla.

- Las clases están pensadas para ser reutilizadas por otras personas distintas a aquella que la programó. Para ello es fundamental documentar nuestro trabajo: el programador, de un vistazo, debe poder saber qué atributos tiene la clase y de qué métodos dispone. Existen dos herramientas básicas para esta misión: la documentación *JavaDoc* (una página web estándar en la que se resume todo el contenido de la clase, cuyo uso no será cubierto en este artículo) y los diagramas de clases, que muestran para cada clase sus atributos, sus métodos y su relación con otras clases. Dichos diagramas pueden tener este aspecto:



Por tanto, conviene habituar al alumno a comenzar su trabajo diseñando las clases sobre el papel utilizando este tipo de diseños. Será interesante que el entorno sea capaz de crear por sí mismo diagramas a partir del código introducido, para que el alumno pueda contrastar su idea original con la clase que realmente está programando. Una buena práctica consistirá en que un alumno trate de utilizar una clase programada por otro compañero, disponiendo únicamente del

diagrama de clases. Pero sigamos con las conclusiones:

- Frente a los lenguajes estructurados, el programa más sencillo que utilicemos en Java tendrá seguramente un mínimo de tres o cuatro clases, lo cuál requiere que tengamos abiertas un buen número de ventanas a la vez. El entorno de desarrollo debe estar capacitado para permitir manejar cómodamente múltiples archivos abiertos.
- Java no es un lenguaje para los impacientes. Para resolver un problema sobre trigonometría debo pasar un tiempo diseñando y programando previamente las clases correspondientes a las figuras; pero hemos dicho que una clase es como un tipo de datos, y por sí misma no hace nada, no puede ejecutarse. Será al final, cuando ya tenga todas las clases listas, cuando escriba un programa principal que creará distintos objetos a imagen y semejanza de sus correspondientes clases, y esos objetos sí que harán cosas. Para evitar el desconcierto o aburrimiento del alumno ante esta situación, algunos entornos ofrecen una posibilidad didácticamente interesante: una vez programada la clase Cuadrado, puedo crear gráficamente un objeto *miCuadrado* e interactuar con él, probando así sus métodos y observando sus atributos, y todo esto a golpe de ratón, sin necesidad de escribir ningún programa principal.

5. Escogiendo un entorno de desarrollo

Ya estamos listos para empezar: con el JDK instalado, y teniendo en mente los aspectos que debemos destacar en la enseñanza de Java, pasaremos a analizar tres entornos de desarrollo en sendas entregas de este monográfico. Para cada uno de ellos encontrarás las siguientes secciones:

- **Puesta a punto:** se trata de una guía rápida que te permitirá ponerte a trabajar en pocos minutos y compilar tu primer programa de prueba.
- **Sacándole partido:** incide en las funcionalidades más interesantes que te ofrece la aplicación.
- **Valoración:** repaso esquemático por las ventajas e inconvenientes que presenta el entorno frente a las otras opciones, siempre desde el punto de vista de la docencia.

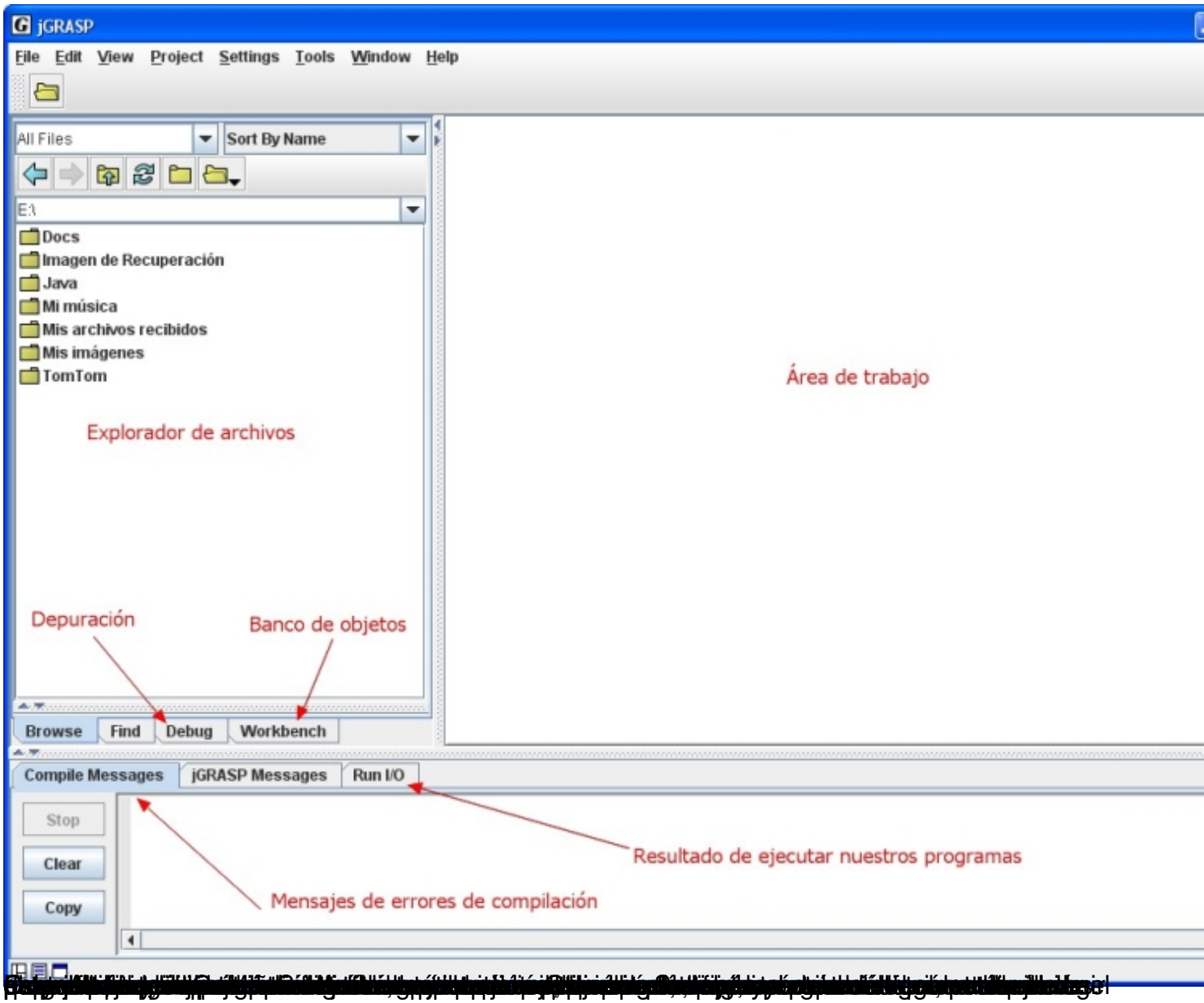
Un último apunte: aunque los tres entornos estudiados tienen versiones para Linux, las explicaciones se han hecho sobre la versión de Windows, por ser el sistema operativo más frecuentemente encontrado a día de hoy en los centros educativos.

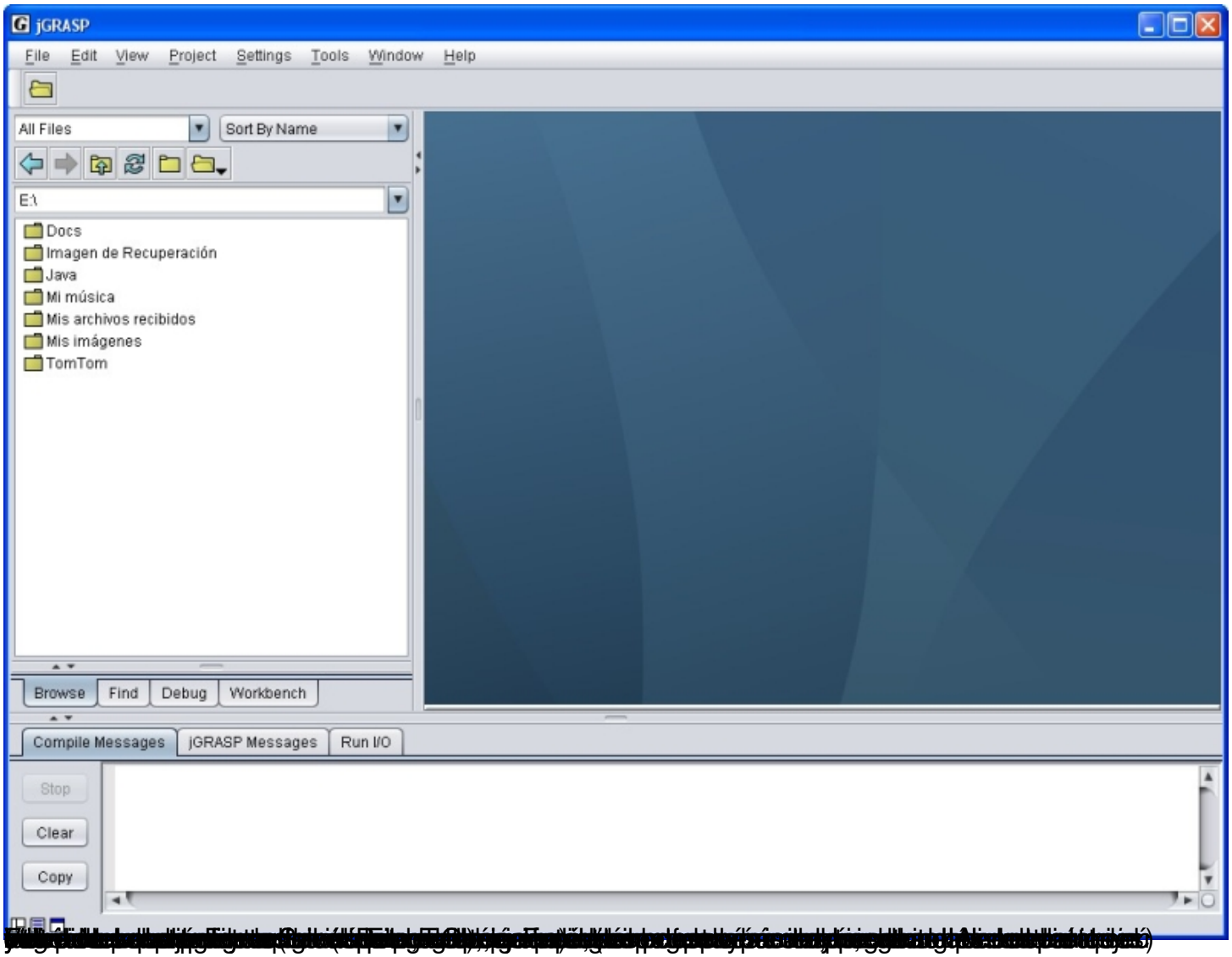
Entorno de desarrollo jGrasp

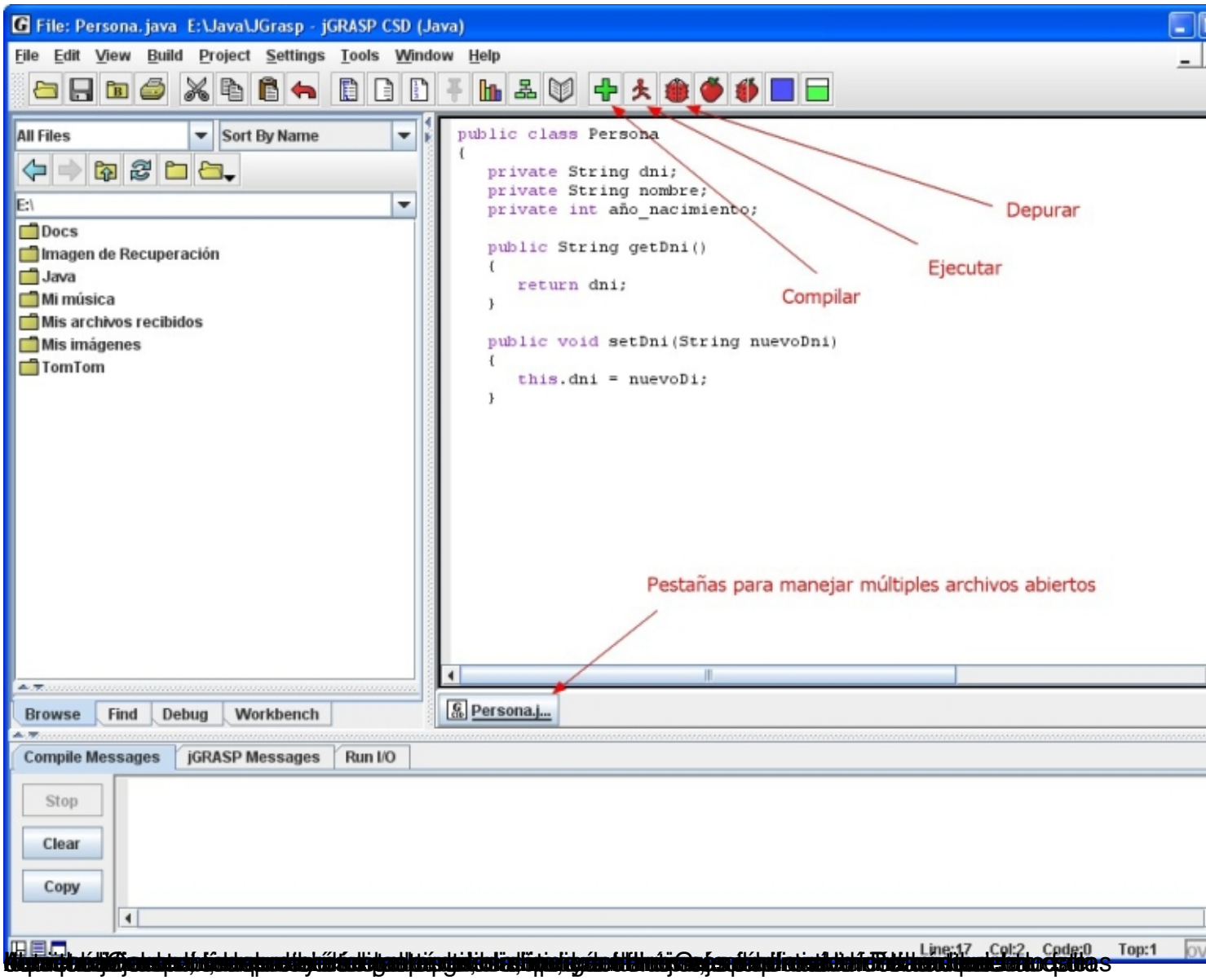
jGrasp es un entorno de programación ligero surgido en el ámbito universitario, concretamente en la universidad de Auburn. Te permitirá trabajar con distintos lenguajes de programación, aunque lógicamente en el contexto de este monográfico nos centraremos en sus posibilidades de cara al lenguaje Java.

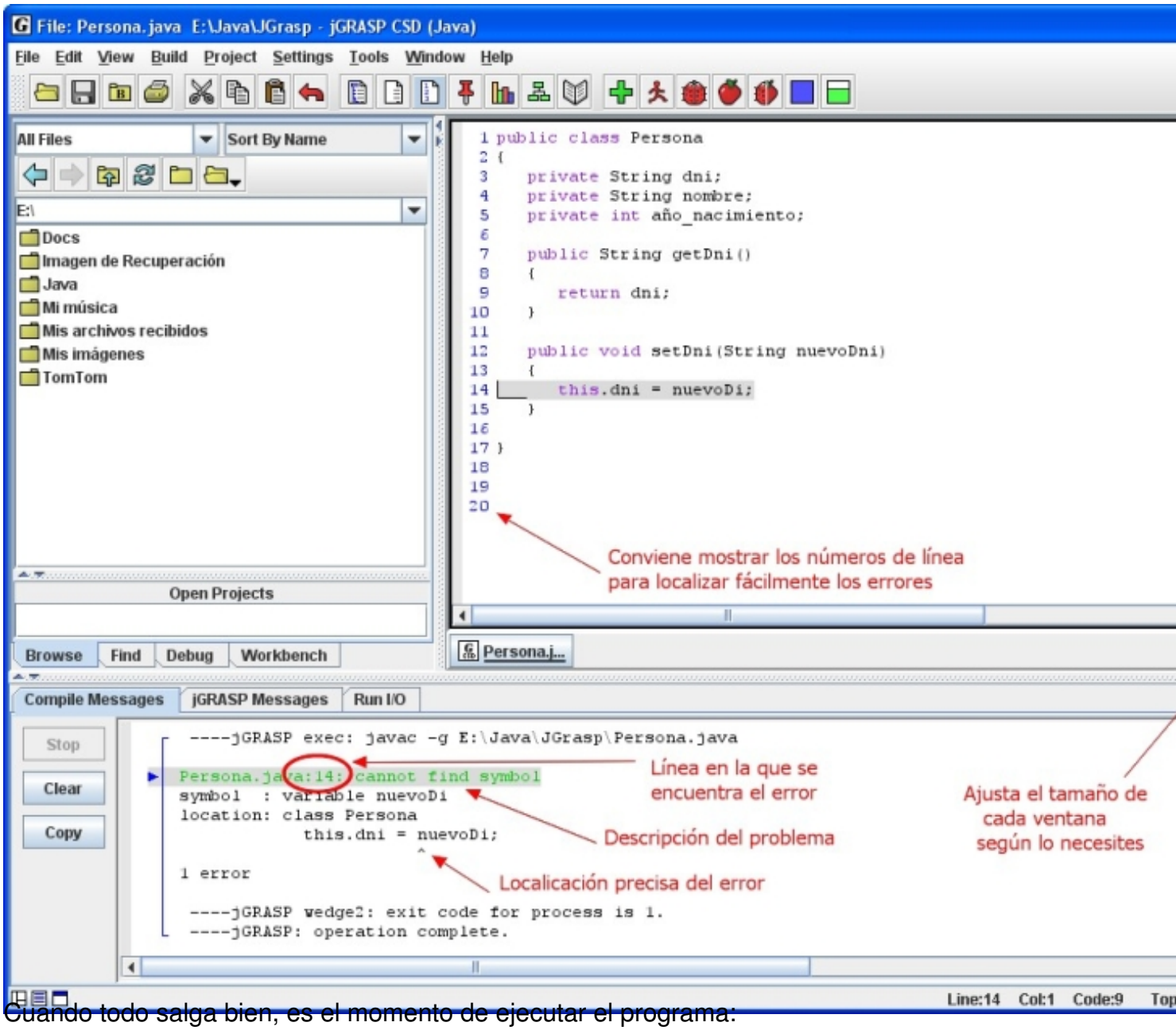
1. Puesta a punto

Lo primero que haremos será acudir a su página web y descargar la última versión disponible. La instalación es sencilla y no se nos preguntará nada durante el proceso; por otra parte, si instalaste el JDK previamente como hemos sugerido, jGrasp lo detectará automáticamente y no tendrás que configurarlo. En cualquier caso y si tuvieras varias versiones de JDK conviviendo en el equipo, siempre puedes acceder a la opción Settings > Compiler Settings para cerciorarte de qué JDK estás utilizando. Es el momento de arrancar el programa:

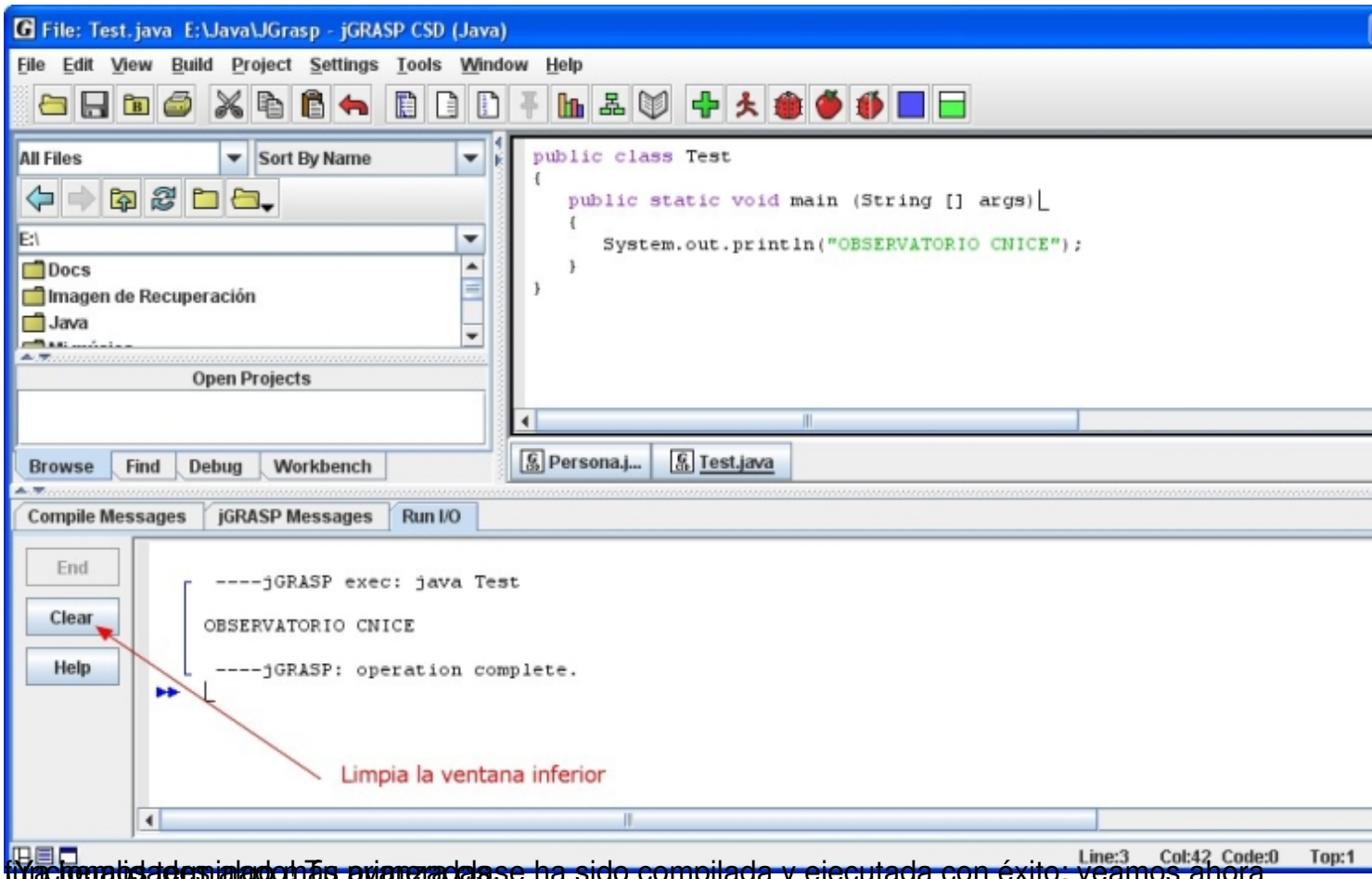

jGRASP
File Edit View Project Settings Tools Window Help
All Files
Sort By Name
E:\
Docs
Imagen de Recuperación
Java
Mi música
Mis archivos recibidos
Mis imágenes
TomTom
Explorador de archivos
Área de trabajo
Depuración
Banco de objetos
Browse
Find
Debug
Workbench
Compile Messages
jGRASP Messages
Run I/O
Stop
Clear
Copy
Resultado de ejecutar nuestros programas
Mensajes de errores de compilación


jGRASP
File Edit View Project Settings Tools Window Help
All Files
Sort By Name
E:\
Docs
Imagen de Recuperación
Java
Mi música
Mis archivos recibidos
Mis imágenes
TomTom
Browse Find Debug Workbench
Compile Messages
jGRASP Messages
Run I/O
Stop
Clear
Copy


File: Persona.java E:\Java\JGrasp - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
All Files
Sort By Name
E:\
Docs
Imagen de Recuperación
Java
Mi música
Mis archivos recibidos
Mis imágenes
TomTom
Browse Find Debug Workbench
public class Persona
{
private String dni;
private String nombre;
private int año_nacimiento;
public String getDni()
{
return dni;
}
public void setDni(String nuevoDni)
{
this.dni = nuevoDi;
}
Depurar
Ejecutar
Compilar
Pestañas para manejar múltiples archivos abiertos
Persona.j...
Compile Messages
jGRASP Messages
Run I/O
Stop
Clear
Copy
Line:17 Col:2 Cpde:0 Top:1



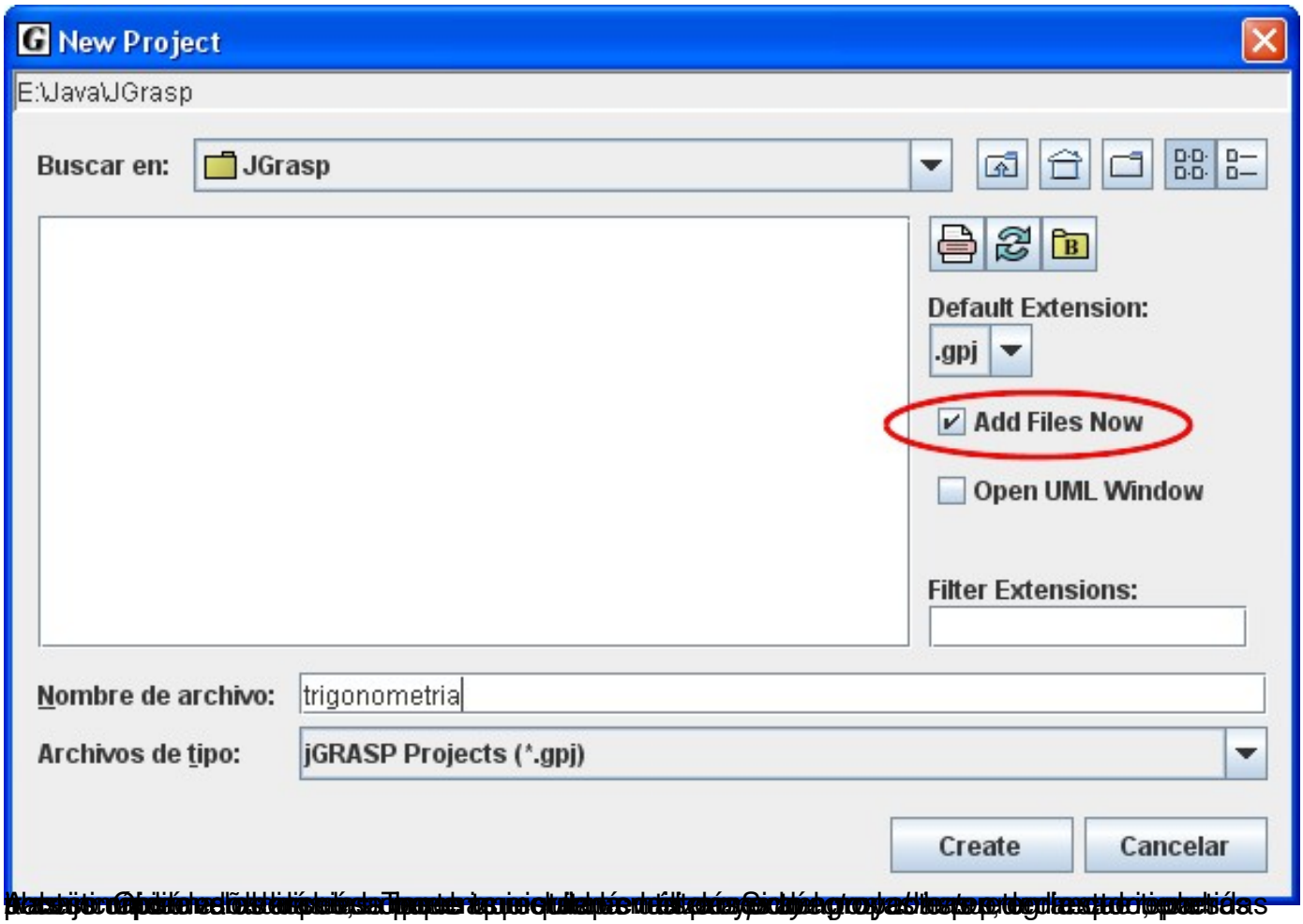
Cuando todo salga bien, es el momento de ejecutar el programa:

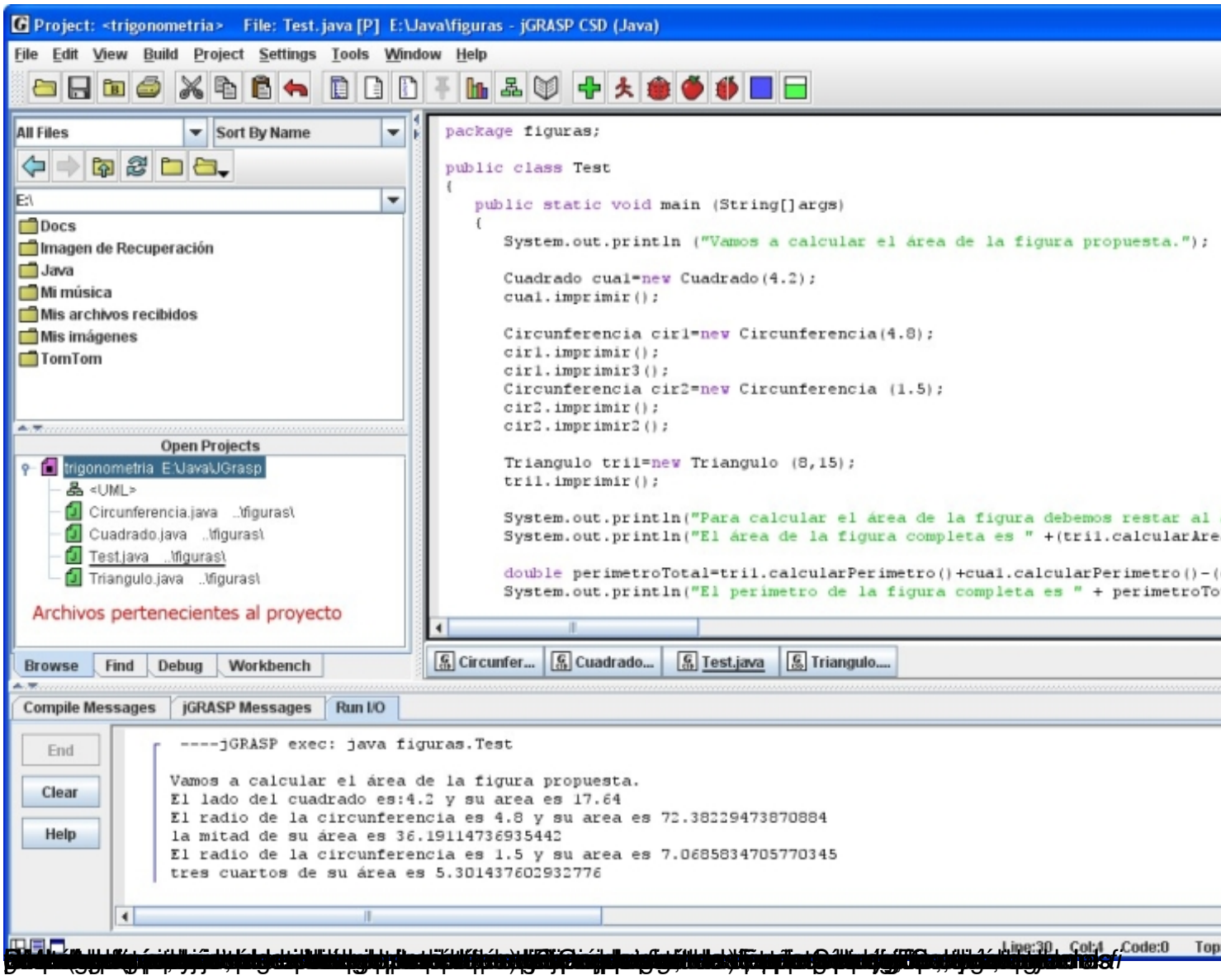


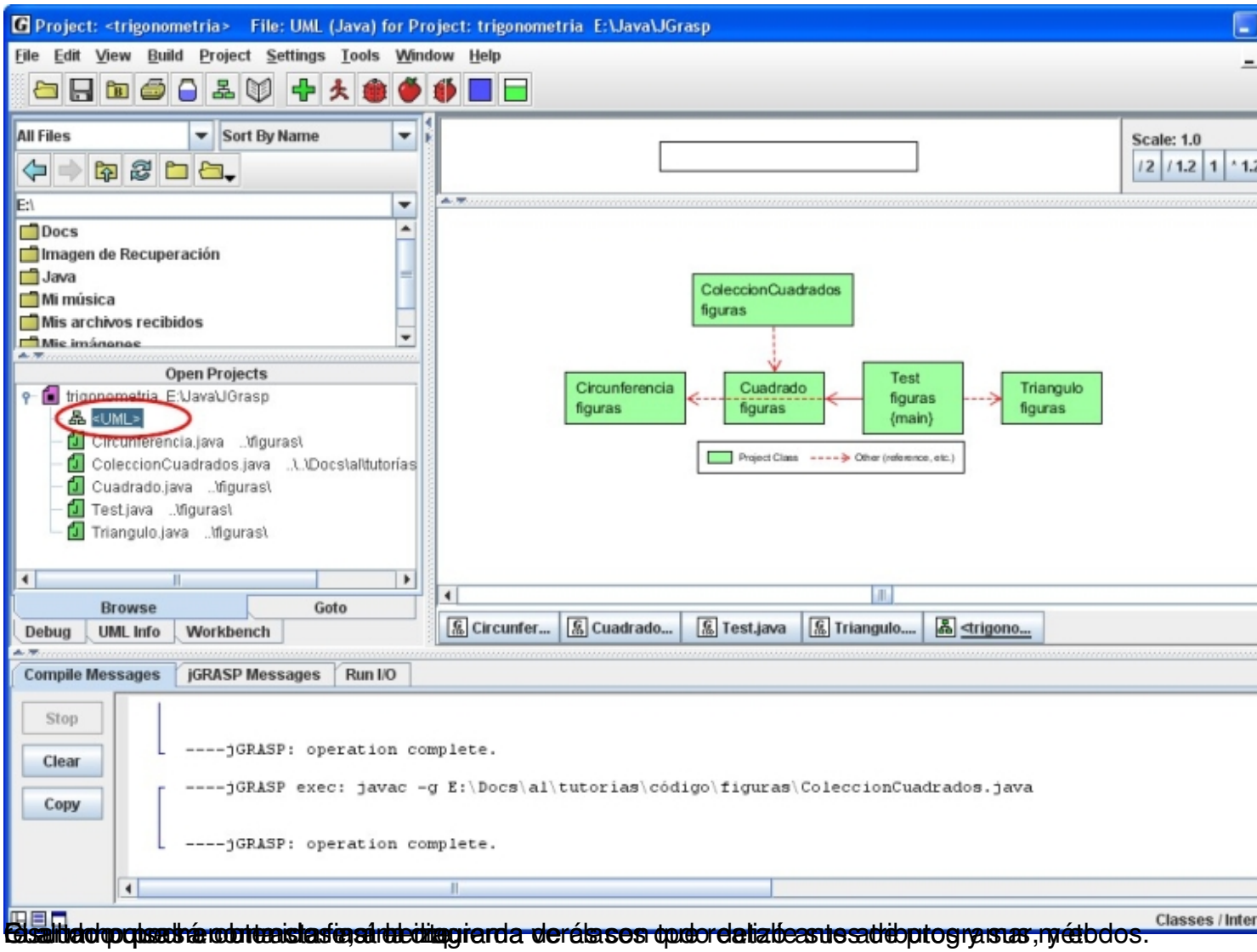
¡Fíjate lo sencillo que ha sido! Ya tenemos nuestra primera clase compilada y ejecutada con éxito; veamos ahora algunas características más avanzadas.

2. Sacándole partido

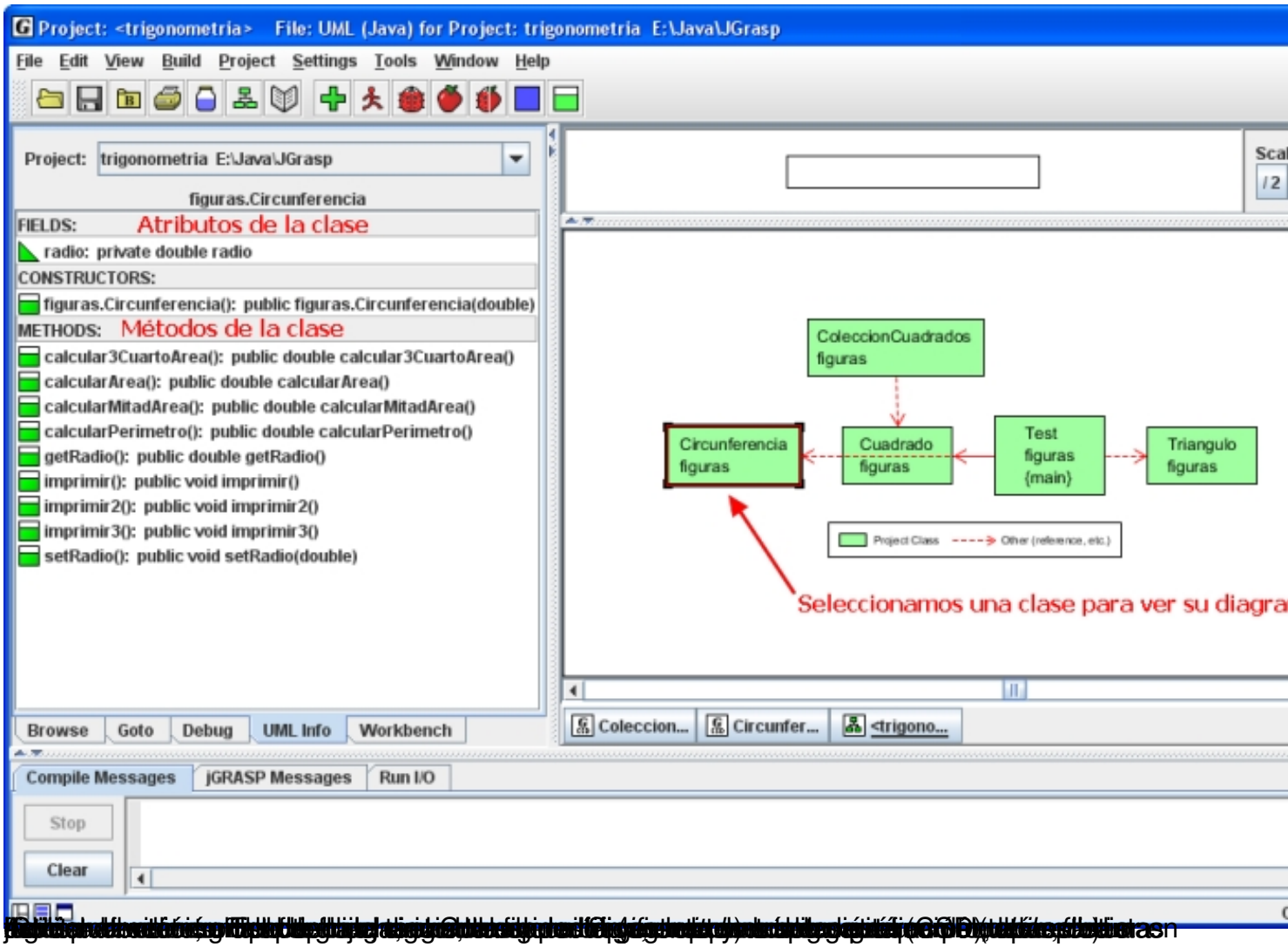
Hemos dicho que los programas en Java están normalmente constituidos por varias clases. Por ello, todos los entornos de desarrollo permiten crear proyectos para tener rápido acceso a todas las clases que van a participar en nuestro programa. jGrasp no es una excepción, así que vamos a ver cómo podemos dar forma a un sencillo proyecto Java. Para empezar, accede al menú Project > New > New Standard Project:

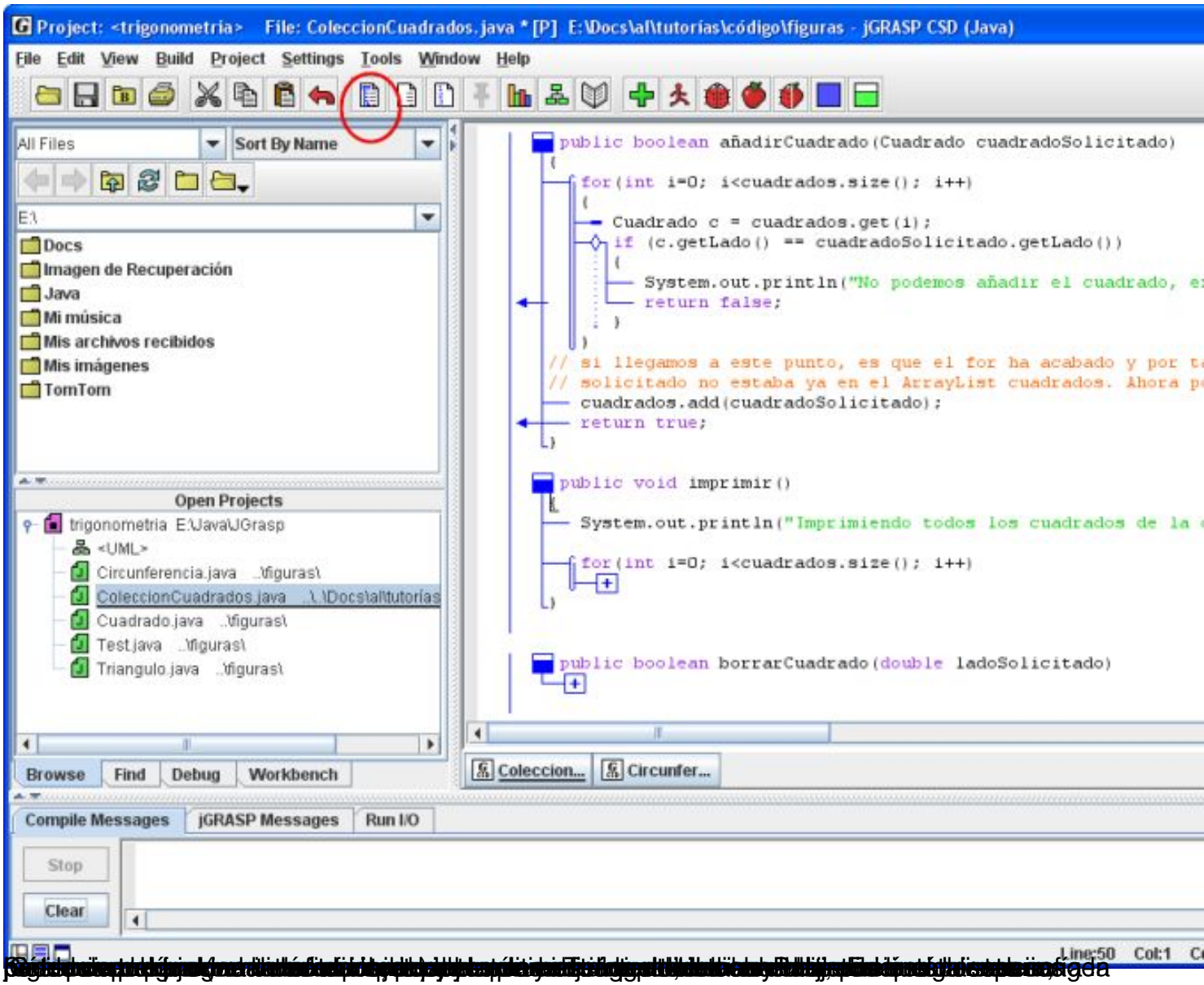

New Project
E:\Java\JGrasp
Buscar en:
JGrasp
Default Extension:
.gpj
Add Files Now
Open UML Window
Filter Extensions:
Nombre de archivo:
trigonometria
Archivos de tipo:
jGRASP Projects (*.gpj)
Create
Cancelar

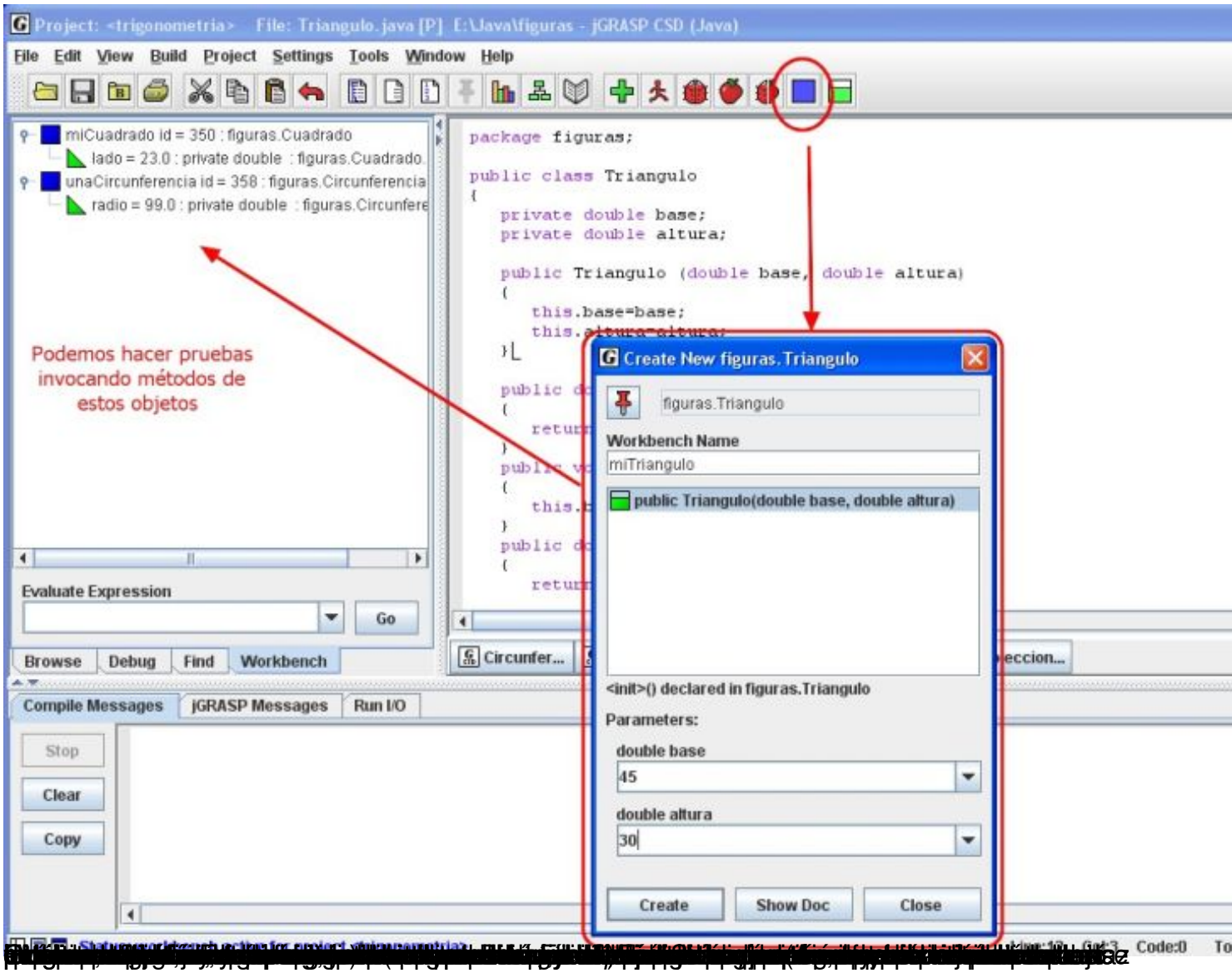

Project: <trigonometria> File: Test.java [P] E:\Java\figuras - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
All Files
Sort By Name
E:\
Docs
Imagen de Recuperación
Java
Mi música
Mis archivos recibidos
Mis imágenes
TomTom
Open Projects
trigonometria E:\Java\JGrasp
<UML>
Circunferencia.java ..\figuras\
Cuadrado.java ..\figuras\
Test.java ..\figuras\
Triangulo.java ..\figuras\
Archivos pertenecientes al proyecto
Browse Find Debug Workbench
package figuras;
public class Test
{
public static void main (String[]args)
{
System.out.println ("Vamos a calcular el área de la figura propuesta.");
Cuadrado cua1=new Cuadrado(4.2);
cua1.imprimir();
Circunferencia cir1=new Circunferencia(4.8);
cir1.imprimir();
cir1.imprimir3();
Circunferencia cir2=new Circunferencia (1.5);
cir2.imprimir();
cir2.imprimir2();
Triangulo tri1=new Triangulo (8,15);
tri1.imprimir();
System.out.println("Para calcular el área de la figura debemos restar al
System.out.println("El área de la figura completa es " +(tri1.calcularAre
double perimetroTotal=tri1.calcularPerimetro()+cua1.calcularPerimetro()-(
System.out.println("El perimetro de la figura completa es " + perimetroTo
Circunfer... Cuadrado... Test.java Triangulo....
Compile Messages jGRASP Messages Run I/O
End
Clear
Help
----jGRASP exec: java figuras.Test
Vamos a calcular el área de la figura propuesta.
El lado del cuadrado es:4.2 y su area es 17.64
El radio de la circunferencia es 4.8 y su area es 72.38229473870884
la mitad de su área es 36.19114736935442
El radio de la circunferencia es 1.5 y su area es 7.0685834705770345
tres cuartos de su área es 5.301437602932776
Line:30 Col:1 Code:0 Top

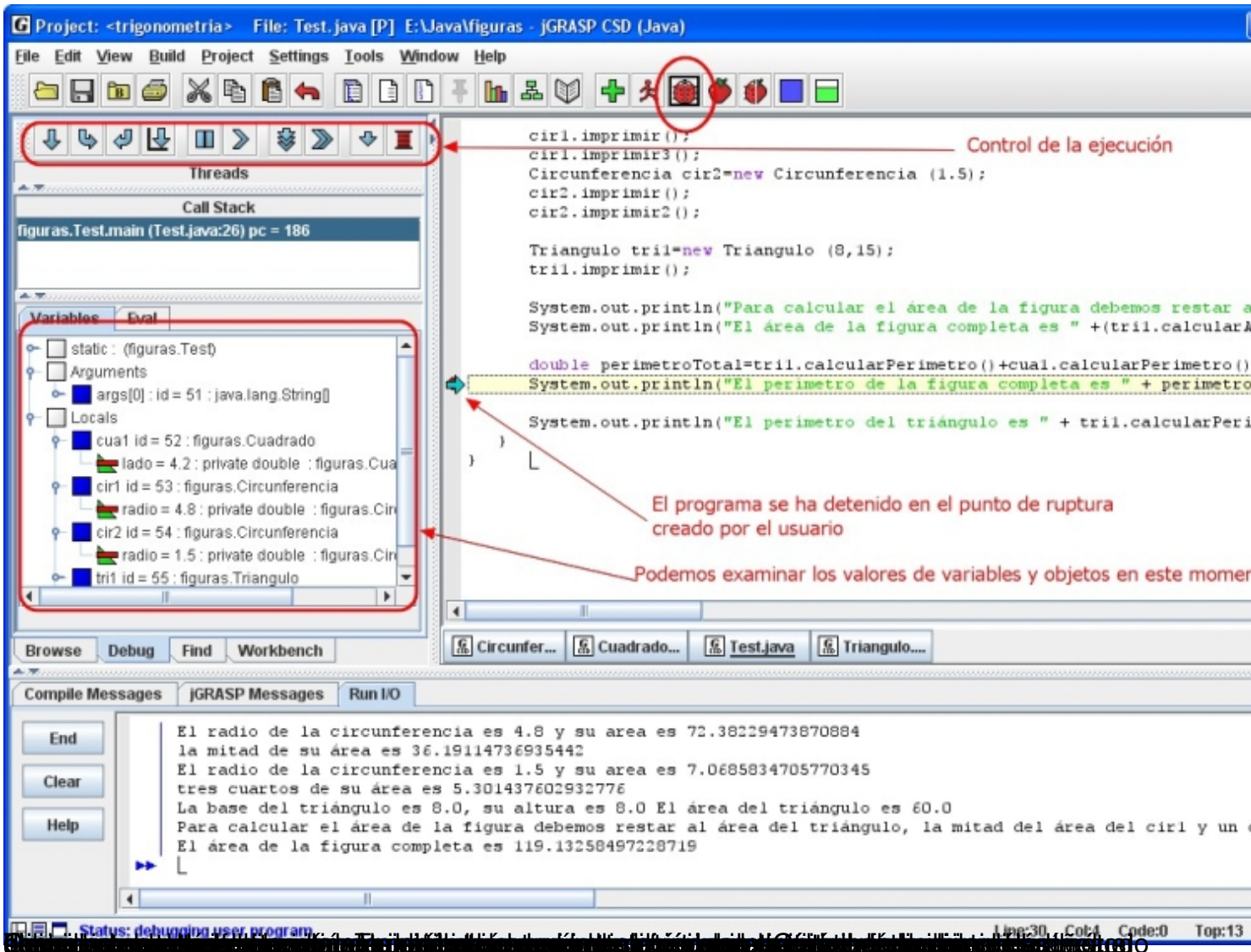


En la parte superior izquierda podemos ver el explorador de clases, donde podemos ver todos los ficheros que componen nuestro programa, métodos.


Project: <trigonometria> File: UML (Java) for Project: trigonometria E:\Java\JGrasp
File Edit View Build Project Settings Tools Window Help
Project: trigonometria E:\Java\JGrasp
figuras.Circunferencia
FIELDS:
Atributos de la clase
radio: private double radio
CONSTRUCTORS:
figuras.Circunferencia(): public figuras.Circunferencia(double)
METHODS:
Métodos de la clase
calcular3CuartoArea(): public double calcular3CuartoArea()
calcularArea(): public double calcularArea()
calcularMitadArea(): public double calcularMitadArea()
calcularPerimetro(): public double calcularPerimetro()
getRadio(): public double getRadio()
imprimir(): public void imprimir()
imprimir2(): public void imprimir2()
imprimir3(): public void imprimir3()
setRadio(): public void setRadio(double)
Browse Goto Debug UML Info Workbench
ColeccionCuadrados figuras
Circunferencia figuras
Cuadrado figuras
Test figuras (main)
Triangulo figuras
Project Class
Other (reference, etc.)
Seleccionamos una clase para ver su diagra
Coleccion...
Circunfer...
<trigono...
Compile Messages
jGRASP Messages
Run I/O
Stop
Clear


Project: <trigonometria> File: ColeccionCuadrados.java * [P] E:\Docs\al\tutorias\código\figuras - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
All Files
Sort By Name
E:\
Docs
Imagen de Recuperación
Java
Mi música
Mis archivos recibidos
Mis imágenes
TomTom
Open Projects
trigonometria E:\Java\JGrasp
<UML>
Circunferencia.java ..\figuras\
ColeccionCuadrados.java ..\Docs\al\tutorias
Cuadrado.java ..\figuras\
Test.java ..\figuras\
Triangulo.java ..\figuras\
Browse
Find
Debug
Workbench
public boolean añadirCuadrado(Cuadrado cuadradoSolicitado)
for(int i=0; i<cuadrados.size(); i++)
Cuadrado c = cuadrados.get(i);
if (c.getLado() == cuadradoSolicitado.getLado())
System.out.println("No podemos añadir el cuadrado, e
return false;
// si llegamos a este punto, es que el for ha acabado y por t
// solicitado no estaba ya en el ArrayList cuadrados. Ahora p
cuadrados.add(cuadradoSolicitado);
return true;
public void imprimir()
System.out.println("Imprimiendo todos los cuadrados de la
for(int i=0; i<cuadrados.size(); i++)
public boolean borrarCuadrado(double ladoSolicitado)
Coleccion...
Circunfer...
Compile Messages
jGRASP Messages
Run I/O
Stop
Clear
Line:50 Col:1


Project: <trigonometria> File: Triangulo.java [P] E:\Java\figuras - jGRASP CSD (Java)
File Edit View Build Project Settings Tools Window Help
miCuadrado id = 350 : figuras.Cuadrado
lado = 23.0 : private double : figuras.Cuadrado.
unaCircunferencia id = 358 : figuras.Circunferencia
radio = 99.0 : private double : figuras.Circunfere
Podemos hacer pruebas invocando métodos de estos objetos
Evaluate Expression
Go
Browse Debug Find Workbench
package figuras;
public class Triangulo
{
private double base;
private double altura;
public Triangulo (double base, double altura)
this.base=base;
Create New figuras.Triangulo
figuras.Triangulo
Workbench Name
miTriangulo
public Triangulo(double base, double altura)
<init>() declared in figuras.Triangulo
Parameters:
double base
45
double altura
30
Create
Show Doc
Close
Compile Messages
jGRASP Messages
Run I/O
Stop
Clear
Copy



3. Valoración

A continuación se indican los aspectos que sobresalen en jGrasp, para bien o para mal, respecto al resto de entornos estudiados:

Puntos fuertes de jGrasp

- Su planteamiento y organización es similar al de IDEs como NetBeans o Eclipse, acostumbrando al alumno a entornos más profesionales
- Permite una cómoda organización cuando hay muchos archivos abiertos
- Genera automáticamente diagramas de clases UML, de gran utilidad didáctica

Puntos débiles de jGrasp

- Su aspecto no es demasiado atractivo
- La gestión de los paquetes podría ser más clara

- Carece de documentación en español

Entorno de desarrollo BlueJ

BlueJ es un sencillo entorno de programación exclusivamente diseñado para la enseñanza y el aprendizaje de Java. Se trata de un proyecto nacido en el seno de un grupo de investigación universitario integrado por miembros británicos y australianos. Por sus novedosas características, en poco tiempo BlueJ alcanzó una gran popularidad en entornos docentes.

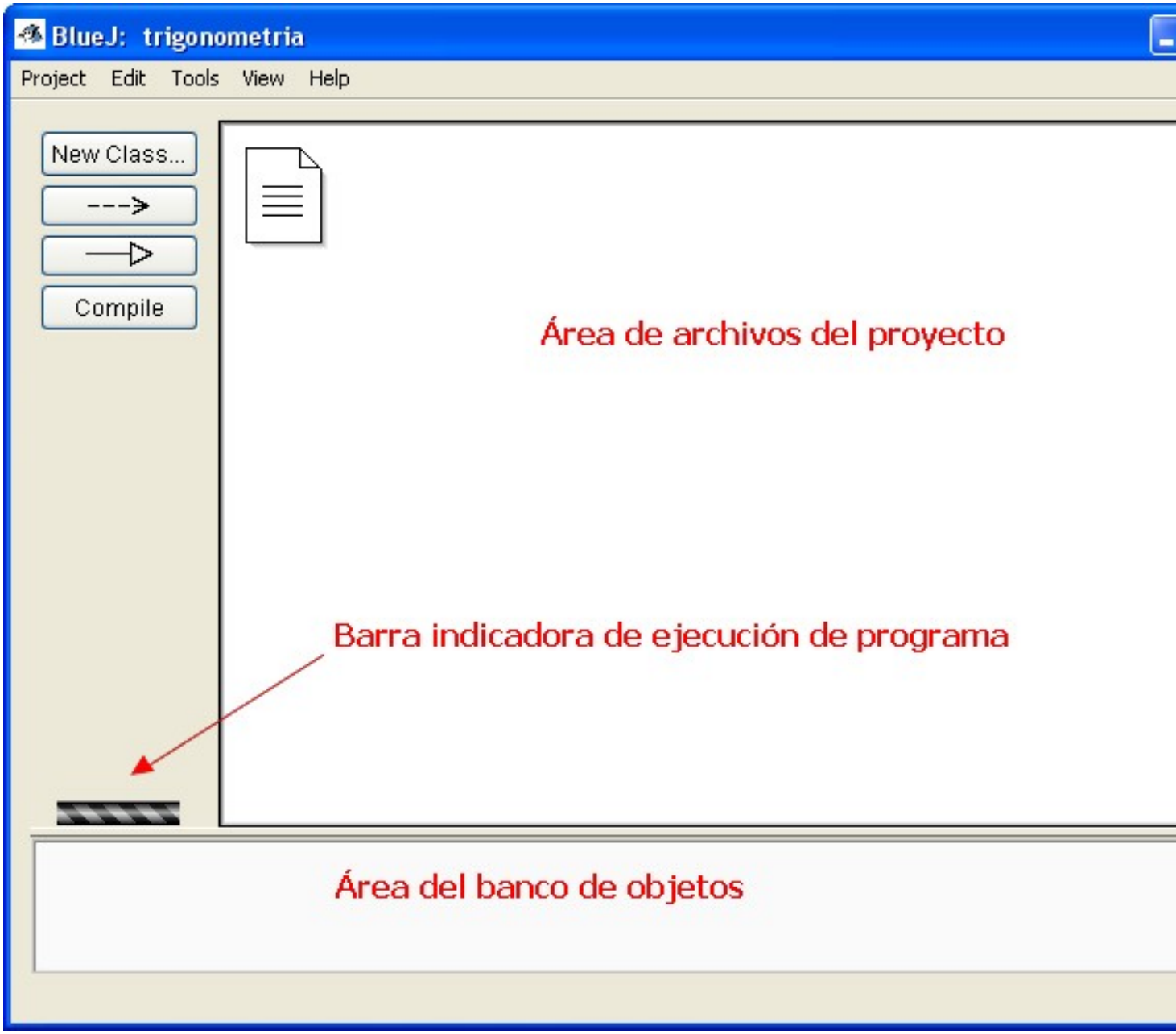
1. Puesta a punto

Lo primero que haremos será descargar el programa de su página web y proceder a instalarlo. El procedimiento es trivial y, asumiendo que ya tuvieses instalado el JDK, no tendrás que configurar absolutamente nada para empezar a trabajar; pero antes de programar nuestra primera clase, debemos hacer una aclaración previa.

La pantalla principal aparecerá en gris hasta que creemos un proyecto. En BlueJ existe una desafortunada confusión entre el término proyecto, tal y como suele entenderse en los IDEs, y el concepto de paquete, antes explicado. Así, un proyecto BlueJ es igual que un paquete, pero

en aras de simplificar se eliminan las líneas de código *package,* obligatorias según la norma de Java. Si trabajar con paquetes es prioritario para ti, BlueJ no es definitivamente la mejor opción. ¿Significa esto que no podemos utilizar en BlueJ paquetes de clases programados en otros entornos? Existe una forma de incluirlos en nuestro proyecto, Project > Open non BlueJ, pero sin duda no resulta nada intuitivo.

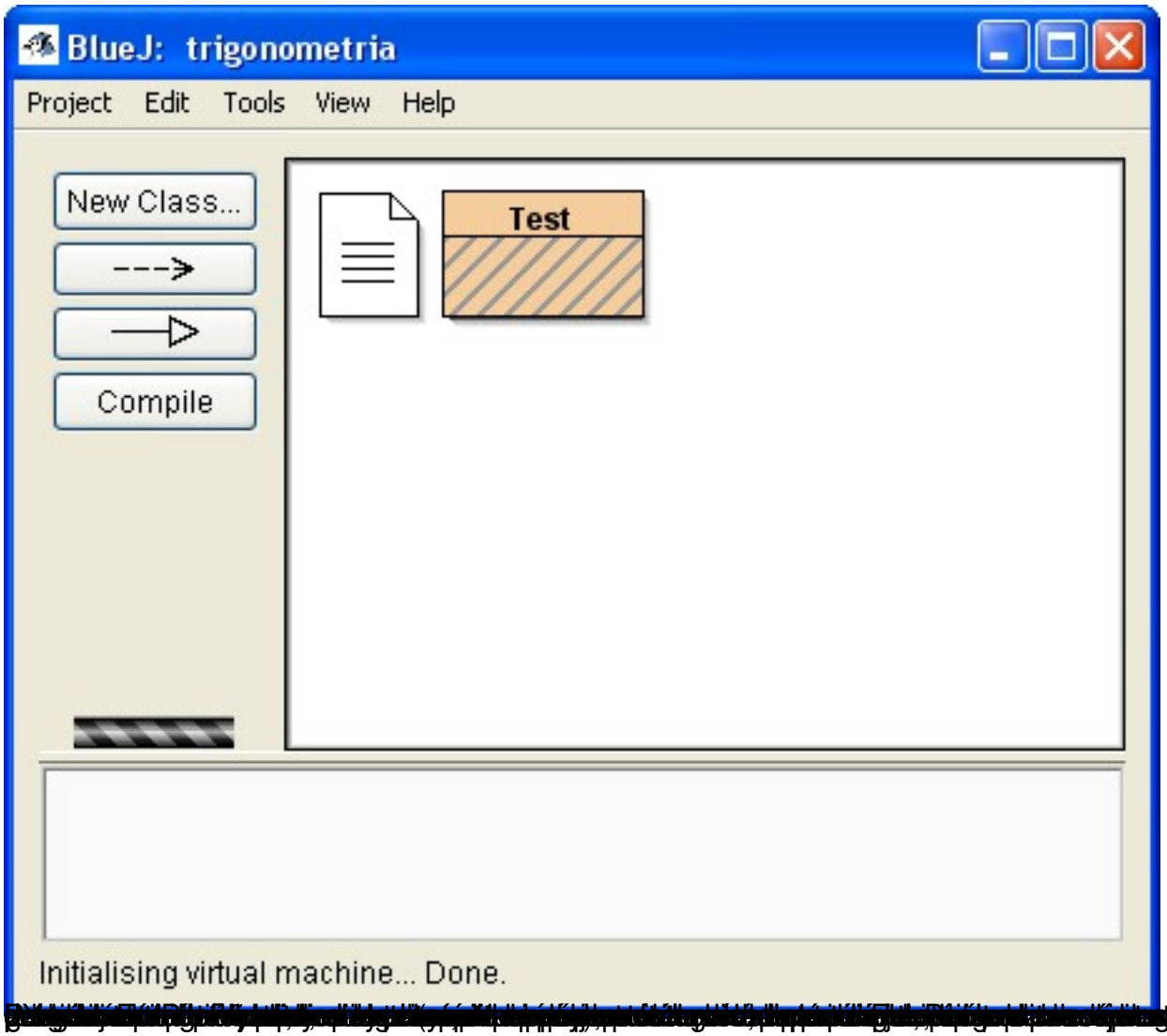
No nos quedemos con este defecto, sin duda uno de los pocos que presenta el entorno: creemos un nuevo proyecto con Project > New Project y ya podremos empezar a trabajar. La pantalla se mostrará así:

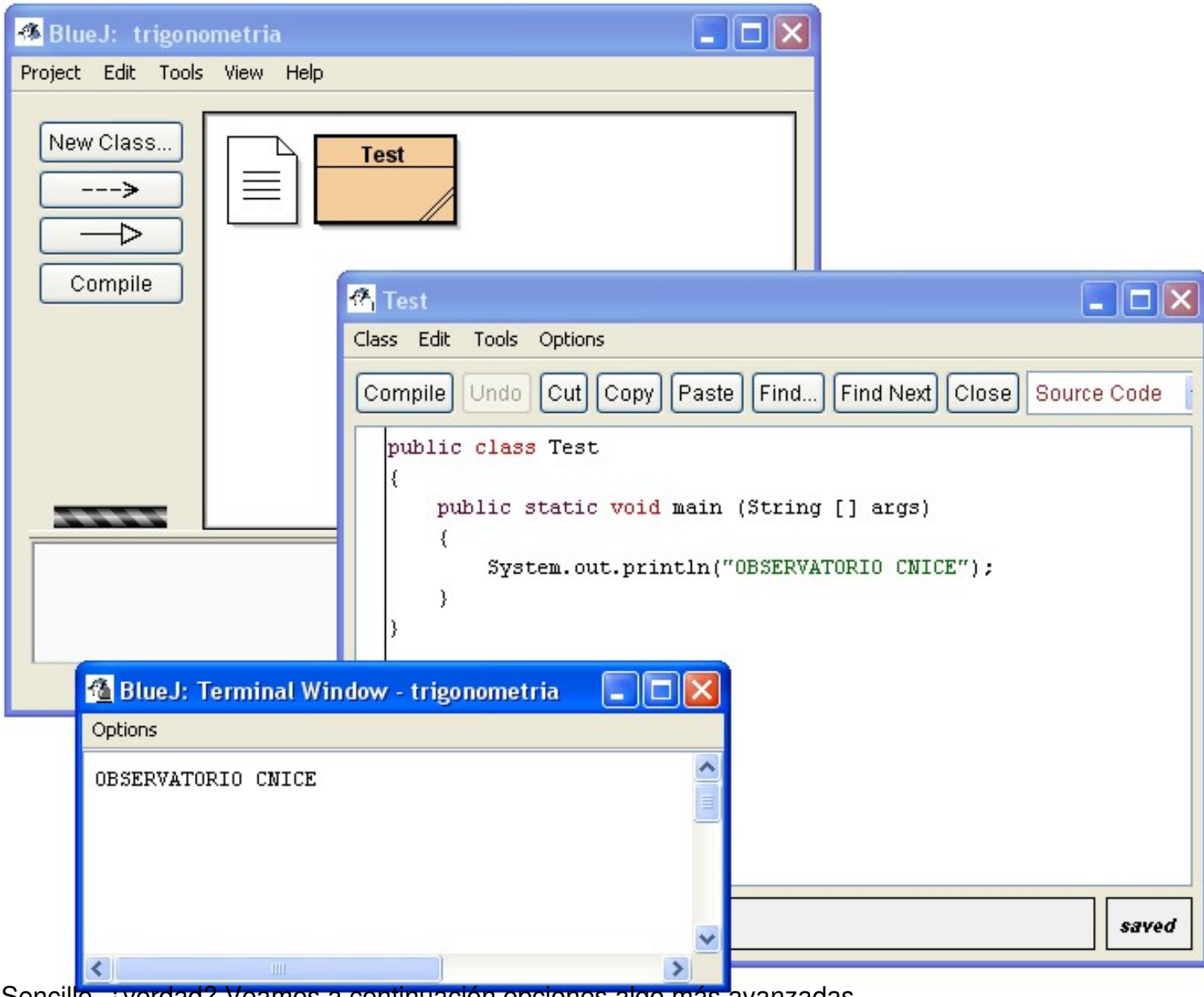


Como ves, la pantalla principal es tremendamente sencilla. A la izquierda tenemos los botones de nueva clase y compilar; a la derecha la ventana principal, en la que veremos el esquema de clases. Abajo está el banco de objetos, similar al de jGrasp, que se estudiará más adelante. Por último, una barra gris cambiará de color y se pondrá en movimiento cuando haya un programa en ejecución.

Existen dos formas de añadir una clase al proyecto: la primera es crearla nosotros mismos, mediante el botón New Class. La segunda es añadir una clase existente, mediante el menú Edit > Add class from file. Ten en cuenta que esta clase se copiará al directorio del proyecto actual, mientras que en jGrasp esto no ocurría: simplemente se utilizaba la clase desde su ubicación original. Por otra parte no hace falta que te preocupes de guardar las clases, ya que BlueJ lo hará por ti cada vez que compiles o cuando cierres el programa.

El entorno incluye un sencillo asistente para crear clases, cuya misión es proporcionarte un modelo o esqueleto de código sobre el que trabajar. Este borrador será distinto si queremos hacer una clase estándar o, por ejemplo, un applet (aplicación gráfica de ventana). Sin embargo, y ya que la misión es aprender, quizá sea mejor borrar la propuesta y comenzar a trabajar con el editor en blanco.

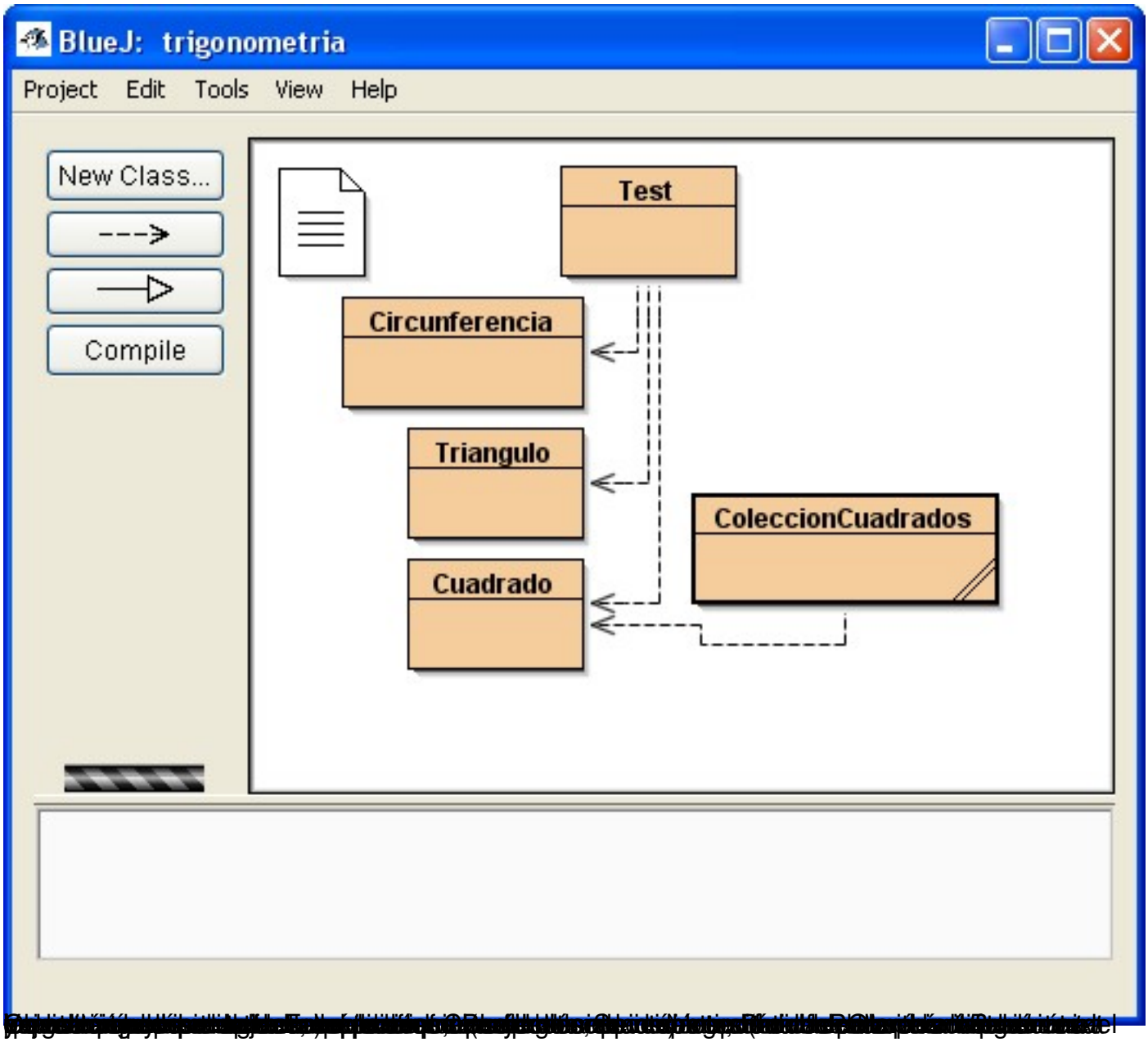

BlueJ: trigonometria
Project Edit Tools View Help
New Class...
Compile
Test
Initialising virtual machine... Done.

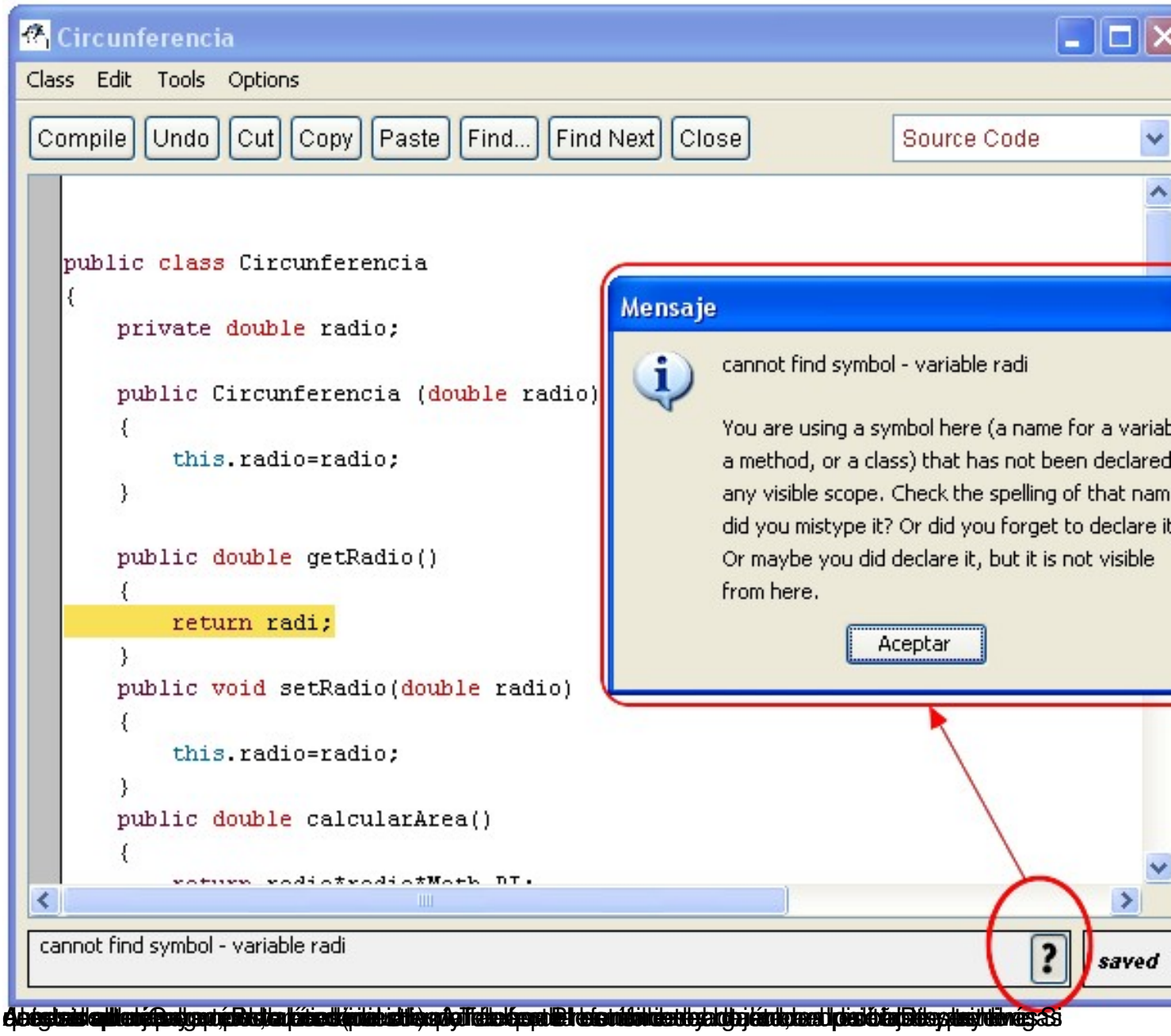


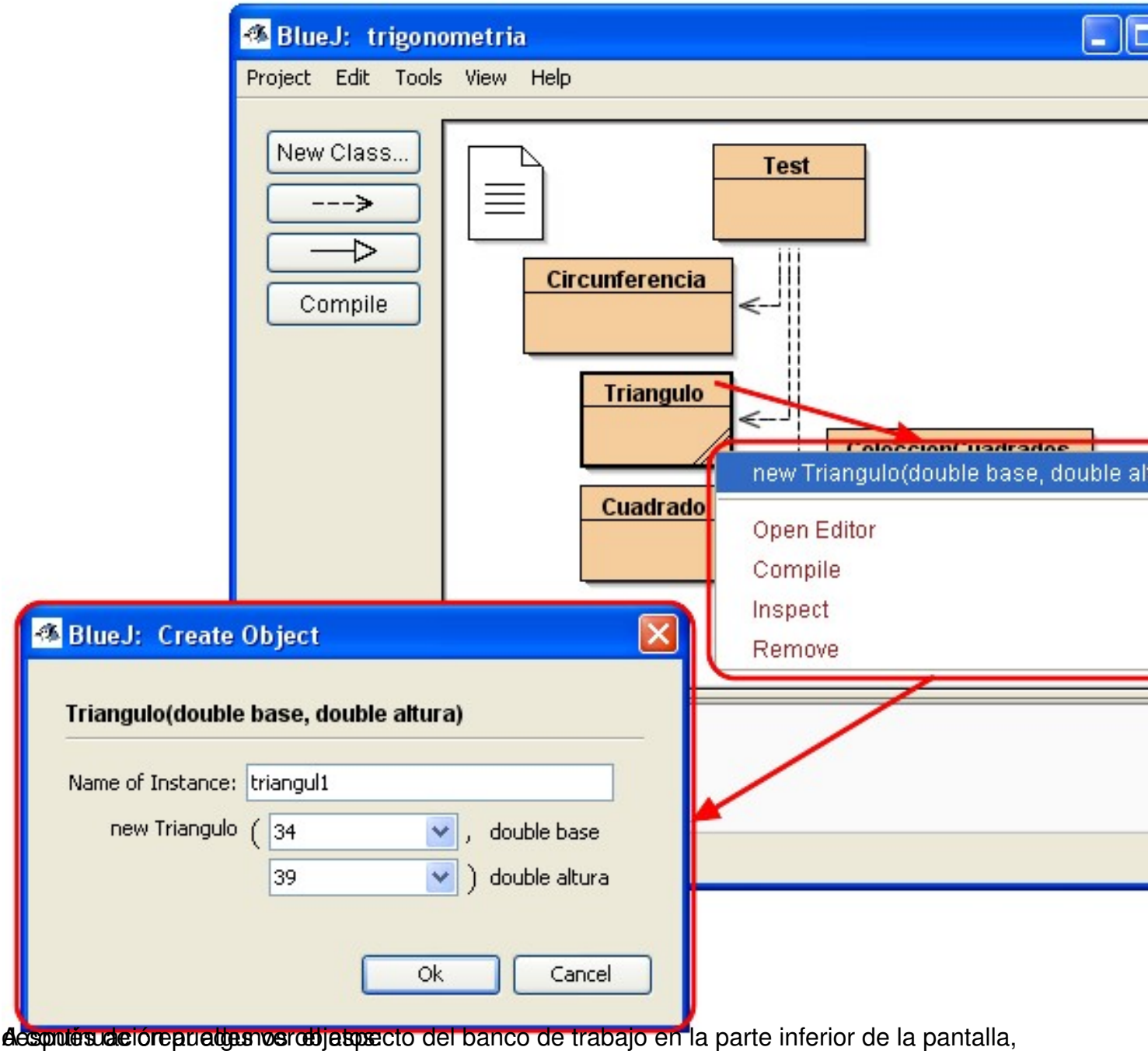
Sencillo, ¿verdad? Veamos a continuación opciones algo más avanzadas.

2. Sacándole partido

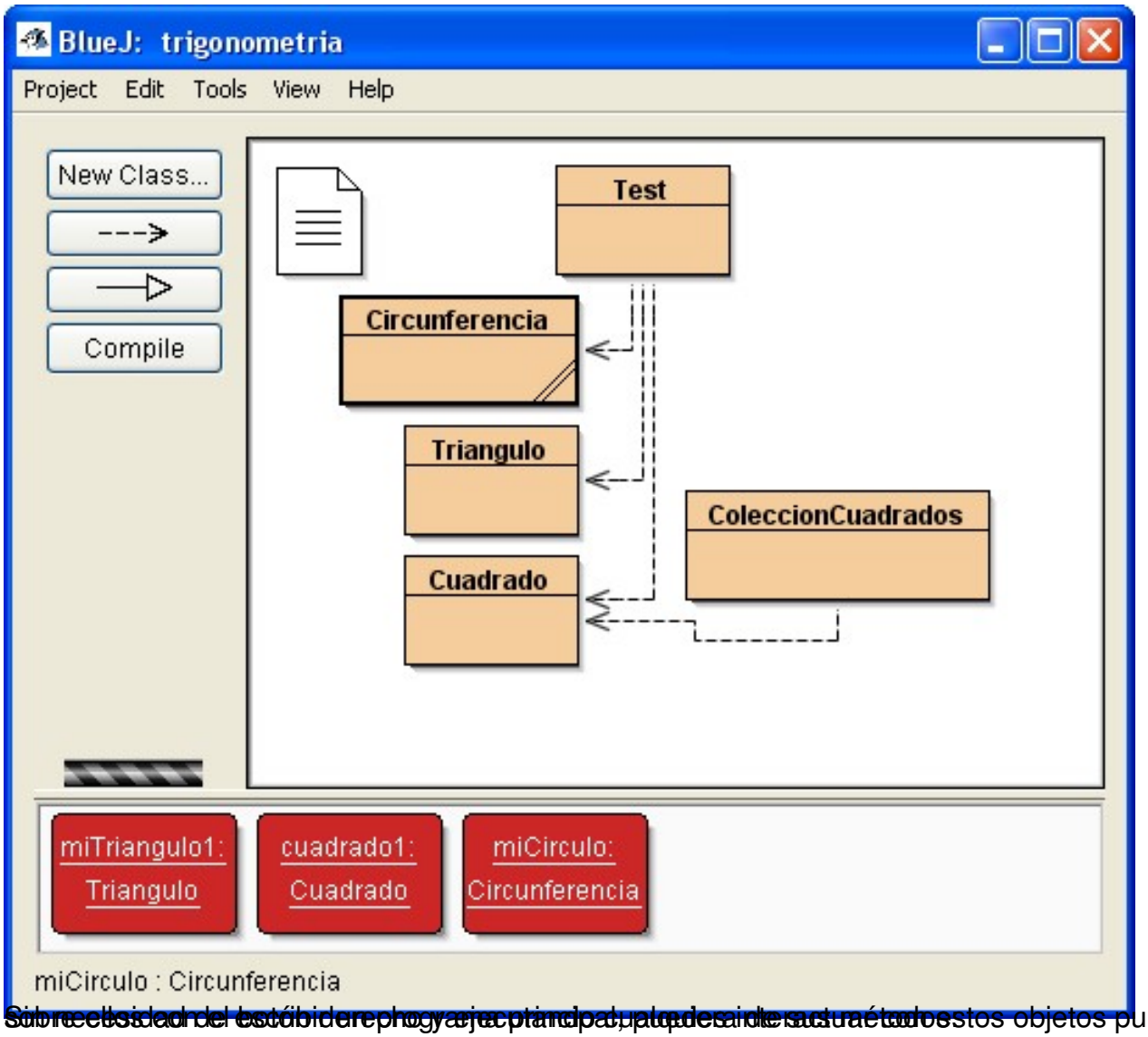
Retomemos un proyecto completo, que involucre distintas clases:


BlueJ: trigonometria
Project
Edit
Tools
View
Help
New Class...
Compile
Test
Circunferencia
Triangulo
ColeccionCuadrados
Cuadrado

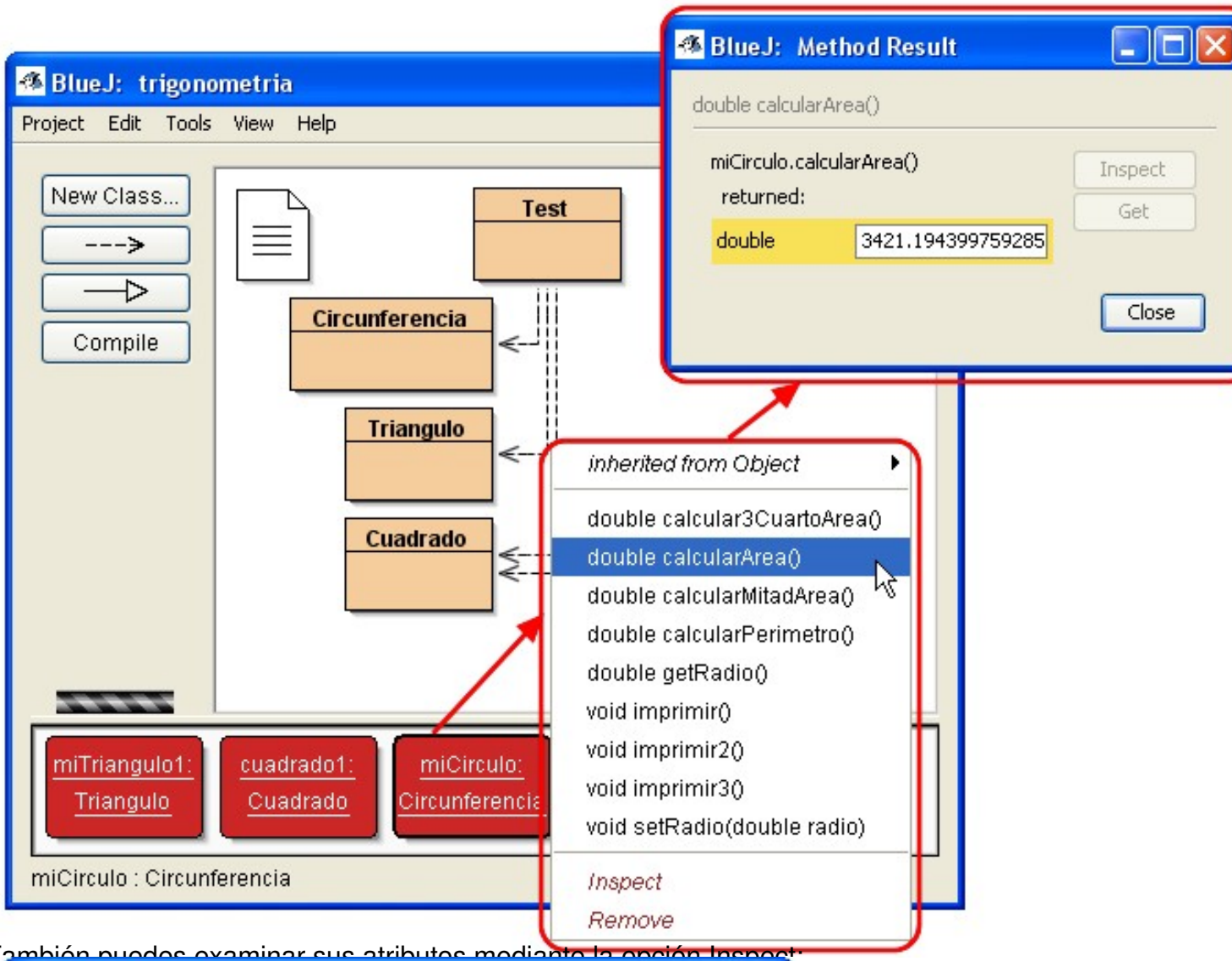

Circunferencia
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
public class Circunferencia
{
private double radio;
public Circunferencia (double radio)
{
this.radio=radio;
}
public double getRadio()
{
return radi;
}
public void setRadio(double radio)
{
this.radio=radio;
}
public double calcularArea()
{
Mensaje
cannot find symbol - variable radi
You are using a symbol here (a name for a variab
a method, or a class) that has not been declared
any visible scope. Check the spelling of that nam
did you mistype it? Or did you forget to declare it
Or maybe you did declare it, but it is not visible
from here.
Aceptar
cannot find symbol - variable radi
?
saved



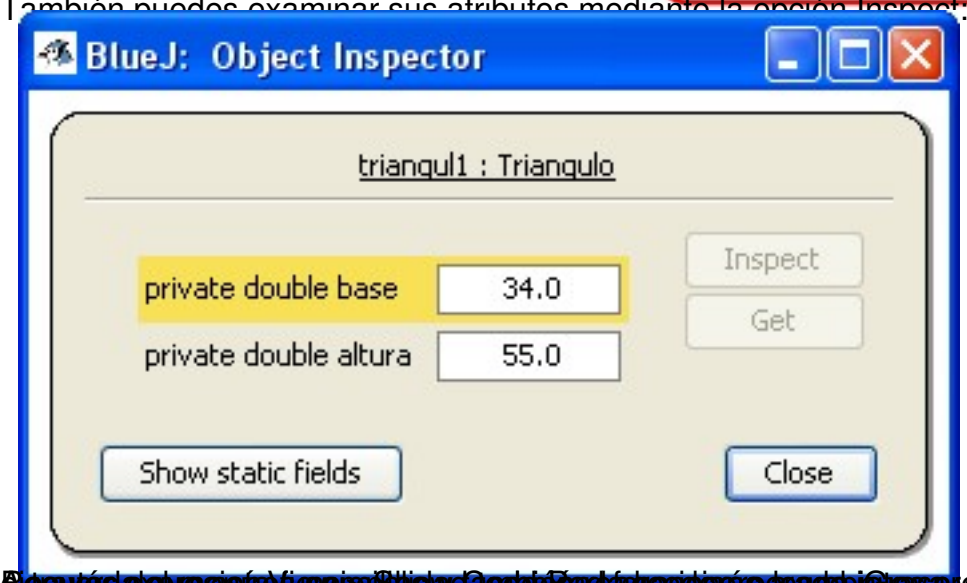
Acontinuación creamos un objeto del banco de trabajo en la parte inferior de la pantalla,



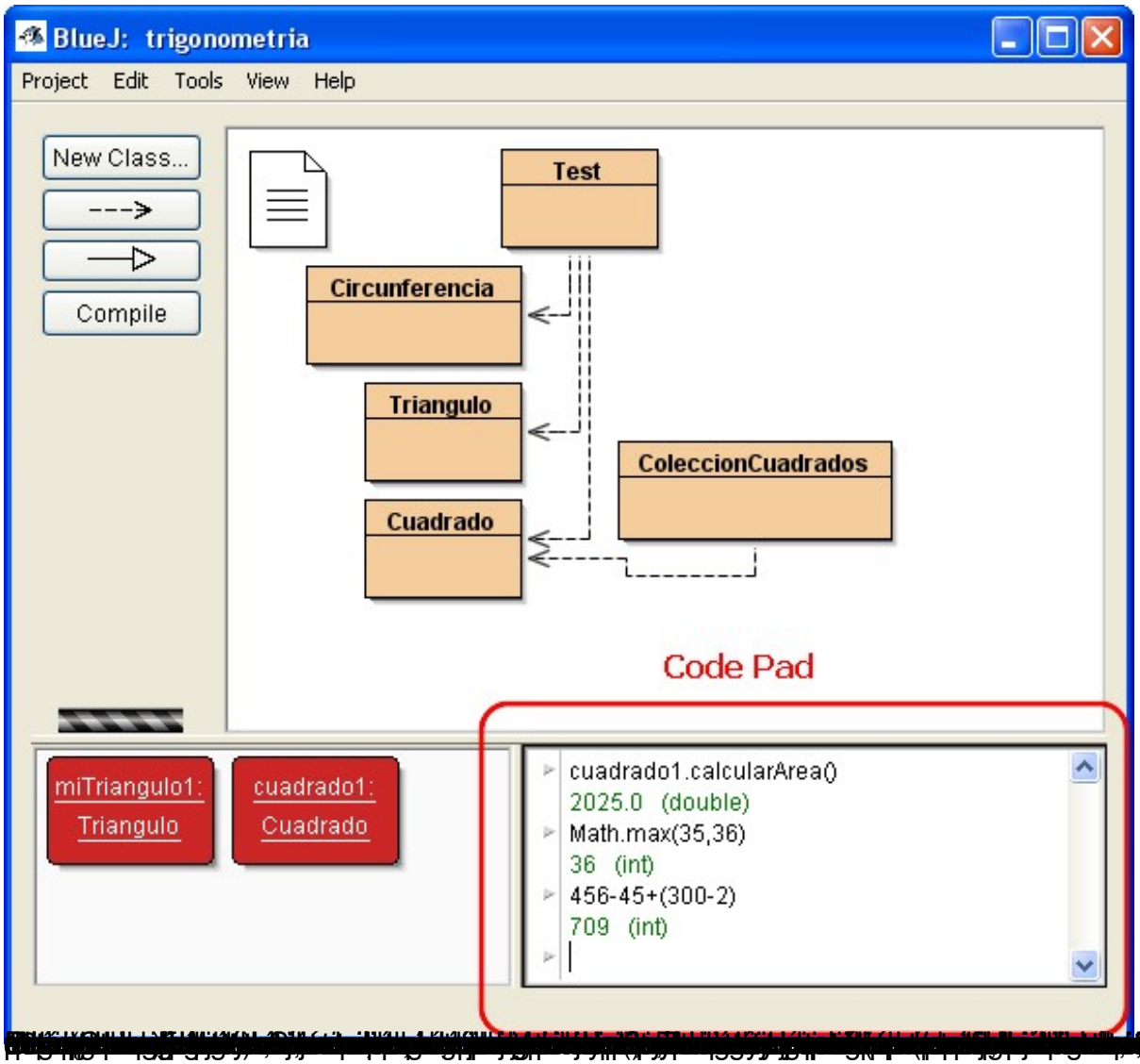
Sobre el suelo del escenario podrás arrastrar todos estos objetos pulsando

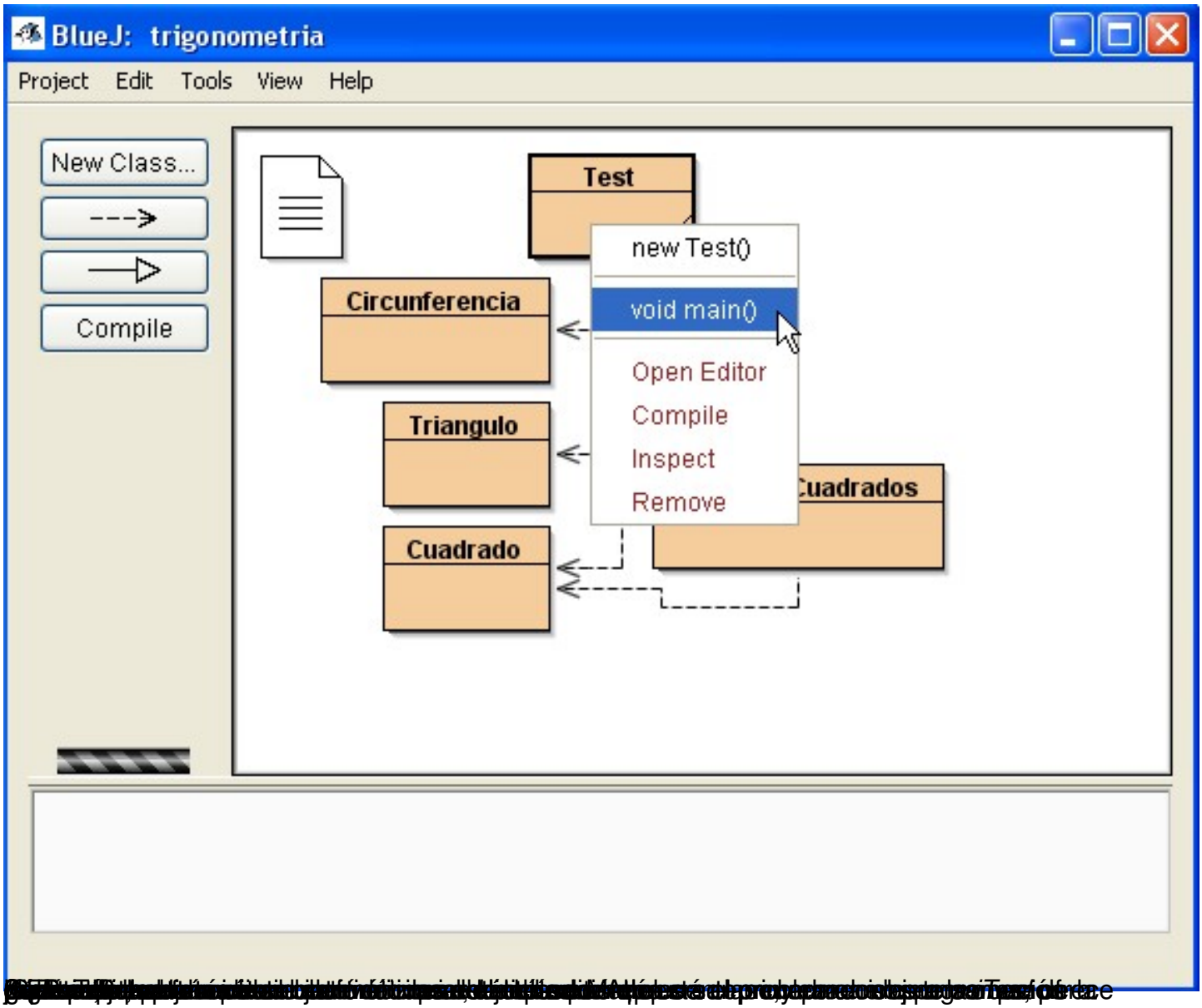


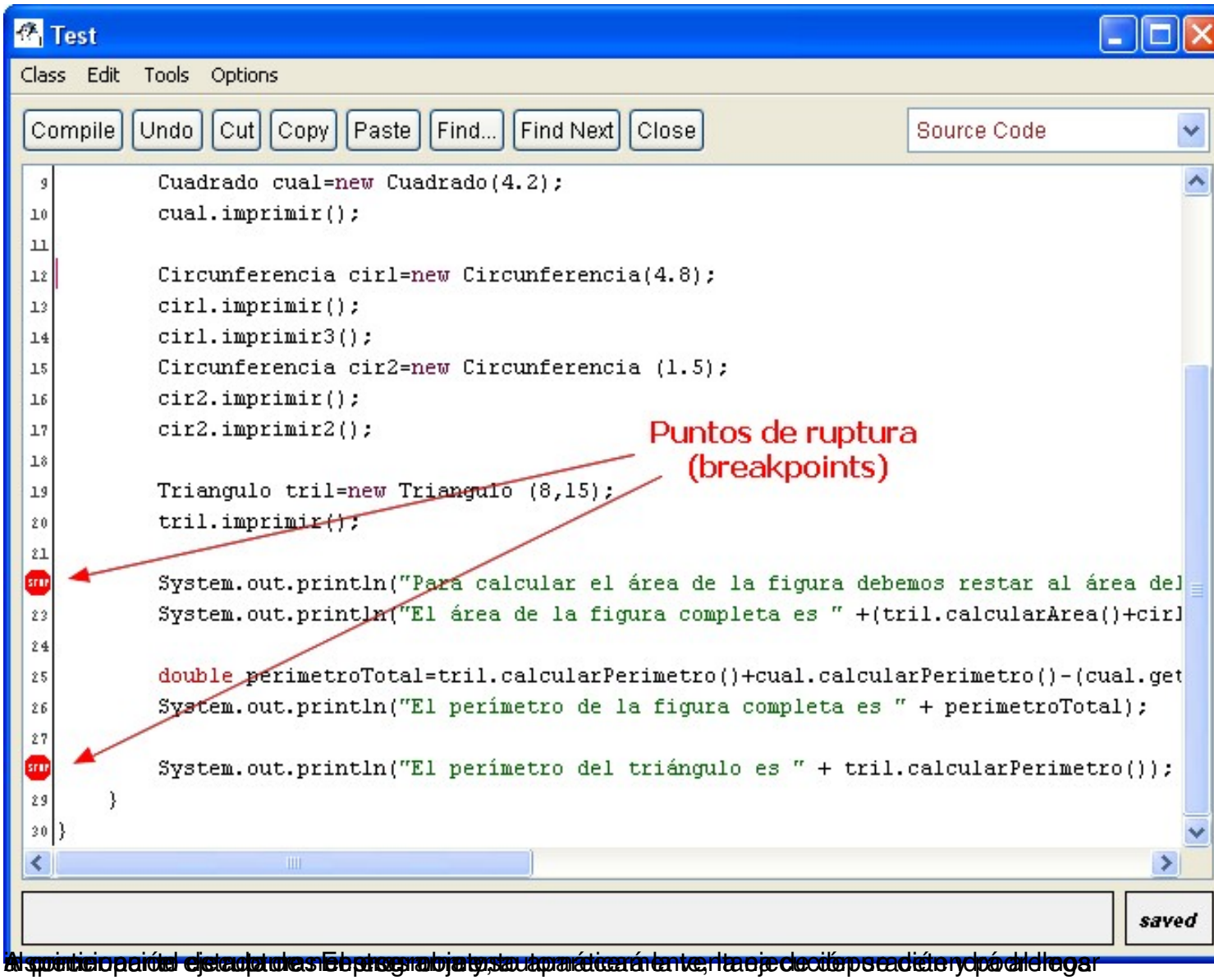
También puedes examinar sus atributos mediante la opción Inspect:

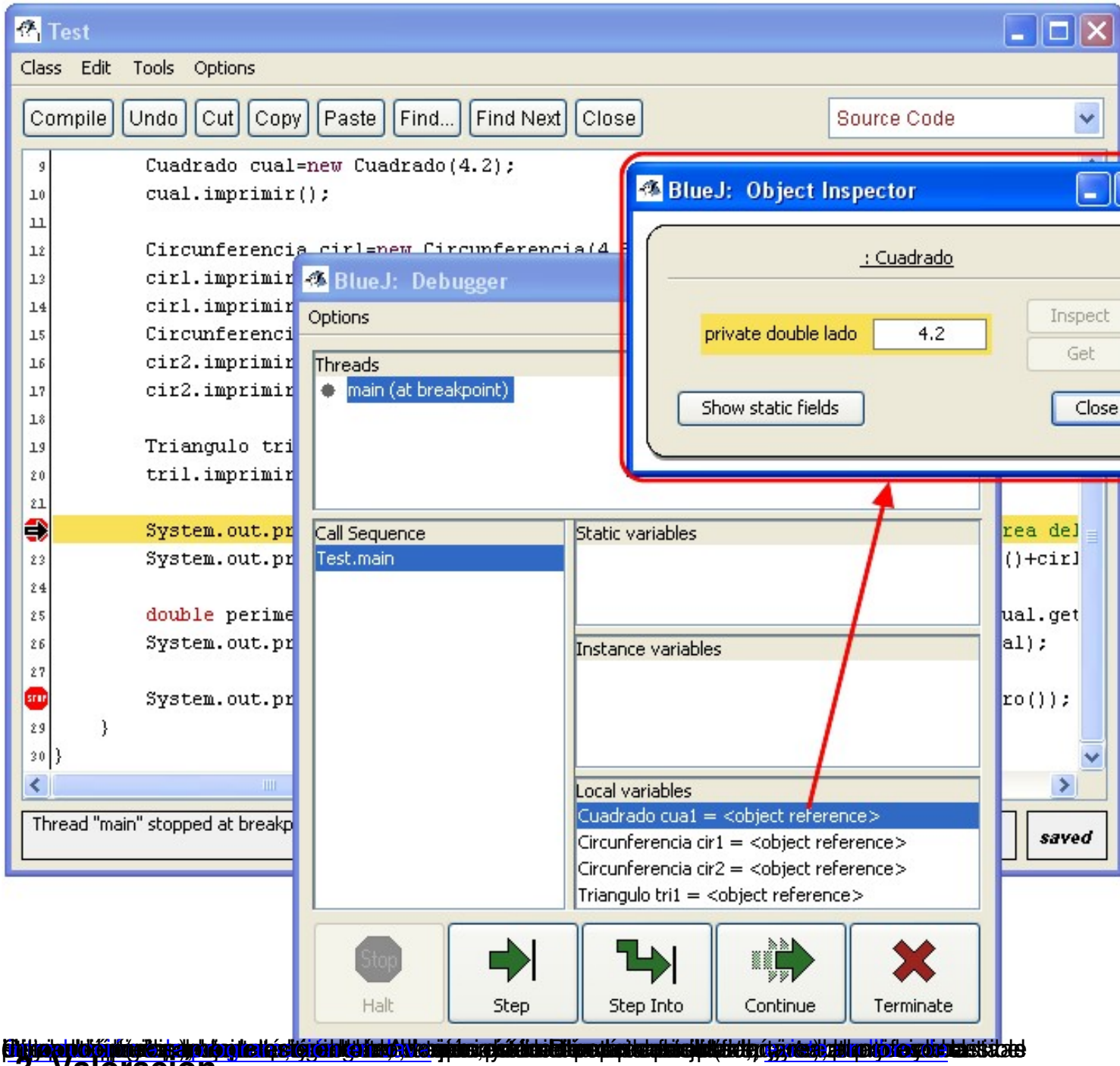


Ejecutando sucesivamente los métodos calcularArea() y calcularPerimetro() podemos ver los valores de la base y la altura de este.


BlueJ: trigonometria
Project Edit Tools View Help
New Class...
Compile
Test
Circunferencia
Triangulo
ColeccionCuadrados
Cuadrado
Code Pad
miTriangulo1: Triangulo
cuadrado1: Cuadrado
cuadrado1.calcularArea()
2025.0 (double)
Math.max(35,36)
36 (int)
456-45+(300-2)
709 (int)


BlueJ: trigonometria
Project
Edit
Tools
View
Help
New Class...
Compile
Test
new Test()
void main()
Open Editor
Compile
Inspect
Remove
Circunferencia
Triangulo
Cuadrado
Cuadrados


Test
Class Edit Tools Options
Compile Undo Cut Copy Paste Find... Find Next Close
Source Code
Cuadrado cual=new Cuadrado(4.2);
cual.imprimir();
Circunferencia cirl=new Circunferencia(4.8);
cirl.imprimir();
cirl.imprimir3();
Circunferencia cir2=new Circunferencia (1.5);
cir2.imprimir();
cir2.imprimir2();
Triangulo tril=new Triangulo (8,15);
tril.imprimir();
System.out.println("Para calcular el área de la figura debemos restar al área del
System.out.println("El área de la figura completa es " +(tril.calcularArea()+cirl
double perimetroTotal=tril.calcularPerimetro()+cual.calcularPerimetro()-(cual.get
System.out.println("El perímetro de la figura completa es " + perimetroTotal);
System.out.println("El perímetro del triángulo es " + tril.calcularPerimetro());
}
}
Puntos de ruptura
(breakpoints)
saved



3. Valoración

Destaquemos para terminar los puntos fuertes y débiles de este entorno:

Puntos fuertes de BlueJ

- Su sencillez y facilidad de manejo resultan simplemente inigualables
- Sus funciones de creación visual de objetos están integradas de forma fácil e intuitiva para el usuario
- Existe documentación oficial en castellano

Puntos débiles de BlueJ

- Se oculta al alumno la gestión de paquetes
- Las ventanas independientes resultan algo caóticas cuando se manejan muchos archivos a la vez
- Los diagramas de clases no aportan ninguna información sobre las mismas, resultando de escasa utilidad

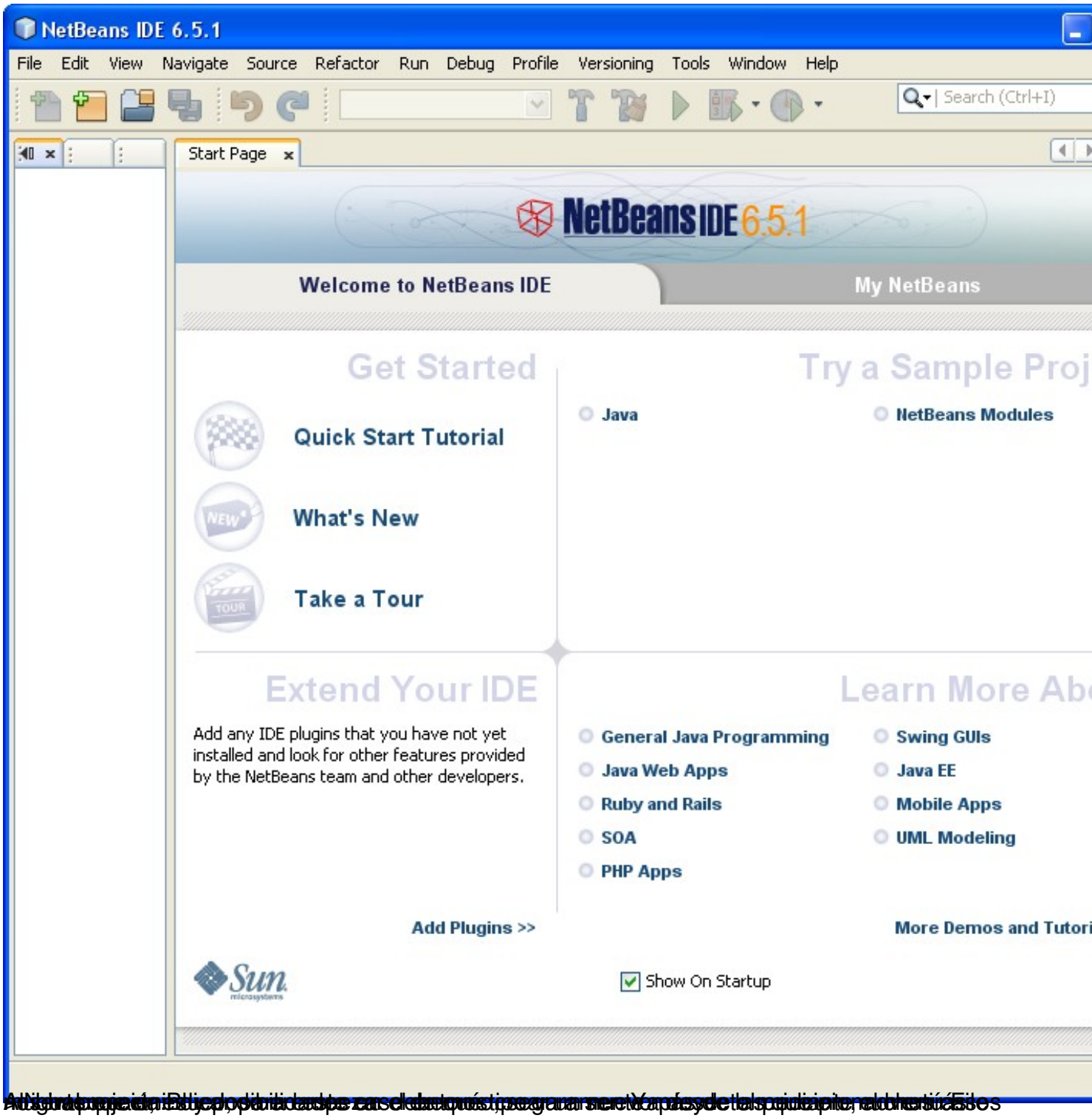
---

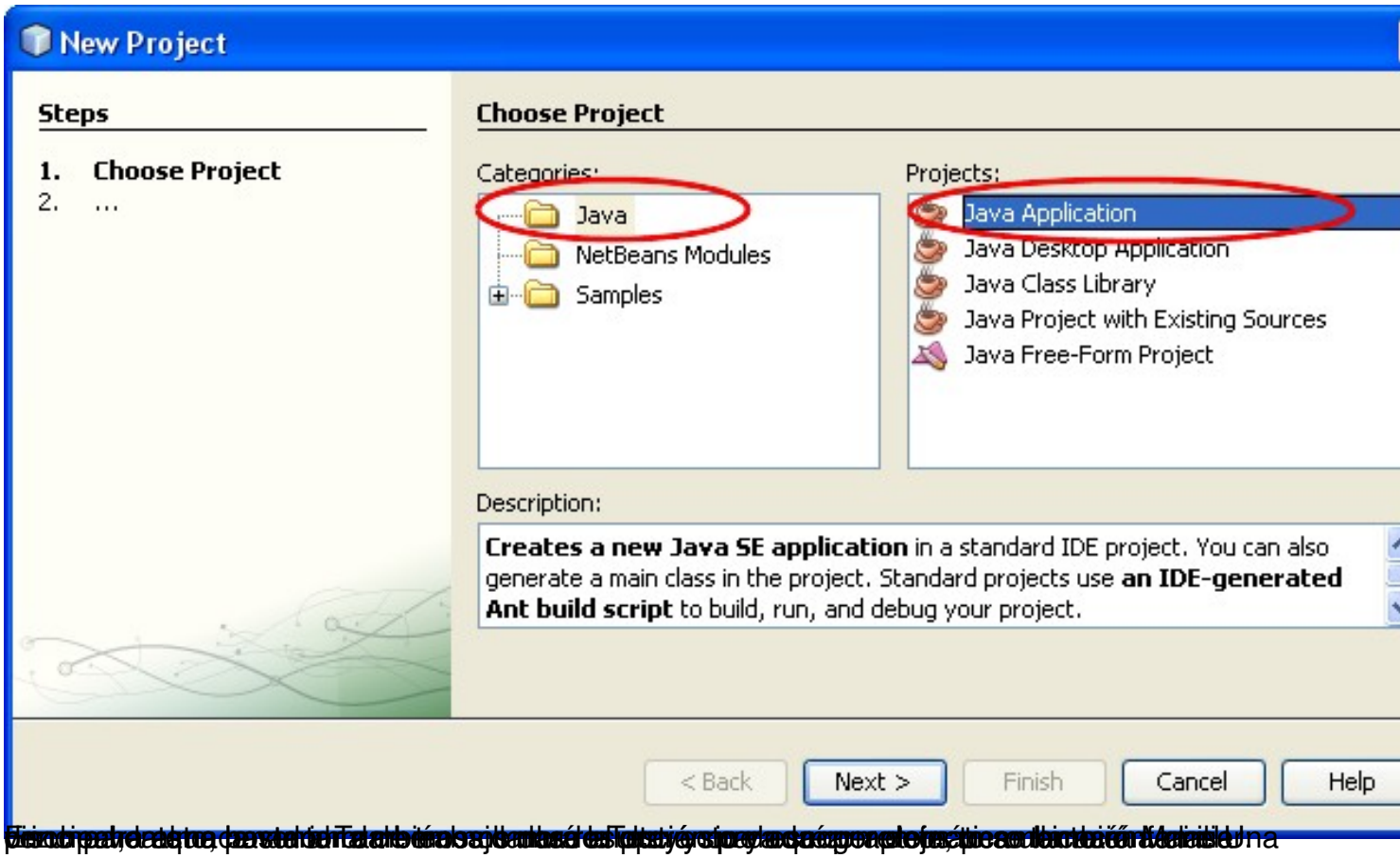
Entorno de desarrollo NetBeans

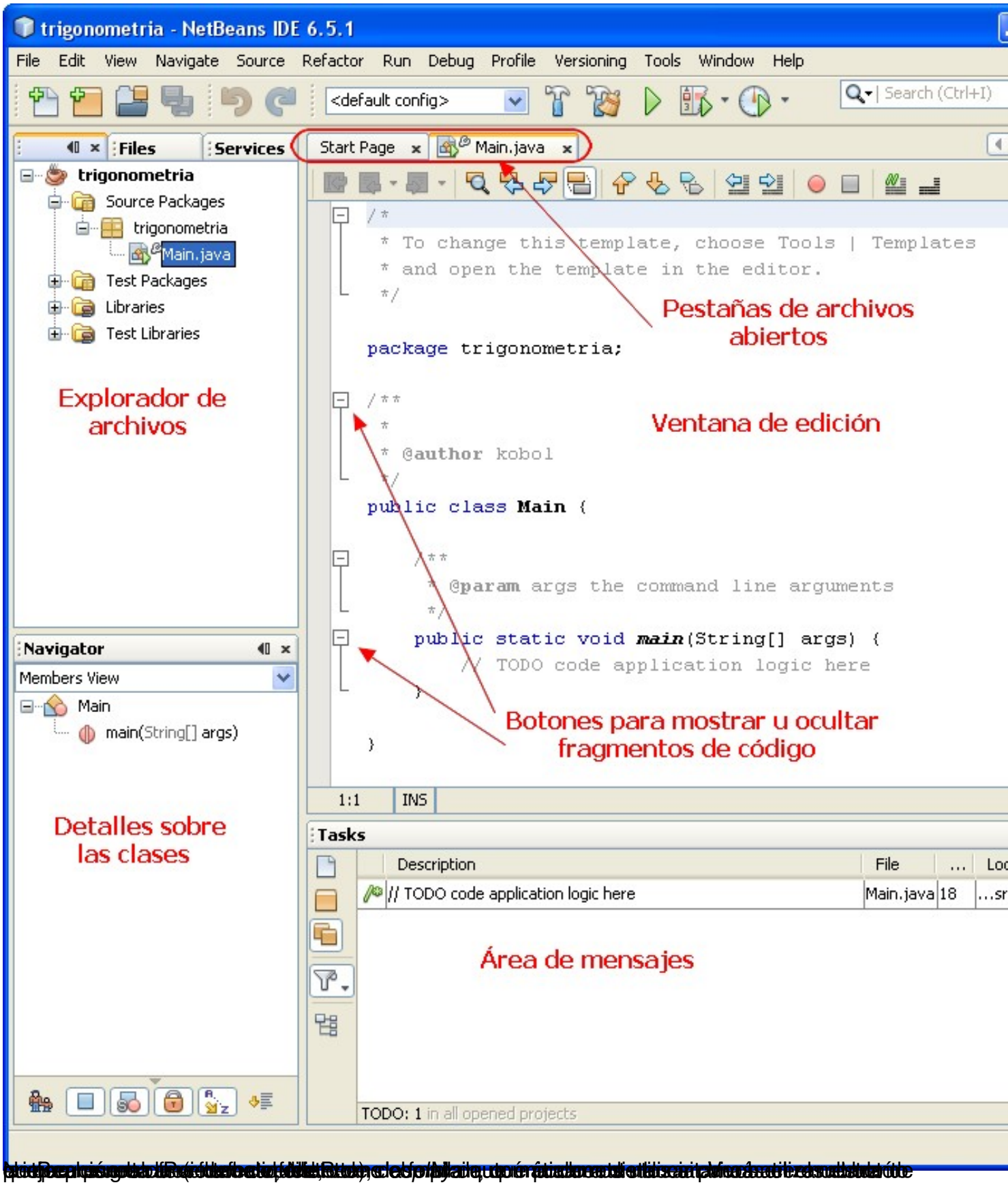
Existen varios entornos que han encontrado aceptación por profesionales y aficionados. Los más conocidos son probablemente Eclipse , NetBeans y IntelliJ IDEA . Los dos primeros resultan más interesantes por ser de código abierto, y para este artículo se ha escogido NetBeans por presentar una ventaja adicional: al ser desarrollado por la compañía Sun, la misma que creó Java, puede descargarse e instalarse en un mismo paquete con el kit de desarrollo JDK, lo cuál simplificará su instalación a los alumnos.

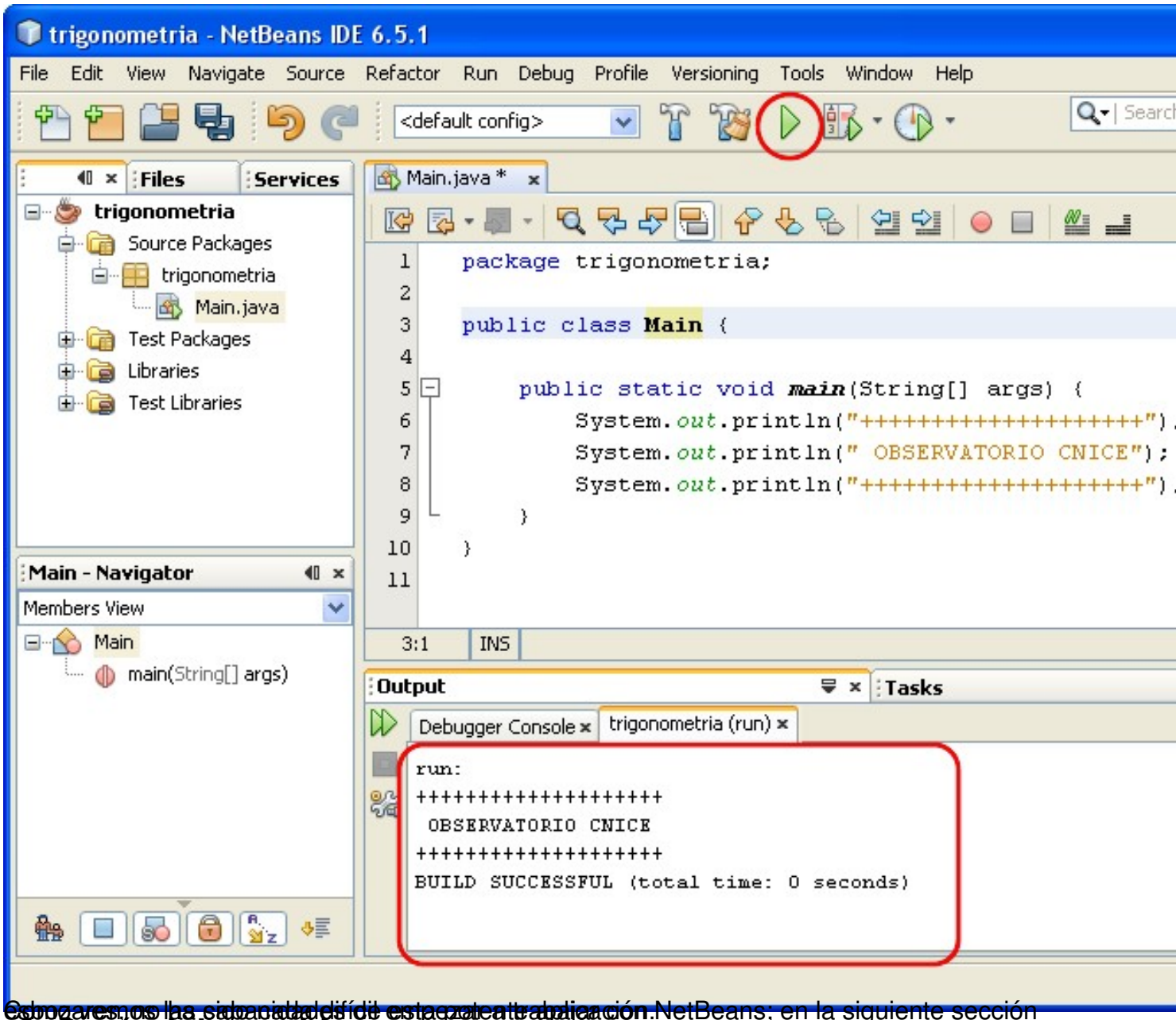
1. Puesta a punto

Al igual que con los otros entornos estudiados, la instalación de NetBeans no presenta ninguna complicación, incluso si se escoge descargarlo por separado desde la web del programa . Sin embargo, al abrir NetBeans nos encontramos una pantalla algo más confusa que en los anteriores entornos:


NetBeans IDE 6.5.1
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Search (Ctrl+I)
Start Page
NetBeans IDE 6.5.1
Welcome to NetBeans IDE
My NetBeans
Get Started
Quick Start Tutorial
What's New
Take a Tour
Try a Sample Proj
Java
NetBeans Modules
Extend Your IDE
Add any IDE plugins that you have not yet installed and look for other features provided by the NetBeans team and other developers.
Learn More Ab
General Java Programming
Java Web Apps
Ruby and Rails
SOA
PHP Apps
Swing GUIs
Java EE
Mobile Apps
UML Modeling
Add Plugins >>
More Demos and Tutori
Sun
Show On Startup


New Project
Steps
1. Choose Project
2. ...
Choose Project
Categories:
Java
NetBeans Modules
Samples
Projects:
Java Application
Java Desktop Application
Java Class Library
Java Project with Existing Sources
Java Free-Form Project
Description:
Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use an IDE-generated Ant build script to build, run, and debug your project.
< Back
Next >
Finish
Cancel
Help

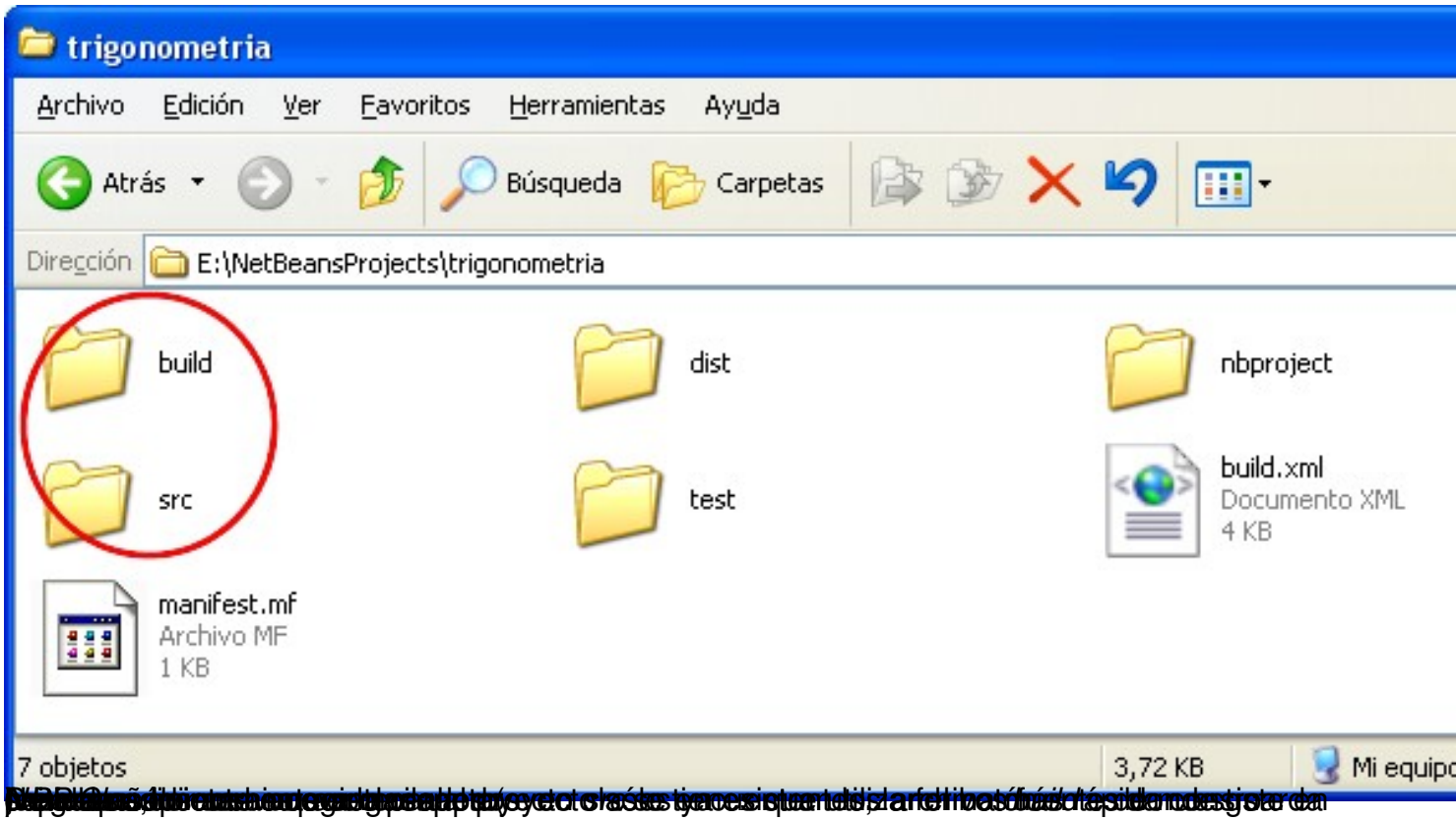

trigonometria - NetBeans IDE 6.5.1
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Search (Ctrl+I)
Files
Services
Start Page
Main.java
trigonometria
Source Packages
trigonometria
Main.java
Test Packages
Libraries
Test Libraries
Explorador de archivos
Pestañas de archivos abiertos
Ventana de edición
Botones para mostrar u ocultar fragmentos de código
Navigator
Members View
Main
main(String[] args)
Detalles sobre las clases
1:1
INS
Tasks
Description
File
// TODO code application logic here
Main.java
18
Área de mensajes
TODO: 1 in all opened projects

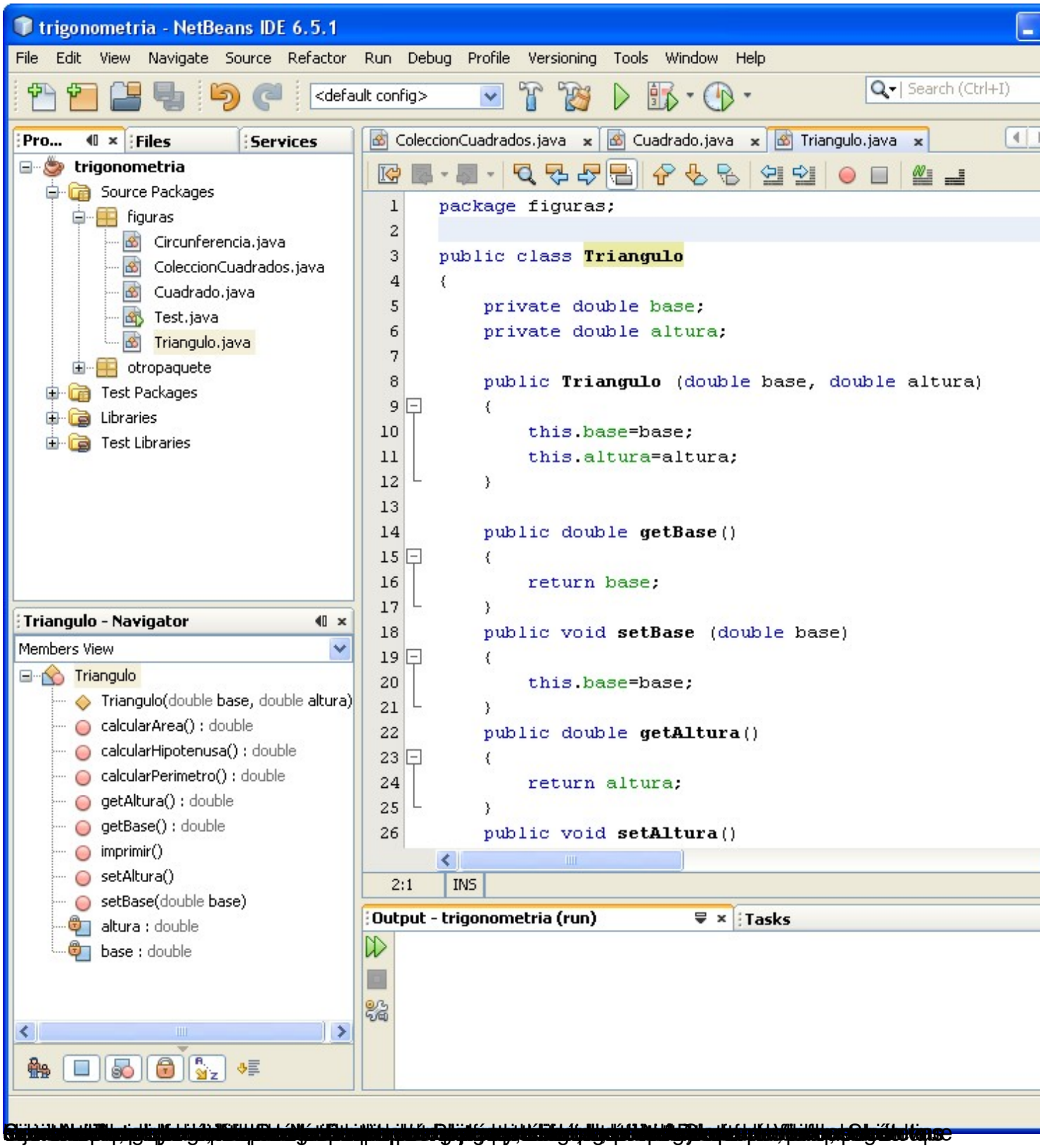


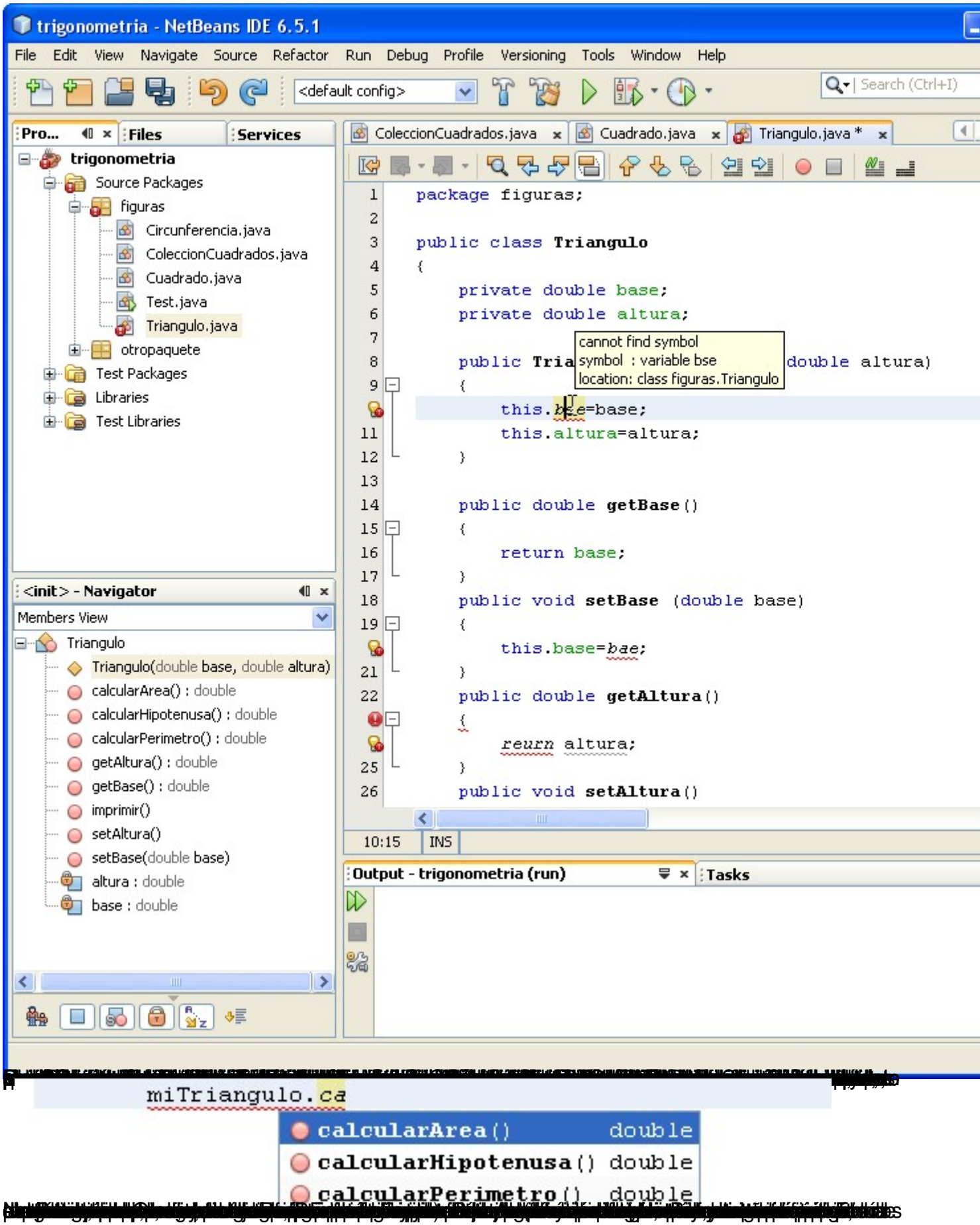
Cabe destacar que la salida de esta aplicación es idéntica a la anterior, ya que hemos utilizado el mismo código. Lo único que hemos cambiado es el entorno de trabajo: NetBeans; en la siguiente sección

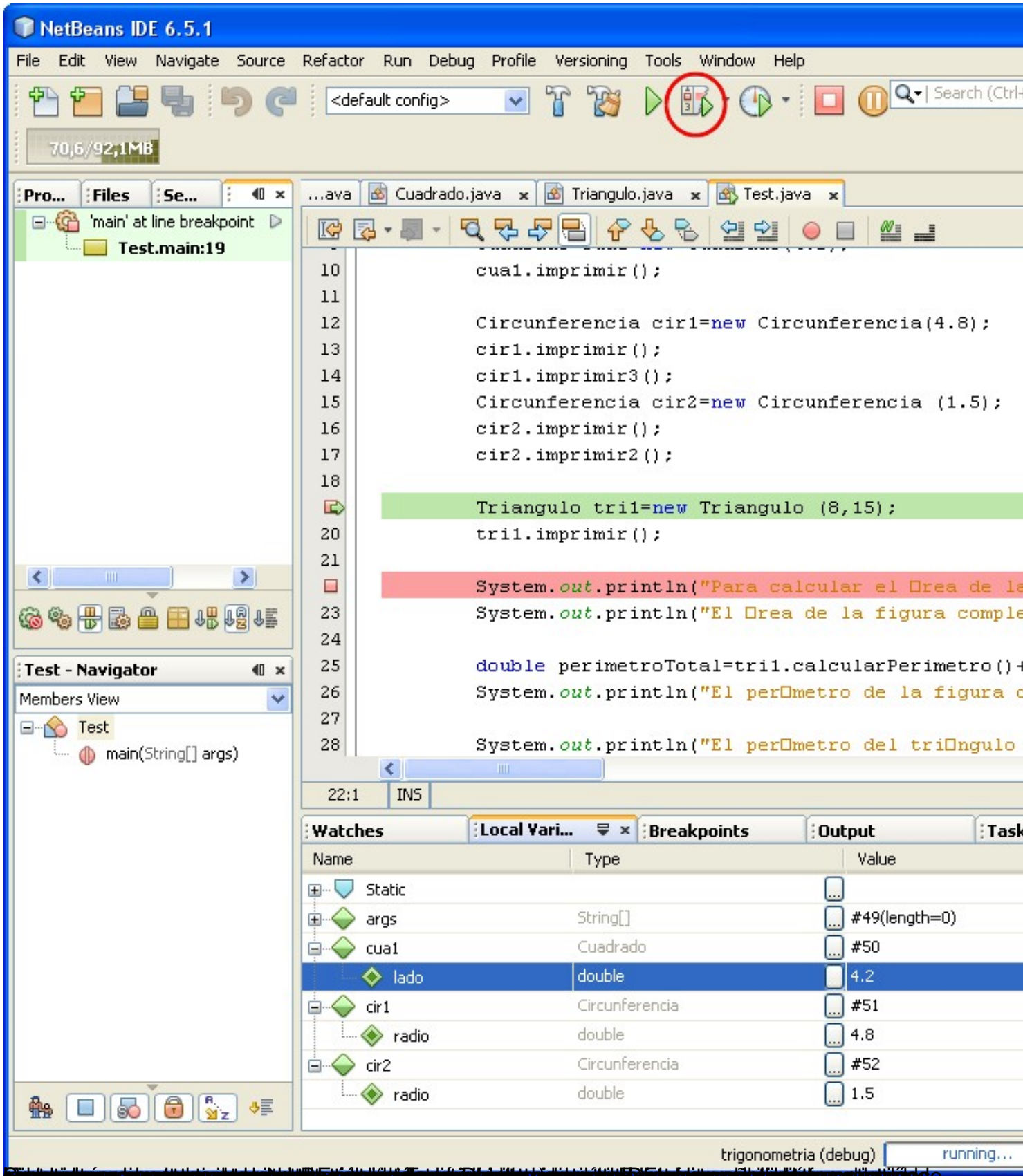
2. Sacándole partido

Examinemos en detalle la carpeta de proyecto. Verás que se han creado múltiples subcarpetas en tu directorio de trabajo:


trigonometria
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás
Búsqueda
Carpetas
Dirección
E:\NetBeansProjects\trigonometria
build
dist
nbproject
src
test
build.xml
Documento XML
4 KB
manifest.mf
Archivo MF
1 KB
7 objetos
3,72 KB
Mi equipo


trigonometria - NetBeans IDE 6.5.1
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Search (Ctrl+I)
Pro... Files Services
trigonometria
Source Packages
figuras
Circunferencia.java
ColeccionCuadrados.java
Cuadrado.java
Test.java
Triangulo.java
otropaquete
Test Packages
Libraries
Test Libraries
Triangulo - Navigator
Members View
Triangulo
Triangulo(double base, double altura)
calcularArea() : double
calcularHipotenusa() : double
calcularPerimetro() : double
getAltura() : double
getBase() : double
imprimir()
setAltura()
setBase(double base)
altura : double
base : double
ColeccionCuadrados.java
Cuadrado.java
Triangulo.java
1 package figuras;
2
3 public class Triangulo
4 {
5 private double base;
6 private double altura;
7
8 public Triangulo (double base, double altura)
9 {
10 this.base=base;
11 this.altura=altura;
12 }
13
14 public double getBase()
15 {
16 return base;
17 }
18 public void setBase (double base)
19 {
20 this.base=base;
21 }
22 public double getAltura()
23 {
24 return altura;
25 }
26 public void setAltura()
2:1 INS
Output - trigonometria (run)
Tasks


trigonometria - NetBeans IDE 6.5.1
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
<default config>
Search (Ctrl+I)
Pro... Files Services
trigonometria
Source Packages
figuras
Circunferencia.java
ColeccionCuadrados.java
Cuadrado.java
Test.java
Triangulo.java
otropaquete
Test Packages
Libraries
Test Libraries
<init> - Navigator
Members View
Triangulo
Triangulo(double base, double altura)
calcularArea() : double
calcularHipotenusa() : double
calcularPerimetro() : double
getAltura() : double
getBase() : double
imprimir()
setAltura()
setBase(double base)
altura : double
base : double
ColeccionCuadrados.java
Cuadrado.java
Triangulo.java *
1 package figuras;
3 public class Triangulo
4 {
5 private double base;
6 private double altura;
8 public Tria double altura)
cannot find symbol
symbol : variable bse
location: class figuras.Triangulo
9 {
this.bse=base;
11 this.altura=altura;
12 }
14 public double getBase()
15 {
16 return base;
17 }
18 public void setBase (double base)
19 {
this.base=bae;
21 }
22 public double getAltura()
{
reurn altura;
25 }
26 public void setAltura()
10:15 INS
Output - trigonometria (run)
Tasks
miTriangulo.ca
calcularArea() double
calcularHipotenusa() double
calcularPerimetro() double



3. Valoración

Al igual que con el resto de entornos, terminemos con una breve exposición de sus aspectos más destacados:

Puntos fuertes de NetBeans

- Sin duda, el más completo, estable y fiable de los tres
- Si un alumno necesita programar en su vida profesional y ha aprendido con NetBeans, podrá enfrentarse con confianza a cualquier entorno presente en la empresa, ya que todos son muy parecidos entre sí
- La gestión de paquetes y sus avanzadas detecciones de errores (incluso antes de compilar) resultan más cómodas e intuitivas que en los otros entornos

Puntos débiles de NetBeans

- Su consumo de recursos es significativamente mayor que el de las otras alternativas
- Enfrentarse a un entorno algo complejo como NetBeans puede desanimar o confundir al alumno, especialmente si no tiene nociones de programación
- Sus múltiples ayudas al programador pueden no ser adecuadas para iniciarse en el lenguaje Java

---

Conclusiones

Seguramente a estas alturas ya te hayas hecho una idea de qué entorno te interesa más; pero por si acaso, hagamos un repaso final.

- BlueJ es un entorno visualmente agradable, sencillo, y con potentes añadidos para facilitar la tarea docente; pero el trabajo intensivo resulta más incómodo y en su búsqueda de simplicidad elude algunos aspectos del lenguaje Java.

- jGrasp es algo menos atractivo y, si bien ofrece también las algunas facilidades interactivas, no quedan tan bien integradas como en BlueJ. Sin embargo, su organización de archivos es mucho más clara, y su uso prepara mejor al alumno para el uso de entornos profesionales.

- Uno de esos entornos profesionales es NetBeans: todas las funciones avanzadas están presentes en este potente IDE, con la ventaja adicional que ofrece su sencilla instalación junto con el kit de desarrollo. No obstante, utilizar un entorno tan potente para iniciarse quizá sea excesivo y desde luego no es buena idea si se dispone de equipos con características más bien modestas.

La siguiente tabla recoge esquemáticamente algunas de las diferencias entre los entornos estudiados:

Característica	BlueJ	jGrasp	NetBeans
Versión del entorno en castellano	No	No	Algunas versiones
Existencia de documentación oficial en castellano	Tutorial y libro	No	No
Disponibilidad en Linux	Sí	Sí	Sí
Tipo de licencia	GNU (software libre)	Propietaria (freeware)	CDDL y GPL2 (software l
Permite trabajar con otros lenguajes de programación	No	Sí	Sí
Generación automática de documentación (JavaDoc)	Sí	Sí	Sí
Información esquemática sobre las clases	Escasa	Completa (diagrama UML)	Abundante
Gestión de paquetes	Confusa	Regular	Buena
Funciones de depuración	Muy básicas	Básicas	Avanzadas
Funcionalidades avanzadas	No (refactorización, control de versiones...)	No	Sí
Creación y manejo de objetos fácil	Muy fácil	Fácil	No
Espacio aproximado en disco duro	10 MB	6 MB	160 MB
Consumo aproximado de memoria (10 clases abiertas en el editor)	70 MB	50 MB	140 MB
Fiabilidad para uso intensivo en proyectos semi o profesionales	No	No	Sí
Comodidad para trabajar con múltiples archivos abiertos	Baja	Alta	Alta
Complejidad de uso	Muy baja	Baja	Media

Ten muy presente que la facilidad de uso de un programa no puede expresarse en una tabla. Si tienes dudas, lo mejor es instalarlos y hacer unas cuántas pruebas: al final todo es cuestión

de gustos, y deberías escoger aquel entorno con el que sientas más seguridad y confianza. ¿Aún no lo tienes claro? Un último criterio: piensa en el motivo por el que tus alumnos aprenderán Java. Si es una mera iniciación a la programación pero no es probable que vayan a programar habitualmente en un futuro próximo, escoge BlueJ; pero si estás preparando a tus alumnos para ser programadores, utiliza NetBeans. Si te encuentras en una situación intermedia, puedes apostar por JGrasp. Cualquiera que sea el entorno elegido, la enseñanza de Java resultará una experiencia estimulante y creativa: y no sólo para los alumnos, sino también para el profesor.