IIP (E.T.S. de Ingeniería Informática)
Curso 2019-2020

Práctica 2. Objetos, clases y programas. El entorno BlueJ

Profesores de IIP
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Contexto y trabajo previo a la sesión de prácticas

En el marco académico, la presente práctica es la segunda del curso y tiene como objetivo principal la definición y uso de clases utilizando el entorno de programación *BlueJ*. Con esta práctica se pretende que el alumno se familiarice con los aspectos básicos de *BlueJ*, siendo capaz de utilizarlo en adelante como entorno de trabajo para las prácticas de la asignatura. El alumno será capaz de utilizar el entorno para:

- Crear un proyecto *BlueJ* que incluya un paquete con las clases que se proporcionan al inicio de la práctica.
- Editar alguna de las clases existentes.

- Crear objetos y ejecutar métodos sobre ellos utilizando el banco de objetos de *BlueJ* (*Object Bench*).
- Evaluar expresiones utilizando el intérprete de instrucciones de *BlueJ* (*Code Pad*).
- Generar de forma automática la documentación del proyecto.
- Validar el funcionamiento de una clase o de cualquier método de la misma, utilizando el depurador de *BlueJ* (*Debugger*).

Para aprovechar al máximo la sesión de prácticas, se aconseja al alumno que, antes, realice una lectura comprensiva de este boletín.

2. Desarrollo de un proyecto *BlueJ*

Como ya se ha indicado en el apartado 7 del boletín de la práctica 1 (*Introducción: Linux, Java y BlueJ*), un proyecto *BlueJ* consta de un conjunto de clases relacionadas entre sí y el entorno *BlueJ* no sólo permite desarrollar dicho proyecto (crear, compilar, ejecutar, depurar, documentar) sino que también *permite interactuar con cualquier método (atributo u objeto) de cualquiera de las clases incluidas en el proyecto*. Además, en un proyecto *BlueJ* las diferentes clases que lo componen pueden estar organizadas en *paquetes* o *librerías* de clases, siguiendo la organización de clases de Java, permitiendo así su posterior reutilización desde otras clases.

El objetivo de esta práctica es que el alumno se familiarice con todos estos conceptos haciendo uso del entorno *BlueJ*.

2.1. Invocación a *BlueJ* y creación de un proyecto

Como ya se indicó en la práctica anterior, *BlueJ* se puede invocar desde el menú desplegable del entorno gráfico del sistema `Aplicaciones - Programación - BlueJ 4.2.1` o desde la línea de comandos, con o sin argumentos:

```
bluej &  ó  bluej nombreProyecto &
```

Nótese que la segunda opción es equivalente a invocar a *BlueJ* sin argumentos y luego, utilizando la opción `Proyecto - Abrir proyecto...` del menú, seleccionar el proyecto `nombreProyecto`. Si se quiere abrir con *BlueJ* una aplicación ya existente pero que **no** ha sido desarrollada con *BlueJ*, se debe invocar *BlueJ* sin argumentos y entonces abrir la aplicación existente con la opción `Proyecto - Abrir No BlueJ...` del menú.

Nota: Se puede cambiar el idioma del entorno *BlueJ* utilizando la opción `Herramientas - Preferencias - Interfaz` o en inglés: `Tools - Preferences - Interface`.

Actividad #1

1. Descargar el fichero `iip.jar` disponible en la carpeta `Recursos/Laboratorio/ Práctica 2/Castellano/Código` de PoliformaT en `$HOME/DiscoW`.
2. Invocar a *BlueJ* sin argumentos.
3. Abrir el fichero `iip.jar` con la opción `Open ZIP/JAR...` del menú. Al hacerlo se descomprimirá el fichero y aparecerán nuevos subdirectorios en el `$HOME/DiscoW`.

`iip` es un proyecto *BlueJ* con un paquete llamado `pract2`. En la ventana principal de *BlueJ* aparecen los iconos de cada una de las clases del paquete (Figura 1) y un icono de

una carpeta con el texto `<go up>` que permite acceder al proyecto `iip`. Nótese que los iconos asociados a los ficheros `.java` aparecen rayados si aún no han sido compilados. Las flechas indican las relaciones de uso que existen entre las clases. Si de un fichero sólo se encuentra el archivo `.class`, entonces es que no se encuentra el código fuente del mismo (no puede editarse pero sí puede utilizarse) y aparece marcado con el texto `(no source)`.

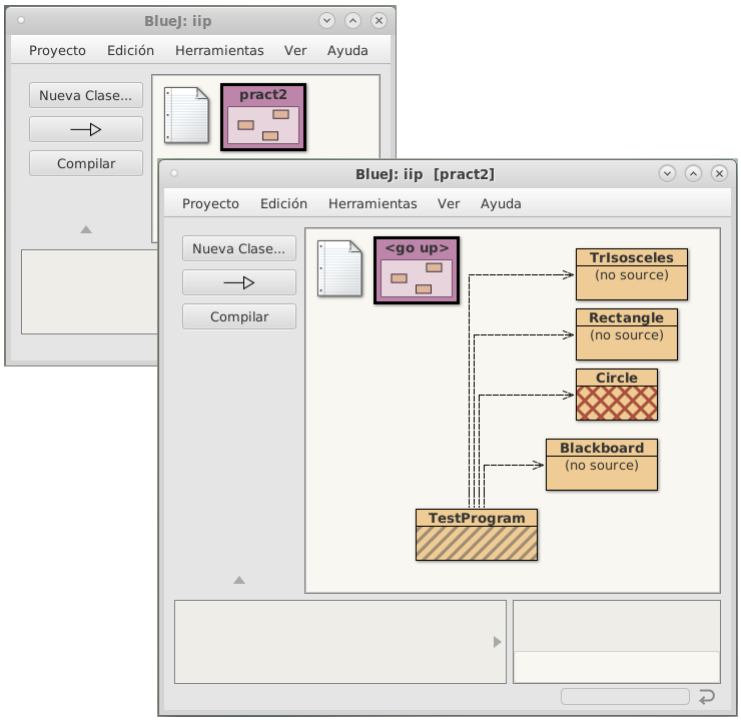


Figura 1: Proyecto `iip` que incluye el paquete `pract2`.

2.2. Operaciones de edición y compilación

Para editar una clase, en nuestro caso `Circle`, se puede usar la opción `Abrir Editor` del menú de la clase, o también haciendo doble "click" sobre el icono de la clase. Una clase se puede compilar desde el propio editor o, *si se desea compilar todo el proyecto*, desde la barra de herramientas de la ventana principal de *BlueJ*. *BlueJ* realiza una precompilación del código y si se producen errores sintácticos el propio editor marca dichos errores con una señal roja a la izquierda del texto y un pequeño subrayado donde considera que hay un error. Por ejemplo, si el compilador detecta que falta un punto y coma, al situar el cursor en la parte subrayada aparece el mensaje en un pequeño recuadro como puede verse en la Figura 2. Es más, si se pulsa en la palabra `Errors` en la parte inferior izquierda de la zona de información del editor, se puede ir pasando de error en error detectado. Una vez corregidos estos errores, se puede pulsar el botón de compilar en la parte superior de la ventana y, entonces, o bien puede resultar el código correctamente compilado o pueden aparecer nuevos errores.

Figura 2: Compilación de la clase `Circle`.

Actividad #2

1. Comprobar que la primera línea de las clases `Circle` y `TestProgram` incluye la directiva del compilador para indicar que son clases pertenecientes a un paquete:

   ```
   package pract2;
   ```

2. Compilar la clase `Circle` y corregir los posibles errores de compilación.

3. Compilar la clase `TestProgram`. Observar qué ocurre en la parte central de la ventana principal de *BlueJ*.

Una de las características que ofrece el editor de *BlueJ* es el autocompletado de código que se activa al pulsar las teclas `Ctrl-Space`.

Actividad #3

La clase `TestProgram` no tiene ningún error de compilación pero sí tiene un error lógico: aunque la intención del programador era mostrar por pantalla el perímetro del círculo creado, realmente no es así. En esta actividad se corregirá este error. Para ello, se debe editar la clase `TestProgram` y completar la instrucción que muestra por pantalla el perímetro del objeto `Circle c`. Escribir `c.` y pulsar `Ctrl-Space`. Comprobar que, como en la Figura 3, aparece un listado de todos los métodos disponibles en la clase junto con una pequeña descripción de los mismos. Seleccionar el que interese y tras pulsar `Enter`, *BlueJ* lo autocompletará.

A continuación, desde la opción `Herramientas - Checkstyle` en la vista de proyecto, comprobar si el código de alguna clase tiene errores de estilo y corregirlos si es el caso (Figura 4).

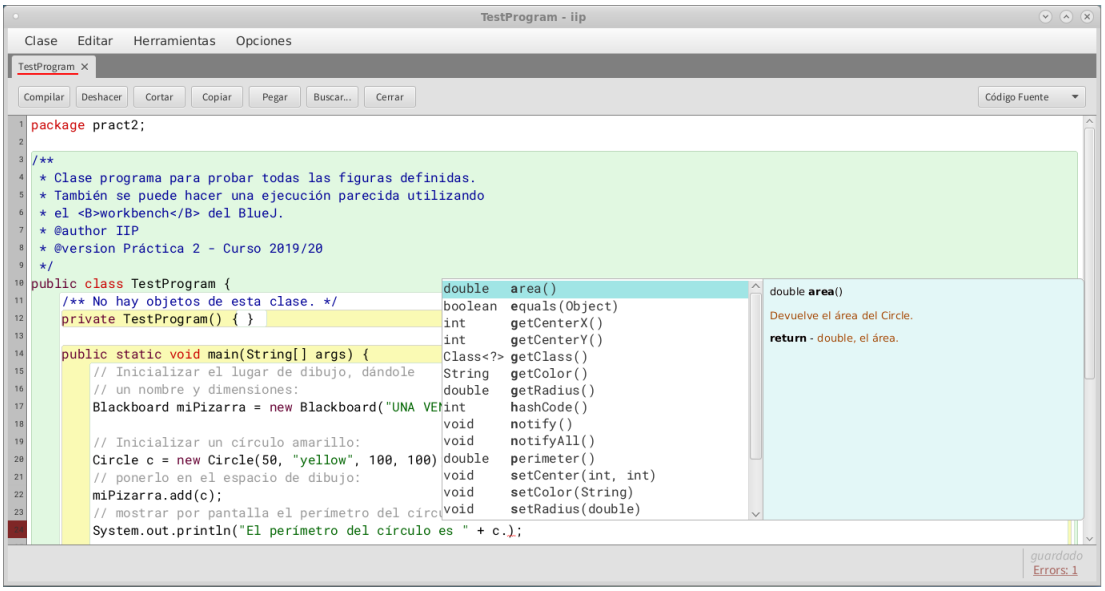


Figura 3: Autocompletado de código en el editor de *BlueJ*.

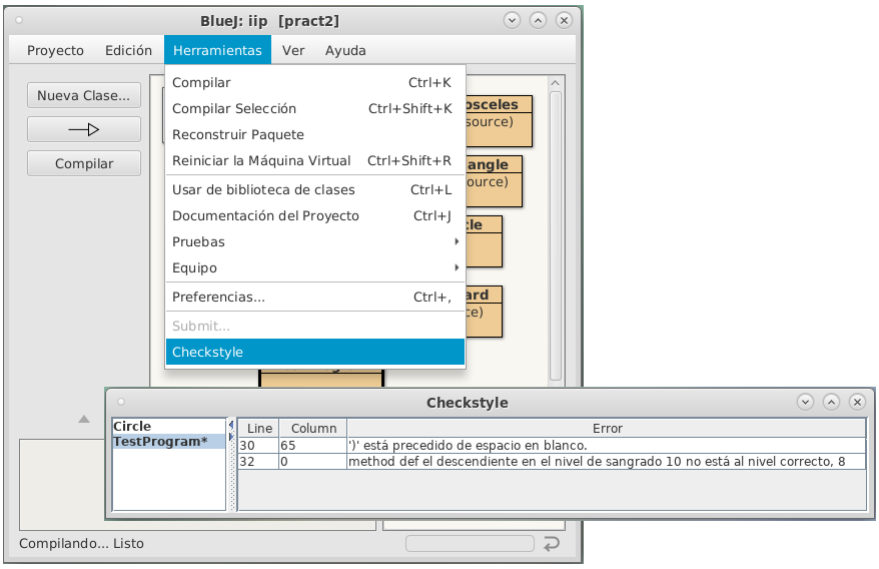


Figura 4: Comprobación del estilo del código en *BlueJ*.

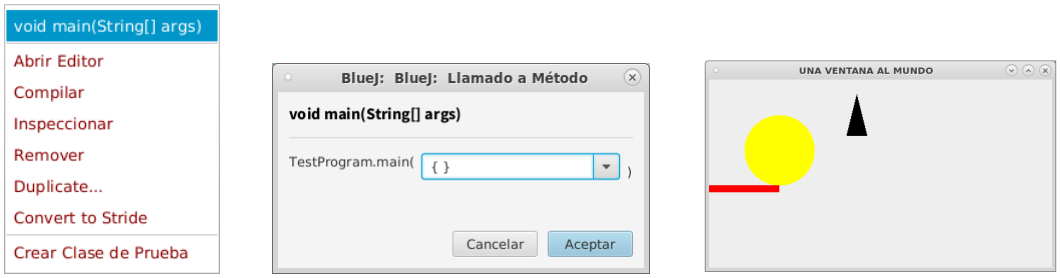
2.3. Ejecución de una clase: llamadas a `main`

Como ya se ha mencionado anteriormente, al marcar el icono de una clase y hacer "clic" con el botón derecho del ratón, se obtiene el menú de la clase; pues bien, de la lista de operaciones que éste contiene destacaremos ahora la operación de llamada al método `main` de la clase; *se puede ejecutar una clase desde su menú*. No es necesario pasar ningún argumento al `main` de `TestProgram`, sólo aparecen las llaves.

En la Figura 5(a) se muestra el menú emergente de la clase `TestProgram` y en la Figura 5(b) la ventana de la llamada al método `main` de la clase `TestProgram`.

Actividad #4

1. Ejecutar el método `main` de la clase `TestProgram`. El resultado debe ser como el que aparece en la Figura 5(c) y en el terminal se debe mostrar el mensaje de la Figura 6.



(a) Menú de la clase. (b) Llamada al método `main`. (c) Resultado de la ejecución.

Figura 5: Ejecución de la clase `TestProgram`.

En caso de que el programa solicite datos desde el teclado o muestre por pantalla algún resultado, aparece de forma automática un terminal de texto (Figura 6). Si no aparece, se debe seleccionar la opción `Mostrar Terminal` del menú `Ver`. Es importante señalar que, en la versión 4.2.1 de *BlueJ*, cuando que hay que introducir datos desde teclado, se hace en una línea adicional en la parte inferior de la ventana de terminal que, si el programa no necesita entrada de datos en ese momento, aparece atenuada.

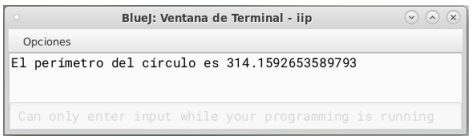


Figura 6: Ventana de terminal de *BlueJ*.

Si se ejecuta nuevamente el `main` de `TestProgram`, se puede comprobar que en la ventana de terminal se vuelve a escribir el mensaje. Para que, en cada nueva ejecución, se borre del terminal el resultado de la ejecución anterior, seleccionar, desde el menú `Opciones` de la ventana de terminal, la opción `Limpiar la pantalla en llamada al método`, como en la Figura 7. Para futuras prácticas, seleccionar también la opción `Buffering ilimitado`.

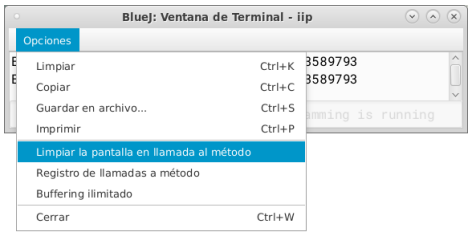


Figura 7: Opciones de la ventana de terminal de *BlueJ*.

2.4. La opción *Herramientas*. Generar documentación

De las diferentes utilidades de la opción `Herramientas` cabe destacar `Documentación del Proyecto`, que genera el subdirectorio `doc` con la documentación sobre las clases en formato `html`. Nótese que la documentación individual de una clase se puede generar también al editar la clase y seleccionar, en vez de `Código Fuente`, la opción `Documentación` o también seleccionando `Herramientas - Activar/Desactivar Vista de Documento`.

Actividad #5

Generar la documentación del proyecto y consultarla. El resultado será como el que aparece en la Figura 8. Haciendo "click", por ejemplo, en el enlace de la clase `Circle` se puede consultar su documentación.

Figura 8: Documentación del proyecto `pract2`.

2.5. La opción *Ayuda*

La utilidad `Librerías de Clases Java` permite acceder a la documentación de Java. En la instalación por defecto esta ayuda se encuentra en `https://docs.oracle.com/en/java/javase/11/docs/api/`. Si se quiere cambiar a un directorio local, se puede hacer desde la opción `Herramientas/Preferencias/Miscelánea` del menú.

Otras utilidades de `Ayuda` permiten consultar el manual de *BlueJ* así como acceder a su web `www.bluej.org`.

3. Uso del banco de objetos (*Object Bench*)

Una de las características más interesantes del entorno *BlueJ* es que permite interactuar con objetos aislados de cualquier clase y ejecutar los métodos que sobre ellos se hayan definido; de este modo se puede comprobar la funcionalidad de la clase antes de escribir cualquier aplicación que la utilice.

3.1. Operaciones de una clase

Para acceder a las operaciones aplicables a una determinada clase hay que marcar el icono de la clase y hacer "click" con el botón derecho del ratón. Aparece una lista con las operaciones constructoras de la clase y otras operaciones permitidas por el entorno como, por ejemplo, borrar la clase o compilarla (Figura 9).

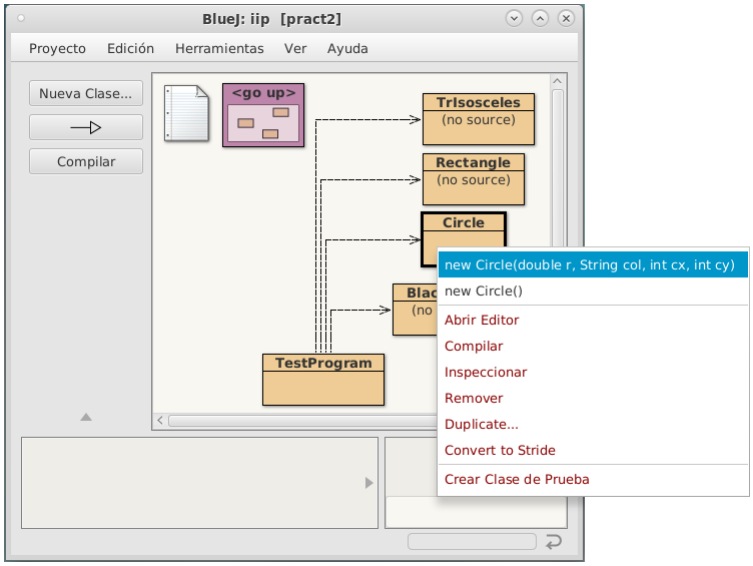


Figura 9: Menú de la clase `Circle`.

3.2. Creación de un objeto

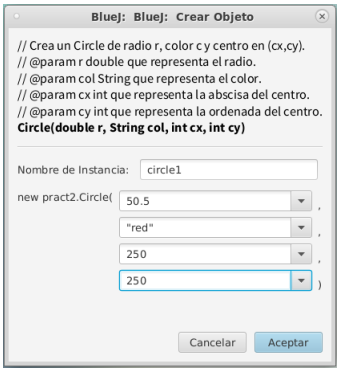
Si se desea crear un objeto se debe seleccionar de este menú una de las operaciones constructoras y seguir el cuadro de diálogo que se abre (Figura 10(a)). En concreto, se pide un nombre para el objeto. Cuando se crea este objeto, aparece en la esquina inferior izquierda de la pantalla principal de *BlueJ*, en la zona conocida como *banco de objetos* (*Object Bench*) (Figura 10(b)).

3.3. Ejecución de métodos sobre el objeto

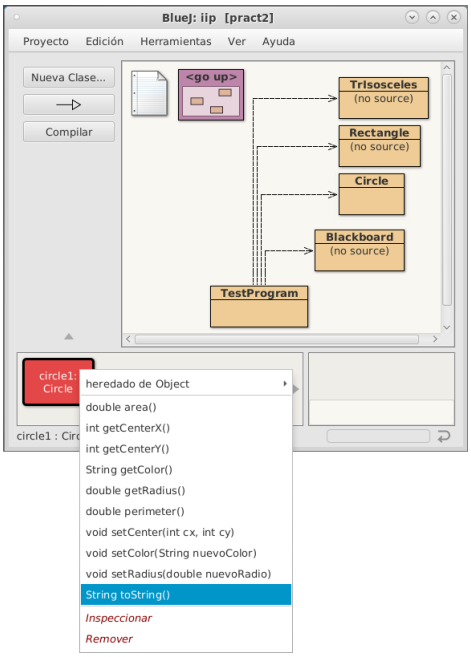
Si se pulsa con el botón derecho del ratón sobre el objeto creado se accede a los métodos que se pueden ejecutar sobre el mismo (Figura 10(b)). Para ejecutar uno de ellos sólo hay que seleccionarlo. Si el objeto hereda métodos de otras clases también aparecen a través de submenús.

3.4. Observación del estado del objeto

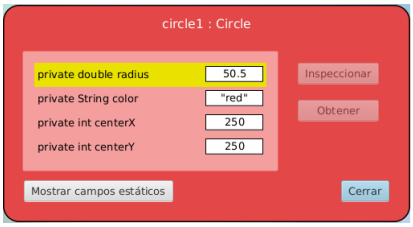
Para depurar los métodos diseñados se puede utilizar la opción `Inspeccionar`. Esta operación permite conocer los valores de los campos (atributos) de los objetos (Figura 10(c)).



(a) Creación de un objeto `Circle`.



(b) Banco de objetos y menú del objeto `Circle`.



(c) Inspección del objeto `Circle`.

Figura 10: Creación de un objeto en el banco de objetos de *BlueJ*, menú y observación del estado de dicho objeto.

Actividad #6

1. Crear un objeto de la clase `Circle` de radio 50.5, color rojo y con centro en $(250, 250)$.
2. Consultar los valores de los atributos de este objeto.
3. Ejecutar el método `toString()` definido en la clase `Circle` sobre el objeto creado.
4. Modificar el radio del `Circle` para que valga 30.0.
5. Ejecutar de nuevo el método `toString()`.
6. Crear un objeto de la clase `Blackboard` con título "Dibujo" y dimensión 500 x 500. Para que pueda verse el efecto del ítem 8, la pizarra creada no debe cerrarse.
7. Consultar los valores de los atributos del objeto creado.
8. Añadir el objeto `Circle` a la `Blackboard`.

4. Uso del evaluador de expresiones (*Code Pad*)

La *zona de código* (*Code Pad*) de *BlueJ* está situada en la esquina inferior derecha junto al banco de objetos (Figura 11). Si no se muestra, se debe seleccionar la opción `Mostrar Bloc de Código` del menú `Ver`.

En la línea especial situada en la parte inferior de esta zona se pueden introducir tanto una expresión como una instrucción Java, en la que pueden aparecer objetos del banco de objetos; tras pulsar *Enter*, lo escrito en esta línea será evaluado y se mostrará el valor resultante, seguido por su tipo (entre paréntesis), o un mensaje de error si la expresión/instrucción es incorrecta.

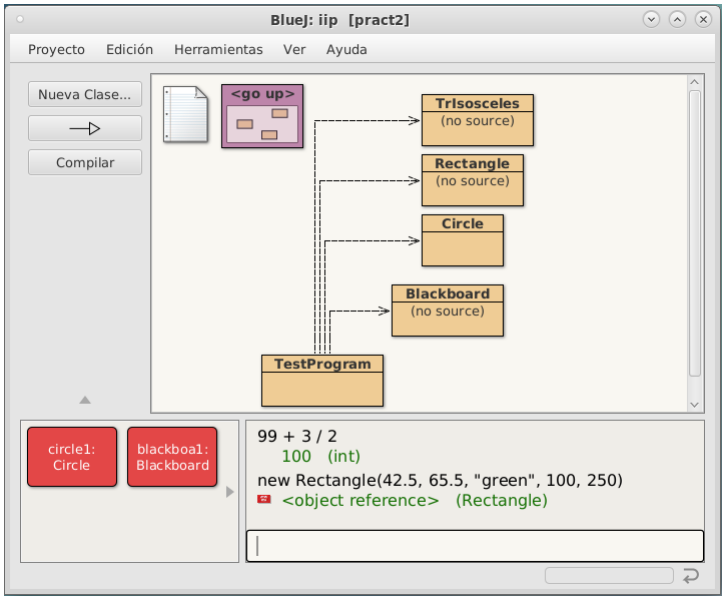


Figura 11: Zona de código de *BlueJ*

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. Dicho icono puede utilizarse ahora para continuar trabajando con el objeto resultante. Se puede arrastrar el icono al banco de objetos. Esto situará el objeto en el banco, donde estará disponible para futuras llamadas a sus métodos, bien vía su menú emergente o bien vía la zona de código.

Actividad #7

1. ¿Qué resultado se obtiene al evaluar cada una de las expresiones siguientes en la zona de código de *BlueJ*?

1	`8 % 3`	6	`9 / 2`
2	`(int) 98.67`	7	`9.0 / 2.0`
3	`Math.round(98.67)`	8	`9 / 2.0`
4	`Math.sqrt(121)`	9	`9 / (double) 2`
5	`Math.sqrt(-5)`	10	`9 / 0`

2. Definir las variables enteras x e y con valores 4 y 6, respectivamente y escribir una expresión aritmética para calcular la expresión algebraica siguiente:

$$\frac{x^2 - y}{x} \tag{1}$$

3. Definir las variables enteras a, b y c, con valores 2, -7 y 3, respectivamente y escribir una expresión aritmética para calcular la expresión algebraica siguiente:

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

4. Escribir en la zona de código una instrucción que muestre por pantalla (ventana de terminal de *BlueJ*) el radio de `circle1`, el `Circle` situado en el banco de objetos desde la Actividad #6.
5. Ejecutar en la zona de código el método `toString()` sobre `circle1` y arrastrar el icono del objeto `String` resultado al banco de objetos.
6. Crear un nuevo objeto `Circle` en la zona de código y arrastrar su icono al banco de objetos. Llamarlo `circle2`.
7. Escribir en la zona de código una instrucción que muestre por pantalla el color de `circle2`.
8. Ejecutar en la zona de código el método `toString()` sobre `circle2`. El icono del objeto `String` resultado se puede arrastrar también al banco de objetos.

5. Uso del depurador (*Debugger*)

El *depurador* de *BlueJ* es una herramienta sencilla, pero de gran utilidad a la hora de validar el funcionamiento de una clase (es decir, el de su `main`) o cualquier método de esta. Su uso permite, básicamente,

- observar la ejecución de cualquier método, es decir, *hacer su traza para unos valores dados*, bien paso a paso, bien de sólo alguna(s) de sus líneas.
- inspeccionar la secuencia de llamadas asociada a la invocación del método en ejecución.
- comprobar el paso de parámetros y los valores que toman las variables locales al método en ejecución.

Para conseguir estos resultados, el depurador dispone de las siguientes funciones:

1. **Establecer puntos de ruptura**. Sólo cuando se detiene la ejecución de un método en un cierto punto de su código resulta posible observar el estado de su ejecución en dicho punto. El depurador proporciona una función que detiene la ejecución de un método en un cierto punto del código, o equivalentemente, *establece puntos de ruptura.*

   En *BlueJ* los puntos de ruptura se establecen en la denominada *área de puntos de ruptura* del editor, situada a la izquierda del texto; basta con hacer "click" en ella, a la altura de la línea de código donde se desea detener la ejecución de un método, y aparecerá un pequeño signo de stop como marca de punto de ruptura. Cuando durante la ejecución de un método se alcanza la línea así marcada, la ejecución se interrumpe. Además, aparecen, una tras otra,

(a) *la ventana del editor*, en la que figura resaltada la línea siguiente a la que contiene el punto de ruptura, ya que es *la siguiente línea a ejecutar*;

(b) *la ventana del depurador*; los diferentes tipos de información y botones que contiene esta ventana se presentan a continuación.

2. **Ejecución paso a paso**. Una vez detenida la ejecución, esta se puede reanudar paso a paso, instrucción a instrucción, lo que permite seguir el código observando cómo progresa la ejecución; es decir, *hacer una traza* del código.

   Para realizar una ejecución paso a paso en *BlueJ* basta con hacer "click" repetidamente sobre el botón `Step` de la ventana del depurador. Cada "click" supone la ejecución de una única línea de código, tras lo cual la ejecución se vuelve a detener.

   Si se desea salir de este proceso, volver a la ejecución normal del método, basta con borrar la marca de ruptura establecida, simplemente haciendo "click" sobre ella, y después pulsar el botón `Continue` de la ventana del depurador.

3. **Inspección de variables y del paso de parámetros**.

   Solamente con observar la ventana de depuración de *BlueJ* se puede ver la secuencia de llamadas asociada a la invocación del método en ejecución, comprobar el paso de parámetros e inspeccionar los valores que toman sus variables locales.

   Para inspeccionar cualquier variable o parámetro de este tipo basta con hacer un doble "click" sobre él, en la ventana del depurador.

Actividad #8

1. En el método `main` de la clase `TestProgram`, establecer un punto de ruptura en las líneas en las que se crean los objetos de tipo `Circle`, `Rectangle` y `TrIsosceles`.

2. Ejecutar el método `main`. Obsérvese qué ocurre cuando, una vez alcanzado el punto de ruptura se hace "click" sobre el botón `Step` de la ventana del depurador.

3. Para inspeccionar las variables del `main`, hacer doble "click" sobre ellas en la ventana del depurador.

En la Figura 12 se muestra el depurador de *BlueJ* una vez alcanzado el último punto de ruptura en la ejecución de la clase `TestProgram`.

Actividad #9

Escribir una clase programa, similar a la clase `TestProgram`, que muestre una figura formada por círculos, rectángulos y triángulos.

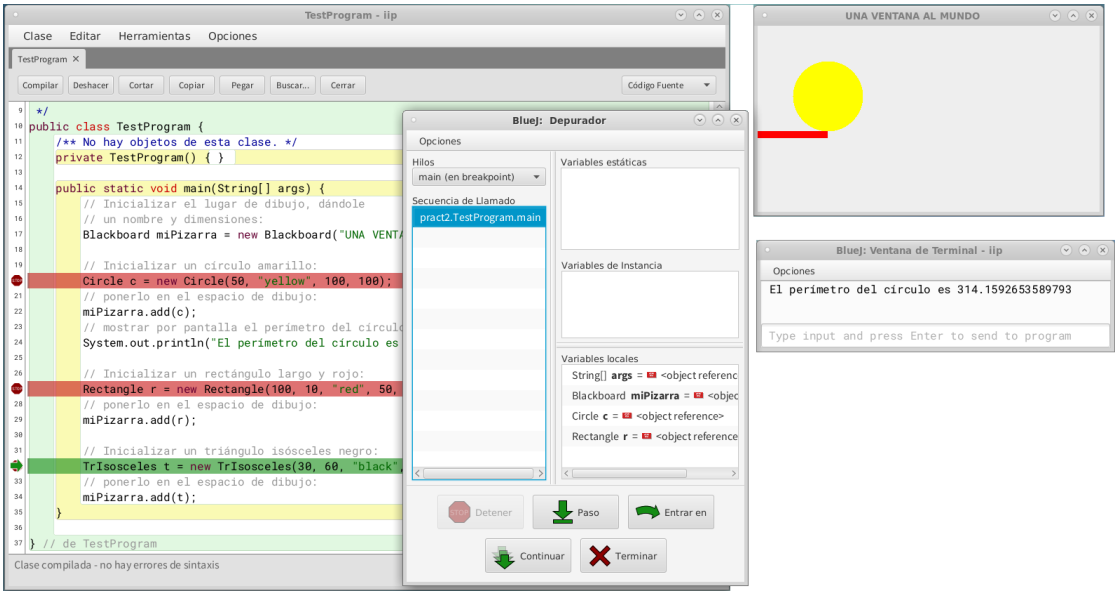


Figura 12: Depurador de *BlueJ*.