Programación I
Grado en Ingeniería Informática
Curso 2015/2016

BLUEJ - ENTORNO DE DESARROLLO JAVA
Inspección y depuración de código

Tabla de contenidos

1 Introducción

Esta práctica sirve como introducción al entorno de desarrollo para Java denominado BlueJ (http://www.bluej.org). Dicho entorno está indicado para quienes empiezan a programar utilizando Java como primer lenguaje y está especialmente diseñado para la enseñanza. Entre otras características, incluye una interfaz gráfica de la estructura de las clases implementadas y un depurador.

1.1 Objetivos

- Saber instalar BlueJ
- Crear proyectos en BlueJ
- Inspeccionar código en BlueJ
- Depurar código en BlueJ

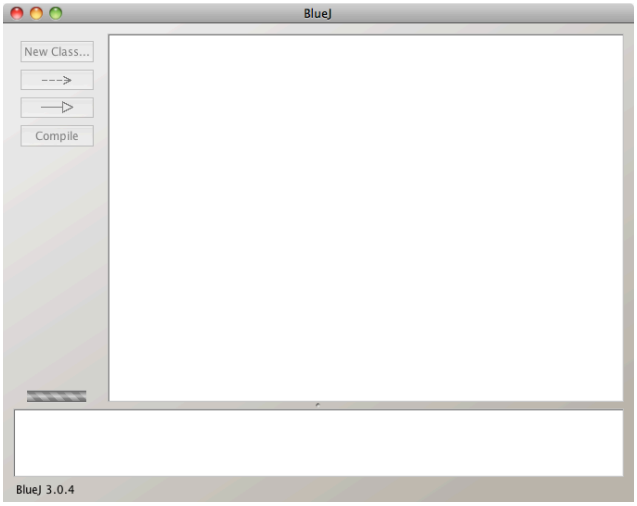
2 Introducción a BlueJ

2.1 Descarga e instalación de BlueJ

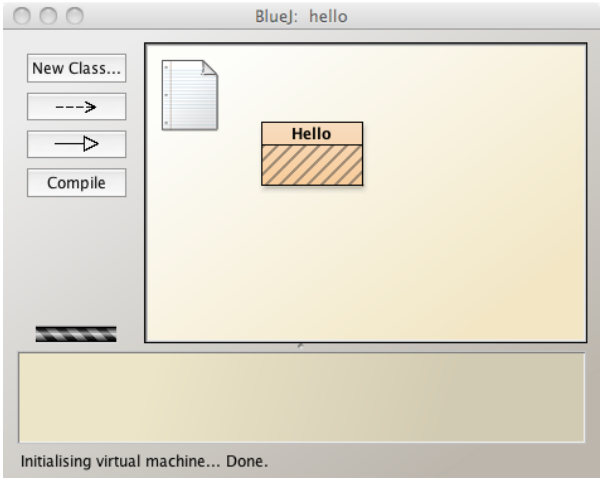
Desde la página principal de BlueJ (http://www.bluej.org), dentro del apartado *download*, existe versión para Windows, MacOS X, Debian y otros sistemas (instrucciones detalladas disponibles en la página de descarga).

2.2 Utilización de BlueJ – Proyectos

Dentro de BlueJ se trabaja con lo que se denomina proyectos. Un proyecto es un directorio que contiene los ficheros java que forman una determinada aplicación. Ejemplo: la instalación de Bluej incluye un directorio *examples* que a su vez incluye el proyecto denominado *Hello*.



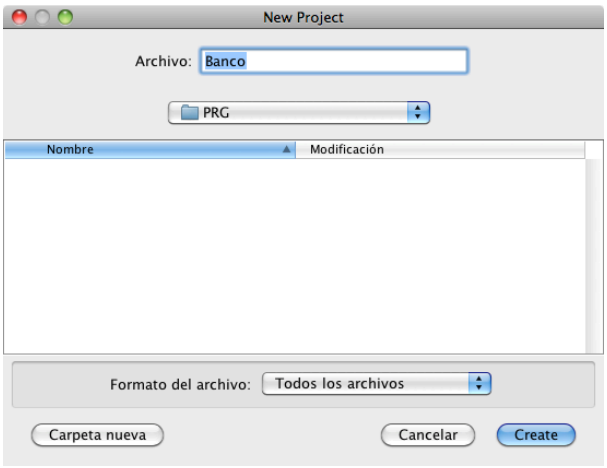
Para abrir un proyecto se utiliza la opción *Open Project* del menú *Project*. Ejemplo: proyecto *Hello* abierto en *BlueJ.*



Una vez abierto un proyecto ya creado se pasará a realizar las tres tareas básicas: codificar (editar), compilar y ejecutar.

2.3 Creación de proyectos en BlueJ

Para crear un proyecto nuevo se utilizará la opción *New Project* del menú *Project*. Crearemos el proyecto *Banco* dentro del directorio de trabajo.

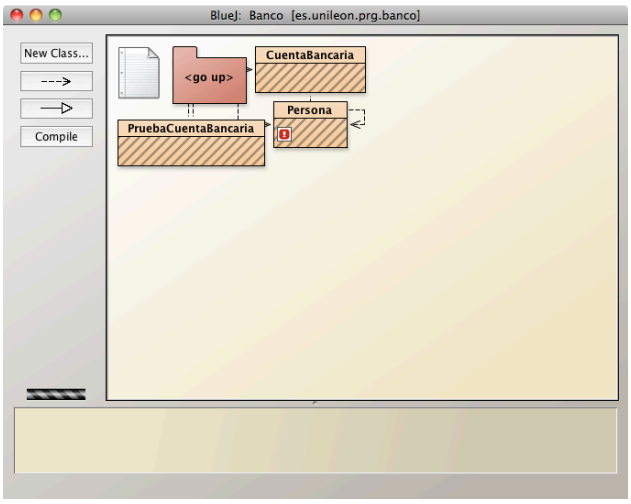


Una vez creado el proyecto incluiremos las clases `CuentaBancaria`, `Persona` y `MainCuentaBancaria` arrastrándolas al proyecto que acabamos de crear.

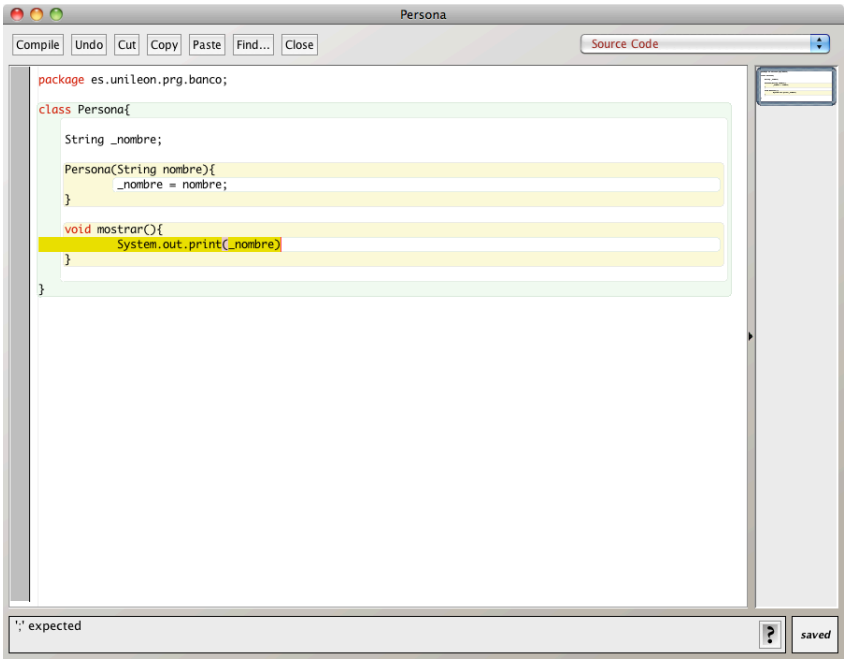
Al partir de código fuente ya creado, los siguientes pasos serán compilar y ejecutar.

2.3.1 Compilación de los ficheros fuente

Para la compilación se utiliza el botón Compile. Los posibles errores de la compilación aparecen señalados directamente sobre la línea de código que ha producido el error dentro de la ventana de edición. Ejemplo: eliminar el ';' final de una de las líneas de código y observar cómo se indica la clase en la que aparece el error



El resultado de la compilación se muestra la ventana que aparece en la siguiente figura. Como se puede apreciar, la línea que ha dado el error aparece sobreimpresa y el mensaje del error aparece en la parte inferior de la ventana.



2.3.2 Ejecución del *bytecode*

Para la ejecución del programa el entorno ofrece la posibilidad de crear objetos de la clase CuentaBancaria de forma gráfica, en lugar de utilizar el método *main* de la clase PruebaCuentaBancaria. Los pasos a seguir son los mismos que aparecerían dentro del *main*:

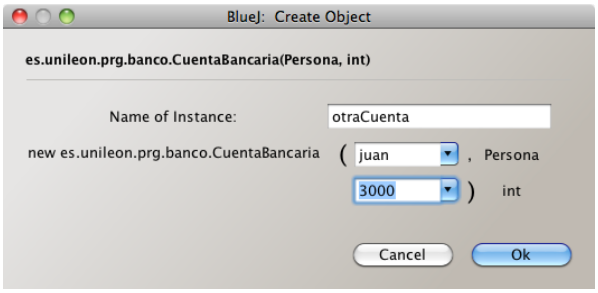
1. Declaración y creación de dos instancia de CuentaBancaria:

   En el *main*:

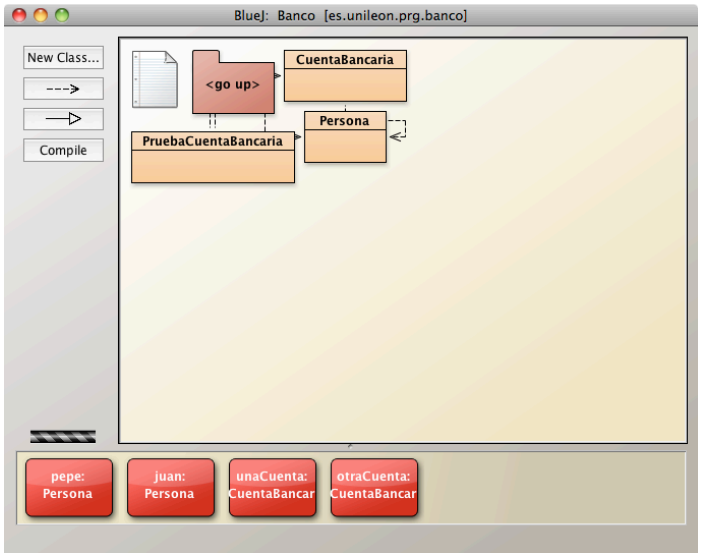
   ```
   CuentaBancaria unaCuenta, otraCuenta;
   unaCuenta = new CuentaBancaria(new Persona("Pepe"));
   otraCuenta = new CuentaBancaria(new Persona("Juan"), 3000);
   ```

   En BlueJ:

   a. Pulsar con el botón derecho del ratón sobre la clase CuentaBancaria
   b. Seleccionar un constructor
   c. Asignarle un nombre al objeto y un valor a los parámetros



Previamente debemos haber creado dos objetos de tipo Persona: juan y pepe.



2. Utilización de los objetos creados

Al crear el objeto `unaCuenta` aparece su representación dentro de BlueJ dentro del espacio reservado a los objetos, que es distinto del reservado a las clases. La utilización del objeto creado se lleva a cabo también desde el *main* y desde BlueJ de formas distintas:

En el main:

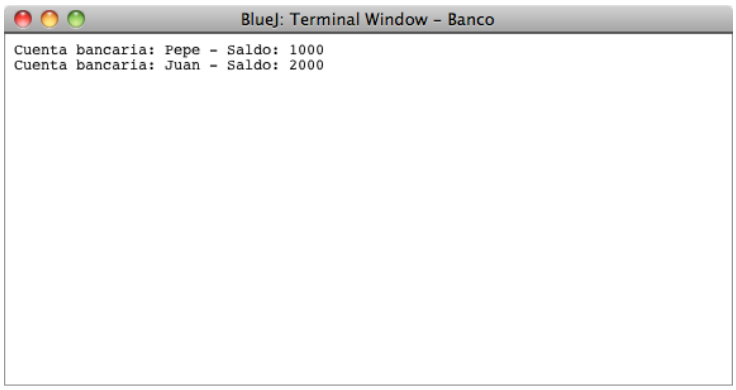
```
unaCuenta.ingresar(1000);
otraCuenta.sacar(1000);
unaCuenta.mostrar();
otraCuenta.mostrar();
```

En BlueJ:

a. Pulsar el botón derecho del ratón sobre el objeto miCuenta creado.

b. Seleccionar el método que se va a utilizar: ingresar, sacar o mostrar.

La salida que se produce es la que muestra la figura.



3 Depuración en BlueJ

La depuración o traza de un programa es la ejecución paso a paso de las instrucciones que se ejecutan, pudiendo así inspeccionar, en un instante determinado, el estado de los objetos que componen el programa.

La inspección del estado de un objeto es la observación de los valores que toman sus atributos en un instante determinado de la vida del objeto.

3.1 Primer ejemplo de depuración

Las funciones del depurador de BlueJ se reducen a tres:

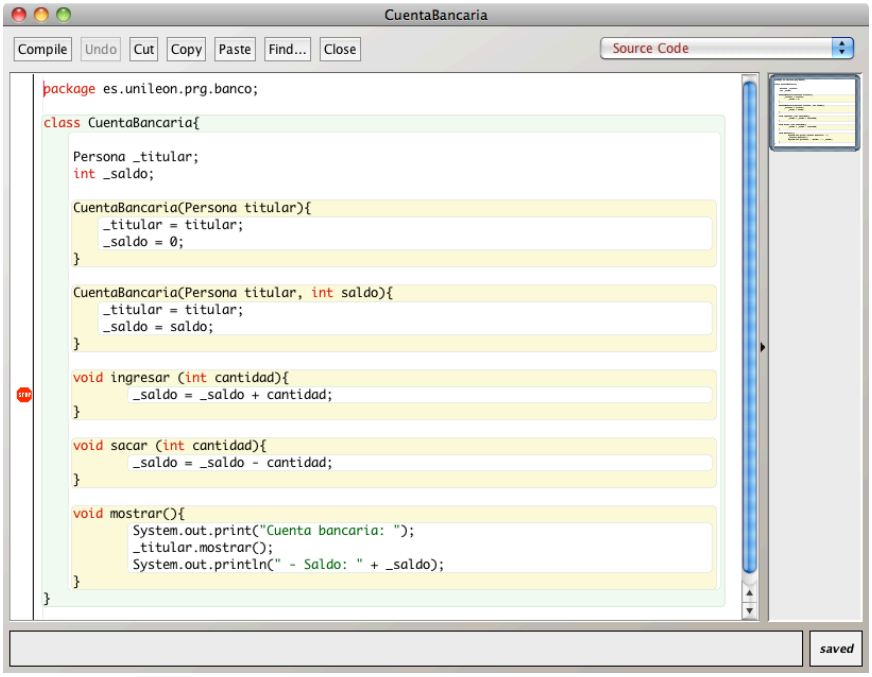
- Fijar *breakpoints* o puntos de ruptura
- Realizar una traza paso a paso del código
- Inspección de variables (atributos) de objetos

Para empezar con un ejemplo de depuración, hay que tener abierto el proyecto recién creado. Este proyecto contiene las clases del ejemplo seguido en las clases de teoría sobre las cuentas bancarias, y que ahora nos va a servir para demostrar las funcionalidades del depurador.

3.1.1 Fijar puntos de ruptura

Un punto de ruptura es una marca que se pone al lado de una instrucción, de forma que cuando el flujo de ejecución del programa pasa por allí, se detiene y se devuelve el control al usuario para que pueda inspeccionar el estado de los objetos en ese instante. Resulta muy útil para averiguar qué está pasando con los objetos, si están haciendo lo que se espera de ellos.

El área de puntos de ruptura está situada a la izquierda del texto, dentro del editor. Para definir un punto de ruptura, basta hacer click sobre ella. Entonces aparece una pequeña señal de stop sobre la instrucción en la que se detendrá la ejecución.

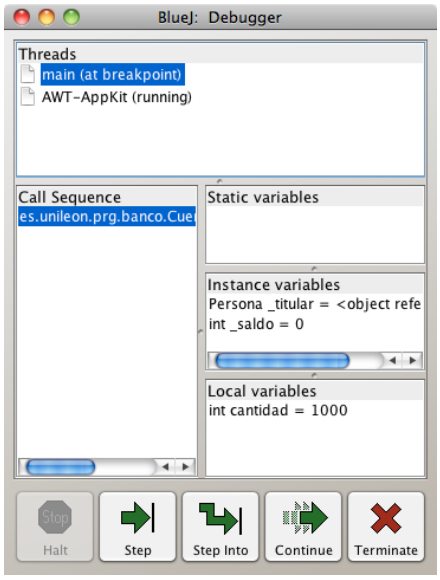



```
package es.unileon.prg.banco;

class CuentaBancaria{

    Persona _titular;
    int _saldo;

    CuentaBancaria(Persona titular){
        _titular = titular;
        _saldo = 0;
    }

    CuentaBancaria(Persona titular, int saldo){
        _titular = titular;
        _saldo = saldo;
    }

    void ingresar (int cantidad){
            _saldo = _saldo + cantidad;
    }

    void sacar (int cantidad){
            _saldo = _saldo - cantidad;
    }

    void mostrar(){
            System.out.print("Cuenta bancaria: ");
            _titular.mostrar();
            System.out.println(" - Saldo: " + _saldo);
    }
}
```

Antes de poder poner los puntos de ruptura, hay que compilar el proyecto. Para probar el punto de ruptura, abramos la clase CuentaBancaria y busquemos el método ingresar. Definamos un punto de ruptura en la única instrucción que aparece dentro del método.

Cuando la línea de código alcanzada durante la ejecución sea la que acabamos de marcar con un signo de stop, la ejecución se interrumpirá.

Para comprobarlo deberemos hacer una llamada al método ingresar de unaCuenta, con un valor de 1000 (euros, se supone) para el parámetro pedido. Tan pronto como se alcance el punto de ruptura, surgirá en pantalla la ventana del editor, mostrando la línea del código donde se ha detenido la ejecución, junto con la ventana del depurador (titulada *BlueJ: Debugger*), tal y como se muestra en la figura.



La línea resaltada del editor indica cuál es la siguiente instrucción que se va a ejecutar. La ejecución se habrá detenido justo antes de esta línea.

3.1.2 Traza del código

Ahora que hemos detenido la ejecución, podemos ir ejecutando paso a paso el código y ver cómo progresa la ejecución. Para hacer esto, haz click repetidamente sobre el botón *step* de la ventana del depurador. La línea resaltada en el código fuente del editor cambiará cada vez, según progrese la ejecución del programa. Cada vez que se pulse *step*, se ejecuta una sola línea de código y la ejecución se detiene de nuevo.

Fijémonos en cómo cambia el valor de la variable _saldo en la ventana del depurador, tras atravesar la instrucción que le suma la cantidad de 1000 especificada.

Podemos eliminar el punto de ruptura haciendo de nuevo click sobre él. Pulsemos luego el botón *Continue* para reanudar la ejecución normal.

Intentémoslo de nuevo con otro método. Definamos un punto de ruptura en el *main* de la clase PruebaCuentaBancaria, en la línea en la que se invoca al método ingresar por primera vez.

Si pulsamos *Step* se ejecutaría la línea entera y se saltaría a la siguiente. Pero si lo que queremos es ver qué hace por dentro el método, hay que pulsar *step into*. Si se pulsa *step into* en la llamada a un método, entonces el depurador entra dentro del código de dicho método y lo ejecuta línea a línea. Así hasta que se alcanza el final del método y se vuelve al punto desde el que se hizo la llamada (justo detrás del punto de ruptura). Probemos las dos opciones, *Step* y *Step into*, una detrás de otra.

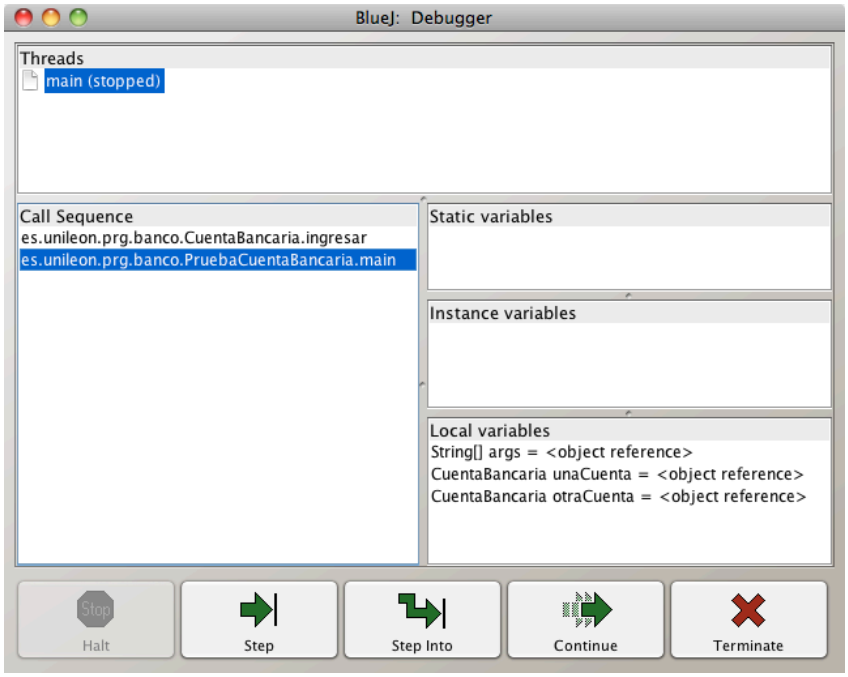
3.1.3 Inspección de variables

Cuando se está depurando el código, a veces resulta necesario observar el valor que van tomando las variables de los objetos creados. Para hacerlo, no hace falta ejecutar ningún comando especial. La ventana de depuración muestra y actualiza los valores de las variables de instancia del objeto que está siendo depurado, así como de las variables locales del método que se está ejecutando.

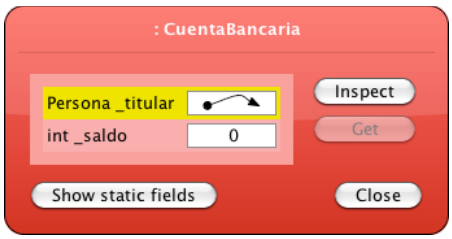
Aunque por defecto se muestran los valores de las variables locales del objeto en ejecución, también se pueden visualizar variables de otros objetos y métodos activos. Para comprobarlo, repitamos el proceso anterior de ejecución del main con el mismo punto de ruptura forzando a que la ejecución se detenga de nuevo en el punto de ruptura ya definido. Si hacemos un *Step into*, veremos que en la parte izquierda de la ventana del depurador, puede verse la secuencia de llamadas, que muestra:

```
es.unileon.prg.banco.CuentaBancaria.ingresar
es.unileon.prg.banco.PruebaCuentaBancaria.main
```

Esto significa que el método CuentaBancaria.ingresar fue llamado por el main de PruebaCuentaBancaria. Si seleccionamos CuentaBancaria.main en esta lista, se puede inspeccionar el código fuente y los valores actuales de las variables de este método.



Si avanzamos paso a paso más allá, podemos observar que el valor de la variable unaCuenta muestra algo así como <object reference>. Esto significa que el valor de esta variable es una referencia a un objeto. Todos los valores de tipo Objeto (excepto String) se muestran de esta forma. Para inspeccionar el objeto referenciado como valor de una variable, basta con hacer doble click sobre ella. Entonces aparecerá una ventana de inspección idéntica a las ya conocidas. No hay diferencias entre inspeccionar un objeto de esta forma o desde el entorno de trabajo donde habíamos creado las instancias.



3.1.4 Parada y terminación

A veces un programa no para nunca su ejecución. Puede ser que haya entrado en lo que se conoce como un bucle infinito, o puede ser que tarde mucho. El botón *Halt* sirve para interrumpir la ejecución justo como si se hubiera definido un punto de ruptura, pero sin terminar el programa. Entonces podemos seguir avanzando paso a paso en la traza de ejecución, observar las variables y comprobar que todo marcha bien. Si no queremos seguir con el programa se puede pulsar *Terminate* para dar por finalizada su ejecución.