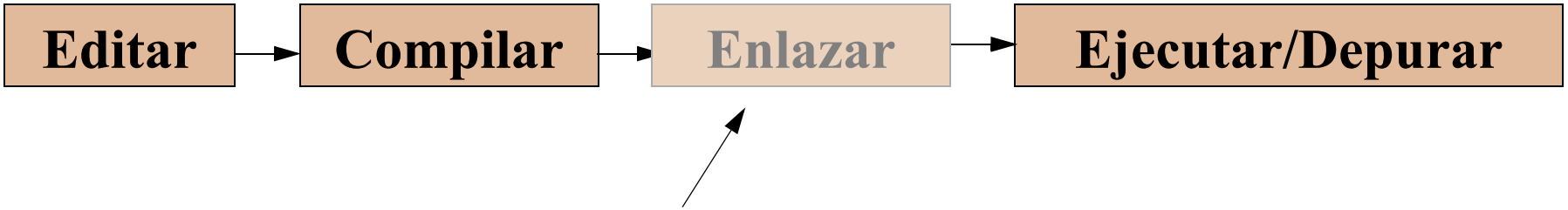
Bloque II. Herramientas

Capítulo 7. Uso de un entorno integrado de desarrollo de programas

- Proceso de desarrollo de programas
- El compilador y la ejecución
- Uso del entorno de desarrollo de programas ***bluej***
- La depuración
- Generación de documentos
- Empaquetamiento del programa

Proceso de desarrollo del programa

Proceso a seguir:



En Java usaremos la máquina virtual y *no* hacemos el proceso de *enlazar*

Se pueden usar diversos editores de texto

- en UNIX utilizar el "`gedit`", "`nano`" o "`emacs`"
  - para invocar el "***gedit***": `gedit Nombre.java`
- en Windows utilizar el `emacs`, el "*bloc de notas*" o el "*wordpad*"

El compilador y la ejecución

Utilizaremos el compilador Java SE JDK de Oracle

Para compilar desde la ***shell*** o intérprete de órdenes una clase almacenada en un fichero llamado `Nombre.java`:

```
javac Nombre.java
```

La compilación crea la clase ya compilada, en un fichero denominado `Nombre.class`

Para ejecutar una clase llamada `Nombre.class` con la máquina virtual:

```
java Nombre
```

Ejemplo

```
/**
 * Programa que pone un mensaje en la pantalla
 */

public class Hola {

   /**
    * Este es el método principal
    */
   public static void main(String[] args) {
      // No hay declaraciones
      System.out.println("Hola, Que tal estas?");
   }
}
```

Uso del entorno de desarrollo de programas Bluej

Un entorno integrado de desarrollo de programas suele presentar las siguientes características:

- interfaz gráfica con menús desplegables, cómoda de usar
- editor de texto orientado al lenguaje
- compilación desde el entorno
- salto automático al lugar donde ocurre un error de compilación
- depuración de alto nivel desde el entorno

Algunos entornos para Java suelen incorporar también herramientas de documentación y de generación de interfaces gráficas ("programación visual")

Gestión de proyectos

Para programar utilizando ***Bluej*** debe crearse un "***proyecto***" por programa.

El proyecto es un directorio en el disco donde se guarda toda la información que se va generando relacionada con el programa:

- código fuente
- clases compiladas
- documentación
- etc.

Creación de un nuevo proyecto

Elegir `Project=>New Project`, y luego situarse en el directorio deseado y darle al proyecto un nombre. Ej:

```
practica1
```

Añadir una clase nueva al proyecto. Para ello, pulsar el botón `New class`

- luego darle un nombre
- y elegir las opciones deseadas (`Class`, para una clase normal)

Para editar la clase, hacer "doble click" sobre ella

- borrar el código predeterminado que no necesitemos

Compilar y ejecutar el programa

Para compilar:

- desde el editor: botón `Compile`
- desde el proyecto: con el botón derecho hacer "click" sobre la clase y elegir `Compile`
- los errores de compilación aparecen en la parte inferior de la pantalla
- la línea con error queda resaltada

Para ejecutar desde la ventana del proyecto

- con el botón derecho hacer "click" sobre la clase
- elegir la ejecución del método `main`

Crear y usar objetos

Para crear un objeto de una clase, pulsar el botón derecho sobre ella y elegir la opción `new`

Para ver los atributos de un objeto pulsar el botón derecho sobre él y elegir la opción `inspect`

Para ejecutar un método de un objeto pulsar el botón derecho sobre él y elegir el método deseado

Si el programa se bloquea el ejecutarse, con el botón derecho pulsar sobre la barra de la máquina virtual (parte inferior izquierda de la ventana del proyecto) y elegir "`reset machine`"

Ejemplo: con la clase `Piedra` que se muestra a continuación:

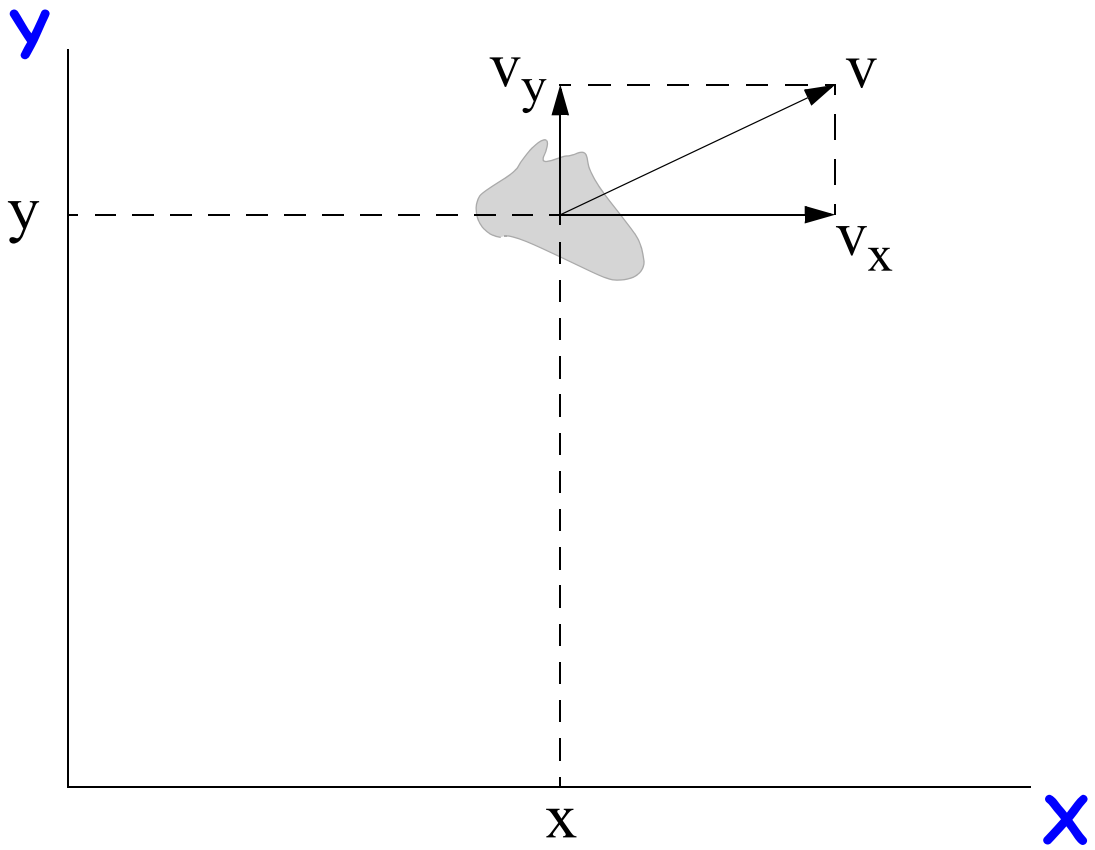
- crear con el ratón un par de objetos
- ejecutar sus métodos

Ejemplo: Simulación de la caída de una piedra

Representaremos la piedra con un objeto

Atributos:

- posición (x,y)
- velocidad (vx,vy)



Situación inicial y ecuaciones

Ecuaciones del movimiento, para calcular nueva posición y velocidad transcurrido un intervalo t

Variable	Cálculo usando valores viejos de las variables
x	$x + v_x \cdot t$
y	$y + v_y \cdot t - \frac{g \cdot t^2}{2}$
v_x	v_x
v_y	$v_y - g \cdot t$

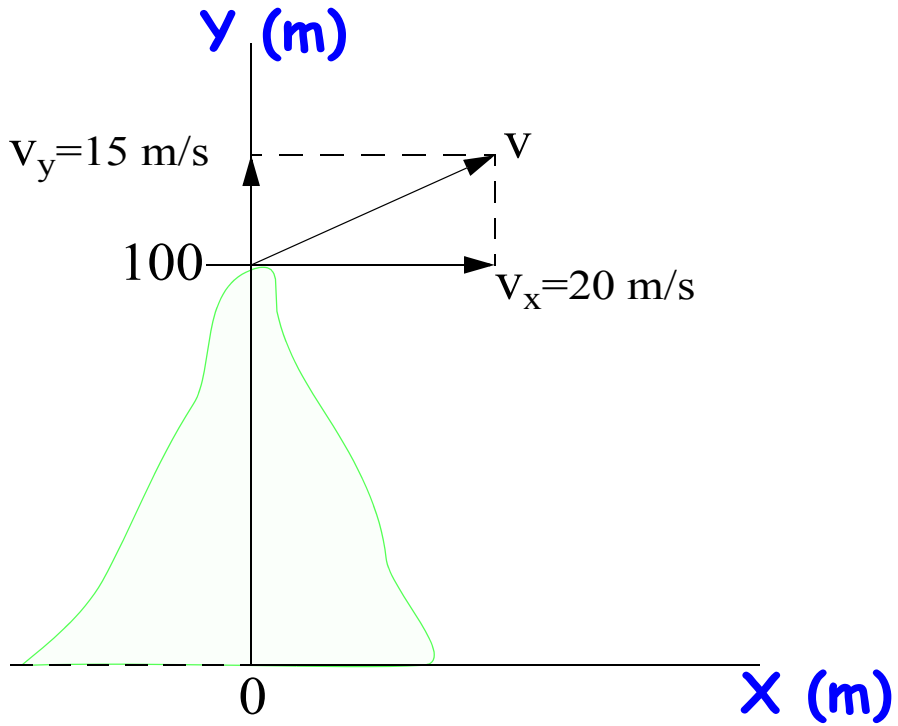
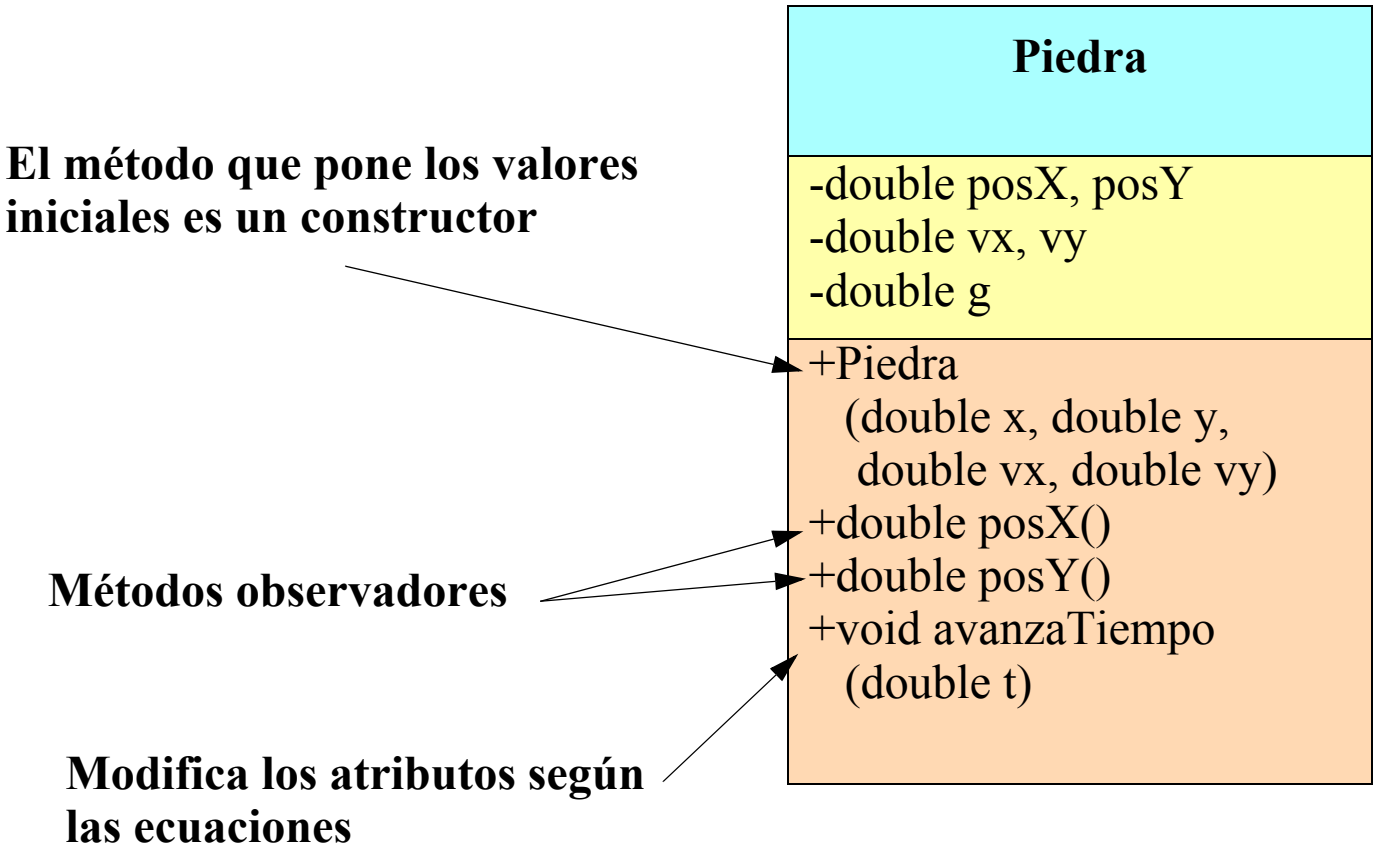


Diagrama de la clase



Código Java del ejemplo

```
/**
 * Esta clase simula la caída de una piedra
 */
public class Piedra
{
    // atributos que definen el movimiento
    private double posX, posY; // posición, metros
    private double velX, velY; // velocidad, m/s

    // constante que define la gravedad
    public final double g=9.8; // m/(seg*seg)
```

Código Java del ejemplo (cont.)

```
/**
 * Constructor que pone las condiciones
 * iniciales, copiándolas de los argumentos.
 * Unidades: m y m/s
 */
public Piedra
    (double posX, double posY,
     double velX, double velY)
{
    this.posX=posX;
    this.posY=posY;
    this.velX=velX;
    this.velY=velY;
}
```

Código Java del ejemplo (cont.)

```
/**
 * Retorna la posición X (m)
 */
public double posX() {
    return posX;
}

/**
 * Retorna la posición Y (m)
 */
public double posY() {
    return posY;
}
```

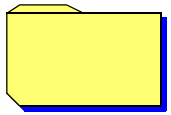
Código Java del ejemplo (cont.)

```
    /**
     * Avanza el tiempo en la cantidad t (s)
     */
    public void avanzaTiempo(double t)
    {
        posX=posX+velX*t;
        posY=posY+velY*t-g*t*t/2.0;
        // velX no cambia
        velY=velY-g*t;
    }
}
```

Uso de paquetes externos al proyecto, con ficheros *jar*

Para poder usar una clase definida en un fichero con extensión `.jar`:

- Añadir ese fichero directamente a las librerías java



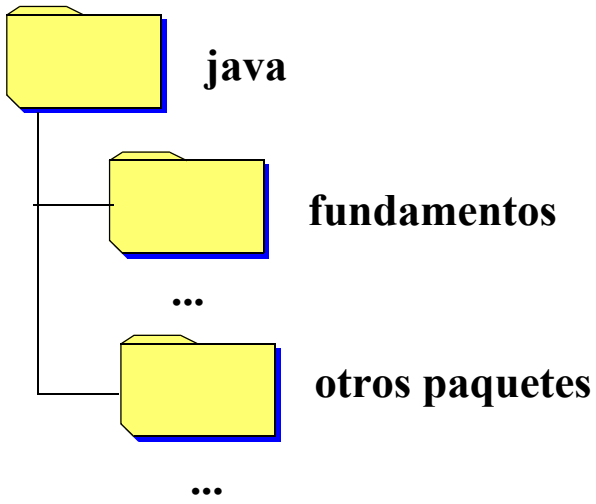
fundamentos.jar

  - `Tools` => `Preferences`
  - Elegir la pestaña "`Libraries`"
  - Elegir la opción "`Add...`" en "`User Libraries`"
  - Pinchar en el fichero jar (`fundamentos.jar`)
  - Pulsar `Abrir -> Aceptar -> OK`

- Acordarse de importar las clases a usar en el programa:
  - `import fundamentos.Lectura;`
  - `import fundamentos.*;`

Uso de paquetes externos al proyecto con directorios que contienen clases

Cuando el paquete es un directorio con clases situar este directorio en otro para los paquetes; por ejemplo, en el directorio `java`



- Añadir el directorio donde están los paquetes (`java` en este caso) a la lista de librerías
  - *no añadir el paquete directamente* (*no* poner `java/fundamentos`)

Ejemplo de uso del paquete fundamentos

Añadir a la clase `Piedra` un método que lee valores iniciales del teclado

```
import fundamentos.*;
public class Piedra {
   ...
   /**
    * Lee de una ventana la posición y velocidad
    *  iniciales
    */
   public void leeValoresIniciales() {
     Lectura l = new Lectura ("Datos iniciales");
     l.creaEntrada("posición X (m)",posX);
     l.creaEntrada("posición Y (m)",posY);
     l.creaEntrada("velocidad X (m/s)",velX);
     l.creaEntrada("velocidad Y (m/s)",velY);
```

Ejemplo

```
        l.esperaYCierra();
        posX=l.leeDouble("posición X (m)");
        posY=l.leeDouble("posición Y (m)");
        velX=l.leeDouble("velocidad X (m/s)");
        velY=l.leeDouble("velocidad Y (m/s)");
    }
}
```

La depuración

Es el proceso de prueba del programa para localizar errores

La depuración se puede realizar:

- manualmente: insertando instrucciones de salida (`println`) que muestren el flujo de control del programa y el valor de las variables de interés
- mediante un depurador de alto nivel

El depurador de alto nivel permite:

- parar el programa en los puntos deseados (***breakpoints***)
- ejecutar paso a paso
- visualizar el contenido de las variables (***watches***)
- visualizar llamadas a métodos y sus argumentos

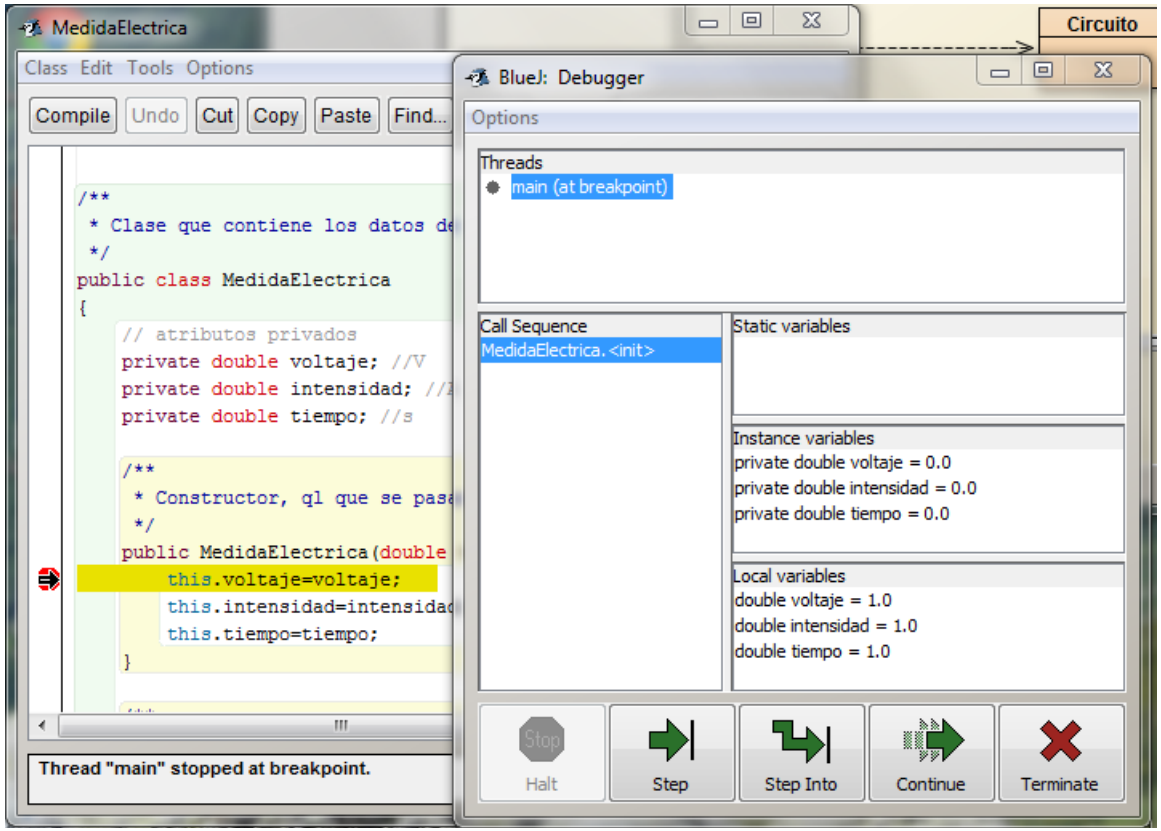
Depurador del Bluej

Es un depurador para programas Java con interfaz gráfica

Permite introducir puntos de ruptura (breakpoints)

- "click" en la columna a la izquierda del código (aparece un STOP)

También permite controlar la ejecución, y visualizar información en la ventana de depuración

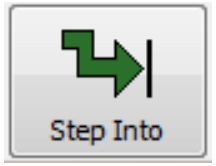


Depuración:
Control de la ejecución

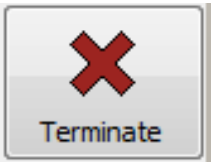
La ventana de depuración se abre poniendo un punto de ruptura y ejecutando el programa o un método

El programa se detiene en un punto de ruptura

El depurador tiene botones para:

	Ejecutar paso a paso (entrando en métodos)
	Ejecutar paso a paso (saltando por encima de métodos)

Depuración:
Control de la ejecución (cont.)

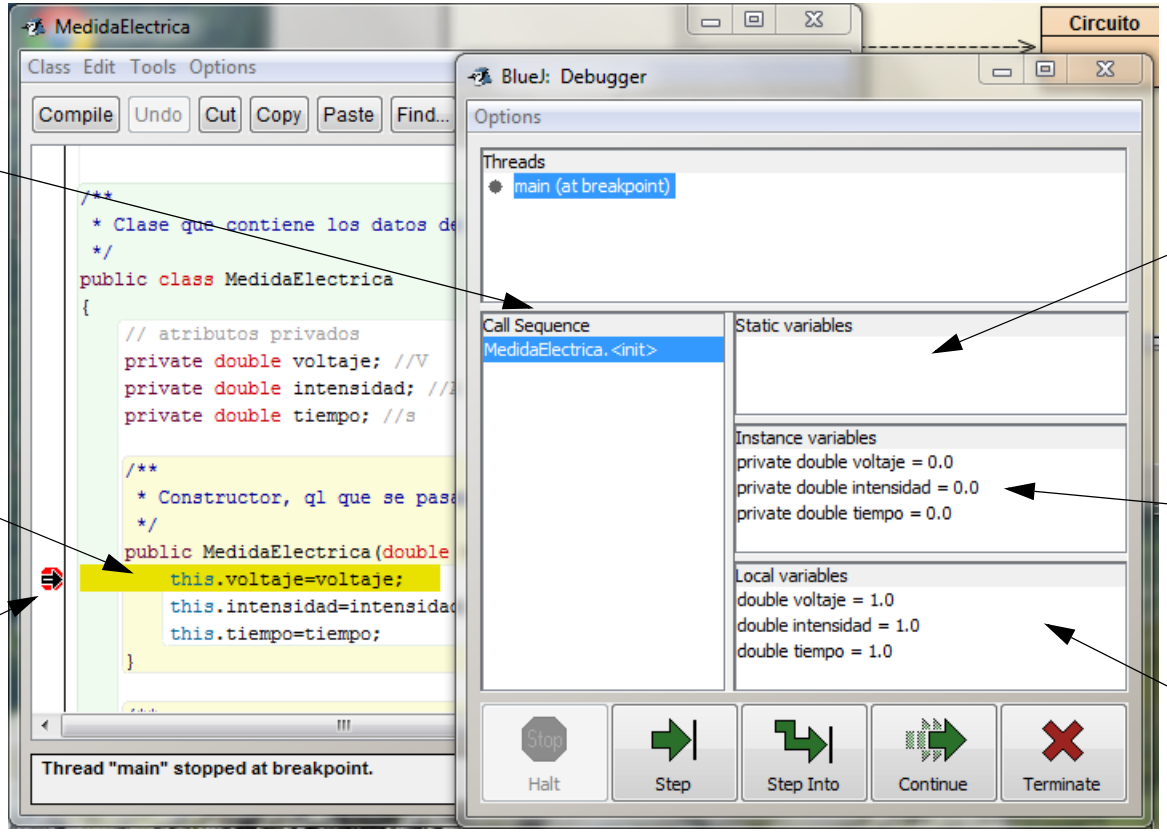
	Continuar la ejecución
	Hacer una pausa en la ejecución `Stop` (útil para lazos largos o infinitos)
	Finalizar la ejecución

Depuración: Visualizar información

Secuencias de llamadas a métodos

Línea actual

Puntos de ruptura



Atributos estáticos

Atributos normales

Variables locales

Generación de documentos

Para usar una clase, lo único que se necesita conocer de ella es la interfaz pública:

- atributos: sus nombres y tipos, y descripción
- métodos: sus cabeceras y descripción de lo que hacen

Se puede extraer esta información de manera automática, por medio de herramientas de documentación.

Elegir en la ventana del proyecto:

- `Tools => Project documentation`

Empaquetamiento del programa

Hay un formato para guardar de forma comprimida en un solo fichero todas las clases y recursos que necesite un programa

- formato `jar`

Conviene para ejecutarlo que el programa haga toda la entrada/salida con ventanas (no usar `System.out.println`)

Se genera desde el ***bluej*** con la opción

- `project => create jar file`

Se ejecuta:

- Linux: `java -jar nombre_fichero.jar`
- Windows: doble click sobre el icono del fichero