

BlueJ 3.0.0

Tutorial Academico

Parte II



Programacion IV-Paradigma Orientado a objetos
Fa.C.E.N.A-U.N.N.E

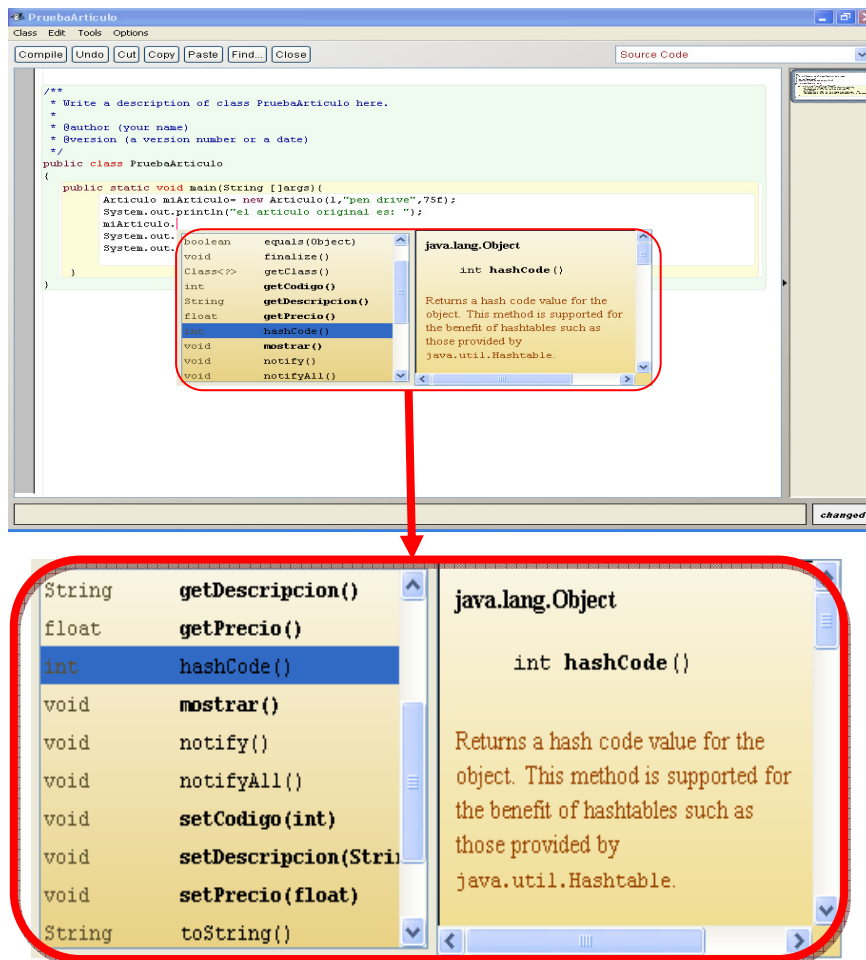
Tabla de contenido

1	Autocompletado de Código	2
2	Añadir clases existentes al proyecto.....	2
3	Generar un JAR en BlueJ	3
4	Librerías	5
4.1	Trabajando con librerías estándar de Java	5
4.2	Librerías de Usuario	6
5	Creación de objetos	8
6	Debugging	9
6.1	Estableciendo Breakpoints (puntos de ruptura)	9
6.2	Avanzando paso a paso por el código	10
7	Aprendiendo a usar Paquetes.....	12
7.1	Creación de Paquetes	12
7.2	Uso de paquetes.....	15
8	Información Adicional en Windows	17
8.1	Configuración del idioma:.....	17

1 Autocompletado de Código

Una de las características más importantes de BlueJ 3.0.0 con respecto a las versiones anteriores es el autocompletado de código. Esta es una característica estándar en otros entornos de java como Eclipse o Netbeans. A diferencia de estos entornos, el panel de autocompletado de código no aparece de forma automática, sino que debe ser activado explícitamente (Ctrl+Espacio, por default).

Una vez activado el autocompletado, se habilitará una ventana de ayuda de consulta, lo que significa poder consultar la documentación de manera ágil.



A la izquierda de esta imagen vemos el autocompletado de código y las posibilidades para completar el código.

A la derecha vemos una ventana adicional de ayuda, que servirá de consulta.

2 Añadir clases existentes al proyecto

Nota: Se pueden copiar clases a un proyecto desde fuera de él utilizando la opción Add Class from File....

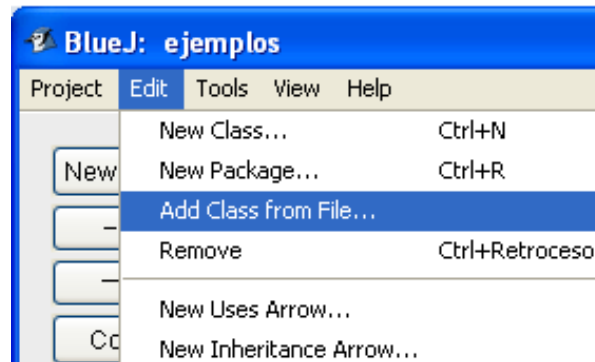
A menudo se desea utilizar en un proyecto BlueJ una clase que se ha obtenido de algún sitio o desarrollado en algún proyecto anterior.

Por ejemplo, un profesor puede entregar una clase Java para que los estudiantes la usen en un proyecto.

Se puede incorporar fácilmente una clase existente en algún proyecto determinando. Para ello acceder a la opción *Edit - Add Class From File* en el menú. Esto permitirá seleccionar un archivo fuente Java (con nombre terminado en .java) para ser importado.

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

La próxima vez que se abra el proyecto, la clase será incluida en el diagrama del proyecto.



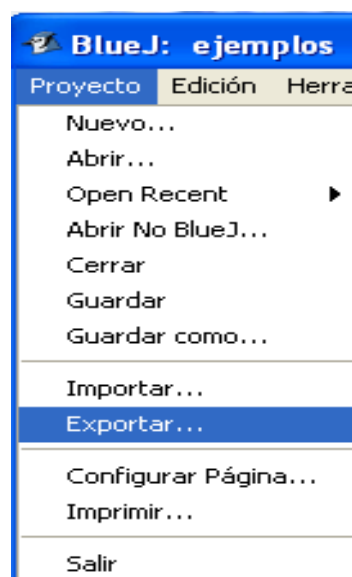
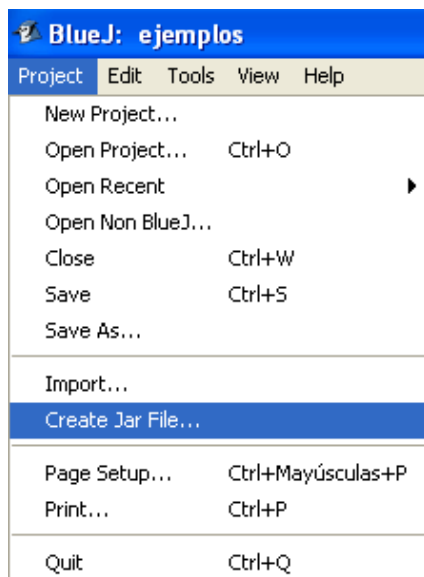
3 Generar un JAR en BlueJ

BlueJ también permite crear ficheros ejecutables JAR. Estos archivos pueden ser ejecutados haciendo doble click en el archivo (Windows), o mediante el comando `java -jar <nombre del archivo>.jar` (DOS o Símbolo de Sistema).

Probaremos con nuestro proyecto "ejemplos". Abrimos el proyecto compilado, vamos al menú *Project* y seleccionamos la función *Create Jar File...*

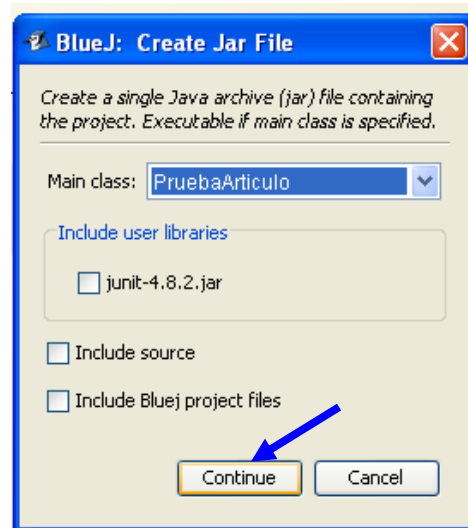
Nota: En caso de tener BlueJ configurado en español, debemos ir al menú *Proyecto* y seleccionamos la opción *Exportar...*

Se abre un cuadro de diálogo que almacena una copia del proyecto actual en formato Java Archive (JAR). Para crear un fichero jar se debe especificar una clase principal; esta clase debe tener definido un método *main* válido (con el prototipo `public static void main(String[] args)`).

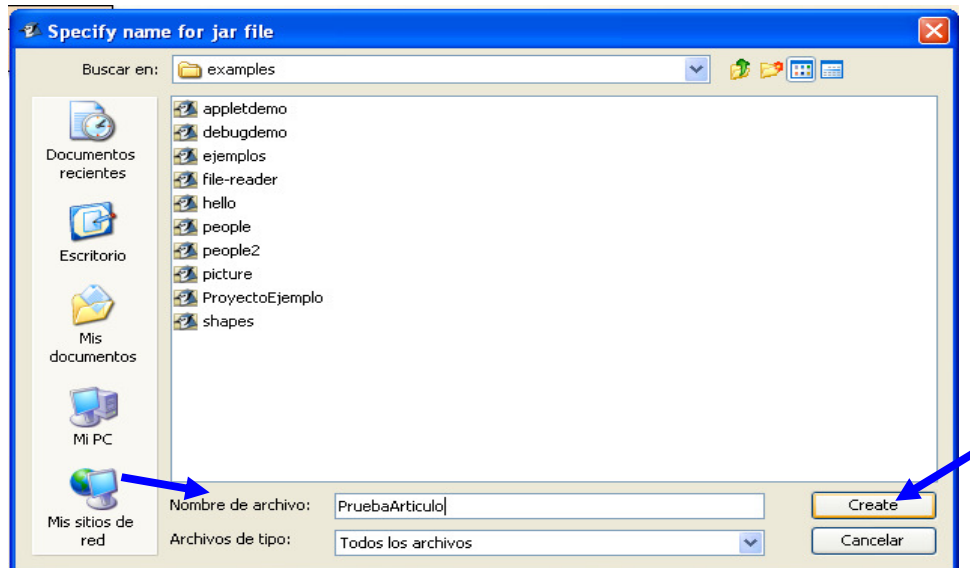


En el ejemplo utilizado previamente, se selecciona la clase principal *PruebaArticulo* en el menú emergente. Si se tiene otra clase que contenga el método “main” y queremos ejecutarlo, elegimos dicha clase del proyecto.

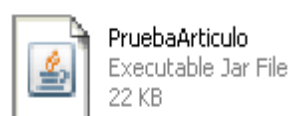
Si se quiere, se puede incluir los archivos fuentes (*Include source-Include BlueJ project files*) en el archivo Jar (se puede usar el formato JAR por ejemplo para enviar el proyecto completo a alguien vía e-mail en un único archivo).



Del menú de clases principales seleccionar la clase *PruebaArticulo* y hacer click en *Continue*. A continuación se verá un cuadro de diálogo de selección que permite especificar algún nombre para el archivo JAR a crear. Pulsar *Create*.

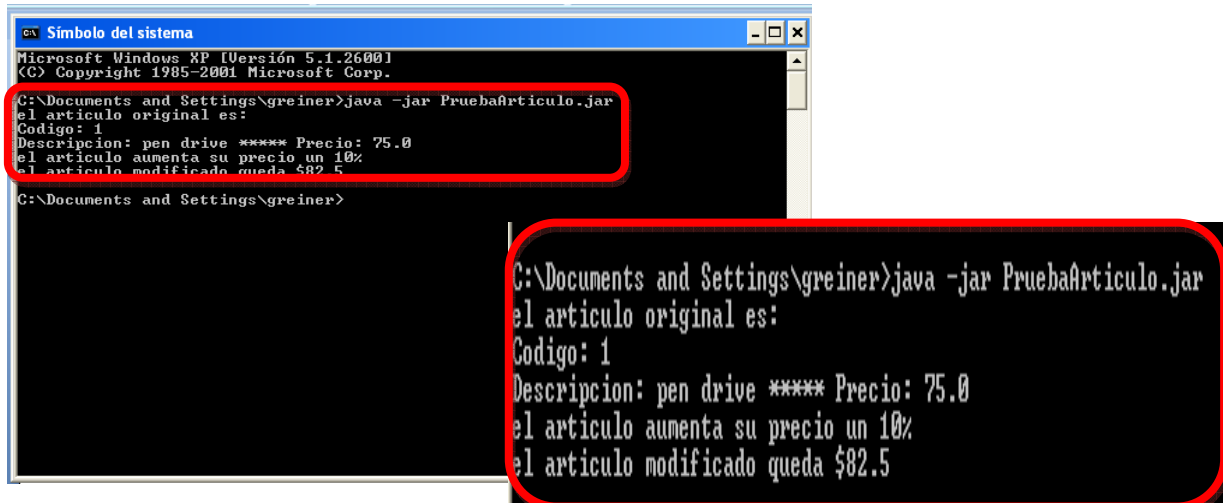


El archivo JAR creado genera una imagen como la siguiente figura:



Para ejecutar un archivo JAR se puede hacer doble click sólo si la aplicación tiene una interfaz gráfica (GUI) definida. Como nuestro ejemplo usa entrada/salida de texto (en forma de constantes), lo que debemos hacer es arrancarlo desde un terminal de texto. Abrir un terminal o una ventana de Símbolo del sistema (DOS). Acceder al directorio donde se guardó el archivo JAR (se debe ver el archivo *PruebaArticulo.jar*). Asumiendo que Java está instalado correctamente, se debe escribir:

```
java -jar PruebaArticulo.jar
```



```
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

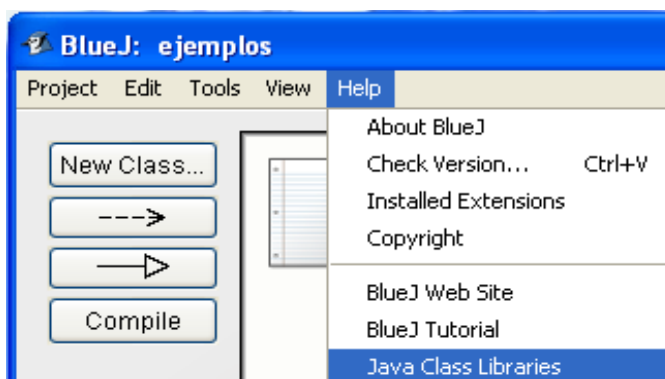
C:\Documents and Settings\greiner>java -jar PruebaArticulo.jar
el articulo original es:
Codigo: 1
Descripcion: pen drive ***** Precio: 75.0
el articulo aumenta su precio un 10%
el articulo modificado queda $82.5

C:\Documents and Settings\greiner>
```

4 Librerías

4.1 Trabajando con librerías estándar de Java

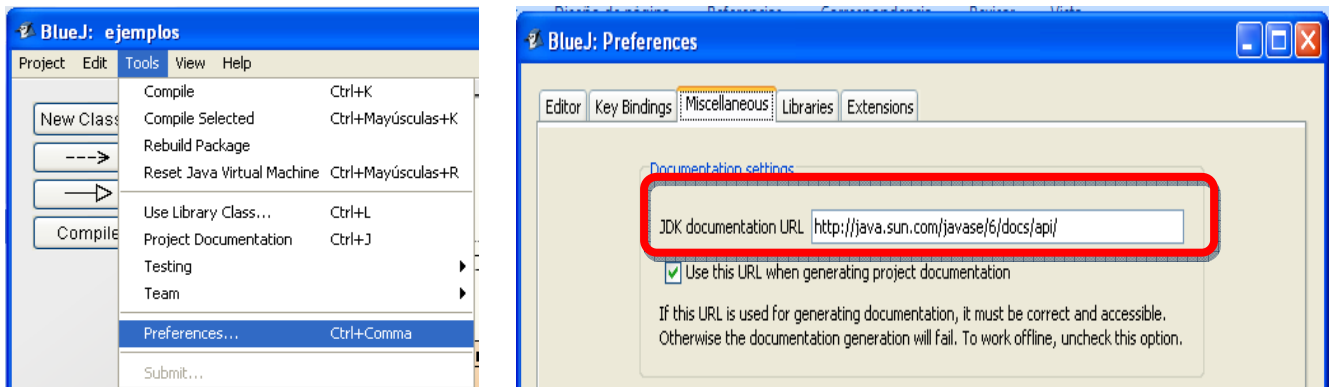
Frecuentemente, cuando se escribe un programa en Java, se necesita hacer referencia a las librerías estándar de Java. Se puede abrir un navegador mostrando la documentación de la API de JDK. Una forma de consulta, si uno se encuentra conectado a la red (online), es seleccionando *Help - Java Class Libraries* en el menú.



Una vez seleccionado la opción *Java Class Libraries*, nos redireccionará hacia el siguiente URL: <http://java.sun.com/javase/6/docs/api/>

Nota: Para acceder a la documentación en forma online debemos seleccionar del menú principal *Tools-Preferences...*

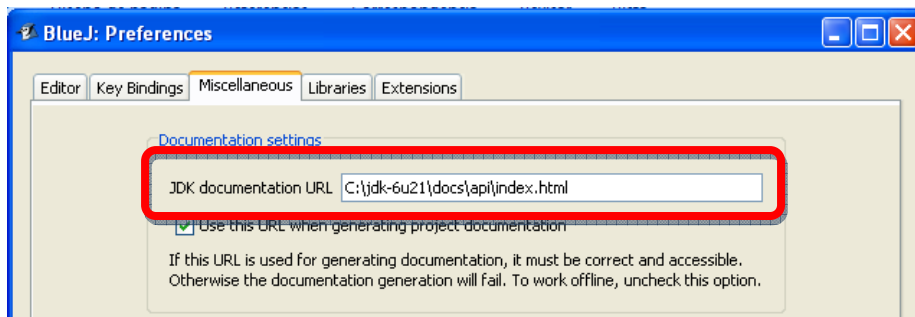
Se abrirá una ventana. Seleccionar la pestaña *Miscellaneous* y en el campo etiquetado como *JDK documentation URL* escribir <http://java.sun.com/javase/6/docs/api/> y hacer click en *Ok*.



También se puede instalar y utilizar la documentación del JDK localmente (offline). Para ello debemos disponer de una copia local de la documentación. Si no es así, se la puede descargar de <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

En el menú de la derecha acceder a *Java SDKs and Tools* y elegir la opción *Java SE* (hacemos click). Se abre otra ventana, acceder a **Java SE 6 Documentation**, click en *Download* y seleccionar el lenguaje inglés, *continue* y empieza a descargar.

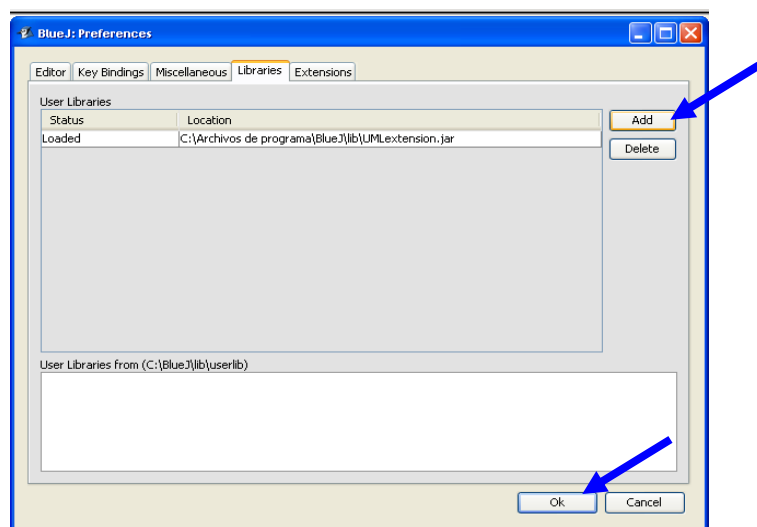
Luego de la descarga dirigirse a *Tools-Preferences*, seleccionar *Miscellaneous*. Y en el campo etiquetado como *JDK documentation URL* colocar la dirección URL de la copia local de la documentación (por ejemplo: en una instalación para Windows C:\jdk-6u21\docs\api\index.html), y hacer click en *Ok*.



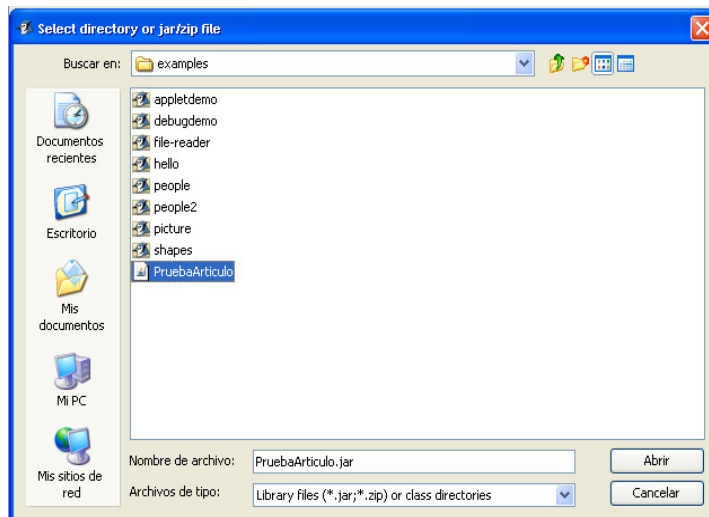
4.2 Librerías de Usuario

Si en el proyecto actual se desea utilizar además de las librerías predefinidas de Java, otras que el proyecto necesite, BlueJ brinda la posibilidad de incluir o agregar archivos JAR ya generados, e incluso otros proyectos creados, y utilizarlos como librerías de usuario.

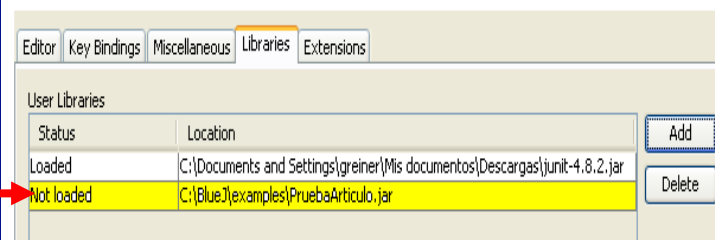
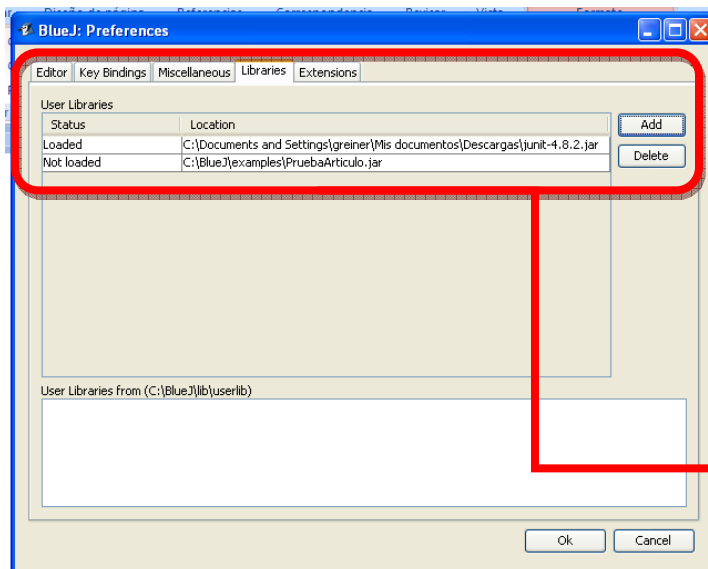
Volviendo al ejemplo anterior; si se desea utilizar el archivo *PruebaArticulo.jar* (creado previamente) como librería de usuario, se deben realizar los siguientes pasos:



Paso 1: Nos dirigimos a *Tools-Preferences-Libraries-Add* para agregar el proyecto, archivo .Zip o .Jar creado

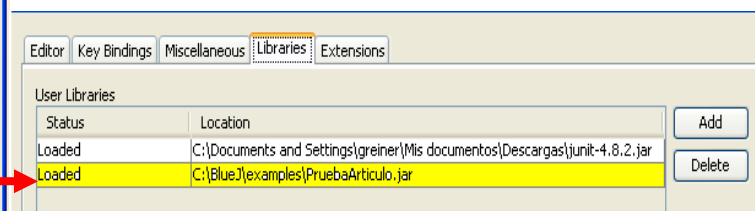
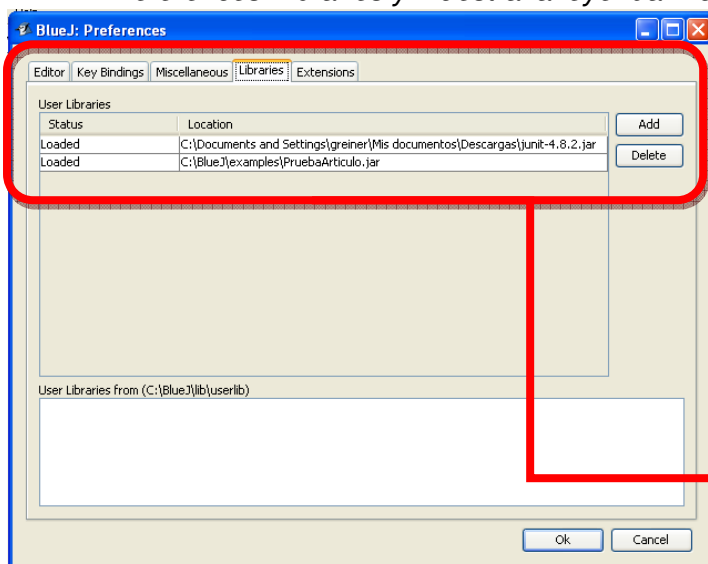


Paso 2: Buscar y seleccionar los proyectos ó archivos .Jar o .Zip que se desea agregar al proyecto. En este caso PruebaArticulo.Jar



Paso 3: Una vez cargada la librería, verificar en Status la leyenda “Not loaded” (no cargado). Es porque aun no estaba cargada esa librería. Hacer click en Ok para cargarla.

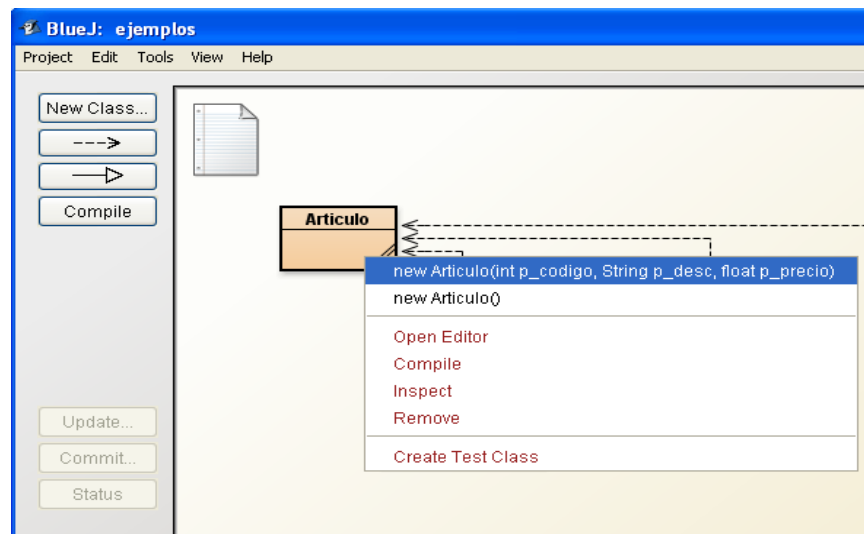
Nota: Se debe reiniciar el entorno. Para revisar si la librería esta instalada dirigirse a Tools-Preferences-Libraries y muestra la leyenda “Loaded” (cargado). Ya se puede usar la librería.



5 Creación de objetos

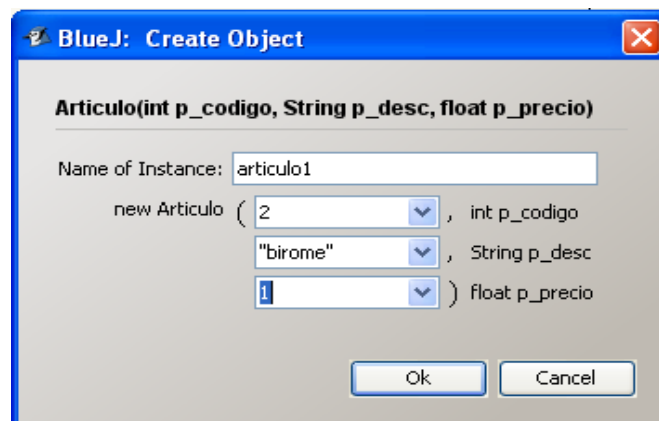
Una de las características fundamentales de BlueJ es que no sólo se puede ejecutar una aplicación completa, sino que también se puede interactuar con objetos aislados de cualquier clase y ejecutar sus métodos públicos. Una ejecución en BlueJ se realiza habitualmente creando un objeto e invocando a uno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

Ejemplo: Se desea crear un objeto *articulo*, por lo tanto se debe hacer "click-derecho" en el icono de la clase *Articulo*. El menú muestra los constructores definidos para crear un objeto. Seleccionar el constructor con parámetros:

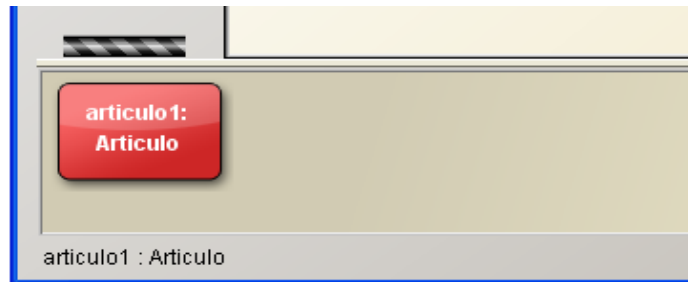


Luego aparecerá una ventana emergente donde se deben ingresar los datos de acuerdo al orden definido en el constructor.

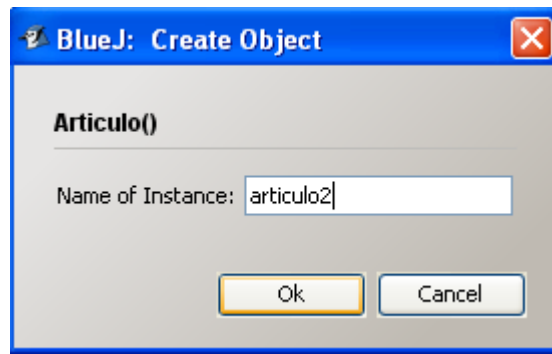
Nota: El nombre del objeto creado (instancia de la clase) que por defecto es *articulo1* se puede cambiar en *Name of Instance*.



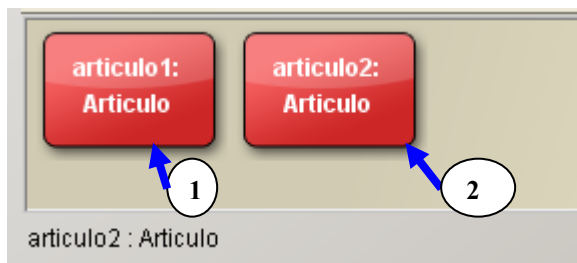
Al presionar en *OK* mostrará en la parte inferior izquierda del editor el objeto creado. Una vez que se ha creado el objeto, es situado en el banco de objetos.



La segunda opción es crear un objeto Articulo sin parámetros. El nombre asignado por defecto puede ser cambiado. Presionar Ok y mostrará en el Banco de Objetos el objeto articulo2.



Como se observa en la ventana anterior, no existe la opción de ingresar los datos correspondientes a los atributos de la clase Articulo, por haber elegido el constructor por defecto (sin parámetros).



Nota:

- 1- articulo1, es el objeto Articulo creado con parámetros.
- 2- articulo2, objeto Articulo creado sin parámetros

6 Debugging

BlueJ 3.0.0 al igual que en las versiones anteriores, trae la funcionalidad del depurador o debugging.

Se deben realizar las siguientes tareas:

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

6.1 Estableciendo Breakpoints (puntos de ruptura)

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

El área de puntos de ruptura está en el editor a la izquierda del texto. Puede establecer un punto de ruptura pulsando en ella. Aparecerá un pequeño signo de stop para marcar el punto de ruptura.

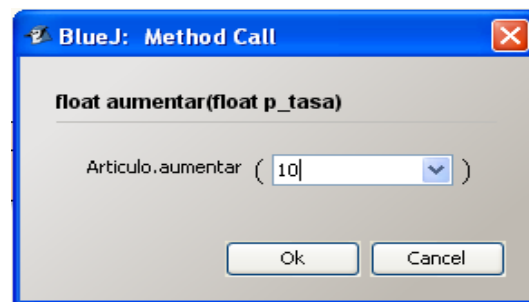
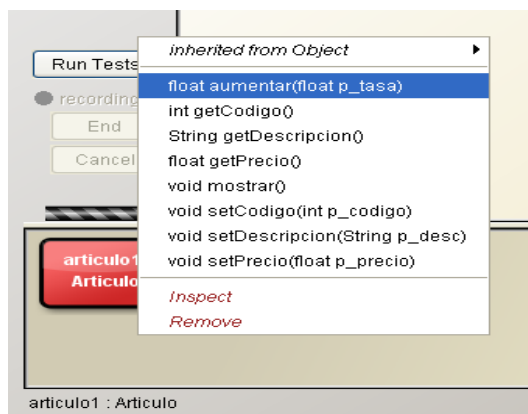
```

29 public float getPrecio(){
40     return this.precio;
41 }
42
43 public float aumentar(float p_tasa){
44     float nuevo_precio=this.precio+(this.precio*(p_tasa/100));
45     return nuevo_precio;
46 }
47
48 public void mostrar(){
49     System.out.println("Codigo: "+this.getCodigo());
50     System.out.println("Descripcion: "+this.getDescripcion()+" **");
51 }
52 }
53

```

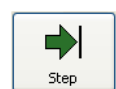
Cuando se alcanza la línea de código que tiene asociado el punto de ruptura, la ejecución se interrumpirá.

Ejemplo: crear un objeto de la clase *Articulo* y llamar al método *incrementar* con el parámetro 10 (que equivale al porcentaje aumentado). Tan pronto como se alcanza el punto de ruptura, emerge una ventana del editor, mostrando la línea de código actual y emerge una ventana de depurador.



6.2 Avanzando paso a paso por el código

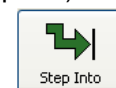
Una vez que la ejecución se ha detenido, se puede avanzar paso a paso por el código y ver como progresa la ejecución. Para hacer esto, pulsar varias veces en el botón *Step* en la ventana del depurador.



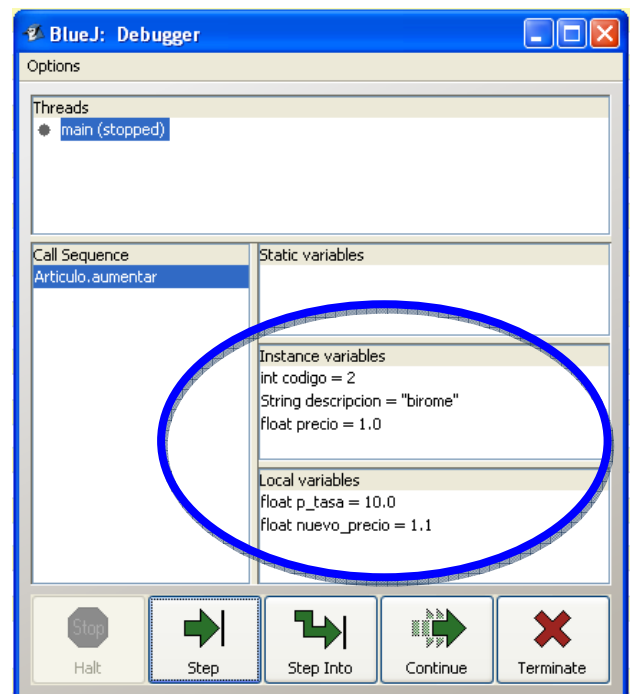
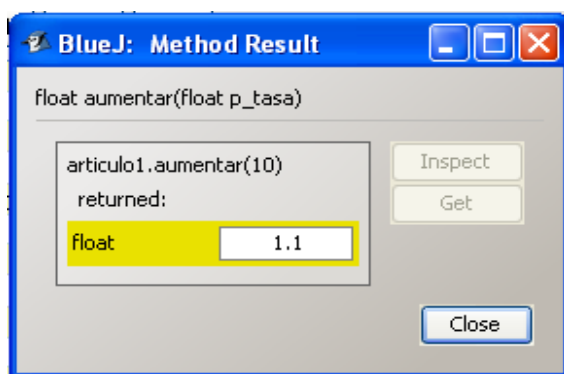
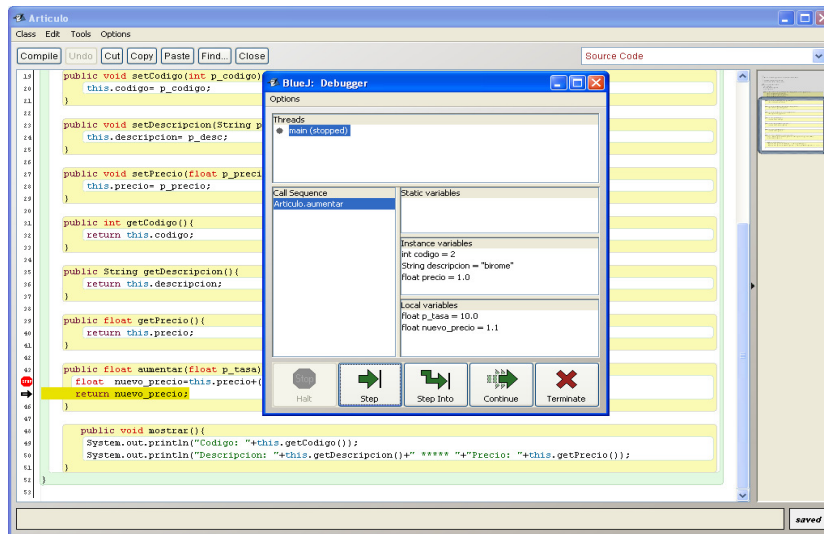
Cada vez que se pulsa *Step*, se ejecuta una única línea de código y la ejecución se detiene de nuevo. También se observa que los valores de las variables mostradas en la ventana del depurador cambian (en este caso *nuevo_precio*). Por lo tanto se puede ejecutar paso a paso y observar qué sucede en el código. También se puede pulsar en el punto de ruptura de nuevo y eliminarlo, y luego pulsar el botón *Continue* en el depurador para reanunciar la ejecución y continuar normalmente.



También existe *Step Into*. Si se desea dar un paso dentro (step into) de una llamada a un método, se entra en el método y se ejecuta línea a línea (no todo como un único paso). En este caso usted es llevado dentro de un método particular de la clase. Se pueden dar pasos por este método hasta que llegue al final y volver al método que lo ha llamado. Y durante esto se puede observar cómo el depurador muestra cambios.



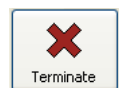
Step y *Step Into* se comportan de modo idéntico si la línea actual no contiene una llamada a un método.



Al finalizar la ejecución del *Step* o *Step Into* se abrirá una ventana mostrando los resultados del método inspeccionado. En este caso es el método *aumentar*, al que se pasó como parámetro la tasa de aumento del 10 por ciento.

Además se observa en la ventana de resultados una cuestión importante en la orientación a objetos: *objeto.mensaje()*. Si se observa en la figura anterior: *articulo1.aumentar(10)*.

Si no se quiere continuar (por ejemplo, si se descubre que realmente está en un ciclo infinito) simplemente pulsar *Terminate* para terminar por completo la ejecución. *Terminate* no debería usarse muy frecuentemente – puede dejar objetos que están perfectamente bien escritos en un estado inconsistente al terminar la ejecución de la máquina, por lo que es recomendable usarlo sólo como un mecanismo de emergencia.



7 Aprendiendo a usar Paquetes

Imaginemos que deseamos construir un software complejo con un compañero, entonces nos repartimos el proyecto entre los dos y cada uno implementa varias clases. Cada uno va a emplear en su código parte de las clases del otro y viceversa, es decir, para construir la funcionalidad de las clases probablemente se apoyen en otras clases auxiliares (colaboran). Dada esta situación: ¿no sería interesante poder "empaquetar" las clases de tal modo que ese "paquete" sólo permita acceder a nuestro compañero a las clases que uno quiera y oculte las demás? Esas clases a las que se podría acceder serían la interface de ese "paquete". Es la misma idea que hay detrás de una clase pero llevada a un nivel superior: una clase puede definir cuáles de sus partes son accesibles y no accesibles para los demás. El paquete permitiría encapsular cuantas clases queramos, pero mostraría al exterior sólo aquellas que considere adecuado.

Esa es precisamente la utilidad de los *package* en Java. Empaquetar muchas clases y decidir cuáles serán accesibles para los demás y cuáles no.

Para empaquetar las clases simplemente se debe poner al principio del archivo donde se define la clase, en la primera línea que no sea un comentario, una sentencia que indique a qué paquete pertenece dicha clase:

```
Package nombre_paquete;
```

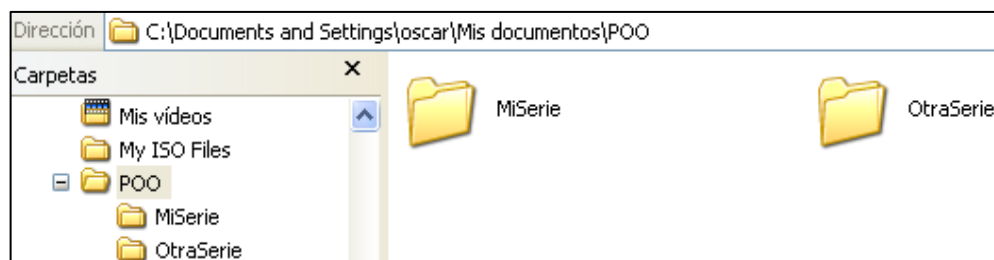
En Java los paquetes se corresponden con una jerarquía de directorios. Por lo tanto en un mismo proyecto podemos emplear el uso de dos o más paquetes que sean necesarios para construir un proyecto.

Uso de paquetes: para emplear clases que se encuentren en otros paquetes diferentes al actual, se utiliza la sentencia **import**. Así, si desde la clase "MiClase" que se encuentra definida dentro de "paquete1" se desea emplear la clase "OtraClase" que se encuentra en "paquete2", en "MiClase" se debe añadir la sentencia:

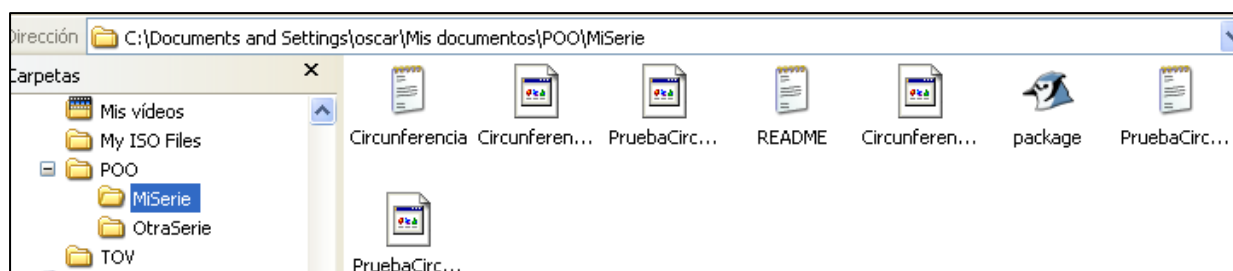
```
import paquete2.OtraClase;
```

7.1 Creación de Paquetes

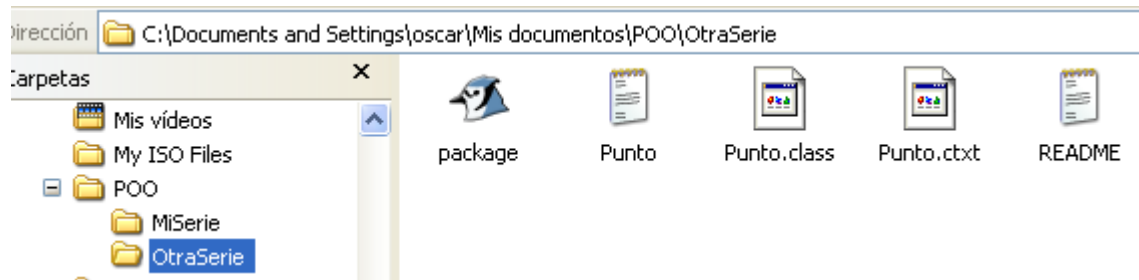
En este ejemplo, se cuenta en la carpeta de nombre "POO" con dos proyectos distintos con los nombres "MiSerie" y "OtraSerie", como muestra la figura:



El contenido de "MiSerie" es:

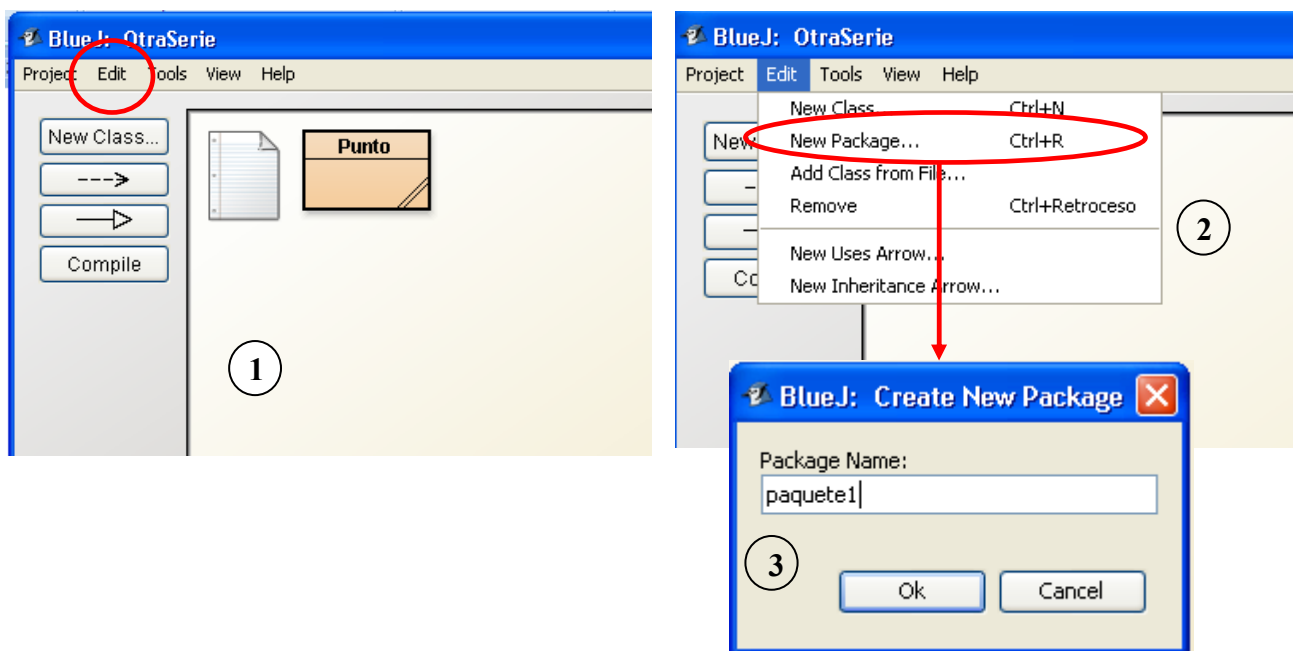


El contenido de “OtraSerie” es:

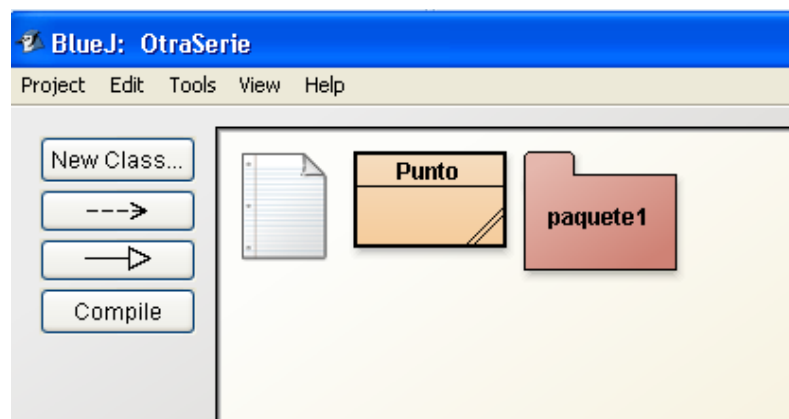


Ahora supongamos que desde “MiSerie” necesitamos las clases definidas en “OtraSerie”. Para ello usaremos los package.

En primer lugar se debe crear un paquete: en el proyecto “OtraSerie”, en BlueJ, seleccionar *Edit* (1) y elegir la opción *New Package...* (2). Luego colocar el nombre deseado para el paquete (3) (en este ejemplo usaremos el nombre “paquete1”) y presionar Ok.



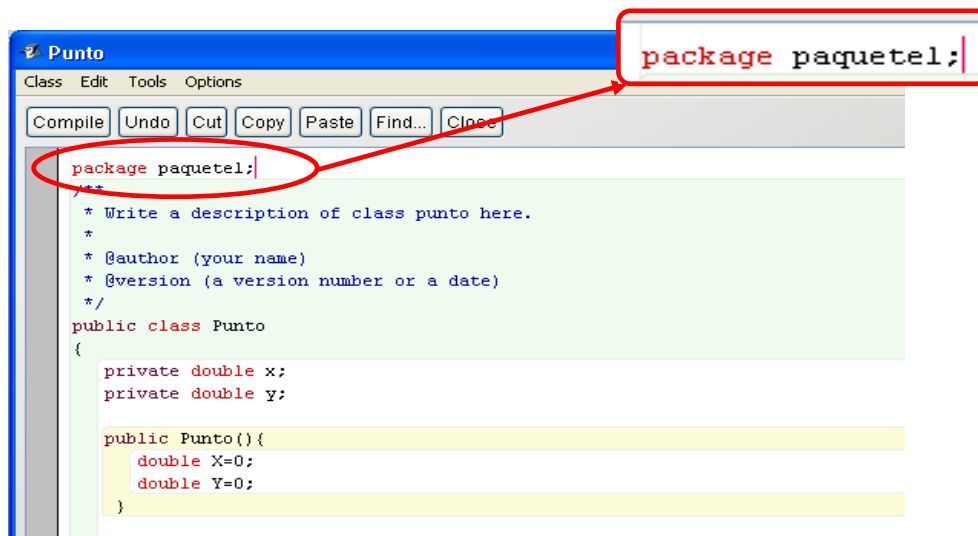
Inmediatamente se creará el paquete, con un ícono como muestra la siguiente figura:



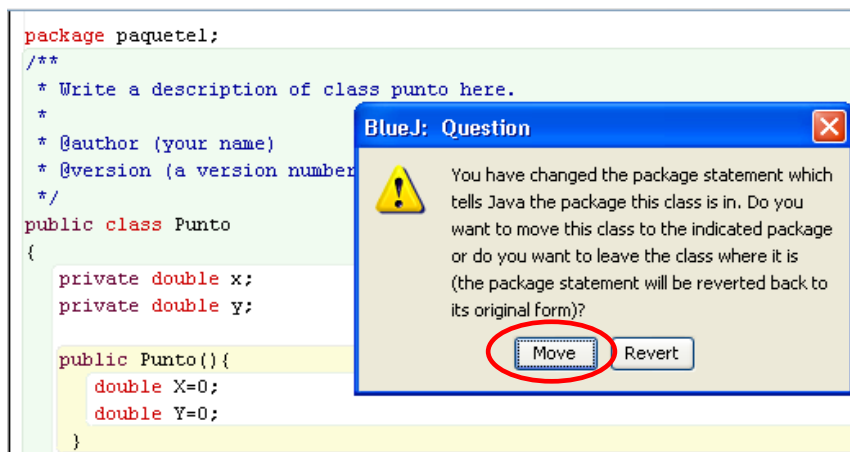
Para incluir la clase “Punto” dentro de “paquete1”, dentro del código de la clase se agrega como primer línea en el código la sentencia:

```
Package paquete1;
```

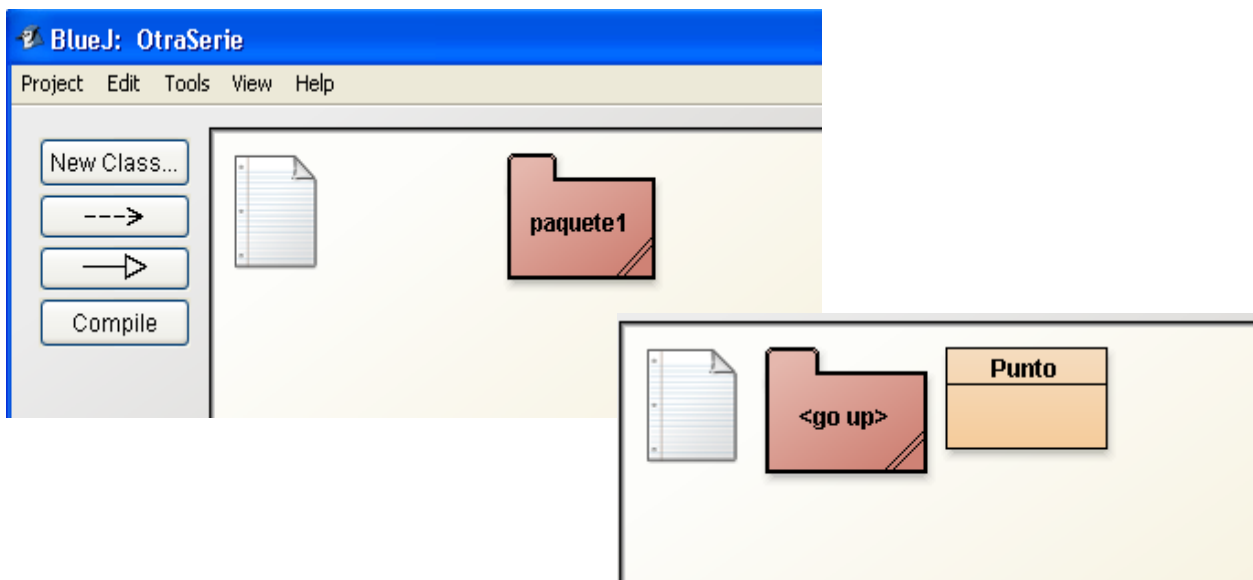
Con esta sentencia estamos diciendo que la clase “Punto” pertenece al paquete “paquete1”, tal como se muestra en la figura



A continuación se abrirá un cuadro de diálogo indicándonos que estamos a punto de trasladar la clase “Punto” del default package (paquete por defecto) al paquete “paquete1”. Presionar el botón **Move**, para hacer efectivo el cambio.

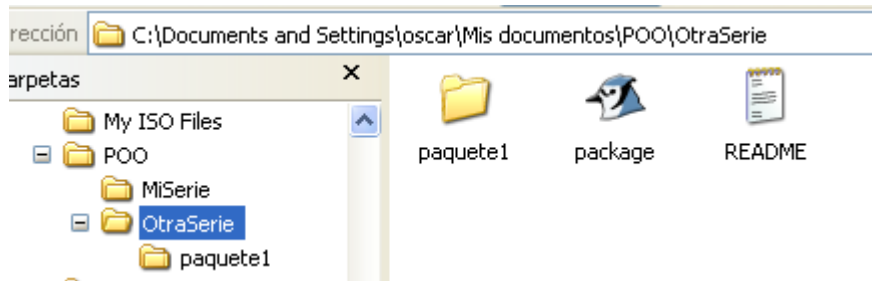


Para ingresar al paquete hacer doble clic sobre él y de ese modo se podrá visualizar su contenido. Por ejemplo, el contenido de "paquete1", es la clase “Punto”.



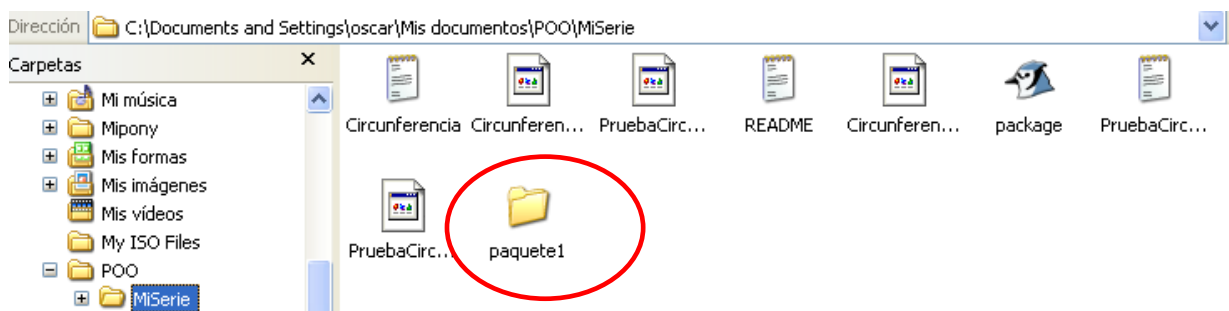
Cuando estamos visualizando el contenido de un paquete si creamos una nueva clase en BlueJ, dicha clase se creará dentro de ese paquete. Las clases creadas en este primer paquete son las que serán usadas desde las clases del segundo proyecto (“MiSerie”). El icono con forma de paquete y con el texto “<go up>” permite subir un nivel en el anidamiento de los paquetes.

Al visualizar el contenido del proyecto “OtraSerie” se observa que se agregó una carpeta de nombre “paquete1”.

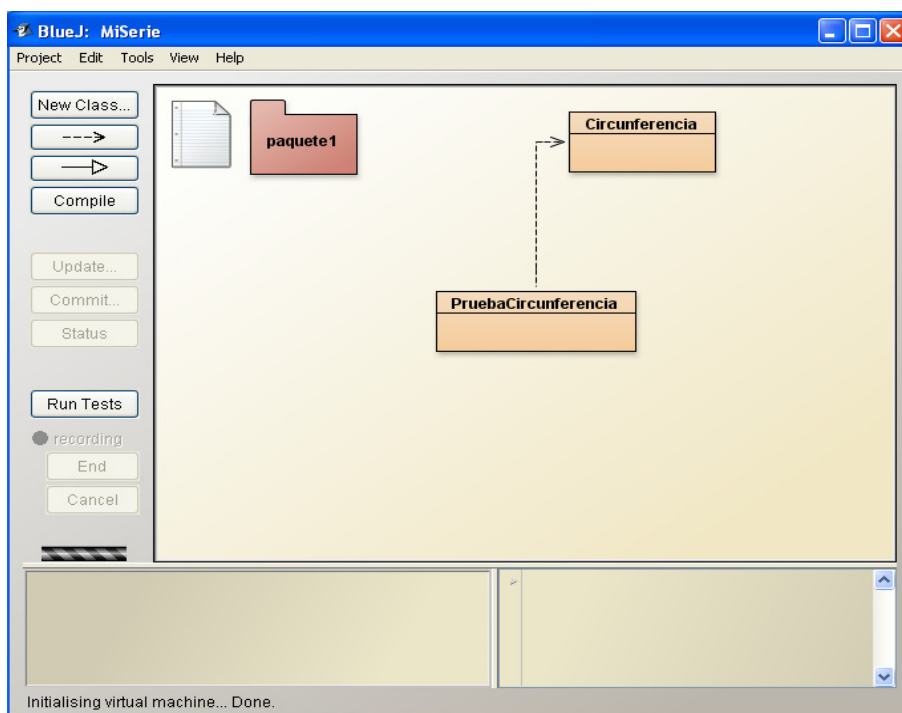


7.2 Uso de paquetes

Una vez creado el paquete se puede usar la carpeta “paquete1” en “MiSerie” y en cualquier proyecto que necesite de las clases incluidas dentro del paquete. Para ello solo movemos, trasladamos o copiamos la carpeta “paquete1” en “MiSerie”.



En el visor de clases de BlueJ aparece el ícono de “paquete1”



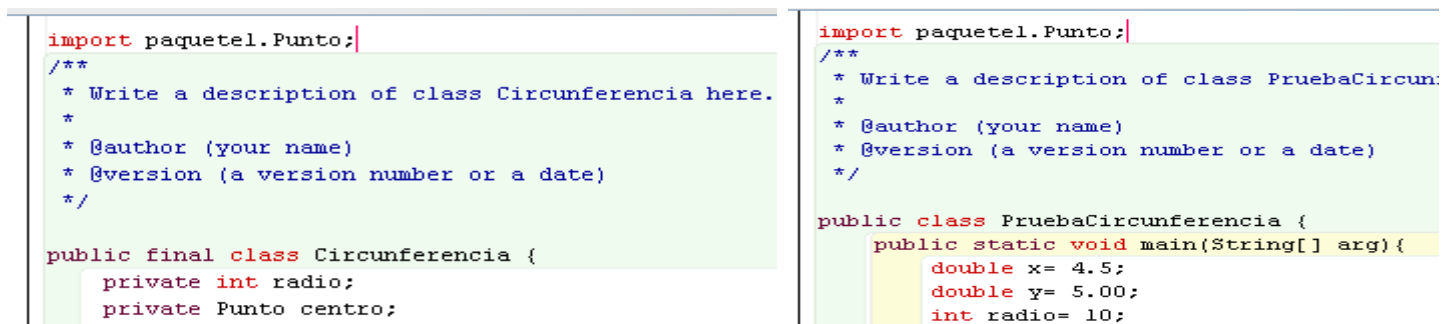
Para usar las clases del paquete (tal como se venía haciendo), se importa el paquete en forma completa, (*), o indicando solamente, la clase a utilizar. Por ejemplo, al importar la clase Scanner del paquete java.util:

```
import java.util.Scanner;
```

En el ejemplo mencionado previamente, se debe colocar la siguiente sentencia:

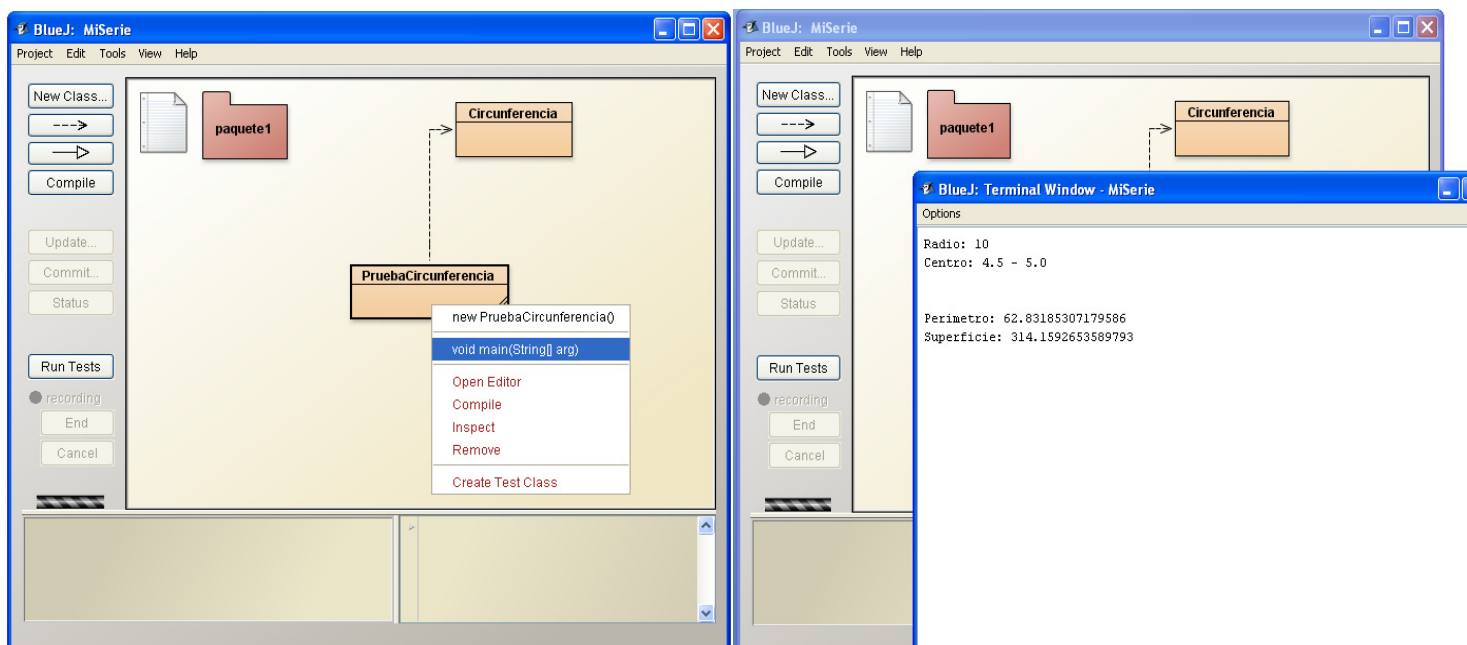
```
import paquete1.Punto;
```

que indica que se está importando del “paquete1” la clase “Punto”. Esto se repite en ambas clases que necesitan de la clase “Punto”, esto es, las clases “Circunferencia” y “PruebaCircunferencia”.



Luego de estos pasos ya se pueden utilizar las clases del paquete creado.

A continuación se muestra la ejecución de la clase “PruebaCircunferencia” con el empleo del paquete “paquete1”.

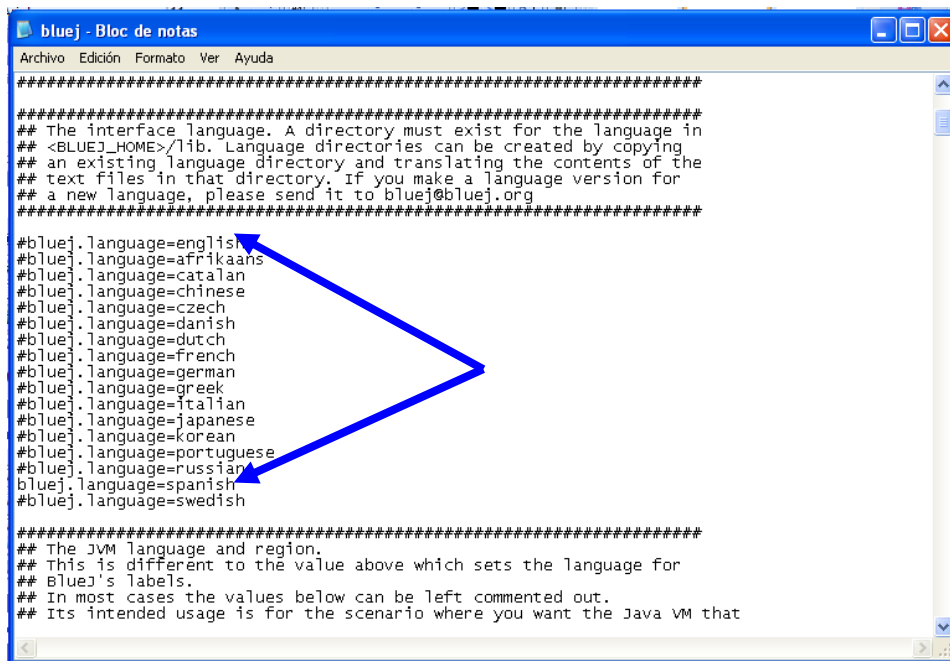


8 Información Adicional en Windows

8.1 Configuración del idioma:

Si se desea modificar el idioma del entorno BlueJ, que por defecto es el inglés, se deben realizar algunas operaciones adicionales:

- localizar el lugar donde se instaló la carpeta BlueJ. En caso de no contar con esta información, hacer click con el botón derecho del mouse sobre el icono que se encuentra en el escritorio.
- Dirigirse a la opción Propiedades y pulsar en “Buscar Destino” o similar, esto nos dará la ubicación exacta de la carpeta BlueJ.
- Ir a la carpeta lib y abrir con algún editor de texto o block de notas el archivo bluej.DEFS.
- Esta operación abrirá una ventana similar a la siguiente, en la cual se debe buscar el directorio de idioma (lenguaje) y eliminar el símbolo numeral (#) a la opción bluej.language=spanish y asignarle numeral (#) a la opción bluej.language=english



```
bluej - Bloc de notas
Archivo Edición Formato Ver Ayuda
#####
## The interface language. A directory must exist for the language in
## <BLUEJ_HOME>/lib. Language directories can be created by copying
## an existing language directory and translating the contents of the
## text files in that directory. If you make a language version for
## a new language, please send it to bluej@bluej.org
#####
#bluej.language=english
#bluej.language=afrikaans
#bluej.language=catalan
#bluej.language=chinese
#bluej.language=czech
#bluej.language=danish
#bluej.language=dutch
#bluej.language=french
#bluej.language=german
#bluej.language=greek
#bluej.language=italian
#bluej.language=japanese
#bluej.language=korean
#bluej.language=portuguese
#bluej.language=russian
#bluej.language=spanish
#bluej.language=swedish
#####
## The JVM language and region.
## This is different to the value above which sets the language for
## BlueJ's labels.
## In most cases the values below can be left commented out.
## Its intended usage is for the scenario where you want the Java VM that
```