TEMA 14: APPLETS

1.- INTRODUCCIÓN

Un ***applet*** es una mini-aplicación, escrita en ***Java***, que se ejecuta en un navegador (browser, tales como ***Mozilla Firefox, Microsoft Internet Explorer***, ***Chrome***, …) al cargar una página HTML que incluye información sobre el ***applet*** a ejecutar por medio de las ***tags*** <APPLET>... </APPLET>.

A continuación se detallan algunas características de las ***applets***:

1. Los ficheros de ***Java*** compilados (*.class) se descargan a través de la red desde un servidor de ***Web*** o servidor ***HTTP*** hasta el browser en cuya *Java Virtual Machine* se ejecutan. Pueden incluir también ficheros de imágenes y sonido.

2. Los ***applets*** no tienen ventana propia: se ejecutan en la ventana del browser (en un "***panel***").

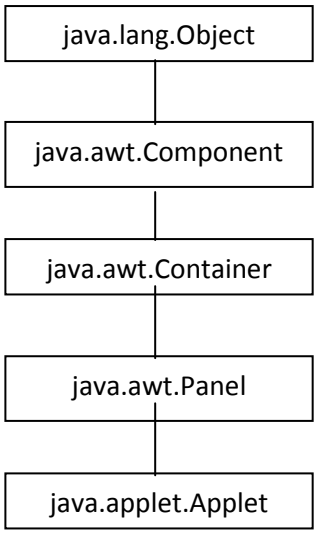
3. Por la propia naturaleza "abierta" de Internet, los ***applets*** tienen importantes restricciones de seguridad, que se comprueban al llegar al browser: sólo pueden leer y escribir ficheros en el servidor del que han venido, sólo pueden acceder a una limitada información sobre el ordenador en el que se están ejecutando, etc. Con ciertas condiciones, los ***applets*** "de confianza" (***trusted applets***) pueden pasar por encima de estas restricciones.

Aunque su entorno de ejecución es un browser, los ***applets*** se pueden probar sin necesidad de browser con la aplicación ***appletviewer*** del JDK de ***Oracle***.

2. CARACTERÍSTICAS DE LOS APPLETS

Las características de los ***applets*** se pueden considerar desde el punto de vista del programador y desde el del usuario. En este manual lo más importante es el punto de vista del programador:

- ✓ Los ***applets*** no tienen un método ***main()*** con el que comience la ejecución. El papel central de su ejecución lo asumen otros métodos que se verán posteriormente.
- ✓ Todos los ***applets*** derivan de la clase ***java.applet.Applet***. La siguiente figura muestra la jerarquía de clases de la que deriva la clase ***Applet***. Los ***applets*** deben redefinir ciertos métodos heredados de ***Applet*** que controlan su ejecución: ***init()***, ***start()***, ***stop()***, ***destroy()***.
- ✓ Se heredan otros muchos métodos de las super-clases de ***Applet*** que tienen que ver con la generación de interfaces gráficas de usuario (AWT). Así, los métodos gráficos se heredan de ***Component***, mientras que la capacidad de añadir componentes de interface de usuario se hereda de ***Container*** y de ***Panel***.
- ✓ Los ***applets*** también suelen redefinir ciertos métodos gráficos: los más importantes son ***paint()*** y ***update()***, heredados de ***Component*** y de ***Container***; y ***repaint()*** heredado de ***Component***.
- ✓ Los ***applets*** disponen de métodos relacionados con la obtención de información, como por ejemplo: ***getAppletInfo()***, ***getAppletContext()***, ***getParameterInfo()***, ***getParameter()***, ***getCodeBase()***, ***getDocumentBase()***, e ***isActive()***. El método ***showStatus()*** se utiliza para mostrar información en la barra de estado del browser. Existen otros métodos relacionados con imágenes y sonido: ***getImage()***, ***getAudioClip()***, ***play()***, etc.



Ciclo de vida de un Applet

MENSAJE	*EVENTO DEL NAVEGADOR*	*COMPORTAMIENTO*
`init`	Carga documento HTML con marca <APPLET> y la clase principal	Código de inicialización (inicializar campos, añadir componentes de interacción, etc.)
`start`	Tras init y cada vez que se vuelva a visitar el applet con el navegador.	Iniciar animaciones y otras tareas.
`stop`	Cada vez que se abandona un applet	Suspender animaciones y otras tareas
`destroy`	Termina el applet (se descarta el documento o se abandona el navegador)	Limpieza de los recursos del sistema
`paint`	Cada vez que se detecta la necesidad de "pintar" el área de pantalla del applet	Redibujar lo necesario en el área de interacción

Para que nuestro applet responda a los eventos del navegador, será necesario sobreescribir el método correspondiente (`init, start, stop, destroy, paint`).

Nota: También existe la clase JApplet propia de Swing

3. METODOS QUE CONTROLAN LA EJECUCION DE UN APPLET

Los métodos que se estudian en este apartado controlan la ejecución de los ***applets***. De ordinario el programador tiene que redefinir uno o más de estos métodos, pero no tiene que preocuparse de llamarlos: el browser se encarga de hacerlo.

3.1. Método init( )

Se llama automáticamente al método ***init()*** en cuanto el browser o visualizador carga el ***applet***. Este método se ocupa de todas las tareas de inicialización, realizando las funciones del ***constructor*** (al que el browser no llama). En este método es donde debemos construir la forma del applet.

3.2. Método start( )

El método ***start()*** se llama automáticamente en cuanto el ***applet*** se hace visible, después de haber sido inicializado. Se llama también cada vez que el ***applet*** se hace de nuevo visible después de haber estado oculto (por dejar de estar activa esa página del browser, al cambiar el tamaño de la ventana del browser, al hacer ***reload***, etc.).

Es habitual crear ***threads*** en este método para aquellas tareas que, por el tiempo que requieren, dejarían sin recursos al ***applet*** o incluso al browser. Las animaciones y ciertas tareas a través de Internet son ejemplos de este tipo de tareas.

3.3. Método stop( )

El método ***stop()*** se llama de forma automática al ocultar el ***applet*** (por haber haber dejado de estar activa la página del browser, por hacer ***reload*** o ***resize***, etc.).

Con objeto de no consumir recursos inútilmente, en este método se suelen parar las ***threads*** que estén corriendo en el ***applet***, por ejemplo para mostrar animaciones.

3.4. Método destroy( )

Se llama a este método cuando el ***applet*** va a ser descargado para liberar los recursos que tenga reservados (excepto la memoria). De ordinario no es necesario redefinir este método, pues el que se hereda cumple bien con esta misión.

3.5. Métodos para dibujar el applet

Los ***applets*** son aplicaciones gráficas que aparecen en una zona de la ventana del browser. Por ello deben redefinir los métodos gráficos ***paint()*** y ***update()***. El método ***paint()*** se declara en la forma:

public void paint(Graphics g)

El objeto gráfico ***g*** pertenece a la clase ***java.awt.Graphics***, que siempre debe ser importada por el ***applet***. Este objeto define un contexto o estado gráfico para dibujar (métodos gráficos, colores, fonts, etc.) y es creado por el browser.

Todo el trabajo gráfico del ***applet*** (dibujo de líneas, formas gráficas, texto, etc.) se debe incluir en el método ***paint()***, porque este método es llamado cuando el ***applet*** se dibuja por primera vez y también de forma automática cada vez que el ***applet*** se debe redibujar.

En general, el programador crea el método ***paint()*** pero no lo suele llamar. Para pedir explícitamente al sistema que vuelva a dibujar el ***applet*** (por ejemplo, por haber realizado algún cambio) se utiliza el método ***repaint()***, que es más fácil de usar, pues no requiere argumentos. El método ***repaint()*** se encarga de llamar a ***paint()*** a través de ***update()***.

El método ***repaint()*** llama a ***update()***, que borra todo pintando de nuevo con el color de fondo y luego llama a ***paint()***. A veces esto produce parpadeo de pantalla o ***flickering***. Existen dos formas de evitar el ***flickering***:

1. Redefinir ***update()*** de forma que no borre toda la ventana sino sólo lo necesario.
2. Redefinir ***paint()*** y ***update()*** para utilizar ***doble buffer***.

Ambas formas fueron estudiadas en el tema anterior.

Ejemplo:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;

public class Applet1 extends java.applet.Applet {
    Font f = new Font("TimesRoman", Font.BOLD, 36);
    public void paint(Graphics pantalla) {
          pantalla.setFont(f);
          pantalla.setColor(Color.red);
          pantalla.drawString("Esto es una prueba de un Applet", 5, 40);
    }
}
```

4. CÓMO INCLUIR UN APPLET EN UNA PÁGINA HTML.

Para llamar a un ***applet*** desde una página HTML se utiliza la tag doble <APPLET>…</APPLET>, cuya forma general es (los elementos opcionales aparecen entre corchetes[]):

```
<APPLET CODE="miApplet.class" [CODEBASE="unURL"] [NAME="unName"]
      WIDTH="wpixels" HEIGHT="hpixels"
      [ALT="TextoAlternativo"]>
      [texto alternativo para browsers que reconocen el tag <applet> pero no pueden
      ejecutar el applet]
      [<PARAM NAME="MyName1" VALUE="valueOfMyName1">]
      [<PARAM NAME="MyName2" VALUE="valueOfMyName2">]
</APPLET>
```

El atributo NAME permite dar un nombre opcional al ***applet***, con objeto de poder comunicarse con otras ***applets*** o con otros elementos que se estén ejecutando en la misma página. El atributo ARCHIVE permite indicar uno o varios ficheros Jar o Zip (separados por comas) donde se deben buscar las clases.

A continuación se señalan otros posibles atributos de <APPLET>:

- ✓ ARCHIVE="file1, file2, file3". Se utiliza para especificar ficheros JAR y ZIP.
- ✓ ALIGN, VSPACE, HSPACE. Tienen el mismo significado que el tag IMG de HTML.

Ejemplo. Fichero Applet1.html

```
<HTML>
<HEAD>
<TITLE> Mi primer ejemplo de Applet </TITLE>
</HEAD>
<BODY>
<APPLET CODE="Applet1.class" WIDTH=600 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

5. PASO DE PARÁMETROS A UN APPLET.

Los tags PARAM permiten pasar diversos parámetros desde el fichero HTML al programa ***Java*** del ***applet***, de una forma análoga a la que se utiliza para pasar argumentos a ***main()***.

Cada parámetro tiene un ***nombre*** y un ***valor***. Ambos se dan en forma de ***String***, aunque el valor sea numérico. El ***applet*** recupera estos parámetros y, si es necesario, convierte los ***Strings*** en valores numéricos. El valor de los parámetros se obtienen con el siguiente método de la clase ***Applet***:

String getParameter(String name)

La conversión de ***Strings*** a los tipos primitivos se puede hacer con los métodos asociados a los ***wrappers*** que ***Java*** proporciona para dichos tipo fundamentales (Integer.parseInt(String), Double.valueOf(String), …). Estas clases de wrappers se estudiaron en temas anteriores.

En los ***nombres*** de los parámetros no se distingue entre mayúsculas y minúsculas, pero sí en los ***valores***, ya que serán interpretados por un programa ***Java***, que sí distingue.

El programador del ***applet*** debería prever siempre unos ***valores por defecto*** para los parámetros del ***applet***, para el caso de que en la página HTML que llama al ***applet*** no se definan.

El método ***getParameterInfo()*** devuelve una matriz de ***Strings*** (*String[][]*) con información sobre cada uno de los parámetros soportados por el ***applet***: *nombre*, *tipo* y *descripción*, cada uno de ellos en un ***String***. Este método debe ser redefinido por el programador del ***applet*** y utilizado por la persona que prepara la página HTML que llama al ***applet***. En muchas ocasiones serán personas distintas, y ésta es una forma de que el programador del ***applet*** dé información al usuario.

6. CARGA DE APPLETS.

6.1. Localización de ficheros.

Por defecto se supone que los ficheros ****.class*** del ***applet*** están en el mismo directorio que el fichero HTML. Si el ***applet*** pertenece a un ***package***, el browser utiliza el nombre del ***package*** para construir un ***path de directorio*** relativo al directorio donde está el HTML.

El atributo CODEBASE permite definir un URL para los ficheros que continen el código y demás elementos del ***applet***. Si el directorio definido por el URL de CODEBASE es *relativo*, se interpreta respecto al directorio donde está el HTML; si es *absoluto* se interpreta en sentido estricto y puede ser cualquier directorio de la Internet.

6.2. Archivos JAR (Java Archives).

Si un ***applet*** consta de varias clases, cada fichero ****.class*** requiere una conexión con el servidor de Web (servidor de protocolo HTTP), lo cual puede requerir algunos segundos. En este caso es conveniente agrupar todos los ficheros en un archivo único, que se puede comprimir y cargar con una sola conexión HTTP.

Los archivos JAR están basados en los archivos ZIP y pueden crearse con el programa ***jar*** que viene con el JDK. Por ejemplo:

*jar cvf myFile.jar *.class *.gif*

crea un fichero llamado ***myFile.jar*** que contiene todos los ficheros ****.class*** y ****.gif*** del directorio actual. Si las clases pertenecieran a un package llamado ***es.ceit.infor2*** se utilizaría el comando:

jar cvf myFile.jar es\ceit\infor2.class *.gif*

7. COMUNICACIÓN DEL APPLET CON EL BROWSER.

La comunicación entre el applet y el browser en el que se está ejecutando se puede controlar mediante la interface ***AppletContext*** (package ***java.applet***). ***AppletContext*** es una interface implementada por el browser, cuyos métodos pueden ser utilizados por el ***applet*** para obtener información y realizar ciertas operaciones, como por ejemplo sacar ***mensajes breves*** en la ***barra de estado*** del browser. Hay que tener en cuenta que la barra de estado es compartida por el browser y las ***applets***, lo que tiene el peligro de que el mensaje sea rápidamente sobre-escrito por el browser u otros ***applets*** y que el usuario no llegue a enterarse del mensaje.

Los mensajes breves a la barra de estado se producen con el método ***showStatus()***, como porejemplo,

getAppletContext().showStatus("Cargado desde el fichero " + filename);

Los mensajes más importantes se deben dirigir a la ***salida estándar*** o a la ***salida de errores***, que en ***Netscape Navigator*** es la ***Java Console*** (la cual se hace visible desde el menú ***Options*** en ***Navigator 3.0***, desde el menú ***Communicator*** en ***Navigator 4.0**** y desde ***Communicator/Tools*** en ***Navigator 4.5***). Estos mensajes se pueden enviar con las sentencias:

```
System.out.print();
System.out.println();
System.error.print();
System.error.println();
```

Para mostrar documentos HTML en una ventana del browser se pueden utilizar los métodos siguientes:

- ✓ ***showDocument(URL miUrl, [String target])***, que muestra un documento HTML en el *frame* del browser indicado por *target* (*name*, *_top*, *_parent*, *_blank*, *_self*).
- ✓ ***showDocument(URL miUrl)***, que muestra un documento HTML en la ventana actual del browser.

Un ***applet*** puede conseguir información de otros ***applets*** que están corriendo en la misma página del browser, enviarles mensajes y ejecutar sus métodos. El mensaje se envía invocando los métodos del otro ***applet*** con los argumentos apropiados.

Algunos browsers exigen, para que los ***applets*** se puedan comunicar, que los ***applets*** provengan del mismo browser o incluso del mismo directorio (que tengan el mismo *codebase*).

Por ejemplo, para obtener información de otros applets se pueden utilizar los métodos:

- ✓ ***getApplet(String name)***, que devuelve el ***applet*** llamada ***name*** (o ***null*** si no la encuentra). El nombre del ***applet*** se pone con el atributo opcional NAME o con el parámetro NAME.
- ✓ ***getApplets()***, que devuelve una *enumeración* con todas las ***applets*** de la página.

Para poder utilizar todos los métodos de un applet que se está ejecutando en la misma página HTML (y no sólo los métodos comunes heredados de ***Applet***), debe hacerse un ***cast*** del objeto de la clase ***Applet*** que se obtiene como valor de retorno de ***getApplet()*** a la clase concreta del ***applet***.

Para que pueda haber ***respuesta*** (es decir, comunicación en los dos sentidos), el primer applet que envía un mensaje debe enviar una referencia a sí misma por medio del argumento ***this***.

Ejemplo.

```
// fichero Applet2.java
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;

public class Applet2 extends java.applet.Applet {
    Font f = new Font("TimesRoman", Font.BOLD, 36);
    String parametro;

    public void paint(Graphics pantalla) {
          pantalla.setFont(f);
          pantalla.setColor(Color.red);
          pantalla.drawString(parametro, 5, 40);
    }

    public void init() {
          parametro=getParameter("Saludo");
          if (parametro==NULL)
                parametro="Esto es un ejemplo de Applet";
```

```
    }
}

// fichero Applet2.html
<HTML>
<HEAD>
<TITLE> Mi segundo ejemplo de Applet </TITLE>
</HEAD>
<BODY>
<APPLET CODE="Applet1.class" CODEBASE= "/JDK.2/APPLETS" WIDTH=600 HEIGHT=100>
<PARAM NAME = parametro VALUE = "Esto es un ejemplo de Applet"
</APPLET>
</BODY>
</HTML>
```

8. SONIDOS EN APPLETS.

La clase ***Applet*** y la interface ***AudioClips*** permiten utilizar sonidos en applets. La siguiente tabla muestra algunos métodos interesantes al respecto.

Métodos de Applet	***Función que realizan***
`public AudioClip getAudioClip(URL url)`	Devuelve el objeto especificado por url, que implementa la interface AudioClip
`public AudioClip getAudioClip(URL url, String name)`	Devuelve el objeto especificado por url (dirección base) y name (dirección relativa)
`play(URL url), play(URL url, String name)`	Hace que suene el AudioClip correspondiente a la dirección especificada
Métodos de la interface AudioClip (en java.applet)	**Función que realizan**
`void loop()`	Ejecuta el sonido repetidamente
`void play()`	Ejecuta el sonido una sola vez
`void stop()`	Detiene el sonido

Respecto a la carga de sonidos, por lo general es mejor cargar los sonidos en un ***thread*** distinto (creado en el método ***init()***) que en el propio método ***init()***, que tardaría en devolver el control y permitir al usuario empezar a interaccionar con el ***applet***.

Si el sonido no ha terminado de cargarse (en la ***thread*** especial para ello) y el usuario interacciona con el ***applet*** para ejecutarlo, el ***applet*** puede darle un aviso de que no se ha terminado de cargar.

9. IMÁGENES EN APPLETS.

Los ***applets*** admiten los formatos JPEG y GIF para representar imágenes a partir de ficheros localizados en el servidor. Estas imágenes se pueden cargar con el método ***getImage()*** de la clase ***Applet***, que puede tener las formas siguientes:

public Image getImage(URL url)
public Image getImage(URL url, String name)

Estos métodos devuelven el control inmediatamente. Las imágenes se cargan cuando se da la orden de dibujar las imágenes en la pantalla. El dibujo se realiza entonces de forma incremental, a medida que el contenido va llegando.

Para dibujar imágenes se utiliza el método ***drawImage()*** de la clase ***Graphics***, que tiene las formas siguientes:

```
public abstract boolean drawImage(Image img, int x, int y,
                                  Color bgcolor, ImageObserver observer)
public abstract boolean drawImage(Image img, int x, int y, int width, int height,
                                  Color bgcolor, ImageObserver observer)
```

El primero de ellos dibuja la imagen con su tamaño natural, mientras que el segundo realiza un cambio en la escala de la imagen.

Los métodos ***drawImage()*** van dibujando la parte de la imagen que ha llegado, con su tamaño, a partir de las coordenadas (x, y) indicadas, utilizando ***bgcolor*** para los pixels transparentes.

Estos métodos devuelven el control inmediatamente, aunque la imagen no esté del todo cargada. En este caso devuelve ***false***. En cuanto se carga una parte adicional de la imagen, el proceso que realiza el dibujo avisa al ***ImageObserver*** especificado. ***ImageObserver*** es una interface implementada por ***Applet*** que permite seguir el proceso de carga de una imagen.

10. OBTENCIÓN DE LAS PROPIEDADES DEL SISTEMA.

Un ***applet*** puede obtener información del sistema o del entorno en el que se ejecuta. Sólo algunas propiedades del sistema son accesibles. Para acceder a las propiedades del sistema se utiliza un método ***static*** de la clase ***System***:

```
String salida = System.getProperty("file.separator");
```

Los nombres y significados de las propiedades del sistema accesibles son las siguientes:
"file.separator" Separador de directorios (por ejemplo, "/" o "\")
"java.class.version" Número de version de las clases de ***Java.***
"java.vendor" Nombre específico del vendedor de Java.
"java.vendor.url" URL del vendedor de Java.
"java.version" Número de versión Java.
"line.separator" Separador de líneas.
"os.arch" Arquitectura del sistema operativo.
"os.name" Nombre del sistema operativo.
"path.separator" Separador en la variable Path (por ejemplo, ":")

No se puede acceder a las siguientes propiedades del sistema: "java.class.path", "java.home", "user.dir", "user.home", "user.name".

11. UTILIZACIÓN DE THREADS EN APPLETS.

Un ***applet*** puede ejecutarse con varias ***threads***, y en muchas ocasiones será necesario o conveniente hacerlo así. Hay que tener en cuenta que un ***applet*** se ejecuta siempre en un browser (o en la aplicación ***appletviewer***).

Así, las ***threads*** en las que se ejecutan los métodos mayores -***init()***, ***start()***, ***stop()*** y ***destroy()***- dependen del browser o del entorno de ejecución. Los métodos gráficos -***paint()***, ***update()*** y ***repaint()***- se ejecutan siempre desde una ***thread*** especial del AWT.

Algunos browsers dedican un ***thread*** para cada ***applet*** en una misma página; otros crean un grupo de ***threads*** para cada ***applet*** (para poderlas matar al mismo tiempo, por ejemplo). En cualquier caso se garantiza que todas las ***threads*** creadas por los métodos mayores pertenecen al mismo grupo.

Se deben introducir ***threads*** en ***applets*** siempre que haya tareas que consuman mucho tiempo (cargar una imagen o un sonido, hacer una conexión a Internet, …). Si estas tareas pesadas se ponen en el método ***init()*** bloquean cualquier actividad del ***applet*** o incluso de la página HTML hasta que se completan. Las tareas pesadas pueden ser de dos tipos:

- ✓ Las que sólo se hacen una vez.
- ✓ Las que se repiten muchas veces.

Un ejemplo de tarea que se repite muchas veces puede ser una animación. En este caso, la tarea repetitiva se pone dentro de un bucle ***while*** o ***do…while***, dentro del ***thread***. El ***thread*** se debería crear dentro del método ***start()*** del ***applet*** y destruirse en ***stop()***. De este modo, cuando el ***applet*** no está visible se dejan de consumir recursos.

Al crear el ***thread*** en el método ***start()*** se pasa una referencia al ***applet*** con la palabra ***this***, que se refiere al ***applet***. El applet deberá implementar la interface ***Runnable***, y por tanto debe definir el método ***run()***, que es el centro del ***Thread.***

Un ejemplo de tarea que se realiza una sola vez es la carga de imágenes ****.gif*** o ****.jpeg***, que ya se realiza automáticamente en un ***thread*** especial.

Sin embargo, los sonidos no se cargan en ***threads*** especiales de forma automática; los debe crear el programador para cargarlos en "background". Este es un caso típico de programa ***producer-consumer***: el ***thread*** es el *producer* y el ***applet*** el *consumer*. Las ***threads*** deben estar ***sincronizadas***, para lo que se utilizan los métodos ***wait()*** y ***notifyAll()***.

A continuación se presenta un ejemplo de ***thread*** con tarea repetitiva:

```
public void start() {
     if (repetitiveThread == null) {
          repetitiveThread = new Thread(this); // se crea un nuevo thread
     }
     repetitiveThread.start(); // se arranca el thread creado: start() llama a
     run()
}

public void stop() {
     repetitiveThread = null; // para parar la ejecución del thread
}

public void run() {
     ...
     while (Thread.currentThread() == repetitiveThread) {
... // realizar la tarea repetitiva.
     }
}
```

El método ***run( )*** se detendrá en cuanto se ejecute el método ***stop( )***, porque la referencia al ***thread*** está a ***null***.

12. APPLETS QUE TAMBIÉN SON APLICACIONES.

Es muy interesante desarrollar ***aplicaciones*** que pueden funcionar también como ***applets*** y viceversa. En concreto, para hacer que un ***applet*** pueda ejecutarse como ***aplicación*** pueden seguirse las siguientes instrucciones:

1. Se añade un método ***main()*** a la clase ***MiApplet*** (que deriva de ***Applet***).
2. El método ***main()*** debe crear un objeto de la clase ***MiApplet*** e introducirlo en un ***Frame***.
3. El método ***main()*** debe también ocuparse de hacer lo que haría el browser, es decir, llamar a los métodos ***init()*** y ***start()*** de la clase ***MiApplet***.

4. Se puede añadir también una ***static inner class*** que derive de ***WindowAdapter*** y que gestione el evento de cerrar la ventana de la aplicación definiendo el método ***windowClosing()***. Este método llama al método ***System.exit(0)***. Según como sea el ***applet***, el método ***windowClosing()*** previamente deberá también llamar a los métodos ***MiApplet.stop()*** y ***MiApplet.destroy()***, cosa que para las ***applets*** se encarga de hacer el browser. En este caso conviene que el objeto de ***MiApplet*** creado por ***main()*** sea ***static***, en lugar de una variable local.

   A continuación se presenta un ejemplo:

```
public class MiApplet extends Applet {
       ...
      public void init() {...}
      ...
      // clase para cerrar la aplicación
            static class WL extends WindowsAdapter {
            public void windowClosing(WindowEvent e) {
            MiApplet.stop();
            MiApplet.destroy();
            System.exit(0);
      }
      } // fin de WindowAdapter

      // programa principal
      public static void main(String[] args) {
            static MiApplet unApplet = new MiApplet();
            Frame unFrame = new Frame("MiApplet");
            unFrame.addWindowListener(new WL());
            unFrame.add(unapplet, BorderLayout.CENTER);
            unFrame.setSize(400,400);
            unApplet.init();
            unApplet.start();
            unFrame.setVisible(true);
      }
} // fin de la clase MiApplet
```