CLASE 10: Applets

10.1 Definición de Applet

La definición más extendida de applet, muy bien resumida por Patrick Naughton, indica que un applet es "una pequeña aplicación accesible en un servidor Internet, que se transporta por la red, se instala automáticamente y se ejecuta in situ como parte de un documento web". Claro que así la definición establece el entorno (Internet, Web, etc.). En realidad, un applet es una aplicación pretendidamente corta (nada impide que ocupe más de un gigabyte, a no ser el pensamiento de que se va a transportar por la red y una mente sensata) basada en un formato gráfico sin representación independiente: es decir, se trata de un elemento a embeber en otras aplicaciones; es un componente en su sentido estricto.

Un ejemplo en otro ámbito de cosas podría ser el siguiente: Imaginemos una empresa, que cansada de empezar siempre a codificar desde cero, diseña un formulario con los datos básicos de una persona (nombre, dirección, etc.). Tal formulario no es un diálogo por sí mismo, pero se podría integrar en diálogos de clientes, proveedores, empleados, etc. El hecho de que se integre estática (embebido en un ejecutable) o dinámicamente (intérpretes, DLLs, etc.) no afecta en absoluto a la esencia de su comportamiento como componente con que construir diálogos con sentido autónomo.

Pues bien, así es un applet. Lo que ocurre es que, dado que no existe una base adecuada para soportar aplicaciones industriales Java en las que insertar nuestras miniaplicaciones (aunque todo se andará), los applets se han construido mayoritariamente, y con gran acierto comercial (parece), como pequeñas aplicaciones interactivas, con movimiento, luces y sonido... en Internet.

Las características de las applets se pueden considerar desde el punto de vista del programador y desde el del usuario. En este curso lo más importante es el punto de vista del programador. Las 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.

Todas las applets derivan de la clase java.applet.Applet. La siguiente figura muestra la jerarquía de clases de la que deriva la clase Applet. Las 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 interfaz de usuario se hereda de Container y de Panel.

Las 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.

10.1.1 Llamadas a Applets

Un applet es una mínima aplicación Java diseñada para ejecutarse en un navegador Web. Por tanto, no necesita preocuparse por un método main() ni en dónde se realizan las llamadas. El applet asume que el código se está ejecutando desde dentro de un navegador. El navegador espera como argumento el nombre del fichero html que debe cargar, no se le puede pasar directamente un programa Java. Este fichero html debe contener una marca que especifica el código que cargará:

```
<HTML>
<APPLET CODE=HolaMundo.class WIDTH=300 HEIGHT=100>
</APPLET>
</HTML>
```

El navegador creará un espacio, incluyendo un área gráfica, donde se ejecutará el applet, entonces llamará a la clase applet apropiada.

10.1.2 Ejemplo de un Applet

```
import java.applet.*;
import java.awt.*;

public class HolaMundo extends Applet {
    public void paint(Graphics g) {
        // Display "Hola Mundo!"
        g.drawString("Hola Mundo!", 50, 25);
    }
}
```

Graba este código en un fichero llamado **HolaMundo.java**. También necesitas un fichero HTML que acompañe el applet:

```
<HTML>
<HEAD>
<TITLE>Un Programa Simple</TITLE>
</HEAD>
<BODY>
Here is the output of my program:
<APPLET CODE="HolaMundo.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

Graben este código en un fichero llamado Hola.html y ejecútenlo.

10.2 La clase Applet

Se puede crear una nueva clase, en este caso HolaMundo, extendiendo la clase básica de Java: Applet. De esta forma, se hereda todo lo necesario para crear un applet. Modificando determinados métodos del applet, podemos lograr que lleve a cabo las funciones que deseamos.

```
import java.applet.Applet;
. . .
public class HolaMundo extends Applet {
```

10.2.1 Metodos de la clase Applet

init(): Esta función miembro es llamada al crearse el applet. Es llamada sólo una vez. La clase Applet no hace nada en init(). Las clases derivadas deben sobrecargar este método para cambiar el tamaño durante su inicialización, y cualquier otra inicialización de los datos que solamente deba realizarse una vez. Deberían realizarse al menos las siguientes acciones:

Carga de imágenes y sonido
El resize del applet para que tenga su tamaño correcto
Asignación de valores a las variables globales

Por ejemplo:

```
public void init() {
    if( width < 200 || height < 200 )
        resize( 200,200 );
```

```
    valor_global1 = 0;
    valor_global2 = 100;

    // cargaremos imágenes en memoria sin mostrarlas
    // cargaremos música de fondo en memoria sin reproducirla
    }
```

destroy(): Esta función miembro es llamada cuando el applet no se va a usar más. La clase Applet no hace nada en este método. Las clases derivadas deberían sobrecargarlo para hacer una limpieza final. Los applet multithread deberán usar destroy() para "matar" cualquier thread del applet que quedase activo.
start(): Llamada para activar el applet. Esta función miembro es llamada cuando se visita el applet. La clase Applet no hace nada en este método. Las clases derivadas deberían sobrecargarlo para comenzar una animación, sonido, etc.

```
public void start() {
    estaDetenido = false;

    // comenzar la reproducción de la música
    musicClip.play();
    }
```

También se puede utilizar start() para eliminar cualquier thread que se necesite.
stop(): Llamada para detener el applet. Se llama cuando el applet desaparece de la pantalla. La clase Applet no hace nada en este método. Las clases derivadas deberían sobrecargarlo para detener la animación, el sonido, etc.

```
public void stop() {
    estaDetenido = true;

    if( /* ¿se está reproduciendo música? */ )
        musicClip.stop();
        }
```

resize(int width,int height)
El método init() debería llamar a esta función miembro para establecer el tamaño del applet. Puede utilizar las variables ancho y alto, pero no es necesario. Cambiar el tamaño en otro sitio que no sea init() produce un reformateo de todo el documento y no se recomienda.
En el navegador Netscape, el tamaño del applet es el que se indica en la marca APPLET del HTML, no hace caso a lo que se indique desde el código Java del applet.
width: Variable entera, su valor es el ancho definido en el parámetro WIDTH de la marca HTML del APPLET. Por defecto es el ancho del icono.
height: Variable entera, su valor es la altura definida en el parámetro HEIGHT de la marca HTML del APPLET. Por defecto es la altura del icono. Tanto width como height están siempre disponibles para que se puede chequear el tamaño del applet.

Podemos retomar el ejemplo de init():

```
public void init() {
    if( width < 200 || height < 200 )
        resize( 200,200 );
    ...
```

paint(Graphics g): Se llama cada vez que se necesita refrescar el área de dibujo del applet. La clase Applet simplemente dibuja una caja con sombreado de tres dimensiones en el área.

Obviamente, la clase derivada debería sobrecargar este método para representar algo inteligente en la pantalla.

Para repintar toda la pantalla cuando llega un evento Paint, se pide el rectángulo sobre el que se va a aplicar paint() y si es más pequeño que el tamaño real del applet se invoca a repaint(), que como va a hacer un update(), se actualizará toda la pantalla.

Podemos utilizar paint() para imprimir nuestro mensaje de bienvenida:

```
void public paint( Graphics g ) {
    g.drawString( "Hola Java!",25,25 );
    // Dibujaremos la imágenes que necesitemos
    }
```

update(Graphics g): Esta es la función que se llama realmente cuando se necesita actualizar la pantalla. La clase Applet simplemente limpia el área y llama al método paint(). Esta funcionalidad es suficiente en la mayoría de los casos. De cualquier forma, las clases derivadas pueden sustituir esta funcionalidad para sus propósitos.

Podemos, por ejemplo, utilizar update() para modificar selectivamente partes del área gráfica sin tener que pintar el área completa:

```
public void update( Graphics g ) {
    if( estaActualizado )
      {
      g.clear(); // garantiza la pantalla limpia
      repaint(); // podemos usar el método padre: super.update()
      }
    else
      // Información adicional
      g.drawString( "Otra información",25,50 );
    }
```

repaint(): A esta función se la debería llamar cuando el applet necesite ser repintado. No debería sobrecargarse, sino dejar que Java repinte completamente el contenido del applet.

Al llamar a repaint(), sin parámetros, internamente se llama a update() que borrará el rectángulo sobre el que se redibujará y luego se llama a paint(). Como a repaint() se le pueden pasar parámetros, se puede modificar el rectángulo a repintar.
getParameter(String attr): Este método carga los valores parados al applet vía la marca APPLET de HTML. El argumento String es el nombre del parámetro que se quiere obtener. Devuelve el valor que se le haya asignado al parámetro; en caso de que no se le haya asignado ninguno, devolverá null.

Para usar getParameter(), se define una cadena genérica. Una vez que se ha capturado el parámetro, se utilizan métodos de cadena o de números para convertir el valor obtenido al tipo adecuado.

```
public void init() {
    String pv;

    pv = getParameter( "velocidad" );
    if( pv == null )
    velocidad = 10;
    else
```

```
velocidad = Integer.parseInt( pv );
}
```

getDocumentBase(): Indica la ruta http, o el directorio del disco, de donde se ha recogido la página HTML que contiene el applet, es decir, el lugar donde está la hoja en todo Internet o en el disco.
getCodeBase(): Indica la ruta http, o el directorio del disco, de donde se ha cargado el código bytecode que forma el applet, es decir, el lugar donde está el fichero .class en todo Internet o en el disco.
print(Graphics g): Para imprimir en impresora, al igual que paint() se puede utilizar print(), que pintará en la impresora el mapa de bits del dibujo.

10.3 Manejo de un Applet

Cuando un applet se carga, comienza su ciclo de vida, que pasaría por las siguientes fases:

- Se crea una instancia de la clase que controla el applet.
- El applet se incializa.
- El applet comienza a ejecutarse.
- El applet empieza a recibir llamadas. Primero recibe una llamada init (inicializar), seguida de un mensaje start (empezar) y paint (pintar). Estas llamadas pueden ser recibidas asíncronamente.

10.3.1 Llamadas a Applets

¿Qué tienen de especial HotJava, Microsoft Explorer o Netscape con respecto a otros navegadores? Con ellos se puede ver html básico y acceder a todo el texto, gráfico, sonido e hipertexto que se pueda ver con cualquier otro navegador. Pero además, pueden ejecutar applets, que no es html estándar. Ambos navegadores entienden código html que lleve la marca <APPLET>:

```
<APPLET CODE="SuCodigo.class" WIDTH=100 HEIGHT=50>
</APPLET>
```

Esta marca html llama al applet SuCodigo.class y establece su ancho y alto inicial. Cuando se acceda a la página Web donde se encuentre incluida la marca, se ejecutará el byte-code contenido en SuCodigo.class, obteniéndose el resultado de la ejecución del applet en la ventana del navegador, con soporte Java, que estemos utilizando.

10.3.2 Prueba de un Applet

El JDK, Kit de Desarrollo de Java, incluye el visor de applets básico, appletviewer, que puede utilizarse para la visualización rápida y prueba de nuestros applets, tal como se ha visto ya. La ejecución de un applet sobre appletviewer se realiza a través de la llamada:
appletviewer fichero.html
En nuestro caso el fichero con el código html que ejecutará nuestro applet HolaMundo es HolaMundo.html que generará la salida que se mostraba en la sección sobre el Ejemplo de uso de appletviewer.

10.3.3 La Marca Applet de HTML

Dado que los applets están mayormente destinados a ejecutarse en navegadores Web, había que preparar el lenguaje HTML para soportar Java, o mejor, los applets. El esquema de marcas de HTML, y la evolución del estándar marcado por Netscape hicieron fácil la adición de una nueva marca que permitiera, una vez añadido el correspondiente código gestor en los navegadores, la ejecución de programas Java en ellos.

La sintaxis de las etiquetas <APPLET> y <PARAM> es la que se muestra a continuación y que iremos explicando en párrafos posteriores:

```
<APPLET CODE= WIDTH= HEIGTH= [CODEBASE=] [ALT=]
[NAME=] [ALIGN=] [VSPACE=] [HSPACE=]>
<PARAM NAME= VALUE= >
</APPLET>
```

Atributos Obligatorios:
CODE : Nombre de la clase principal
WIDTH : Anchura inicial
HEIGHT : Altura inicial

Atributos opcionales:
CODEBASE : URL base del applet
ALT : Texto alternativo
NAME : Nombre de la instancia
ALIGN : Justificación del applet
VSPACE : Espaciado vertical
HSPACE : Espaciado horizontal

10.3.4 Atributos de Applet

Los atributos que acompañan a la etiqueta <APPLET>, algunos son obligatorios y otros son opcionales. Todos los atributos, siguiendo la sintaxis de html, se especifican de la forma: atributo=valor.

Los atributos obligatorios son:
CODE: Indica el fichero de clase ejecutable, que tiene la extensión .class. No se permite un URL absoluto, como ya se ha dicho, aunque sí puede ser relativo al atributo opcional CODEBASE.
WIDTH: Indica la anchura inicial que el navegador debe reservar para el applet en pixels.
HEIGHT: Indica la altura inicial en pixels. Un applet que disponga de una geometría fija no se verá redimensionado por estos atributos. Por ello, si los atributos definen una zona menor que la que el applet utiliza, únicamente se verá parte del mismo, como si se visualiza a través de una ventana, eso sí, sin ningún tipo de desplazamiento.

Los atributos opcionales que pueden acompañar a la marca APPLET comprenden los que se indican a continuación:
CODEBASE: Se emplea para utilizar el URL base del applet. En caso de no especificarse, se utilizará el mismo que tiene el documento.
ALT: Como ya se ha dicho, funciona exactamente igual que el ALT de la marca <IMG>, es decir, muestra un texto alternativo, en este caso al applet, en navegadores en modo texto o que entiendan la etiqueta APPLET pero no implementen una máquina virtual Java.
NAME: Otorga un nombre simbólico a esta instancia del applet en la página que puede ser empleado por otros applets de la misma página para localizarlo. Así, un applet puede ser cargado varias veces en la misma página tomando un nombre simbólico distinto en cada momento.
ALIGN: Se emplea para alinear el applet permitiendo al texto fluir a su alrededor. Puede tomas los siguientes valores: LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM y ABSBOTTOM.
VSPACE: Indica el espaciado vertical entre el applet y el texto, en pixels. Sólo funciona cuando se ha indicado ALIGN = LEFT o RIGHT.
HSPACE: Funciona igual que el anterior pero indicando espaciamiento horizontal, en pixels. Sólo funciona cuando se ha indicado ALIGN = LEFT o RIGHT.
Es probable encontrar en algunas distribuciones otras etiquetas para la inclusión de applets, como <APP>. Esto se debe a que estamos ante la tercera revisión de la extensión de HTML para la

incrustación de applets y ha sido adoptada como la definitiva. Por ello, cualquier otro medio corresponde a implementaciones obsoletas que han quedado descartadas.

10.3.5 Paso de Parámetros a Applets

El espacio que queda entre las marcas de apertura y cierre de la definición de un applet, se utiliza para el paso de parámetros al applet. Para ello se utiliza la marca PARAM en la página HTML para indicar los parámetros y el método getParameter() de la clase java.applet.Applet para leerlos en el código interno del applet. La construcción puede repetirse cuantas veces se quiera, una tras otra.

Los atributos que acompañan a la marca PARAM son los siguientes:
NAME: Nombre del parámetro que se desea pasar al applet.
VALUE: Valor que se desea transmitir en el parámetro que se ha indicado antes.
Para mostar esta posibilidad vamos a modificar nuestro applet básico HolaMundo para que pueda saludar a cualquiera. Lo que haremos será pasarle al applet el nombre de la persona a quien queremos saludar. Generamos el código para ello y lo guardamos en el fichero HolaTal.java

```
import java.awt.Graphics;
import java.applet.Applet;

public class HolaTal extends Applet {
    String nombre;

    public void init() {
      nombre = getParameter( "Nombre" );
      }

    public void paint( Graphics g ) {
      g.drawString( "Hola "+nombre+"!",25,25 );
      }
    }
```

Si compilamos el ejemplo obtendremos el fichero HolaTal.class que incluiremos en nuestra página Web. Vamos a generar el fichero HolaTal.html, en el que incluiremos nuestro applet, y que debería tener el siguiente contenido:

```
<HTML>
<APPLET CODE=HolaTal.class WIDTH=300 HEIGHT=100>
<PARAM NAME="Nombre" VALUE="Federico">
</APPLET>
</HTML>
```

Por supuesto, que pueden sustituir mi nombre por el de ustedes. Este cambio no afectará al código Java, no será necesario recompilarlo para que los salude el applet.

Los parámetros no se limitan a uno solo. Se puede pasar al applet cualquier número de parámetros y siempre hay que indicar un nombre y un valor para cada uno de ellos.

El método getParameter() es fácil de entender. El único argumento que necesita es el nombre del parámetro cuyo valor queremos recuperar. Todos los parámetros se pasan como Strings, en caso de necesitar pasarle al applet un valor entero, se ha de pasar como String, recuperarlo como tal y luego convertirlo al tipo que deseemos. Tanto el argumento de NAME como el de VALUE deben ir colocados entre dobles comillas (") ya que son String.

El hecho de que las marcas <APPLET> y <PARAM> sean ignoradas por los navegadores que no

entienden Java, es inteligentemente aprovechado a la hora de definir un contenido alternativo a ser mostrado en este último caso. Así la etiqueta es doble:

```
<APPLET atributos>
parámetros
contenido alternativo
</APPLET>
```

Nuestro fichero para mostrar el applet de ejemplo lo modificaremos para que pueda ser visualizado en cualquier navegador y en unos casos presente la información alternativa y en otros, ejcute nuestro applet:

```
<HTML>
<APPLET CODE=HolaTal.class WIDTH=300 HEIGHT=100>
<PARAM NAME="Nombre" VALUE="Federico">
No verás lo bueno hasta que consigas un navegador
<I>Java Compatible</I>
</APPLET>
</HTML>
```

10.3.6 Tokens en Parámetros de Llamada

Ya de forma un poco más avanzada vamos a ver como también se pueden pasar varios parámetros en la llamada utilizando separadores, o lo que es lo mismo, separando mediante delimitadores los parámetros, es decir, tokenizando la cadena que contiene el valor del parámetro, por ejemplo:

```
<PARAM NAME=Nombre VALUE="Federico|Guillermo">
```

En este caso el separador es la barra vertical "|", que delimita los dos tokens, pero también podemos redefinirlo y utilizar cualquier otro símbolo como separador:

```
<PARAM NAME=Separador VALUE="#">
<PARAM NAME=Nombre VALUE=" Federico#Guillermo ">
```

Si ahora intentamos cambiar de color de fondo en que aparecen los textos en el applet, utilizando el mismo método, podríamos tener:

```
<PARAM NAME=Nombre VALUE=" Federico|Guillermo ">
<PARAM NAME=Color VALUE="green|red">
```

Es más, podríamos hacer que parpadeasen los mensajes en diferentes colores, cambiando el color de fondo y el del texto:

```
<PARAM NAME=Nombre1 VALUE=" Federico|green|yellow">
<PARAM NAME=Nombre2 VALUE=" Guillermo|red|white">
```

Para recoger los parámetros pasados en este último caso, bastaría con hacer un pequeño bucle de lectura de los parámetros que deseamos:

```
for( int i=1; ; i++ )
    p = getParameter( "Nombre"+i );
    if( p == null )
```

```
        break;
    . . .
    }
```

incluso podríamos utilizar un fichero para pasar parámetros al applet. La llamada sería del mismo tipo:

```
<PARAM NAME=Fichero VALUE="FicheroDatos">
```

y el FicheroDatos debería tener un contenido, en este caso, que sería el siguiente:

```
Federico
    fondoColor=green
    textoColor=yellow
    fuente=Courier
    fuenteTam=14
Guillermo
    fondoColor=red
    textocolor=white
```

E incluso ese FicheroDatos, podríamos hacer que se encontrase en cualquier URL, de forma que utilizando el método getContent() podríamos recuperar el contenido del fichero que contiene los parámetros de funcionamiento del applet:

```
String getContent( String url ) {
    URL url = new URL( null,url );

    return( (String).url.getContent() );
    }
```

Para recuperar los parámetros que están incluidos en la cadena que contiene el valor podemos utilizar dos métodos:

```
StringTokenizer( string,delimitadores )
StreamTokenizer( streamentrada )
```

Así en la cadena Agustin|Antonio si utilizamos el método:

```
StringTokenizer( cadena,"|" );
```

obtenemos el token Federico, el delimitador "|" y el token Guillermo. El código del método sería el que se muestra a continuación:

```
// Capturamos el parámetro
p = getParameter( "p" );

// Creamos el objeto StringTokenizer
st = new StringTokenizer( p,"|" );

// Creamos el almacenamiento
cads = new String[ st.countTokens() ];

// Separamos los tokens de la cadena del parámetro
for( i=0; i < cads.length; i++ )
cadenas[i] = st.nextToken();
```

En el caso de que utilicemos un fichero como verdadera entrada de parámetros al applet y el fichero se encuentre en una dirección URL, utilizamos el método StreamTokenizer() para obtener los tokens que contiene ese fichero:

```
// Creamos el objeto URL para acceder a él
url = new URL( "http://www.prueba.es/Fichero" );

// Creamos el canal de entrada
ce = url.openStream();

// Creamos el objeto StreamTokenizer
st = new StreamTokenizer( ce );

// Capturamos los tokens
st.nextToken();
```

10.3.7 El Parámetro Archive

Una de las cosas que se achacan a Java es la rapidez. El factor principal en la percepción que tiene el usuario de la velocidad y valor de los applets es el tiempo que tardan en cargarse todas las clases que componen el applet. Algunas veces tenemos que estar esperando más de un minuto para ver una triste animación, ni siquiera buena. Y, desafortunadamente, esta percepción de utilidad negativa puede recaer también sobre applets que realmente sí son útiles.

Para entender el porqué de la necesidad de un nuevo método de carga para acelerarla, necesitamos comprender porqué el método actual es lento. Normalmente un applet se compone de varias clases, es decir, varios ficheros .class. Por cada uno de estos ficheros .class, el cargador de clases debe abrir una conexión individual entre el navegador y el servidor donde reside el applet. Así, si un applet se compone de 20 ficheros .class, el navegador necesitará abrir 20 sockets para transmitir cada uno de los ficheros. La sobrecarga que representa cada una de estas conexiones es relativamente significante. Por ejemplo, cada conexión necesita un número de paquetes adicionales que incrementan el tráfico en la Red.

Me imagino que ya habrán pensado la solución al problema: poner todos los ficheros en uno solo, con lo cual solamente sería necesaria una conexión para descargar todo el código del applet. Bien pensado. Esto es lo mismo que han pensado los dos grandes competidores en el terreno de los navegadores, Netscape y Microsoft.

Desafortunadamente, las soluciones que han implementado ambas compañías no son directamente compatibles. Microsoft, en su afán de marcar diferencia, crea su propio formato de ficheros CAB. La solución de Netscape es utilizar el archiconocido formato ZIP. Por suerte, nosotros podemos escribir nuestro código HTML de forma que maneje ambos formatos, en caso necesario. Esto es así porque podemos especificar cada uno de estos formatos de ficheros especiales en extensiones separadas de la marca <APPLET>.

No vamos a contar la creación de ficheros CAB; quien esté interesado puede consultar la documentación de Java que proporciona Microsoft con su SDK para Java, que es bastante exhaustiva al respecto. Una vez que disponemos de este fichero, podemos añadir un parámetro CABBASE a la marca <APPLET>:

```
<APPLET NAME="Hola" CODE="HolaMundo" WIDTH=50 HEIGHT=50 >
<PARAM NAME=CODEBASE VALUE="http://www.ejemplo.es/classes">
<PARAM NAME=CABBASE VALUE="hola.cab">
</APPLET>
```

El VALUE del parámetro CABBASE es el nombre del fichero CAB que contiene los ficheros .class que componen el conjunto de applet.
Crear un archivo ZIP para utilizarlo con Netscape es muy fácil. Se deben agrupar todos los ficheros .class necesarios en un solo fichero .zip. Lo único a tener en cuenta es que solamente hay que almacenar los ficheros .class en el archivo; es decir, no hay que comprimir.
Si se está utilizando pkzip, se haría:

```
Pkzip -e0 archivo.zip listaFicherosClass
```

El parámetro de la línea de comandos es el número cero, no la "O" mayúscula.
Para utilizar un fichero .zip hay que indicarlo en la marca ARCHIVE de la sección <APPLET>:

```
<APPLET NAME="Hola" CODE="HolaMundo" WIDTH=50 HEIGHT=50
 CODEBASE VALUE="http://www.ejemplo.es/classes"
 ARCHIVE="hola.zip">
</APPLET>
```

Pero hay más. Podemos crear ambos tipos de ficheros y hacer que tanto los usuarios de Netscape Navigator como los de Microsoft Internet Explorer puedan realizar descargas rápidas del código del applet. No hay que tener en cuenta los usuarios de otros navegadores, o de versiones antiguas de estos dos navegadores, porque ellos todavía podrán seguir cargando los ficheros a través del método lento habitual. Para compatibilizarlo todo, ponemos las piezas anteriores juntas:

```
<APPLET NAME="Hola" CODE="HolaMundo" WIDTH=50 HEIGHT=50
 CODEBASE VALUE="http://www.ejemplo.es/classes"
 ARCHIVE="hola.zip">
<PARAM NAME=CABBASE VALUE="hola.cab">
<PARAM NAME=CODEBASE VALUE="http://www.ejemplo.es/classes">
<PARAM NAME=CABBASE VALUE="hola.cab">
</APPLET>
```

Ahora que se puede hacer esto con ficheros .cab y .zip, JavaSoft ha definido un nuevo formato de ficheros, que incorporará en del JDK 1.1, para incluir juntos todos los ficheros de imágenes, sonido y class. JavaSoft llama a esto formato JAR (Java Archive). La marca <APPLET> de HTML se modificará para manejar este nuevo formato JAR a través del parámetro ARCHIVES. Y dejamos al lector el trabajo de poner los tres formatos juntos bajo el mismo paraguas de la marca <APPLET>.

10.4 Ciclo de Vida de un Applet

Para seguir el ciclo de vida de un applet, supondremos que estamos ejecutando en nuestro navegador el applet básico HolaMundo, a través de la página HTML que lo carga y corre.
Lo primero que aparece son los mensajes "initializing... starting...", como resultado de la carga del applet en el navegador. Una vez cargado, lo que sucede es:

1. Se crea una instancia de la clase que controla al Applet

2. El applet se inicializa a si mismo

3. Comienza la ejecución del applet

4. Cuando se abandona la página, por ejemplo, para ir a la siguiente, el applet detiene la ejecución. Cuando se regresa a la página que contiene el applet, se reanuda la ejecución.Si se utiliza la opción del navegador de Reload, es decir, volver a cargar la página, el applet es descargado y vuelto a cargar. El applet libera todos los recursos que hubiese acaparado, detiene su ejecución y ejecuta su finalizador para realizar un

proceso de limpieza final de sus trazas. Después de esto, el applet se descarga de la memoria y vuelve a cargarse volviendo a comenzar su inicialización.

5. Finalmente, cuando se concluye la ejecución del navegador, o de la aplicación que está visualizando el applet, se detiene la ejecución del applet y se libera toda la memoria y recursos ocupados por el applet antes de salir del navegador.

10.5 Aplicaciones

10.5.1 La Aplicación Fecha

Veamos ahora una aplicación un poco más útil que HolaMundo, presentaremos en pantalla la fecha y hora del sistema. Aprovecharemos también para realizar un introducción muy sencilla a los conceptos fundamentales de la programación orientada a objetos, clases y objetos, a través de esta simple aplicación.

```
import java.util.Date;

class FechaApp {
    public static void main( String args[] ) {
    Date hoy = new Date();
    System.out.println( hoy );
    }
}
```

Esta aplicación es una versión modificada de HolaMundoApp de la que difiere porque se importa la clase Date, la aplicación se llama ahora FechaApp en vez de HolaMundoApp, se crea un objeto Date y el mensaje de salida a pantalla es diferente. Almacenaremos esta nueva aplicación en el fichero FechaApp.java.

La aplicación FechaApp es el programa más simple que podemos hacer que realice algo interesante, pero por su misma sencillez no necesita ninguna clase adicional. Sin embargo, la mayoría de los programas que escribamos serán más complejos y necesitarán que escribamos otras clases y utilizar las que nos proporciona Java como soporte.

Nuestra aplicación FechaApp utiliza dos clases, la clase System y la clase Date, que nos proporciona el entorno de desarrollo de Java. La clase System proporciona un acceso al sistema independiente del hardware sobre el que estemos ejecutando la aplicación y la clase Date proporciona un acceso a las funciones de Fecha independientemente del sistema en que estemos ejecutando la aplicación.

Los applets anteriores son simplemente ejemplos de lo que se puede conseguir con Java. Son un poco más complicados que los desarrollados en el Tutorial, pero tampoco tan complicados como para que no se pueda seguir adecuadamente el código fuente.

Se han desarrollado sobre Windows'95, por lo que no estoy seguro de que en otros sistemas se visualice todo correctamente, sobre todo teniendo en cuenta que no todas las fuentes de caracteres están disponibles siempre. Si se desea hacer un uso comercial de estos applets, habría que añdirles más detalles de configuración y, sobre todo, capturar todas las excepciones posibles, porque lo que en un sitio va bien, en otro puede generar una excepción

10.5.2 Etiqueta

Vamos a desarrollar un nuevo Componente para el AWT, para mejorar el Label que se proporciona con el AWT, añadiéndole al original cosas como bordes, efectos de sombreado, etc., o posibilidad de

fijarle características como el color o el font de caracteres con que se presentará en pantalla.

En el código fuente de la clase, Etiqueta.java, hemos introducido gran cantidad de comentarios, por lo que explicaciones aquí serían redundantes, en la mayoría de las sentencias para que se vaya siguiendo paso a paso la construcción y entendiendo qué hace cada uno de los métodos y, para que resulte sencillo al lector el añadir nuevos tipos de efectos sobre el texto de la Etiqueta.

El applet EjEtiqueta.java, presenta las tres posibilidades de presentar texto en pantalla que hemos implementado: Texto normal, resaltado y hundido. A continuación se muestra el resultado de la ejecución del applet.

10.5.3 Reloj Digital

Seguramente, siempre que hagamos una aplicación con una envergadura mínima, necesitaremos un temporizador. Así que vamos a implementar uno que llegue a resolución de segundos; porque, también es casi seguro, que en una aplicación normal, no necesitaremos más precisión.

Vamos a implementar nuestro temporizador, Timer.java, de la forma más general, haciendo que tenga un time-out, que se pueda resetear automática o manualmente y lo vamos a construir como un thread individual, con lo cual, podremos tener tantos timers como queramos.

La interfase Temporizador, Temporizador.java, nos proporciona los métodos que necesitamos para implementar el funcionamiento del Timer, es decir, que consideraremos al Timer como un elemento observable, de forma que notifique a los observadores cuando algún evento ocurra, como puede ser el arranque del timer, la pérdida o su muerte. Así, si una clase determinada quiere ser avisada cuando le ocurra algo al timer, debe implementar la interfase Temporizador, dejando vacíos los métodos que corresponden a los eventos que no le interese recoger.

La interfase Temporizador consiste en cuatro métodos, de los cuales los más interesante son timerMuerto(), que se llama cuando la duración del timer se ha cumplido y, timerIntervalo(), que ocurre a intervalos regulares, según el tiempo que hayamos fijado.

Si una clase necesita más de un timer, puede implementar esta interfase, pero entonces necesitaríamos incorporar a la clase que la use métodos para que fije e indique su nombre; de este modo, los timers estarán identificados dentro de la interface Temporizador y no habrá problemas al estar varios timers corriendo a la vez.

El applet que presentamos, RelojDigital.java, simplemente recoge la hora del sistema y la va actualizando cada segundo utilizando un timer. Hacemos uso también del componente Etiqueta que hemos desarrollado en otro ejemplo, para presentar el texto con la hora en la pantalla.

10.5.4 Persiana

Este applet presenta información desplazándose a través de la zona de pantalla que constituye el applet. Podemos incorporar cualquier tipo de información para que sea presentada en pantalla, tal como se muestra en el ejemplo de esta misma página. Se pueden conseguir múltiples efectos, a través de la selección de fuentes y utilización de colores.

El texto que se presenta en la pantalla va incluido en el fichero de código fuente, Persiana.java, y no en la llamada al applet. Además, se puede incorporar un dibujo de fondo, que será presentado a la vez que se ejecute el applet. El código de este applet es muy sencillo y fácil de comprender, a pesar de que el efecto que se visualiza en pantalla sugiera una programación complicada.

10.5.5 Solapas

Este applet nos muestra cómo se pueden incorporar nuevos Componentes al AWT. Hemos diseñado un sistema de solapas o Tabs, como se suelen reconocer, para incorporar a nuestras aplicaciones. En el fichero EjFichas.java, se encuentra el código que muestra cómo se usa y que corresponde a la

ejecución del applet que aparece en esta misma página.

Los ficheros de código fuente que se emplean para la implementación del sistema de solapas propuesto son:
CuerpoFicha.java

Se encarga de presentar el cuerpo de la ficha que corresponde a una solapa. Nos permite incorporar cualquier otro elemento que deseemos a esa ficha, tal como muestra el ejemplo, en el cual hemos añadido diferentes componentes a cada una de las fichas que se abren al seleccionar las solapas.
Solapa.java

Se encarga de gestionar cada una de las solapas. Nos indica cuando se ha pulsado en una solapa y tiene implementados los métodos necesarios para indicarnos cuando una solapa está selecciona o no, cuando se pica con el ratón dentro de una solapa, o para resaltar la solapa cuando se selecciona, entre otros.
Ficha.java

Este fichero contiene una clase compuesta de dos Objetos, uno de clase Solapa y otro de clase CuerpoFicha. Se encarga de gestionar la presentación de la Ficha al completo, de forma que cuando se selecciona una solapa, se presenta el contenido del objeto CuerpoFicha que tiene asociado ese objeto Solapa que se ha elegido.
ListaFichas.java

Contiene un objeto Vector con todas las fichas que hayamos incorporado al applet. Aquí es donde podemos indicar que el applet contendrá más o menos Fichas, que estarán representadas por las Solapas que el usuario podrá elegir con el ratón.

10.5.6 Transparencia

Vamos a mostrar en esta aplicación un ejemplo de las nuevas técnicas que se están desarrollando actualmente. La técnica es muy sencilla, consiste en tener un dibujo grande con varias imágenes de la zona que queremos reproducir en pantalla, y utilizando los eventos del ratón, presentamos una imagen u otra. Primero desarrollamos una clase para que controle el repintado de las distintas imágenes, AppletFijo.java, que se van a ir presentando y que no se repinte la zona total del applet, lo que produciría un desagradable efecto de parpadeo. Luego extenderemos esta clase para la imagen completa, creando instancias de objetos BotonImg (BotonImg.java), que son los que van a controlar cada una de las opciones. Funcionan como botones normales, sólo que cotnrolan de forma exhaustiva los eventos que llegan del ratón, BotonImg.java. Finalmente, en la clase MenuDeBotones (MenuDeBotones.java) es donde creamos la clase principal, cargando la imagen de fondo y recuperando todos los parámetros que se pasen en la llamada al applet.

Como podrá observarse, el código es muy sencillo y fácil de entender, y el resultado es altamente profesional (tanto, que hasta Sun lo utiliza).

10.5.7 Calculadora

El applet Calculadora.java presenta una calculadora básica, tomando como herramienta de construcción el AWT. Es una calculadora que realiza las operaciones más habituales, tipo calculadora solar, pero que ilustra perfectamente los mecanismos de presentación en pantalla de Componentes del AWT sobre un layout propio, ya que todos los botones se colocan en posiciones determinadas.

Las operaciones matemáticas no son motivo de estudio, pero sí se podría completar la calculadora con más funciones y hacerla semejante a las que se proporcionan con casi todos los entornos gráficos. La clase BotonCalculadora es la fundamental en este ejemplo, ya que nos permite colocar botones en cualquier sitio y con cualquier tamaño.

10.6.6 Cuenta-Kilómetros

El applet Contador.java presenta una imagen conocida de cuenta-kilómetros de coche, pero cuyo código nos sirve para demostrar varias cosas. Podemos comprobar cómo se manipula el offset de un dibujo porque la imagen de los números es un fichero gráfico que contiene los 10 dígitos posibles, y el contador va seleccionando cada uno de ellos en función de la cifra que tiene que representar el cuanta-kilómetros. También podemos comprobar la utilización de threads para la implementación del intervalo de tiempo que tarda el cuenta-kilómetros en pasar de una cifra a la siguiente.

En la llamada al applet, podemos indicar el número de dígitos que va a contener el cuenta-kilómetros, el intervalo de tiempo que transcurrirá entre los cambios de cifras y el valor inicial en que arranca el contador. Además, como es un applet muy sencillo, lo hemos adornado con algunas opciones, por ejemplo:
Con la tecla "+" aceleramos el contador
Con la tecla "-" disminuimos la velocidad con que cambian los números en el cuenta-kilómetros
Con la tecla "0" ponemos a cero el contador
Utilizando el ratón, si picamos una vez sobre el cuenta-kilómetros, se detiene la cuenta, y si picamos una segunda vez, vuelve a seguir contando