Objetivos Hilos, applets y firma de código.

Índice

1. Applets

Un applet consta de una o varias clases y recursos que se ejecutan dentro de un navegador Web. Las tareas que un applet puede realizar están limitadas por razones de seguridad. Mientras el applet se ejecuta, el sistema de seguridad en tiempo de ejecución de Java analiza el código.

Las mayores limitaciones de los applets son:

- Un applet no puede acceder al disco local (limitación de seguridad).
- Un applet no debe tardar mucho en mostrarse (limitación de tamaño).

y sus ventajas:

- No hay que preocuparse por la instalación. La carga de la aplicación se realiza de forma automática cada vez que el usuario visite la página Web que contiene el applet. Por lo tanto sólo habría que actualizar la aplicación en el servidor.
- Debido a las limitaciones de seguridad, es más complicado que código erróneo cause daños al sistema del usuario.

Salvo estas consideraciones, un applet no es diferente de cualquier otra aplicación y se pueden insertar componentes de javax.swing, tratamiento de eventos, dibujo, etc.

Un patrón habitual en Java es el de extender una clase de la biblioteca de Java e implementar los métodos apropiados. Por ejemplo, una forma de crear un hilo es extendiendo Thread y sobrescribiendo el método run() que se define en la superclase. Otro ejemplo lo encontramos a la hora de crear una aplicación que muestre una ventana, en este caso hay que extender la clase JFrame y en el constructor crear los componentes deseados. Uno más se ha visto a la hora de tratar eventos en las interfaces gráficas de usuario, se extiende la clase apropiada (MouseAdapter, WindowAdapter,...) y se sobrescriben los métodos apropiados.

Los applets siguen este esquema. Para crear un applet se extenderá la clase JApplet del paquete javax.swing y se sobreescribirán los métodos apropiados. El plug-in instalado en el navegador espera una referencia del tipo JApplet y llamará a sus métodos (en realidad a los métodos que hallamos sobreescrito) al cargar el applet, al cerrar la ventana del navegador...
Los métodos que controlan la creación, ejecución, visualización y destrucción de un applet en una página Web se describen a continuación:

Método	Descripción
init()	Es un método que hereda de la clase java.applet.Applet. Es llamado por el navegador o visor de applets para cargar el applet y sólo se llama una vez. Es llamado antes del método start(). Una subclase debe sobreescribir este método si realiza alguna tarea de iniciación ya que por defecto este método no realiza nada. Por ejemplo, un applet con hilos podría usar este método para crear los hilos. Si el applet tiene componentes (botones, campos de texto, etc) se deben crear y añadir dentro de este método.
start()	Es un método que hereda de la clase java.applet.Applet. Se le llama después del método init() y cada vez que se hace visible la página que contiene el applet. Una subclase de JApplet debe sobreescribir este método si hay alguna operación que se debe realizar cada vez que la página Web que contiene el applet se active (por ejemplo al pasar de segundo plano a primer plano), ya que por defecto este método no realiza nada.
stop()	Es un método que hereda de la clase java.applet.Applet. Es llamado por el navegador o visor de applets para informar al applet de que debe parar su ejecución. Es llamado cuando la página Web que contiene el applet pasa a segundo plano y justo antes de que el applet sea destruido. Una subclase de JApplet debe sobreescribir este método si hay alguna operación que se debe detener cuando la página Web no sea visible, ya que la implementación por defecto no hace nada. Por ejemplo, un applet con animación puede utilizar start() para continuar con la animación y stop() para detenerla.
destroy()	Es un método que hereda de la clase java.applet.Applet. Es llamado por el navegador o visor de applets para informar al applet de que debe destruir cualquier recurso que haya reservado. El método stop() es llamado antes de llamar a éste. Una subclase de JApplet debe sobreescribir este método si hay alguna operación que se desee realizar antes de que sea eliminado, ya que la implementación por defecto no hace nada.

No se requiere que los applets tengan un método main(), todo el código de inicialización se escribe en el método init() que es llamado al cargarse el applet en el navegador.

La página que contenga el applet debe utilizar la etiqueta <APPLET>. Esta etiqueta admite los atributos CODE, HEIGHT, WIDTH, ALIGN, ALT, ARCHIVE, OBJECT, CODEBASE, HSPACE y VSPACE. El atributo CODE indica cual es la clase que extiende a JApplet. La anchura y altura del applet se determina mediante los atributos

WIDTH y HEIGHT respectivamente. El atributo ARCHIVE indica el fichero jar donde se deben buscar las clases y demás recursos. CODEBASE indica la ruta relativa a las clases que definen el applet desde donde se encuentra la página html.

A modo de ejemplo, el código que se muestra a continuación supone que se tiene una clase A.class que es la que hereda de JApplet y que esta clase junto con todas las demás (si las hay) y los recursos que utiliza el applet se encuentran en el fichero fichero.jar.

```
<HTML>
<HEAD>
   <TITLE>Un Applet</TITLE>
</HEAD>
<BODY>
   <HR>
      <center>
         <APPLET code="A.class" ARCHIVE="fichero.jar" WIDTH="420" HEIGHT="400"></APPLET>
      </center>
   </HR>
</BODY>
</HTML>
```

El código anterior se ejecuta en el navegador y no en el JRE (Java Runtime Environment) de Sun. Para forzar a que el applet se ejecute en el JRE de Sun se recomienda utilizar esta otra estructura que es válida tanto para Internet Explorer como para Netscape. Quien desee ampliar información está disponible en:

http://java.sun.com/j2se1.4.1/docs/guide/plugin/developer_guide/using_tags.html

```
<HTML>
<HEAD>
   <TITLE>Un Applet</TITLE>
</HEAD>
<BODY>
   <HR>
   <OBJECT classid="clsid:8AD9C840 - 044E - 11D1 - B3E9 - 00805F499D93"
           width="200" height="200" align="baseline"
           codebase="http://java.sun.com/products/plugin/autodl/jinstall - 1 4 1 03 - windows - i586.cab">
           <PARAM NAME="code" VALUE="A.class">
           <PARAM NAME="archive" VALUE="fichero.jar">
           <PARAM NAME="type" VALUE="application/x - java - applet;jpi - version=1.4">
           <COMMENT>
              <EMBED type="application/x - java - applet;jpi - version=1.4" width="200"
                height="200" align="baseline" code="A.class"
                archive="fichero.jar"
                pluginspage="http://java.sun.com/products/plugin/1.4/plugin - install.html">
                <NOEMBED>
                     No Java 2 SDK, Standard Edition v 1.4 support for APPLET!!
                </NOEMBED>
              </EMBED>
           </COMMENT>
   </OBJECT>
   <HR>
</BODY>
</HTML>
```

NOTA: Sun dispone de una utilidad (HTMLConverter) para obtener un fichero HTML modificado a partir de uno simple que contenga el applet.

Supongamos que ya se dispone de un applet y de un fichero HTML con el c´ odigo necesario para cargar el applet. Existen dos formas de visualizarlo:

- Se puede usar un navegador que esté habilitado para ejecutar java.

- Usando la utilidad appletwiewer. Esta es una utilidad que viene con el JDK (en el directorio bin) y que nos sirve para visualizar el applet cargando el fichero html.

Como ejemplo, en la página http://www.jhu.edu/ signals/index.html hay un listado de applets relacionados con las señales y sistemas. En concreto en la página

http://www.jhu.edu/ signals/discreteconv2/index.html

hay un applet que muestra el efecto de pasar una señal por un sistema en el dominio discreto. Este applet se puede visualizar mediante appletviewer utilizando la siguiente instrucción:

appletviewer http://www.jhu.edu/ signals/discreteconv2/index.html

2. La utilidad jar

Además disponemos (esto no solo está relacionado con applets) de la utilidad jar para el empaquetamiento de archivos. Este programa que se distribuye con el JDK sirve para empaquetar varios ficheros (ficheros compilados de java, sonidos, imágenes, ...) en un único fichero con extensión jar. Cuando un navegador carga una página que contiene varias imágenes, varias clases de java, etc. tiene que realizar una conexión para cada uno de estos elementos, aumentando el tiempo necesario para mostrar la página completa. En estos casos se aconseja que se use jar para comprimir y empaquetar toda la información que necesita el applet en un único fichero.

Supongamos que tenemos varias clases que forman el applet (A.class, B.class, C.class) y una serie de imágenes y sonidos (sonido1.au, sonido2.au, imagen1.gif e imagen2.gif). Para empaquetarlo todo en un único fichero

```
jar cvf fichero.jar A.class B.class C.class sonido1.au sonido2.au imagen1.gif imagen2.gif
```

Si queremos extraer todos los ficheros que contiene un fichero jar:

```
jar xvf fichero.jar
```

3. Código firmado

Firmar código significa añadir información a un fichero jar sobre la procedencia del fichero. Firmando el código se puede conseguir que quien ejecute el código sepa de donde procede y pueda confiar en que el código no realiza una tarea dañina para el sistema y si lo hace, se sabe quien lo ha realizado.

Uno de los formatos más comunes para los certificados firmados es el X.509. Estos certificados son utilizados por Verisign, Microsoft, Netscape (por citar algunas) para firmar e-mail, autenticar el código de los programas y certificar datos. Un certificado contiene información sobre el nombre del firmante, el periodo de validez de la firma, la clave pública de la firma, etc.

El JDK incluye el programa keytool que permite generar y manipular un conjunto de certificados. Este programa trabaja con almacenes de claves (son ficheros que siguen un formato específico). Cada entrada en el almacén de claves tiene un **alias**.

Escenario	Vamos a suponer que Juan desea firmar código. Lo primero que debe hacer es crear un almacén de claves (cada entrada contiene un par de claves una privada que no se distribuye y una pública

que se debe distribuir). El almacén de claves debe colocarse en un sitio seguro y estar protegido por password. Cuando Juan firme un fichero y lo distribuya, los usuarios que deseen verificar la identidad del firmante deberán disponer de la clave pública. Para ello Juan debe extraer del almacén el certificado con la clave pública. Vamos a suponer que el usuario Jesús recibe de Juan el fichero firmado. Para comprobar la identidad del firmante deberá tener el fichero con el certificado.

Vamos a ver en detalle cómo se desarrollan cada una de las situaciones descritas en este escenario.

Creación del almacén de claves.

Vamos a ver cómo puede añadir Juan una nueva entrada en un almacén llamado LP.almacen para el alias Juan:

```
keytool - genkey - keystore LP.almacen - alias Juan
```

Al ejecutar esta instrucción el programa pide más información sobre esta clave:

```
Escriba la contraseña del almacén de claves: password

¿Cuales son su nombre y su apellido?
[ Unknown]: Juan
¿Cual es el nombre de su unidad de organizacion?
[ Unknown]: Informatica
¿Cual es el nombre de su organizacion?
[ Unknown]: ETSE
¿Cual es el nombre de su ciudad o localidad?
[ Unknown]: Valencia
¿Cual es el nombre de su estado o provincia?
[ Unknown]: Valencia
¿Cual es el codigo de pais de dos letras de la unidad?
[ Unknown]: ES
¿Es correcto CN=LP, OU=Informatica, O=ETSE, L=Valencia, ST=Valencia, C=ES?
[ no ]: y
Escriba la contraseña clave para <Juan>
(INTRO si es la misma contraseña que la del almacen de claves): passwordJuan
```

Al finalizar se habrá creado el almacén de claves LP.almacen.

Obtención del certificado.

Ahora hay que extraer un certificado del almacén, para ello se ejecuta la siguiente instrucción.

```
keytool - export - keystore LP.almacen - alias Juan - file juan.cert
```

Tras ejecutar esta instrucción se ha creado el fichero juan.cert que contiene el certificado.

Firma del código.

Supongamos que Juan tiene una serie de clases y desea firmarlas. Para ello las empaqueta en un fichero jar:

```
jar - cvf fichero . jar _ . class
```

Ahora, firma el fichero jar utilizando jarsigner:

```
jarsigner - keystore LP.almacen - signedjar firmado.jar fichero.jar Juan
```

Al ejecutar esta instrucción se pide el password del almacén y del alias Juan (se muestra una posible elección de passwords, no demasiado aconsejable):

```
Enter Passphrase for keystore : password
Enter key password for Juan: passwordJuan
```

Se ha creado el fichero firmado.jar.

Visualizar la información del certificado.

Supongamos que Jesús ya dispone del fichero con el certificado (Juan se lo ha proporcionado de algún modo). Para ver la información que contiene se puede ejecutar la instrucción:

```
keytool - printcert - file juan.cert
```

En la ventana aparece la siguiente información:

```
Propietario : CN=Juan, OU=Informatica, O=ETSE, L=Valencia, ST=Valencia, C=ES
Emisor: CN=Juan, OU=Informatica, O=ETSE, L=Valencia, ST=Valencia, C=ES
Numero de serie: 3fe08d7e
Valido desde: Wed Dec 17 18:08:14 CET 2003 hasta: Tue Mar 16 18:08:14 CET 2004
Huellas digitales del certificado :
MD5: C9:14:E5:96:49:A8:9F:03:97:7E:6B:E4:0E:25:E6:A5
SHA1: A3:A9:2A:05:47:43:B0:66:85:65:5E:14:13:68:1B:96:E1:94:D4:A3
```

Este certificado está autofirmado (el emisor y el propietario es el mismo). Esto significa que cualquiera podría haber generado este certificado. En situaciones reales hay terceras partes implicadas que firman los certificados asegurando que el certificado está emitido por la persona o entidad que figura en el certificado.

Importar un certificado a un almacén de claves.

Supongamos que Jesús confía en Juan y que (quizá tras haber comprobado que las huellas digitales son correctas por ejemplo llamando por teléfono a Juan) desea almacenar este certificado a un almacén de claves:

```
keytool - import - keystore jesus.almacen - alias Juan - file juan.cert
```

Al ejecutar esta instrucción se pide un password para el almacén, se nos muestra la información del certificado y se nos pide si confiamos en este certificado. Tras responder afirmativamente se crea el fichero jesus.almacen.

Verificar un archivo firmado.

Supongamos que Juan ha enviado en fichero firmado.jar a Jesús y éste desea verificarlo:

```
jarsigner - verify - keystore jesus.almacen firmado.jar
```

Al ejecutar esta instrucción aparece el mensaje: jar verified.

3.1. Applets firmados

Como ya hemos comentado, por defecto, los applets no tienen acceso a recursos de la máquina sobre la que se ejecuta por motivos de seguridad, pero un applet firmado y aceptado por el usuario puede acceder a los recursos de la máquina en la que se ejecuta.

A partir del JDK 1.2 se proporcionan una serie de utilidades para permitir la firma de applets y de aplicaciones, y para definir políticas de seguridad locales especificando en un fichero a qué recursos puede acceder el applet firmado o la aplicación.

3.1.1. Desarrollo de applet firmados

La secuencia de acciones es la siguiente (los detalles de cómo realizar cada una de estas acciones se han descrito en el apartado anterior):

1. Crear un archivo jar con las clases y demás recursos necesarios para la ejecución del applet.

2. Firmar el archivo jar utilizando un alias.

Supongamos ahora que este applet firmado junto con la página HTML que lo carga que ponen en un determinado servidor para hacerlo visible.

3.1.2. Visualización de applets firmados y permisos

Para visualizar el applet hay dos posibilidades:

- Utilizar la utilidad appletviewer

- Utilizar un navegador web.

En ambos casos es posible definir la política de seguridad que se utilizará para ejecutar el applet. La política de seguridad se puede especificar en un fichero en el que se asignan los permisos que se otorgan a un applet firmado por un alias del que hemos importado su certificado.

Vamos a ver un ejemplo de fichero de política de seguridad.

```
keystore ”file: jesus . almacen”, ”JKS”

grant SignedBy ”Juan” {
        permission java. io . FilePermission ”c: \\tmp \\- ”, ”read”;
} ;
```

- La primera línea indica cual es el almacén de claves donde se ha importado el certificado y el tipo de almacén (en este caso JKS (Java Key Store que es el tipo de almacén creado por keytool).
- La segunda línea indica que vamos a describir los permisos a otorgar al código firmado por Juan
- La tercera línea indica que se otorga permiso de lectura de ficheros en el directorio c:\tmp\

Este fichero se puede crear utilizando la utilidad policytool que viene con el JDK.

Utilizando este fichero de políticas, si el código firmado por Juan intentara escribir en c:\tmp\ o leer de otro directorio se lanzaría una excepción, ya que estaría intentando realizar una acción para la que no está autorizado. De este modo se puede tener un control muy fino sobre los permisos que se otorgan a las aplicaciones.

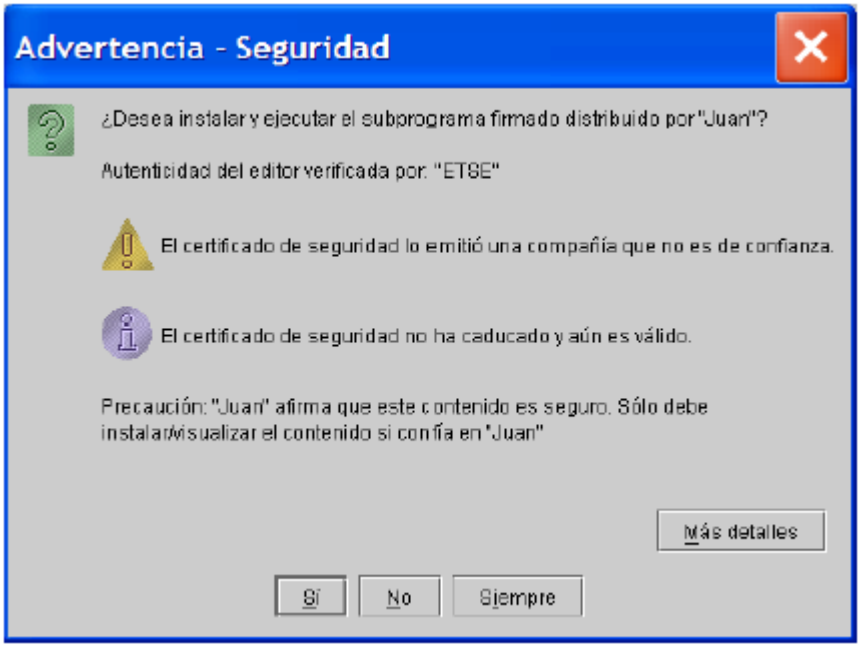

Figura 1: Mensaje que muestra el navegador al cargar un applet firmado.

Utilización de un fichero de política de seguridad con appletviewer.

appletviewer -J-Djava.security.policy=seguridad.jp AppletFirmado.html

Visualización de un applet firmado en un navegador

En el navegador en el que se visualice el applet aparecerá un mensaje como el que se muestra en la figura 1 y el usuario debe decidir si deja ejecutar el applet o no.

4. Hilos

Vamos a ver varias posibilidades a la hora de crear hilos:

4.1. Extendiendo Thread

La clase Thread contiene un metodo run() genérico que no realiza ninguna tarea. Extendiendo la clase Thread con nuestra clase deberemos sobreescribir el método run() de tal forma que realice la tarea que deseamos: ordenar datos, realizar alguna animación, ... El método run() es llamado cuando llamamos al método start() heredado de Thread.

Ejemplo: Un hilo que calcula en resto de la división, según el estándar IEEE 754, entre un número dado y todos los números comprendidos entre 1 y el número dado :

```
class Resto extends Thread {
   private int maximo;

   Resto(int max) {
              maximo=max;
   }

   public void run() {
              for (int i =1 ; i <= maximo; i++)
                System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
   }
}
```

En otra parte del programa creamos los hilos y los ejecutamos:

```
              Resto r1 = new Resto(100);
              Resto r2 = new Resto(200);
              r1. start () ;
              r2. start () ;
              ...
```

4.2. Implementando Runnable

La interfaz Runnable contiene un único método: run( ) Una clase que implemente la interfaz Runnable tiene que implementar el método run( ). Uno de los constructores de la clase Thread permite la creación de un hilo a partir de un objeto que implemente la interfaz Runnable del siguiente modo:

```
              Thread h = new Thread(Runnable target)
```

Cuando se crea un hilo de este modo, el Thread obtiene el método run( ) del objeto que implementa a la interface Runnable. El ejemplo anterior sería ahora:

```
class Resto implements Runnable {
   private int maximo;

   Resto(int max) {
              maximo=max;
   }

   public void run() {
              for (int i =1 ; i <= maximo; i++)
                System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
   }
}
```

Y donde queramos crear y ejecutar los hilos:

```
              ...
              Thread r1 = new Thread(new Resto(20));
              Thread r2 = new Thread(new Resto(100));
              r1. start () ;
              r2. start () ;
              ...
```

Otra posibilidad es lanzar el hilo desde el constructor de Resto:

```
class Resto implements Runnable {
  private int maximo;

  Resto(int max) {
            maximo=max;
            new Thread(this).start();
  }

  public void run() {
             for (int i =1 ; i <= maximo; i++)
               System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
  }
}
```

En otra parte del programa creando los objetos del tipo Resto se lanzan los hilos.

```
            ...
            new Resto(100);
            new Resto(200);
            ...
```

Finalmente, si queremos tener una referencia al hilo dentro de la clase Resto (para usarla en otra parte) modificaríamos la clase del siguiente modo:

```
class Resto implements Runnable {
  private int maximo;
  private Thread hilo;

  Resto(int max) {
            maximo=max;
            hilo = new Thread(this);
            hilo.start () ;
  }

  public void run() {
             for (int i =1 ; i <= maximo; i++)
               System.out.println("Resto = " + Math.IEEEremainder((double)maximo,(double)i));
  }
}
```

5. Sincronización y monitores

La sincronización es necesaria para evitar colisiones entre hilos cuando todos ellos comparten el acceso a un determinado recurso. Hay dos modos de sincronización:

- En una clase se puede sincronizar un método para evitar que cuando un hilo haya llamado al método ningún otro hilo pueda llamar a ningún otro método sincronizado de la clase. La siguiente línea de código muestra cómo se puede hacer que un método de una clase sea sincronizado:

  public synchronized void almacena(· · ·)

- Se puede sincronizar el acceso a un objeto a nivel de bloque de código. La siguiente línea de código muestra cómo se puede hacer que un hilo bloquee el acceso a un objeto:

  synchronized(objeto) · · ·

Un monitor es un objeto que implementa el acceso en exclusión mutua a sus métodos. En Java se aplica a los métodos de la clase declarados como synchronized. Los métodos que no sean declarados como synchronized pueden ser utilizados concurrentemente.

Dentro de un método synchronized se pueden utilizar los métodos (heredados de Object): wait(), notify() o notifyAll(). Una descripción de estos métodos se muestra en el cuadro 1.

Método	Descripción
wait()	una llamada a este método causa que el hilo (llamémosle H) que posee el monitor espere hasta que otro hilo llame a notify() o notifyAll() (si no se le pasan argumentos), o hasta que transcurra un determinado tiempo (si se le pasa un argumento con el tiempo). El hilo queda deshabilitado para propósitos de planificación de hilos y está dormido hasta que ocurra alguna de las siguientes situaciones: • algún otro hilo invoca al método notify() de este objeto y el hilo H es elegido para despertar. • Otro hilo llama al método nofifyAll() del objeto. • Otro hilo interrumpe al hilo H. • Si ha transcurrido el tiempo pasado en el argumento. El hilo H se quita del conjunto de hilos en espera y se le habilita para la planificación de los hilos compitiendo con los demás para sincronizar el objeto. Una vez que gana el control sobre el objeto continua al estado en el que estaba antes de llamar al wait(). Este método debe ser llamado por un hilo que sea el propietario del monitor del objeto. Un hilo se convierte en el propietario del monitor del objeto de alguna de las siguiente formas: • Ejecutando un método synchronized del objeto. • Ejecutando el cuerpo de una sentencia synchronized que sincroniza el objeto. • Para objetos del tipo Class ejecutando un método estático synchronized de esta clase.
notify()	Despierta un único hilo que esté esperando en el monitor de este objeto. Un hilo espera para tomar el monitor de un objeto llamando a uno de los métodos wait(· · ·). Si hay varios hilos esperando, se elige a uno de ellos. La elección es arbitraria (no se toma por ejemplo el que lleva más tiempo esperando). El hilo despertado debe competir con otros hilos por obtener el monitor ya que no tiene privilegios especiales para bloquear el objeto. Este método debe ser llamado por un hilo que sea el propietario del monitor del objeto (ver wait()).
notifyAll()	Despierta a todos los hilos que están esperando para tomar el monitor de este objeto. Un hilo espera para tomar el monitor de un objeto llamando a uno de los métodos wait(· · ·). Los hilos despertados competirán para sincronizar este objeto. Este método debe ser llamado por un hilo que sea el propietario del monitor del objeto (ver wait())

Cuadro 1: Métodos para que un objeto controle la sincronización de los hilos. Se heredan de Object

6. Tareas

Hay que desarrollar un applet cuya funcionalidad consista en la codificación en línea de texto utilizando el alfabeto fonético internacional (por ejemplo, la frase “hola pepe” se codificará en dos lineas, una por palabra, siendo para hola “HOTEL OSCAR LIMA ALFA” y para pepe “PAPA ECHO PAPA ECHO”).

El applet debe dividirse en 2 zonas, la primera contiene un JTextArea donde el usuario puede ir escribiendo las palabras. La segunda contiene otro JTextArea donde el programa va mostrando la codificacion de las palabras (el usuario no puede editar el contenido de este JTextArea).

El applet lanza un hilo (productores) por cada palabra introducida por el usuario. El hilo codifica secuencialmente cada letra de la palabra en su palabra correspondiente (por ejemplo ‘f’ por “FOXTROT”) y las irá almacenando en una LinkedList. Al finalizar la búsqueda cada hilo almacenará la LinkedList con la codificación resultante para la palabra original en una cola circular.

El applet lanza también un hilo (consumidor) que lee secuencialmente las posiciones de la cola circular. La cola circular es de tamaño fijo de tal forma que si uno de los hilos productores quiere escribir en una posición que sobrepasa el inicio de la cola, debe esperar a que el hilo consumidor lea esa posición para liberarla. El hilo consumidor debe esperar mientras que algún hilo productor escriba en la posición inicial de la cola (la posición inicial irá avanzando ya que se trata de una cola circular). Se puede controlar si algún hilo productor ha escrito algo en una posición si el contenido es diferente de null.

Las codificaciones en el JTextArea inferior se deben mostrar respetando el orden en el que el usuario las ha introducido en el JTextArea superior.

El JTextArea superior debe llevar asociado un auditor de eventos de teclado de tal forma que cuando se pulse un espacio o el cambio de línea, se cree un hilo que busque la palabra que se ha escrito en el fichero indicado en el JTextField.

La figura 2 muestra un ejemplo de ejecución del applet. Como se puede ver, el applet muestra en la parte inferior la codificación de las palabras que el usuario introduce en la parte superior. Como se puede observar, las codificaciones aparecen en el mismo orden que las palabras introducidas (cada linea una palabra, y cada palabra una letra del original).

6.1. Material adicional

6.1.1. Ficheros de ejemplo

- Applet PruebaApplet.jar y la página que lo contiene PruebaApplet.html. Este applet contiene código que accede al disco donde se ejecuta. Se puede ver que al ejecutarlo lanza una excepción de seguridad (se puede ver en la ventana del plug-in).
- Applet PruebaAppletFirmado.jar y la página que lo contiene PruebaAppletFirmado.html. Este applet realiza la misma acción que el anterior pero a diferencia de aquél está firmado, por lo que una vez aceptada su ejecución no da la excepción de seguridad.
- Los dos applets anteriores intentan acceder a un fichero colocado en el directorio c:/tmp. El fichero es fichero.txt
- El código fuente del applet PruebaApplet.java

- Alfabeto.txt con el alfabeto fonetico internacional

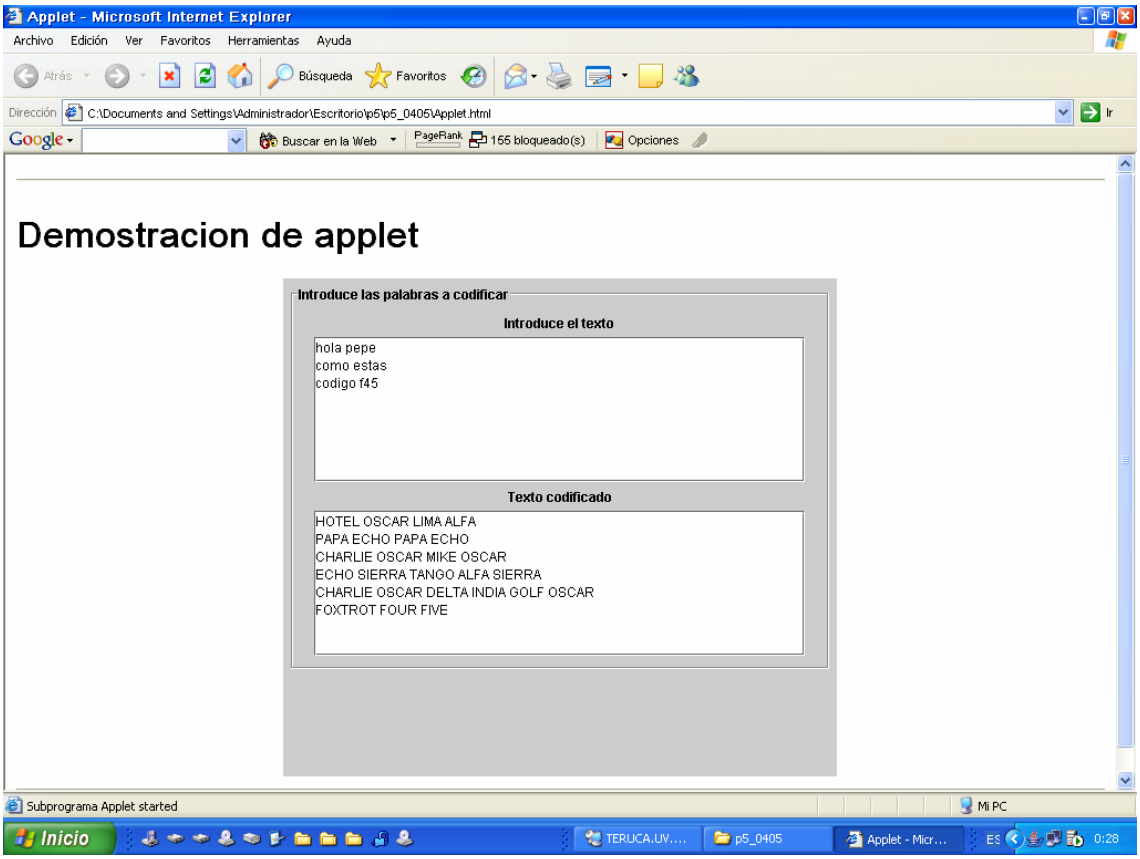

Figura 2: Ejemplo de ejecución del programa