Java y Herramientas de Desarrollo

Sesión 6: Conceptos avanzados de AWT y Swing

© 2006-2007 Depto. Ciencia de la Computación e IA

Puntos a tratar

- Applets
- Seguridad en Java
- Conceptos avanzados de Swing

Applets

- Un applet es una aplicación gráfica Java accesible desde internet
- No se ejecutan por sí mismas, sino a través de navegadores Web
- Se construyen como una aplicación gráfica normal, salvo que:
  - No se hereda de *Frame* o *JFrame*, sino de *Applet* (paquete **java.applet)** o *JApplet*
  - No hay constructor, sino un método *init( )*
  - No hay método *main(...)* al no poder ejecutarse solos

© 2006-2007 Depto. Ciencia de la Computación e IA

Inserción y ejecución de applets

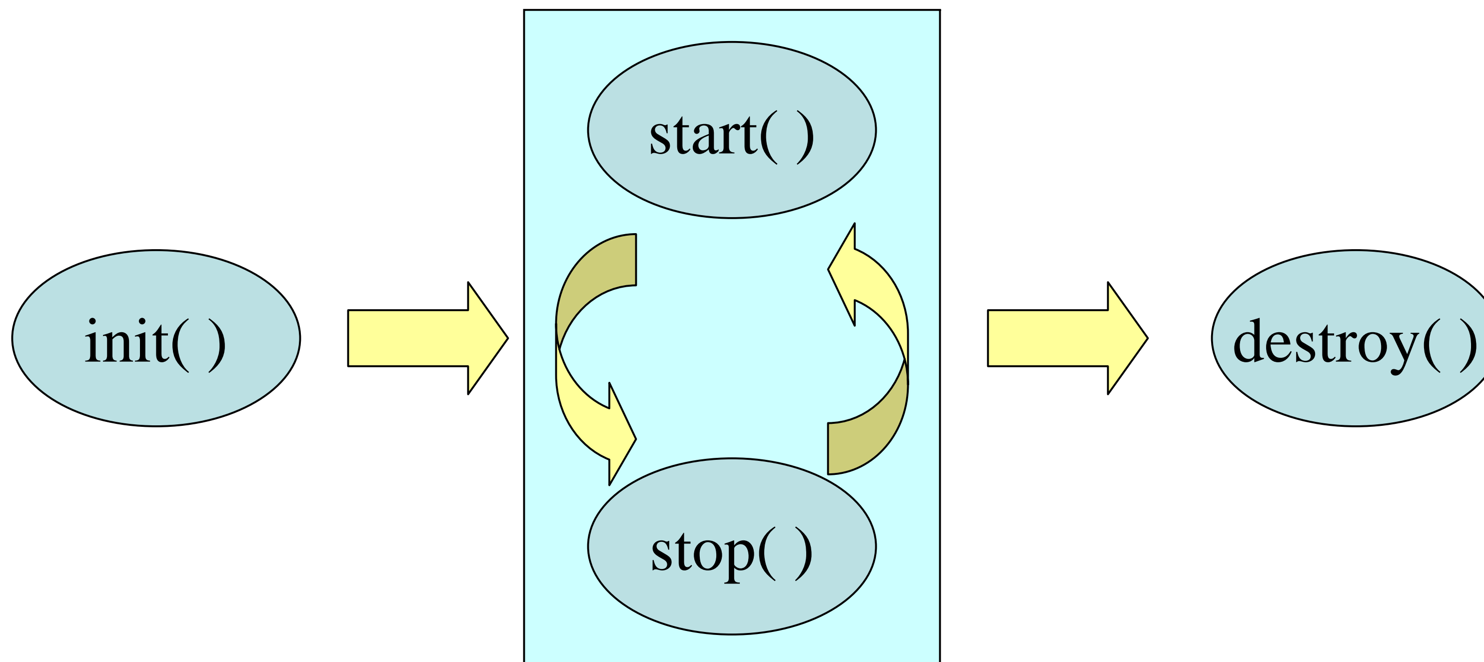
- Para ejecutar un applet, se coloca una etiqueta <APPLET> u <OBJECT> en una página HTML:

```
<HTML>
<BODY>

<APPLET code="nombre_prog.class" width=anchura height=altura>
</APPLET>

<OBJECT codetype="application/java"
classid="java:nombre_prog.class" width=anchura height=altura>
</OBJECT>
```

- Después se puede abrir la página desde el navegador o utilizando la herramienta *appletviewer* de JDK:

```
appletviewer pagina.html
```

Ciclo de vida de un applet



© 2006-2007 Depto. Ciencia de la Computación e IA

Ciclo de vida de un applet

- ***init( )***: llamado cuando el applet se carga por primera vez. Hace de constructor, e inicializa las características del applet: controles, valores de variables, etc.
- ***stop( )***: llamado para detener el applet cuando se requiera (por ejemplo, al superponer una ventana a la del applet)
- ***start( )***: llamado para rearrancar el applet cuando se requiera (por ejemplo, al volver a poner la ventana del applet en primer plano)
- ***destroy( )***: llamado cuando el applet deja de utilizarse (por ejemplo, cuando cerramos la ventana del navegador)

© 2006-2007 Depto. Ciencia de la Computación e IA

Seguridad en las versiones Java

- La seguridad en Java se centra básicamente en programas remotos que se ejecutan en otras máquinas (applets)
- En JDK 1.0 simplemente se les restringían ciertas operaciones
- En JDK 1.1 se introdujeron los *applets firmados*, con menores restricciones
- Desde JDK 1.2 se permite aplicar restricciones de seguridad sobre cualquier programa

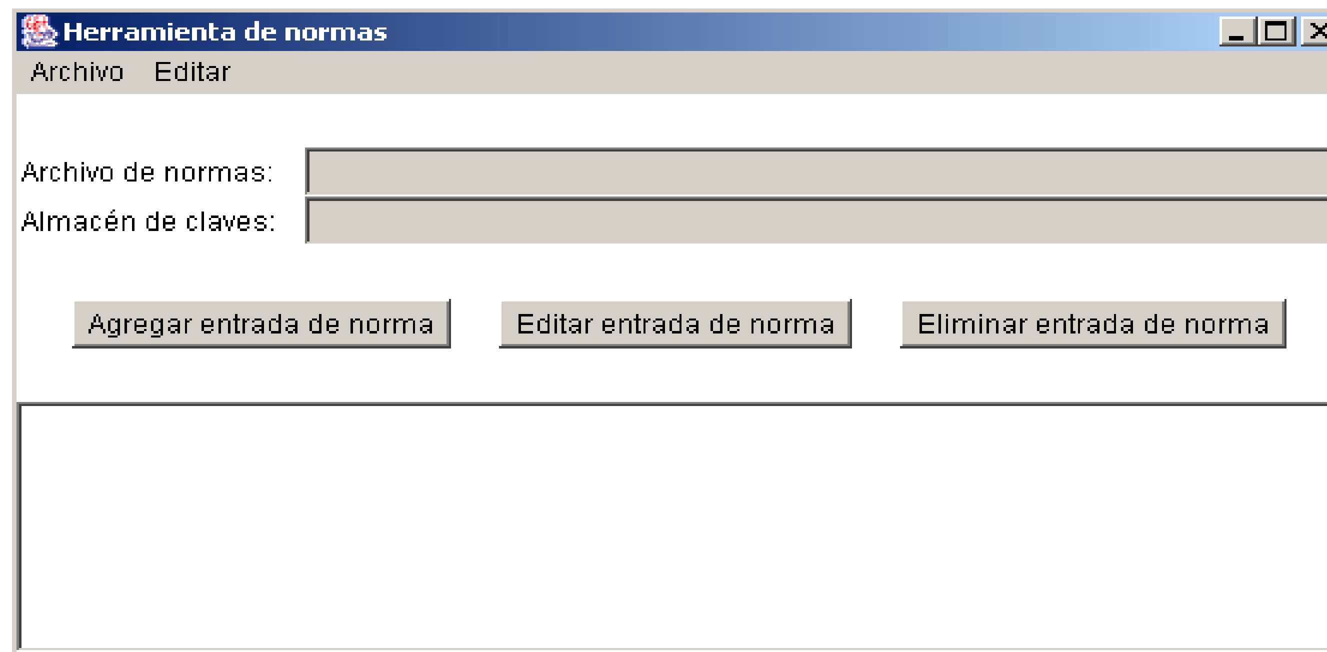
© 2006-2007 Depto. Ciencia de la Computación e IA

Restricciones de seguridad en applets

- No pueden acceder a métodos nativos
- No pueden acceder a ficheros de la máquina local, salvo que proporcionen la URL absoluta
- No pueden conectar con ningún *host* distinto del *host* desde el que se descarga el applet
- No pueden ejecutar otros programas
- No pueden leer las propiedades del sistema
- Las ventanas que muestran los applets son diferentes a las del sistema operativo

Ficheros de políticas de seguridad

- La herramienta *policytool* de JDK permite definir ficheros de políticas de seguridad para aplicar en applets y aplicaciones:



Ficheros de políticas de seguridad

- Cuando ejecutamos una aplicación o applet, los ficheros de políticas que se cargan son los definidos en ***{java.home}/lib/security/java.security***

- Al ejecutar la aplicación o applet también podemos indicarle como parámetro el fichero de políticas de seguridad a emplear:

```
appletviewer –J-Djava.security.policy=fichero MiApplet

java -Djava.security.manager –Djava.security.policy=fichero
   MiAplicacion
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Gestores de seguridad

- Es el objeto encargado de determinar si cierta operación está permitida o no
- En el caso de applets, es el navegador quien carga el gestor, por lo que siempre están sujetos a restricciones
- En las aplicaciones, podemos cargarlo y/o obtenerlo con los métodos:

```
System.getSecurityManager()
System.setSecurityManager()
```

- Podemos definir nuestro gestor heredando de *SecurityManager* y redefiniendo los métodos *checkXXX(...)* necesarios
- Sólo se puede establecer un gestor de seguridad para los programas que no tengan ya uno establecido

© 2006-2007 Depto. Ciencia de la Computación e IA

Posibilidades de Swing

- Swing ofrece algunas posibilidades que no tiene AWT:
  - Uso de acciones para coordinar eventos de diferentes fuentes
  - Incorporación de funciones de accesibilidad
  - Uso de bordes o iconos
  - Modificación de la apariencia (*look & feel*) independientemente de la plataforma
  - Uso de temporizadores
  - Hilos seguros para multitarea en aplicaciones gráficas

© 2006-2007 Depto. Ciencia de la Computación e IA

Uso de acciones

- Coordinan y centralizan un mismo evento de tipo *ActionListener* desde varias fuentes
- Primero definimos una clase que implemente la interfaz *Action* o que herede de *AbstractAction* que ya implementa la interfaz:

```
class MiAction extends AbstractAction {
   public MiAction() {
   }
   public void actionPerformed(ActionEvent e) {
        ... // Acción a realizar
   }
}
```

 © 2006-2007 Depto. Ciencia de la Computación e IA

Uso de acciones

- Después añadimos como *ActionListener* el objeto creado, a los controles oportunos:
  (p. ej, el mismo evento al pulsar un botón o pulsar Intro en un cuadro de texto)

```
MiAction ma = new MiAction();
JButton btn = new JButton("Boton");
JTextField txt = new JTextField();
btn.addActionListener(ma);
txt.addActionListener(ma);
```

 © 2006-2007 Depto. Ciencia de la Computación e IA

Teclas de método abreviado

- Podemos hacer que al pulsar una tecla sobre un componente se ejecute una acción:
  - Mapeamos cada pulsación de tecla(s) deseada con un nombre de acción
  - Mapeamos cada nombre de acción con el objeto de tipo *Action* que desarrollará la acción
- Por ejemplo, si queremos que al pulsar F1 sobre un *JPanel* se ejecute el código de una acción de tipo *MiObjetoAction*, haríamos:

```
MiObjetoAction ma = new MiObjetoAction();
JPanel p = new JPanel();
...
p.getInputMap().put(KeyStroke.getKeyStroke("F1"), "accion1");
p.getActionMap().put("accion1", ma);
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Teclas de método abreviado

- ***getInputMap( )*** admite un parámetro entero, que indica el nivel al que queremos disparar la acción (**JComponent.WHEN_FOCUSED** para dispararla si estamos sobre el componente, **JComponent.WHEN_IN_FOCUSED_WINDOW**, para dispararla si estamos sobre la ventana, etc)

- ***KeyStroke*** permite obtener objetos que representan pulsaciones de teclas, bien pasándole su representación en comillas ("F1"), o con otros métodos como:

```
KeyStroke.getKeyStroke('C', InputEvent.CTRL_MASK &
                            InputEvent.ALT_MASK);
```

Uso de bordes

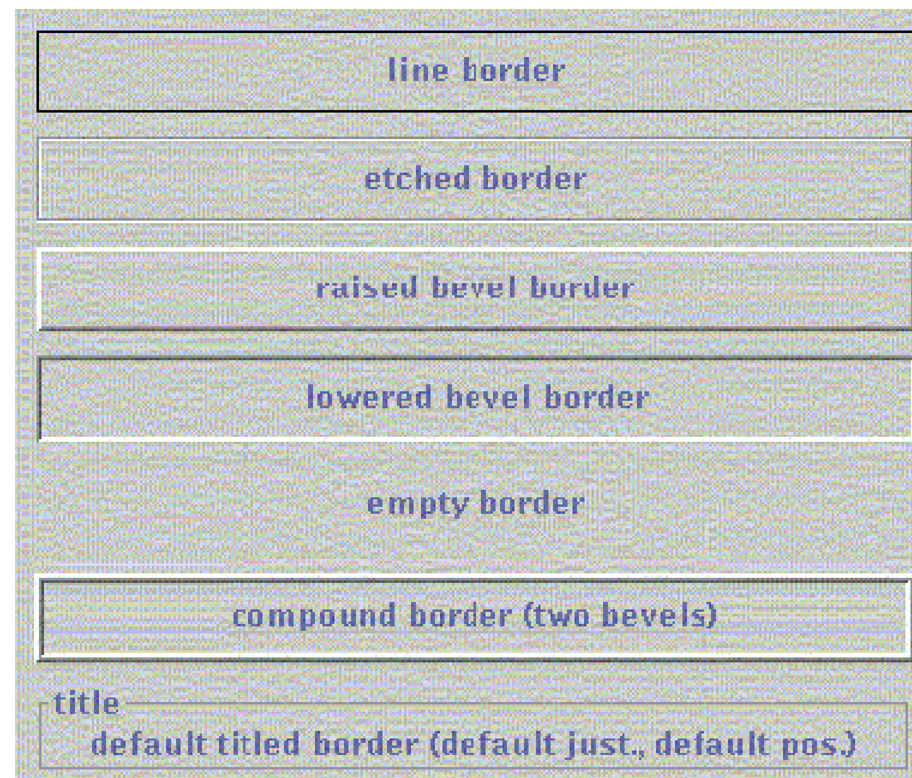
- Podemos definir bordes sobre cualquier subtipo de *JComponent* (en general, *JPanel* o *JLabel*).
- Utilizamos el método ***setBorder(...)*** del componente, pasándole el tipo de borde adecuado, de los que ofrece ***javax.swing.BorderFactory***:

```
JPanel p = new JPanel();
p.setBorder(BorderFactory.createLineBorder(Color.red));
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Uso de bordes

- Algunos de los bordes disponibles:



© 2006-2007 Depto. Ciencia de la Computación e IA

Transferencia de datos

- Podemos transferir datos:
  - Entre controles de una aplicación
  - Entre aplicaciones Java diferentes
  - Entre aplicaciones Java y programas nativos
- Podemos transferirla de dos formas:
  - Arrastrando y soltando (*drag & drop*) la información a transferir de un lugar a otro
  - Copiando/Cortando y pegando la información
- Para arrastrar información de un control Swing que lo permita, llamamos a su método:

```
control.setDragEnabled(true);
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Uso de iconos

- Algunos controles (*JButton, JLabel, JTabbedPane*) permiten definir iconos en ellos
- Para ello se tiene la interfaz *Icon*, y la implementación de la misma en la clase *ImageIcon*, para cargar JPG, GIF o PNG
- Por ejemplo, para añadir un icono "*icono.gif*" a una etiqueta:

```
JLabel lbl = new JLabel ("Etiqueta",
                         new ImageIcon("icono.gif"),
                         SwingConstants.CENTER);
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Temporizadores

- La clase *javax.swing.Timer* permite indicar cuándo queremos disparar un evento de acción, y si queremos dispararlo periódicamente
- Para utilizar temporizadores creamos un *Timer*, pasándole 2 parámetros:
  - El tiempo en ms. desde que se lanza el *Timer* hasta que se disparará
  - El objeto *ActionListener* con el código de la acción que se realizará

```
Timer t = new Timer(1000, new ActionListener()
{
   public void actionPerformed(ActionEvent e)
   {
        // Codigo de la acción
   }
});
```

 © 2006-2007 Depto. Ciencia de la Computación e IA

Temporizadores

- Una vez creado el *Timer*, para lanzarlo llamamos a su método *start( )*
- Por defecto el Timer se ejecuta periódicamente. Si sólo queremos ejecutarlo una vez, llamamos a su método *setRepeats(false)*

```
Timer t = new Timer(1000, new ActionListener()
{
   ...
});

t.setRepeats(false);
t.start();
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Apariencia

- El método *setLookAndFeel(...)* de la clase *UIManager* permite indicar qué apariencia queremos que tenga nuestra aplicación
- Podemos descargarnos nuevas apariencias en Internet (normalmente en ficheros JAR) y luego cargarlas pasándole al método el nombre de la clase entre comillas:

```
UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

UIManager.setLookAndFeel(
        "com.sun.java.swing.plaf.gtk.GTKLookAndFeel");
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Hilos y Swing

- Cuando se crea un componente gráfico, sólo se puede acceder a él desde el código de los eventos que tenga asociados (*event-dispatching thread*).
- Cuando necesitemos acceder desde fuera, podemos:
  - Utilizar los métodos *invokeLater()* o *invokeAndWait()* de *SwingUtilities* para lanzar un hilo que acceda al componente
  - Utilizar *Timers* si queremos actualizar componentes periódicamente (sin necesidad de utilizar los métodos anteriores)

```
Thread t = new Thread(...);
...
SwingUtilities.invokeLater(t);
SwingUtilities.invokeAndWait(t);
```

 © 2006-2007 Depto. Ciencia de la Computación e IA