GESTIÓN DE EVENTOS

CURSO:	TALLER DE PROGRAMACIÓN II	CICLO
		III
Elaborado por:	Ing. Manuel Jesús Sánchez Chero Profesor del Curso	SEMANAS

Material Didáctico para uso exclusivo en clase

INTRODUCCIÓN

La gestión de eventos, es el proceso de repuesta que se genera al hacer clic sobre un botón, los movimientos del ratón, etc. Ha llegado hacer un tema complejo en java desde la versión de java 1.1, la gestión de eventos ha cambiado significativamente. El modelo actual se llama gestión de eventos delegado, En este modelo, se deber registrar específicamente en java si se quiere gestionar un evento, como puede ser hacer clic sobre un botón, la idea es que se mejore la ejecución si sólo se informa de los eventos al código que necesita gestionarlo y no al resto.

Los eventos se registran implementando una interfaz de ***listener*** de eventos, estos son los eventos de ***Listeners*** disponibles y los tipos de eventos que gestionan:

Listener	Descripción
ActionListener	Gestiona los eventos de acción, como hacer clic sobre los botones
AdjustemenListener	Gestiona los casos en los que un componente es escondido, movido, redimensionado o mostrado.
ContainerListener	Gestiona el caso en el que un componente coge o pierde el foco.
ItemListener	Gestiona el caso en el que cambia el estado de un elemento.
KeyListener	Recibe los eventos de teclado.
MouseListener	Recibe en los casos en que es pulsado el ratón, mete un componente, sale un componente o es presionado.
MouseMotionListener	Recibe en el caso en que se arrastra o mueve el ratón.
TextListener	Recibe los cambios de valor de texto.
WindowListener	Gestiona los casos en que una ventana ésta activada, desactivada, con o sin forma de ícono, abierta, cerrada o se sale de ella.

Cada Listener es una interfaz y se debe implementar los métodos de la interfaz. A cada uno de estos métodos se le pasa un tipo de objeto que corresponde al tipo de evento.

Método	Descripción
ActionEvent	Gestiona botones, el hacer clic en la lista o hacer clic en un elemento del menú.
AdjustemenEvent	Gestiona los movimientos de la barra de desplazamiento.
ComponentEvent	Gestiona el caso en el que un componente es escogido, movido, redimensionado o llega a ser visible.
FocusEvent	Gestiona el caso en el que un componente coge o pierde el foco.
InputEvent	Gestiona la marca de activación en una casilla de activación y el hacer clic de un elemento de la lista, hacer selecciones en los controles de opción y las selecciones de los elementos de menú.
KeyEvent	Gestiona la entrada del teclado.
MouseEvent	Gestiona los casos en que se arrastra el mouse, se mueve, se pulsa, se presiona, se suelta o entra o sale un componente.
TextEvent	Gestiona el valor de un cuadro de texto o se ha cambiado.
WindowEvent	Gestiona el caso en que una ventana ésta activada, desactivada, en forma de ícono, sin forma de ícono, abierta, cerrada o abandonada.

En la siguiente tabla se recogen las interfaces para cada tipo de suceso.

Suceso (Event)	Interface (Listener)	Método
ActionEvent	ActionListener	actionPerformed
AdjustementEvent	AdjustementListener	adjustementValueChanged
FocusEvent	FocusListener	focusGained focusLost
ItemEvent	ItemListener	itemStateChanged
KeyEvent	KeyListener	keyTyped keyPressed keyReleased
MouseEvent	MouseListener	mouseClicked mouseEntered mouseExited mousePressed mouseReleased
	MouseMotionListener	mouseDragged mouseMoved

WindowEvent	WindowListener	windowActivated windowClosed windowClosing windowDeactivated windowDeiconified windowIconified windowOpened

La fuente de los sucesos mantiene una lista de objetos interesados (listeners) en los mismos. Se añaden a la lista mediante una función denominada addXXXListener, donde XXX es el tipo de suceso.

boton.addActionListener(accion);

Se muestra los objetos fuente de sucesos y los tipos de objetos interesados en dichos sucesos (listeners) que se pueden añadir (add)

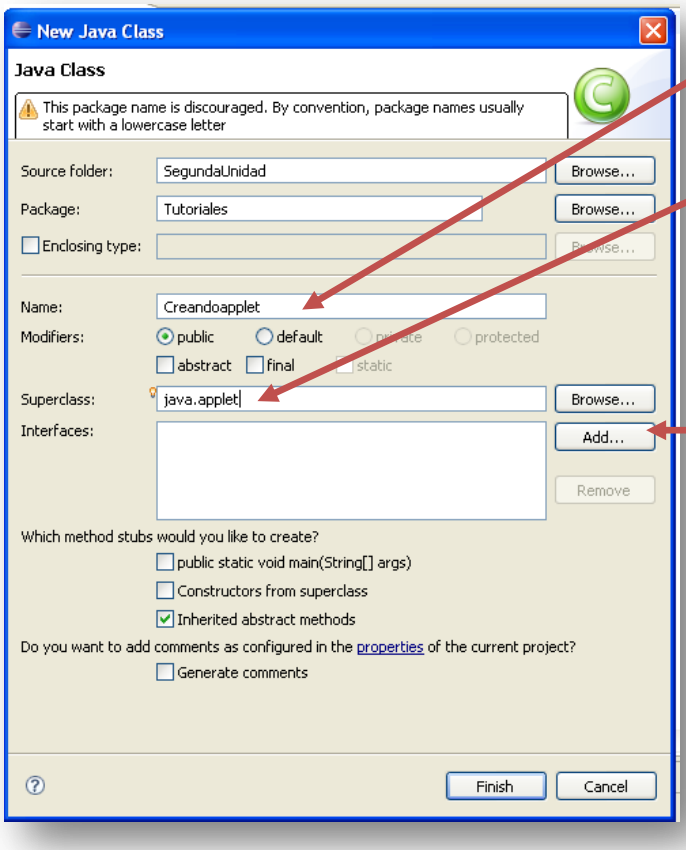
Control	Interfaces
Button	ActionListener
Choice	ItemListener.
Checkbox	ItemListener.
Component	FocusListener KeyListener MouseListener MouseMotionListener
List	ActionListener ItemListener

IMPLEMENTANDO LISTENER

IMPLEMENTANDO LA INTERFAZ ACTIONLISTENER

Empecemos implementando la interfaz ActionListener, que gestiona los eventos de acción, como hacer clic sobre los botones.

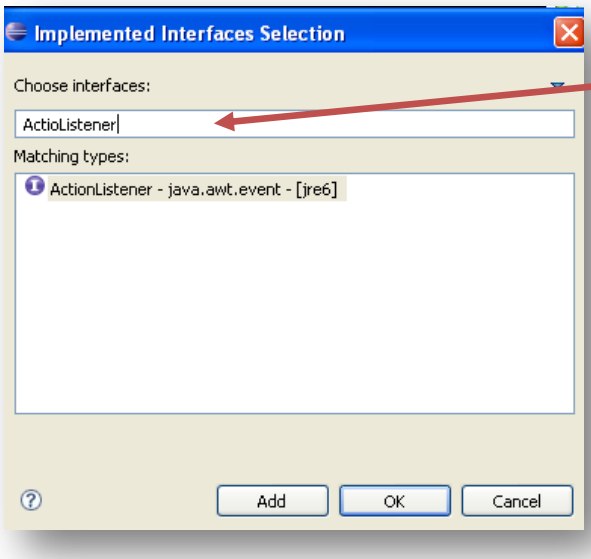
- Utilizando un Applet, empecemos creando la clase ***CreandoApplet***



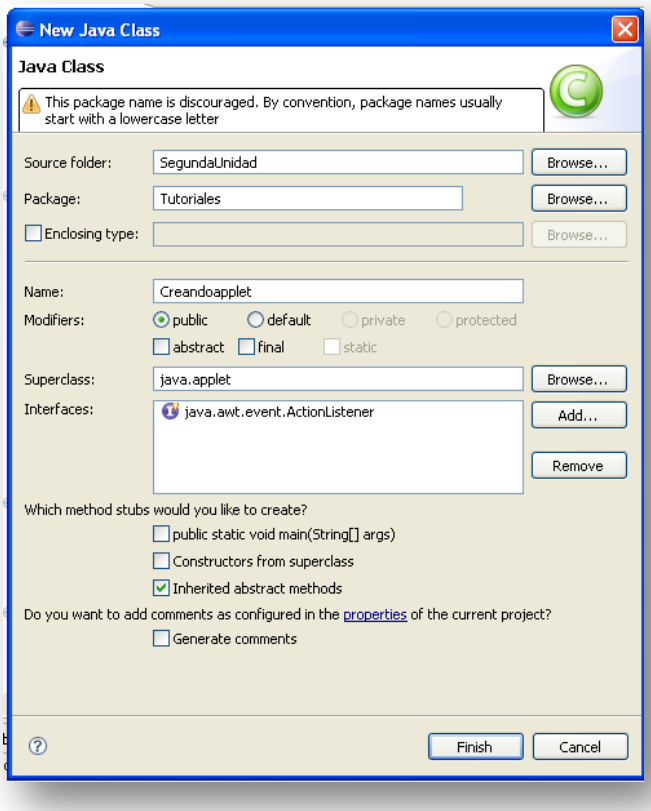
Nombre de la Clase ***CreandoApplet***

En la super clase escribir ***java.applet*** para que herede de la clase Applet

Para implementar las interfaces se debe hacer clic en el botón Add.



Mostrará una ventana para poder escoger las interfaces y luego hacer clic en el botón ***Ok***.



Para terminar hacer clic en el botón ***Finish***, se mostrará el código de la clase CreandoApplet, con sus respectivas librerías e interfaz implementada.

Cómo puedes observar se ha creado el método ***ActionPerformed***.

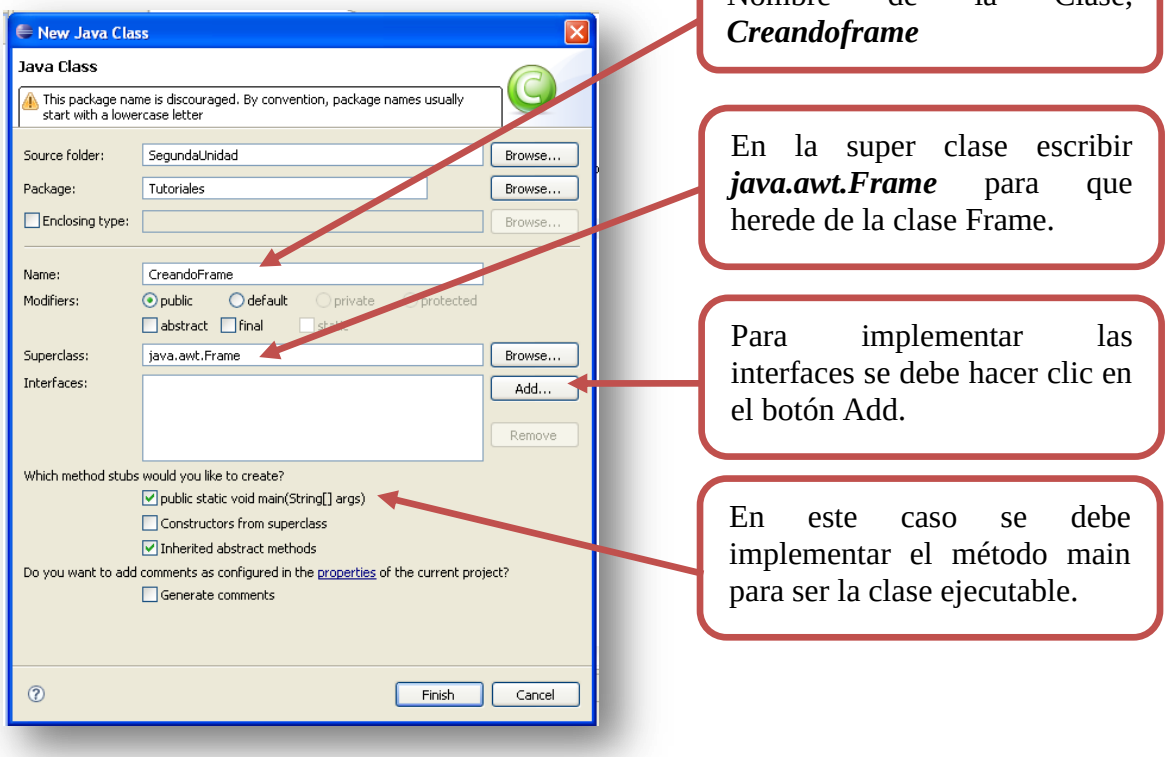
Esto se debe a la implementación de la interfaz ***ActionListener.***

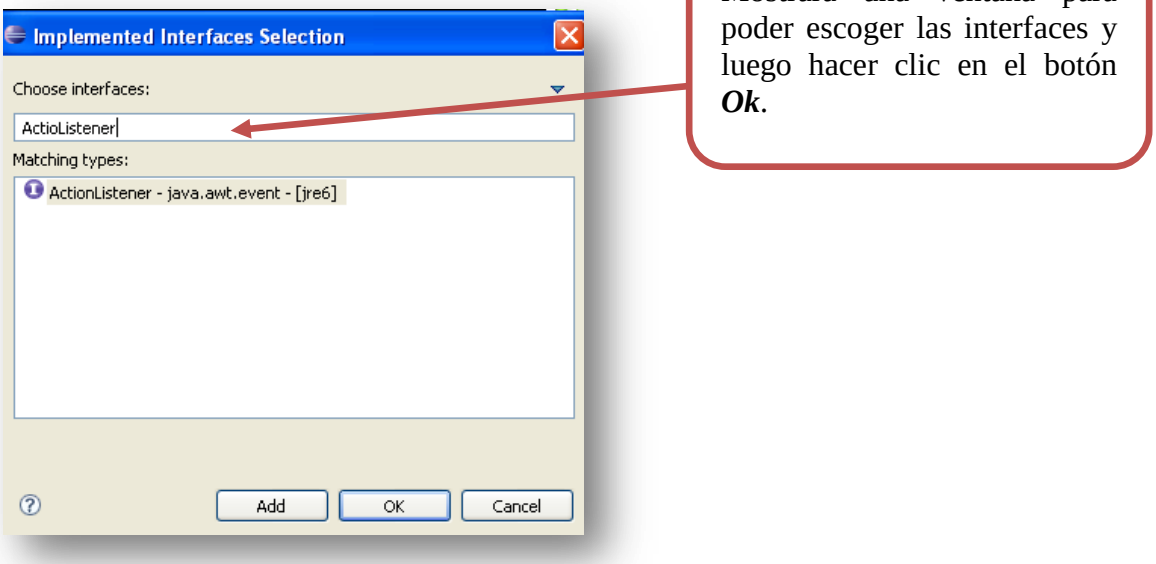
```
import java.applet.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Creandoapplet extends Applet implements ActionListener{

      public void actionPerformed(ActionEvent arg0) {

      }

}
```

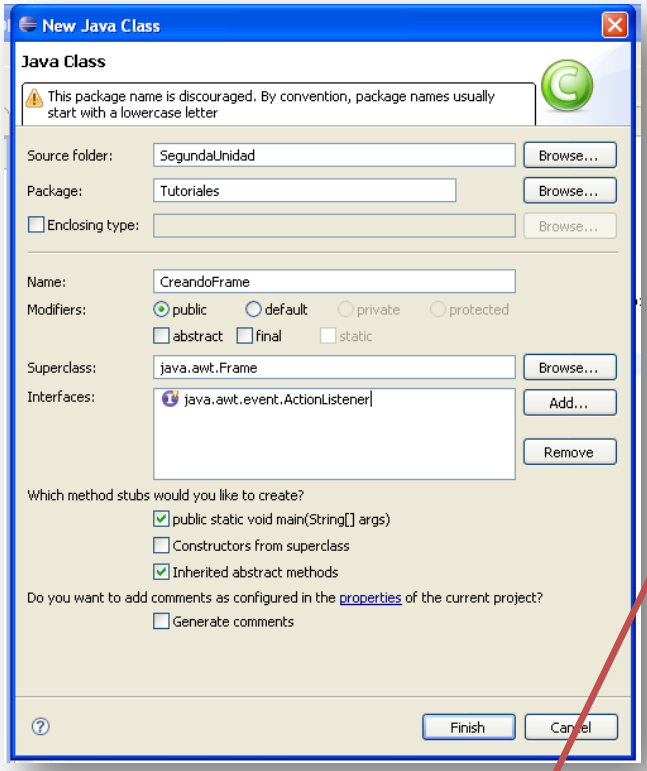
Como se ha implementado un applet hay que agregar de forma manual el método init() a la clase.

```
import java.applet.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Creandoapplet extends Applet implements ActionListener{
    public void init( ) {
        }
public void actionPerformed(ActionEvent arg0) {

      }
```

- Utilizando un Frame, Ahora crearemos la clase ***CreandoFrame*** e implementamos el ActionListener







Para terminar hacer clic en el botón ***Finish***, se mostrará el código de la clase CreandoFrame, con sus respectivas librerías e interfaz implementada.

Cómo puedes observar se ha creado el método ***ActionPerformed***.

Esto se debe a la implementación de la interfaz ***ActionListener.***

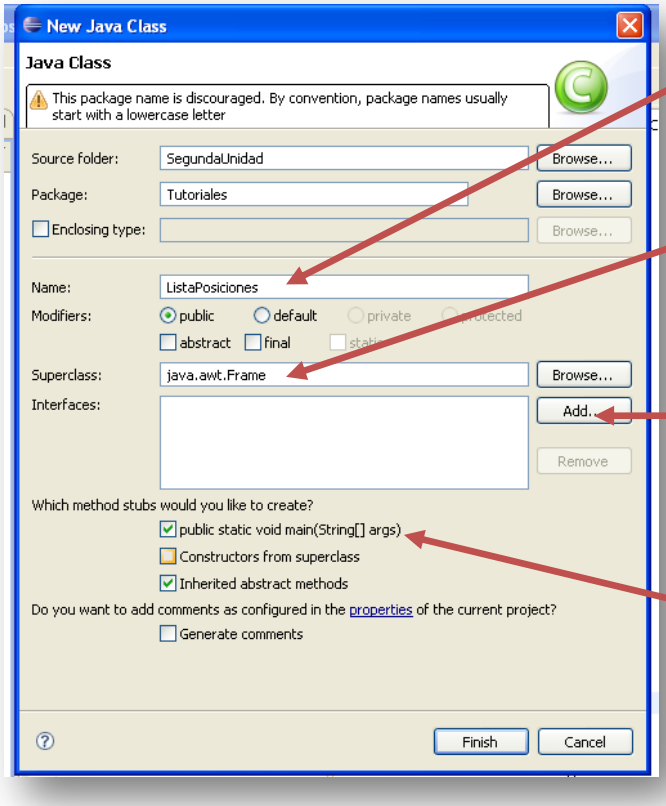
```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CreandoFrame extends Frame implements ActionListener {

	public void actionPerformed(ActionEvent arg0) {

	}

	public static void main(String[] args) {

	}

}
```

Se ha creado el método ***main***.

Aquí se pueden crear objetos**.**

IMPLEMENTANDO LA INTERFAZ ITEMLISTENER

Continuamos implementando la interfaz ItemListener, que gestiona el caso en el que cambia el estado de un elemento.

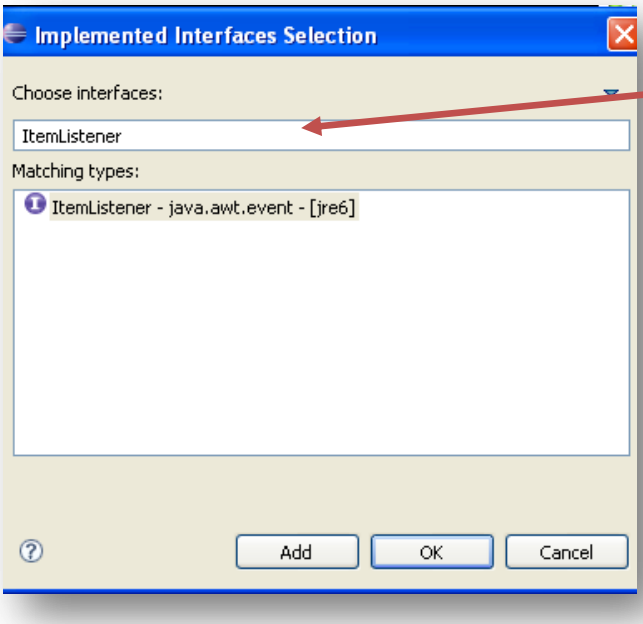


Nombre de la Clase.

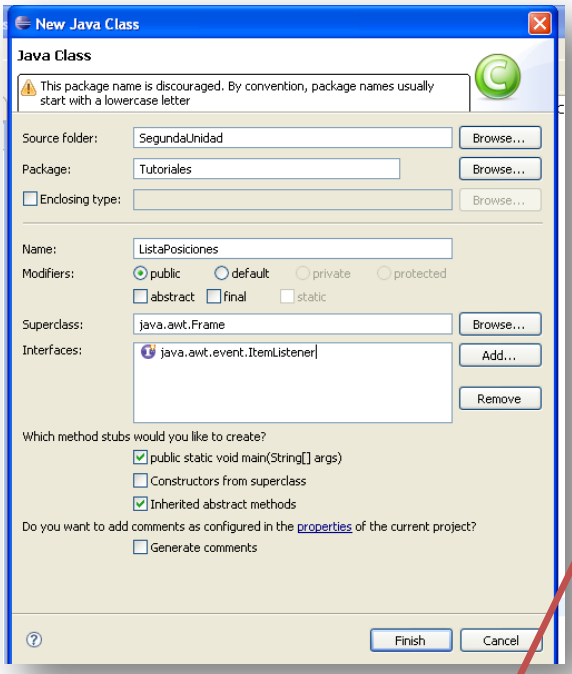
En la super clase escribir ***java.awt.Frame*** para que herede de la clase Frame.

Para implementar las interfaces se debe hacer clic en el botón Add.

En este caso se debe implementar el método main para ser la clase ejecutable.



Mostrará una ventana para poder escoger las interfaces y luego hacer clic en el botón ***Ok***.
En este caso escogemos la interfaz ***ItemListener.***



Para terminar hacer clic en el botón ***Finish***, se mostrará el código de la clase ListaPosiciones, con sus respectivas librerías e interfaz implementada.

Cómo puedes observar se ha creado el método ***itemStateChanged***.

Esto se debe a la implementación de la interfaz ***ItemListener.***

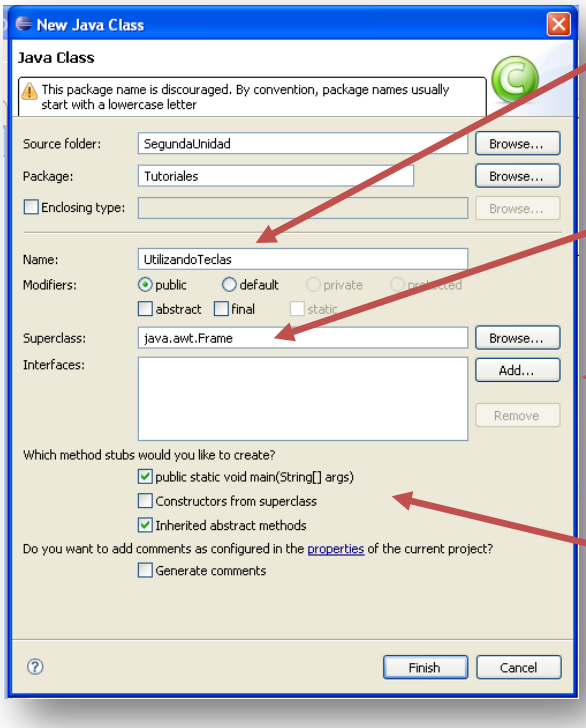
```
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

public class ListaPosiciones extends Frame implements ItemListener {

	public void itemStateChanged(ItemEvent arg0) {

	}

	public static void main(String[] args) {

	}

}
```

Se ha creado el método ***main***.

Aquí se pueden crear objetos**.**

IMPLEMENTANDO LA INTERFAZ KEYLISTENER

Ahora pasamos a implementar la interfaz KeyListener, que permite gestionar la entrada del teclado.

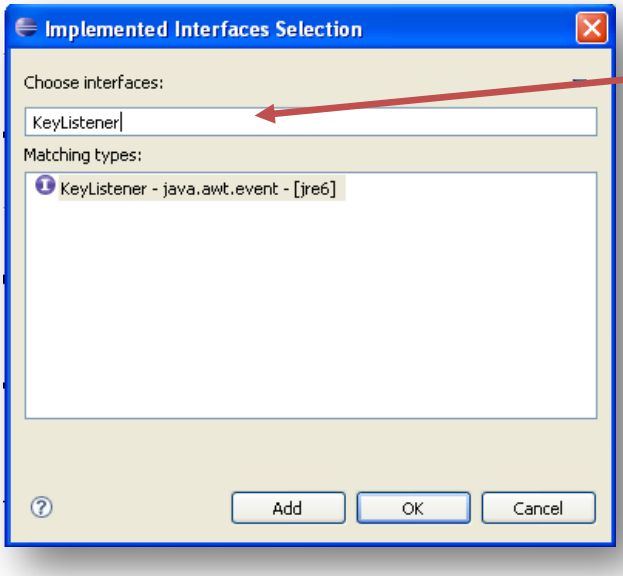


Nombre de la Clase ***UtilizandoTeclas***

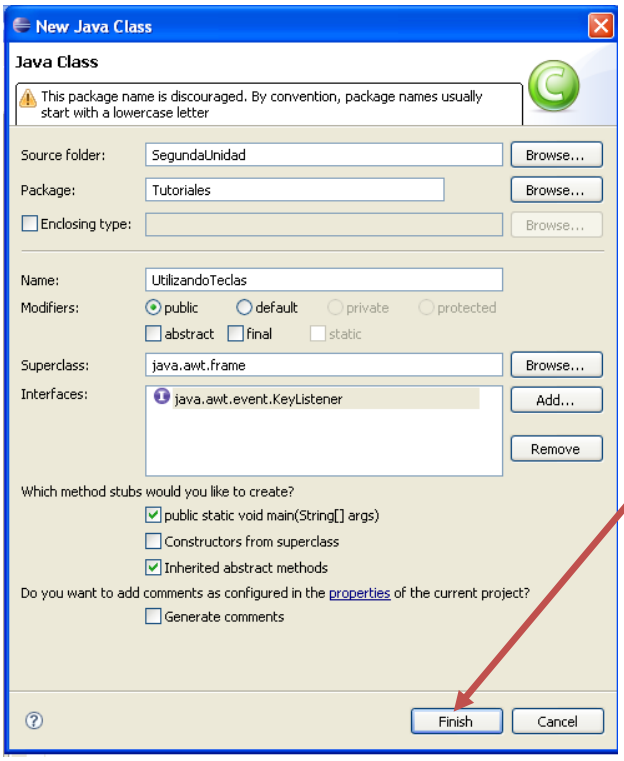
En la super clase escribir ***java.awt.Frame*** para que herede de la clase Frame.

Para implementar las interfaces se debe hacer clic en el botón Add.

En este caso se debe implementar el método main para ser la clase ejecutable.



Mostrará una ventana para poder escoger las interfaces y luego hacer clic en el botón ***Ok***.
En este caso escogemos la interfaz ***KeyListener.***



Para terminar hacer clic en el botón ***Finish***, se mostrará el código de la clase ***UtilizandoTeclas***, con sus respectivas librerías e interfaz implementada.

Cómo puedes observar se ha creado los métodos ***keyPressed, keyReleased y keyTyped***.

Esto se debe a la implementación de la interfaz ***KeyListener.***

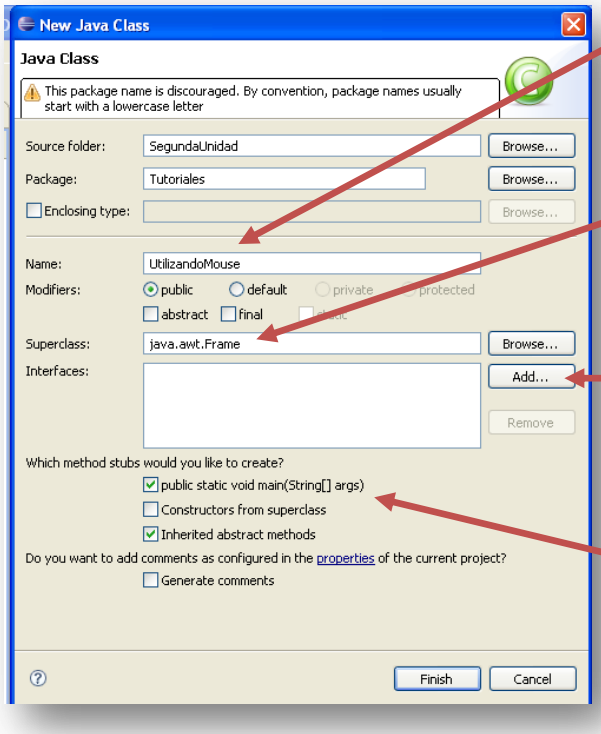
```
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class UtilizandoTeclas extends Frame implements KeyListener {

	public void keyPressed(KeyEvent arg0) {
		}

	public void keyReleased(KeyEvent arg0) {
		}

	public void keyTyped(KeyEvent arg0) {
		}


	public static void main(String[] args) {

	}
}
```

Se ha creado el método ***main***.

Aquí se pueden crear objetos**.**

IMPLEMENTANDO LA INTERFAZ MOUSELISTENER

Ahora nos toca revisar la implementación de la interfaz MouseListener, que se utiliza en los casos en que es pulsado el ratón, mete un componente, sale un componente o es presionado.

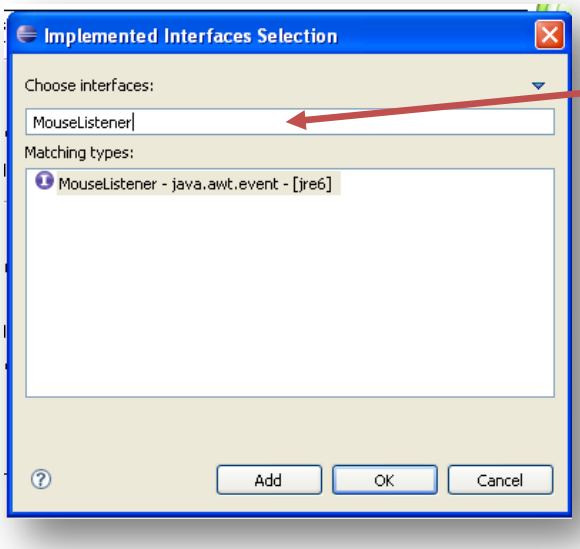


Nombre de la Clase ***UtilizandoMouse***

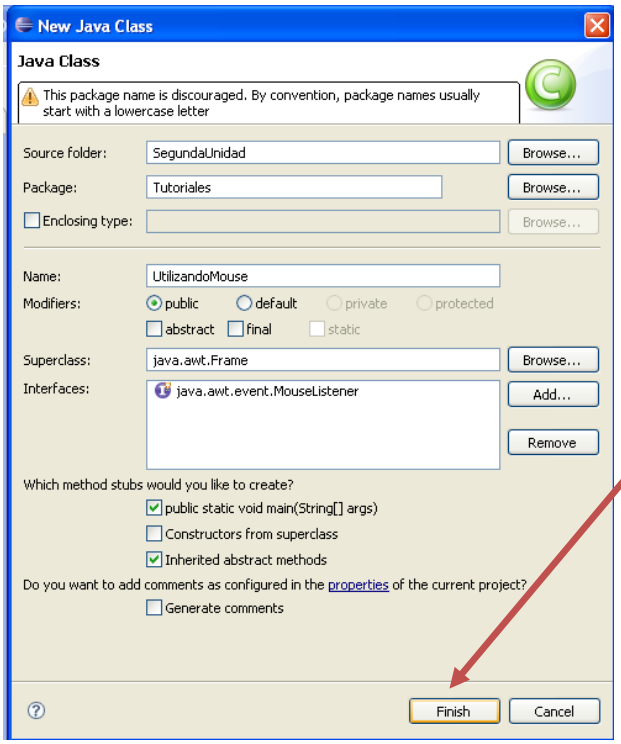
En la super clase escribir ***java.awt.Frame*** para que herede de la clase Frame.

Para implementar las interfaces se debe hacer clic en el botón Add.

En este caso se debe implementar el método main para ser la clase ejecutable.



Mostrará una ventana para poder escoger las interfaces y luego hacer clic en el botón ***Ok***.
En este caso escogemos la interfaz ***MouseListener.***



Para terminar hacer clic en el botón ***Finish***, se mostrará el código de la clase ***UtilizandoMouse***, con sus respectivas librerías e interfaz implementada.

Cómo puedes observar se ha creado los métodos ***mouseClicked, mouseEntered, mouseExited, mousePressed y mouseReleased***.

Esto se debe a la implementación de la interfaz ***MouseListener.***

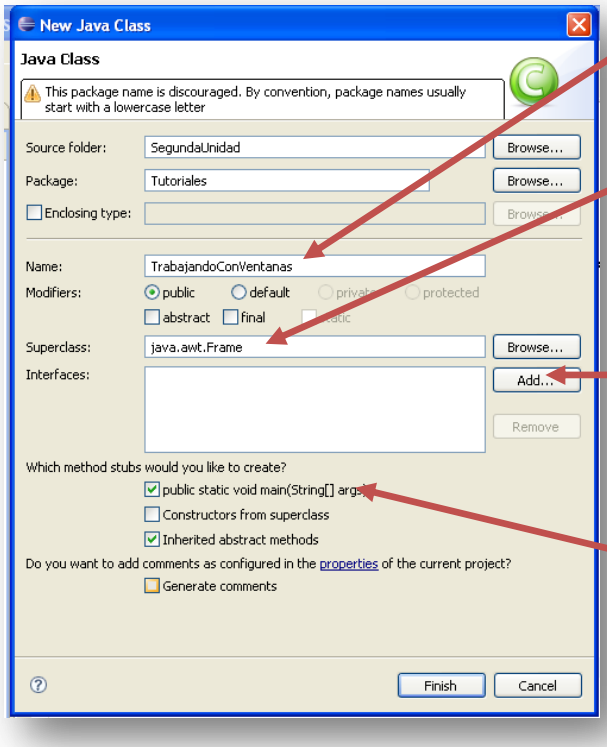
```
Import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class UtilizandoMouse extends  Frame implements
MouseListener {

	public void mouseClicked(MouseEvent arg0) {
	}

	public void mouseEntered(MouseEvent arg0) {
	}

	public void mouseExited(MouseEvent arg0) {
	}

	public void mousePressed(MouseEvent arg0) {
	}

	public void mouseReleased(MouseEvent arg0) {
	}

	public static void main(String[] args) {
	}

}
```

Se ha creado el método ***main***.

Aquí se pueden crear objetos**.**

IMPLEMENTANDO LA INTERFAZ WINDOLISTENER

Por último implementaremos la interfaz WindowListener, que se utiliza para gestionar los casos en que una ventana ésta activada, desactivada, con o sin forma de ícono, abierta, cerrada o se sale de ella.

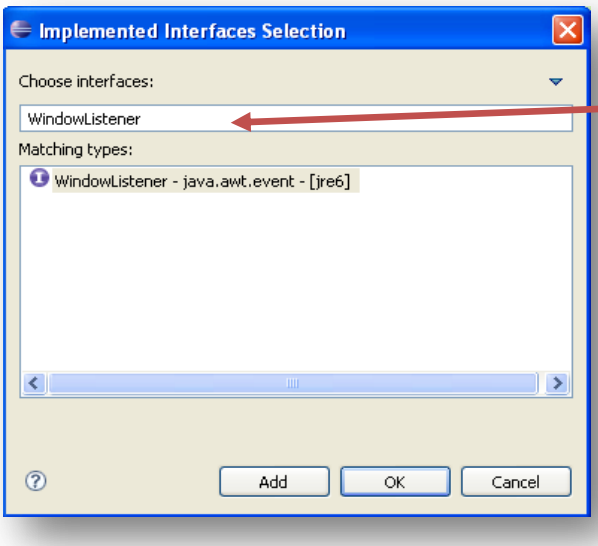


Nombre de la Clase ***TrabajandoConVentana***

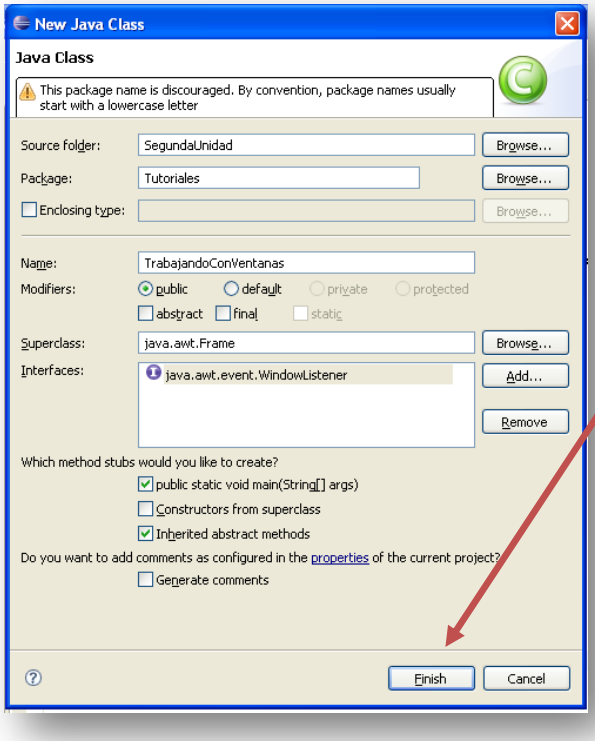
En la super clase escribir ***java.awt.Frame*** para que herede de la clase Frame.

Para implementar las interfaces se debe hacer clic en el botón Add.

En este caso se debe implementar el método main para ser la clase ejecutable.



Mostrará una ventana para poder escoger las interfaces y luego hacer clic en el botón ***Ok***.
En este caso escogemos la interfaz ***WindowListener.***



Para terminar hacer clic en el botón ***Finish***, se mostrará el código de la clase ***TrabajandoConVentanas***, con sus respectivas librerías e interfaz implementada.

Cómo puedes observar se ha creado los métodos ***winwdoActivated, windowClosed, windowClosing, windowDeactivated, windowDeiconifield*** y ***windowOpened***.

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

public class TrabajandoConVentanas extends Frame implements
WindowListener {

	public void windowActivated(WindowEvent arg0) {
		}

	public void windowClosed(WindowEvent arg0) {
		}

	public void windowClosing(WindowEvent arg0) {
		}

	public void windowDeactivated(WindowEvent arg0) {
		}

	public void windowDeiconified(WindowEvent arg0) {
		}

	public void windowIconified(WindowEvent arg0) {
		}

	public void windowOpened(WindowEvent arg0) {
		}

	public static void main(String[] args) {
		}
}
```

Se ha creado el método ***main***. Aquí se pueden crear objetos.

EJEMPLOS

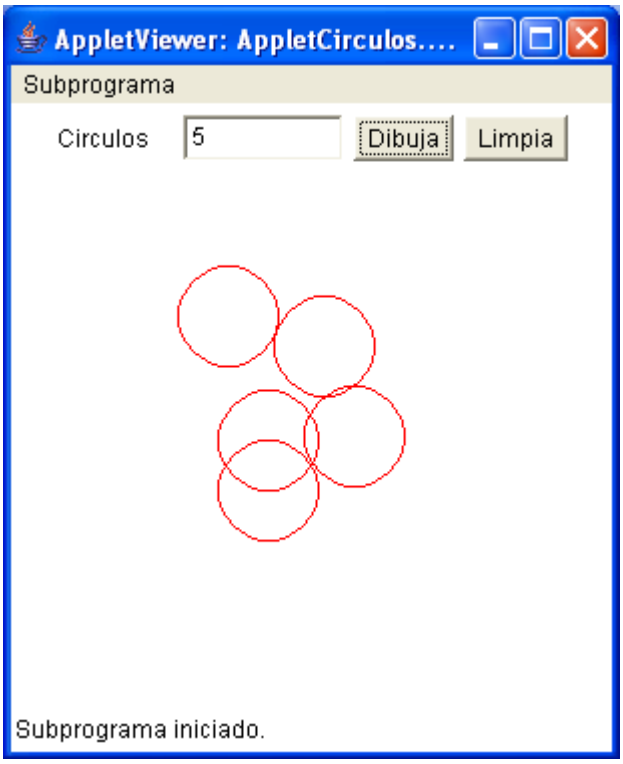
Eventos de Botón

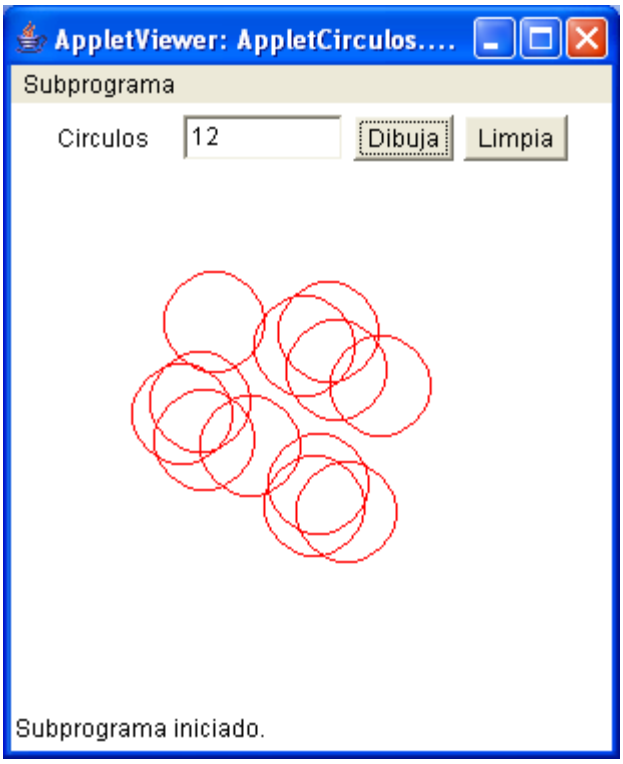
Ahora mostramos un ejemplo en el que utilizamos dos botones, un botón para dibujar círculos en el applet al azar (empezando más debajo de la coordenada 60, 80), el otro botón es para limpiar el applet.

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

public class AppletCirculos extends Applet implements ActionListener {
   Label l1;
   TextField t1;
   Button b1,b2;
   boolean dibuja = false;  // se inicializa dibuja en falso para no dibujar

   public AppletCirculos() {
      l1 = new Label("Circulos");
      t1 = new TextField(8);
      b1 = new Button("Dibuja");
      b2 = new Button("Limpia");
      add(l1);
      add(t1);
      add(b1);
      add(b2);
      b1. addActionListener(this);
      b2. addActionListener(this);
   }

   public void actionPerformed(ActionEvent ae) {
      if (ae.getSource() == b1) {
         dibuja = true;      // si el usuario selecciona dibujar se pone verdadero
      }
      if (ae.getSource() == b2) {
         dibuja = false;    // si el usuario selecciona limpiar se pone en falso
      }
      repaint();
   }
      public void paint(Graphics g) {
           if (dibuja) {   // si dibuja es verdadero se dibuja
                  g.setColor(Color.red);  // se cambia a rojo el color de dibujo
                  int circulos = Integer.parseInt(t1.getText());
                 // se toma el   numero del texto
```

```
                for (int i = 1; i <= circulos ; i++) {  // ciclo de circulos
                        int x1 = (int) (Math.random()*100)+60;
                        int y1 = (int) (Math.random()*100)+80;
                    g.drawOval(x1,y1, 50, 50);  // se dibuja un circulo en
                            // una posición al azar
                 }
       }
       else {  // si dibuja es falso se limpia el applet
       g.setColor(Color.white);
       g.fillRect(0,0,200,200);
       }
   }
}
```

Algunos ejemplos de la vista del applet son:





Eventos de Barra de Desplazamiento

En estos eventos hacemos uso de barras de desplazamiento para realizar alguna instrucción o grupo de instrucciones, y para esto es importante tomar eventos de la clase **Scrollbar**. Los objetos de la clase Srollbar son escuchados a través de implementar una interfaz llamada **AdjustmentListener**, la cual utiliza el método **adjustmentValueChanged**, un método muy parecido al actionPerformed, pero trabaja sobre diferentes elementos de interfaz gráfica.

Para entender este applet debemos consultar la clase ScrollBar que se encuentra en el paquete java.awt. Esta clase tiene diferentes constructores que pueden ser utilizados para crear el objeto de la barra deslizadora, tomemos uno de los disponibles en la siguiente tabla que aparece en las clases de la API de Java:

Constructor Summary

Scrollbar()
Constructs a new vertical scroll bar.

Scrollbar(int orientation)
Constructs a new scroll bar with the specified orientation.

Scrollbar(int orientation, int value, int visible, int minimum, int maximum)
Constructs a new scroll bar with the specified orientation, initial value, visible amount, and minimum and maximum values.

Tomemos el tercer constructor, el cual es utilizado en este ejemplo, primero está la orientación, que es un entero y en este caso utilizaremos uno definido en la misma clase Scrollbar.HORIZONTAL, después viene un valor que es el valor inicial con el que queremos que empiece la barra deslizadora, después esta el valor visible, si queremos que se vea definimos un 1, posteriormente el valor mínimo de la barra, y el máximo número de números a tener.

Veamos el applet:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class AppletEventos2 extends Applet implements AdjustmentListener {
  Label l;
  Scrollbar s;
  TextField t;

  public AppletEventos2() {
    l = new Label("Valor de la barra de Scroll");
    t = new TextField(3);
    s = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 100);
    add(s);
    add(l);
    add(t);
    s.addAdjustmentListener(this);
  }

  public void adjustmentValueChanged(AdjustmentEvent ae) {
    int valor = s.getValue();
    t.setText(""+valor);
  }
}
```

Dicho applet muestra una barra de scroll que al ser deslizada muestra un valor en el campo texto, como aparece en seguida:

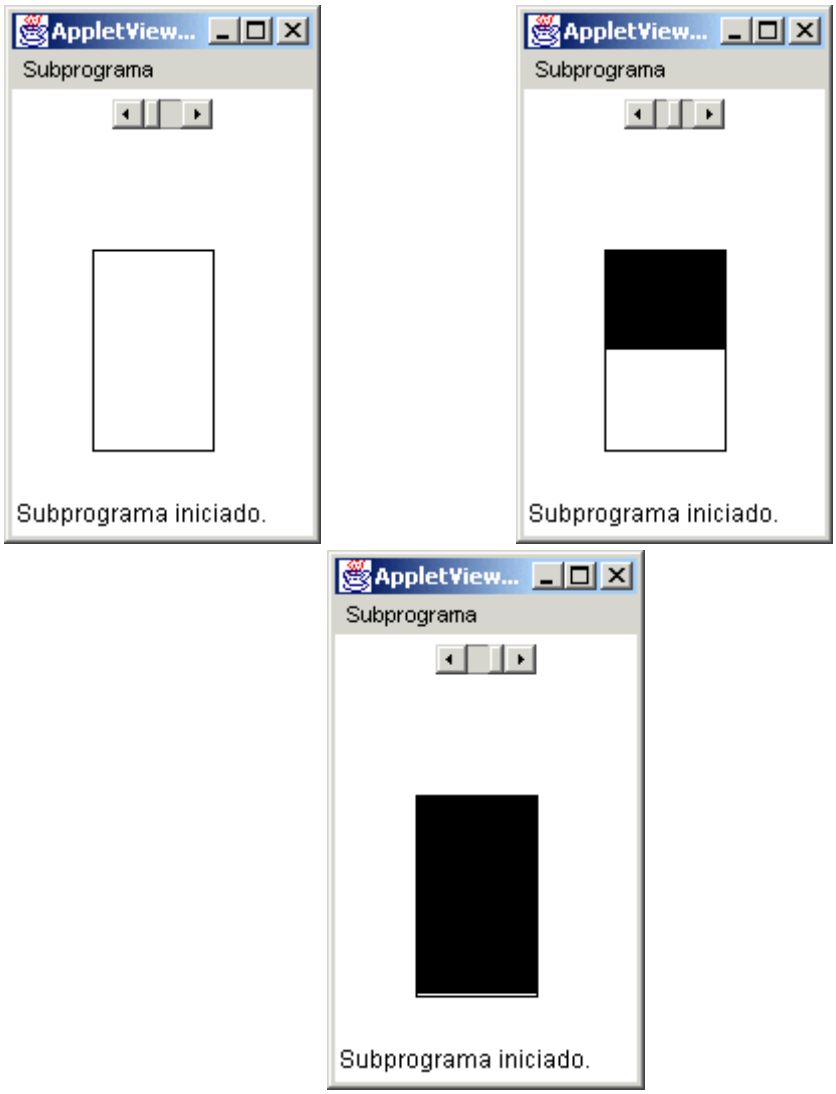
Utilizamos el método getValue de la clase Scrollbar para tomar el valor de la barra deslizadora en base al movimiento que el usuario hizo en ella.

Otro ejemplo que puede ilustrar mejor el uso de una barra deslizadora es el siguiente:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class AppletEventos3 extends Applet implements AdjustmentListener {
  Scrollbar s;
  int barra = 0;

  public AppletEventos3() {
    s = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 0, 100);
    add(s);
    s.addAdjustmentListener(this);
  }

  public void paint(Graphics g) {
    g.drawRect(40, 80, 60, 100);
    g.fillRect(40, 80, 60, barra);
  }

  public void adjustmentValueChanged(AdjustmentEvent ae) {
    barra = s.getValue();
    repaint();
  }
}
```

En este ejemplo al mover la barra deslizadora se ve cómo va cambiando el llenado del rectángulo, utilizando para ello el método fillRect de la clase

Graphics, el cual tiene como parámetros la coordenada en x, la coordenada en y, el ancho y el alto en pixeles, rellenando de color negro dependiendo del valor de la barra, es por esto que en el método paint() se utiliza la variable barra:



En el caso de este applet recurrimos a utilizar el método paint() para que se redibuje cada vez que se mueve la barra deslizadora, utilizando el método fillRect() rellenamos la parte del rectángulo. Es importante hacer notar que la variable barra se definió al inicio de la clase, para que cualquier método la pueda utilizar sin problemas.

Eventos del Ratón

Para implementar este tipo de eventos debemos implementar la interfaz MouseListener y MouseMotionListener.

A continuación se muestra un ejemplo de applet que los utiliza. Cada vez que el botón es presionado la palabra "Abajo" es desplegada en el lugar donde

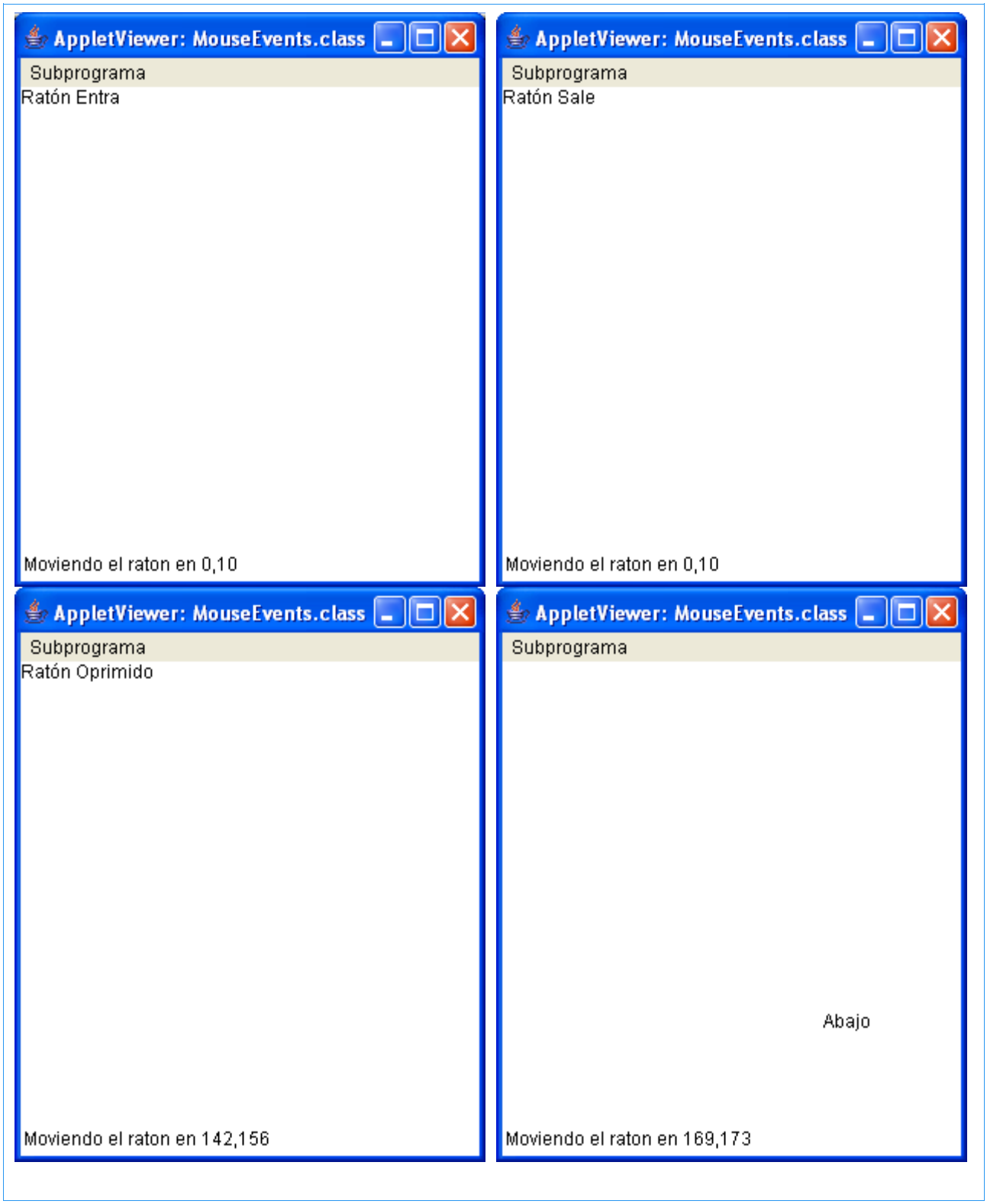
está el apuntador del ratón. Cada vez que el ratón es liberado, la palabra "Arriba" es mostrada. Si un botón es oprimido el mensaje "Ratón oprimido" es desplegado en la esquina superior izquierda del área del applet.

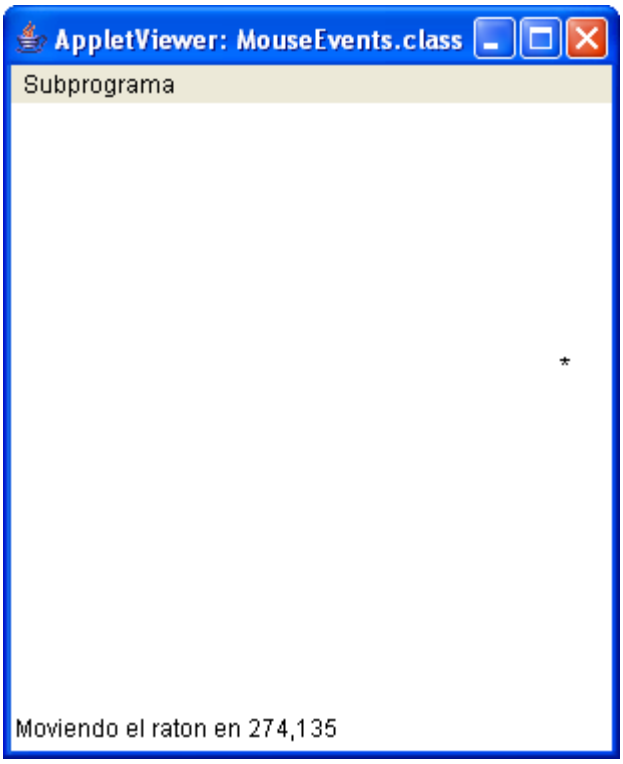
Cada vez que el ratón entra o sale del área del applet, un mensaje es desplegado. Cuando el ratón es arrastrado, un asterisco es mostrado, el cual es arrastrado con el apuntador del ratón. Es importante notar las dos variables **mouseX** y **mouseY** que guardan los lugares del ratón, cuando ocurre uno de los siguientes eventos del ratón: presionado, liberado, o arrastrado. Estas coordenadas son utilizadas dentro del método *paint()*.

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

public class MouseEvents extends Applet implements MouseListener,
                                                    MouseMotionListener {
  String msg = "";
  int mouseX = 0;
  int mouseY = 0;

  public void init() {
      addMouseListener(this);  // se añade el escuchador del ratón
                                                          //a este applet
      addMouseMotionListener(this);
      // se añande el escuchador del movimiento  del raton a este applet
  }

  // cuando se da un clic en el ratón
  public void mouseClicked(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      msg = "Ratón Oprimido";
      repaint();
  }

  // cuando el ratón entra
  public void mouseEntered(MouseEvent me) {
      mouseX = 0;
      mouseY = 10;
      msg = "Ratón Entra";
      repaint();
  }

  // cuando sale el ratón
  public void mouseExited(MouseEvent me) {
      mouseX = 0;
```

```
        mouseY = 10;
        msg = "Ratón Sale";
        repaint();
    }

    // cuando se presiona el ratón
    public void mousePressed(MouseEvent me) {
        mouseX = me.getX();     // se toma el valor de la coordenada de x
        mouseY = me.getY();     // se toma el valor de la coordenada de y
        msg = "Abajo";
        repaint();
    }

    // cuando se libera el ratón
    public void mouseReleased(MouseEvent me) {
        mouseX = me.getX();
        mouseY = me.getY();
        msg = "Arriba";
        repaint();
    }

    // cuando se arrastra el ratón
    public void mouseDragged(MouseEvent me) {
        mouseX = me.getX();
        mouseY = me.getY();
        msg = "*";
        showStatus("Moviendo el raton en " + mouseX + "," + mouseY);
        repaint();
    }

    // moviendo el ratón
    public void mouseMoved(MouseEvent me) {
        showStatus("Moviendo el raton en " + mouseX + "," + mouseY);
    }

    // desplegando el mensaje en el applet
    public void paint(Graphics g) {
        g.drawString(msg, mouseX, mouseY);
    }
}
```

Algunas vistas del applet se dan a continuación:


AppletViewer: MouseEvents.class
Subprograma
Ratón Entra
Moviendo el raton en 0,10
AppletViewer: MouseEvents.class
Subprograma
Ratón Sale
Moviendo el raton en 0,10
AppletViewer: MouseEvents.class
Subprograma
Ratón Oprimido
Moviendo el raton en 142,156
AppletViewer: MouseEvents.class
Subprograma
Abajo
Moviendo el raton en 169,173


AppletViewer: MouseEvents.class
Subprograma
*
Moviendo el raton en 274,135