

CURSO INICIACIÓN JAVA III

Tutoriales de pildorasinformaticas

Descripción breve

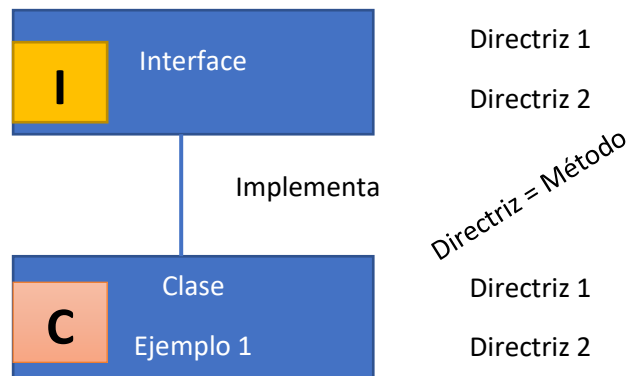
Curso introductorio de Java por pildorasinformaticas.



Pere Manel Verdugo Zamora
pereverdugo@gmail.com

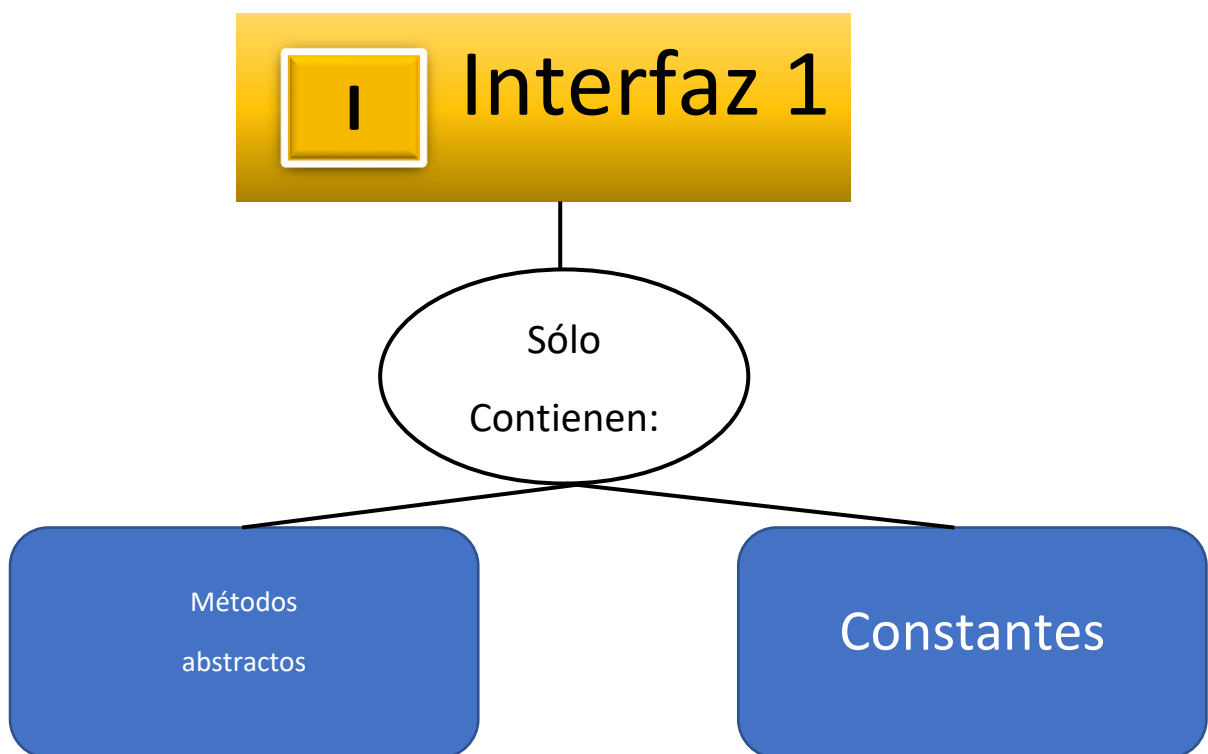
Interfaces y clases internas. Interfaces I. (Vídeo 49)

¿Qué son las interfaces? Conjunto de directrices que deben cumplir las clases.



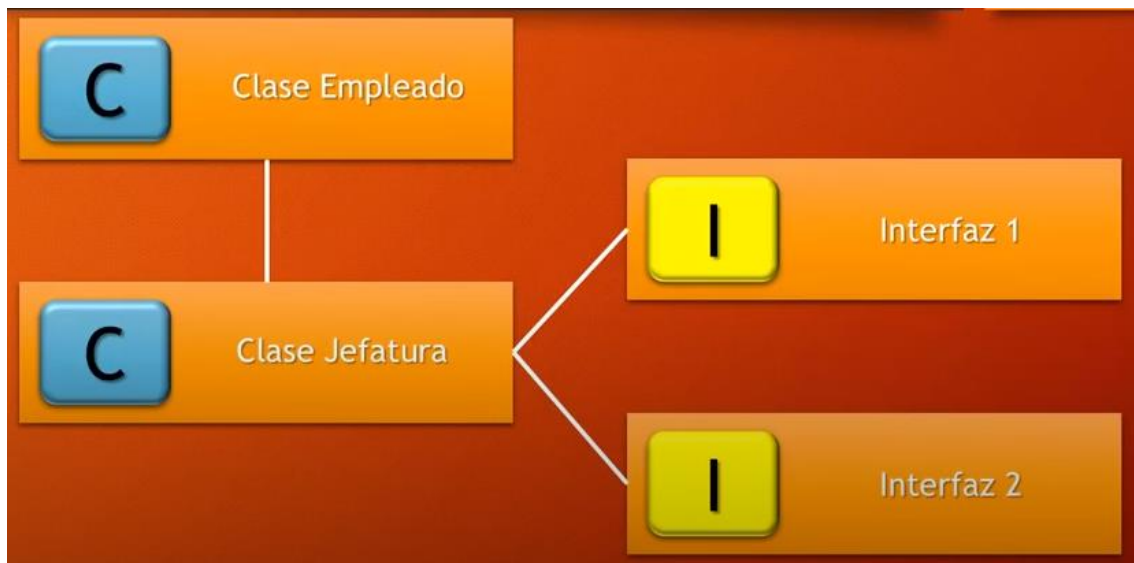
Las interfaces establecen los comportamientos (directrices) a cumplir por las clases.

Características de las interfaces tanto predefinidas (API) como propias.



- Se almacenan en un fichero .class.
- No se pueden instanciar (no uso de new).
- Todos sus métodos son public y abstract. No se implementan.
- No tienen variables. Si constantes.

¿Y esto no lo hacían ya las clases abstractas? Sí pero... Problema herencia simple.



Class Jefatura extends Empleado implements Interfaz1, Interfaz2

Todas las clase de API de Java que está en cursiva no son clases son interfaces.

En este ejemplo vamos a ordenar la matriz por sueldo de menor a mayor.

```
package poo;

import java.util.*;

public class Uso_Empleado {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 3, 25);
        jefe_RRHH.estableceIncentivo(2570);

        Empleado[] misEmpleados= new Empleado[6];

        misEmpleados[0]=new Empleado("Ana", 30000, 1990, 12, 17);
        misEmpleados[1]=new Empleado("Carlos", 50000,1995 ,6 ,2 );
        misEmpleados[2]=new Empleado("Paco",25000 ,2002 ,3 , 15);
        misEmpleados[3]=new Empleado("Antonio", 47500, 2009, 11, 9);
        misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de
sustitución
        misEmpleados[5]=new Jefatura("María", 95000, 1999, 5, 26);

        Jefatura jefa_finanzas=(Jefatura)misEmpleados[5];
        jefa_finanzas.estableceIncentivo(55000);

        for(Empleado i: misEmpleados) {
            i.suboSueeldo(5);
        }
    }
}
```

```
Arrays.sort(misEmpleados);
```

Para ordenar la Array

```
for(Empleado i: misEmpleados) {  
    System.out.println("Nombre: " + i.dameNombre() +  
        " Sueldo: " + i.dameSueldo() +  
        " Fecha de alta : " + i.dameFechaContrato());  
}  
  
}
```

```
class Empleado implements Comparable{
```

La clase Empleado implemente el interface Comparable.

```
public Empleado(String nom, double sue, int agno, int mes, int dia) {  
  
    nombre=nom;  
    sueldo=sue;  
    GregorianCalendar calendario= new GregorianCalendar(agno, mes-1,  
dia);  
    altaContrato=calendario.getTime();  
    ++IdSiguiente;  
    Id=IdSiguiente;  
}
```

```
public Empleado(String nom) {  
    nombre=nom;  
}
```

```
public String dameNombre() { //getter  
    return nombre + " Id: " + Id;  
}
```

```
public double dameSueldo() //getter  
{  
    return sueldo;  
}
```

```
public Date dameFechaContrato() { //getter  
    return altaContrato;  
}
```

```
public void subeSueldo(double porcentaje) { //setter  
    double aumento=sueldo*porcentaje/100;  
    sueldo+=aumento;  
}
```

```
public int compareTo(Object miObjeto) {  
    Empleado otroEmpleado=(Empleado) miObjeto;  
    if(this.sueldo<otroEmpleado.sueldo) {  
        return -1;  
    }  
    if(this.sueldo>otroEmpleado.sueldo) {  
        return 1;  
    }  
    return 0;  
}
```

Refundición.

Creamos nuestro interface compateTo, este método es obligado al implementar Comparable, este método retorna -1, 0 y 1 con el campo que queremos ordenar.

Estamos sobrescribiendo un método compareTo de la interface Comparable.

El método de como ordena lo ha realizado el programador.

```

    private String nombre;
    private double sueldo;
    private Date altaContrato;
    private static int IdSiguiente;
    private int Id;
}

class Jefatura extends Empleado{
    public Jefatura(String nom, double sue, int agno, int mes, int dia) {
        super(nom, sue, agno, mes, dia);
    }

    public void estableceIncentivo(double b) {
        incentivo=b;
    }

    public double dameSueldo() {
        double sueldoJefe=super.dameSueldo();
        return sueldoJefe + incentivo;
    }

    private double incentivo;
}

```

Este será el resultado:

```

Nombre: Paco Id: 4 Sueldo: 26250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Ana Id: 2 Sueldo: 31500.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Antonio Id: 5 Sueldo: 49875.0 Fecha de alta :Mon Nov 09 00:00:00 CET 2009
Nombre: Carlos Id: 3 Sueldo: 52500.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: Juan Id: 1 Sueldo: 60320.0 Fecha de alta :Sat Mar 25 00:00:00 CET 2006
Nombre: María Id: 6 Sueldo: 154750.0 Fecha de alta :Wed May 26 00:00:00 CEST 1999

```

Si lo quisiéramos ordenar por id lo haríamos de la siguiente forma:

```

77 public int compareTo(Object miObjeto) {
78     Empleado otroEmpleado=(Empleado) miObjeto;
79     if(this.Id<otroEmpleado.Id) {
80         return -1;
81     }
82     if(this.Id>otroEmpleado.Id) {
83         return 1;
84     }
85     return 0;
86 }

```

Ahora lo ordena por Id.

```

Nombre: Juan Id: 1 Sueldo: 60320.0 Fecha de alta :Sat Mar 25 00:00:00 CET 2006
Nombre: Ana Id: 2 Sueldo: 31500.0 Fecha de alta :Mon Dec 17 00:00:00 CET 1990
Nombre: Carlos Id: 3 Sueldo: 52500.0 Fecha de alta :Fri Jun 02 00:00:00 CEST 1995
Nombre: Paco Id: 4 Sueldo: 26250.0 Fecha de alta :Fri Mar 15 00:00:00 CET 2002
Nombre: Antonio Id: 5 Sueldo: 49875.0 Fecha de alta :Mon Nov 09 00:00:00 CET 2009
Nombre: María Id: 6 Sueldo: 154750.0 Fecha de alta :Wed May 26 00:00:00 CEST 1999

```

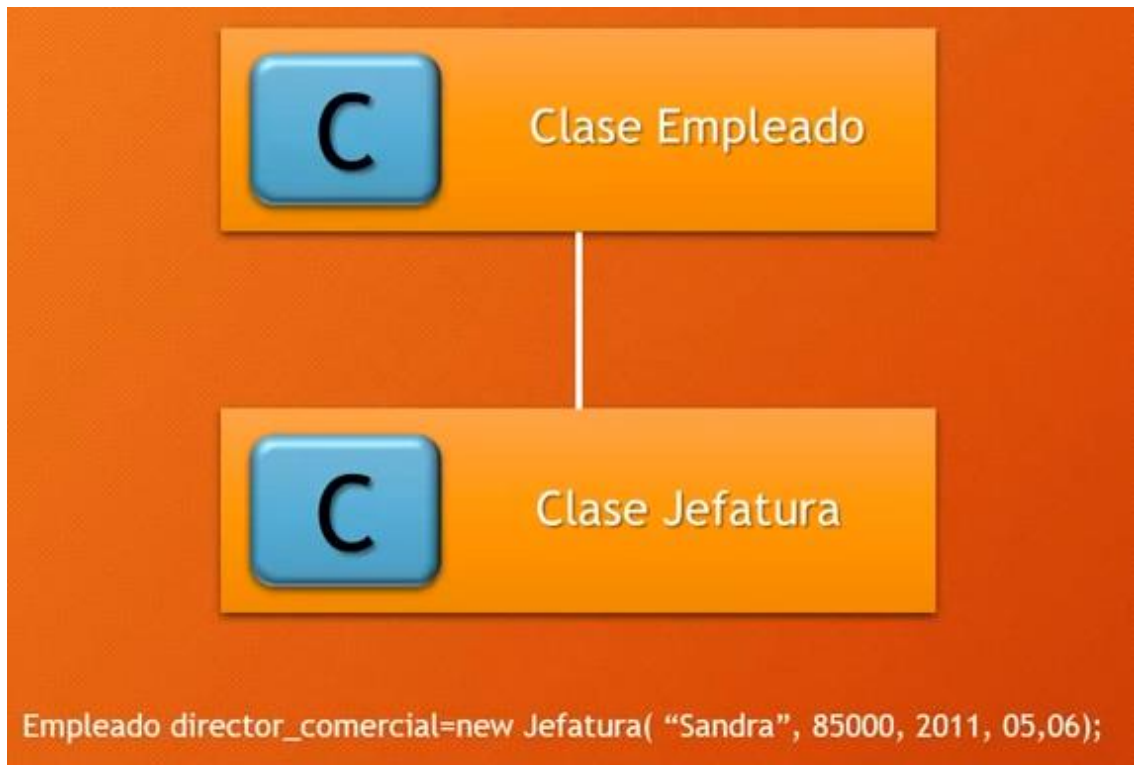


The image shows the cover of a course titled "CURSO JAVA". The background is a dark orange-red gradient with a faint world map. In the top left corner, there is a small Java logo icon. The main title "CURSO JAVA" is written in large, bold, white capital letters. To the right of the title, the number "49" is displayed in white inside a yellow square. Below the main title, the text "INTERFACES Y CLASES INTERNAS" and "INTERFACES I" is written in smaller white capital letters. In the bottom left corner, the word "eclipse" is written in a light, lowercase font. In the bottom right corner, the Java logo is prominently displayed in its characteristic blue and orange colors.

Interfaces y clases internas. Interfaces II (Vídeo 50)

Propiedades de las interfaces.

Instanciación (crear objetos de...)



Lo que sí se puede hacer es:

```
Comparable ejemplo = new empleado("Antonio", 5500, 1998, 06, 08);
```

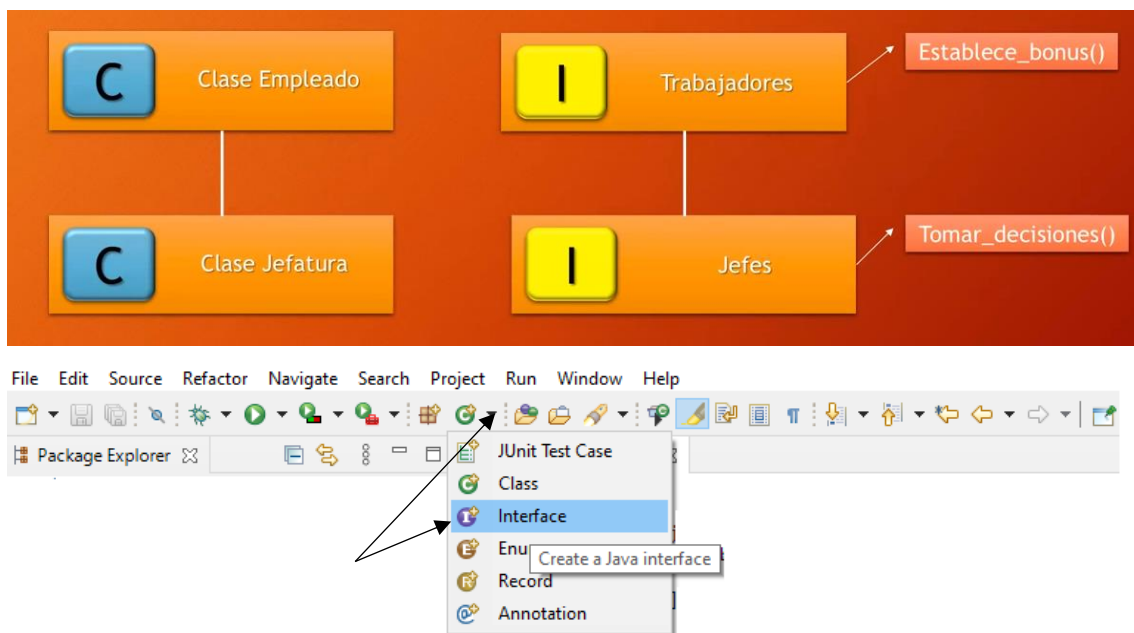
```
if (ejemplo instanceof Empleado) {.....}
```

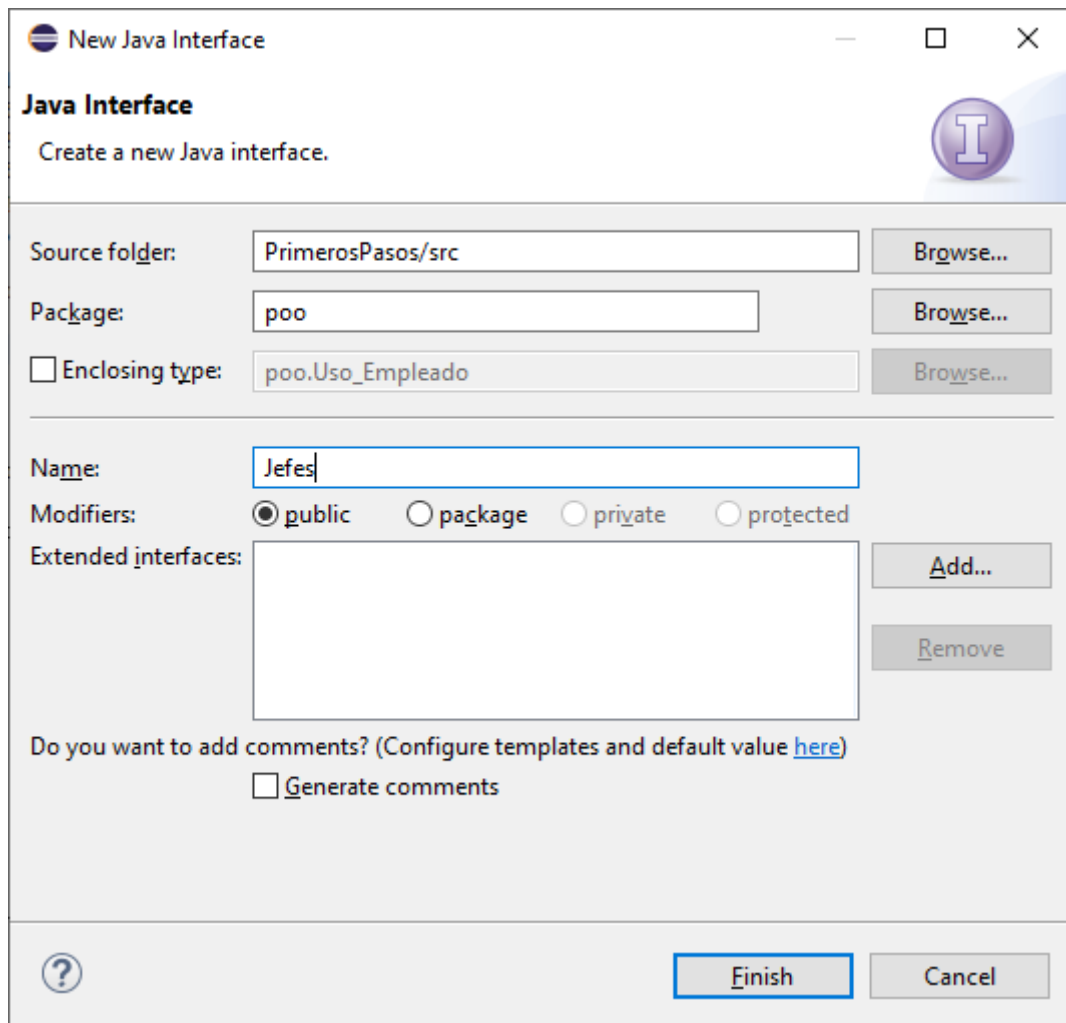
```
26     Empleado director_comercial=new Jefatura("Sandra", 85000, 2012, 05, 05);
27
28     Comparable ejemplo = new Empleado("Elisabeth", 95000, 2011, 06, 07);
29
30     if(director_comercial instanceof Empleado) {
31         System.out.println("Es de tipo Jefatura");
32     }
33
34     if(ejemplo instanceof Comparable) {
35         System.out.println("Implementa la interface comparable");
36     }
```

Este será el resultado:

```
Es de tipo Jefatura
Implementa la interface comparable
```

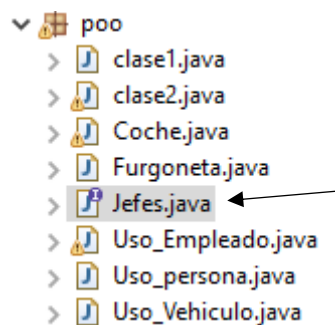
La instrucción instanceof lo podemos utilizar tanto en clases como en interfaces.





Botón Finalizar.

Crea un fichero java más.



```

1 package poo;
2
3 public interface Jefes {
4
5     //public abstract String tomar_decisiones(String decision);
6
7     String tomar_decisiones(String decisión); // Puedes omitir public y abstract
8
9 }

```

Guardamos los cambios y nos vamos a `Usu_Empleados`.

```

93     class Jefatura extends Empleado implements Jefes{
94
95
96     public Jefatura(String nom, double sue, int agno, int mes, int dia) {
97         super(nom, sue, agno, mes, dia);
98     }
99
100    public String tomar_decisiones(String decision) {
101
102        return "Un miembro ha tomado la decisión de: " + decision+ ".";
103    }
104
105    public void estableceIncentivo(double b) {
106        incentivo=b;
107    }
108
109    public double dameSueldo() {
110        double sueldoJefe=super.dameSueldo();
111        return sueldoJefe + incentivo;
112    }
113
114
115    private double incentivo;
116 }

```

En la línea 93 le estamos diciendo que implemente el interfaz Jefes.

Luego estamos obligados a crear un método tomar_decisiones(con un parámetros string) y nos retorne un texto.

Nos vamos a la clase principal donde está el main.

```

7    public static void main(String[] args) {
8        // TODO Auto-generated method stub
9
10
11    Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 3, 25);
12    jefe_RRHH.estableceIncentivo(2570);
13
14    Empleado[] misEmpleados= new Empleado[6];
15
16    misEmpleados[0]=new Empleado("Ana", 30000, 1990, 12, 17);
17    misEmpleados[1]=new Empleado("Carlos", 50000,1995 ,6 ,2 );
18    misEmpleados[2]=new Empleado("Paco",25000 ,2002 ,3 , 15);
19    misEmpleados[3]=new Empleado("Antonio", 47500, 2009, 11, 9);
20    misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de sustitución
21    misEmpleados[5]=new Jefatura("María", 95000, 1999, 5, 26);
22
23    Jefatura jefa_finanzas=(Jefatura)misEmpleados[5];
24    jefa_finanzas.estableceIncentivo(55000);
25
26    System.out.println(jefa_finanzas.tomar_decisiones("Dar más días de vacaciones."));
27
28    Arrays.sort(misEmpleados);
29
30    for(Empleado i: misEmpleados) {
31        System.out.println("Nombre: " + i.dameNombre() +
32            " Sueldo: " + i.dameSueldo() +
33            " Fecha de alta : " + i.dameFechaContrato());
34    }
35
36 }
37
38 }

```

Ejecutamos:

Un miembro ha tomado la decisión de: Dar más días de vacaciones..



Interfaces y clases internas. Interfaces III (Vídeo 51)

```
26 System.out.println(jefa_finanzas.tomar_decisiones("Dar más días de vacaciones."));
27
28 misEmpleados[3].tomar_decisiones("Me cojo un día de permiso");|
29
```

Si en la línea 20 intentamos ejecutar el método `tomar_decisiones`, este nos dará error ya que esta interface no está implementada en la clase `Empleado`.

Necesito poder utilizar el método de la interface `tomar_decisiones`, tanto a objetos de tipo `Empleado` como de `Jefatura`.

Lo borramos de la clase `Jefatura`.

```
class Jefatura extends Empleado {
```

Y lo pasamos a `Empleado`.

```
41 class Empleado implements Comparable, Jefes{
```

Eso obliga que implementemos el método `tomar_decisiones`.

```
class Empleado implements Comparable, Jefes{
    public Empleado(String nom, double sue, int agno, int mes, int dia) {
        nombre=nom;
        sueldo=sue;
        GregorianCalendar calendario= new GregorianCalendar(agno, mes-1,
dia);
        altaContrato=calendario.getTime();
        ++IdSiguiente;
        Id=IdSiguiente;
    }
}
```

```
public String tomar_decisiones(String decision) {
    return "Un miembro ha tomado la decisión de: " + decision+ ".";
}
```

```
public Empleado(String nom) {
    nombre=nom;
}

public String dameNombre() { //getter
    return nombre + " Id: " + Id;
}

public double dameSueldo() //getter
{
    return sueldo;
}

public Date dameFechaContrato() { //getter
    return altaContrato;
}
```

```

}

public void subeSueldo(double porcentaje) { //setter
    double aumento=sueldo*porcentaje/100;
    sueldo+=aumento;
}

public int compareTo(Object miObjeto) {
    Empleado otroEmpleado=(Empleado) miObjeto;
    if(this.Id<otroEmpleado.Id) {
        return -1;
    }
    if(this.Id>otroEmpleado.Id) {
        return 1;
    }
    return 0;
}

private String nombre;
private double sueldo;
private Date altaContrato;
private static int IdSiguiente;
private int Id;
}

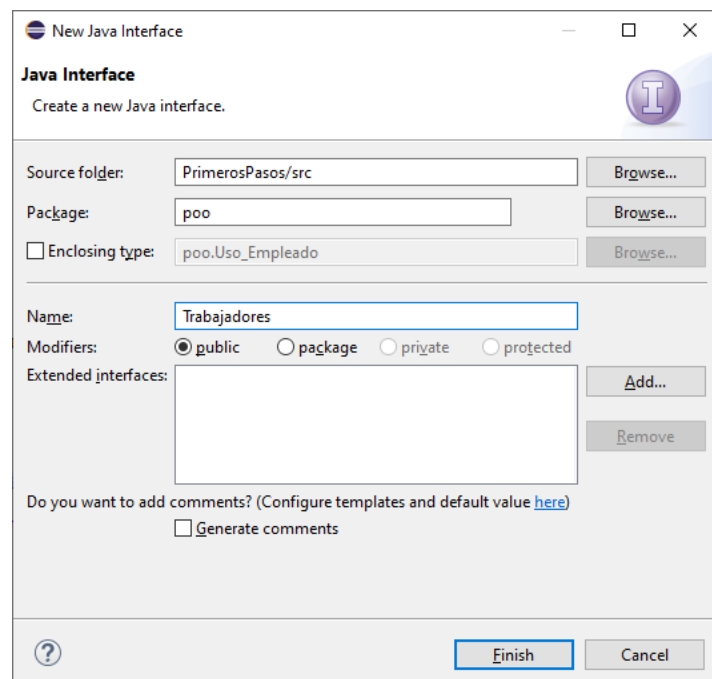
```

Como también la clase Jefatura hereda de la clase Empleado, también nos permite utilizar el método tomar_decisiones.

Pero en este ejemplo no tiene motivo que la interface Jefes la implementemos a Empleado, pues lo dejaremos como estaba antes.

Ahora para seguir con el ejemplo imaginarnos que la empresa para navidad estable un bonus y es para todos los trabajadores, desde los empleados hasta los jefes.

Vamos a crear una nueva interface llamada Trabajadores.



```

1 package poo;
2
3 public interface Trabajadores {
4     double establece_bones(double gratificacion);
5
6     double bonus_base = 1500; // son public static y final
7
8 }

```

Ahora queremos que la interface Jefes herede la jerarquía Trabajadores.

```

1 package poo;
2
3 public interface Jefes extends Trabajadores{
4
5     //public abstract String tomar_decisiones(String decision);
6
7     String tomar_decisiones(String decisión); // Puedes omitir public y abstract
8
9 }

```

Ahora en la clase Jefatura a implementar la interface Jefes como esta hereda de la interface Trabajadores, hemos de insertar los métodos de dicha interface.

```

102     class Jefatura extends Empleado implements Jefes {
103
104
105     public Jefatura(String nom, double sue, int agno, int mes, int dia) {
106         super(nom, sue, agno, mes, dia);
107     }
108
109     public String tomar_decisiones(String decision) {
110
111         return "Un miembro ha tomado la decisión de: " + decision+ ".";
112     }
113
114     public double establece_bonus(double gratificacion) {
115         double prima=2000;
116         return Trabajadores.bonus_base + prima;
117     }
118
119     public void estableceIncentivo(double b) {
120         incentivo=b;
121     }
122
123     public double dameSueldo() {
124         double sueldoJefe=super.dameSueldo();
125         return sueldoJefe + incentivo;
126     }
127
128
129     private double incentivo;
130 }

```

En la clase Empleado agregamos la interface Trabajadores.

Esto nos obliga a crear el correspondiente método.

```

41 class Empleado implements Comparable, Trabajadores{
42
43     public Empleado(String nom, double sue, int agno, int mes, int dia) {
44
45         nombre=nom;
46         sueldo=sue;
47         GregorianCalendar calendario= new GregorianCalendar(agno, mes-1, dia);
48         altaContrato=calendario.getTime();
49         ++IdSiguiente;
50         Id=IdSiguiente;
51
52
53     }
54
55     public double establece_bonus(double gratificacion) {
56         return Trabajadores.bonus_base + gratificacion;
57     }

```

Si nos vamos a la clase principal main y escribimos el siguiente código:

```

7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10
11     Jefatura jefe_RRHH=new Jefatura("Juan", 55000, 2006, 3, 25);
12     jefe_RRHH.estableceIncentivo(2570);
13
14     Empleado[] misEmpleados= new Empleado[6];
15
16     misEmpleados[0]=new Empleado("Ana", 30000, 1990, 12, 17);
17     misEmpleados[1]=new Empleado("Carlos", 50000,1995 ,6 ,2 );
18     misEmpleados[2]=new Empleado("Paco",25000 ,2002 ,3 , 15);
19     misEmpleados[3]=new Empleado("Antonio", 47500, 2009, 11, 9);
20     misEmpleados[4]=jefe_RRHH; //Polimorfismo en acción. Principio de sustitución
21     misEmpleados[5]=new Jefatura("María", 95000, 1999, 5, 26);
22
23     Jefatura jefa_finanzas=(Jefatura)misEmpleados[5];
24     jefa_finanzas.estableceIncentivo(55000);
25
26     System.out.println(jefa_finanzas.tomar_decisiones("Dar más días de vacaciones.));
27
28     System.out.println("El empleado " + misEmpleados[2].dameNombre() + " tiene un bonus de: " +
29     misEmpleados[2].establece_bonus(1000));
30
31     System.out.println("El jefe " +jefa_finanzas.dameNombre() + " tiene un bonus de: " +
32     jefa_finanzas.establece_bonus(500));
33
34     Arrays.sort(misEmpleados);
35
36     for(Empleado i: misEmpleados) {
37         System.out.println("Nombre: " + i.dameNombre() +
38         " Sueldo: " + i.dameSueldo() +
39         " Fecha de alta : " + i.dameFechaContrato());
40     }
41
42     }
43
44 }

```

Este será el resultado:

```

El empleado Paco Id: 4 tiene un bonus de: 2500.0
El jefe María Id: 6 tiene un bonus de: 3500.0

```

A slide with an orange background and a world map. It features the text 'CURSO JAVA' in large white letters, '51' in a yellow box, and 'INTERFACES Y CLASES INTERNAS INTERFACES III' in smaller white letters. Logos for 'eclipse' and 'Java' are also present.

CURSO JAVA 51

INTERFACES Y CLASES INTERNAS
INTERFACES III

eclipse Java

Interfaces y clases internas. Interfaces IV (Vídeo 52)

Temporizador

Hemos de recurrir a la clase Timer

La que vamos a utilizar pertenece al paquete javax.swing.Timer

Tiene dos parámetros: Timer(int delay, ActionListener listener)

Si al parámetro delay le pasamos 8000 que haga una acción cada 8 segundos.

Y como segundo parámetro un objeto de tipo interface.

Vamos a crear una nueva clase llamada PreubaTemporizador y que implemente el "public static void main(Sstring[] args).

```
1 package poo;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5 import java.util.*;
6 import javax.swing.Timer;
7
8 public class PreubaTemporizador {
9
10 public static void main(String[] args) {
11     // TODO Auto-generated method stub
12
13     DameLaHora oyente=new DameLaHora();
14
15     //ActionListener oyente = new DameLaHora();
16
17     Timer mitemporizador = new Timer(5000, oyente);
18
19     mitemporizador.start();
20
21     JOptionPane.showMessageDialog(null, "Pulsa aceptar para detener");
22
23     System.exit(0);
24
25 }
26
27 }
28
29 class DameLaHora implements ActionListener{
30 public void actionPerformed(ActionEvent e) {
31     Date ahora = new Date();
32     System.out.println("Te pongo la hora cada 5 sg: " + ahora );
33 }
34 }
```

En la línea 13 creamos una instancia perteneciente a la clase DameLaHora esto lo que hace es implementar ActionListener que se encuentra en la línea 29.

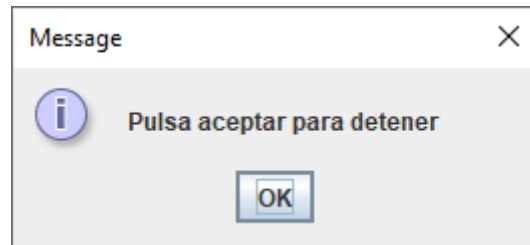
En la línea 17 utilizamos la clase Timer y el constructor de la clase Timer nos pedía una frecuencia de ejecución 5000 son 5 segundos y un objeto de ActionListener "oyente".

oyente no es de ActionListener si no de DameLaHora, sin embargo la clase DameLaHora implementa la interface Actionlistenr, subrayad en rojo.

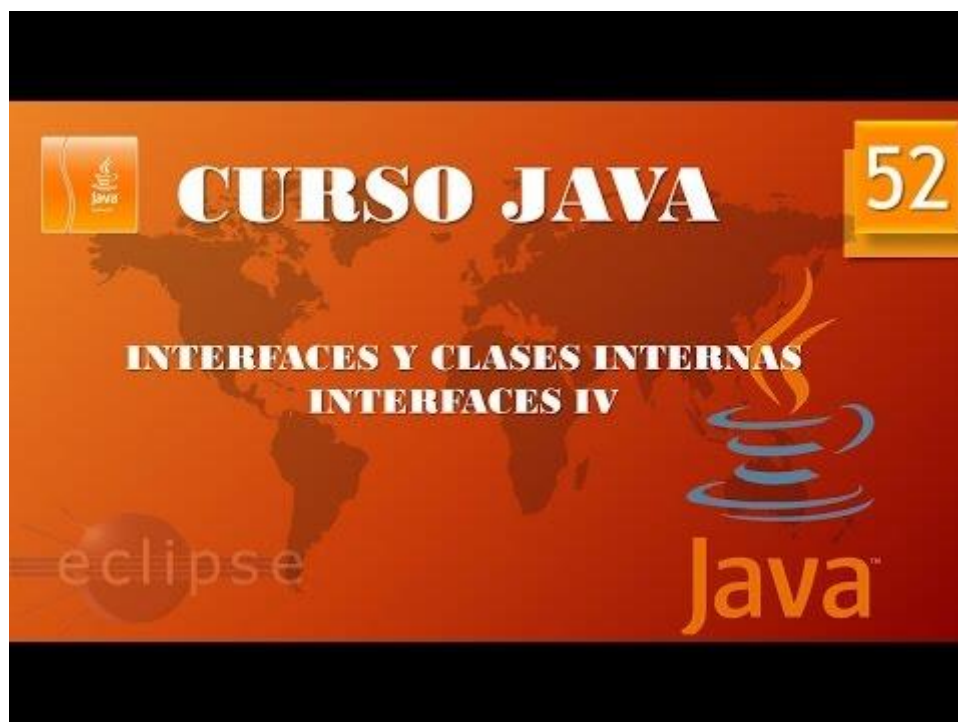
En la línea 19 le decimos que empieza a utilizar nuestro temporizador, cada 5 segundos y llama al método `actionPerformed`, subrayado de azul.

Este será el resultado.

```
Te pongo la hora cada 5 sg: Fri Sep 04 12:22:38 CEST 2020  
Te pongo la hora cada 5 sg: Fri Sep 04 12:22:43 CEST 2020  
Te pongo la hora cada 5 sg: Fri Sep 04 12:22:48 CEST 2020
```



Cuando pulsemos Ok detendrá el programa.



Interfaces y clases internas. Clases internas I. (Vídeo 53)

¿Qué son las clases internas? (Inner Class)

Como su nombre indica, una clase interna es una clase dentro de otra.

```
Public class Clase1{
```

```
    class Clase2{  
        Código de la Clase2  
    }
```

```
    Código de la Clase1
```

```
}
```

- Para acceder a los campos privados de una clase desde otra clase.
- Para ocultar una clase de otras pertenecientes al mismo paquete.
- Para crear clases internas “anónimas”, muy útiles para gestionar eventos y retrollamadas.
- Cuando solo una clase debe acceder a los campos de ejemplar de otra clase.

Vamos a crear una nueva clase llamada PruebaTemporizador2.

New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

Enclosing type:

Name:

Modifiers: public package private protected
 abstract final static

Superclass:

Interfaces:

Which method stubs would you like to create?

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

Constructors from superclass

Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

```

package poo;

import javax.swing.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.Timer;
import java.awt.Toolkit;

public class PruebaTemporizador2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Reloj mireloj=new Reloj(3000, true);
        mireloj.enMarcha();
        JOptionPane.showMessageDialog(null, "Pulsa Aceptar para
terminar");
        System.exit(0);
    }
}

```

```

class Reloj{
    public Reloj(int intervalo, boolean sonido){
        this.intervalo=intervalo;
        this.sonido=sonido;
    }
    public void enMarcha() {
        ActionListener oyente=new DameLaHora2();
        Timer mitemporizador=new Timer(intervalo, oyente);
        mitemporizador.start();
    }
    private int intervalo;
    private boolean sonido;
    private class DameLaHora2 implements ActionListener{
        public void actionPerformed(ActionEvent evento) {
            Date ahora=new Date();
            System.out.println("Te pongo la hora cada 3 segundos." +
                if(sonido) {
                    Toolkit.getDefaultToolkit().beep();
                }
        }
    }
}

```

No hace falta definir sonido ya que está definido fuera de su ámbito.

La ventaja de la clase interna puede acceder a los campos de la clase que la engloba.

Nos evitamos tener que hacer los métodos setters y getters correspondientes.

Si solo una clase tiene que acceder a los métodos correspondientes de una clase y la hacemos interna solo accederá esta clase y las variables estarán encapsuladas.



Interfaces y clases internas. Clases internas II. (Vídeo 54)

Clases internas locales ¿Qué son?

- Una clase dentro de un método.
- ¿Cuándo se utilizan estos tipos de clases y por qué?
 - Son útiles cuando solo se va a utilizar (instanciar) la clase interna una vez. El objetivo es simplificar aún más el código.
 - Su ámbito queda restringido al método donde son declaradas. ¿Ventajas?
 - Están muy “encapsuladas”. Ni siquiera la clase a la que pertenecen puede acceder a ella. Tan solo puede acceder a ella el método donde está declarada.
 - El código resulta más simple.

```
Class Clase_externa{
```

```
    Public void método(){
```

```
        Class clase_interna_local{  
            Código de la clase interna;  
        }
```

```
    Código del método;
```

```
}
```

```
Código de la clase externa;
```

```
}
```

Las clases internas locales no deben de llevar ningún modificador de acceso.

Esta clase interna está más encapsulada, ni siquiera el código que tenemos en la clase externa podrá acceder a esta clase a no ser que utilice el método correspondiente.

Vamos a cambiar el código de la clase PruebaTemporizador2 para ver si cumple y si lo cumple transformaremos la clase interna en una clase interna local.

```

1 package poo;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5 import java.util.*;
6 import javax.swing.Timer;
7 import java.awt.Toolkit;
8
9 public class PruebaTemporizador2 {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13         Reloj mireloj=new Reloj(3000, true);
14         mireloj.enMarcha();
15         JOptionPane.showMessageDialog(null, "Pulsa Aceptar para terminar");
16         System.exit(0);
17     }
18 }
19
20 class Reloj{
21     public Reloj(int intervalo, boolean sonido){
22         this.intervalo=intervalo;
23         this.sonido=sonido;
24     }
25     public void enMarcha() {
26         class DameLaHora2 implements ActionListener{
27             public void actionPerformed(ActionEvent evento) {
28                 Date ahora=new Date();
29                 System.out.println("Te pongo la hora cada 3 segundos." + ahora);
30                 if(sonido) {
31                     Toolkit.getDefaultToolkit().beep();
32                 }
33             }
34         }
35         ActionListener oyente=new DameLaHora2();
36         Timer mitemporizador=new Timer(intervalo, oyente);
37         mitemporizador.start();
38     }
39     private int intervalo;
40     private boolean sonido;
41
42
43 }

```

Lo que está seleccionado con una llave azul es una clase interna local, la otra clase está seleccionada con una llave de color rojo.

Una clase interna local puede acceder a los campos de la clase externa además de las variables del método enMarcha.

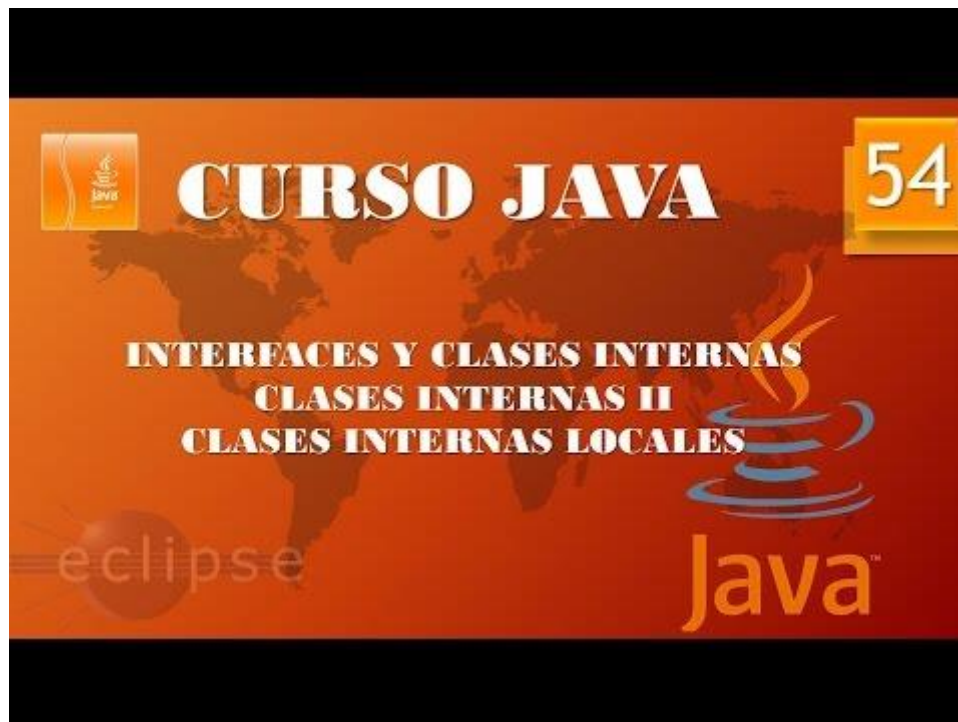
Vamos a simplificar más el código:

```

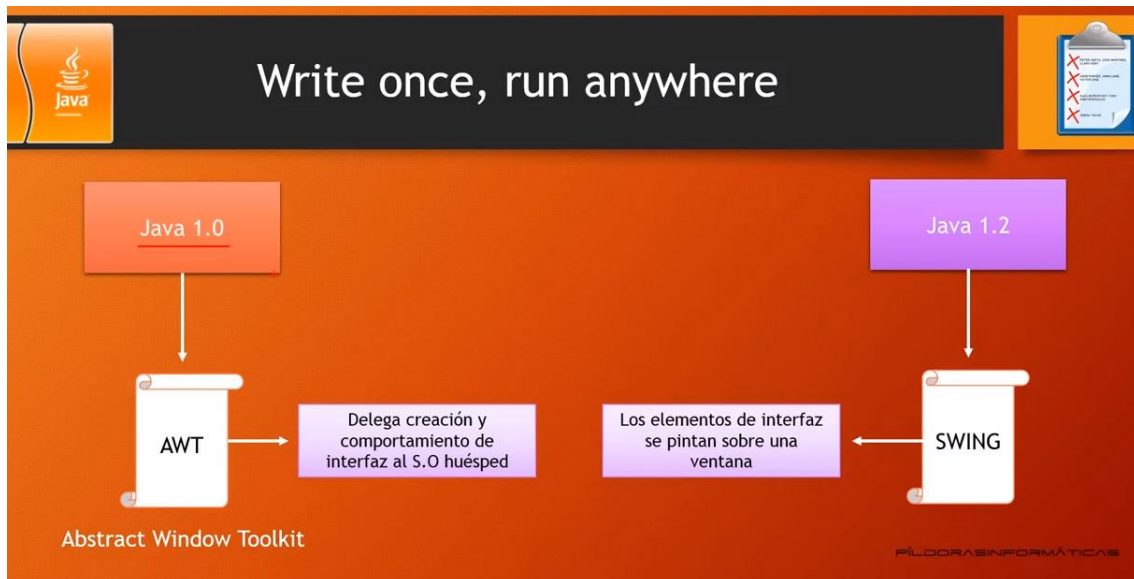
1 package poo;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5 import java.util.*;
6 import javax.swing.Timer;
7 import java.awt.Toolkit;
8
9 public class PruebaTemporizador2 {
10
11     public static void main(String[] args) {
12         // TODO Auto-generated method stub
13         Reloj mireloj=new Reloj();
14         mireloj.enMarcha(3000,true);
15         JOptionPane.showMessageDialog(null, "Pulsa Aceptar para terminar");
16         System.exit(0);
17     }
18 }
19
20 class Reloj{
21
22     public void enMarcha(int intervalo, final boolean sonido) {
23         class DameLaHora2 implements ActionListener{
24             public void actionPerformed(ActionEvent evento) {
25                 Date ahora=new Date();
26                 System.out.println("Te pongo la hora cada 3 segundos." + ahora);
27                 if(sonido) {
28                     Toolkit.getDefaultToolkit().beep();
29                 }
30             }
31         }
32     }
33     ActionListener oyente=new DameLaHora2();
34     Timer mitemporizador=new Timer(intervalo, oyente);
35     mitemporizador.start();
36 }
37 }

```





Aplicaciones gráficas Swing I. (Vídeo 55)



- Nacen invisibles. Se necesita el método setVisible para hacerlos visibles.
- Nacen con un tamaño inútil. Se necesita el método setSize para darles tamaño.
- Conviene decir qué debe hacer el programa si se cierra un frame.

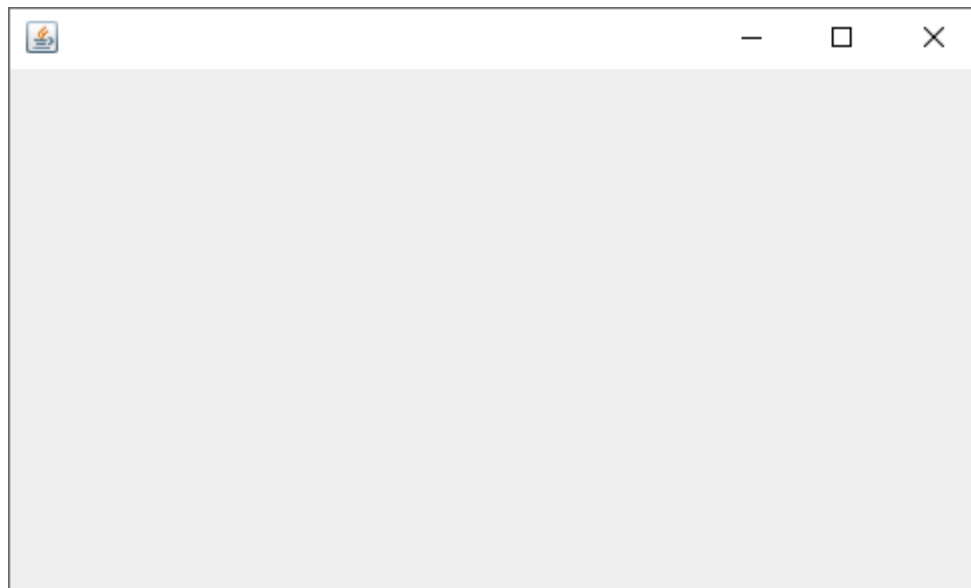
Para estos nuevos proyectos vamos a crear un paquete llamado gráficos.

Vamos a crear una nueva clase llamada CreandoMarcos con public static void main.

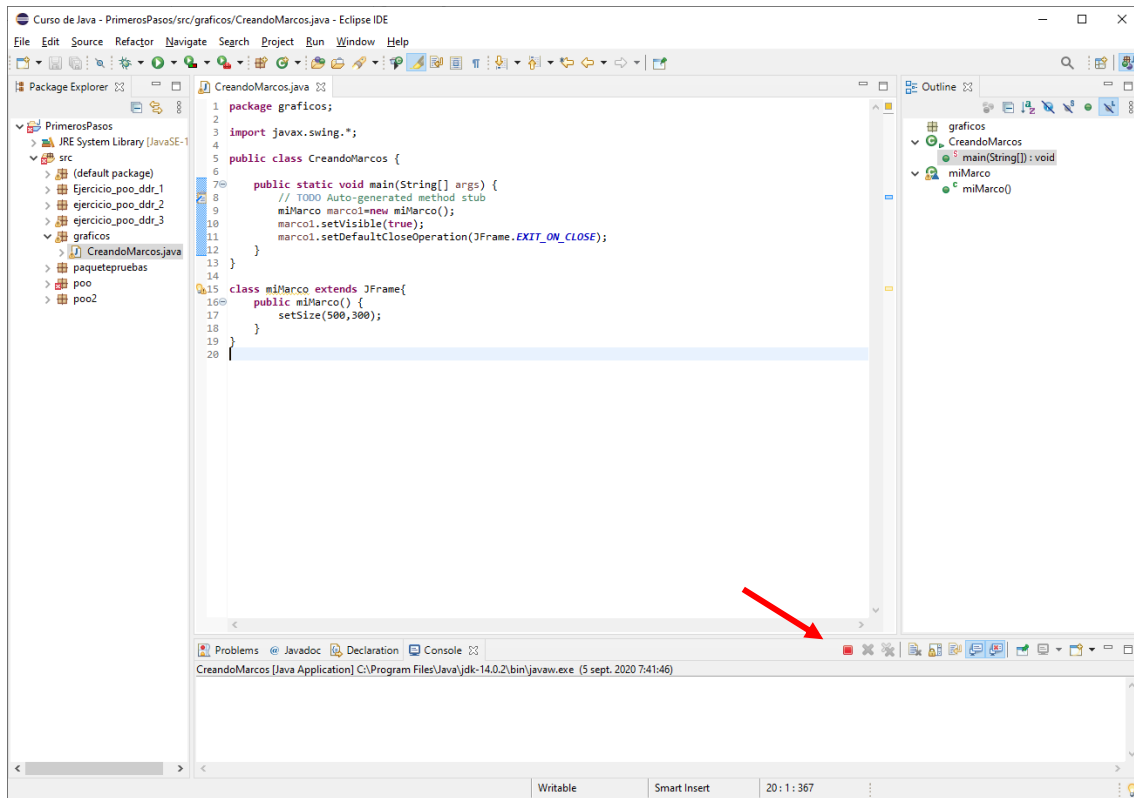
Vamos a escribir el siguiente código:

```
1 package graficos;
2
3 import javax.swing.*;
4
5 public class CreandoMarcos {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         miMarco marcol=new miMarco();
10        marcol.setVisible(true);
11        marcol.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    }
13 }
14
15 class miMarco extends JFrame{
16     public miMarco() {
17         setSize(500,300);
18     }
19 }
```

Este será el resultado:



Por defecto esta ventana se muestra en la parte superior izquierda de nuestro monitor.



Mientras la ventana está abierta el programa permanece en ejecución.

Podemos modificar la línea 11 con el siguiente código:

`Marco1.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);`

Con esta instrucción no cerramos el programa sino que estamos ocultando la ventana.

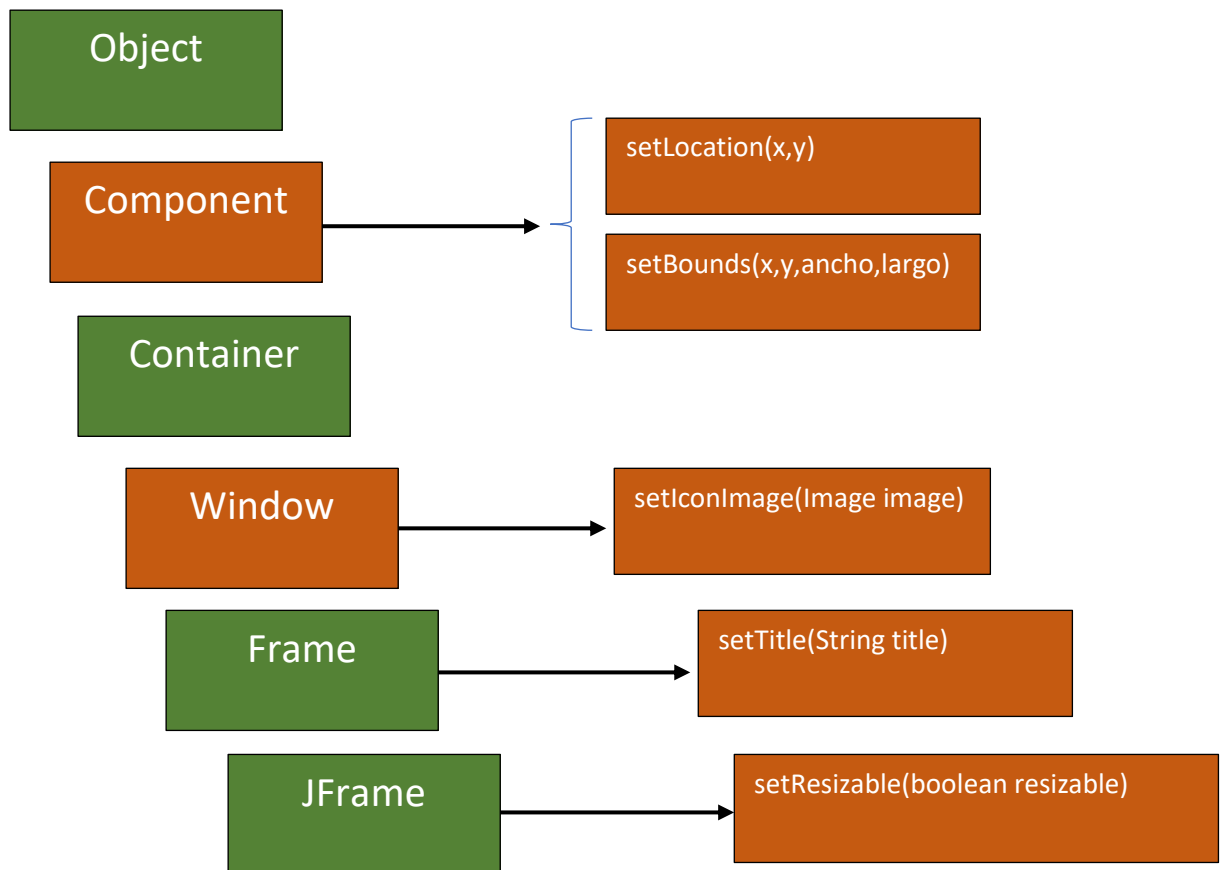




Aplicaciones gráficas. Swing II. Colocando el Frame. (Vídeo 56)

En este capítulo vamos a ver como mover este frame por la pantalla.

Métodos importantes de JFrame



```
1 package graficos;
2
3 import javax.swing.*;
4
5 public class CreandoMarcos {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         miMarco marco1=new miMarco();
10        marco1.setVisible(true);
11        marco1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    }
13 }
14
15 class miMarco extends JFrame{
16     public miMarco() {
17         setSize(500,300);
18         setLocation(500,300); //horizontal, vertical en pixeles
19     }
20 }
```

En la línea 17 estamos dando anchura y altura a nuestra ventana en pixeles.

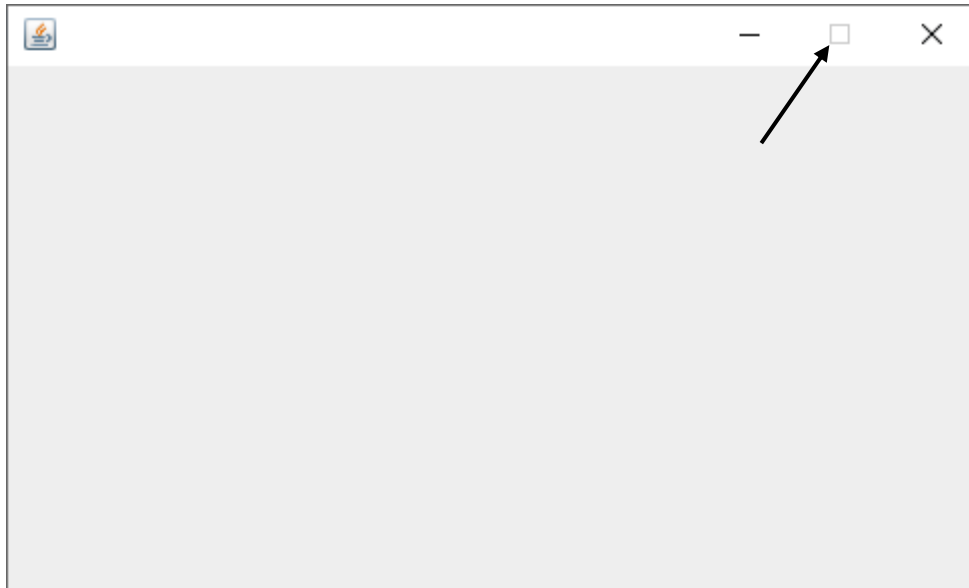
En la línea 18 le estamos diciendo la posición de la ventana en pixeles el primer parámetros corresponde a la posición horizontal y el segundo a la posición vertical.

```
1 package graficos;
2
3 import javax.swing.*;
4
5 public class CreandoMarcos {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         miMarco marco1=new miMarco();
10        marco1.setVisible(true);
11        marco1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    }
13 }
14
15 class miMarco extends JFrame{
16     public miMarco() {
17         //setSize(500,300);
18         //setLocation(500,300); //horizontal, vertical en pixeles
19
20         setBounds(500,300,500,300);
21     }
22 }
```

La línea 20 hace lo que hacen las dos líneas 17 y 18, da posición de la ventana y tamaño.

```
1 package graficos;
2
3 import javax.swing.*;
4
5 public class CreandoMarcos {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         miMarco marco1=new miMarco();
10        marco1.setVisible(true);
11        marco1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12    }
13 }
14
15 class miMarco extends JFrame{
16     public miMarco() {
17         //setSize(500,300);
18         //setLocation(500,300); //horizontal, vertical en pixeles
19
20         setBounds(500,300,500,300);
21
22         setResizable(false); ←
23     }
24 }
```

La línea 22 con setResizable de tipo boolean en modo false no impide modificar el tamaño de la ventana cuando nos poner en el borde de la misma, por defecto está en true.



Además el botón de maximizar está desactivado.

```
1 package graficos;
2
3 import java.awt.Frame;
4
5 import javax.swing.*;
6
7 public class CreandoMarcos {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         miMarco marco1=new miMarco();
12         marco1.setVisible(true);
13         marco1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15 }
16
17 class miMarco extends JFrame{
18     public miMarco() {
19         //setSize(500,300);
20         //setLocation(500,300); //horizontal, vertical en pixeles
21
22         setBounds(500,300,500,300);
23
24         //setResizable(false);
25
26         setExtendedState(Frame.MAXIMIZED_BOTH);
27     }
28 }
```

La línea 26 hace que la ventana se ejecute maximizada, es decir que ocupará toda la pantalla.

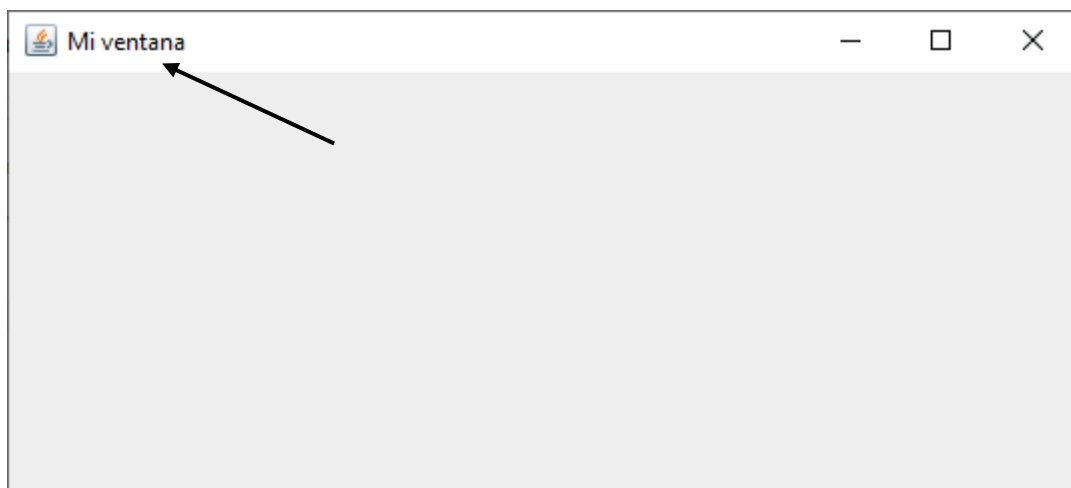
El método `setExtendedState` es un método que como parámetro necesita un valor de tipo `int`, el campo de clase `MAXIMIZED_BOTH` es una constante que tiene el valor 6, si cambiamos `Frame.MAXIMIZED_BOTH` por el número 6 hará exactamente lo mismo.


```

1 package graficos;
2
3 import java.awt.Frame;
4
5 import javax.swing.*;
6
7 public class CreandoMarcos {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         miMarco marco=new miMarco();
12         marco.setVisible(true);
13         marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15 }
16
17 class miMarco extends JFrame{
18     public miMarco() {
19         //setSize(500,300);
20         //setLocation(500,300); //horizontal, vertical en pixeles
21
22         setBounds(500,300,550,250);
23
24         //setResizable(false);
25
26         //setExtendedState(Frame.MAXIMIZED_BOTH);
27
28         setTitle("Mi ventana");
29     }
30 }

```

La línea 28 permite poner título a nuestra ventana.



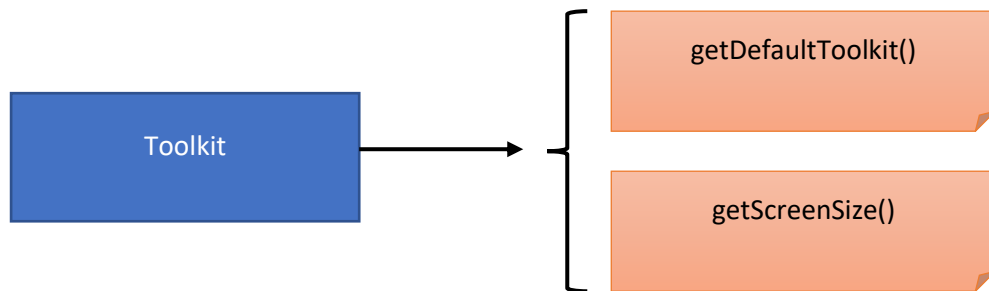
A slide with a dark orange background featuring a world map. The slide contains the following text and logos:

- Top left: A small Java logo.
- Top center: **CURSO JAVA** in large white letters.
- Top right: The number **56** in white on a yellow square background.
- Center: **INTERFACES DE USUARIO**, **SWING II**, and **Colocando el Frame** in white text.
- Bottom left: The word **eclipse** in a light, semi-transparent font.
- Bottom right: The Java logo (a blue coffee cup with orange steam) and the word **Java** in orange.

Aplicaciones gráficas. Swing III. Colocando el Frame II. (Vídeo 57)

En este capítulo vamos a colocar nuestro frame justo al medio de nuestra pantalla.

- Almacén de multitud de métodos que se comunican con el sistema huésped de ventanas.



Creamos una clase nueva:

The screenshot shows the "New Java Class" dialog box. The "Name" field is filled with "CreandoMarcoCentrado". The "Superclass" field is filled with "java.lang.Object". Under "Which method stubs would you like to create?", the checkboxes for "public static void main(String[] args)", "Constructors from superclass", and "Inherited abstract methods" are checked. The "Generate comments" checkbox is unchecked. The "Finish" button is highlighted.

```

1 package graficos;
2 import java.awt.Toolkit;
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class CreandoMarcoCentrado {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoCentrado mimarco=new MarcoCentrado();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        mimarco.setVisible(true);
13    }
14
15 }
16
17 class MarcoCentrado extends JFrame{
18
19     public MarcoCentrado() {
20         Toolkit mipantalla=Toolkit.getDefaultToolkit();
21         Dimension tamanoPantalla=mipantalla.getScreenSize();
22         int alturaPantalla=tamanoPantalla.height;
23         int anchoPantalla=tamanoPantalla.width;
24         setSize(anchoPantalla/2, alturaPantalla/2);
25         setLocation(anchoPantalla/4, alturaPantalla/4);
26     }
27
28 }

```

En la línea 17 creamos una clase llamada MarcoCentrado que tiene que heredar de JFrame.

En la línea 19 creamos un método llamado MarcoCenrado.

En la línea 20 creamos un objeto llamado mipantalla de tipo Toolkit.

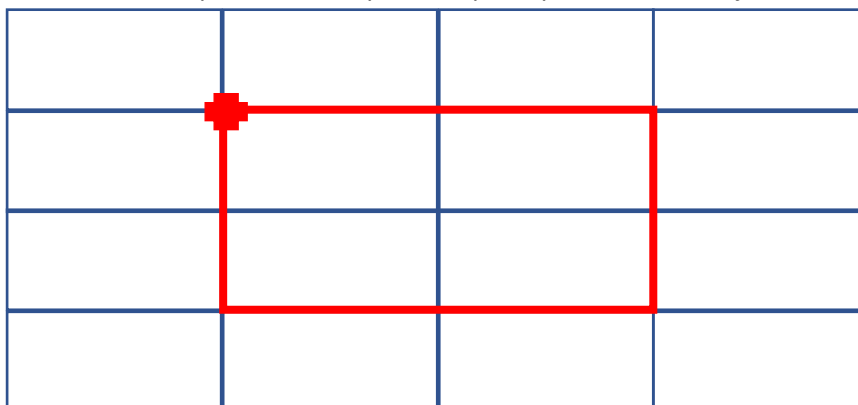
En la línea 21 creamos un objeto llamado tamanoPantalla de tipo Dimension que asume las dimensiones de nuestra pantalla.

En la línea 22 definimos una variable int llamada altuaPantalla que guarda la altura de nuestra pantalla en pixeles.

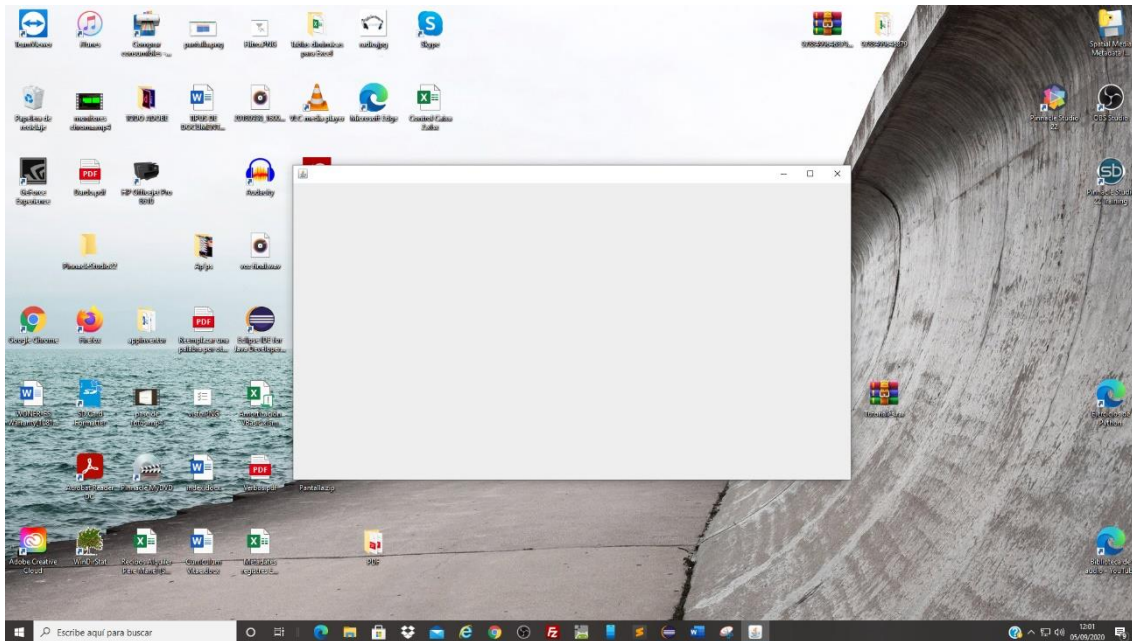
En la línea 23 definimos una variable int llamada anchoPantalla que guarda la anchura de nuestra pantalla en pixeles.

En la línea 24 le decimos que nuestra ventana tiene que tener en altura la mitad del tamaños de nuestra pantalla y la mitad de la anchura del tamaño de nuestra pantalla.

En la línea 25 colocamos la pantalla en la posición para que se encuadre justo en la mitad.



Si ejecutamos este será el resultado:

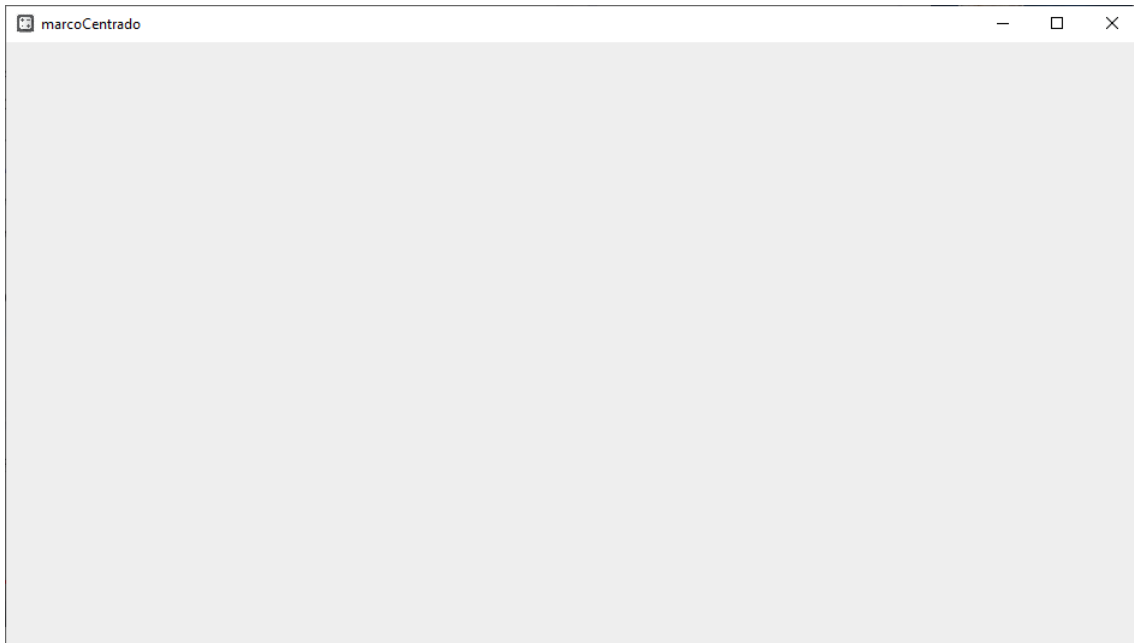


```
1 package graficos;
2 import java.awt.Toolkit;
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class CreandoMarcoCentrado {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoCentrado mimarco=new MarcoCentrado();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        mimarco.setVisible(true);
13    }
14 }
15
16
17 class MarcoCentrado extends JFrame{
18
19     public MarcoCentrado() {
20         Toolkit mipantalla=Toolkit.getDefaultToolkit();
21         Dimension tamanoPantalla=mipantalla.getScreenSize();
22         int alturaPantalla=tamanoPantalla.height;
23         int anchoPantalla=tamanoPantalla.width;
24         setSize(anchoPantalla/2, alturaPantalla/2);
25         setLocation(anchoPantalla/4, alturaPantalla/4);
26         setTitle("marcoCentrado");
27         Image miIcono=mipantalla.getImage("calculadora.jpg");
28         setIconImage(miIcono);
29     }
30 }
31 }
```

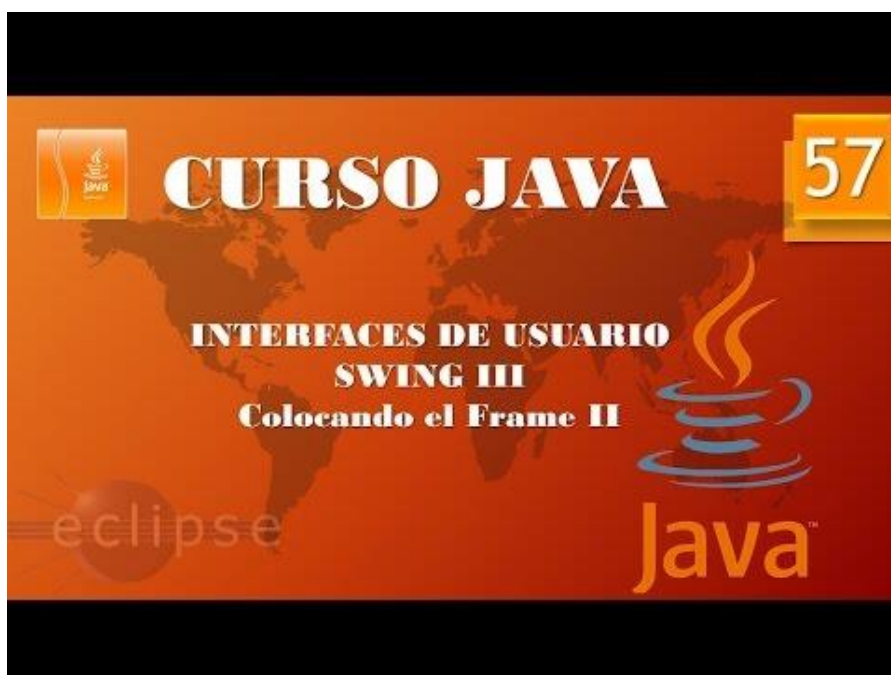
En la línea 26 ponemos un título en la ventana.

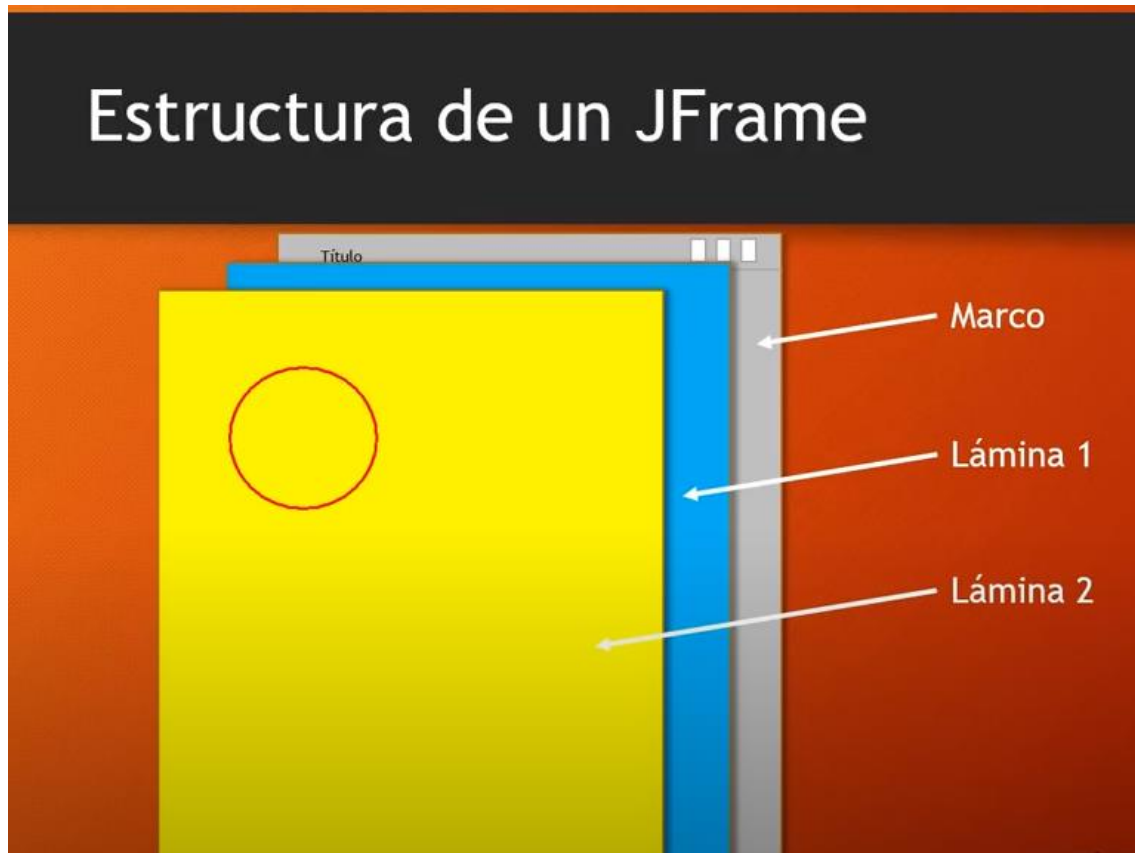
En la línea 27 creamos un objeto de tipo Image donde le decimos que icono queremos que muestre nuestra ventana, este archivo tiene que estar guardado en la carpeta del proyecto.

En la línea 28 le asignamos el icono.



Si el icono lo guardamos en alguna carpeta de nuestro proyecto habrá que indicarle la ruta relativa de nuestro archivo.





- La clase JPanel es la encargada de construir láminas donde poder dibujar y escribir. Debemos crear una clase que herede de JPanel.

- Object
 - Component
 - Container
 - JComponent → paintComponent(Graphics g)
 - JPanel

El método `paintComponent(Graphics g)`: se encarga de dibujar en la lámina.

`Graphics` es una clase de java que se encarga en proporcionarnos todos los elementos necesarios para dibujar gráficos, que tiene un método llamado `drawString`.

Vamos a crear una clase llamada `EscribiendoEnMarco`.

```

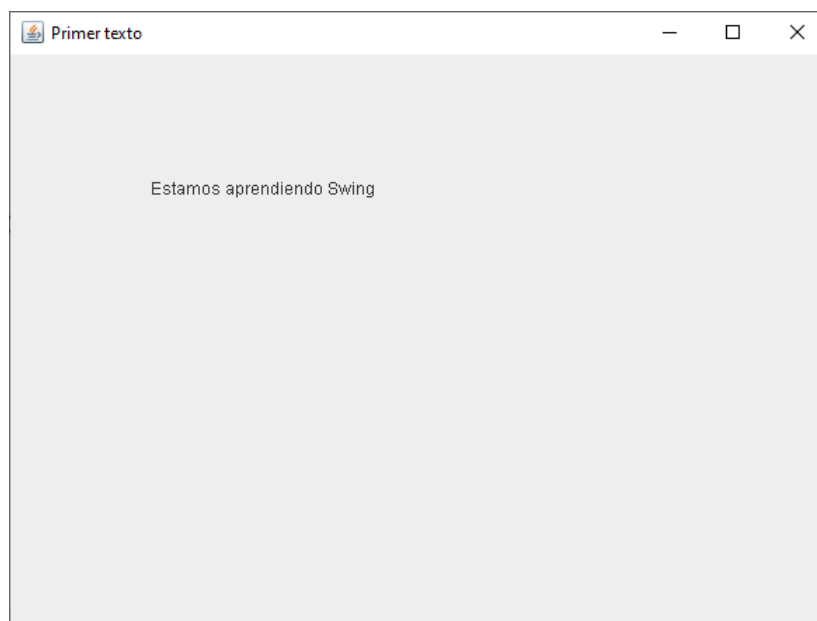
1 package graficos;
2 import javax.swing.*;
3 import java.awt.*;
4
5 public class EscribiendoEnMarco {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoConTexto mimarco=new MarcoConTexto();
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12 }
13
14 }
15
16 class MarcoConTexto extends JFrame{
17     public MarcoConTexto() {
18         setVisible(true);
19         setSize(600,450);
20         setLocation(400,200);
21         setTitle("Primer texto");
22
23         Lamina milamina=new Lamina();
24         add(milamina);
25     }
26 }
27
28 class Lamina extends JPanel{
29     public void paintComponent(Graphics g) {
30         super.paintComponent(g);
31         g.drawString("Estamos aprendiendo Swing", 100, 100);
32     }
33 }
34 }

```

En la línea 16 creamos la clase MarcoConTexto que herede de JFrame.

En la línea 28 creamos la clase Lamina que hereda de JPanel.

Este será el resultado:



The image shows a course cover with a dark orange background and a world map. At the top left is the Eclipse logo. The main title 'CURSO JAVA' is in large white letters. To its right, the number '58' is in a yellow box. Below the title, the text 'INTERFACES DE USUARIO SWING IV' and 'Escribiendo en el Frame' is centered. At the bottom left is the Eclipse logo and at the bottom right is the Java logo.

CURSO JAVA 58

**INTERFACES DE USUARIO
SWING IV**
Escribiendo en el Frame

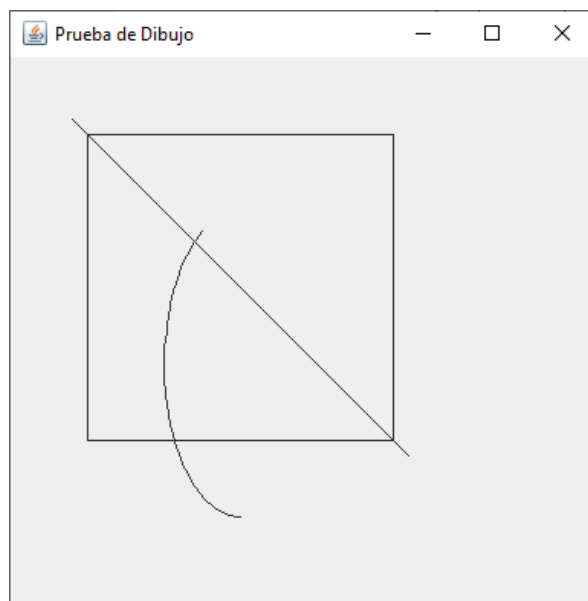
eclipse Java

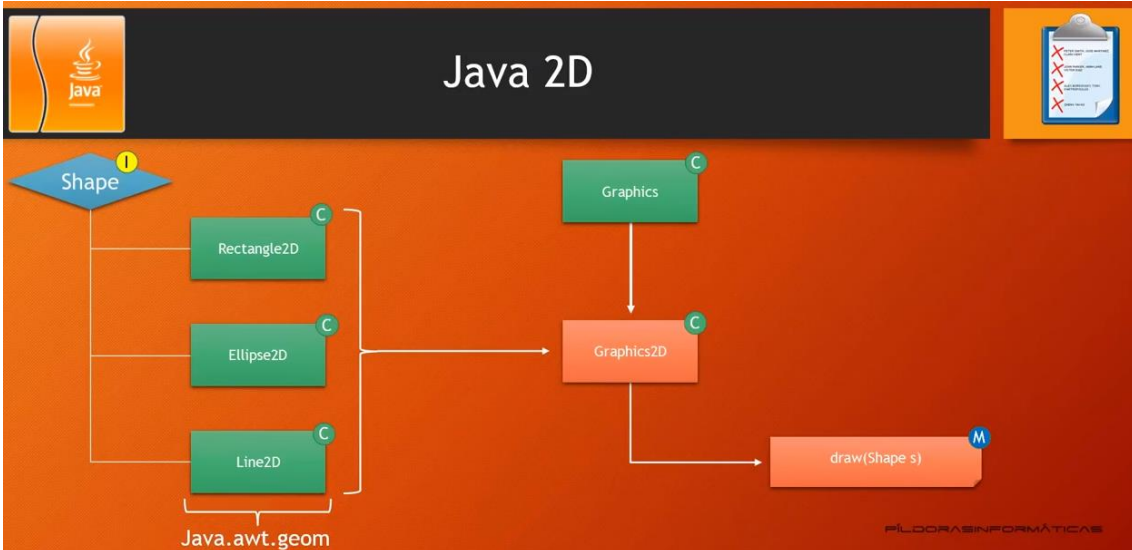
Aplicaciones gráficas. Swing V. Dibujando en el Frame. (Vídeo 59)

Vamos a crear una clase llamada PruebaDibujo.

```
1 package graficos;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class PruebaDibujo {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoConDibujos mimarco=new MarcoConDibujos();
11        mimarco.setVisible(true);
12        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13    }
14
15 }
16
17 class MarcoConDibujos extends JFrame{
18     public MarcoConDibujos(){
19         setTitle("Prueba de Dibujo");
20         setSize(400,400);
21         LaminaConFiguras milamina=new LaminaConFiguras();
22         add(milamina);
23     }
24 }
25
26
27 class LaminaConFiguras extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         g.drawRect(50, 50, 200, 200);
31         g.drawLine(40,40,260,260);
32         g.drawArc(100,100,100,200,120,150);
33     }
34 }
35 }
```

Este será el resultado:





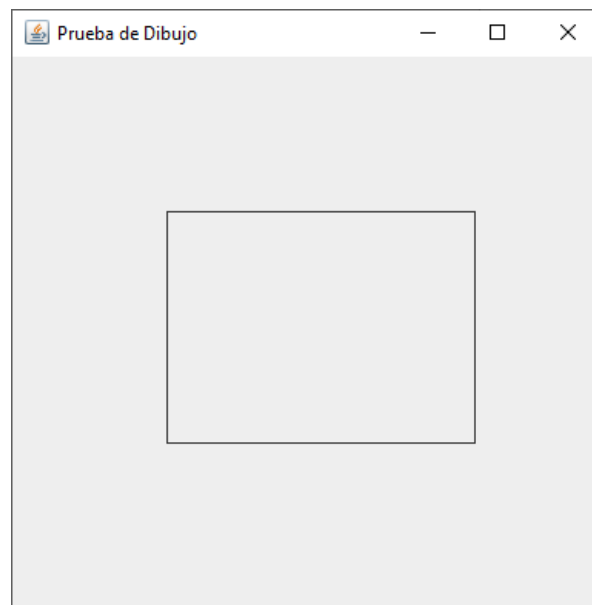
The slide features a dark orange background with a world map and the Java logo. In the top left corner is the Java logo. The main title is **CURSO JAVA** in large white letters. To the right of the title is a yellow box containing the number **59**. Below the title, the text reads **INTERFACES DE USUARIO SWING V** and **Dibujando en el Frame**. At the bottom left, the word **eclipse** is written in a light font. At the bottom right, the **Java** logo is displayed in its characteristic blue and orange colors.

Aplicaciones gráficas. Swing VI. Dibujando en el Frame II (Video 60)

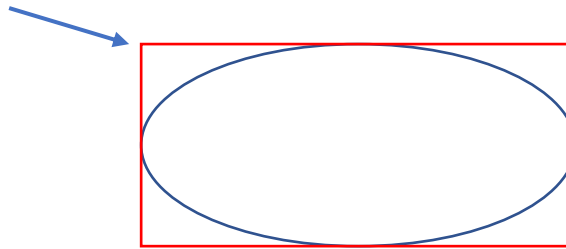
```
1 package graficos;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.geom.*;
6
7 public class PruebaDibujo {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         MarcoConDibujos mimarco=new MarcoConDibujos();
12         mimarco.setVisible(true);
13         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15
16 }
17
18 class MarcoConDibujos extends JFrame{
19     public MarcoConDibujos(){
20         setTitle("Prueba de Dibujo");
21         setSize(400,400);
22         LaminaConFiguras milamina=new LaminaConFiguras();
23         add(milamina);
24     }
25 }
26
27
28 class LaminaConFiguras extends JPanel{
29     public void paintComponent(Graphics g) {
30         super.paintComponent(g);
31         Graphics2D g2=(Graphics2D) g;
32         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
33         g2.draw(rectangulo);
34     }
35 }
36
37
```

Refundición

Este será el resultado:



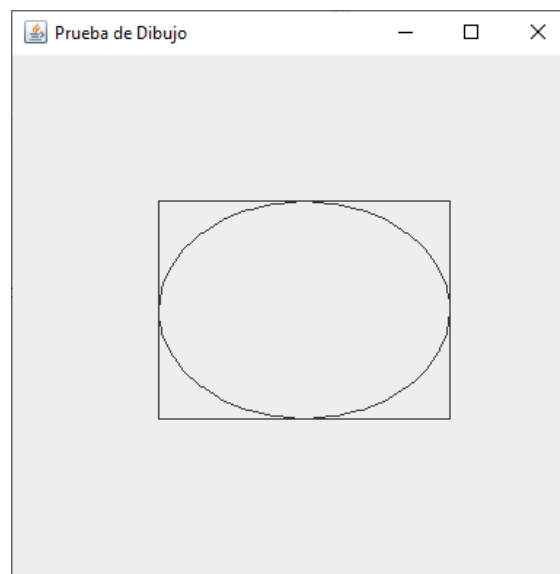
Vamos a dibujar una elipse, como referencia.



En la clase LaminaConFiguras agregaríamos el siguiente código:

```
28 class LaminaConFiguras extends JPanel{
29     public void paintComponent(Graphics g) {
30         super.paintComponent(g);
31         Graphics2D g2=(Graphics2D) g;
32         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
33         g2.draw(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double(100,100,200,150);
35         g2.draw(elipse);
36     }
37 }
38 }
```

Este será el resultado:



También se puede hacer de la siguiente forma:

```

28 class LaminaConFiguras extends JPanel{
29     public void paintComponent(Graphics g) {
30         super.paintComponent(g);
31         Graphics2D g2=(Graphics2D) g;
32         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
33         g2.draw(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double();
35         elipse setFrame(rectangulo);
36         g2.draw(elipse);
37     }
38 }
39 }

```

El resultado será el mismo.

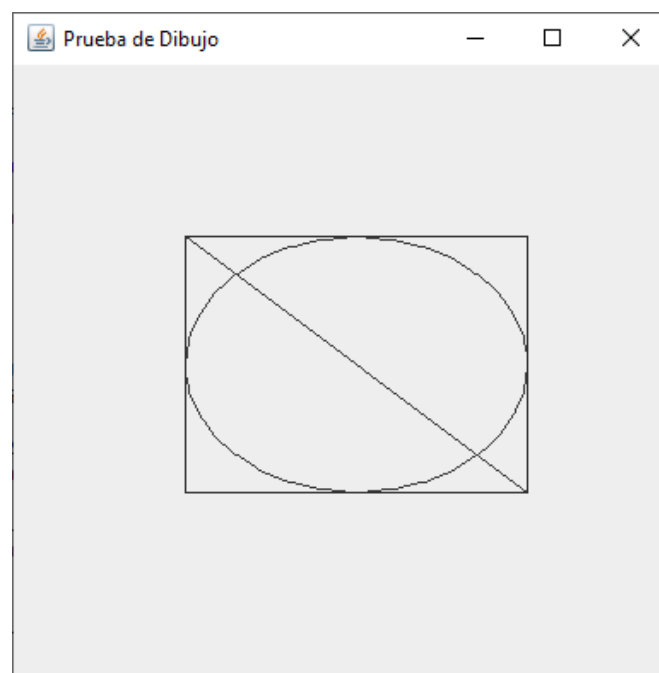
Ahora vamos a instanciar la clase dentro del método draw para agregar una línea.

```

28 class LaminaConFiguras extends JPanel{
29     public void paintComponent(Graphics g) {
30         super.paintComponent(g);
31         Graphics2D g2=(Graphics2D) g;
32         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
33         g2.draw(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double();
35         elipse setFrame(rectangulo);
36         g2.draw(elipse);
37     }
38     g2.draw(new Line2D.Double(100,100,300,250));
39 }
40 }
41 }
42 }
43 }

```

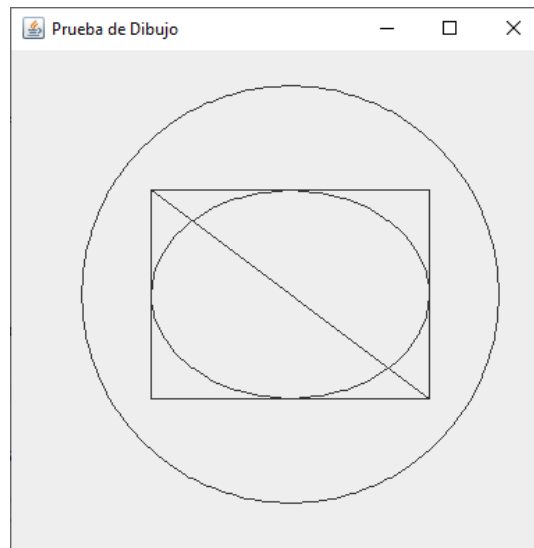
Este será el resultado:



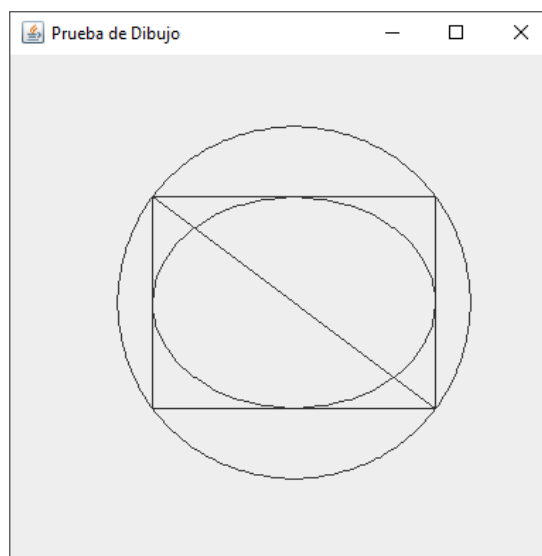
Ahora dibujaremos un círculo que abarque todas las figuras.

```
28 class LaminaConFiguras extends JPanel{
29     public void paintComponent(Graphics g) {
30         super.paintComponent(g);
31         Graphics2D g2=(Graphics2D) g;
32         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
33         g2.draw(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double();
35         elipse setFrame(rectangulo);
36         g2.draw(elipse);
37
38         g2.draw(new Line2D.Double(100,100,300,250));
39
40         double CentroenX=rectangulo.getCenterX();
41         double CentroenY=rectangulo.getCenterY();
42         double radio=150;
43         Ellipse2D circulo=new Ellipse2D.Double();
44         circulo.setFrameFromCenter(CentroenX, CentroenY, CentroenX+radio, CentroenY+radio);
45         g2.draw(circulo);
46     }
47 }
48 }
```

Este será el resultado:



¿Que tendremos que modificar para que quede de la siguiente forma?



The image shows a course cover with a dark orange to red gradient background. At the top left is a small logo with the word 'eclipse'. The main title 'CURSO JAVA' is in large white letters. To the right of the title is a yellow box containing the number '60'. Below the title, the text 'INTERFACES DE USUARIO SWING VI' and 'Dibujando en el Frame II' is centered. At the bottom left is the 'eclipse' logo, and at the bottom right is the 'Java' logo with its characteristic coffee cup icon. A faint world map is visible in the background.

CURSO JAVA 60

**INTERFACES DE USUARIO
SWING VI**
Dibujando en el Frame II

eclipse Java



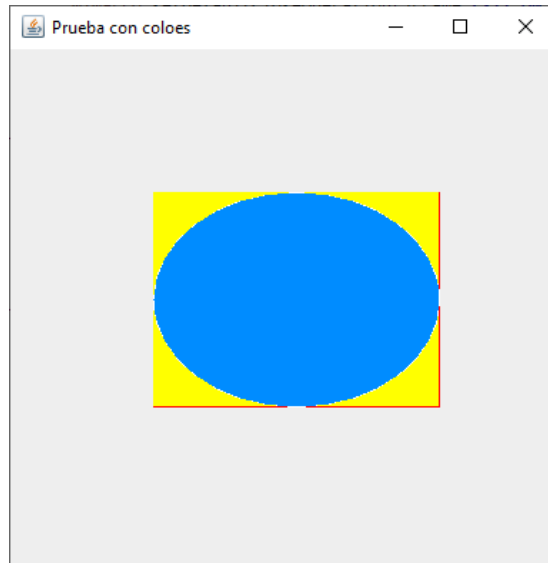
```

1 package graficos;
2 import java.awt.*;
3 import javax.swing.*;
4 import java.awt.geom.*;
5
6 public class TrabajandoColores {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoConColor mimarco=new MarcoConColor();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        mimarco.setVisible(true);
13    }
14
15 }
16
17 class MarcoConColor extends JFrame{
18     public MarcoConColor() {
19         setTitle("Prueba con colores");
20         setSize(400,400);
21         LaminaConColor milamina=new LaminaConColor();
22         add(milamina);
23     }
24 }
25 class LaminaConColor extends JPanel{
26     public void paintComponent(Graphics g) {
27         super.paintComponent(g);
28         Graphics2D g2=(Graphics2D) g;
29         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
30         g2.setPaint(Color.RED); ←
31         g2.draw(rectangulo);
32         g2.setPaint(Color.YELLOW); ←
33         g2.fill(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double();
35         elipse setFrame(rectangulo);
36         g2.setPaint(Color.WHITE); ←
37         g2.draw(elipse);
38         g2.setPaint(new Color(0,140,255)); ←
39         g2.fill(elipse);
40     }
41 }

```

Fill para el relleno y draw par la línea.

Este será el resultado:

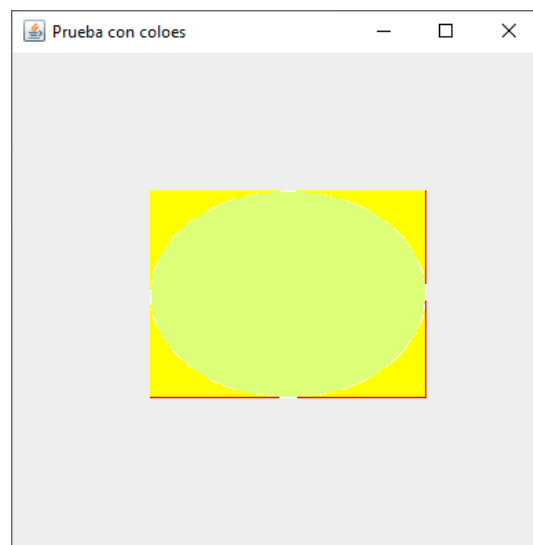


```
25 class LaminaConColor extends JPanel {
26     public void paintComponent(Graphics g) {
27         super.paintComponent(g);
28         Graphics2D g2=(Graphics2D) g;
29         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
30         g2.setPaint(Color.RED);
31         g2.draw(rectangulo);
32         g2.setPaint(Color.YELLOW);
33         g2.fill(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double();
35         elipse setFrame(rectangulo);
36         g2.setPaint(Color.WHITE);
37         g2.draw(elipse);
38         g2.setPaint(new Color(109,172,59).brighter().brighter());
39         g2.fill(elipse);
40     }
41 }
```

Dar más brillo



Antes



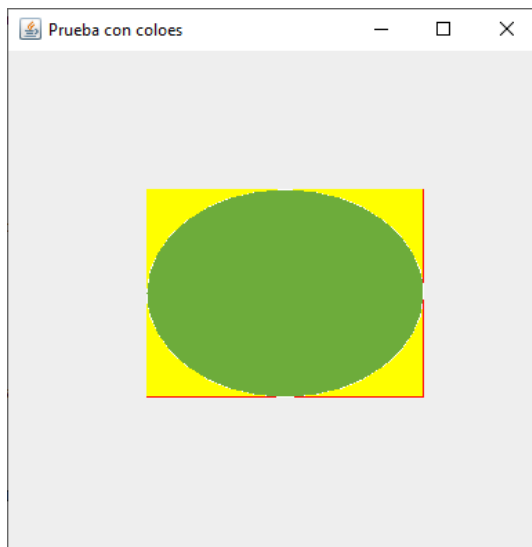
Después

```

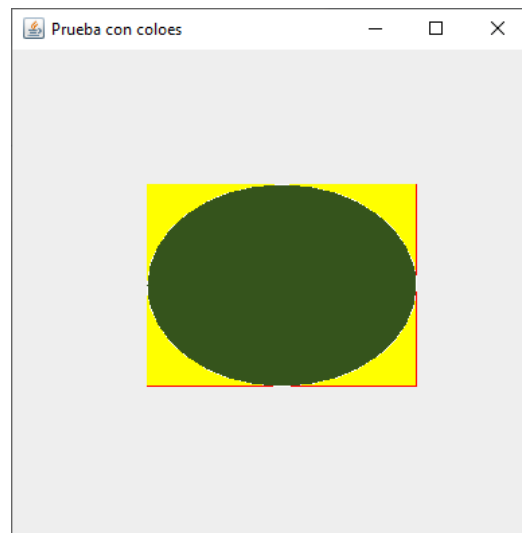
25 class LaminaConColor extends JPanel{
26     public void paintComponent(Graphics g) {
27         super.paintComponent(g);
28         Graphics2D g2=(Graphics2D) g;
29         Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
30         g2.setPaint(Color.RED);
31         g2.draw(rectangulo);
32         g2.setPaint(Color.YELLOW);
33         g2.fill(rectangulo);
34         Ellipse2D elipse=new Ellipse2D.Double();
35         elipse setFrame(rectangulo);
36         g2.setPaint(Color.WHITE);
37         g2.draw(elipse);
38         g2.setPaint(new Color(109,172,59).darker().darker());
39         g2.fill(elipse);
40     }
41 }

```

Oscurece



Antes



Después

```

class LaminaConColor extends JPanel{
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2=(Graphics2D) g;
        Rectangle2D rectangulo=new Rectangle2D.Double(100,100,200,150);
        g2.setPaint(Color.RED);
        g2.draw(rectangulo);
        g2.setPaint(Color.YELLOW);
        g2.fill(rectangulo);
        Ellipse2D elipse=new Ellipse2D.Double();
        elipse setFrame(rectangulo);
        g2.setPaint(Color.WHITE);
        g2.draw(elipse);

        Color micolor=new Color(125,189,200);
        g2.setPaint(micolor);
        g2.fill(elipse);
    }
}

```

Podemos crear nuestro color y utilizarlo las veces que sea necesario.

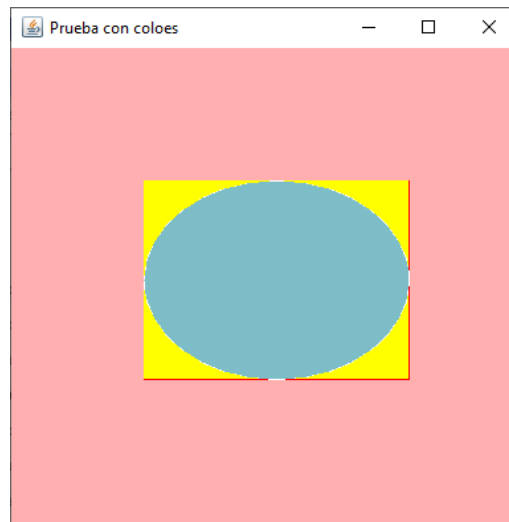
```

17 class MarcoConColor extends JFrame{
18     public MarcoConColor() {
19         setTitle("Prueba con colores");
20         setSize(400,400);
21         LaminaConColor milamina=new LaminaConColor();
22         add(milamina);
23         milamina.setBackground(Color.PINK); ←
24     }
25 }

```

Si queremos cambiar el color de la lámina en la clase MarcoConColor definimos el color.

Este será el resultado:

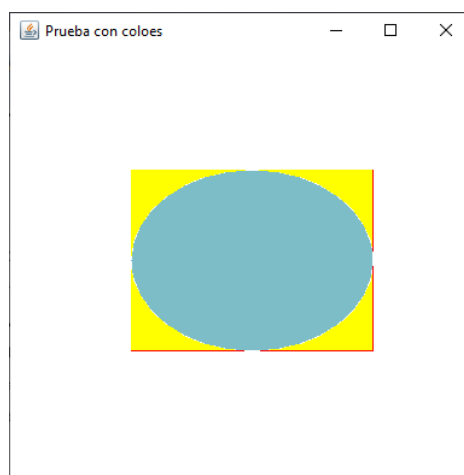


```

17 class MarcoConColor extends JFrame{
18     public MarcoConColor() {
19         setTitle("Prueba con colores");
20         setSize(400,400);
21         LaminaConColor milamina=new LaminaConColor();
22         add(milamina);
23         milamina.setBackground(SystemColor.window); ←
24     }
25 }

```

Si queremos que nuestra ventana tenga una apariencia a las ventanas de nuestro sistema operativo estableceremos SystemColor.window y según en qué sistema operativo ejecutemos la aplicación esta tendrá un aspecto u otro.



A slide with an orange-to-red gradient background. At the top left is a small Java logo. The main title 'CURSO JAVA' is in large white letters. To its right, the number '61' is in a yellow box. Below the title, the text 'INTERFACES DE USUARIO SWING VII' and 'Colores en el Frame' is centered. On the right is the Java logo. At the bottom left is the Eclipse logo, and at the bottom right is the Java logo. A world map is faintly visible in the background.

CURSO JAVA 61

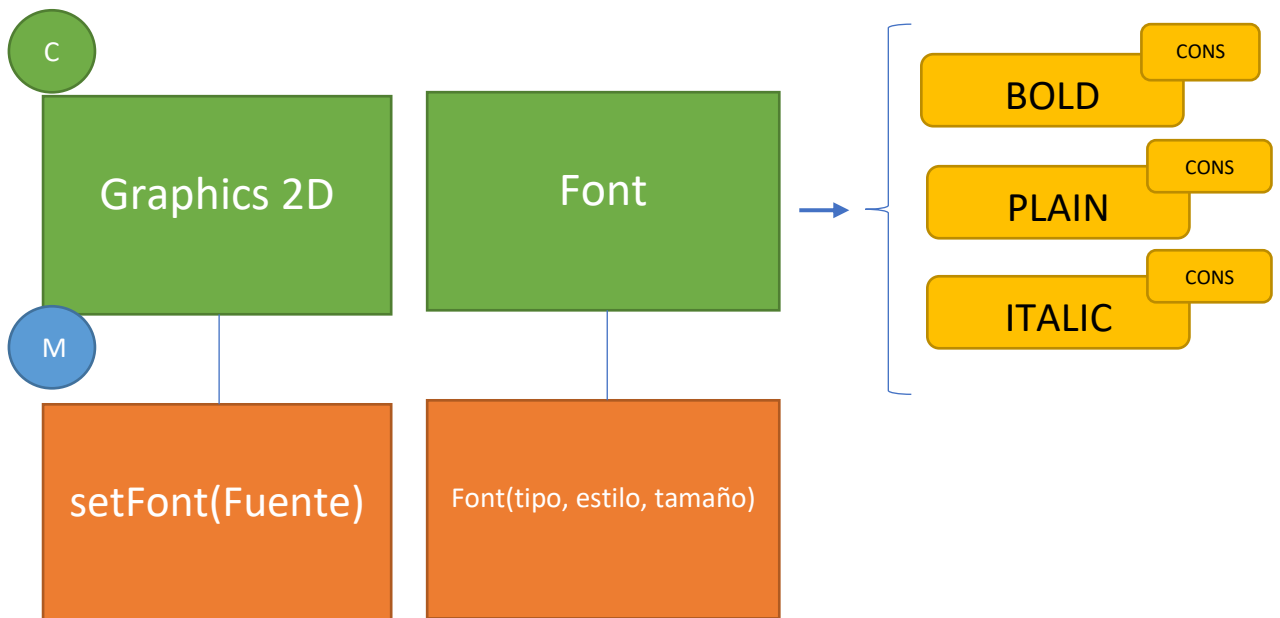
**INTERFACES DE USUARIO
SWING VII**
Colores en el Frame

eclipse Java

Aplicaciones gráficas. Swing VIII. Cambiando la letra en el Frame.

(Vídeo 62)

Trabajando con fuentes



Vamos a elaborar una clase de pruebas para saber que fuentes tenemos instaladas en nuestro ordenador.

```
1 package graficos;
2
3 import java.awt.GraphicsEnvironment;
4
5 public class PreubaFuentes {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10        String [] nombresDeFuentes=
11            GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
12
13        for(String nombredelaFuente: nombresDeFuentes) {
14            System.out.println(nombredelaFuente);
15        }
16    }
17 }
18 }
19 }
20 }
```

Este será el resultado:

```
|18thCentury
ABeeZee
ABSALOM
AcmeFont
Advent Pro
Advent Pro ExtraLight
Advent Pro Light
Advent Pro Medium
Advent Pro SemiBold
```

Para poder ver 10 tipos de fuentes por línea vamos a realizar los siguientes cambios.

```
1 package graficos;
2
3 import java.awt.GraphicsEnvironment;
4
5 public class PreubaFuentes {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9
10        String [] nombresDeFuentes=
11            GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
12
13        int contador=0;
14        for(String nombredelaFuente: nombresDeFuentes) {
15            System.out.print(nombredelaFuente+ ", ");
16            contador++;
17            if (contador==10) {
18                System.out.println();
19                contador=0;
20            }
21        }
22    }
23 }
24
25 }
```

Este será el resultado:

```
18thCentury, AbeeZee, ABSALOM, AceFont, Advent Pro, Advent Pro Extralight, Advent Pro Light, Advent Pro Medium, Advent Pro SemiBold, Advent Pro Thin,
Agency FB, Alex Brush, Alfredo, Algerian, Alibi, Alien Encounters, Almonte Snow, Amatic, Amatic SC, Amethyst,
Anticfont, Antonio, AR BERKLEY, AR BLANKA, AR BONNIE, AR CARTER, AR CENA, AR CHRISTY, AR DARLING, AR DECODE,
AR DELANEY, AR DESTINE, AR ESSECE, AR HERMAN, AR JULIAN, archicon, Architects Daughter, archive Narrow, Arial, Arial Black,
Arial Narrow, Arial Nova, Arial Nova Cond, Arial Nova Cond Light, Arial Nova Light, Arial Rounded MT Bold, Asimov, Autumn, Averia, Averia Sans,
Averia Serif, Baby Kruffy, Bahnschrift, Balharzar, Baskerville Old Face, Bastion, BATAVIA, Bauhaus 93, Bell MT, Bellota,
Belwe Rd BT, Belwe Cn BT, Belwe Lt BT, Berlin Sans FB, Berlin Sans FB Demi, Bernard MT Condensed, Blackadder ITC, Blackout, Blade Runner Movie Font, BN Jinx,
BN Machine, Bobcat, Bodoni MT, Bodoni MT Black, Bodoni MT Condensed, Bodoni MT Poster Compressed, BolsterBold, Book Antiqua, Bookman Old Style, Bookshelf Symbol 7,
Borealis, BOUTON International Symbols, Bradley Hand ITC, Brandish, Britannic Bold, Broadway, Brush Script MT, Brussels, Cabinsketch, Calibri,
Calibri Light, Californian FB, Calisto MT, Calligraphic, Calvia, Cambria, Cambria Math, Candara, Candara Light, Candlex,
Candy Round BTN, Candy Round BTN Cond, Candy Round BTN Cond Lt, Candy Round BTN Lt, CASHIRA, Castellar, Centaur, Century, Century Gothic, Century Schoolbook,
Chiller, Chinyen, ChunkFive Roman, Clarendon, Clarendon Blk BT, Clarendon BT, Clarendon Cn BT, Clarendon Hv BT, Clarendon Lt BT, Colbert,
Colonna MT, Comfortaa, Comic Sans NS, Commons, ConcorsoItaliana BTN, ConcorsoItaliana BTN Wide, Consoles, Constantia, Coolvetica, Cooper Black,
Copperplate Gothic Bold, Copperplate Gothic Light, Corbel, Corbel Light, Corporate, Courier New, Cracked Johnnie, Creepygirl, Curlz MT, Cut Me Out,
Distant Galaxy, Dominico, Dubai, Dubai Light, Dubai Medium, Ebrima, Edwardian Script ITC, ELEGANCE, Elephant, ELISS,
Emmett, English157 BT, Engravers MT, Enliven, Eras Bold ITC, Eras Demi ITC, Eras Light ITC, Eras Medium ITC, Ethnocentric, EXCESS,
Felix Titling, Fingerprop, Flubber, FontAwesome, Footlight MT Light, Forte, Frankfurter Venetian TT, Franklin Gothic Book, Franklin Gothic Demi, Franklin Gothic Demi Cond,
Franklin Gothic Heavy, Franklin Gothic Medium, Franklin Gothic Medium Cond, Freehand001 BT, Freehand075 BT, Freestyle Script, French Script MT, Gabriola, Gadget, Galeforce BTN,
Garamond, Gauge, Gauge Heavy, Gazzarini, Gen M01, Gen M01 Light, GENUINE, Georgia, Georgia Pro, Georgia Pro Black,
Georgia Pro Cond, Georgia Pro Cond Black, Georgia Pro Cond Light, Georgia Pro Cond SemiBold, Georgia Pro Light, Georgia Pro SemiBold, Geotype TT, Germanica, Gidole, Gigi,
Gill Sans MT, Gill Sans MT Condensed, Gill Sans MT Ext Condensed Bold, Gill Sans Nova, Gill Sans Nova Cond, Gill Sans Nova Cond Lt, Gill Sans Nova Cond Ultra Bold, Gill Sans Nova Cond Xbd, Gill Sans Nova Light, Gill Sans Nova
Gill Sans Ultra Bold, Gill Sans Ultra Bold Condensed, Glockenspiel, Gloucester MT Extra Condensed, Good Times, Goudy Old Style, Goudy Stout, Grand Hotel, Greek Diner Inline TT, GrilledCheese BTN Cn,
GrilledCheese BTN Toasted, GrilledCheese BTN Wide Blk, Haettenschweiler, Hand Me Down 5 (BRK), Hansen, Harlow Solid Italic, Harrington, Harvest, Harvestital, Haxton Logos TT,
Heavy Hoop, HELTERSKELTER, HERMAN, High Tower Text, Hollywood Hills, Hollands HDLI Assets, Hombre, Hot Mustard BTN, Hot Mustard BTN Poster, HP Simplified,
HP Simplified Jpan, HP Simplified Jpan Light, HP Simplified Light, HP Simplified ME, HP Simplified ME Light, Huxley Titling, IMG Baby, IMG Extreme, IMG Love, IMG Seasons,
IMG Symbols, IMG Travel, Impact, Imprint MT Shadow, Incised901 Bdcn BT, Incised901 Ct BT, Incised901 Nd BT, Incised901 Ndit BT, Induction, Informal Roman,
Ink Free, ISABELLE, Italianate, Javanese Text, Joan, Jokerman, Josefina Slab, Julice ITC, Just Me Again Down Here, JUSTICE,
Kanalisprung, Kristen ITC, Kunstler Script, Lato, Lato Black, Lato Hairline, Lato Light, Leelawadee, Leelawadee UI, Leelawadee UI Semilight,
LetterOmatic, Limousine, LittleLeondFonterooy, Lucida Bright, Lucida Calligraphy, Lucida Console, Lucida Fax, Lucida Handwriting, Lucida Sans, Lucida Sans Typewriter,
Lucida Sans Unicode, Nael, Nagasaki, Palandra GD, Palgun Gothic, Palgun Gothic Semilight, PANDELA, Manoly, Harlette, Marquiesette BTN,
Marquiesette BTN Light, Marquiesette BTN Lined, Martina, Material Icons, MATTEROFFACT, Matura MT Script Capitals, MeiodBold, Miama, Microdot, Microsoft Himalaya,
Microsoft Jhenghei, Microsoft Jhenghei Light, Microsoft Jhenghei UI, Microsoft Jhenghei UI Light, Microsoft New Tai Lue, Microsoft PhagsPa, Microsoft Sans Serif, Microsoft Tai Le, Microsoft Uighur, Microsoft Yaei,
Microsoft Yahei Light, Microsoft Yahei UI, Microsoft Yahei UI Light, Microsoft Yi Baiti, Minerva, Mingliu-ExtB, Mingliu-HKSCS-ExtB, Mistral, Modern No. 20, Mongolian Baiti,
Monospace, Monotype Corsiva, Moonbeam, NS Gothic, NS Outlook, NS PGothic, NS Reference Sans Serif, NS Reference Specialty, NS UI Gothic, NT Extra,
Nova Cycle, Niagara Engraved, Niagara Solid, Nightclub BTN, Nightclub BTN Cn, Nightclub BTN UltraCn, Nirmala UI, Nirmala UI Semilight, Noale, Notram,
Nova Cut, Nova Flat, Nova Oval, Nova Round, Nova Script, Nova Slim, Nova Square, November, NSIaSun, Nunito,
OCR A Extended, Old English Text MT, Onyx, Open Sans, Open Sans Condensed, Open Sans Condensed Light, Open Sans Extrabold, Open Sans Light, Open Sans SemiBold, OPENCLASSIC,
Optineavy, Ostrich Sans, Ostrich Sans Dashed, Ostrich Sans Rounded, Oswald, Oswald Regular, Oswald Stencil, Pacifico, Palace Script MT, Palatino Linotype,
Papyrus, Parchment, Parry Hotter, Penultimate, PenultimateLight, PenultimateLightItal, Permanent Marker, Perpetua, Perpetua Titling MT, Persia BT,
PhrastructMedium, Pirate, Playbill, PHINGLIU-ExtB, Poor Richard, PR Celtic Narrow, PRETEXT, PrimaSans BT, Pristina, PUPPYLIKE,
Quicksand, Quicksand Dash, Quivertal, ROUNDRUN, Sage Italic, Railway, Ravie, Realizme, Rockwell, Rockwell Condensed,
Rockwell Extra Bold, Rockwell Nova, Rockwell Nova Cond, Rockwell Nova Cond Light, Rockwell Nova Extra Bold, Rockwell Nova Light, Roland, Rondalo, RowdyHeavy, Rumburak,
Rundursiv, Russel Write TT, Salina, SansSerif, Script MT Bold, Segoe MDL Assets, Segoe Print, Segoe Script, Segoe UI, Segoe UI Black,
Segoe UI Emoji, Segoe UI Historic, Segoe UI Light, Segoe UI Semibold, Segoe UI Symbol, Selena, Serif, SF Movie Poster, SHELWAN,
Showcard Gothic, SIMSun, SIMSun-ExtB, Sitka Banner, Sitka Display, Sitka Heading, Sitka Small, Sitka Subheading, Sitka Text, Skinny,
Snap ITC, Sneakerhead BTN, Sneakerhead BTN Condensed, Sneakerhead BTN Outline, Sneakerhead BTN Shadow, Snowdrift, Sonic Xbd BT, SonicCutThru Hv BT, Space Bd BT, Splash,
Starliner BTN, Stencil, Stephen, Steppes TT, Swiss021 BT, Sylfaen, Symbol, Tahome, Tangerine, Tarzan,
TeamViewer15, Tempus Sans ITC, Terminator Two, Times New Roman, Toledo, Trebuchet MS, TRENDDY, Tw Cen MT, Tw Cen MT Condensed, Tw Cen MT Condensed Extra Bold,
VADounded BT, Valken, Verdana, Verdana Pro, Verdana Pro Black, Verdana Pro Cond, Verdana Pro Cond Black, Verdana Pro Cond Light, Verdana Pro Cond SemiBold, Verdana Pro Light,
Verdana Pro SemiBold, Viner Hand ITC, Vivavid, Vivian, Vladimir Script, Vollkorn Bold, Vollkorn Bold Italic, Vollkorn Italic, Vollkorn Regular, Waverly,
```

A continuación vamos a realizar un programa que nos pregunte por una determinada fuente y que nos diga si está instalada o no.

```
package graficos;

import java.awt.GraphicsEnvironment;

import javax.swing.JOptionPane;

public class PreubaFuentes {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String fuente=JOptionPane.showInputDialog("Introduce fuente");

        boolean estalafuente=false;
    }
}
```

```

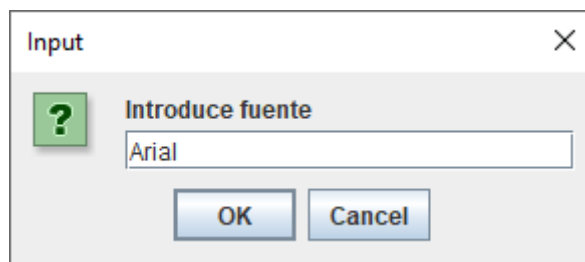
String [] nombresDeFuentes=
GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFami
lyNames();

for(String nombredelaFuente: nombresDeFuentes) {
    if(nombredelaFuente.equals(fuente)) {
        estalaFuente=true;
    }
}

if(estalaFuente) {
    System.out.println("Fuente instalada.");
}
else {
    System.out.println("Fuente no instalada.");
}
}
}

```

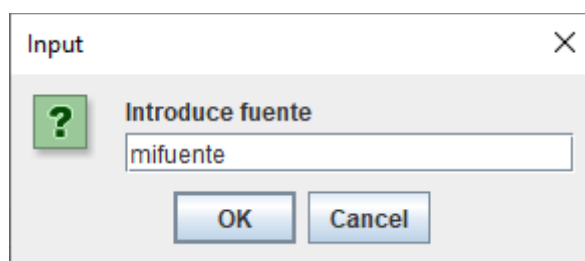
Si ejecutamos este será el resultado:



Pulsamos Ok.

`Fuente instalada.`

Ejecutamos de nuevo.



`Fuente no instalada.`


```

1 package graficos;
2 import java.awt.*;
3 import javax.swing.*;
4 public class TrabajandoConFuentes {
5
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8         MarcoConFuentes mimarco=new MarcoConFuentes();
9         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        mimarco.setVisible(true);
11    }
12
13 }
14
15 class MarcoConFuentes extends JFrame{
16     public MarcoConFuentes() {
17         setTitle("Prueba con fuentes");
18         setSize(400,400);
19         LaminaConFuentes milamina=new LaminaConFuentes();
20         add(milamina);
21     }
22 }
23 class LaminaConFuentes extends JPanel{
24     public void paintComponent(Graphics g) {
25         super.paintComponent(g);
26         Graphics2D g2=(Graphics2D)g;
27         Font mifunete=new Font("Arial", Font.BOLD, 26);
28         g2.setFont(mifunete);
29         g2.drawString("Juan",100,100 );
30     }
31 }

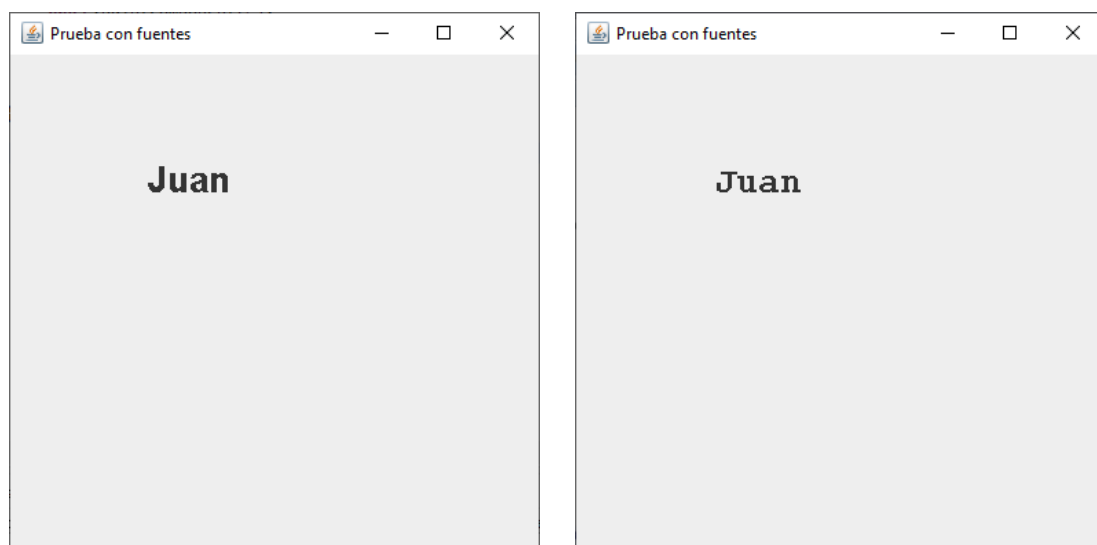
```

Línea 27 definimos un objeto llamado mifunete de tipo Font a los que le pasamos el tipo de funete, estilo y tamaño.

Línea 28 utilizando un objeto de tipo Graphics2D llamado g2 con el método setFont le pasamos los valores de mifunete.

En la línea 29 le decimos que escriba el string "Juan" en las coordenadas 100, 100.

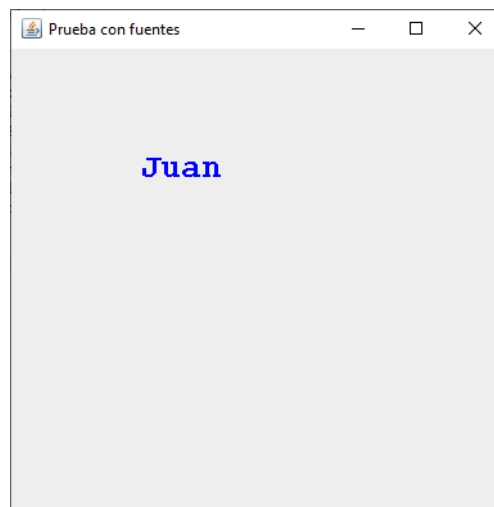
Este será el resultado primero con Arial y luego con Courier.



Vamos a agregar color.

```
23 class LaminaConFuentes extends JPanel{
24     public void paintComponent(Graphics g) {
25         super.paintComponent(g);
26         Graphics2D g2=(Graphics2D)g;
27         Font mifunte=new Font("Courier", Font.BOLD, 26);
28         g2.setFont(mifunte);
29         g2.setColor(Color.BLUE); ←
30         g2.drawString("Juan",100,100 );
31     }
32 }
```

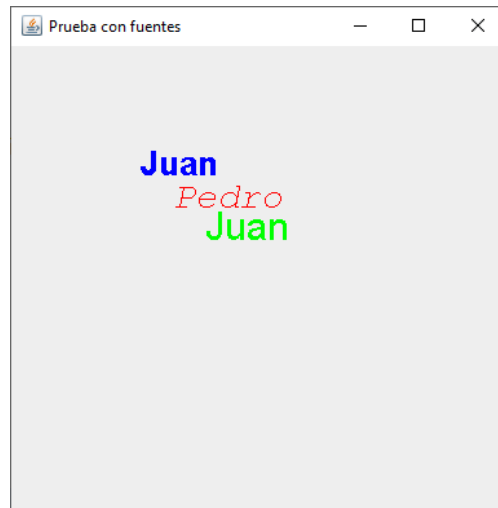
En la clase LaminaConFuente le asignamos el color, este será el resultado:



Vamos a agregar más textos y distintos formatos y posiciones.

```
23 class LaminaConFuentes extends JPanel{
24     public void paintComponent(Graphics g) {
25         super.paintComponent(g);
26         Graphics2D g2=(Graphics2D)g;
27
28         Font mifunte=new Font("Arial", Font.BOLD, 26);
29         g2.setFont(mifunte);
30         g2.setColor(Color.BLUE);
31         g2.drawString("Juan",100,100 );
32
33         g2.setFont(new Font("Courier", Font.ITALIC, 28));
34         g2.setColor(new Color(255,0,0));
35         g2.drawString("Pedro",125,125 );
36
37         g2.setFont(new Font("Gothic", Font.PLAIN, 30));
38         g2.setColor(Color.GREEN);
39         g2.drawString("Juan",150,150 );
40     }
41 }
```

Este será el resultado:



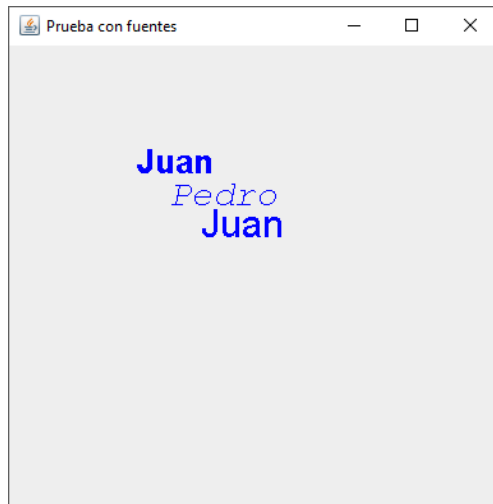
Para establecer el mismo color en una lámina, borraremos todos los colores o los ponemos como comentario.

```
23 class LaminaConFuentes extends JPanel{
24     public void paintComponent(Graphics g) {
25         super.paintComponent(g);
26         Graphics2D g2=(Graphics2D)g;
27
28         Font mifuente=new Font("Arial", Font.BOLD, 26);
29         g2.setFont(mifuente);
30         //g2.setColor(Color.BLUE); ←
31         g2.drawString("Juan",100,100 );
32
33         g2.setFont(new Font("Courier", Font.ITALIC, 28));
34         //g2.setColor(new Color(255,0,0)); ←
35         g2.drawString("Pedro",125,125 );
36
37         g2.setFont(new Font("Gothic", Font.PLAIN, 30));
38         //g2.setColor(Color.GREEN); ←
39         g2.drawString("Juan",150,150 );
40     }
41 }
```

En la clase MarcoConFuente le decimos el color para todas las fuentes de esta lámina.

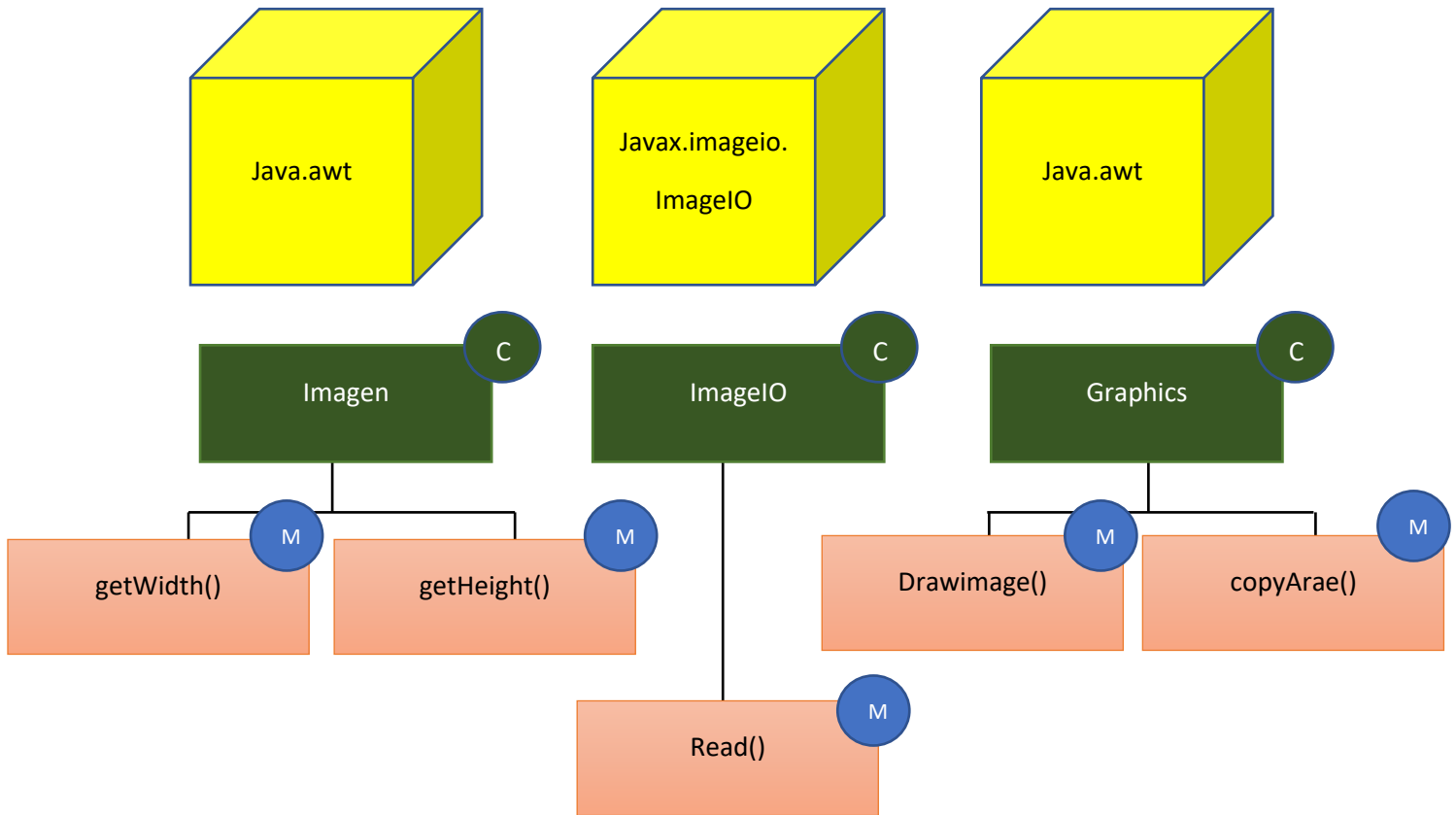
```
15 class MarcoConFuentes extends JFrame{
16     public MarcoConFuentes() {
17         setTitle("Prueba con fuentes");
18         setSize(400,400);
19         LaminaConFuentes milamina=new LaminaConFuentes();
20         add(milamina);
21         milamina.setForeground(Color.BLUE); ←
22     }
23 }
```

Este será el resultado:



Aplicaciones gráficas. Swing IX. Incluyendo imágenes. (Vídeo 63)

Clases y métodos necesarios.



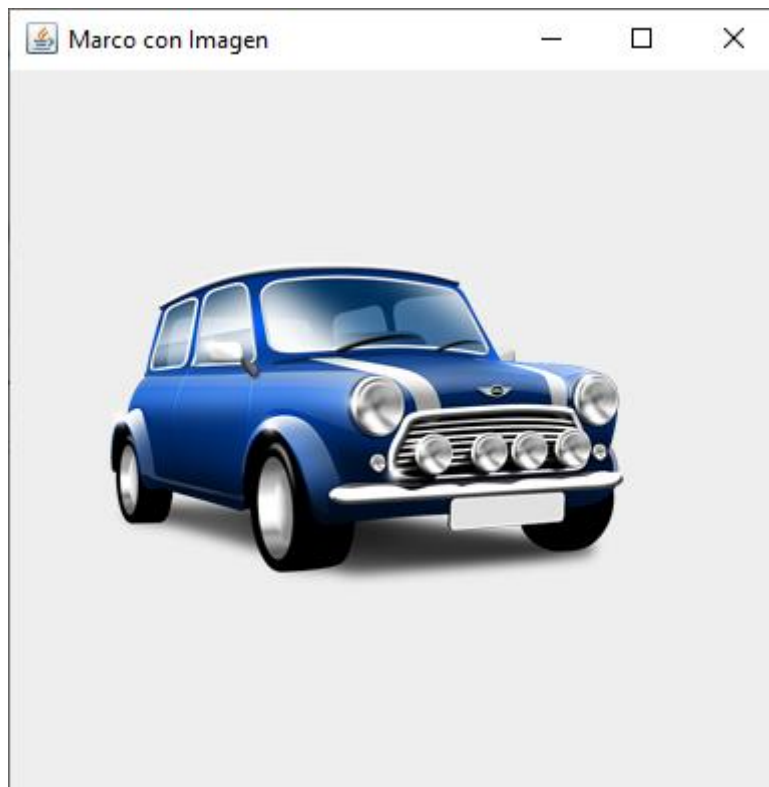
```
1 package graficos;
2 import java.awt.*;
3 import javax.swing.*;
4 import javax.imageio.*;
5 import java.io.*;
6
7 public class PruebaImágenes {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         MarcoImagen mimarco=new MarcoImagen();
12         mimarco.setVisible(true);
13         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15
16 }
17
18 class MarcoImagen extends JFrame{
19     public MarcoImagen() {
20         setTitle("Marco con Imagen");
21         setBounds(100,100,400,400);
22         LaminaConImagen milamina=new LaminaConImagen();
23         add(milamina);
24     }
25 }
26
```

```

27 class LaminaConImagen extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         File miimagen=new File("src/graficos/coche.png");
31         try {
32             imagen=ImageIO.read(miimagen);
33         }catch(IOException e) {
34             System.out.println("Imagen no encontrada.");
35         }
36         g.drawImage(imagen, 50,50,null);
37     }
38     private Image imagen;
39 }

```

Este será el resultado:



En este ejemplo hemos guardado la imagen en la carpeta donde se guardan los proyectos "src/graficos/coche.png".

```

27 class LaminaConImagen extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         //File miimagen=new File("src/graficos/coche.png");
31         try {
32             imagen=ImageIO.read(new File("src/graficos/coche.png"));
33         }catch(IOException e) {
34             System.out.println("Imagen no encontrada.");
35         }
36         g.drawImage(imagen, 50,50,null);
37     }
38     private Image imagen;
39 }

```

También podemos suprimir la línea 30 y en la línea 31 hacerlo todo en una sola línea.

Imaginaros que en la línea 32 a la hora de poner el nombre o ruta nos equivocamos.

```
27 class LaminaConImagen extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         //File miimagen=new File("src/graficos/coche.png");
31         try {
32             imagen=ImageIO.read(new File("src/graficos/coche1.png"));
33         }catch(IOException e) {
34             System.out.println("Imagen no encontrada.");
35         }
36         g.drawImage(imagen, 50,50,null);
37     }
38     private Image imagen;
39 }
```

Cuando ejecutemos:

```
Imagen no encontrada.
```



Aplicaciones gráficas. Swing X Incluyendo imágenes II (Vídeo 64)

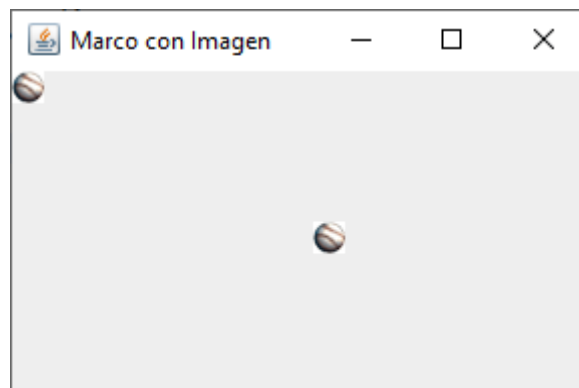
```
1 package graficos;
2 import java.awt.*;
6
7 public class PruebaImagenes {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         MarcoImagen mimarco=new MarcoImagen();
12         mimarco.setVisible(true);
13         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15
16 }
17
18 class MarcoImagen extends JFrame{
19     public MarcoImagen() {
20         setTitle("Marco con Imagen");
21         setBounds(700,300,304,200);
22         LaminaConImagen milamina=new LaminaConImagen();
23         add(milamina);
24     }
25 }
26
27 class LaminaConImagen extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         try {
31             imagen=ImageIO.read(new File("src/graficos/bola.gif"));
32         }catch(IOException e) {
33             System.out.println("Imagen no encontrada.");
34         }
35         g.drawImage(imagen, 0,0,null);
36         g.copyArea(0,0,16,16,150,75);
37     }
38     private Image imagen;
39 }
```

En la línea 31 hemos cambiado por una imagen llamada bola.gif de unas dimensiones de 16 x 16.

En la línea 35 le decimos que la dibuje en la coordenada 00.

En la línea 36 que en la coordenada 0,0 con un ancho y alto de 16 pixeles los copie en la posición 150,75.

Este será el resultado:



Ahora te planteo que rellenes todo el marco de bolas, el código lo tienes en la siguiente página.


```

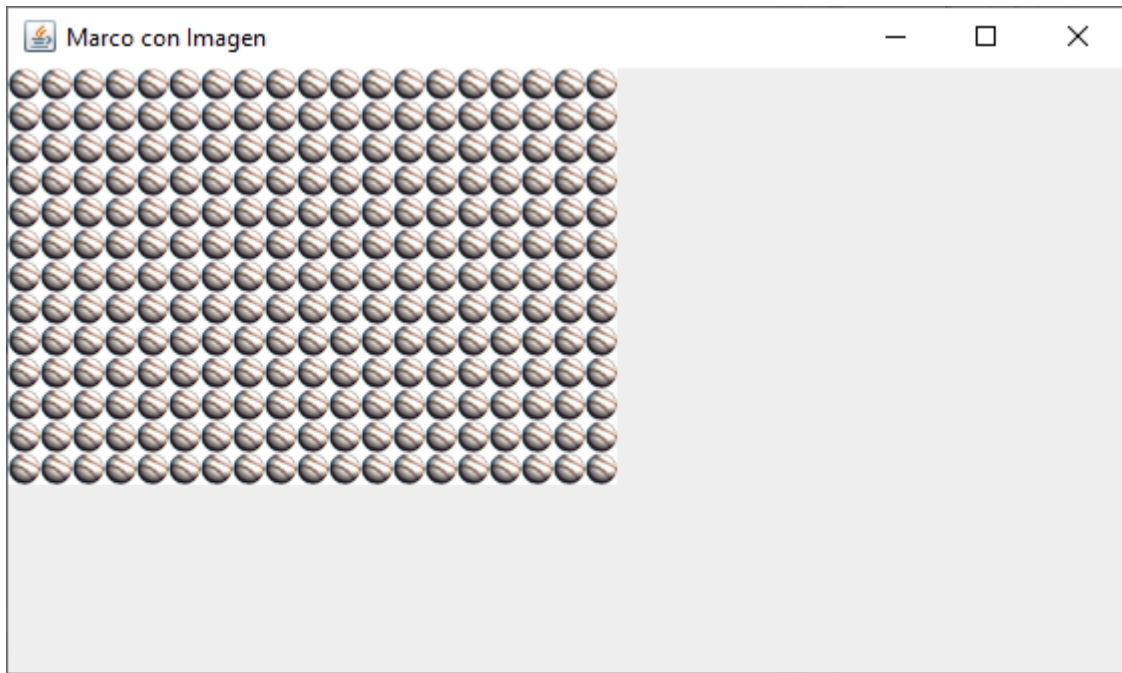
1 package graficos;
2 import java.awt.*;
6
7 public class PruebaImagenes {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         MarcoImagen mimarco=new MarcoImagen();
12         mimarco.setVisible(true);
13         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15
16 }
17
18 class MarcoImagen extends JFrame{
19     public MarcoImagen() {
20         setTitle("Marco con Imagen");
21         setBounds(700,300,304,200);
22         LaminaConImagen milamina=new LaminaConImagen();
23         add(milamina);
24     }
25 }
26
27 class LaminaConImagen extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         try {
31             imagen=ImageIO.read(new File("src/graficos/bola.gif"));
32         }catch(IOException e) {
33             System.out.println("Imagen no encontrada.");
34         }
35         g.drawImage(imagen, 0,0,null);
36         for(int i=0;i<300; i=i+16) {
37             for(int j=0;j<200; j=j+16) {
38                 g.copyArea(0,0,16,16,i,j);
39             }
40         }
41     }
42     private Image imagen;
43 }

```

Este será el resultado:



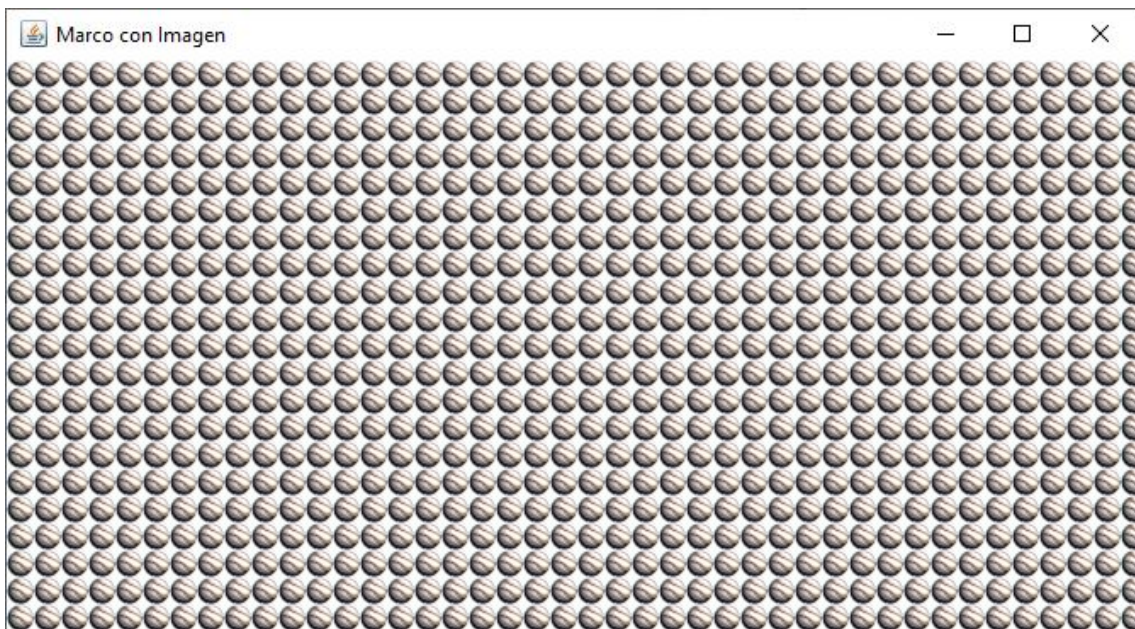
Pero si redimensionamos.



Para evitar que al redimensionar no se vean más bolas vamos a modificar el for.

```
for(int i=0;i<1300; i=i+16) {  
    for(int j=0;j<1200; j=j+16) {  
        g.copyArea(0,0,16,16,i,j);  
    }  
}
```

Este será el resultado:



```

27 class LaminaConImagen extends JPanel{
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         try {
31             imagen=ImageIO.read(new File("src/graficos/bola.gif"));
32         }catch(IOException e) {
33             System.out.println("Imagen no encontrada.");
34         }
35
36         int anchuraImagen=imagen.getWidth(this);
37         int alturaImagen=imagen.getHeight(this);
38
39         g.drawImage(imagen, 0,0,null);
40         for(int i=0;i<300; i++) {
41             for(int j=0;j<200; j++) {
42                 g.copyArea(0,0,anchuraImagen,alturaImagen,anchuraImagen*i,alturaImagen*j);
43             }
44         }
45     }
46     private Image imagen;
47 }

```

Con las líneas 36 y 37 podremos ver las dimensiones de la imagen que cargamos en la línea 31.

En el ciclo for sustituimos por las variables anchuraImagen y alturaImagen.

```

1 package graficos;
2 import java.awt.*;
6
7 public class PruebaImagenes {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         MarcoImagen mimarco=new MarcoImagen();
12         mimarco.setVisible(true);
13         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     }
15 }
16
17 class MarcoImagen extends JFrame{
18     public MarcoImagen() {
19         setTitle("Marco con Imagen");
20         setBounds(700,300,304,200);
21         LaminaConImagen milamina=new LaminaConImagen();
22         add(milamina);
23     }
24 }
25
26 class LaminaConImagen extends JPanel{
27
28     public LaminaConImagen() {
29         try {
30             imagen=ImageIO.read(new File("src/graficos/bola.gif"));
31         }catch(IOException e) {
32             System.out.println("Imagen no encontrada.");
33         }
34     }
35 }

```

```

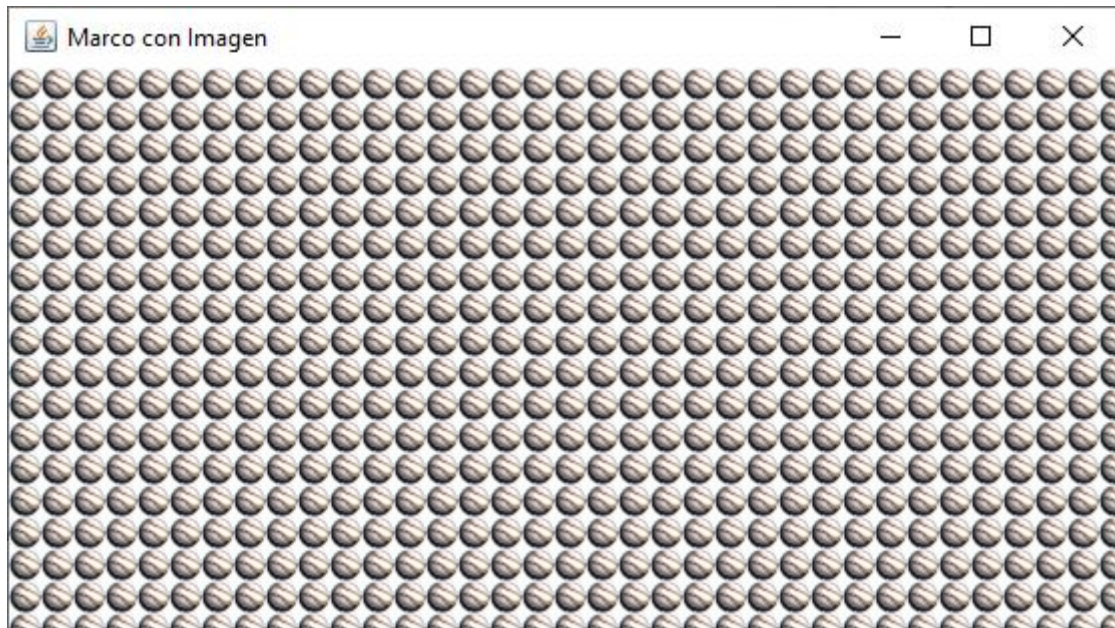
36 public void paintComponent(Graphics g) {
37     super.paintComponent(g);
38
39
40     int anchuraImagen=imagen.getWidth(this);
41     int alturaImagen=imagen.getHeight(this);
42
43     g.drawImage(imagen, 0,0,null);
44     for(int i=0;i<300; i++) {
45         for(int j=0;j<200; j++) {
46             g.copyArea(0,0,anchuraImagen,alturaImagen,anchuraImagen*i,alturaImagen*j);
47         }
48     }
49 }
50 private Image imagen;
51 }

```

Observarás que en la línea 28 hemos creado un constructor, para agregar el código que está en el recuadro, esto también funcionará correctamente.

Esto código estaba en la línea después de `super.paintComponent(g)`, que hemos borrado, para que el código no se duplique.

El programa funciona exactamente igual.



```

g.drawImage(imagen, 0,0,null);
for(int i=0;i<300; i++) {
    for(int j=0;j<200; j++) {
        g.copyArea(0,0,anchuraImagen,alturaImagen,anchuraImagen*i,alturaImagen*j);
    }
}

```

En el ciclo for al principio `i=0` y `j=0` esto significa que la primera figura la copiaría en las coordenadas que esta la figura a copiar, no lo vemos porque copia una figura encima de la otra, pero si queremos evitar este fallo modificaríamos el código de la siguiente forma.

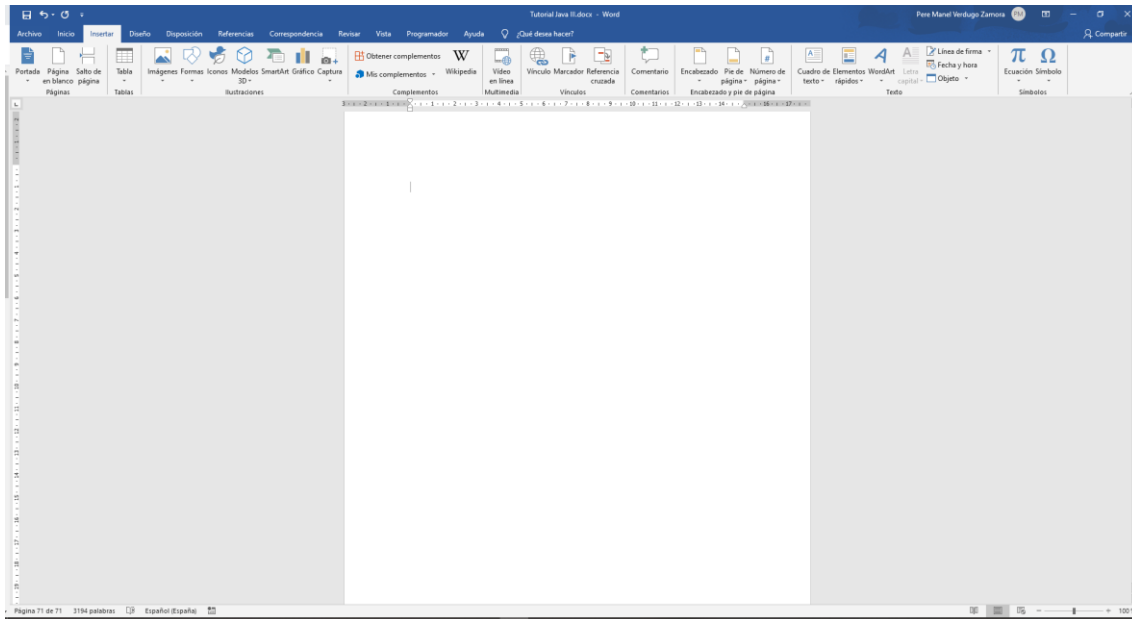
```
43     g.drawImage(imagen, 0,0,null);
44     for(int i=0;i<300; i++) {
45         for(int j=0;j<200; j++) {
46             if(i+j>0) {
47                 g.copyArea(0,0,anchuraImagen,alturaImagen,anchuraImagen*i,alturaImagen*j);
48             }
49         }
50     }
```

En la línea 47 le estamos diciendo que copyArea cuando i+j sean mayor de 0 de este modo en la coordenada 0,0 no copiará la figura.



Eventos 1. (Vídeo 65)

Como ejemplo para explicar los eventos vamos a ejecutar Word.



Estamos viendo la interface de Word pero si hubieran eventos no se podría interactuar con él.

Por ejemplo para acceder a la cinta de opciones haremos clic en cada una de las pestañas a la que podemos acceder.

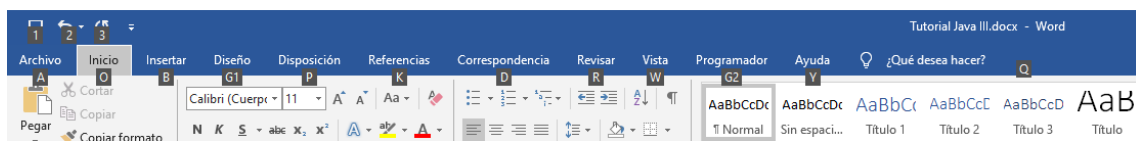
El evento se llama al hacer clic, hemos cambiado de cinta.

El evento clic puede realizar varias acciones diferentes como desplegar menú.

Otro ejemplo es cuando pulsamos F1 genera un evento al pulsar la tecla F1 en este caso ejecuta la ayuda.

Evento genera acción.

Si pulsamos la tecla Alt genera un evento llamado atajo del teclado.



Se muestran una letra para seleccionar las pestañas de la cinta.

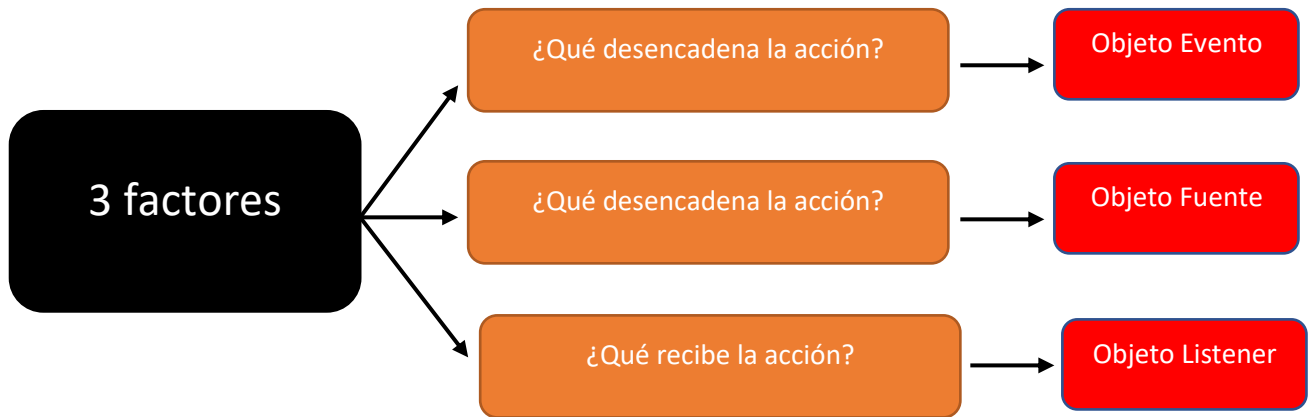
El evento "Al pulsar la tecla Alt". La acción ejecutar atajos del teclado.

También hay eventos de ventana al cambiar el tamaño de la ventana, al abrir una ventana, al cerrar una ventana, etc.

Al evento se le denomina el desencadenante de la acción.

Eventos: fuentes y oyentes

Definición: Desencadenantes de la acción.



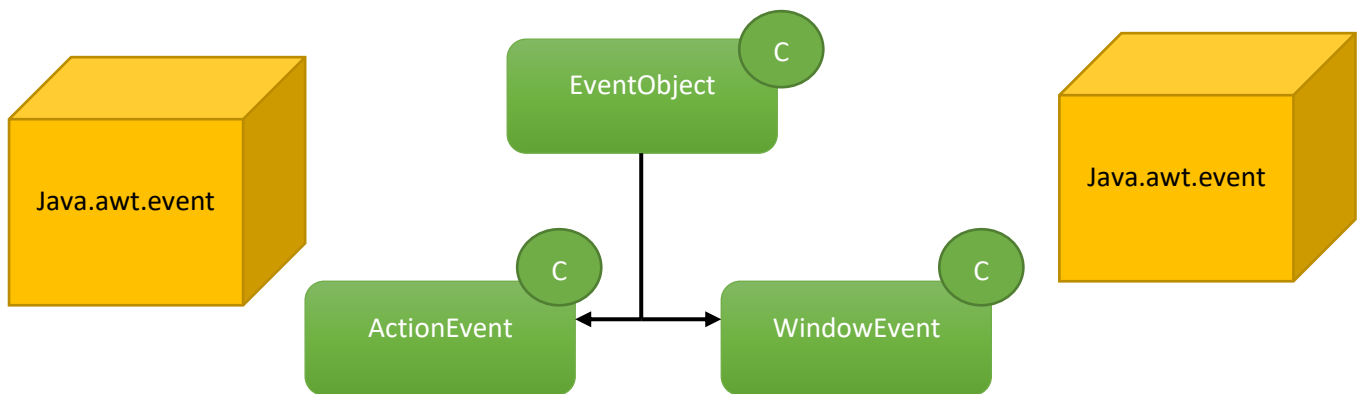
Como ejemplo hacemos clic en la ficha Correspondencia de Word.

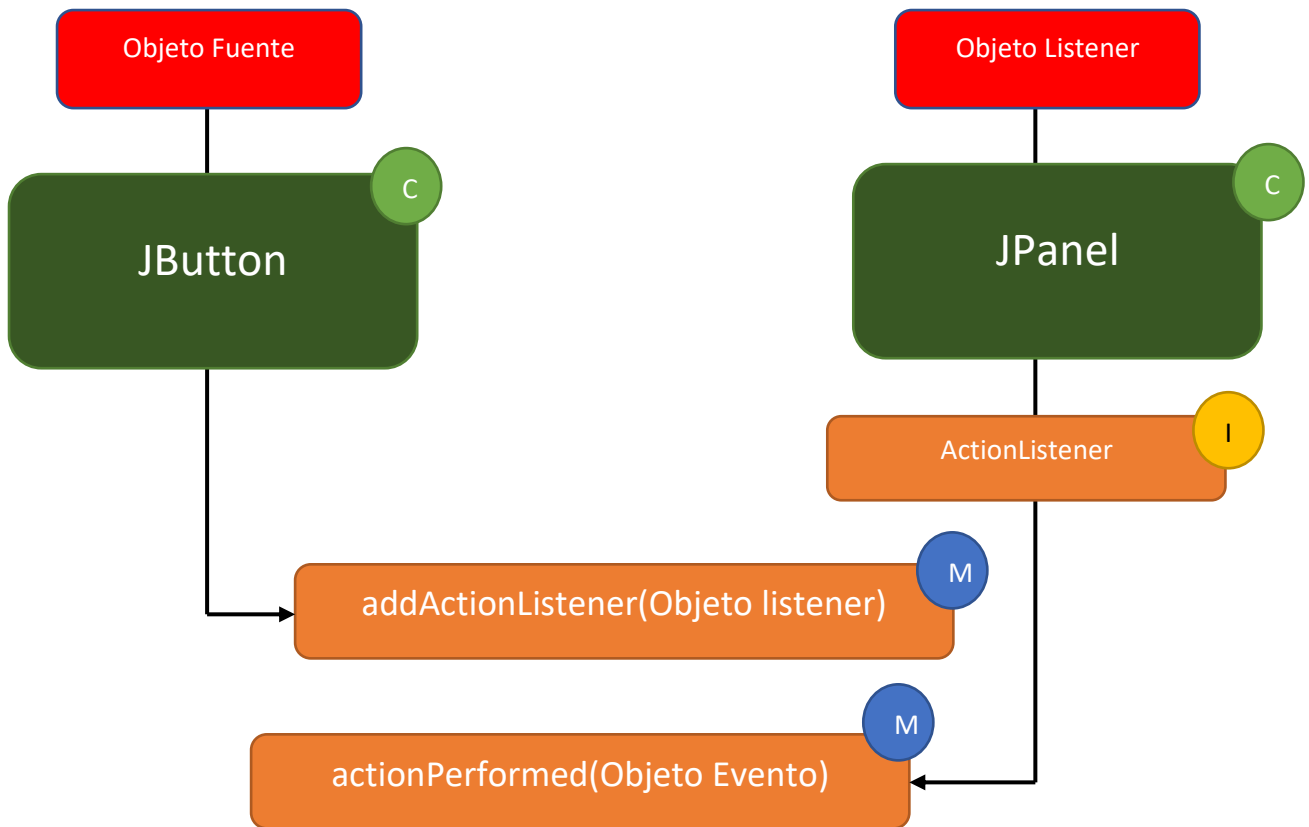
Al hacer clic,

La acción en la ficha correspondencia.

La acción la recibe la cinta de opciones (oyente), Objeto Listener.

Objeto Evento





Vamos a crear una nueva clase llamada PruebaEventos.

Vamos a escribir el siguiente código:

```

1 package graficos;
2 import java.awt.*;
3 import javax.swing.*;
4 public class PruebaEventos {
5
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8         MarcoBotones mimarco=new MarcoBotones();
9         mimarco.setVisible(true);
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12
13 }
14
15 class MarcoBotones extends JFrame{
16     public MarcoBotones() {
17         setTitle("Botones y Eventos");
18         setBounds(700,300,500,300);
19         LaminaBotones milamina=new LaminaBotones();
20         add(milamina);
21     }
22 }
23
24 class LaminaBotones extends JPanel{
25
26 }
  
```

En el siguiente capítulo partiendo de este código seguiremos con el proyecto.



Eventos II. (VÍdeo 66)

```
2 import java.awt.*;
3 import javax.swing.*;
4 import java.awt.event.*;
5
6 public class PruebaEventos {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoBotones mimarco=new MarcoBotones();
11        mimarco.setVisible(true);
12        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13    }
14
15 }
16
17 class MarcoBotones extends JFrame{
18     public MarcoBotones() {
19         setTitle("Botones y Eventos");
20         setBounds(700,300,500,300);
21         LaminaBotones milamina=new LaminaBotones();
22         add(milamina);
23     }
24 }
25
26 class LaminaBotones extends JPanel implements ActionListener{
27     JButton botonAzul=new JButton("Azul");
28     public LaminaBotones() {
29         add(botonAzul);
30         botonAzul.addActionListener(this);
31     }
32
33     public void actionPerformed(ActionEvent e) {
34         setBackground(Color.blue);
35     }
36 }
```

Este será el resultado:



Vamos a agregar dos botones más para el color Amarillo y Rojo.

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class PruebaEventos {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MarcoBotones mimarco=new MarcoBotones();
        mimarco.setVisible(true);
        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

class MarcoBotones extends JFrame{
    public MarcoBotones() {
        setTitle("Botones y Eventos");
        setBounds(700,300,500,300);
        LaminaBotones milamina=new LaminaBotones();
        add(milamina);
    }
}

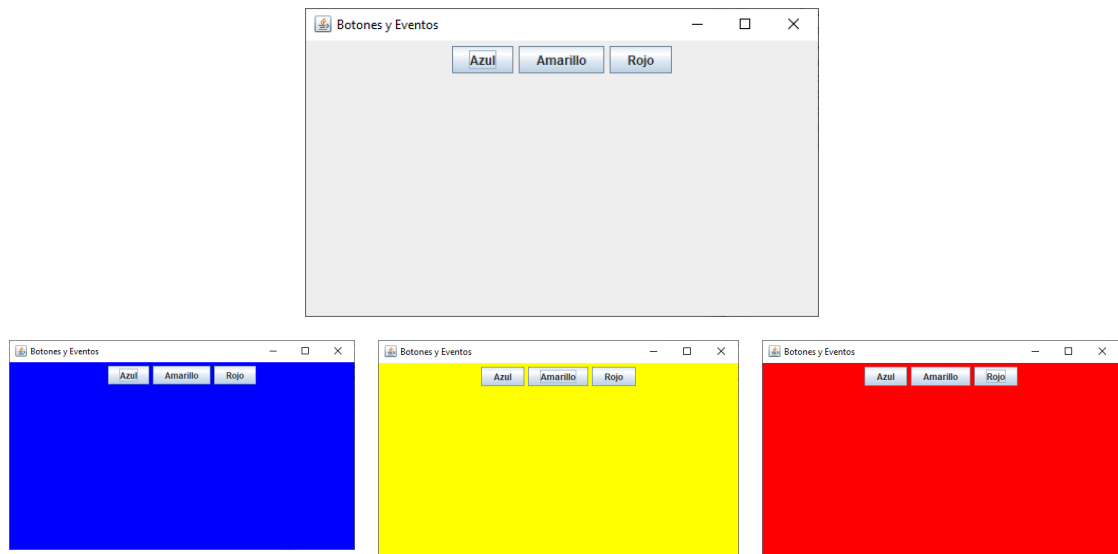
class LaminaBotones extends JPanel implements ActionListener{
    JButton botonAzul=new JButton("Azul");
    JButton botonAmarillo=new JButton("Amarillo");
    JButton botonRojo=new JButton("Rojo");
    public LaminaBotones() {
        add(botonAzul);
        add(botonAmarillo);
        add(botonRojo);
        botonAzul.addActionListener(this);
        botonAmarillo.addActionListener(this);
        botonRojo.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {

        Object botonPulsado=e.getSource();

        if(botonPulsado==botonAzul) {
            setBackground(Color.blue);
        }
        else if(botonPulsado==botonAmarillo) {
            setBackground(Color.yellow);
        }
        else {
            setBackground(Color.red);
        }
    }
}
```

Este será el resultado:



Eventos III (VÍdeo 67)

```
package graficos;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class PruebaEventos {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MarcoBotones mimarco=new MarcoBotones();
        mimarco.setVisible(true);
        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

class MarcoBotones extends JFrame{
    public MarcoBotones() {
        setTitle("Botones y Eventos");
        setBounds(700,300,500,300);
        LaminaBotones milamina=new LaminaBotones();
        add(milamina);
    }
}

class LaminaBotones extends JPanel {
    JButton botonAzul=new JButton("Azul");
    JButton botonAmarillo=new JButton("Amarillo");
    JButton botonRojo=new JButton("Rojo");
    public LaminaBotones() {
        add(botonAzul);
        add(botonAmarillo);
        add(botonRojo);
        ColorFondo Amarillo=new ColorFondo(Color.yellow);
        ColorFondo Azul=new ColorFondo(Color.blue);
        ColorFondo Rojo=new ColorFondo(Color.red);
        botonAzul.addActionListener(Azul);
        botonAmarillo.addActionListener(Amarillo);
        botonRojo.addActionListener(Rojo);
    }
}

private class ColorFondo implements ActionListener{
    public ColorFondo(Color c) {
        colorDeFondo=c;
    }
    public void actionPerformed(ActionEvent e) {
        setBackground(colorDeFondo);
    }
    private Color colorDeFondo;
}
}
```

Clase interna

El resultado es el mismo que en el anterior ejemplo.

Ahor intenta poner el botón verde.

```

26 class LaminaBotones extends JPanel {
27     JButton botonAzul=new JButton("Azul");
28     JButton botonAmarillo=new JButton("Amarillo");
29     JButton botonRojo=new JButton("Rojo");
30     JButton botonVerde=new JButton("Verde");
31     public LaminaBotones() {
32         add(botonAzul);
33         add(botonAmarillo);
34         add(botonRojo);
35         add(botonVerde);
36         ColorFondo Amarillo=new ColorFondo(Color.yellow);
37         ColorFondo Azul=new ColorFondo(Color.blue);
38         ColorFondo Rojo=new ColorFondo(Color.red);
39         ColorFondo Verde=new ColorFondo(Color.green);
40         botonAzul.addActionListener(Azul);
41         botonAmarillo.addActionListener(Amarillo);
42         botonRojo.addActionListener(Rojo);
43         botonVerde.addActionListener(Verde);
44     }
45
46     private class ColorFondo implements ActionListener{
47         public ColorFondo(Color c) {
48             colorDeFondo=c;
49         }
50         public void actionPerformed(ActionEvent e) {
51             setBackground(colorDeFondo);
52         }
53
54         private Color colorDeFondo;
55     }
56 }

```

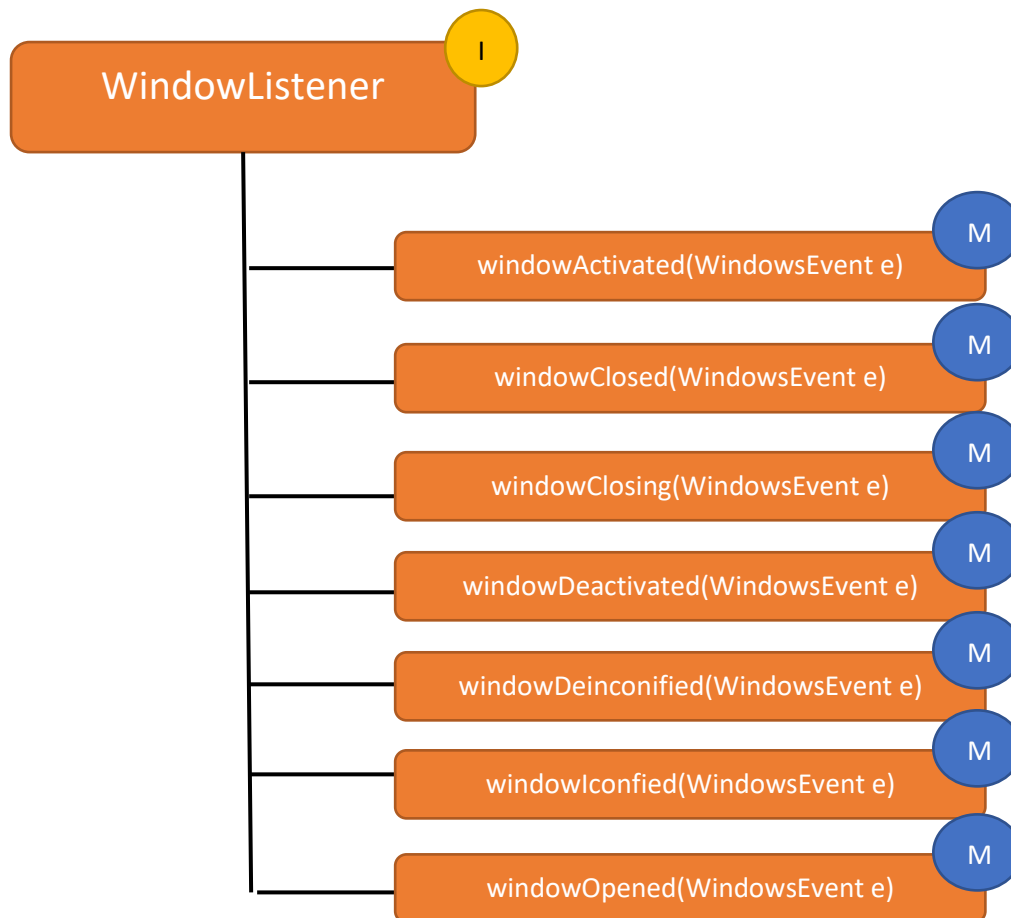
Solo tienes que modificar la clase LaminaBotones, este será el resultado:





Eventos IV. Eventos de ventana I. (VÍdeo 68)

Eventos de ventana:



```
1 package graficos;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class Eventos_Ventana {
5
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8         MarcoVentana mimarco=new MarcoVentana();
9         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10    }
11
12 }
13 class MarcoVentana extends JFrame{
14     public MarcoVentana() {
15         setTitle("Respondiento");
16         setBounds(300,300,500,350);
17         setVisible(true);
18         M_Ventana oyente_ventana=new M_Ventana();
19         addWindowListener(oyente_ventana);
20     }
21 }
```

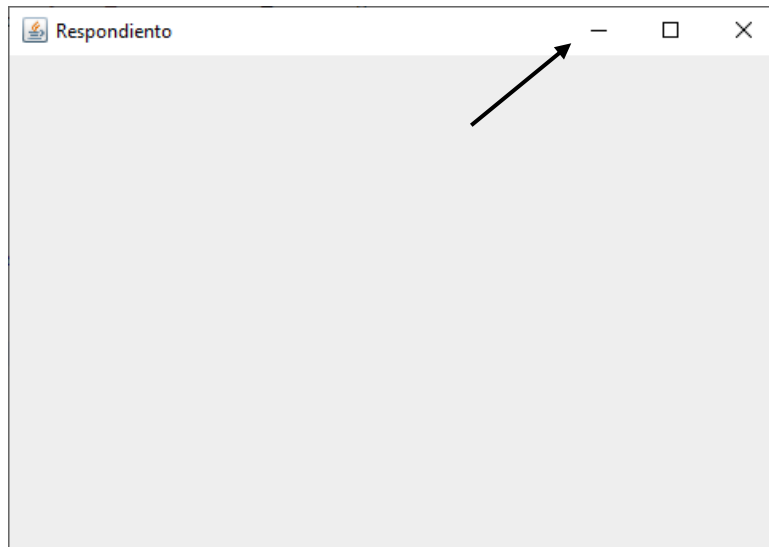


```

22 class M_Ventana implements WindowListener{
△23     public void windowActivated(WindowEvent e) {}
△24     public void windowClosed(WindowEvent e) {}
△25     public void windowClosing(WindowEvent e) {}
△26     public void windowDeactivated(WindowEvent e) {}
△27     public void windowDeiconified(WindowEvent e) {}
△28     public void windowIconified(WindowEvent e) {
29         System.out.println("Ventana minimizada");
30     }
△31     public void windowOpened(WindowEvent e) {}
32 }

```

Si ejecutamos observaremos el siguiente resultado:



Ahora vamos a minimizar y observaremos el resultado en consola.

Ventana minimizada

Vamos completar el resto de eventos:

```

22 class M_Ventana implements WindowListener{
△23     public void windowActivated(WindowEvent e) {
24         System.out.println("Ventana activada");
25     }
△26     public void windowClosed(WindowEvent e) {}
△27     public void windowClosing(WindowEvent e) {
28         System.out.println("Cerrando ventana");
29     }
△30     public void windowDeactivated(WindowEvent e) {
31         System.out.println("Ventana desactivada");
32     }
△33     public void windowDeiconified(WindowEvent e) {
34         System.out.println("Ventana restaurada");
35     }
△36     public void windowIconified(WindowEvent e) {
37         System.out.println("Ventana minimizada");
38     }
△39     public void windowOpened(WindowEvent e) {
40         System.out.println("Ventana abierta");
41     }
42 }

```

Si ejecutamos y minimizamos la ventana, la restauramos, la activamos y la cerramos observaremos en la consola los siguientes mensajes.

```
Ventana activada
Ventana abierta
Ventana minimizada
Ventana desactivada
Ventana restaurada
Ventana activada
Ventana desactivada
Ventana activada
Cerrando ventana
```

Ahora en este proyecto vamos a crear una segunda ventana.

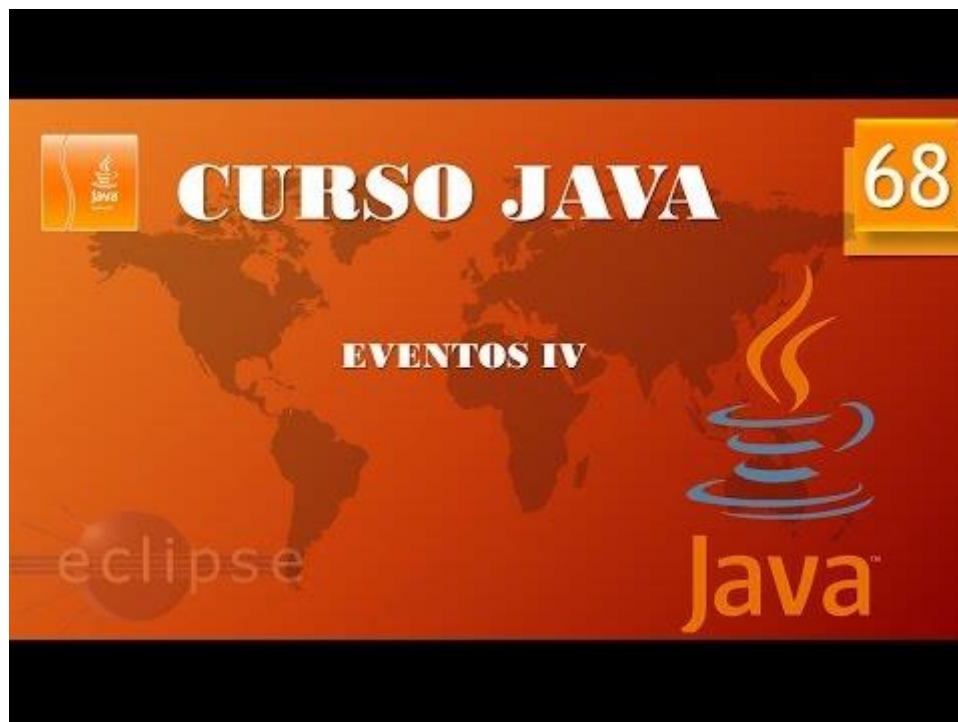
```
1 package graficos;
2 import javax.swing.*;
3 import java.awt.event.*;
4 public class Eventos_Ventana {
5
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8         MarcoVentana mimarco=new MarcoVentana();
9         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        mimarco.setTitle("Ventana 1");
11        mimarco.setBounds(400,300,500,350);
12        MarcoVentana mimarco2=new MarcoVentana();
13        mimarco2.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
14        mimarco2.setTitle("Ventana 2");
15        mimarco2.setBounds(900,300,500,350);
16    }
17
18 }
19 class MarcoVentana extends JFrame{
20     public MarcoVentana() {
21         //setTitle("Respondiente");
22         //setBounds(300,300,500,350);
23         setVisible(true);
24         M_Ventana oyente_ventana=new M_Ventana();
25         addWindowListener(oyente_ventana);
26     }
27 }
28 class M_Ventana implements WindowListener{
29     public void windowActivated(WindowEvent e) {
30         System.out.println("Ventana activada");
31     }
32     public void windowClosed(WindowEvent e) {
33         System.out.println("La ventana ha sido cerrada");
34     }
35     public void windowClosing(WindowEvent e) {
36         System.out.println("Cerrando ventana");
37     }
38     public void windowDeactivated(WindowEvent e) {
39         System.out.println("Ventana desactivada");
40     }
41     public void windowDeiconified(WindowEvent e) {
42         System.out.println("Ventana restaurada");
43     }
44 }
```

```
△44⊖ public void windowIconified(WindowEvent e) {  
45     System.out.println("Ventana minimizada");  
46 }  
△47⊖ public void windowOpened(WindowEvent e) {  
48     System.out.println("Ventana abierta");  
49 }  
50 }
```

En recuadro verde comentamos los valores que tienen que ser diferentes en cada ventana y lo marcado con flechas lo detallamos para cada ventana.

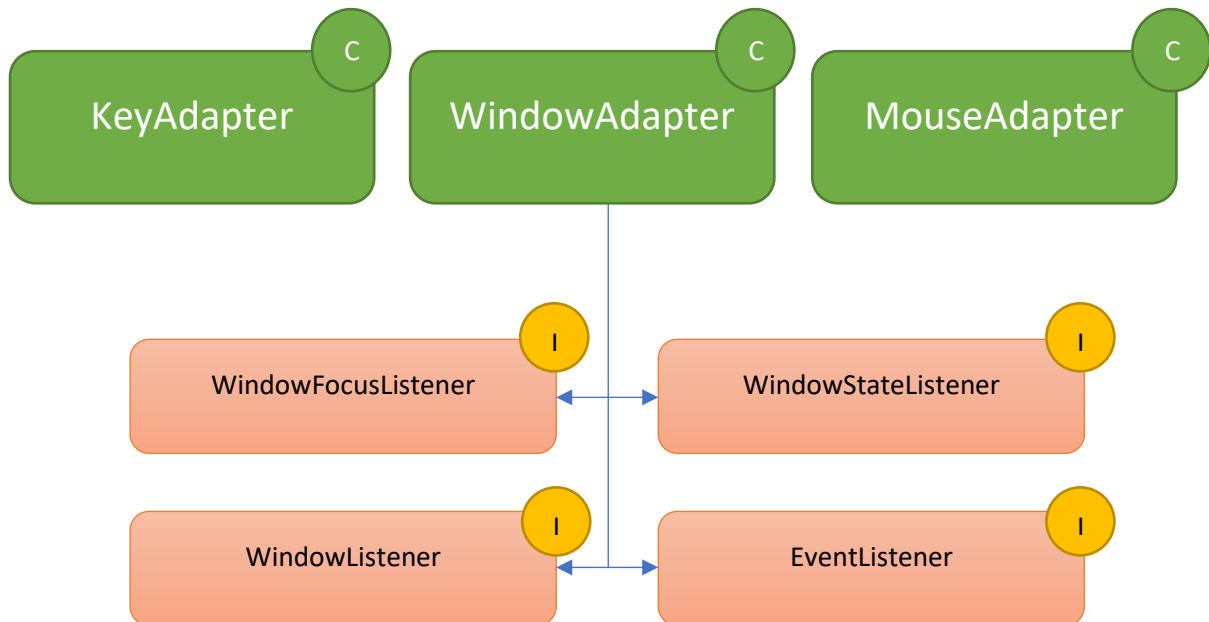
Ahora cuando ejecutes verás que se abren dos ventanas.

Ahora puedes probar con las dos ventanas y observa los mensajes en consola.



Eventos V. Eventos de ventana II. Clases adaptadas. (Vídeo 69)

Clases adaptadoras (o adapter Classes)



```
1 package graficos;
2 import javax.swing.*;
4 public class Eventos_Ventana {
5
6     public static void main(String[] args) {
7         // TODO Auto-generated method stub
8         MarcoVentana mimarco=new MarcoVentana();
9         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        mimarco.setTitle("Ventana 1");
11        mimarco.setBounds(400,300,500,350);
12        MarcoVentana mimarco2=new MarcoVentana();
13        mimarco2.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
14        mimarco2.setTitle("Ventana 2");
15        mimarco2.setBounds(900,300,500,350);
16    }
17
18 }
19 class MarcoVentana extends JFrame{
20     public MarcoVentana() {
21         //setTitle("Respondiento");
22         //setBounds(300,300,500,350);
23         setVisible(true);
24         M_Ventana oyente_ventana=new M_Ventana();
25         addWindowListener(oyente_ventana);
26     }
27 }
```

```

28 class M_Ventana extends WindowAdapter{ ←
29     /*public void windowActivated(WindowEvent e) {
30         System.out.println("Ventana activada");
31     }
32     public void windowClosed(WindowEvent e) {
33         System.out.println("La ventana ha sido cerrada");
34     }
35     public void windowClosing(WindowEvent e) {
36         System.out.println("Cerrando ventana");
37     }
38     public void windowDeactivated(WindowEvent e) {
39         System.out.println("Ventana desactivada");
40     }
41     public void windowDeiconified(WindowEvent e) {
42         System.out.println("Ventana restaurada");
43     }
44     public void windowIconified(WindowEvent e) {
45         System.out.println("Ventana minimizada");
46     }*/
47     public void windowOpened(WindowEvent e) {
48         System.out.println("Ventana abierta");
49     }
50 }

```

En el capítulo anterior en el código de la M_Ventana con “class M_Ventana implements WindowListener{...” estábamos obligados a sobrescribir todos los métodos aunque no los necesitáramos.

Si lo modificamos por extends WindowAdapter, solo tendremos que escribir aquellos métodos que vayamos a utilizar.

Como podréis comprobar hemos comentado seis métodos y no aparecen errores.

Si ejecutamos solo responderá cuando abramos las ventanas.

Ventana abierta
Ventana abierta

```

19 class MarcoVentana extends JFrame{
20     public MarcoVentana() {
21         //setTitle("Respondiente");
22         //setBounds(300,300,500,350);
23         setVisible(true);
24         //M_Ventana oyente_ventana=new M_Ventana();
25         //addWindowListener(oyente_ventana);
26         addWindowListener(new M_Ventana());
27     }
28 }

```

En la clase MarcoVentana aun podemos reducir el código como podréis comprobar hemos comentado las líneas 24 y 25, pero también las podemos eliminar y suplirla por la línea 26.

El programa funcionará exactamente igual.

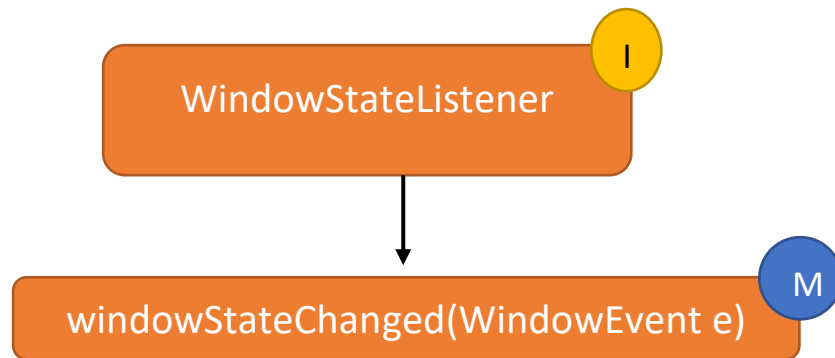
A slide with an orange background and a world map. It features the Java logo in the top left, the text 'CURSO JAVA' in large white letters, and a yellow box with the number '69' in the top right. The main title is 'EVENTOS V' with the subtitle 'Clases Adaptadoras (Adapter Classes)'. The Eclipse logo is in the bottom left, and the Java logo is in the bottom right.

CURSO JAVA 69

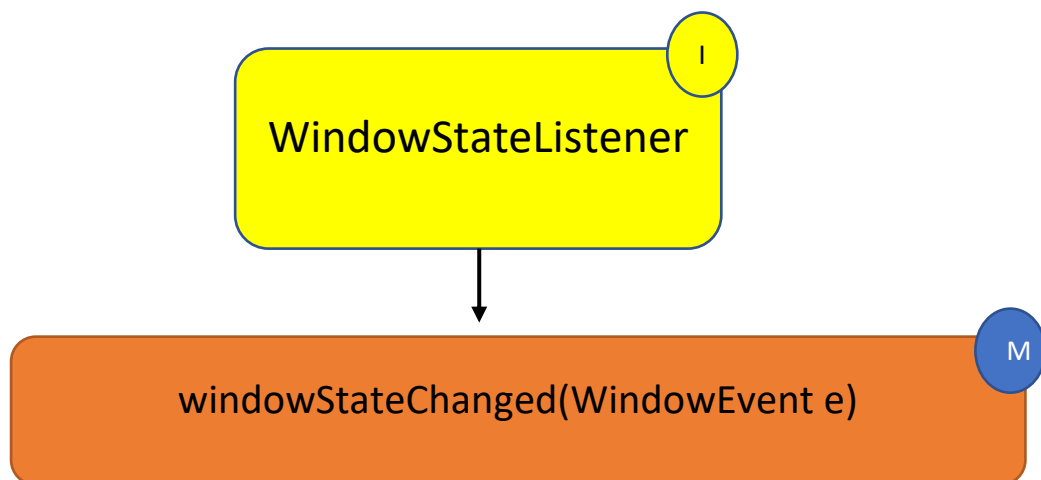
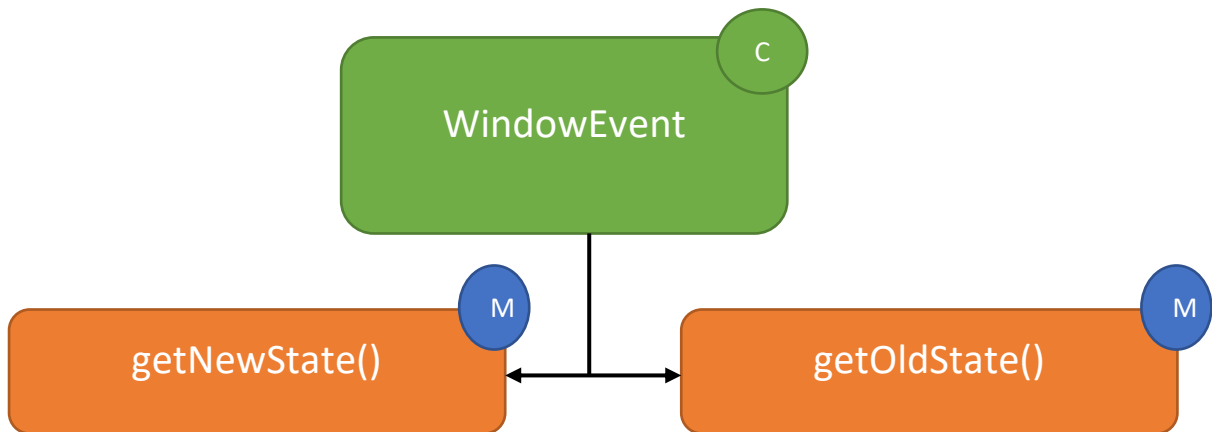
EVENTOS V
Clases Adaptadoras (Adapter Classes)

eclipse Java

Eventos VI. Eventos de ventana III. Controlando estado de ventana. (Vídeo 70)



¿Cómo controlar el estado de la ventana?



Vamos a realizar una nueva clase llamada CambioEstado.

```
1 package graficos;
2 import javax.swing.*;
3 import java.awt.Frame;
4 import java.awt.event.*;
5
6 public class CambioEstado {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoEstado mimarco=new MarcoEstado();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13    }
14
15 }
16 class MarcoEstado extends JFrame{
17     public MarcoEstado() {
18         setVisible(true);
19         setBounds(300,300,500,350);
20         CambiaEstado nuevo_estado=new CambiaEstado();
21         addWindowStateListener(nuevo_estado);
22     }
23 }
24
25 class CambiaEstado implements WindowStateListener{
26     public void windowStateChanged(WindowEvent e) {
27         System.out.println("La ventana ha cambiado de estado");
28     }
29 }
```

Ahora ejecutaremos el programa, minimizamos, restauramos, maximizamos, etc.

Por cada cambio nos aparecerá el siguiente mensaje:

```
La ventana ha cambiado de estado
La ventana ha cambiado de estado
La ventana ha cambiado de estado
La ventana ha cambiado de estado
```



```

1 package graficos;
2 import javax.swing.*;
3 import java.awt.Frame;
4 import java.awt.event.*;
5
6 public class CambioEstado {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoEstado mimarco=new MarcoEstado();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13    }
14
15 }
16 class MarcoEstado extends JFrame{
17     public MarcoEstado() {
18         setVisible(true);
19         setBounds(300,300,500,350);
20         CambiaEstado nuevo_estado=new CambiaEstado();
21         addWindowStateListener(nuevo_estado);
22     }
23 }
24
25 class CambiaEstado implements WindowStateListener{
26     public void windowStateChanged(WindowEvent e) {
27         System.out.println("La ventana ha cambiado de estado");
28         System.out.println(e.getNewState()); ←
29     }
30 }

```

Retorna un valor según el nuevo estado de la ventana.

| | |
|------------------|----|
| MAXIMIZED_BOTH | 6 |
| MAXIMIZED_HORIZ | 2 |
| MAXIMIZED_VERT | 4 |
| MOVE_CURSOR | 13 |
| N_RESIZE_CURSOR | 8 |
| NE_RESIZE_CURSOR | 7 |
| NORMAL | 0 |
| NW_RESIZE_CURSOR | 6 |
| S_RESIZE_CURSOR | 9 |
| SE_RESIZE_CURSOR | 5 |
| SW_RESIZE_CURSOR | 4 |
| TEXT_CURSOR | 2 |
| W_RESIZE_CURSOR | 10 |
| WAIT_CURSOR | 3 |

Si ejecutamos maximizamos, restablecemos y minimizamos este será el resultado en consola.

```

La ventana ha cambiado de estado
6
La ventana ha cambiado de estado
0
La ventana ha cambiado de estado
1

```

Ahora queremos que además de decirnos que ha cambiado de estado, cuando maximicemos la ventana además nos diga “La ventana se ha maximizado”.

```

1 package graficos;
2 import javax.swing.*;
3 import java.awt.Frame;
4 import java.awt.event.*;
5
6 public class CambioEstado {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoEstado mimarco=new MarcoEstado();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13    }
14
15 }
16 class MarcoEstado extends JFrame{
17     public MarcoEstado() {
18         setVisible(true);
19         setBounds(300,300,500,350);
20         CambiaEstado nuevo_estado=new CambiaEstado();
21         addWindowStateListener(nuevo_estado);
22     }
23 }
24
25 class CambiaEstado implements WindowStateListener{
26     public void windowStateChanged(WindowEvent e) {
27         System.out.println("La ventana ha cambiado de estado");
28         //System.out.println(e.getNewState());
29         if(e.getNewState()==6) {
30             System.out.println("La ventana se ha maximizado");
31         }
32     }
33 }

```

Este será el resultado cuando maximicemos la ventana:

```

La ventana ha cambiado de estado
La ventana se ha maximizado

```

```

25 class CambiaEstado implements WindowStateListener{
26     public void windowStateChanged(WindowEvent e) {
27         System.out.println("La ventana ha cambiado de estado");
28         //System.out.println(e.getNewState());
29         if(e.getNewState()==Frame.MAXIMIZED_BOTH) { ←
30             System.out.println("La ventana se ha maximizado");
31         }
32     }
33 }

```

En lugar del 6 podemos poner la contante Frame.MAXINIZED_BOTH, ya que tiene el valor 6.

Ahora queremos controlar cuando está en estado normal y minimizada.

La solución está en la siguiente página.

```

1 package graficos;
2 import javax.swing.*;
3 import java.awt.Frame;
4 import java.awt.event.*;
5
6 public class CambioEstado {
7
8     public static void main(String[] args) {
9         // TODO Auto-generated method stub
10        MarcoEstado mimarco=new MarcoEstado();
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13    }
14
15 }
16 class MarcoEstado extends JFrame{
17     public MarcoEstado() {
18         setVisible(true);
19         setBounds(300,300,500,350);
20         CambiaEstado nuevo_estado=new CambiaEstado();
21         addWindowStateListener(nuevo_estado);
22     }
23 }
24
25 class CambiaEstado implements WindowStateListener{
26     public void windowStateChanged(WindowEvent e) {
27         System.out.println("La ventana ha cambiado de estado");
28         //System.out.println(e.getNewState());
29         if(e.getNewState()==Frame.MAXIMIZED_BOTH) {
30             System.out.println("La ventana se ha maximizado");
31         }else if(e.getNewState()==Frame.NORMAL){
32             System.out.println("La ventana se ha restaurado");
33         }else if(e.getNewState()==Frame.ICONIFIED){
34             System.out.println("La ventana se ha Minimizado");
35         }
36     }
37 }

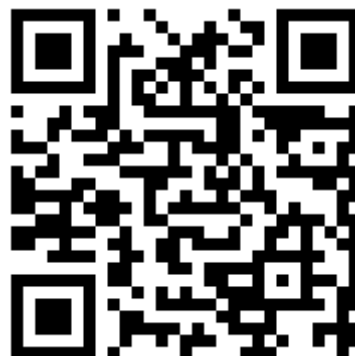
```

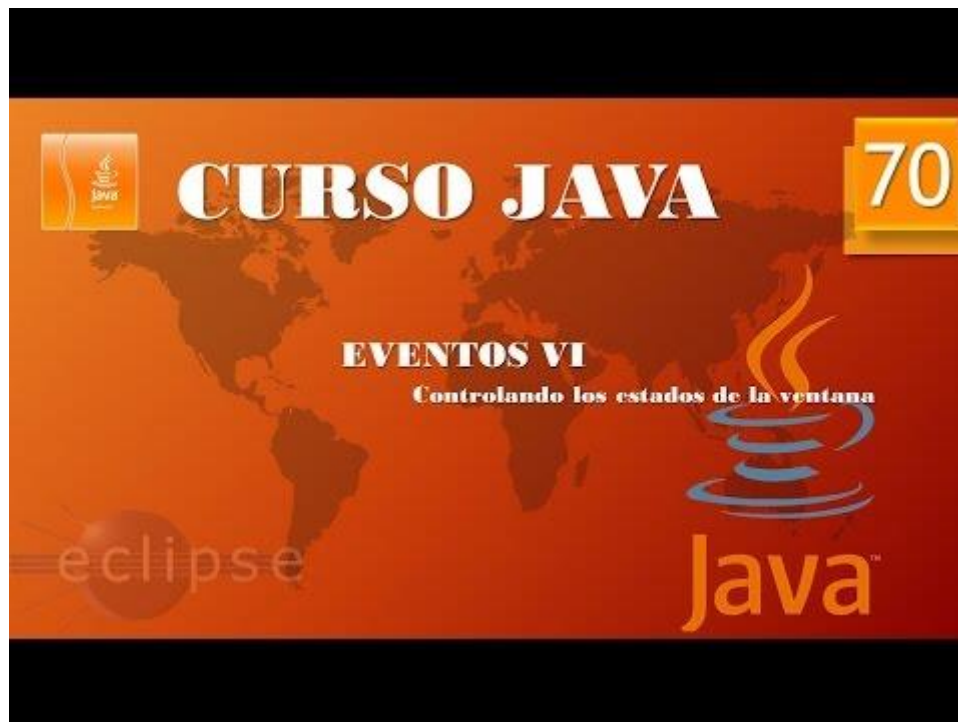
Este será el resultado en consola:

```

La ventana ha cambiado de estado
La ventana se ha maximizado
La ventana ha cambiado de estado
La ventana se ha restaurado
La ventana ha cambiado de estado
La ventana se ha Minimizado

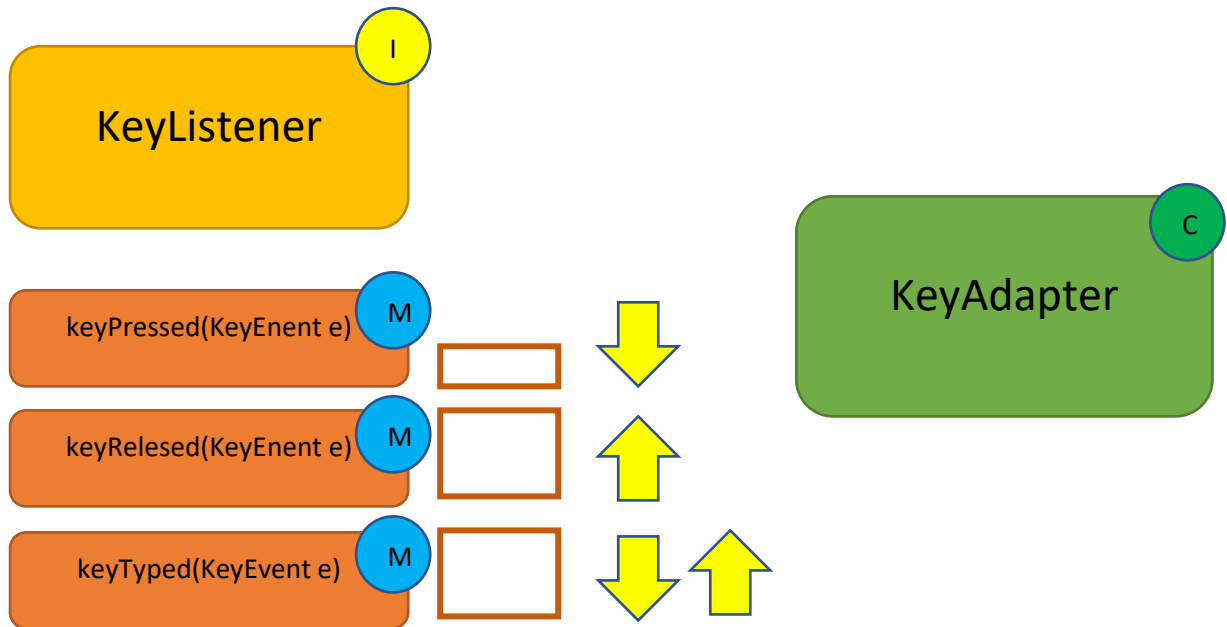
```



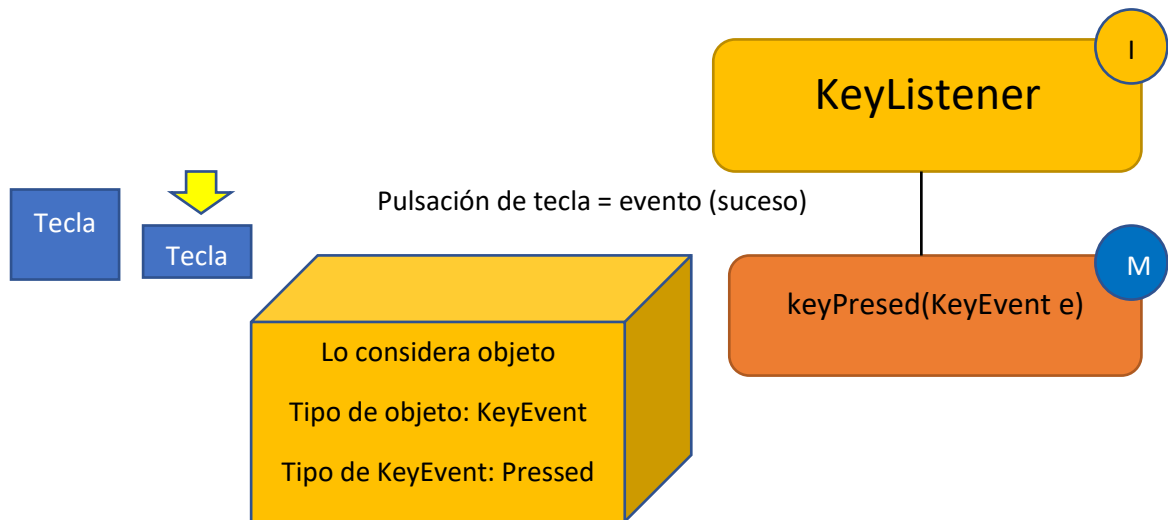


Eventos VII. Eventos de teclado I. (Vídeo 71)

Eventos de teclado.



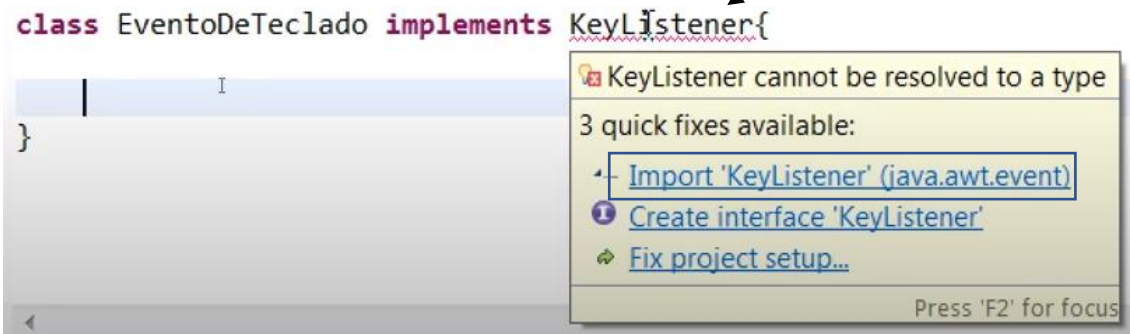
¿Qué sucede realmente?



Con el método `getKeyChar()` → podemos saber que letra hemos pulsado.

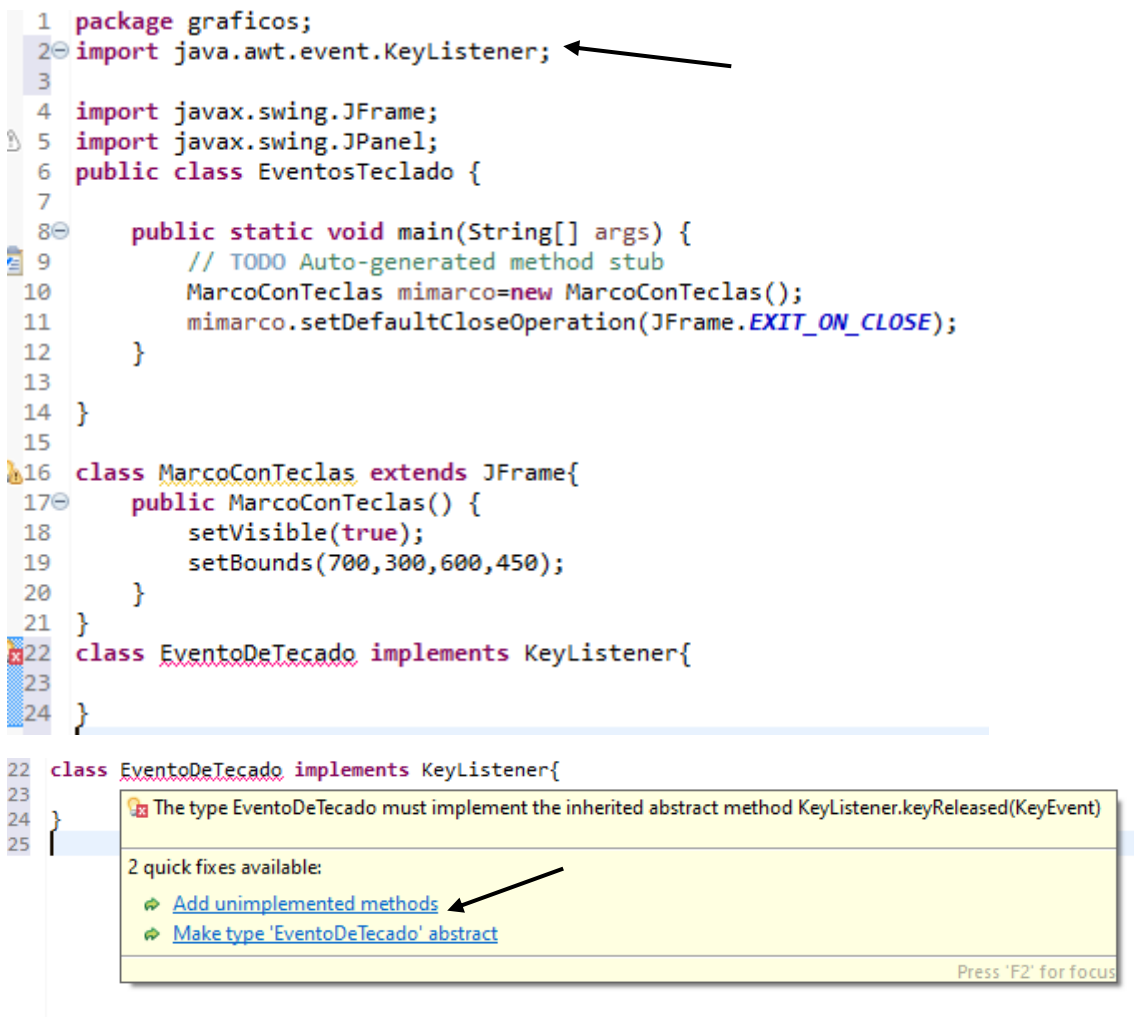
Todas las teclas tienen su código, ejemplo `VK_a` para la letra a y `VK_p` para la letra p que podremos ver con la clase `KeyEvent`.

```
class EventoDeTeclado implements KeyListener{  
    |  
}
```



Gracias a Eclipse si nos colocamos la instancia KeyListener cuando esta da error, nos informa que tenemos que importar el paquete java.awt.event el primer error ya estaría solucionado.

```
1 package graficos;  
2 import java.awt.event.KeyListener; ←  
3  
4 import javax.swing.JFrame;  
5 import javax.swing.JPanel;  
6 public class EventosTeclado {  
7  
8     public static void main(String[] args) {  
9         // TODO Auto-generated method stub  
10        MarcoConTeclas mimarco=new MarcoConTeclas();  
11        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
12    }  
13  
14 }  
15  
16 class MarcoConTeclas extends JFrame{  
17     public MarcoConTeclas() {  
18         setVisible(true);  
19         setBounds(700,300,600,450);  
20     }  
21 }  
22 class EventoDeTecado implements KeyListener{  
23  
24 }  
25
```



Si la clase EventoDeTeclado da error nos colocamos con el ratón y encontramos Add unimplemented methods.

```

23 class EventoDeTeclado implements KeyListener{
24
25     @Override
26     public void keyTyped(KeyEvent e) {
27         // TODO Auto-generated method stub
28
29     }
30
31     @Override
32     public void keyPressed(KeyEvent e) {
33         // TODO Auto-generated method stub
34
35     }
36
37     @Override
38     public void keyReleased(KeyEvent e) {
39         // TODO Auto-generated method stub
40
41     }
42
43 }

```

Eclipse ha agregado los tres métodos de la instancia KeyListener.

Ahora vemos el código completo:

```

1 package graficos;
2 import java.awt.event.KeyEvent;
3 import java.awt.event.KeyListener;
4
5 import javax.swing.JFrame;
6 import javax.swing.JPanel;
7 public class EventosTeclado {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         MarcoConTeclas mimarco=new MarcoConTeclas();
12         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13     }
14 }
15
16 class MarcoConTeclas extends JFrame{
17     public MarcoConTeclas() {
18         setVisible(true);
19         setBounds(700,300,600,450);
20         EventoDeTeclado tecla=new EventoDeTeclado();
21         addKeyListener(tecla);
22     }
23 }
24 class EventoDeTeclado implements KeyListener{
25
26     @Override
27     public void keyTyped(KeyEvent e) {
28         // TODO Auto-generated method stub
29
30     }

```

```

31
32 @Override
33 public void keyPressed(KeyEvent e) {
34     // TODO Auto-generated method stub
35     int codigo=e.getKeyCode();
36     System.out.println(codigo);
37 }
38 @Override
39 public void keyReleased(KeyEvent e) {
40     // TODO Auto-generated method stub
41
42 }
43 }

```

En las líneas 35 y 36 definimos una variable de tipo int llamada código que almacena el código de la tecla que presionamos que luego imprimimos por consola.

En la líneas 20 y 21 una variable llamada tecla de tipo EventoDeTeclado y para poner nuestro objeto a la escucha utilizaremos el método AddKeyListener(tecla) con el parámetro tecla.

Ahora queremos obtener el siguiente resultado:

```

Has pulsado la tecla: j que tiene un código de 74
Has pulsado la tecla: y que tiene un código de 89
Has pulsado la tecla: r que tiene un código de 82
Has pulsado la tecla: l que tiene un código de 76

```

Vamos a modificar el siguiente código del método keyPressed(KeyEvent e)

```

33 public void keyPressed(KeyEvent e) {
34     // TODO Auto-generated method stub
35     int codigo=e.getKeyCode();
36     System.out.println("Has pulsado la tecla: "+
37     e.getKeyChar()+ " que tiene un código de " +codigo);
38 }

```

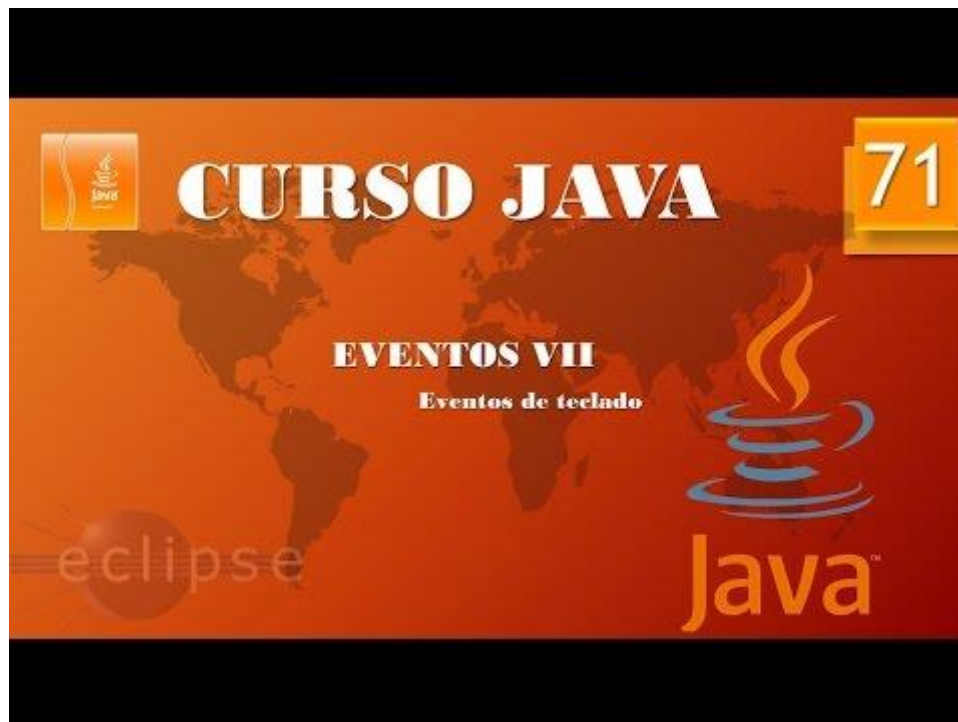
Ahora queremos que la soltar la tecla nos diga He soltado la tecla X, para eso vamos a modificar el método keyTyped(keyEvent e).

```

27 public void keyTyped(KeyEvent e) {
28     // TODO Auto-generated method stub
29     char letra=e.getKeyChar();
30     System.out.println("He soltado la tecla " + letra );
31 }

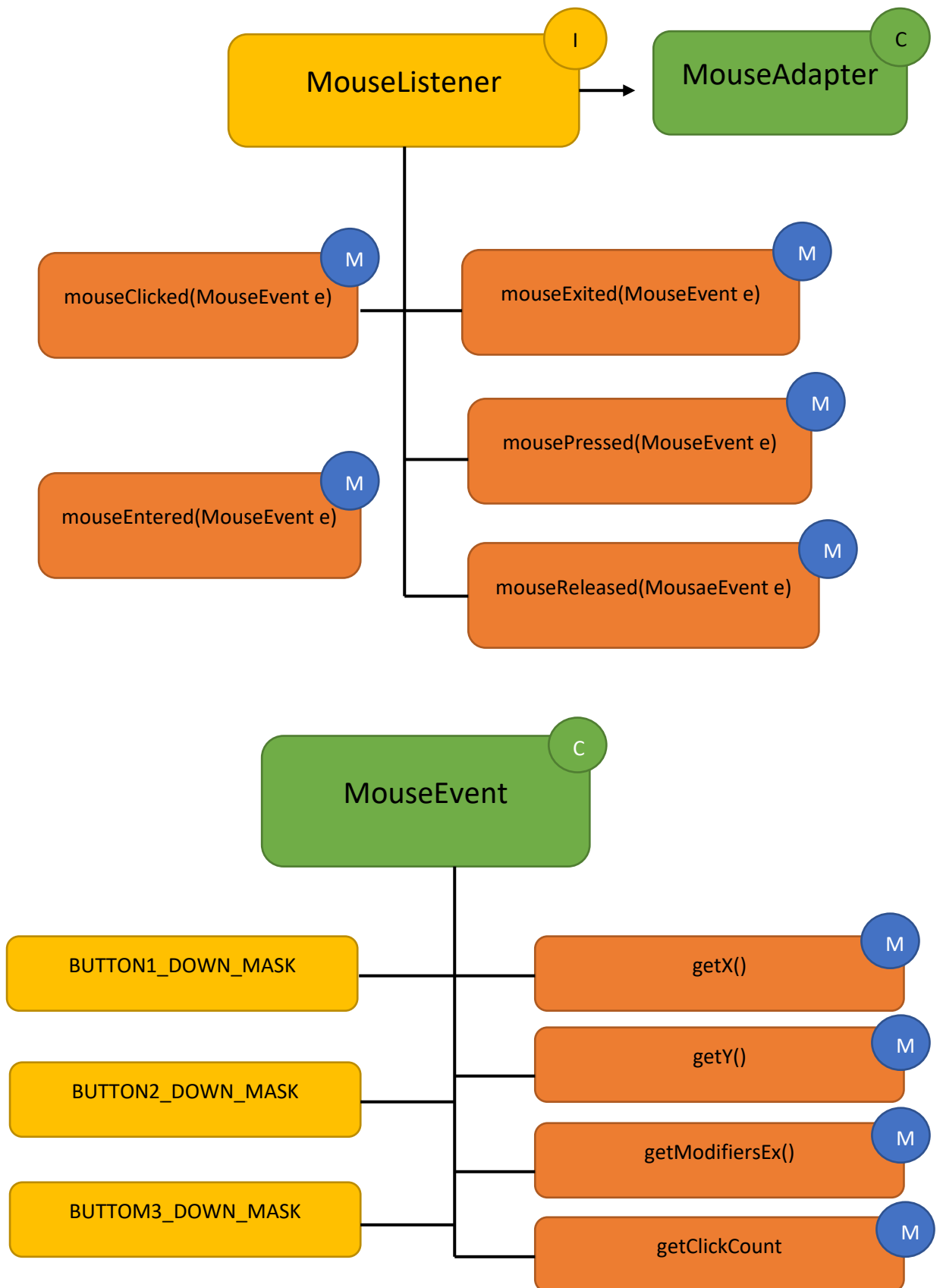
```





Eventos VIII. Eventos de ratón I. (Vídeo 72)

Eventos de ratón.



```

1 package graficos;
2 import java.awt.event.*;
3 import javax.swing.JFrame;
4
5 public class Eventos_Raton {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoRaton mimarco=new MarcoRaton();
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12 }
13 class MarcoRaton extends JFrame{
14     public MarcoRaton() {
15         setVisible(true);
16         setBounds(700,300,600,540);
17         EventosDeRaton EventoRaton=new EventosDeRaton();
18         addMouseListener(EventoRaton); //Ponemos MarcoRaton a la escucha
19     }
20 }
21
22 class EventosDeRaton implements MouseListener{
23     @Override
24     public void mouseClicked(MouseEvent e) {
25         // TODO Auto-generated method stub
26         System.out.println("Has hecho click");
27     }
28     @Override
29     public void mousePressed(MouseEvent e) {
30         // TODO Auto-generated method stub
31     }
32     @Override
33     public void mouseReleased(MouseEvent e) {
34         // TODO Auto-generated method stub
35     }
36     @Override
37     public void mouseEntered(MouseEvent e) {
38         // TODO Auto-generated method stub
39     }
40     @Override
41     public void mouseExited(MouseEvent e) {
42         // TODO Auto-generated method stub
43     }
44 }

```

Evento click

Ahora haremos clic sobre la ventana.

Este será el resultado:

```
Has hecho click
```

Ahora vamos a modificar el código de la clase EventosDeRaton.

```

22 class EventosDeRaton implements MouseListener{
23     @Override
24     public void mouseClicked(MouseEvent e) {
25         // TODO Auto-generated method stub
26         System.out.println("Has hecho click");
27     }
28     @Override
29     public void mousePressed(MouseEvent e) {
30         // TODO Auto-generated method stub
31         System.out.println("Acabas de presionar."); ←
32     }
33     @Override
34     public void mouseReleased(MouseEvent e) {
35         // TODO Auto-generated method stub
36         System.out.println("Acabas de levantar."); ←
37     }
38     @Override
39     public void mouseEntered(MouseEvent e) {
40         // TODO Auto-generated method stub
41     }
42     @Override
43     public void mouseExited(MouseEvent e) {
44         // TODO Auto-generated method stub
45     }
46 }

```

Ahora cuando presionemos el botón izquierdo, sin levantar el dedo se realizará el evento mousePressed, cuando levantemos el dedo mouseReleased y por último el evento mouseClicked. Este será el resultado:

Acabas de presionar.
Acabas de levantar.
Has hecho click

Vamos a modificar el código de la clase EventosDeRaton.

```

22 class EventosDeRaton implements MouseListener{
23     @Override
24     public void mouseClicked(MouseEvent e) {
25         // TODO Auto-generated method stub
26         System.out.println("Has hecho click");
27     }
28     @Override
29     public void mousePressed(MouseEvent e) {
30         // TODO Auto-generated method stub
31         System.out.println("Acabas de presionar.");
32     }
33     @Override
34     public void mouseReleased(MouseEvent e) {
35         // TODO Auto-generated method stub
36         System.out.println("Acabas de levantar.");
37     }
38     @Override
39     public void mouseEntered(MouseEvent e) {
40         // TODO Auto-generated method stub
41         System.out.println("Acabas de entrar."); ←
42     }
43     @Override
44     public void mouseExited(MouseEvent e) {
45         // TODO Auto-generated method stub
46         System.out.println("Acabas de salir."); ←
47     }
48 }

```

Ejecuta y con el ratón entra y salte de la ventana, este será el resultado:

Acabas de entrar.
Acabas de salir.

Otra forma de realizar la clase EventosDeRaton sin implementar todos los métodos ya que solo nos interesa el evento click.

```
22 class EventosDeRaton extends MouseAdapter {  
23     public void mouseClicked(MouseEvent e) {  
24         System.out.println("Has hecho click.");  
25     }  
26 }
```

Si ejecutamos solo reconocerá el evento click.

```
Has hecho click.
```



Eventos IX. Eventos de ratón II. (Vídeo 73)

Vamos a trabajar con el ejemplo anterior.

```
1 package graficos;
2 import java.awt.event.*;
3
4
5 public class Eventos_Raton {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoRaton mimarco=new MarcoRaton();
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12 }
13 class MarcoRaton extends JFrame{
14     public MarcoRaton() {
15         setVisible(true);
16         setBounds(700,300,600,540);
17         EventosDeRaton EventoRaton=new EventosDeRaton();
18         addMouseListener(EventoRaton); //Ponemos MarcoRaton a la escucha
19     }
20 }
21
22 class EventosDeRaton extends MouseAdapter {
23     public void mouseClicked(MouseEvent e) {
24         System.out.println("Cordenada X: " + e.getX() +
25             " Coordenada Y: " + e.getY());
26     }
27 }
```

Con e.getX() y e.getY() podemos saber en qué coordenada del marco de la ventana hacemos clic, este será el resultado después de realizar 3 clicks en distintas zonas del marco.

```
Cordenada X: 155 Coordenada Y: 115
Cordenada X: 137 Coordenada Y: 315
Cordenada X: 394 Coordenada Y: 449
```

```
22 class EventosDeRaton extends MouseAdapter {
23     public void mouseClicked(MouseEvent e) {
24         System.out.println(e.getClickCount());
25     }
26 }
```

Ahora en la clase EventosDeRaton vamos a modificar el código con e.getClickCount(), este método cuenta las veces que hacemos un click con el botón izquierdo, para que cuente los clicks estos tienen que hacerse con rapidez, ya que sí no contara 1, 1, 1 y 1 .

Este será el resultado:

```
1
2
3
4
```

```

1 package graficos;
2 import java.awt.event.*;
3
4
5 public class Eventos_Raton {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoRaton mimarco=new MarcoRaton();
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12 }
13 class MarcoRaton extends JFrame{
14     public MarcoRaton() {
15         setVisible(true);
16         setBounds(700,300,600,540);
17         EventosDeRaton EventoRaton=new EventosDeRaton();
18         addMouseListener(EventoRaton); //Ponemos MarcoRaton a la escucha
19     }
20 }
21
22 class EventosDeRaton extends MouseAdapter {
23     public void mousePressed(MouseEvent e) {
24         System.out.println(e.getModifiersEx());
25     }
26 }
27 }

```

Con el método `getModifiersEx()` podemos saber que botón del ratón hemos pulsado, al ejecutar y pulsar botón izquierdo, central o rueda y derecho, este será el resultado:

```

1024
2048
4096

```

Si queremos que en consola nos diga si hemos pulsado el botón izquierdo, central o derecho vamos a modificar la clase `EventosDeRaton`.

```

22 class EventosDeRaton extends MouseAdapter {
23     public void mousePressed(MouseEvent e) {
24         if(e.getModifiersEx()==1024) {
25             System.out.println("Ha pulado el botón izquierdo del ratón.");
26         }
27         if(e.getModifiersEx()==2048) {
28             System.out.println("Ha pulado el botón central del ratón.");
29         }
30         if(e.getModifiersEx()==4096) {
31             System.out.println("Ha pulado el botón derecho del ratón.");
32         }
33     }
34 }

```

Este será el resultado:

```

Ha pulado el botón izquierdo del ratón.
Ha pulado el botón central del ratón.
Ha pulado el botón derecho del ratón.

```

También puedes utilizar las contantes

BUTTON1_DOWN_MASK que tiene un valor de 1024.

BUTTON2_DOWN_MASK que tiene un valor de 2048.

BUTTON3_DOWN_MASK que tiene un valor de 4096.

```
22 class EventosDeRaton extends MouseAdapter {
23     public void mousePressed(MouseEvent e) {
24         if(e.getModifiersEx()==MouseEvent.BUTTON1_DOWN_MASK) {
25             System.out.println("Ha pulado el botón izquierdo del ratón.");
26         }
27         else if(e.getModifiersEx()==MouseEvent.BUTTON2_DOWN_MASK) {
28             System.out.println("Ha pulado el botón central del ratón.");
29         }
30         else if(e.getModifiersEx()==MouseEvent.BUTTON3_DOWN_MASK) {
31             System.out.println("Ha pulado el botón derecho del ratón.");
32         }
33     }
34 }
```

De esta forma también funcionaria.

```
1 package graficos;
2 import java.awt.event.*;
3
4
5 public class Eventos_Raton {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoRaton mimarco=new MarcoRaton();
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12 }
13 class MarcoRaton extends JFrame{
14     public MarcoRaton() {
15         setVisible(true);
16         setBounds(700,300,600,540);
17         EventosDeRaton EventoRaton=new EventosDeRaton();
18         addMouseListener(EventoRaton); //Ponemos MarcoRaton a la escucha
19     }
20 }
21
22 class EventosDeRaton implements MouseMotionListener {
23
24     @Override
25     public void mouseDragged(MouseEvent e) {
26         // TODO Auto-generated method stub
27         System.out.println("Estás arrastando.");
28     }
29
30     @Override
31     public void mouseMoved(MouseEvent e) {
32         // TODO Auto-generated method stub
33         System.out.println("Estás moviendo.");
34     }
35
36 }
37 }
```


Con este ejemplo vamos a controlar si estaos moviendo el ratón o arrastrándolo, es decir manteniendo pulsado el botón izquierdo del ratón.

Para ello la clase EventosDeRaton le vamos a implementar la interface MouseMotionListener.

Esto obliga a sobrescribir dos métodos, uno para arrastrar y otro para mover.

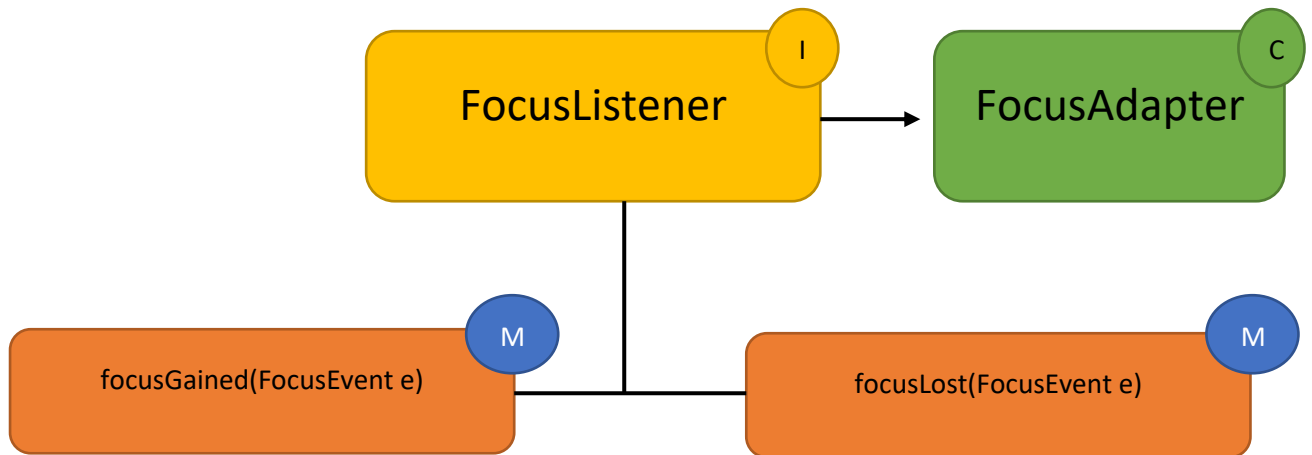
Este será el resultado en consola.

```
Estás arrastando.  
Estás arrastando.  
Estás arrastando.  
Estás moviendo.  
Estás moviendo.  
Estás moviendo.  
Estás moviendo.
```

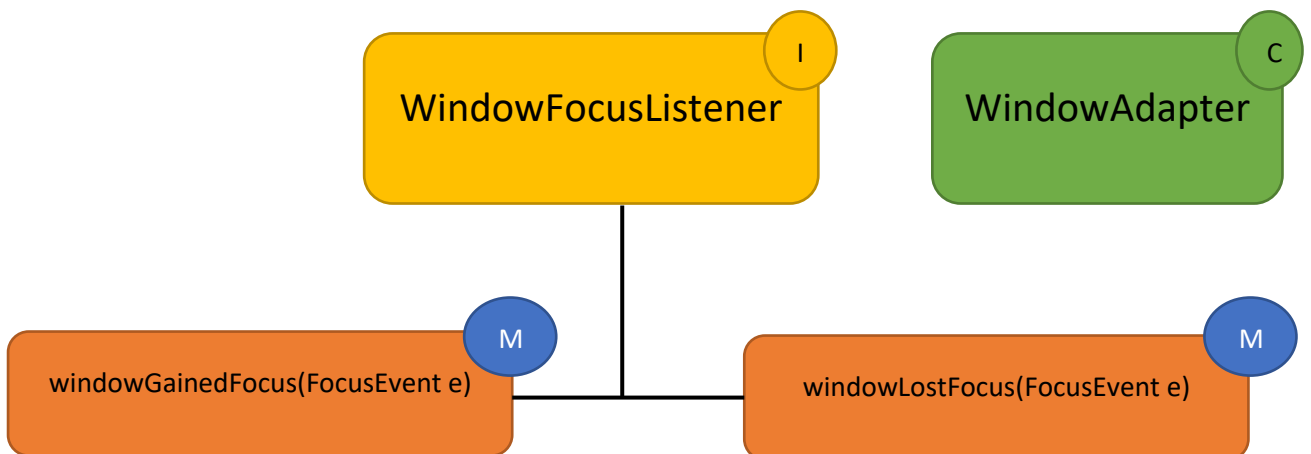


Eventos X. Eventos de foco. (Vídeo 74)

Eventos de foco (componentes)



En el caso de ventanas:



Vamos a crear una clase llamada FocoEvento.

```
1 package graficos;
2 import java.awt.*;
3 import javax.swing.*;
4
5 public class FocoEvento {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoFoco mimarco=new MarcoFoco();
10        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11    }
12 }
13
14 class MarcoFoco extends JFrame{
15     public MarcoFoco() {
16         setVisible(true);
17         setBounds(300,300,600,450);
18         add(new LaminaFoco());
19     }
20 }
21 class LaminaFoco extends JPanel{
22     public void paintComponent(Graphics g) {
23         super.paintComponent(g);
24         setLayout(null); //Desactiva el centrado automático.
25         cuadro1=new JTextField();
26         cuadro2=new JTextField();
27         cuadro1.setBounds(120,10,150,20);
28         cuadro2.setBounds(120,50,150,20);
29         add(cuadro1);
30         add(cuadro2);
31     }
32     JTextField cuadro1;
33     JTextField cuadro2;
34 }
```

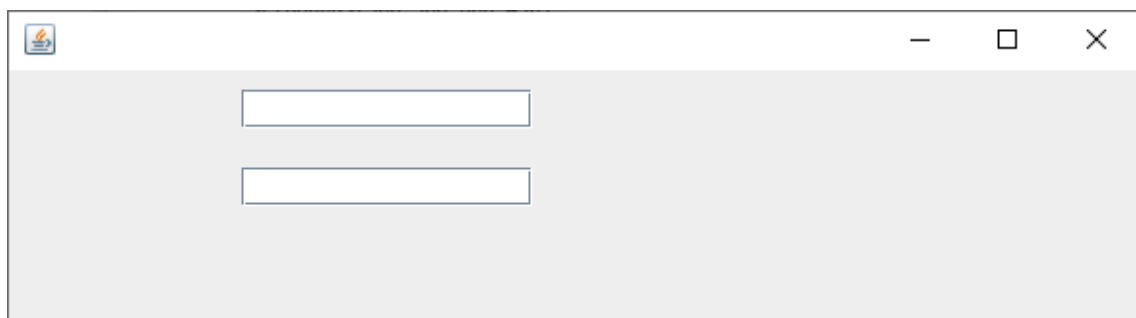
En la clase LaminaFoco declaramos dos objetos de tipo JTextField cuadro1 y cuadro2.

Las definimos de nuevo en el método paintComponent las dos variables de Tipo JTextField, le damos posicionamiento y dimensiones con setBound y la agregamos a la lámima con add.

En la clase MarcoFoco agregamos la LaminaFoco(), con add.

El la clase principal creamos un objeto llamado mimarco de tipo MarcoFoco.

Si ejecutamos este será el resultado:



Una ventana con dos cajas de texto.

```
23 class LaminaFoco extends JPanel{
24     public void paintComponent(Graphics g) {
25         super.paintComponent(g);
26         setLayout(null); //Desactiva el centrado automático.
27         cuadro1=new JTextField();
28         cuadro2=new JTextField();
29         cuadro1.setBounds(120,10,150,20);
30         cuadro2.setBounds(120,50,150,20);
31         add(cuadro1);
32         add(cuadro2);
33     }
34     private class LanzaFocos implements FocusListener{
35
36         @Override
37         public void focusGained(FocusEvent e) {
38             // TODO Auto-generated method stub
39
40         }
41
42         @Override
43         public void focusLost(FocusEvent e) {
44             // TODO Auto-generated method stub
45
46         }
47     }
48
49     JTextField cuadro1;
50     JTextField cuadro2;
51 }
```

La clase LanzaFocos la hacemos interna de la clase LaminaFoco con el fin de poder tener acceso a los objetos cuadro1 y cuadro de tipo JTextField.

```
23 class LaminaFoco extends JPanel{
24     public void paintComponent(Graphics g) {
25         super.paintComponent(g);
26         setLayout(null); //Desactiva el centrado automático.
27         cuadro1=new JTextField();
28         cuadro2=new JTextField();
29         cuadro1.setBounds(120,10,150,20);
30         cuadro2.setBounds(120,50,150,20);
31         add(cuadro1);
32         add(cuadro2);
33         LanzaFocos elFoco = new LanzaFocos();
34         cuadro1.addFocusListener(elFoco);
35     }
36     private class LanzaFocos implements FocusListener{
```

Ahora el cuadro1 lo hemos puesto a la escucha agregando estas dos líneas.

```

private class LanzaFocos implements FocusListener{

    @Override
    public void focusGained(FocusEvent e) {
        // TODO Auto-generated method stub
        System.out.println("He ganado el foco.");
    }

    @Override
    public void focusLost(FocusEvent e) {
        // TODO Auto-generated method stub
        System.out.println("He perdido el foco.");
    }

}

```

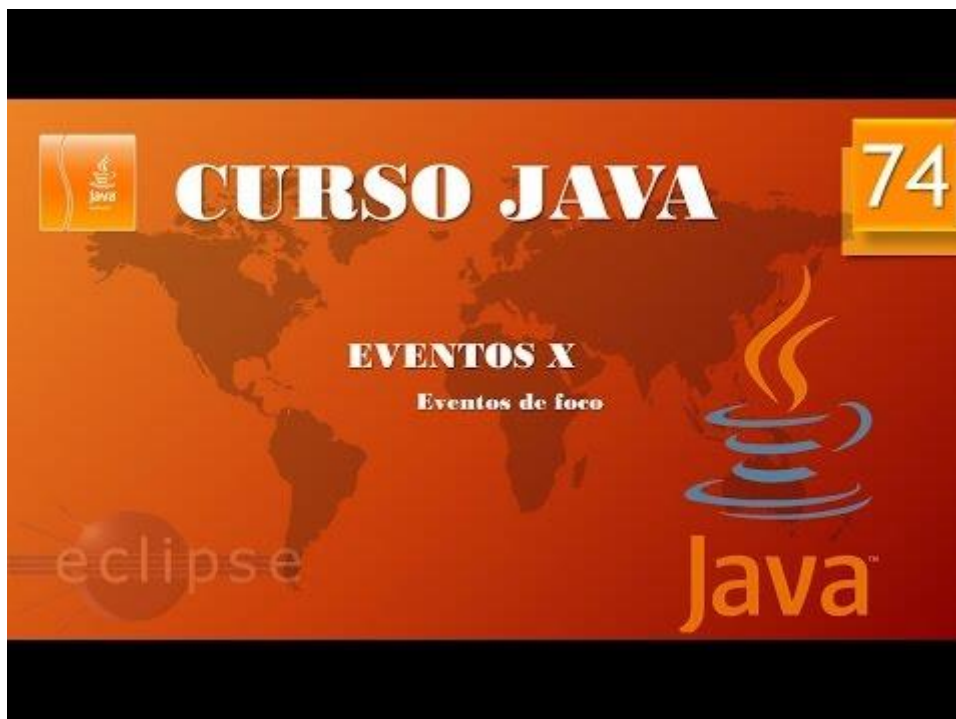
En la clase LanzaFocos hemos agregado el código que nos diga del cuadro1 cuando perdemos y ganamos el foco.

Este será el resultado:

```

He ganado el foco.
He perdido el foco.

```



Eventos XI. Eventos de foco II. (Vídeo 75)

Vamos a seguir con el proyecto anterior.

Al introducir una dirección de correo electrónico de valore si es correcta o no.

Solo vamos a evaluar si el correo introducido tiene la @ o no.

```
package graficos;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class FocoEvento {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MarcoFoco mimarco=new MarcoFoco();
        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

class MarcoFoco extends JFrame{
    public MarcoFoco() {
        setVisible(true);
        setBounds(300,300,600,450);
        add(new LaminaFoco());
    }
}

class LaminaFoco extends JPanel{
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        setLayout(null);//Desactiva el centrado automático.
        cuadro1=new JTextField();
        cuadro2=new JTextField();
        cuadro1.setBounds(120,10,150,20);
        cuadro2.setBounds(120,50,150,20);
        add(cuadro1);
        add(cuadro2);
        LanzaFocos elFoco = new LanzaFocos();
        cuadro1.addFocusListener(elFoco);
    }

    private class LanzaFocos implements FocusListener{

        @Override
        public void focusGained(FocusEvent e) {
            // TODO Auto-generated method stub

        }

        @Override
        public void focusLost(FocusEvent e) {
            // TODO Auto-generated method stub
            String email=cuadro1.getText();
            boolean comprobacion=false;
            for(int i=0;i<email.length(); i++ ) {
                if(email.charAt(i)=='@') {
                    comprobacion=true;
                }
            }
        }
    }
}
```

```

    }
    }
    JTextField cuadro1;
    JTextField cuadro2;
}

```

```

    if(comprobacion) {
        System.out.println("Email correcto.");
    }
    else {
        System.out.println("Email no correcto");
    }
}

```

Para pasar el valor del cuadro1 a una variable utilizamos → `String email=cuadro1.getText();`

Definimos una variable de tipo boolean llamada comprobación igual a false.

Hacemos un ciclo for donde la variable i empieza por 0 y con `email.length()` sabemos el número de caracteres que tiene la variable email.

`Email.charAt(i)` recorre carácter por carácter comprobando si encuentra una @.

En caso afirmativo la variable comprobación pasa a valer true.

En el siguiente recuadro con un if comprobamos si el email es correcto o no.

Perdida y ganancia de foco por ventana.

Vamos a crear una nueva clase llamada FocoVentana.

```

package graficos;

import java.awt.event.WindowEvent;
import java.awt.event.WindowFocusListener;

import javax.swing.*.*;

public class FocoVentana extends JFrame implements WindowFocusListener{

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        FocoVentana miv=new FocoVentana();
        miv.iniciar();
    }

    public void iniciar() {
        marco1=new FocoVentana();
        marco2=new FocoVentana();
        marco1.setVisible(true);
        marco2.setVisible(true);
        marco1.setBounds(300,100,600,350);
        marco2.setBounds(1200,100,600,350);
        marco1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marco2.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marco1.addWindowFocusListener(this);
        marco2.addWindowFocusListener(this);
    }
}

```

```

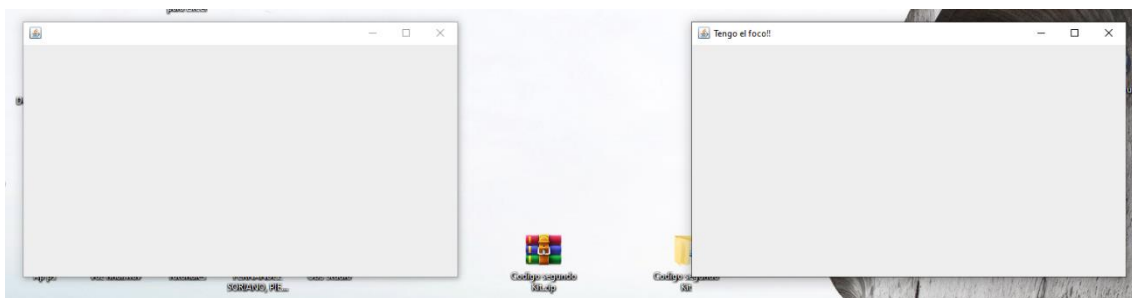
@Override
public void windowGainedFocus(WindowEvent e) {
    // TODO Auto-generated method stub
    if(e.getSource()==marco1) {
        marco1.setTitle("Tengo el foco!!");
    }else {
        marco2.setTitle("Tengo el foco!!");
    }
}

@Override
public void windowLostFocus(WindowEvent e) {
    // TODO Auto-generated method stub
    if(e.getSource()==marco1) {
        marco1.setTitle("");
    }else {
        marco2.setTitle("");
    }
}

FocoVentana marco1;
FocoVentana marco2;
}

```

Este será el resultado:

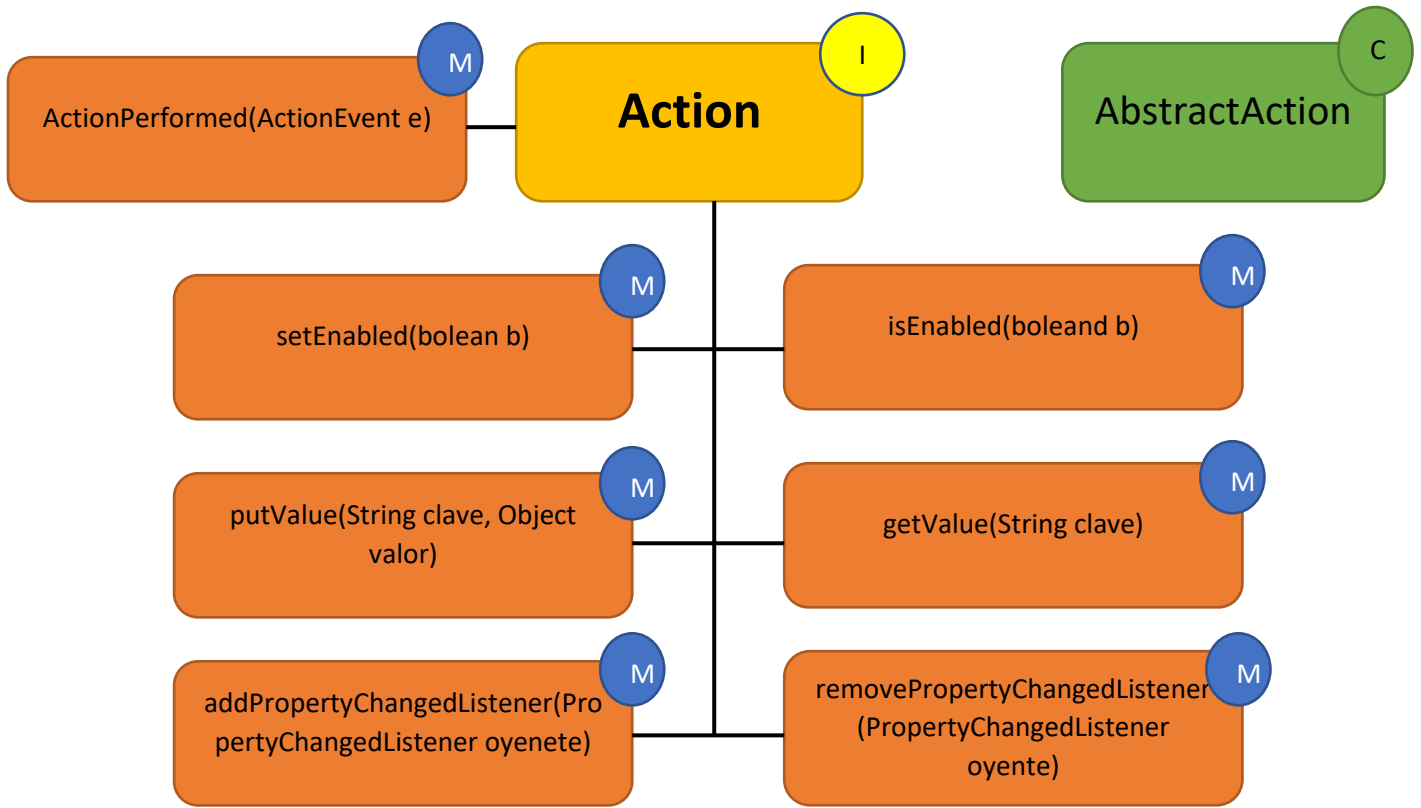


Haz clic a una ventana y luego a la otra y observarás como en la barra de títulos veremos al ventana que tiene el foco.



Eventos XII. Múltiples fuentes I. (Vídeo 76)

Múltiples fuentes de evento.



Vamos a crear una nueva clase llamada PruebaAcciones.

```
1 package graficos;
2 import java.awt.*;
3 import java.awt.event.*;
4 import javax.swing.*;
5 public class PruebaAcciones {
6
7     public static void main(String[] args) {
8         // TODO Auto-generated method stub
9         MarcoAccion marco=new MarcoAccion();
10        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        marco.setVisible(true);
12
13    }
14
15 }
16
17 class MarcoAccion extends JFrame{
18     public MarcoAccion() {
19         setTitle("Prueba Acciones");
20         setBounds(600,350,600,300);
21         PanelAccion lamina=new PanelAccion();
22         add(lamina);
23     }
24
25 }
```

```

26 class PanelAccion extends JPanel{
27     public PanelAccion() {
28         JButton botonAmarillo = new JButton("Amarillo");
29         JButton botonAzul = new JButton("Azul");
30         JButton botonRojo = new JButton("Rojo");
31         add(botonAmarillo);
32         add(botonAzul);
33         add(botonRojo);
34     }
35 }
36 class AccionColor extends AbstractAction{
37
38     @Override
39     public void actionPerformed(ActionEvent e) {
40         // TODO Auto-generated method stub
41
42     }
43
44 }

```

En el siguiente capitulo seguiremos con este proyecto.

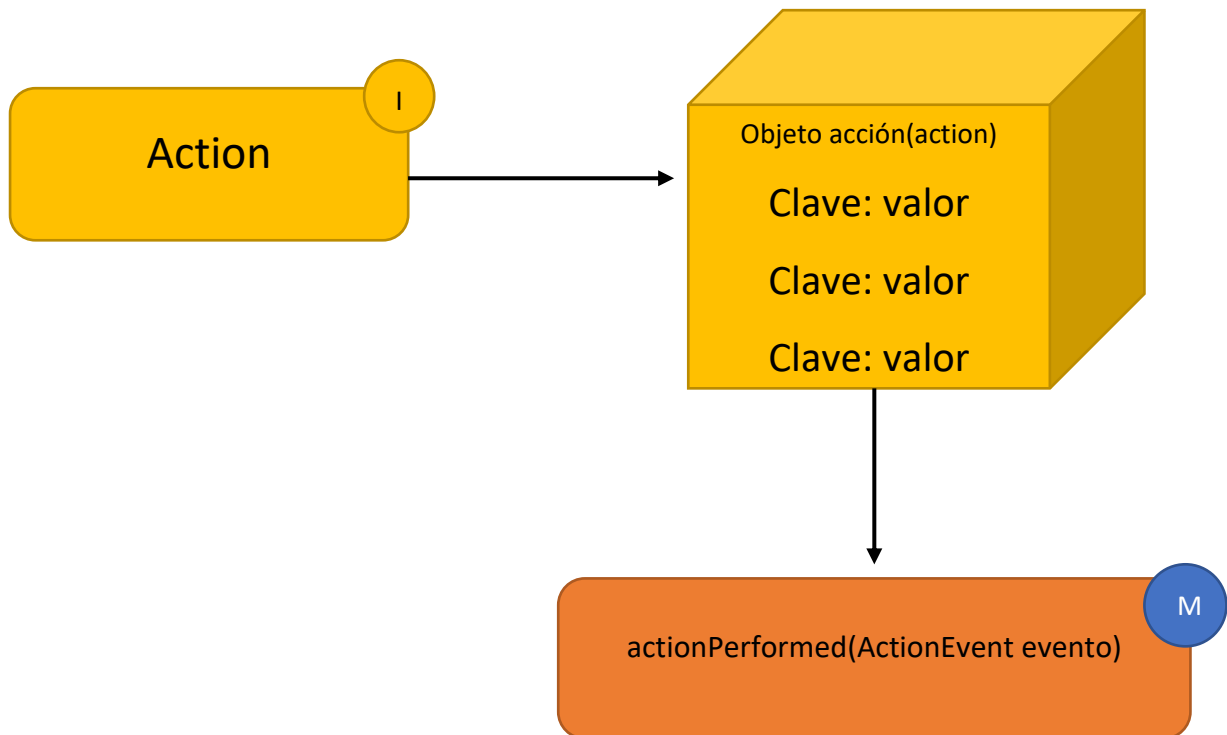
Este es el resultado hasta el momento.





Eventos XIII. Múltiples fuentes II. (Vídeo 77)

Múltiples fuentes de evento.



```
package graficos;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class PruebaAcciones {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MarcoAccion marco=new MarcoAccion();
        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marco.setVisible(true);
    }
}

class MarcoAccion extends JFrame{
    public MarcoAccion() {
        setTitle("Prueba Acciones");
        setBounds(600,350,600,300);
        PanelAccion lamina=new PanelAccion();
        add(lamina);
    }
}

class PanelAccion extends JPanel{
    public PanelAccion() {
        AccionColor accionAmarillo=new AccionColor("Amarillo", new
        ImageIcon("src/graficos/bolaAmarilla.png"), Color.YELLOW);
```

```

        AccionColor accionAzul=new AccionColor("Azul", new
ImageIcon("src/graficos/bolaAzul.png"), Color.BLUE);
        AccionColor accionRojo=new AccionColor("Rojo", new
ImageIcon("src/graficos/bolaRoja.png"), Color.RED);
        //JButton botonAmarillo = new JButton("Amarillo");
        //JButton botonAzul = new JButton("Azul");
        //JButton botonRojo = new JButton("Rojo");
        //add(botonAmarillo);
        //add(botonAzul);
        //add(botonRojo);
    }
}
class AccionColor extends AbstractAction{
    public AccionColor(String nombre, Icon icono, Color color_boton) {
        putValue(Action.NAME, nombre);
        putValue(Action.SMALL_ICON, icono);
        putValue(Action.SHORT_DESCRIPTION, "Poner la lámina de color "+
nombre);
        putValue("color_de_fondo", color_boton);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
    }
}
}

```

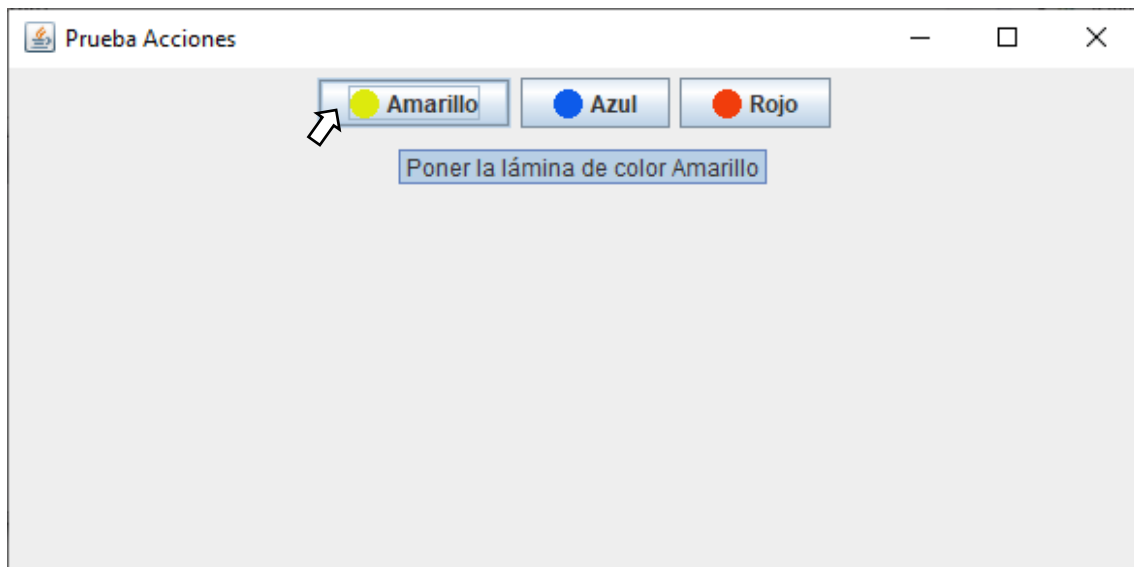


Eventos XIV. Múltiples fuentes III. (Vídeo 78)

Vamos a modificar la clase PanelAccion.

```
class PanelAccion extends JPanel{
    public PanelAccion() {
        AccionColor accionAmarillo=new AccionColor("Amarillo", new
ImageIcon("src/graficos/bolaAmarilla.png"), Color.YELLOW);
        AccionColor accionAzul=new AccionColor("Azul", new
ImageIcon("src/graficos/bolaAzul.png"), Color.BLUE);
        AccionColor accionRojo=new AccionColor("Rojo", new
ImageIcon("src/graficos/bolaRoja.png"), Color.RED);
        JButton botonAmarillo=new JButton(accionAmarillo);
        add(botonAmarillo);//Es una forma
        add(new JButton(accionAzul));
        add(new JButton(accionRojo));
        //JButton botonAmarillo = new JButton("Amarillo");
        //JButton botonAzul = new JButton("Azul");
        //JButton botonRojo = new JButton("Rojo");
        //add(botonAmarillo);
        //add(botonAzul);
        //add(botonRojo);
    }
}
```

Este será el resultado:



Los botones aun no funcionan.

```

class PanelAccion extends JPanel{
    public PanelAccion() {
        AccionColor accionAmarillo=new AccionColor("Amarillo", new
ImageIcon("src/graficos/bolaAmarilla.png"), Color.YELLOW);
        AccionColor accionAzul=new AccionColor("Azul", new
ImageIcon("src/graficos/bolaAzul.png"), Color.BLUE);
        AccionColor accionRojo=new AccionColor("Rojo", new
ImageIcon("src/graficos/bolaRoja.png"), Color.RED);
        JButton botonAmarillo=new JButton(accionAmarillo);
        add(botonAmarillo);//Es una forma
        add(new JButton(accionAzul));
        add(new JButton(accionRojo));
        //JButton botonAmarillo = new JButton("Amarillo");
        //JButton botonAzul = new JButton("Azul");
        //JButton botonRojo = new JButton("Rojo");
        //add(botonAmarillo);
        //add(botonAzul);
        //add(botonRojo);
    }
    private class AccionColor extends AbstractAction{
        public AccionColor(String nombre, Icon icono, Color color_boton)
        {
            putValue(Action.NAME, nombre);
            putValue(Action.SMALL_ICON, icono);
            putValue(Action.SHORT_DESCRIPTION,"Poner la lámina de
color "+ nombre);
            putValue("color_de_fondo", color_boton);
        }
        @Override
        public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            Color c=(Color)getValue("color_de_fondo");
            setBackground(c);
        }
    }
}

```

La clase accionColor la pasamos a ser interna de la clase PanelAccion y de este modo reconoce los métodos JPanel, que estando fuera no los reconocía.

Este será el resultado:



Los botones ya cambian el color de la lámina.

Para entender mejor el método `getValue` vamos a realizar una pequeña modificación en el código que después borraremos.

```
public void actionPerformed(ActionEvent e) {  
    // TODO Auto-generated method stub  
    Color c=(Color)getValue("color_de_fondo");  
    setBackground(c);  
    System.out.println("Nombre: "+ getValue(Action.NAME) + " Descripción:" + getValue(Action.SHORT_DESCRIPTION));  
}
```

```
System.out.println("Nombre: "+ getValue(Action.NAME) + " Descripción:" +  
getValue(Action.SHORT_DESCRIPTION));
```

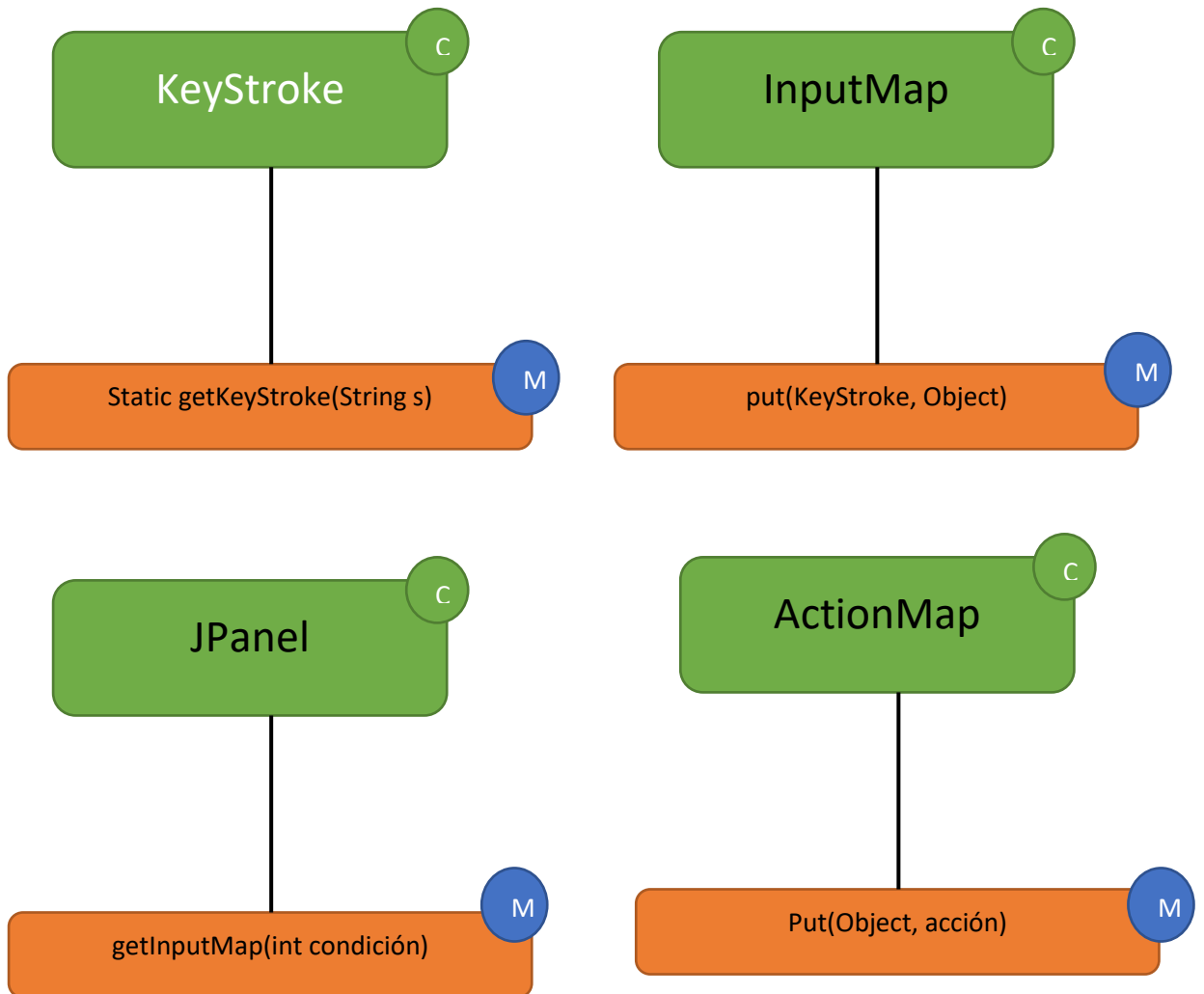
Así obtendremos por consola la siguiente información, según el botón que accionemos.

```
Nombre: Amarillo Descripción:Poner la lámina de color Amarillo  
Nombre: Azul Descripción:Poner la lámina de color Azul  
Nombre: Rojo Descripción:Poner la lámina de color Rojo
```

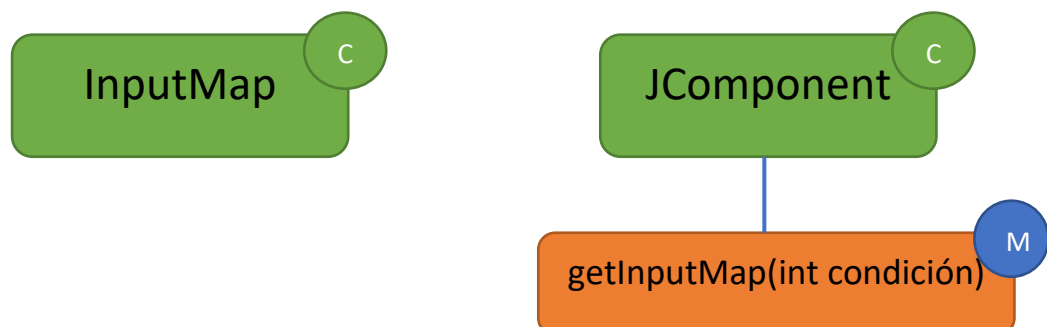


Eventos XV. Múltiples fuentes IV. (Vídeo 79)

Asignando acciones por teclado.

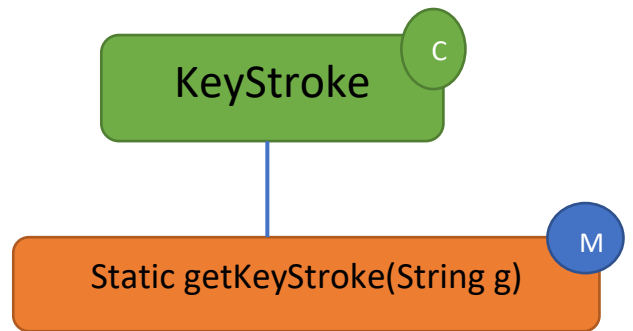


1. Crear mapa de entrada.
2. Crear combinación de teclas (Ctrl +A, Ctrl + B y Ctrl +R).
3. Asignar combinación de teclas a objeto (fondo_amarillo, fondo_azul y fondo_rojo).
4. Asignar objeto a acción(accionAmarillo, accionAzul y accionRojo).

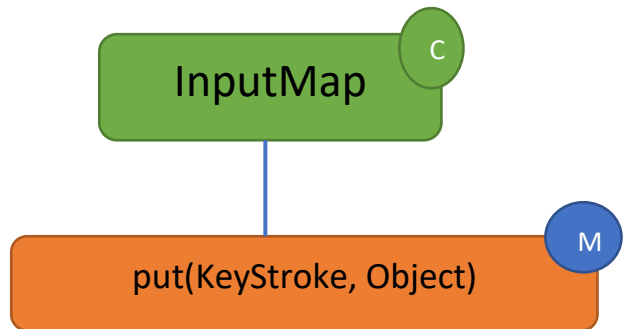


2.- Crear combinación de teclas.

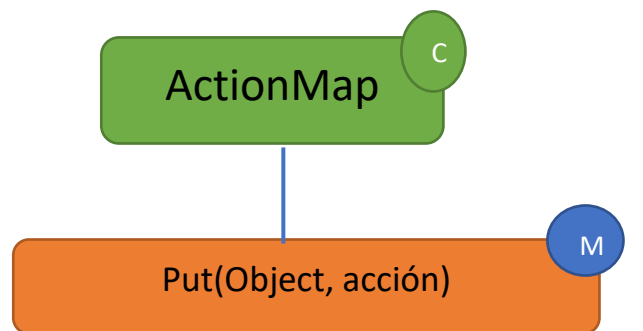
- Guarda descripción de teclas



3.- Asignar combinación de teclas a objetos.



4.- Asignar objetos a acciones.



```
26 class PanelAccion extends JPanel{
27     public PanelAccion() {
28         AccionColor accionAmarillo=new AccionColor("Amarillo", new ImageIcon("src/graficos/bolaAmarilla.png"), Color.YELLOW);
29         AccionColor accionAzul=new AccionColor("Azul", new ImageIcon("src/graficos/bolaAzul.png"), Color.BLUE);
30         AccionColor accionRojo=new AccionColor("Rojo", new ImageIcon("src/graficos/bolaRoja.png"), Color.RED);
31         JButton botonAmarillo=new JButton(accionAmarillo);
32         add(botonAmarillo); //Es una forma
33         add(new JButton(accionAzul));
34         add(new JButton(accionRojo));
35
36         InputMap mapaEntrada=getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
37
38         KeyStroke teclaAmarillo=KeyStroke.getKeyStroke("ctrl A");
39         mapaEntrada.put(teclaAmarillo, "fondo_amarillo");
40
41         KeyStroke teclaAzul=KeyStroke.getKeyStroke("ctrl B");
42         mapaEntrada.put(teclaAzul, "fondo_azul");
43
44         KeyStroke teclaRojo=KeyStroke.getKeyStroke("ctrl R");
45         mapaEntrada.put(teclaRojo, "fondo_rojo");
46
47         ActionMap mapaAccion=getActionMap();
48
49         mapaAccion.put("fondo_rojo", accionRojo);
50         mapaAccion.put("fondo_azul", accionAzul);
51         mapaAccion.put("fondo_amarillo", accionAmarillo);
52
53     }
```

```
InputMap
mapaEntrada=getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);

KeyStroke teclaAmarillo=KeyStroke.getKeyStroke("ctrl A");
mapaEntrada.put(teclaAmarillo, "fondo_amarillo");

KeyStroke teclaAzul=KeyStroke.getKeyStroke("ctrl B");
mapaEntrada.put(teclaAzul, "fondo_azul");

KeyStroke teclaRojo=KeyStroke.getKeyStroke("ctrl R");
mapaEntrada.put(teclaRojo, "fondo_rojo");

ActionMap mapaAccion=getActionMap();

mapaAccion.put("fondo_rojo", accionRojo);
mapaAccion.put("fondo_azul", accionAzul);
mapaAccion.put("fondo_amarillo", accionAmarillo);
```

Ahora si ejecutamos podremos utilizar las correspondientes combinaciones de teclas para cambiar de color.

Ahora vamos a modificar el código para poderlo simplificar más.

```
InputMap
mapaEntrada=getInputMap(JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);

mapaEntrada.put(KeyStroke.getKeyStroke("ctrl A"), "fondo_amarillo");

mapaEntrada.put(KeyStroke.getKeyStroke("ctrl B"), "fondo_azul");

mapaEntrada.put(KeyStroke.getKeyStroke("ctrl R"), "fondo_rojo");

ActionMap mapaAccion=getActionMap();

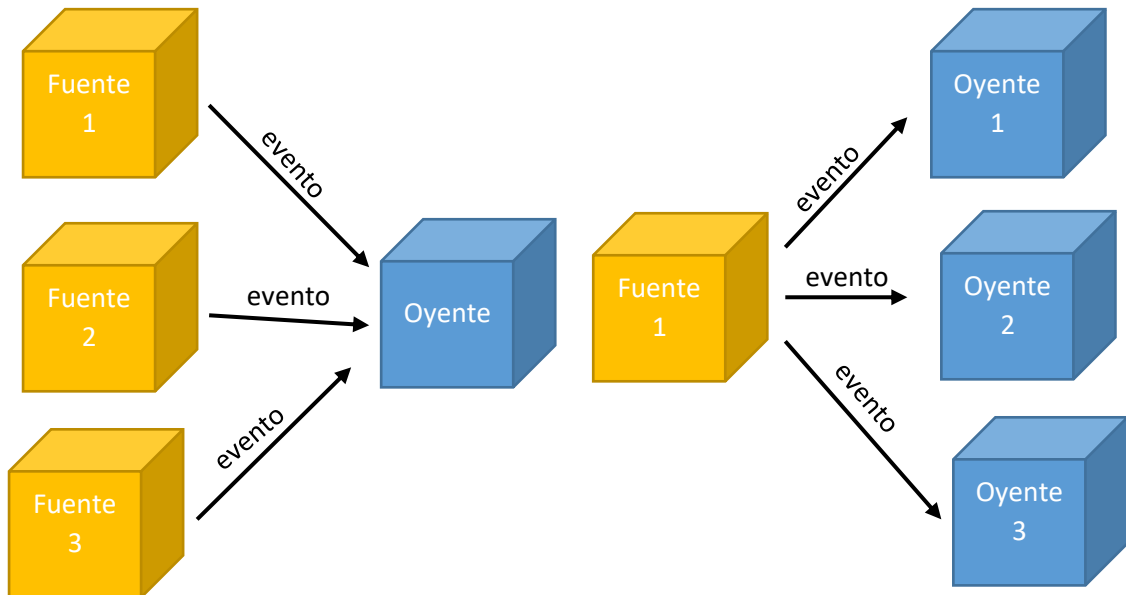
mapaAccion.put("fondo_rojo", accionRojo);
mapaAccion.put("fondo_azul", accionAzul);
mapaAccion.put("fondo_amarillo", accionAmarillo);
```





Eventos XVI. Múltiples oyentes. (Vídeo 80)

Múltiples fuentes vs Múltiples oyentes.



Vamos a crear una nueva clase llamada Varios_oyentes.

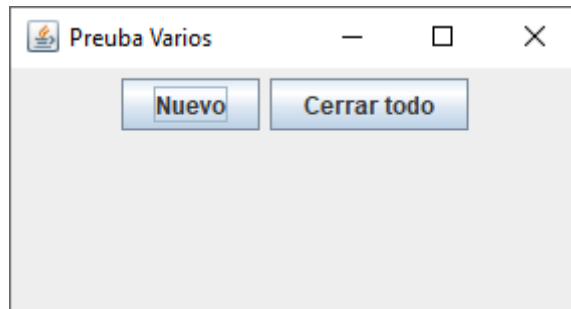
```
1 package graficos;
2
3 import javax.swing.*;
4 import java.awt.event.*;
5
6
7 public class Varios_oyentes {
8
9     public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         Marco_Principal mimarco=new Marco_Principal();
12         mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         mimarco.setVisible(true);
14     }
15 }
16 }
17
18 class Marco_Principal extends JFrame {
19     public Marco_Principal() {
20         setTitle("Preuba Varios");
21         setBounds(1300,100,300,200);
22         Lamina_Principal lamina=new Lamina_Principal();
23         add(lamina);
24     }
25 }
26 }
27 }
```

```

28 class Lamina_Principal extends JPanel{
29     public Lamina_Principal() {
30         JButton boton_nuevo=new JButton("Nuevo");
31         add(boton_nuevo);
32         boton_cerrar=new JButton("Cerrar todo");
33         add(boton_cerrar);
34     }
35     JButton boton_cerrar;
36 }

```

Este será el resultado:



El objeto botón_cerrar de tipo JButton se declara fuera de constructor para que esta pueda ser llamada desde el constructor y desde otros métodos luego se inicia dentro del constructor.

Ahora cada vez que pulsemos el botón Nuevo cree una ventana nueva.

```

package graficos;

import javax.swing.*;
import java.awt.event.*;

public class Varios_oyentes {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Marco_Principal mimarco=new Marco_Principal();
        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mimarco.setVisible(true);
    }
}

```

```

class Marco_Principal extends JFrame {
    public Marco_Principal() {
        setTitle("Preuba Varios");
        setBounds(1300,100,300,200);
        Lamina_Principal lamina=new Lamina_Principal();
        add(lamina);
    }
}

```

```

class Lamina_Principal extends JPanel{
    public Lamina_Principal() {
        JButton boton_nuevo=new JButton("Nuevo");
        add(boton_nuevo);
        boton_cerrar=new JButton("Cerrar todo");
        add(boton_cerrar);
        OyenteNuevo miOyente = new OyenteNuevo();
        boton_nuevo.addActionListener(miOyente);
    }
    private class OyenteNuevo implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            Marco_Emergente marco=new Marco_Emergente();
            marco.setVisible(true);
        }
    }
    JButton boton_cerrar;
}

```

Clase interna

```

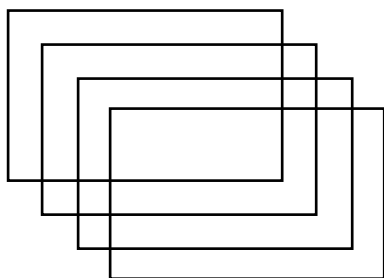
class Marco_Emergente extends JFrame{
    public Marco_Emergente() {
        contador++;
        setTitle("Ventana " + contador);

        setBounds(contador*40,contador*40,300,150);
    }
    private static int contador=0;
}

```

Con la clase Marco_Emergente iremos abriendo ventanas según pulsemos el botón de Nuevo con la peculiaridad que cada ventana irá numerada por un contador.

Con setBounds haremos que cada ventana nueva se abra modificando su posición horizontal y vertical en 40 píxeles.



Para evitar que una ventana tape a la siguiente.

La clase interna OyenteNuevo permite que la clase Lamina_Principal tenga acceso a dicha clase ya que esta es private y si estuviera fuera de dicha clase no tendría acceso a la clase OyenteNuevo.

Ahora nos interesa que al pulsar el botón “Cerrar todo” cierre todas las ventanas que son objetos diferentes, instancias diferentes, que hemos ido abriendo.

```
package graficos;

import javax.swing.*;
import java.awt.event.*;

public class Varios_oyentes {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Marco_Principal mimarco=new Marco_Principal();
        mimarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mimarco.setVisible(true);
    }
}

class Marco_Principal extends JFrame {
    public Marco_Principal() {
        setTitle("Preuba Varios");
        setBounds(1300,100,300,200);
        Lamina_Principal lamina=new Lamina_Principal();
        add(lamina);
    }
}

class Lamina_Principal extends JPanel{
    public Lamina_Principal() {
        JButton boton_nuevo=new JButton("Nuevo");
        add(boton_nuevo);
        boton_cerrar=new JButton("Cerrar todo");
        add(boton_cerrar);
        OyenteNuevo miOyente = new OyenteNuevo();
        boton_nuevo.addActionListener(miOyente);
    }
    private class OyenteNuevo implements ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            Marco_Emergente marco=new Marco_Emergente(boton_cerrar);
            marco.setVisible(true);
        }
    }
    JButton boton_cerrar;
}
}
```



```

class Marco_Emergente extends JFrame{
    public Marco_Emergente(JButton boton_de_principal) {
        contador++;
        setTitle("Ventana " + contador);

        setBounds(contador*40, contador*40, 300, 150);
        CierraTodos oyenteCerrar=new CierraTodos();
        boton_de_principal.addActionListener(oyenteCerrar);
    }

    private class CierraTodos implements ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            dispose();
        }
    }

    private static int contador=0;
}

```

Cierra todas las ventanas emergentes.

Dentro de la clase Marco_Emergente haremos una clase interna llamada CierraTodos.

Desde la clase Marco_Emergente tendremos acceso a la clase CierraTodos.



Contenido

| | |
|---|-----|
| Interfaces y clases internas. Interfaces I. (Vídeo 49)..... | 1 |
| Interfaces y clases internas. Interfaces II (Vídeo 50)..... | 6 |
| Interfaces y clases internas. Interfaces III (Vídeo 51)..... | 11 |
| Interfaces y clases internas. Interfaces IV (Vídeo 52) | 16 |
| Interfaces y clases internas. Clases internas I. (Vídeo 53) | 18 |
| Interfaces y clases internas. Clases internas II. (Vídeo 54) | 21 |
| Aplicaciones gráficas Swing I. (Vídeo 55) | 25 |
| Aplicaciones gráficas. Swing II. Colocando el Frame. (Vídeo 56) | 29 |
| Aplicaciones gráficas. Swing III. Colocando el Frame II. (Vídeo 57) | 34 |
| Aplicaciones gráficas Swing IV. Escribiendo en el Frame. (Vídeo 58) | 38 |
| Aplicaciones gráficas. Swing V. Dibujando en el Frame. (Vídeo 59) | 41 |
| Aplicaciones gráficas. Swing VI. Dibujando en el Frame II (Vídeo 60) | 43 |
| Aplicaciones gráficas. Swing VII. Manejando colores (Vídeo 61)..... | 48 |
| Aplicaciones gráficas. Swing VIII. Cambiando la letra en el Frame. (Vídeo 62) | 53 |
| Aplicaciones gráficas. Swing IX. Incluyendo imágenes. (Vídeo 63) | 60 |
| Aplicaciones gráficas. Swing X Incluyendo imágenes II (Vídeo 64) | 63 |
| Eventos 1. (Vídeo 65) | 69 |
| Eventos II. (Vídeo 66) | 73 |
| Eventos III (Vídeo 67) | 76 |
| Eventos IV. Eventos de ventana I. (Vídeo 68) | 79 |
| Eventos V. Eventos de ventana II. Clases adaptadas. (Vídeo 69)..... | 83 |
| Eventos VI. Eventos de ventana III. Controlando estado de ventana. (Vídeo 70) | 86 |
| Eventos VII. Eventos de teclado I. (Vídeo 71)..... | 92 |
| Eventos VIII. Eventos de ratón I. (Vídeo 72)..... | 97 |
| Eventos IX. Eventos de ratón II. (Vídeo 73)..... | 101 |
| Eventos X. Eventos de foco. (Vídeo 74)..... | 105 |
| Eventos XI. Eventos de foco II. (Vídeo 75)..... | 109 |
| Eventos XII. Múltiples fuentes I. (Vídeo 76) | 113 |
| Eventos XIII. Múltiples fuentes II. (Vídeo 77) | 116 |
| Eventos XIV. Múltiples fuentes III. (Vídeo 78)..... | 118 |
| Eventos XV. Múltiples fuentes IV. (Vídeo 79)..... | 121 |
| Eventos XVI. Múltiples oyentes. (Vídeo 80) | 125 |