



SESIÓN 08:

GRASP parte 2: - Bajo acoplamiento. - Alta cohesión. - Controlador



Sumario

- 1 Introducción
- 2 Definiciones del MVC
- 3 Funcionamiento de capas del MVC
- 4 Diagrama del MVC en JAVA EE
- 5 Ejemplos del MVC

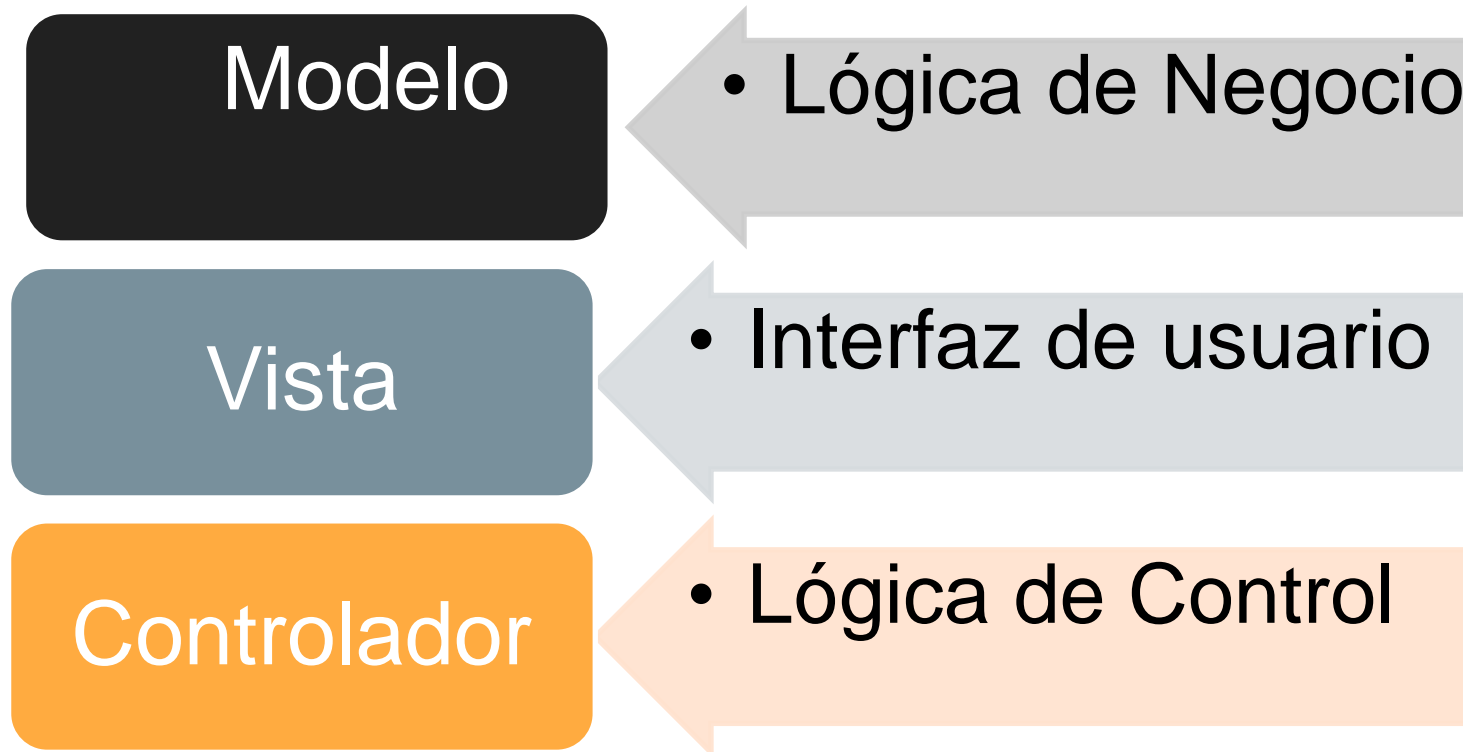


Introducción

- ❑ El **modelo–vista–controlador (MVC)** fue desarrollado en el Centro de Investigaciones Xerox Corporation a finales de los años setenta en California por Trygve Reenskaug.
- ❑ La arquitectura del patrón Modelo-Vista-Controlador es un paradigma de programación bien conocido para el desarrollo de aplicaciones con interfaz gráfica de usuario (GUI).
- ❑ El MVC es un patrón de software que separa los componentes de aplicación en tres niveles por sus diferentes responsabilidades.



Relación del MVC





¿Por qué utilizar MVC?

- ❖ La razón es que nos permite separar los componentes de nuestra aplicación dependiendo de la responsabilidad que tienen, esto significa que cuando hacemos un cambio en alguna parte de nuestro código, esto no afecte otra parte del mismo.
- ❖ Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la reutilización del código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

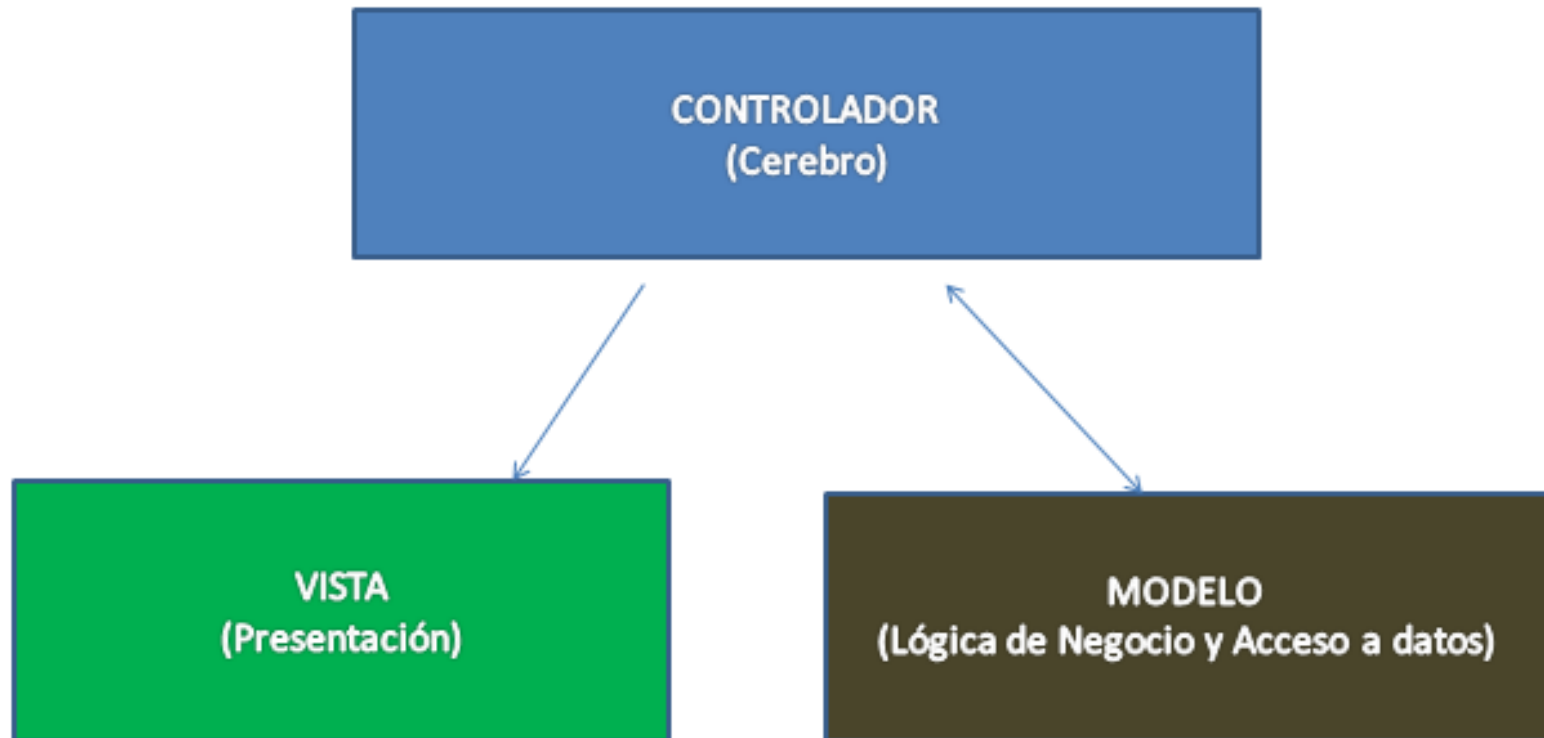


Ejemplo

- ❖ Si modificamos nuestra Base de Datos, sólo deberíamos modificar el modelo que es **quién se encarga de los datos** y el resto de la aplicación debería permanecer intacta.
- ❖ Esto respeta el principio de la ***responsabilidad única***. Es decir, una parte de tu código no debe de saber qué es lo que hace toda la aplicación, sólo debe de tener una responsabilidad.



Diagrama de MVC





El Modelo

- El modelo es la porción que implementa la “**Lógica del Negocio**”.
- Se le suele llamar a la parte del sistema que representa objetos y sus interacciones del mundo real.
- Son rutinas que realizan entradas de datos, consultas, generación de informes y más específicamente todo el procesamiento que se realiza detrás de la aplicación.
- Las peticiones de acceso o manipulación de información llegan al 'Modelo' a través del 'controlador', y este envía a la 'vista' aquella información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario).



El Modelo (cont.)

- **Es la capa donde se trabaja con los datos**, por tanto contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes *selects*, *updates*, *inserts*, etc.





El controlador

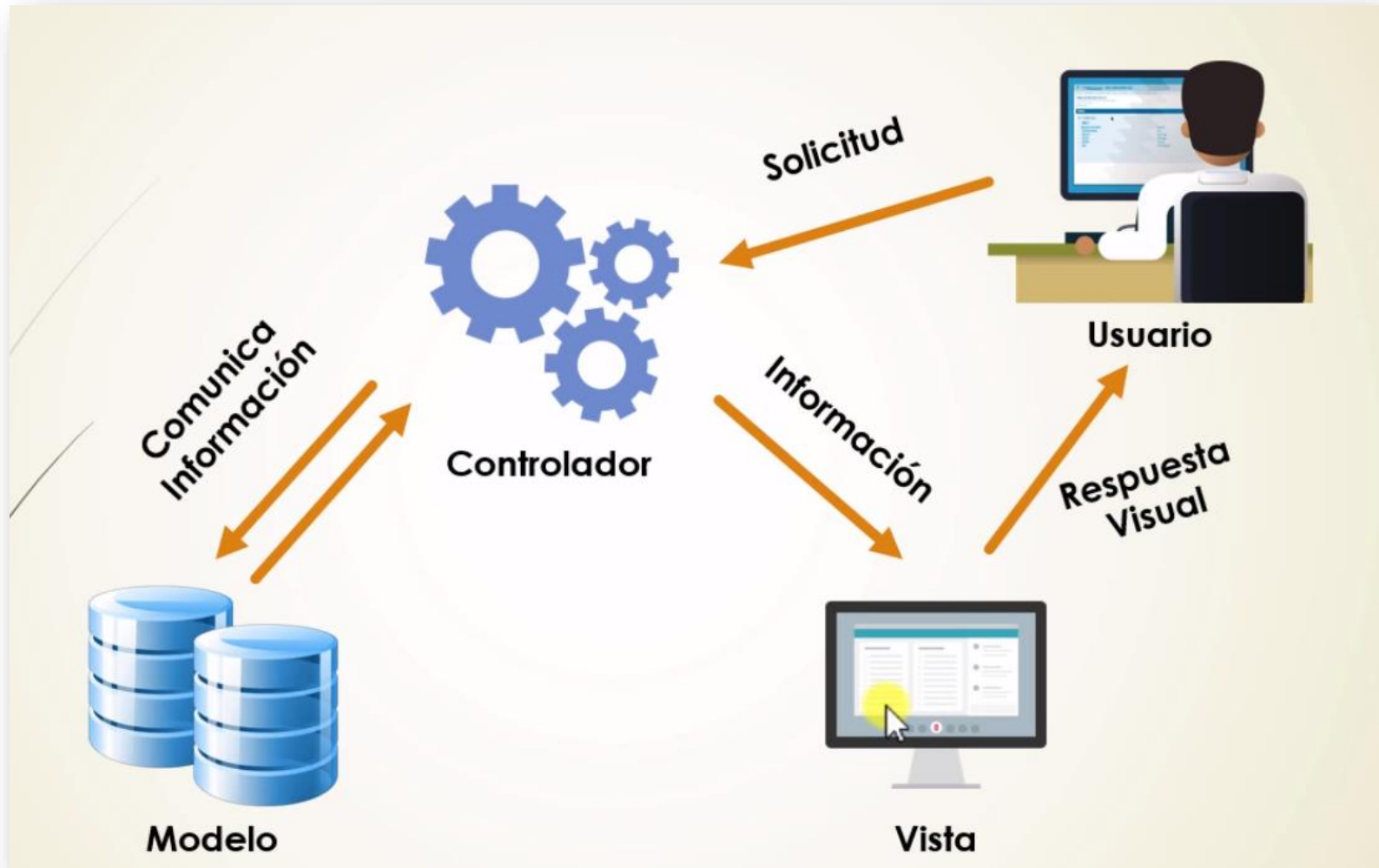
- El controlador es el cerebro de la aplicación MVC.
- Responde a eventos (acciones del usuario) e invoca peticiones al “modelo” cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su “vista” asociada si se solicita.
- Empleado como un mediador entre el medio gráfico ("View") y el modelo ("Model"), coordina las acciones que son llevadas a cabo entre ambos.
- El controlador generalmente crea instancias y utiliza métodos de esos modelos para conseguir los datos que se presentan a los usuarios, enviándolos a la vista correspondiente.



La vista

- Las vistas son las porciones de la aplicación MVC que presentan salida al usuario.
- Presenta el “modelo” (información y lógica de negocio) en un formato adecuado para interactuar (interfaz de usuario).
- Ni el modelo ni el controlador se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la vista.
- Por ejemplo: La salida más común para aplicaciones web es el HTML. Podrían ser otras como un formulario, gráficos, etc.

Analogía del MVC





Flujo de control

1. El usuario realiza una acción en la interfaz.
2. El controlador trata el evento de entrada.
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta).
4. Se genera una nueva vista. La vista toma los datos del modelo.
 - El modelo no tiene conocimiento directo de la vista
5. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.



Funcionamiento en la Web

En la web, el MVC funcionaría así.

- ❑ Cuando el usuario manda una petición al navegador, digamos quiere ver un sitio específico el **controlador** responde a la solicitud, porque él es el que controla la lógica de la app, luego le pide al modelo la información del curso.
- ❑ **El modelo**, que se encarga de los datos de la app, consulta la base de datos y obtiene toda la información.



Funcionamiento en la Web

- ❑ Luego, el modelo responde al controlador con los datos que pidió.
- ❑ Finalmente, el controlador tiene los datos solicitados, se los manda a la vista, pudiendo aplicar los estilos (Hojas de estilos CSS), organizar la información y construir la página que se observa en el navegador.



Ventajas del MVC

- ☐ Fácil organización del código en tres componentes diferentes.
- ☐ Crea independencia del funcionamiento.
- ☐ Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de otras capas.
- ☐ Si trabaja con un equipo de programadores entonces les da una mayor facilidad para poder seguir el trabajo entre varios integrantes.
- ☐ Facilita el mantenimiento en caso de errores.
- ☐ Hacen que las aplicaciones sean fácilmente extensibles.
- ☐ Poder adaptarse a los frameworks de hoy en día.



Desventajas del MVC

- ❖ La separación de conceptos en capas agrega complejidad al sistema.
- ❖ La cantidad de archivos a mantener y desarrollar se incrementa considerablemente.
- ❖ La curva de aprendizaje del patrón de diseño es más alta que usando otros modelos sencillos.

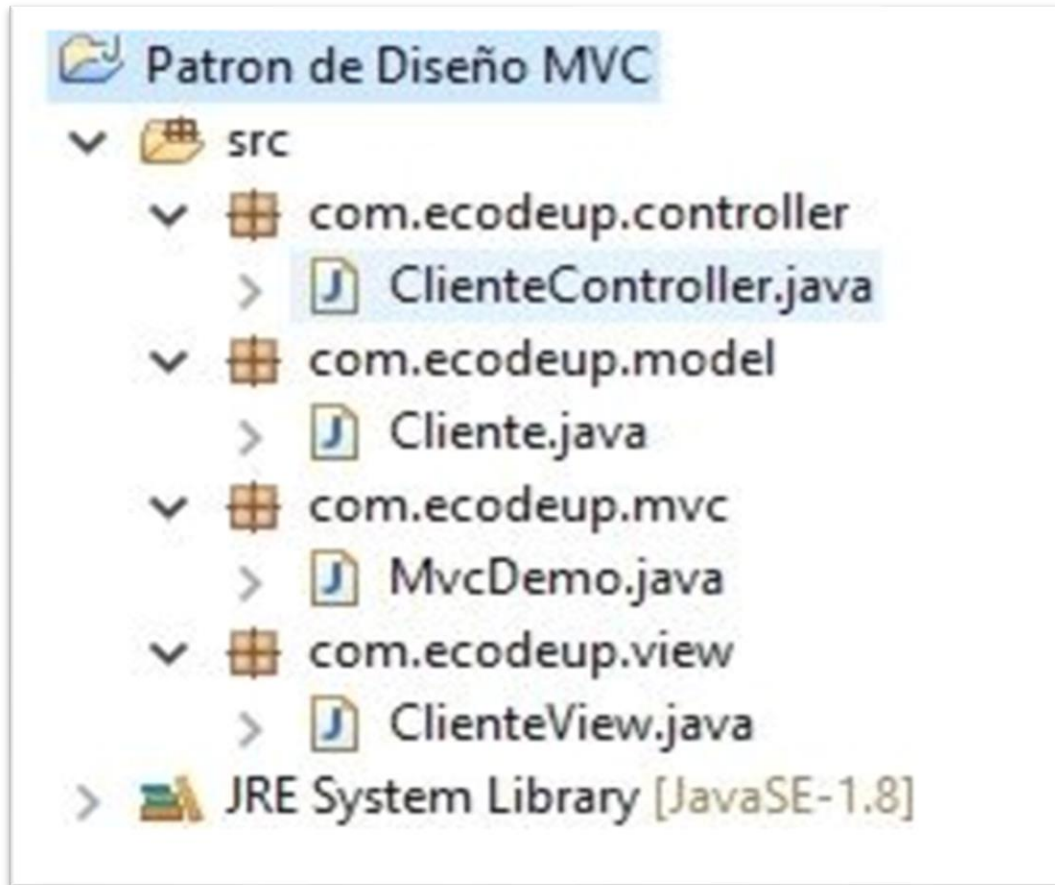


EJEMPLOS DEL MVC





Ejemplo 1: Implementación del MVC en Java





Ejercicio

- Para aplicar el MVC a nuestro ejemplo tendremos que desarrollar una serie de módulos independientes que se encarguen, en dos capas (acción y estado) del acceso a los datos.

Paso 1:Modelo

Primero se crea el modelo, que es una clase en java y se llama ***Cliente.java***, esta clase sólo contiene los atributos, constructor, getters y setters.

A screenshot of a Java IDE window titled 'Modelo'. The window shows the code for a Java class named 'Cliente'. The code includes three private attributes: 'id' (int), 'nombre' (String), and 'apellido' (String). It also includes a no-argument constructor, and getter and setter methods for each attribute. The code is as follows:

```
Modelo
2
3 public class Cliente {
4     private int id;
5     private String nombre;
6     private String apellido;
7
8
9     public Cliente() {
10    }
11
12    public int getId() {
13        return id;
14    }
15    public void setId(int id) {
16        this.id = id;
17    }
18    public String getNombre() {
19        return nombre;
20    }
21    public void setNombre(String nombre) {
22        this.nombre = nombre;
23    }
24    public String getApellido() {
25        return apellido;
26    }
27    public void setApellido(String apellido) {
28        this.apellido = apellido;
29    }
30 }
```

Paso 1: Modelo

Se definen los campos ("fields") utilizados en un contexto privado (private).

Se definen constructores Java, Constructor "default" sin datos de entrada.

A través de los diversos métodos get/set es posible modificar los valores iniciales definidos en el Java Bean.



Paso 2: Vista

Luego se crea la vista, la clase ***ClienteView.java***, para el ejemplo y su función es presentar los datos del modelo.

Luego creas la vista, la clase ***ClienteView.java***, que es un clase que va hacer de vista para el ejemplo y su función es presentar los datos del modelo.

```
1 package com.ecodeup.view;
2
3 public class ClienteView {
4     public void imprimirDatosCliente(int id,String nombre, String apellido) {
5         System.out.println("**** DATOS CLIENTE ****");
6         System.out.println("Id: "+id);
7         System.out.println("Nombre: "+nombre);
8         System.out.println("Apellido: "+apellido);
9     }
10 }
```

Esta clase lo único que va hacer es imprimir los datos del modelo que es la clase Cliente.java.

Ahora creas el controlador, el controlador **contiene 2 objetos el modelo**, la vista así como los getters y setters para llenar las propiedades del modelo y un método(actualizarVista()) que llama a la vista que a su vez imprime las propiedades del modelo cliente.

```
1 package com.ecodeup.controller;
2
3 import com.ecodeup.model.Cliente;
4 import com.ecodeup.view.ClienteView;
5
```



Paso 3: Controlador

- Ahora se crea el controlador que contiene 2 objetos: **el modelo y la vista.**
- Así como los getters y setters para llenar las propiedades del modelo y un método(actualizarVista()) que llama a la vista que a su vez imprime las propiedades del modelo cliente.

Paso 3: Controlador

```
1 package com.ecodeup.controller;
2
3 import com.ecodeup.model.Cliente;
4 import com.ecodeup.view.ClienteView;
5
6 public class ClienteController {
7     //objetos vista y modelo
8     private ClienteView vista;
9     private Cliente modelo;
10
11     //constructor para inicializar el modelo y la vista
12     public ClienteController(Cliente modelo, ClienteView vista) {
13         this.modelo = modelo;
14         this.vista = vista;
15     }
16
17     //getters y setters para el modelo
18     public int getId() {
19         return modelo.getId();
20     }
21     public void setId(int id) {
22         this.modelo.setId(id);
23     }
24     public String getNombre() {
25         return modelo.getNombre();
26     }
27     public void setNombre(String nombre) {
28         this.modelo.setNombre(nombre);
29     }
30     public String getApellido() {
31         return modelo.getApellido();
32     }
33     public void setApellido(String apellido) {
34         this.modelo.setApellido(apellido);
35     }
36
37     //pasa el modelo a la vista para presentar los datos
38     public void actualizarVista() {
39         vista.imprimirDatosCliente(modelo.getId(), modelo.getNombre(), modelo.getApellido());
40     }
41 }
```



Paso 3: Controlador

```
1 package com.ecodeup.controller;
2
3 import com.ecodeup.model.Cliente;
4 import com.ecodeup.view.ClienteView;
5
6 public class ClienteController {
7     //objetos vista y modelo
8     private ClienteView vista;
9     private Cliente modelo;
10
11     //constructor para inicializar el modelo y la vista
12     public ClienteController(Cliente modelo, ClienteView vista) {
13         this.modelo = modelo;
14         this.vista = vista;
15     }
16
17     //getters y setters para el modelo
18     public int getId() {
19         return modelo.getId();
20     }
21     public void setId(int id) {
22         this.modelo.setId(id);
23     }
24     public String getNombre() {
25         return modelo.getNombre();
26     }
27     public void setNombre(String nombre) {
28         this.modelo.setNombre(nombre);
29     }
30     public String getApellido() {
31         return modelo.getApellido();
32     }
33     public void setApellido(String apellido) {
34         this.modelo.setApellido(apellido);
35     }
36
37     //pasa el modelo a la vista para presentar los datos
38     public void actualizarVista() {
39         vista.imprimirDatosCliente(modelo.getId(), modelo.getNombre(), modelo.getApellido());
40     }
41 }
```

Se declara los objetos

Finalmente queda hacer un test para comprobar el patrón de diseño Modelo Vista Controlador funciona:



Paso 3: Controlador

```
1 package com.ecodeup.controller;
2
3 import com.ecodeup.model.Cliente;
4 import com.ecodeup.view.ClienteView;
5
6 public class ClienteController {
7     //objetos vista y modelo
8     private ClienteView vista;
9     private Cliente modelo;
10
11     //constructor para inicializar el modelo y la vista
12     public ClienteController(Cliente modelo, ClienteView vista) {
13         this.modelo = modelo;
14         this.vista = vista;
15     }
16
17     //getters y setters para el modelo
18     public int getId() {
19         return modelo.getId();
20     }
21     public void setId(int id) {
22         this.modelo.setId(id);
23     }
24     public String getNombre() {
25         return modelo.getNombre();
26     }
27     public void setNombre(String nombre) {
28         this.modelo.setNombre(nombre);
29     }
30     public String getApellido() {
31         return modelo.getApellido();
32     }
33     public void setApellido(String apellido) {
34         this.modelo.setApellido(apellido);
35     }
36
37     //pasa el modelo a la vista para presentar los datos
38     public void actualizarVista() {
39         vista.imprimirDatosCliente(modelo.getId(), modelo.getNombre(), modelo.getApellido());
40     }
41 }
```

Presentación
de los datos

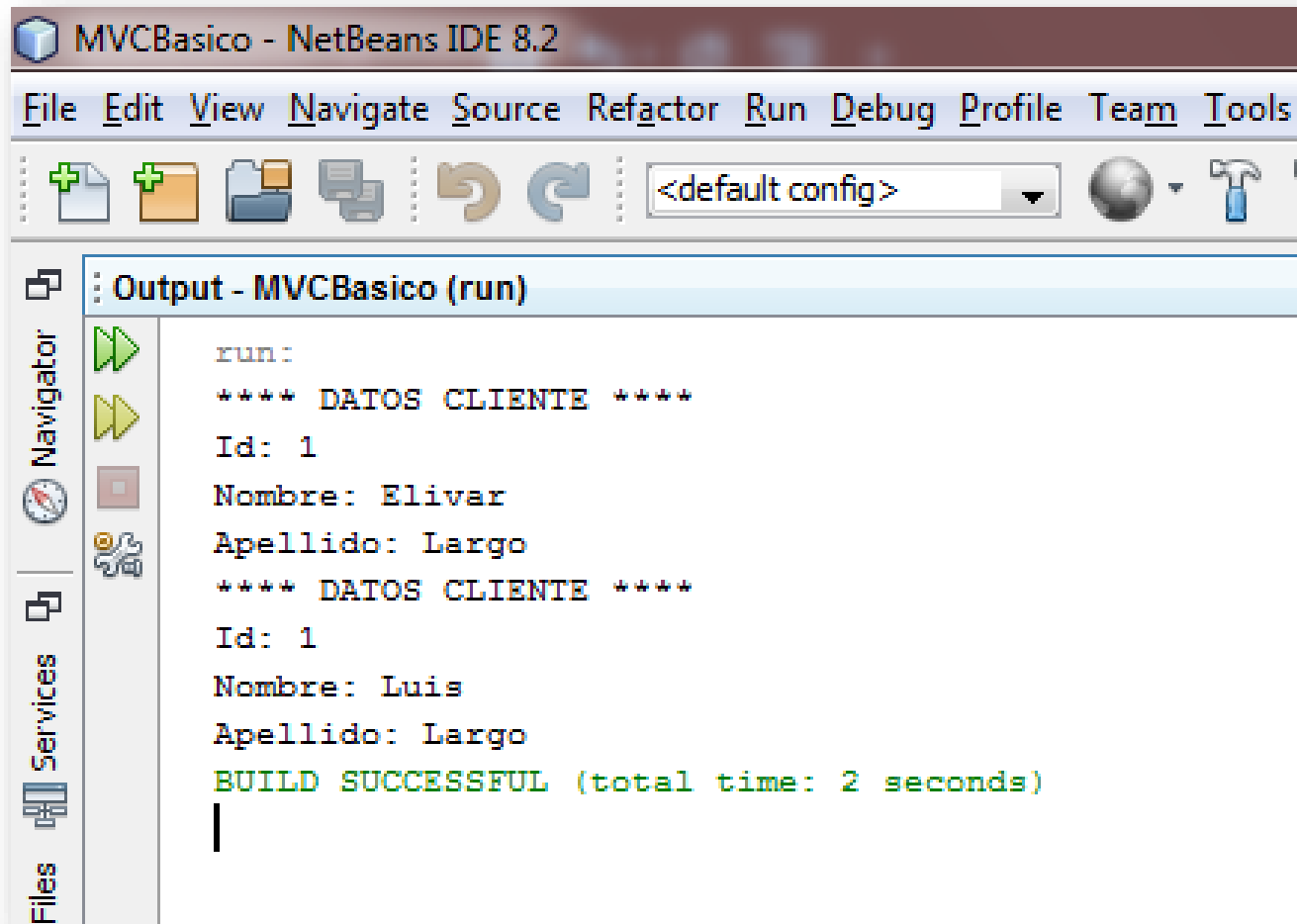
Paso 4: Test MVC

Comprobar patrón MVC

```
2
3 import com.ecodeup.controller.ClienteController;
4 import com.ecodeup.model.Cliente;
5 import com.ecodeup.view.ClienteView;
6
7 public class MvcDemo {
8
9     public static void main (String [] args){
10         // objeto vista, y modelo creado con el método estático
11         Cliente modelo= llenarDatosCliente();
12         ClienteView vista= new ClienteView();
13
14         //se crea un objeto controlador y se le pasa el modelo y la vista
15         ClienteController controlador= new ClienteController(modelo, vista);
16
17         // se muestra los datos del cliente
18         controlador.actualizarVista();
19
20         // se actualiza un cliente y se muestra de nuevo los datos
21         controlador.setNombre("Luis");
22         controlador.actualizarVista();
23     }
24     //método estático que retorna el cliente con sus datos
25     private static Cliente llenarDatosCliente() {
26         Cliente cliente = new Cliente();
27         cliente.setId(1);
28         cliente.setNombre("Elivar");
29         cliente.setApellido("Largo");
30         return cliente;
31     }
32 }
```



Resultado en consola



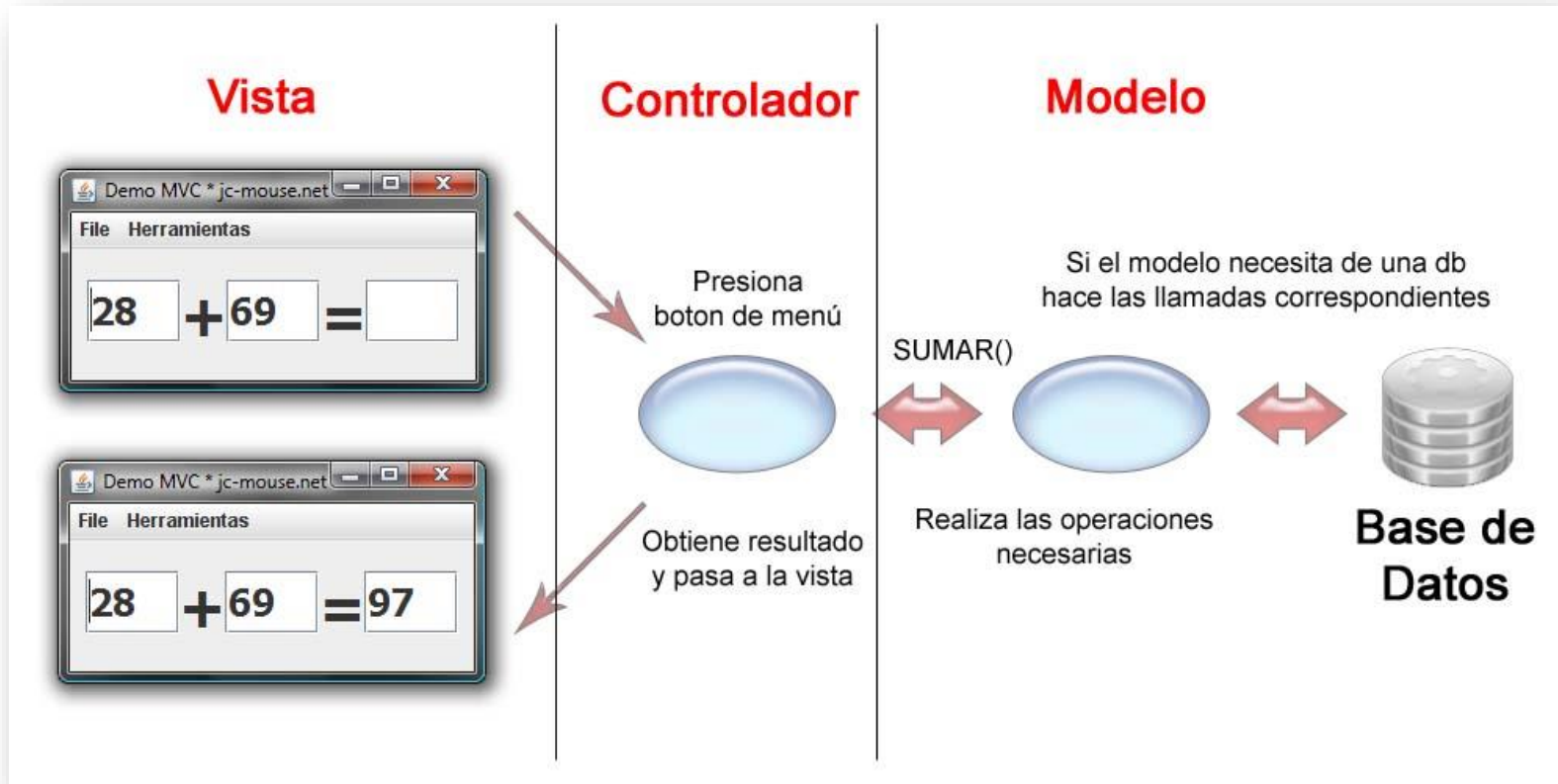
The screenshot shows the NetBeans IDE 8.2 interface. The title bar reads 'MVCBasico - NetBeans IDE 8.2'. The menu bar includes 'File', 'Edit', 'View', 'Navigate', 'Source', 'Refactor', 'Run', 'Debug', 'Profile', 'Team', and 'Tools'. The toolbar contains icons for file operations and a configuration dropdown set to '<default config>'. On the left, the 'Navigator' and 'Services' tabs are visible. The main console window, titled 'Output - MVCBasico (run)', displays the following text:

```
run:
**** DATOS CLIENTE ****
Id: 1
Nombre: Elivar
Apellido: Largo
**** DATOS CLIENTE ****
Id: 1
Nombre: Luis
Apellido: Largo
BUILD SUCCESSFUL (total time: 2 seconds)
|
```



Ejemplo 2: Calculadora con Java Swing

- Una clase sencilla para sumar variables





Ejemplo 2:

Calculadora con Java Swing

Crea un nuevo proyecto en netbeans, para este ejemplo, el proyecto se llama "MVC".

Crea una estructura de paquetes (Controller, Model, View), hacemos esto para separar cada componente y ser más organizados.

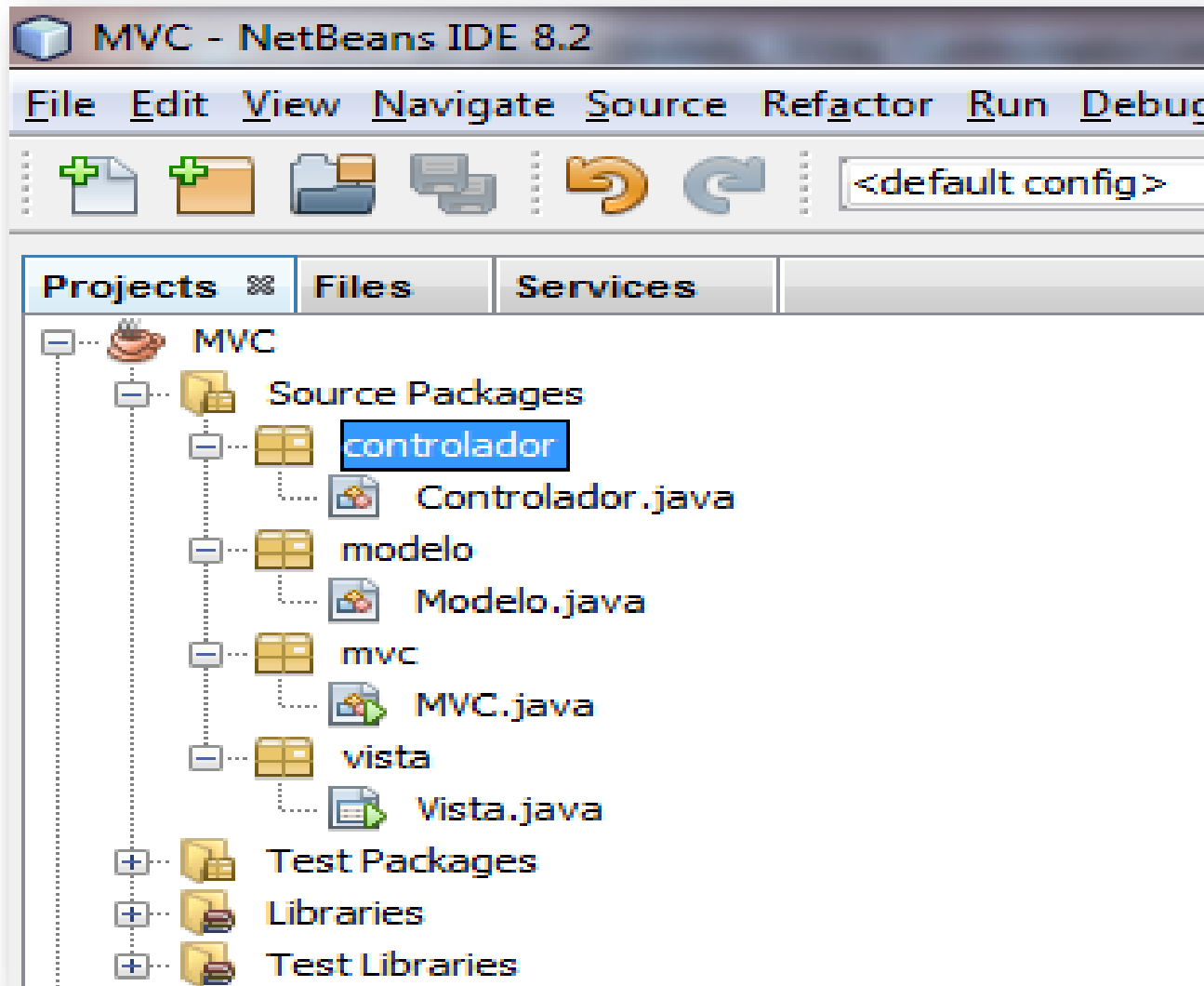


Pasos

1. Creación del proyecto
2. Creación del modelo
3. Codificación de la clase Vista
4. Codificación de la clase Controlador
5. Codificación de la clase Principal
6. Prueba del proyecto MVC



Paso 1: Creación del proyecto





Paso 1:

Creación del proyecto

- Como puedes observar, mantenemos el paquete default junto al **MVC.java** que nos crea netbeans, este main es el que nos servirá como nuestro index de nuestra aplicación, nuestro "lanzador".



Paso 2: El modelo

Empecemos creando la lógica de la aplicación, crea una nueva clase en el paquete Model, llámalo "**modelo.java**" y añade el siguiente código:



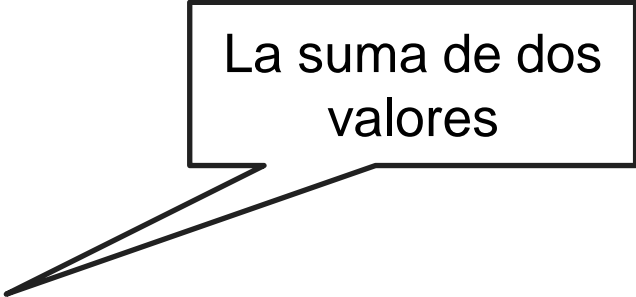
Paso 2: El modelo

```
package modelo;

public class Modelo

    private int numeroUno;
    private int numeroDos;
    private int resultado;

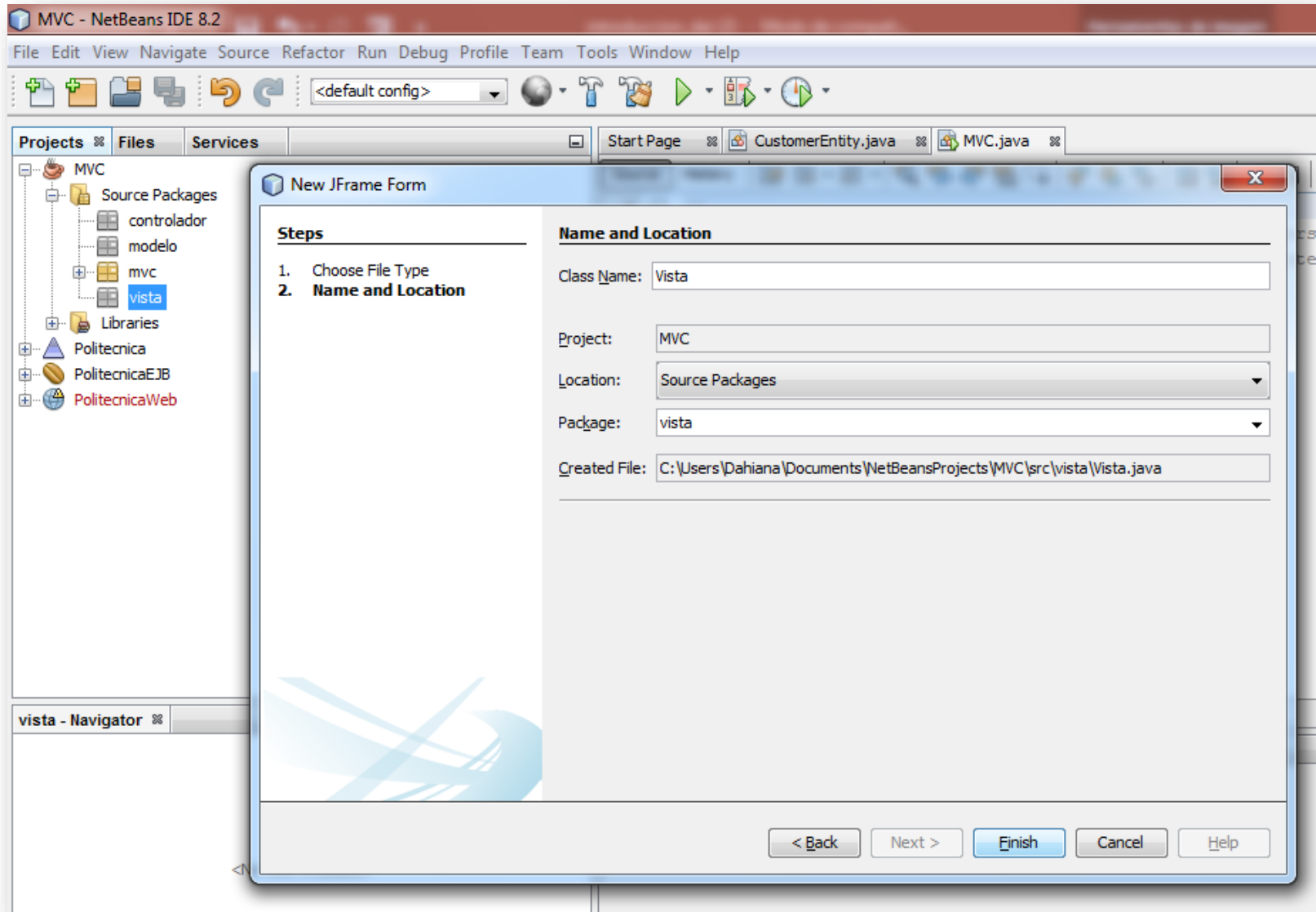
    public int getNumeroUno() {
        return numeroUno;
    }
    public void setNumeroUno(int numeroUno) {
        this.numeroUno = numeroUno;
    }
    public int getNumeroDos() {
        return numeroDos;
    }
    public void setNumeroDos(int numeroDos) {
        this.numeroDos = numeroDos;
    }
    public int getResultado() {
        return resultado;
    }
    public void setResultado(int resultado) {
        this.resultado = resultado;
    }
    public int sumar()
    {
        this.resultado= this.numeroUno+this.numeroDos;
        return this.resultado;
    }
}
```



La suma de dos valores

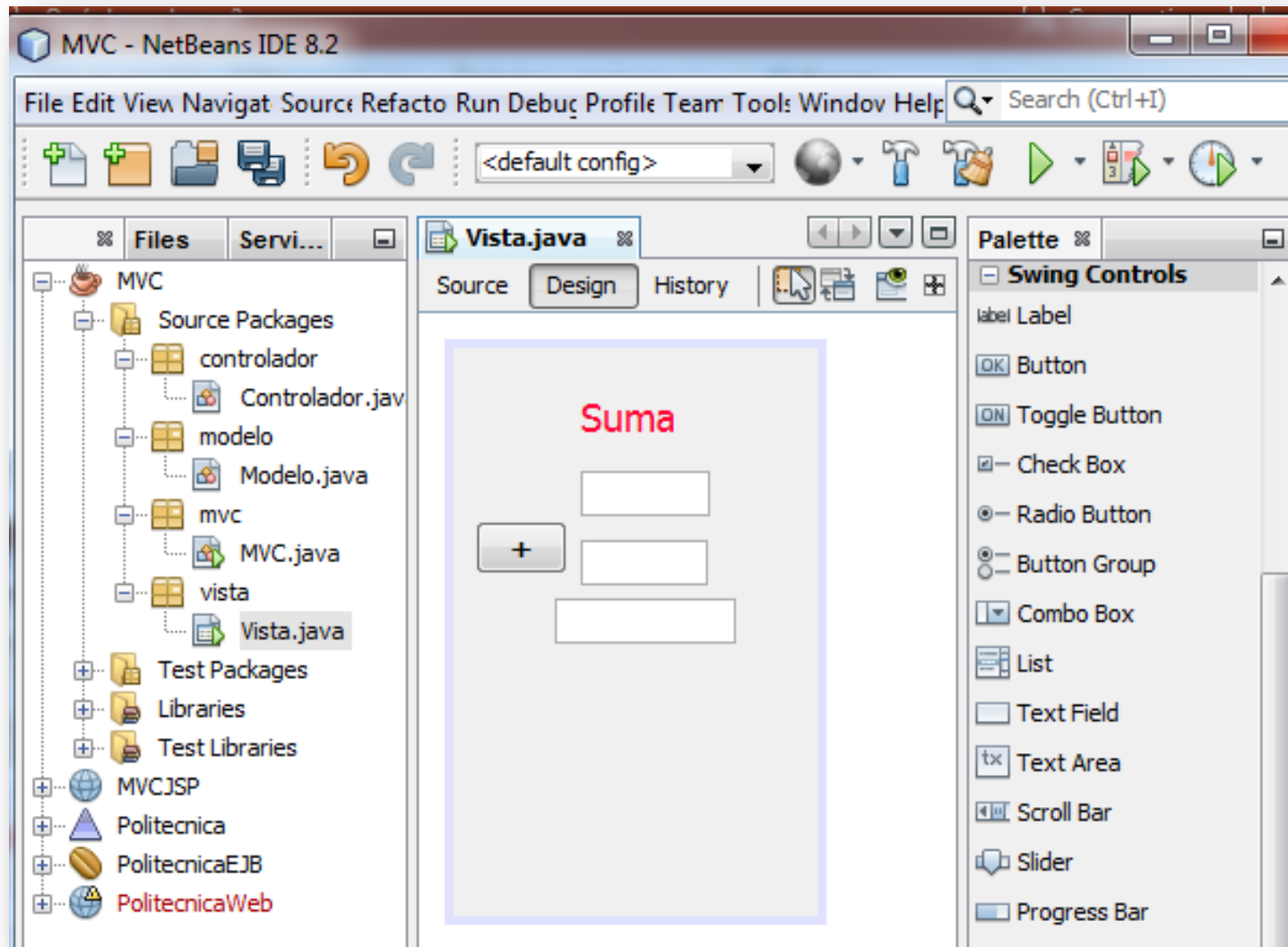


Paso 3: La vista



Paso 3: La vista

Se añade un JFrame al paquete VIEW, llámalo “Vista.java”.





Paso 3: Vista

- En Java existe una biblioteca gráfica (**Componentes Swing**) la cual incluye widgets para la interfaz gráfica de usuario (cajas de texto, botones, menús entre otros...).
- Para esta "MiniCalculadora« haremos uso **JTextField** (campos de texto) para los operando y uno para mostrar el resultado, un **JButtons** (botón) para la operación , a su vez algunos **JLabels** para mostrar ciertos textos en la ventana.



Paso 4: Controlador

```
Controlador.java
source History
package controlador;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import modelo.Modelo;
import vista.Vista;

public class Controlador implements ActionListener
{
    private Vista view;
    private Modelo model;

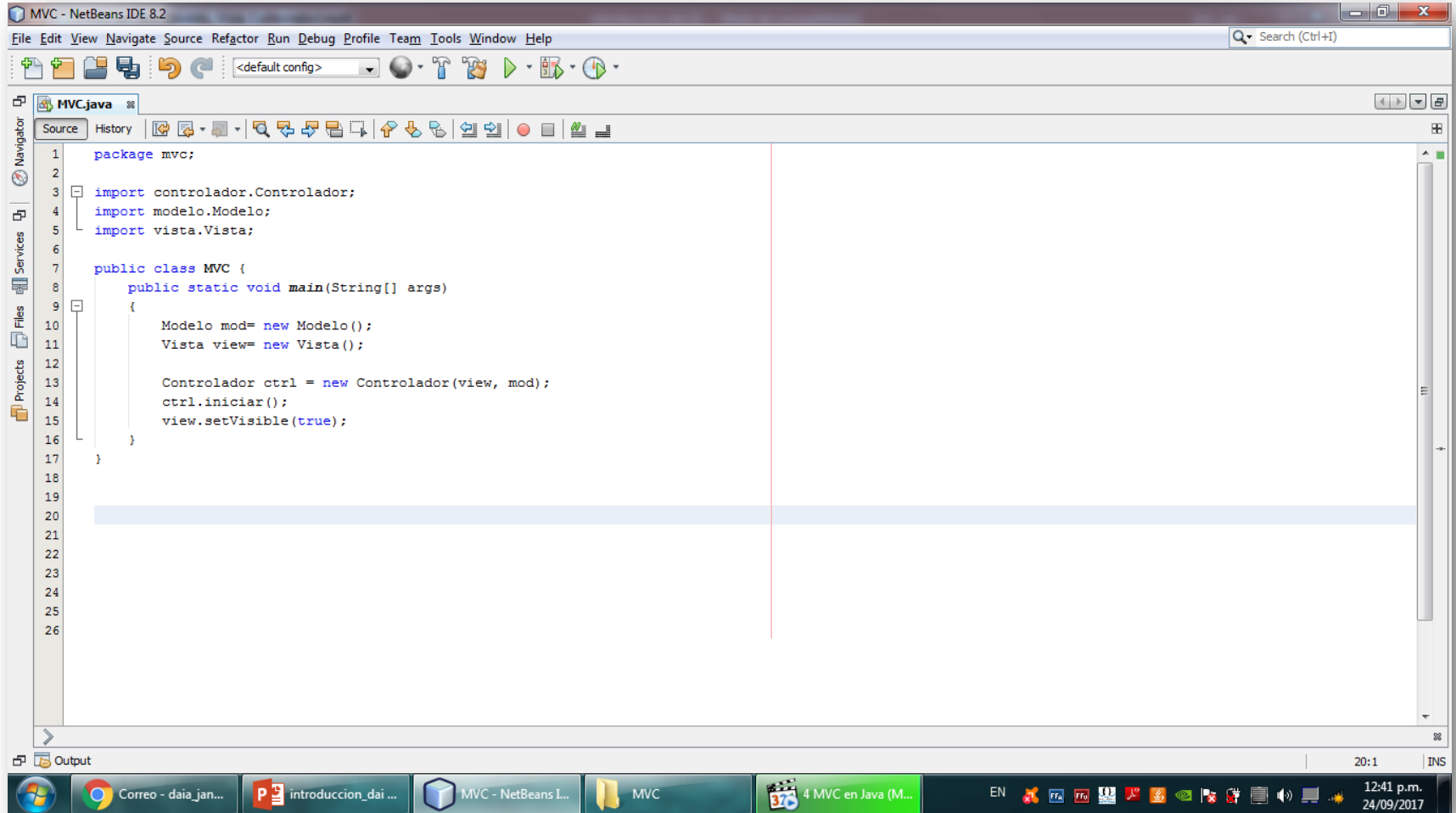
    public Controlador(Vista view, Modelo model){
        this.view= view;
        this.model=model;
        this.view.btnSumar.addActionListener(this);
    }

    public void iniciar(){
        view.setTitle("MVC Sumar");
        view.setLocationRelativeTo(null);
    }

    public void actionPerformed(ActionEvent e)
    {
        model.setNumeroUno(Integer.parseInt(view.txtNumeroUno.getText()));
        model.setNumeroDos(Integer.parseInt(view.txtNumeroDos.getText()));
        model.sumar();
        view.txtResultado.setText(String.valueOf(model.getResultado()));
    }
}
```




Paso 5: Clase Principal



```
1 package mvc;
2
3 import controlador.Controlador;
4 import modelo.Modelo;
5 import vista.Vista;
6
7 public class MVC {
8     public static void main(String[] args)
9     {
10         Modelo mod= new Modelo();
11         Vista view= new Vista();
12
13         Controlador ctrl = new Controlador(view, mod);
14         ctrl.iniciar();
15         view.setVisible(true);
16     }
17 }
18
19
20
21
22
23
24
25
26
```

Output

20:1 | INS

Correo - daia_jan... | introduccion_dai ... | MVC - NetBeans I... | MVC | 4 MVC en Java (M... | EN | 12:41 p.m. 24/09/2017



Paso 6: Prueba

The screenshot displays the NetBeans IDE 8.2 interface. The top menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. The toolbar shows various icons for file operations and development tools. The left sidebar contains the Projects, Files, and Services tabs. The Projects tab shows the MVC project structure, including Source Packages (controlador, modelo, mvc, vista), Test Packages, Libraries, and Test Libraries. The Files tab shows the MVC.java file. The Source tab displays the MVC.java source code, which includes package declarations, imports, and a public class definition. A dialog box titled 'MVC Sumar' is overlaid on the source code, showing a simple addition interface with two input fields (10 and 5), a '+' button, and an output field displaying the result 15. The bottom status bar shows the MVC (run) tab, indicating the application is running. The system tray at the bottom includes icons for Google, XAMPP, Explorer, and other applications, along with the date and time (04:06 p.m. 28/09/2017).

NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Files Services

MVC

- Source Packages
 - controlador
 - modelo
 - mvc
 - MVC.java
 - vista
 - Vista.java
- Test Packages
- Libraries
- Test Libraries
- MVCJSP
- Politecnica
- PolitecnicaEJB
- PolitecnicaWeb

Navigator

Members

MVC

- main(String[] args)

Source

```
1 package mvc;
2
3 import
4 import
5 import
6
7 public
8 public
9 {
10
11
12
13
14
15
16
17
18
19
20
```

MVC Sumar

Suma

10

5

+

15

view, mod);

Output

Java DB Database Process GlassFish Server MVC (run)

run:

MVC (run) running...

1:1 INS

ES 04:06 p.m. 28/09/2017



Resultado

Interfaz 1

A screenshot of a Windows application window titled "MVC Sumar". The window has a light gray background. At the top, the word "Suma" is written in red. Below it, there are three input fields. The first field contains the number "10" in red, and the second field contains the number "5" in red. To the left of these fields is a blue button with a white "+" sign. Below the two input fields is a third input field containing the number "15" in red.

Interfaz 2

A screenshot of a Windows application window titled "MVC Sumar". The window has a light gray background. At the top, the word "Suma" is written in red. Below it, there is a horizontal layout of elements: an input field containing "10", followed by a "+" sign, another input field containing "5", a blue button labeled "Sumar", an equals sign "=", and a final input field containing "15". All numbers are in red.



Breve explicación

- ❑ Nuestra clase controlador, implementa el `ActionListener`, esto para responder desde esta clase, los eventos realizados desde la interfaz (VISTA).
- ❑ El constructor de la clase pasa como parámetros, la clase VISTA y la clase MODELO.
- ❑ Nuestra clase además cuenta las funciones, `INICIAR()` la cual inicializa los valores de la interfaz, como ser el atributo título del `JFrame`, posicionamiento en pantalla, valores iniciales de los `jtextbox`, etc.
- ❑ El método **`action performed`** captura el evento realizado desde la interfaz.
- ❑ Un `CLICK EN EL BOTON SUMAR`, obtiene los datos correspondientes e invoca al modelo para procesar la información y obtener una respuesta.



Universidad **César Vallejo**

Licenciada por Sunedu
para que puedas salir adelante