



Tecnicatura Universitaria **en Desarrollo de Software**

Ingeniería del Software

2020

Modelo-Vista-Controlador
(MVC)



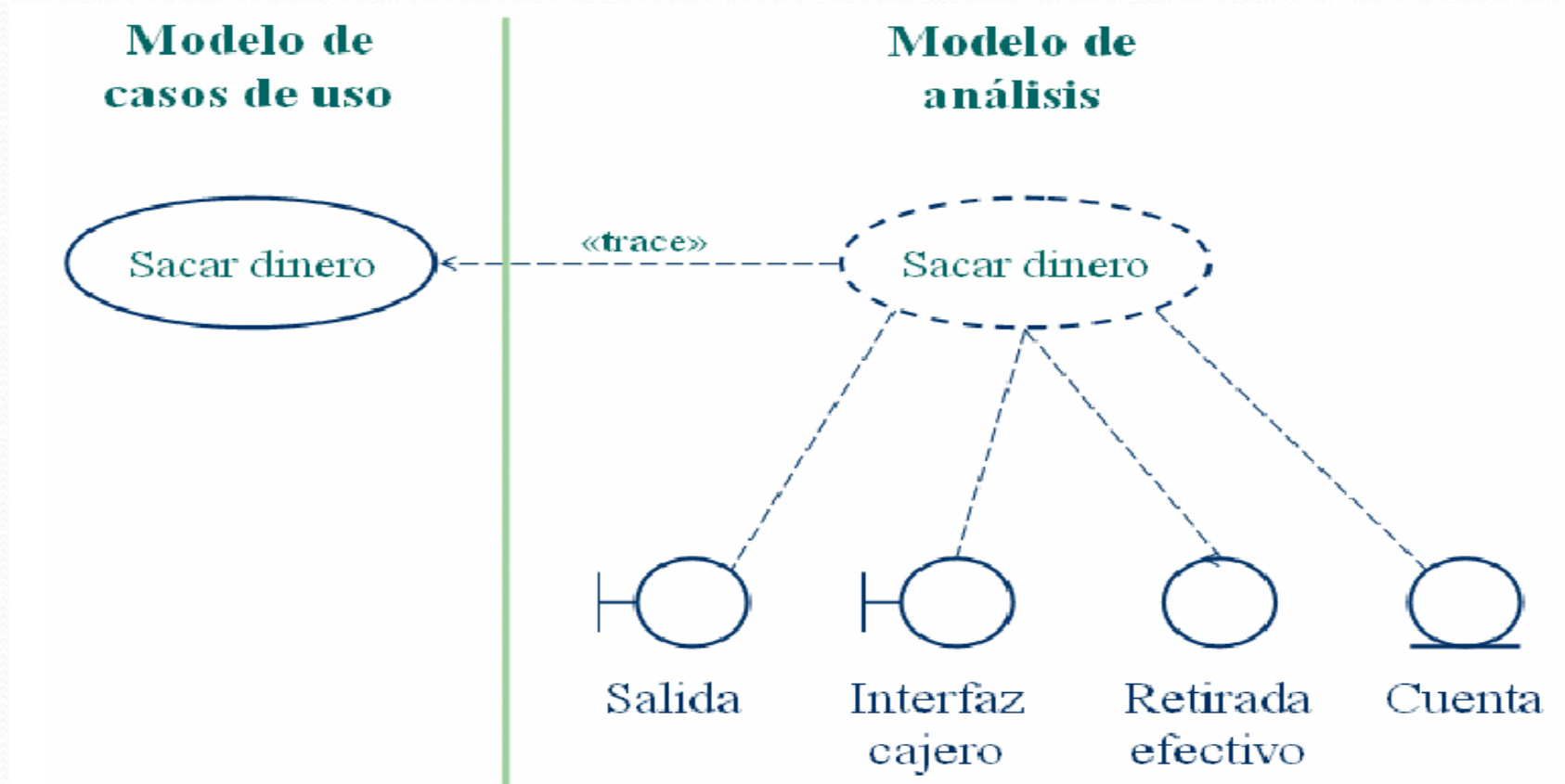
Modelo de Análisis

- El modelo de análisis **crece incrementalmente** a medida que se analizan más y más Casos de Uso.
- *En cada iteración, elegimos un conjunto de CU y construimos el sistema como un conjunto de clasificadores (clases de análisis) y relaciones entre ellas.*
- También describimos colaboraciones que llevan a cabo los Casos de Uso, es decir las Realizaciones de Casos de Uso.
- Después en la siguiente iteración tomamos otro conjunto de CU para desarrollar, y los añadimos a la iteración anterior.

Estereotipos de análisis

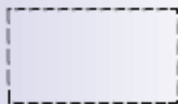
- En el modelo de análisis se utilizan 3 tipos diferentes de clases:
 - **Clases de interfaz:** se utilizan generalmente para modelar la interacción entre el sistema y sus actores.
 - **Clases de control:** se utilizan para representar coordinación, secuenciamiento, transacciones y control de otros objetos, y se utilizan con frecuencia para encapsular el control relativo a un CU.
 - **Clases de entidad:** se usan para modelar información que tiene vida larga y que a veces es persistente.
- Cada uno encapsula un tipo diferente de comportamiento o funcionalidad.

Estereotipos de análisis



Cada uno de estos estereotipos de clase encapsula un tipo diferente de comportamiento o funcionalidad.

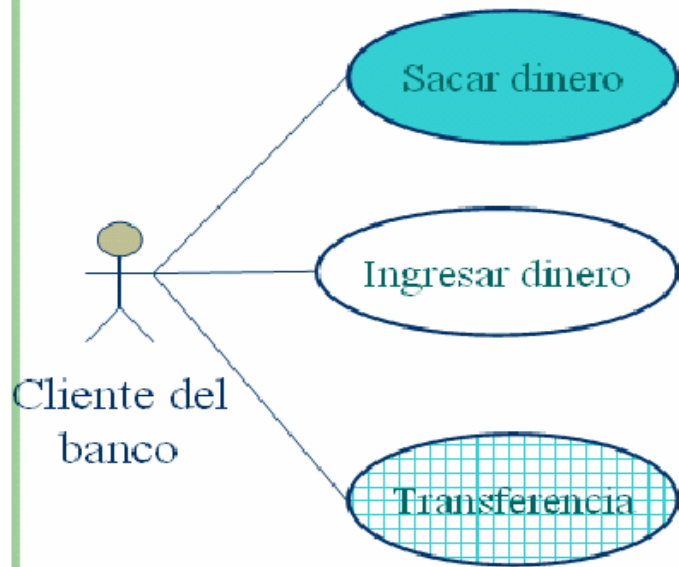
Cada CU se realiza con Clases



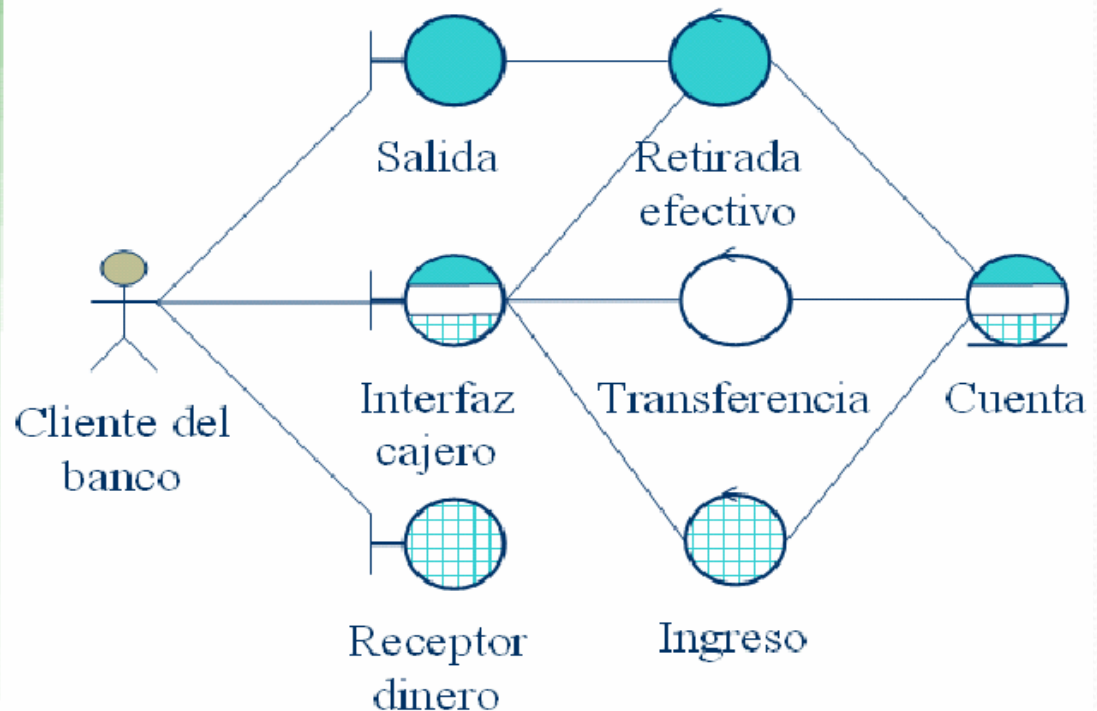
- Actor: Entidad que interactúa con el sistema
- Clase límite: Representa una interfaz con un actor
- Clase Control: Representa un elemento con lógica del sistema
- Clase Entidad: Representa un elemento con conocimiento de los datos
- Límite: Define interior y exterior del sistema

Cada CU se realiza con Clases

Modelo de casos de uso

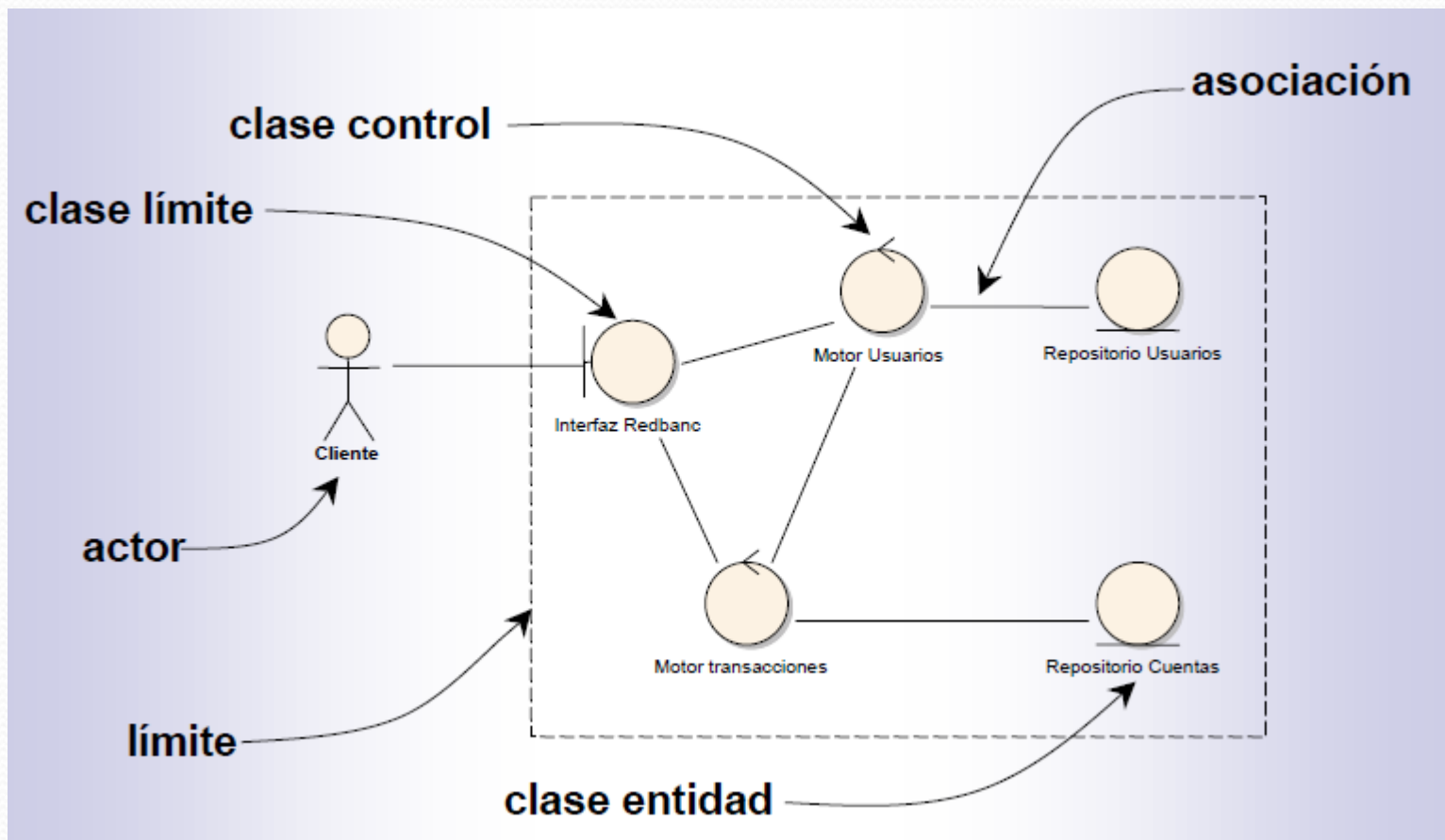


Modelo de análisis



El diagrama muestra como cada CU se realiza como una estructura de clases de Análisis.

Cada CU se realiza con Clases

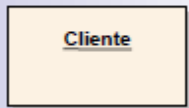


Diagramas de colaboración

- Cada CU se desarrolla como una realización de CU.
- Comprender los patrones de interacción significa que describimos como se lleva a cabo o se ejecuta (o se instancia) una realización de CU.
- Utilizamos **diagramas de colaboración** para modelar las interacciones entre objetos del análisis (también diagramas de secuencia para modelar las interacciones).

Diagramas de Colaboración

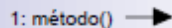
- Utilizamos **diagramas de colaboración** para modelar las interacciones entre objetos del análisis



Objeto: Instancia de una clase que participa en la acción



Enlace: Línea que indica relación estructural entre objetos



Mensaje: Invocación de un objeto a un método propio o de otro objeto, posee un número de secuencia para reconocer el orden

Diagramas de Colaboración

- Un diagrama de colaboración es como el de clases, pero contiene instancias y enlaces, en lugar de clases y asociaciones.
- Propósito similar al diagrama de secuencia: mostrar interacción entre objetos
- Resalta la organización estructural de los objetos que interactúan
- Muestra Objetos, Enlaces y Mensajes
- En UML 2.0 se llama “Diagrama de Comunicación”

Juan:Persona

- Objetos: Instancias de clases. Objetos “vivos”

Juan:Persona

Boby:Perro

- Enlace: Relación entre dos objetos vivos

enlace

objeto

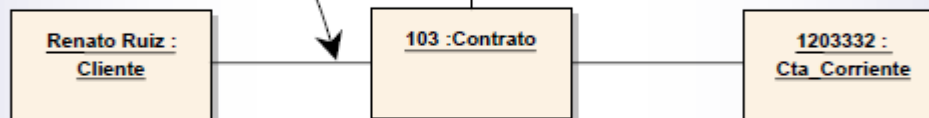


Diagrama de clases

Diagrama de objetos

Clase

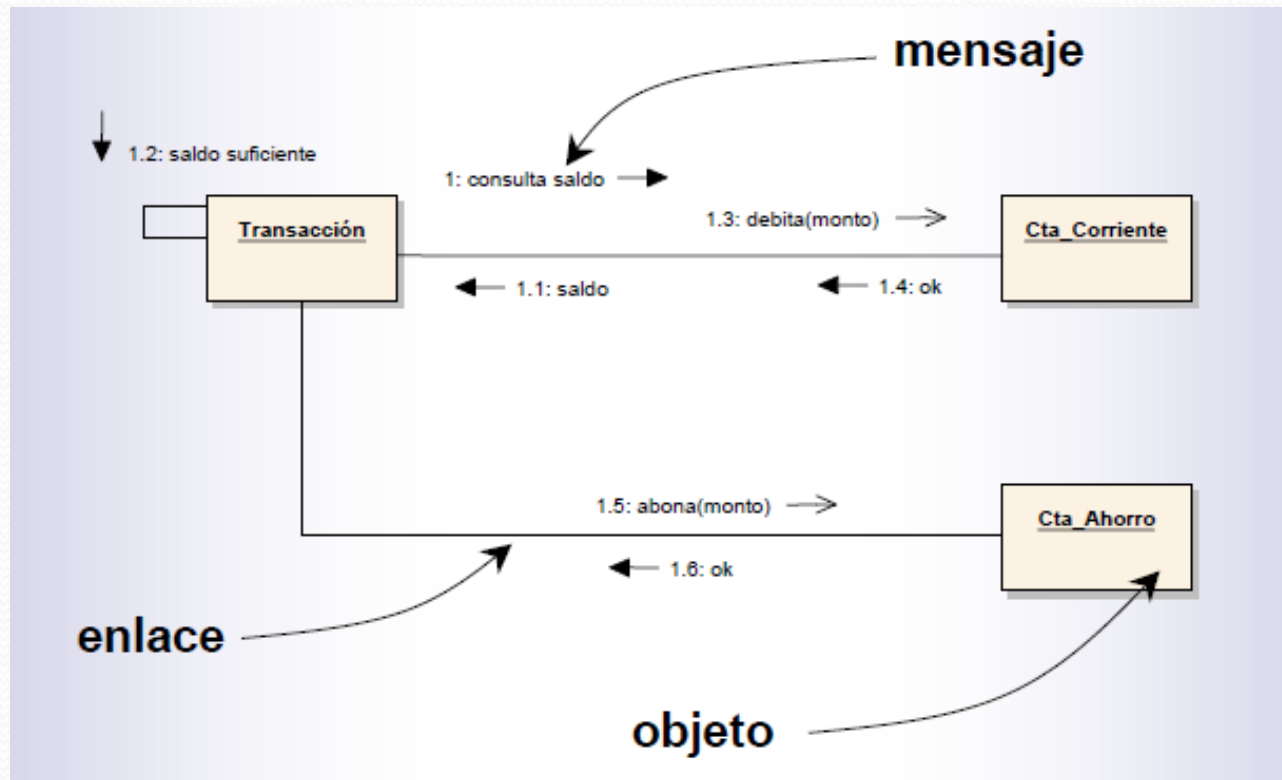
Objeto

Asociación

enlace

Diagramas de Colaboración

- El nombre de un mensaje indica el motivo del objeto que realiza la llamada en su interacción con el objeto invocado.



- En el diseño cada mensaje se refinará en una o más operaciones proporcionadas por clases de diseño.

¿Secuencia o Colaboración?

	Pocos Mensajes	Muchos Mensajes
Pocos objetos	Preferible Colaboración	Secuencia
Muchos objetos	Colaboración	

Modelo-Vista-Controlador

- Este patrón fue descrito por primera vez por Trygve Reenskaug en 1979, y la implementación original fue realizada en Smalltalk en los laboratorios Xerox.
- MVC se basa en la separación de la aplicación en tres capas principales: **Modelo, Vista y Controlador**.
- Se usa (él o alguna de sus variantes) en la gran mayoría de las interfaces de usuario.

Modelo-Vista-Controlador

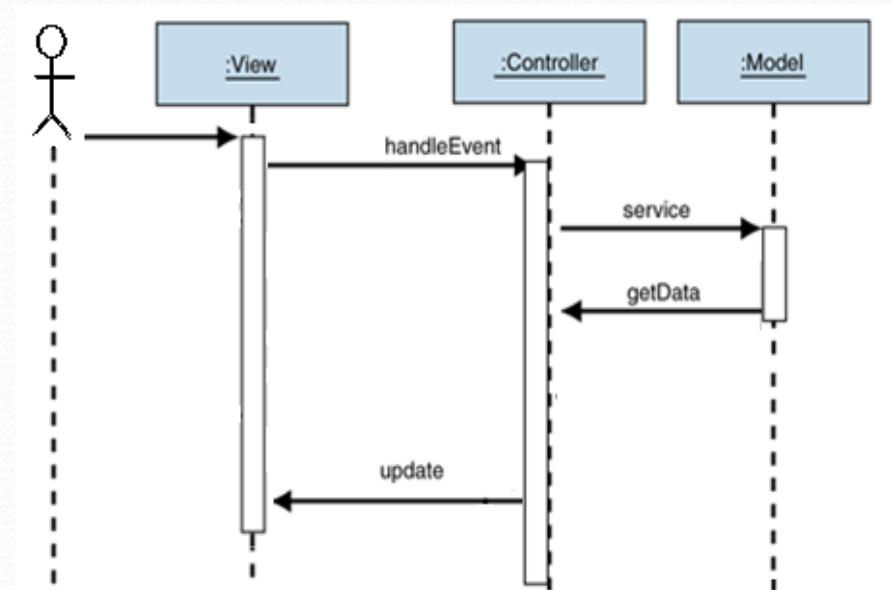
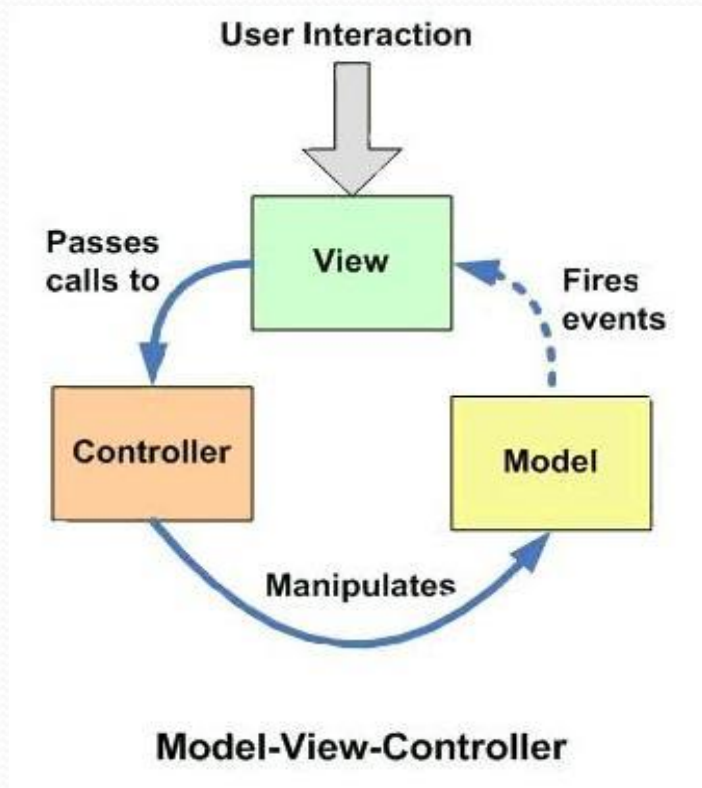
- **Modelo:** es la representación específica del dominio de la información sobre la cual funciona la aplicación.
- El modelo es otra forma de llamar a la capa de dominio.
- La lógica de dominio añade significado a los datos; por ejemplo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o portes en un carrito de la compra.

Modelo-Vista-Controlador

- **Vista:** Se presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario.
- **Controlador:** Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Modelo-Vista-Controlador

En general



Modelo-Vista-Controlador

- Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos. MVC no menciona específicamente esta capa de acceso a datos porque supone que está encapsulada por el modelo.
- El objetivo primordial del MVC es la reutilización del código ya implementado.
- A la hora de programar **separamos el código en varias partes que sean susceptibles de ser reutilizadas.**

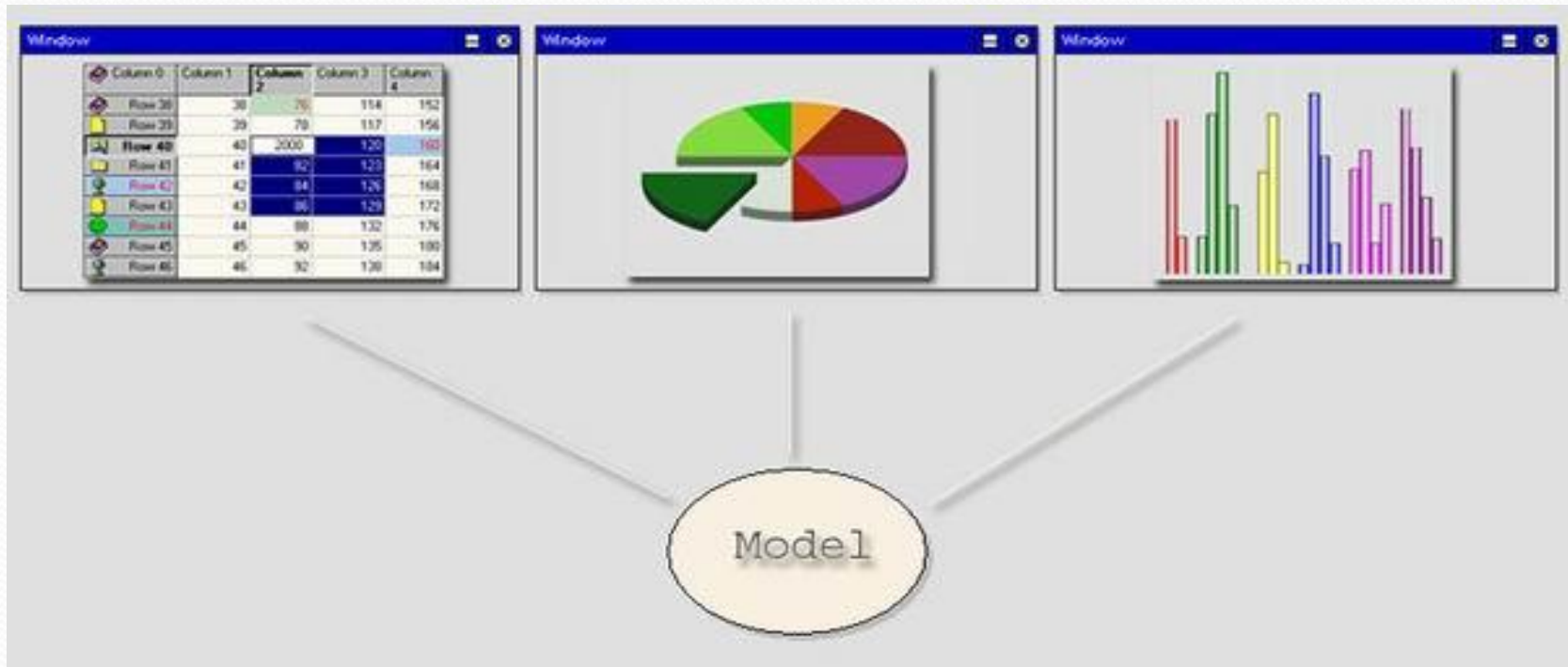
Modelo-Vista-Controlador

Ejemplos

- Los datos de una hoja de cálculo pueden mostrarse de en formato tabular, con un gráfico de barras, con uno de sectores.
- Los datos son el modelo.
- Si cambia el modelo, las vistas deberían actualizarse en consonancia.
- El usuario manipula el modelo a través de las vistas.
(en realidad, a través de los controladores)

Modelo-Vista-Controlador

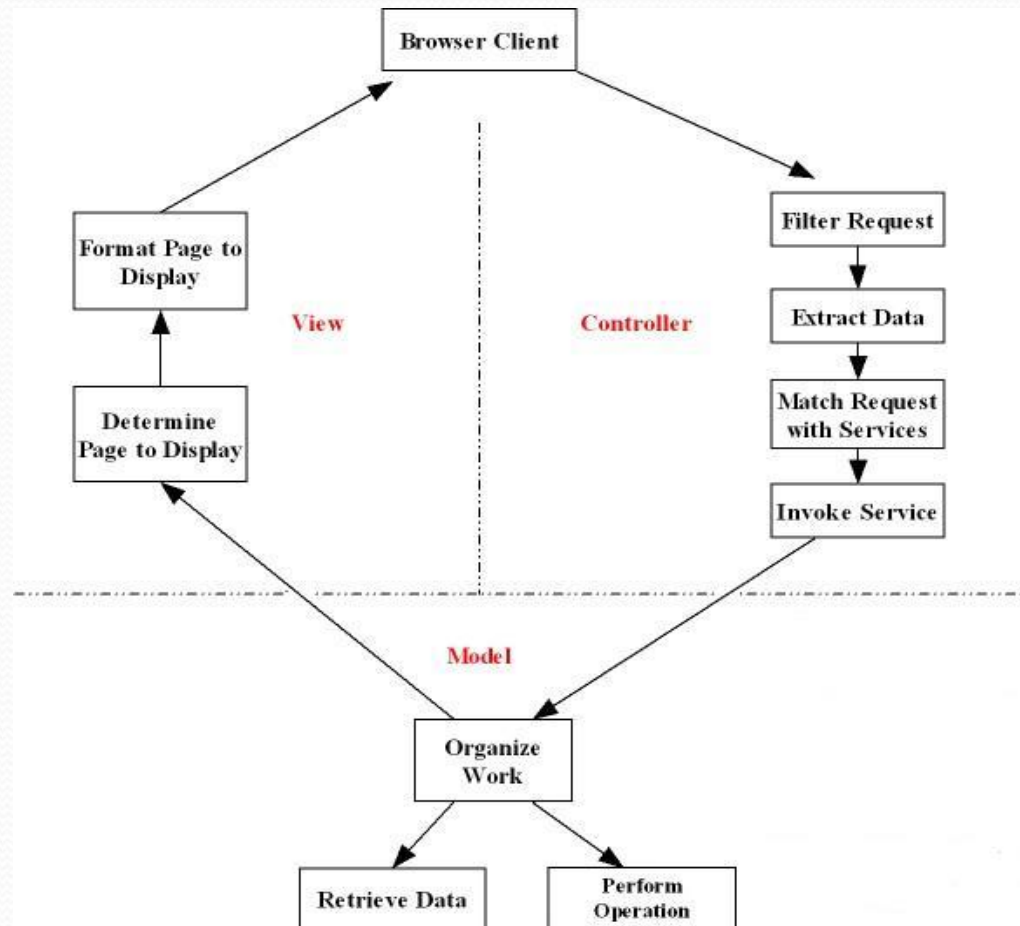
Mas de una Vista de un Modelo de Datos



Modelo-Vista-Controlador

- MVC es utilizado con mayor frecuencia en las aplicaciones web, donde la Vista es la página HTML, y el Controlador es el código que reúne la data dinámica y genera el contenido de la página.
- El Modelo es representado por el contenido actual, que usualmente se encuentra almacenado en una base de datos o en archivos XML.

Modelo-Vista-Controlador



Modelo-Vista-Controlador

Fortalezas

- Se presenta la misma información de distintas formas.
- Las vistas y comportamiento de una aplicación deben reflejar las manipulaciones de los datos de forma inmediata.
- Debería ser fácil cambiar la interfaz de usuario (incluso en tiempo de ejecución).
- Permitir diferentes estándares de interfaz de usuario o portarla a otros entornos no debería afectar al código de la aplicación.

Modelo-Vista-Controlador

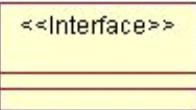
- En UML

Se propone para el desarrollo del Modelo de Análisis de las aplicaciones, tres tipos de clases fundamentales, con las cuales podemos expresar todas las funciones de cualquier software, con sus respectivas responsabilidades

Clase Interfaz <<Interface>>:

Recepcionar peticiones al sistema.

Mostrar respuestas del sistema.

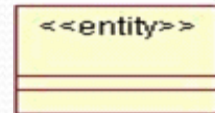


Clase Entidad <<Entity>>:

Gestionar datos (información) necesaria para el sistema.

Almacenar datos (información) persistentes del sistema.

Provee la funcionalidad principal de la aplicación



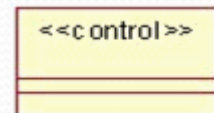
Clase Controlador

<<Controller>>:

Procesar Información del sistema.

Gestionar visualización de respuesta del sistema.

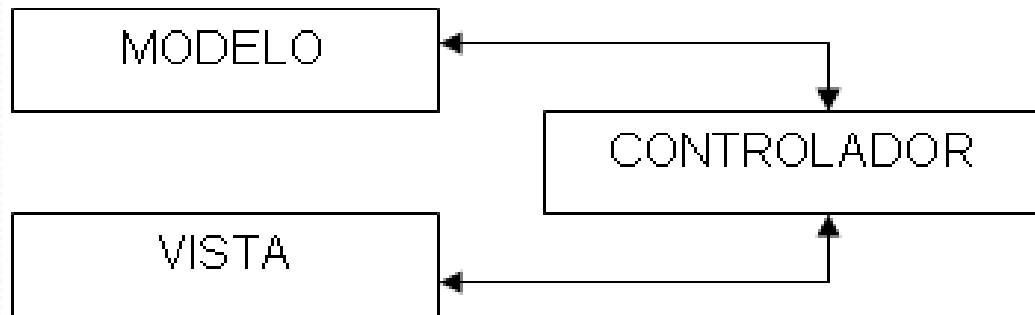
Obtiene los datos del modelo.



Modelo-Vista-Controlador

Variantes del Modelo.

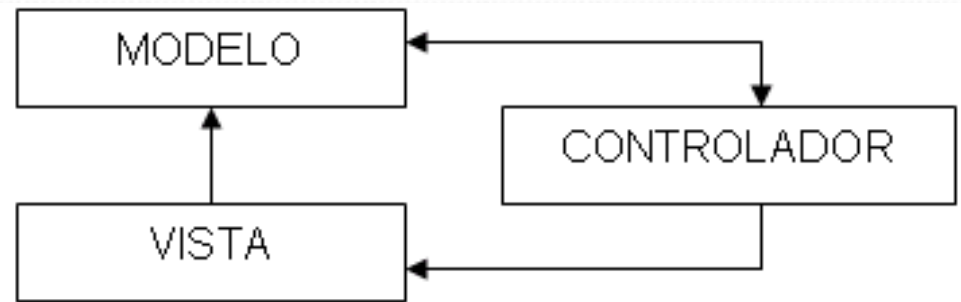
- Variante en la cual no existe ninguna comunicación entre el Modelo y la Vista y esta última recibe los datos a mostrar a través del Controlador.



- Variante inicial del Patrón MVC.

Modelo-Vista-Controlador

- Variante en la cual se desarrolla una comunicación entre el Modelo y la Vista, donde esta última al mostrar los datos los busca directamente en el Modelo dada una indicación del Controlador, disminuyendo el conjunto de responsabilidades de este último.



Variante Intermedia del Patrón MVC.

Modelo-Vista-Controlador

Muchas interfaces gráficas de usuario, como Swing o MFC, hacen innecesario el uso de un controlador.

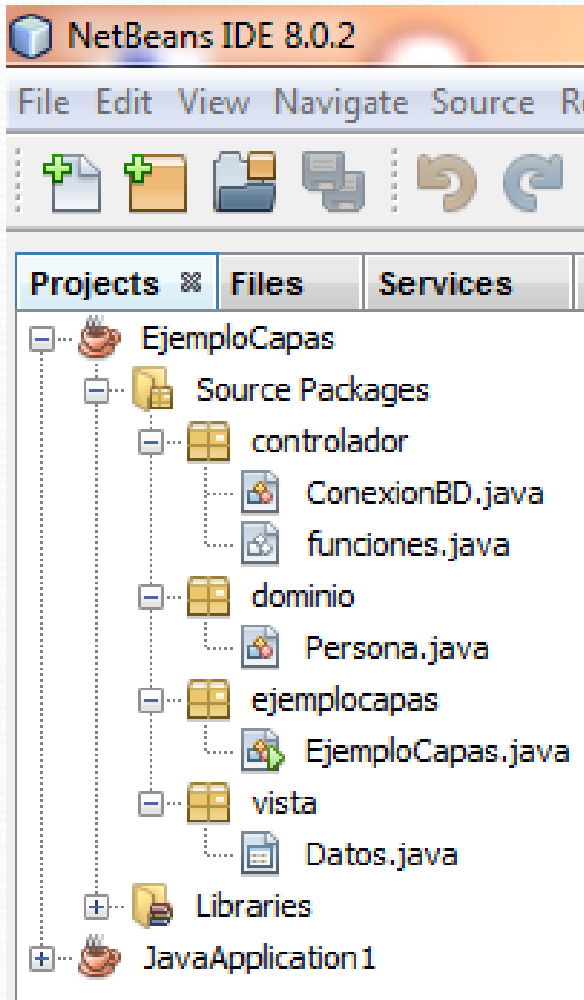
- Definen su propio flujo de control y manejan los eventos internamente.
- Integran, así, la vista y el controlador.
- A esta variante se la suele denominar *Document-View*

Modelo-Vista-Controlador

Un controlador (controlador.java, por ejemplo) puede gestionar el clic en un botón, de tal forma que recoge datos por medio del Modelo (model.cargar_texto(..)) y los manda a la Vista (el applet) para su actualización (vista.mostrar_texto()):

```
/******  
Responde al click en botón "abrir" La respuesta al evento es hacer que se abra en la  
vista el archivo correspondiente a la referencia seleccionada en el combo box  
*****/  
void b_abrir_actionPerformed(ActionEvent e) {  
...  
String texto_archivo = model.cargar_texto( indice_ref ); // Obtener texto de  
archivo  
/** Si la carga de archivo es ok, lo muestro. Si no, aviso de error ****/  
if (texto_archivo != null) {  
    vista.mostrar_texto(texto_archivo); // Mostrar texto  
    vista.mostrar_aviso("Carga de " + path + " completada.");  
}else  
    vista.mostrar_aviso("Error en la carga de " + path);  
}
```


Un Ejemplo en NetBeans



- La aplicación toma datos y los muestra en Jlabels



Un Ejemplo en NetBeans

- La App (programa principal) invoca o crea las vistas correspondientes. En este caso Datos.java

```
package ejemplocapas;  
import vista.Datos;  
/**...4 lines */  
public class EjemploCapas {  
    /**...3 lines */  
    public static void main(String[] args) {  
        Datos datos =new Datos();  
        datos.setVisible(true);  
    }  
}
```

Vista: Datos

- Cada botón de la vista envía los datos al controlador para validar lo ingresado en los campos del form

```
private void btnCreaActionPerformed(java.awt.event.ActionEvent evt) {  
    persona=new Persona(Integer.parseInt(txtDocumento.getText()),txtNombre.getText(),txtApellido.getText())  
    JOptionPane.showMessageDialog(null,"Persona creada Ok! ","",JOptionPane.INFORMATION_MESSAGE);  
}  
  
private void btnMuestraActionPerformed(java.awt.event.ActionEvent evt) {  
    try{  
        lblDocumento.setText(Integer.toString(persona.getDni()));  
    }  
    catch (NumberFormatException e){  
        JOptionPane.showMessageDialog(null,"Error "+e,"",JOptionPane.ERROR_MESSAGE);  
    }  
    lblApellido.setText(persona.getApellido());  
    lblNombre.setText(persona.getNombre());  
}
```

Vista: Datos

- La vista contiene todos los componentes visuales que permiten la interacción con el usuario.

```
1  package vista;
2  import controlador.funciones;
3  import dominio.Persona;
4  import javax.swing.JOptionPane;
5
6  public class Datos extends javax.swing.JFrame {
7      private Persona persona=null;
8      public Datos() {
9          initComponents();
10         limitar();
11     }
12     public final void limitar(){
13         funciones.SLetras(txtNombre);
14         funciones.SLetras(txtApellido);
15         funciones.SNumero(txtDocumento);
16     }
```


Controlador: Funciones.java

```
1 package controlador;
2 import java.awt.Toolkit;
3 import java.awt.event.KeyEvent;
4 import java.awt.event.KeyListener;
5 import javax.swing.JTextField;
6
7 public abstract class funciones {
8     //-----Sólo permite letras-----
9     public static void SLetras(JTextField a) {
10         a.addKeyListener(new KeyListener() {
11             @Override
12             public void keyTyped(KeyEvent e) {
13                 char c=e.getKeyChar();
14                 if(Character.isDigit(c)) {
15                     Toolkit.getDefaultToolkit().beep();
16                     e.consume();
17                 }
18             }
19             @Override
20             public void keyPressed(KeyEvent e) {
```

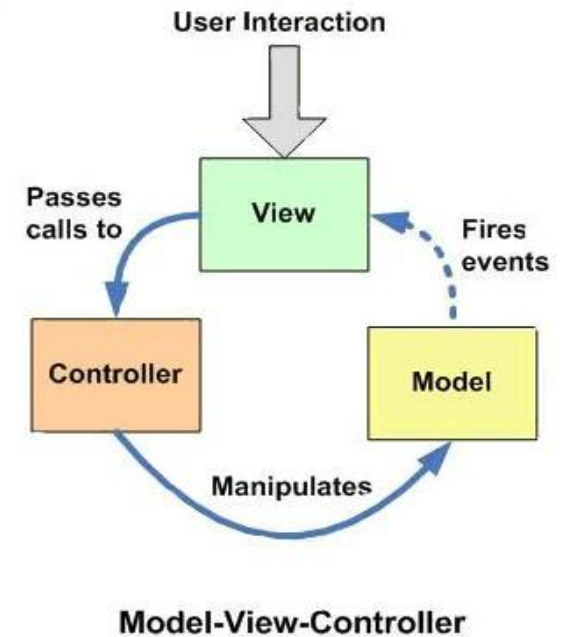
- La aplicación solo consume lo ingresado si es una letra(SLetras) o solamente si es un número(SNumero)

```
31 //-----Sólo permite números-----
32 public static void SNumero(JTextField a)
33 {
34     a.addKeyListener(new KeyListener() {
35         @Override
36         public void keyTyped(KeyEvent ke) {
37             char c = ke.getKeyChar();
38             if(Character.isLetter(c))
39             {
40                 Toolkit.getDefaultToolkit().beep();
41                 ke.consume();
42             }
43         }
44     }
45     @Override
46     public void keyPressed(KeyEvent ke) {
47         //throw new UnsupportedOperationException
48     }
```

- De lo contrario, no toma la tecla presionada

Modelo: Persona.java

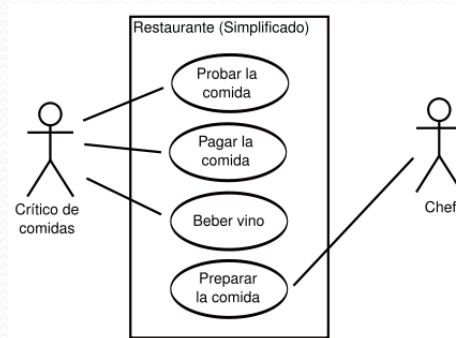
```
1  package dominio;
2  public class Persona {
3      private int dni;
4      private String nombre;
5      private String apellido;
6      /**...6 lines */
12 public Persona(int dni, String nombre, String apellido) {
13     this.dni = dni;
14     this.nombre = nombre;
15     this.apellido = apellido;
16 }
17 public int getDni() {
18     return dni;
19 }
20 public void setDni(int dni) {
21     this.dni = dni;
22 }
23 public String getNombre() {
24     return nombre;
25 }
26 public void setNombre(String nombre) {
```

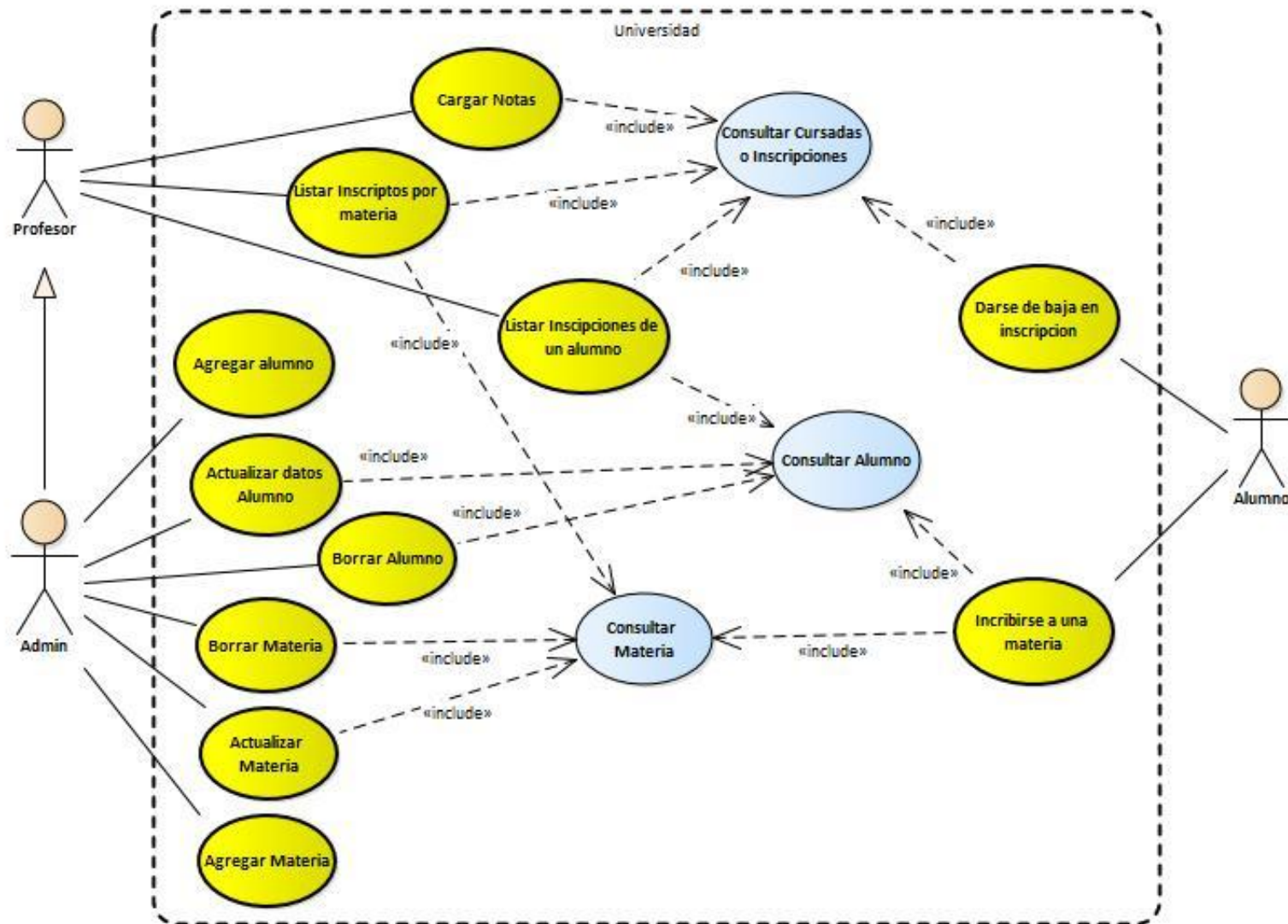


Sistema Universidad

- Durante la segunda certificación vimos el proyecto de alumnos que cursan materias.
- Podemos ver que estuvimos capturando requisitos, analizando y diseñando implícitamente
- Se los guió en la construcción del sistema usando MVC

¿Como imaginas el Modelo de casos de Uso?





Se construyeron entidades

```
4 public class Alumno {
5     private int id = -1;
6     private String nombre;
7     private LocalDate fecNac;
8     private boolean activo;
9
10    public Alumno(int id, String nombre, LocalDate fecNac, boolean activo) {
11        this.id = id;
12        this.nombre = nombre;
13        this.fecNac = fecNac;
14        this.activo = activo;
15    }
16
17    public Alumno(String nombre, LocalDate fecNac, boolean activo) {...5 lines }
18
19    public Alumno() {...3 lines }
20
21    public int getId() {...3 lines }
22
23    public void setId(int id) {...3 lines }
24
25    public String getNombre() {...3 lines }
26
27    public void setNombre(String nombre) {...3 lines }
28
29    public LocalDate getFecNac() {...3 lines }
30
31    public void setFecNac(LocalDate fecNac) {...3 lines }
32
33    public boolean getActivo() {...3 lines }
34
35    public void setActivo(boolean activo) {...3 lines }
36
37    public String toString() {...4 lines }
38
39 }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
```

Hay clases de control

```
public class AlumnoData {
    private Connection connection = null;

    public AlumnoData(Conexion conexion) {
        try {
            connection = conexion.getConnection();
        } catch (SQLException ex) {
            System.out.println("Error al abrir al obtener la conexion");
        }
    }

    public void guardarAlumno(Alumno alumno) {...25 lines }

    public List<Alumno> obtenerAlumnos() {...25 lines }

    public void borrarAlumno(int id) {...20 lines }

    public void actualizarAlumno(Alumno alumno) {...21 lines }

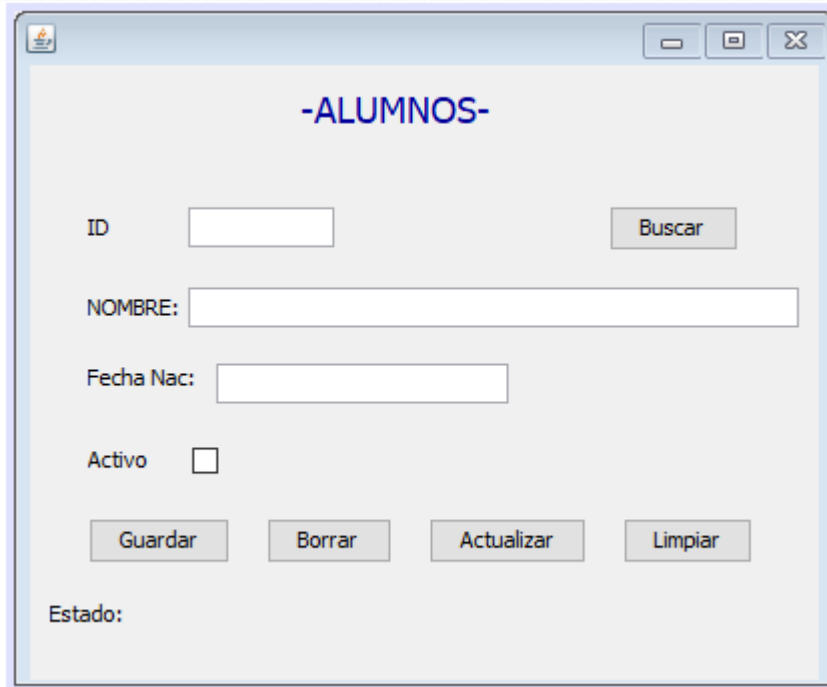
    public Alumno buscarAlumno(int id) {...33 lines }

}
```


Hay clases de control

```
11
12 public class AlumnoData {
13     private Connection connection = null;
14
15     public AlumnoData(Conexion conexion) {
16         try {
17             connection = conexion.getConexion();
18         } catch (SQLException ex) {
19             System.out.println("Error al abrir al obtener la conexion");
20         }
21     }
22
23     public void guardarAlumno(Alumno alumno) {
24         try {
25
26             String sql = "INSERT INTO alumno (nombre, fecNac, activo) VALUES ( ? , ? , ? );"; // 1
27
28             PreparedStatement statement = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
29             statement.setString(1, alumno.getNombre());
30             statement.setDate(2, Date.valueOf(alumno.getFecNac())); //2
31             statement.setBoolean(3, alumno.getActivo());
32
33             statement.executeUpdate(); //3
34
35             ResultSet rs = statement.getGeneratedKeys();
36
37             if (rs.next()) {
38                 alumno.setId(rs.getInt(1));
39             } else {
40                 System.out.println("No se pudo obtener el id luego de insertar un alumno");
41             }
42         }
43     }
44 }
```

Se construyeron interfaces



The image shows a Java Swing window titled "-ALUMNOS-". The window has a standard title bar with minimize, maximize, and close buttons. The main content area is light gray and contains the following elements:

- A label "ID" followed by a text input field and a "Buscar" button.
- A label "NOMBRE:" followed by a wide text input field.
- A label "Fecha Nac:" followed by a date input field.
- A label "Activo" followed by an unchecked checkbox.
- A row of four buttons: "Guardar", "Borrar", "Actualizar", and "Limpiar".
- A label "Estado:" at the bottom left.

Se construyeron interfaces

```
10 public class VistaAlumnos extends javax.swing.JInternalFrame {
11     private AlumnoData alumnoData;
12     private Conexion conexion;
13     public VistaAlumnos() {
14         initComponents();
15         try {
16             conexion = new Conexion("jdbc:mysql://localhost/universidad", "root", "");
17             alumnoData = new AlumnoData(conexion);
18         } catch (ClassNotFoundException ex) {
19             Logger.getLogger(VistaAlumnos.class.getName()).log(Level.SEVERE, null, ex);
20         }
21     }
22     @SuppressWarnings("unchecked")
23     // <editor-fold defaultstate="collapsed" desc="Generated Code">
24     private void initComponents() {
25         jLabel1 = new javax.swing.JLabel();
26         jLabel2 = new javax.swing.JLabel();
27         jLabel3 = new javax.swing.JLabel();
28         btBuscar = new javax.swing.JButton();
29         btGuardar = new javax.swing.JButton();
30         btBorrar = new javax.swing.JButton();
31         btActualizar = new javax.swing.JButton();
32         jtId = new javax.swing.JTextField();
33         jtNombre = new javax.swing.JTextField();
34         btLimpiar = new javax.swing.JButton();
35         jLabel4 = new javax.swing.JLabel();
36         jLabel5 = new javax.swing.JLabel();
37         chActivo = new javax.swing.JCheckBox();
38         bEstado = new javax.swing.JLabel();
```


Se construyeron interfaces

```
181 private void btGuardarActionPerformed(java.awt.event.ActionEvent evt) {  
182     String nombre=jtNombre.getText();  
183     LocalDate fecNac = LocalDate.parse(jtFecha.getText(), DateTimeFormatter.ofPattern("dd/MM/yyyy"));  
184     boolean activo=chActivo.isSelected();  
185     Alumno alumno=new Alumno(nombre,fecNac,activo);  
186     alumnoData.guardarAlumno(alumno);  
187     jtId.setText(alumno.getId()+"");  
188 }  
189 private void btBorrarActionPerformed(java.awt.event.ActionEvent evt) {  
190     // TODO add your handling code here:  
191     int id=Integer.parseInt(jtId.getText());  
192     alumnoData.borrarAlumno(id);  
193 }  
194 private void btBuscarActionPerformed(java.awt.event.ActionEvent evt) {  
195     try{  
196         int id=Integer.parseInt(jtId.getText());  
197         Alumno alumno=alumnoData.buscarAlumno(id);  
198         if(alumno!=null){  
199             jtId.setText(alumno.getId()+"");  
200             jtNombre.setText(alumno.getNombre());  
201             jtFecha.setText(alumno.getFecNac().toString());  
202             chActivo.setSelected(alumno.getActivo());  
203         }  
204     }catch (NumberFormatException e) {  
205         System.out.println("Debe ingresar un Id!");  
206     }  
207 private void btActualizarActionPerformed(java.awt.event.ActionEvent evt) {...14 lines }  
221 public void limpiar(){...9 lines }
```

Se construyeron interfaces

Otro tipo de interfaz:
*Nuestra clase conexión
hacia mariadb*

```
1 package accesoabasededatos.modelo;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5
6 public class Conexion {
7     private String url;
8     private String usuario;
9     private String password;
10
11     private Connection conexion;
12
13     public Conexion(String url, String usuario, String password) throws ClassNotFoundException {
14         this.url = url;
15         this.usuario = usuario;
16         this.password = password;
17         //Cargamos las clases de mariadb que implementan JDBC
18         Class.forName("org.mariadb.jdbc.Driver");
19     }
20     public Connection getConexion() throws SQLException{
21         if(conexion == null){
22             // Setup the connection with the DB
23             conexion = DriverManager
24                 .getConnection(url + "?useLegacyDatetimeCode=false&serverTimezone=UTC"
25                     + "&user=" + usuario + "&password=" + password);
26         }
27         return conexion;
28     }
29 }
30
```

