



# *Tema 3*

---

## *Diseño Basado en Patrones*



# Contenidos

---

- Objetivos
- Introducción
- Patrón *Modelo-Vista-Controlador*
  - Descripción
  - Dependencias
  - Funcionamiento
  - Ventajas
- Ejemplo de aplicación



# Objetivos

---

- *Se presenta la importancia del proceso de diseño de software*
- *Se introduce el concepto de **patrón de diseño**, centrándose en el caso concreto del patrón Model-View-Controller*
- *Para finalizar, se realizan diversos ejemplos siguiendo dicho patrón*



# Introducción

---

- El **diseño** es una **representación** del sistema, realizada con una serie de principios y técnicas, que permite describir el sistema **con el suficiente detalle como para ser implementado**.
- En el contexto de diseño podemos observar que los buenos ingenieros tienen **esquemas y estructuras** de solución que usan numerosas veces en función del contexto del problema.
- Estos esquemas y estructuras son conceptos reusables y nos permiten no reinventar la rueda. Un buen ingeniero reutiliza un esquema de solución ante problemas similares.



# Introducción

---

- El concepto de *patrón de diseño* que tenemos en Ingeniería del Software se ha tomado prestado de la arquitectura. El **patrón** es un esquema de solución que se aplica a un tipo de problema, esta aplicación del patrón no es mecánica, sino que requiere de **adaptación y matices**. Los numerosos usos de un patrón no se repiten dos veces de la misma forma. (*Christopher Alexander*)
- Los patrones de diseño son **descripciones de clases** cuyas **instancias colaboran entre sí**. Cada patrón es adecuado para ser adaptado a un cierto tipo de problema.



# Introducción

---

- Una arquitectura Orientada a Objetos bien estructurada está llena de patrones. La calidad de un sistema Orientado a Objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles (*Grady Booch*)
- Existen múltiples patrones de diseño. En este tema nos centraremos en el patrón “**Modelo-Vista-Controlador**”.



# Patrón *Modelo-Vista-Controlador*

---

- Tres fases en toda aplicación



- Diseño Modular de los componentes responsables de cada una de las fases
  - Entrada de datos ➔ Vista (parte visual) +  
Controlador (responde a lo que ocurre en la vista)
  - Proceso ➔ Modelo
  - Salida ➔ Vista



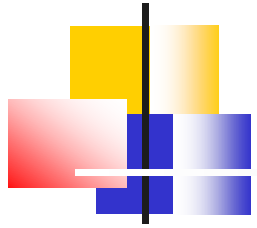
# Patrón *Modelo-Vista-Controlador*

---

Separa los datos de una aplicación, la interfaz de usuario, y la lógica de control:

- **Modelo:** problema que se trata de resolver. Representación específica del dominio de la información sobre la cual funciona la aplicación. La lógica del dominio añade significado a los datos.
- **Vista:** presenta el modelo en un formato adecuado para interactuar con el usuario (*interfaz de usuario*).
- **Controlador:** código que toma decisiones. Responde a eventos, usualmente acciones del usuario, e invoca cambios en el modelo y, probablemente, en la vista.





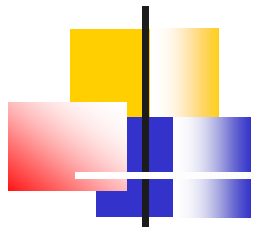
# *Patrón Modelo-Vista-Controlador*

---

Separar estos tres componentes



Es posible modificar cualquiera de ellos sin que eso afecte al funcionamiento del resto, favoreciendo su reutilización



# Patrón *Modelo-Vista-Controlador*

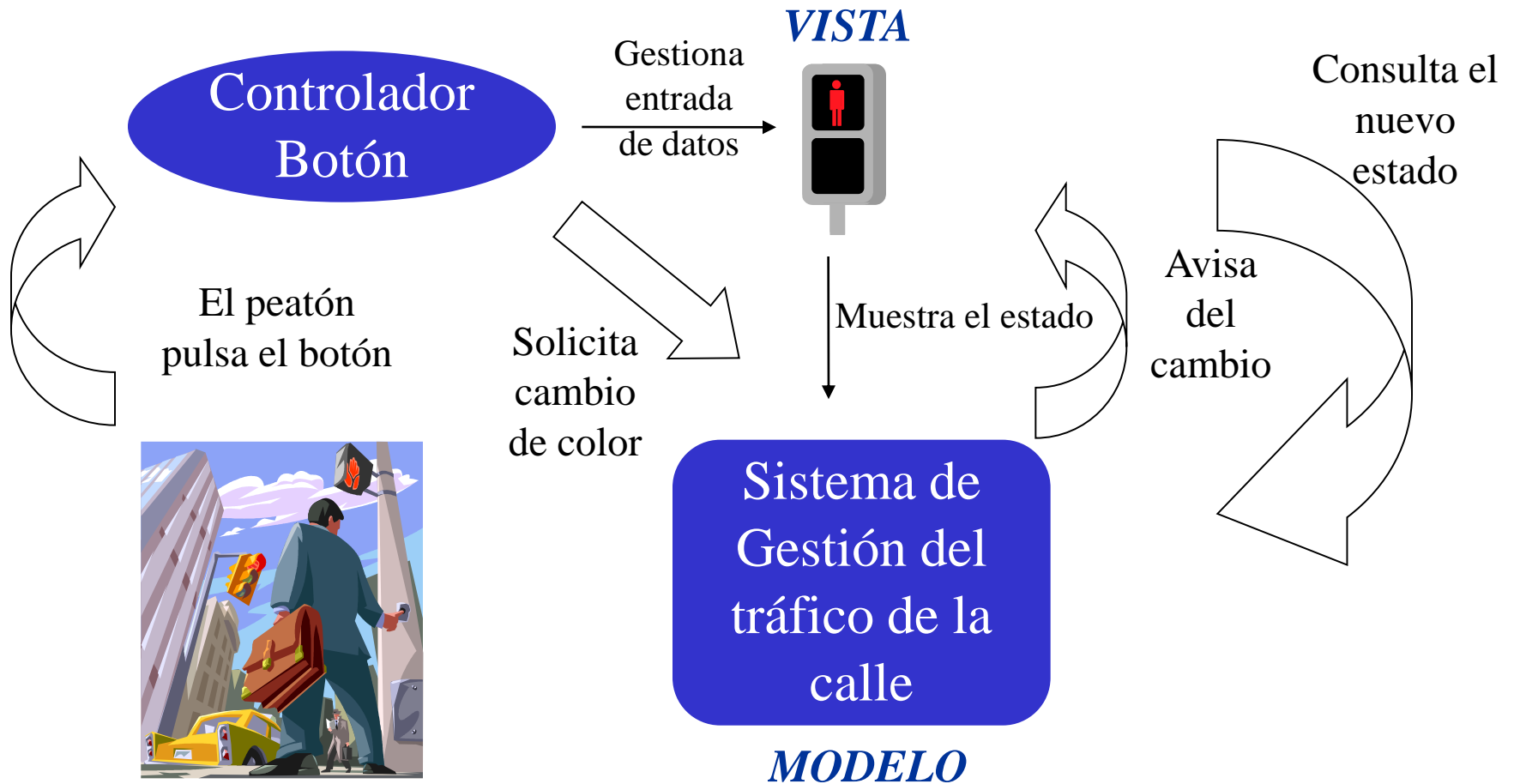
---

## *Ejemplo sencillo:*

Sistema de control de tráfico en una calle pequeña con un semáforo.

- **Modelo:** Aplicación que determina cuándo pueden pasar los peatones y cuándo pueden pasar los vehículos. Implementa la lógica para el cambio de color automático del semáforo.
- **Vista:** Semáforos, pulsador para pedir el paso,...
- **Controlador:** Sistema asociado que detecta la pulsación del botón y se lo comunica al sistema

# Ejemplo de funcionamiento





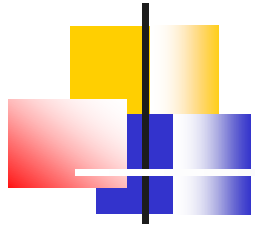
# Dependencias entre M-V-C

---

- El **Modelo** debe ser independiente. Las clases (o funciones y estructuras) del modelo no deben ver a ninguna clase de los otros grupos. De esta forma es posible compilar el modelo en una librería independiente que podrá utilizarse en cualquier programa.

## *Ejemplo del semáforo:*

El modelo es la implementación del funcionamiento del semáforo, el algoritmo para cambiar el color del disco, el algoritmo que controla el tiempo para realizar el cambio automático,...



# Dependencias entre M-V-C

---

- El **Controlador** debe ver el **modelo** que contiene el estado y funcionalidad de los objetos de la aplicación

## *Ejemplo del semáforo:*

Cuando un peatón pulsa el botón del semáforo, el **controlador** debe invocar el metodo correspondiente contenido en el **modelo**.

- El **Controlador** no debe ver la **vista**. De esta forma, el cambio de interfaz gráfica no implicará retocar el algoritmo ni recompilarlo, con los consiguientes riesgos de estropearlo además del trabajo del retoque.



# Dependencias entre M-V-C

---

- La **Vista** es lo más cambiante, así que podemos hacer que vea clases del **modelo** y del **controlador**. Si cambiamos algo del controlador o del modelo, es bastante seguro que tendremos como mínimo que recompilar la interfaz gráfica.

*La mayor parte de las herramientas de desarrollo incorporan en las clases de la **vista** gran parte o todo el procesamiento de eventos, con lo que el **controlador** queda semioculto dentro de la **vista***

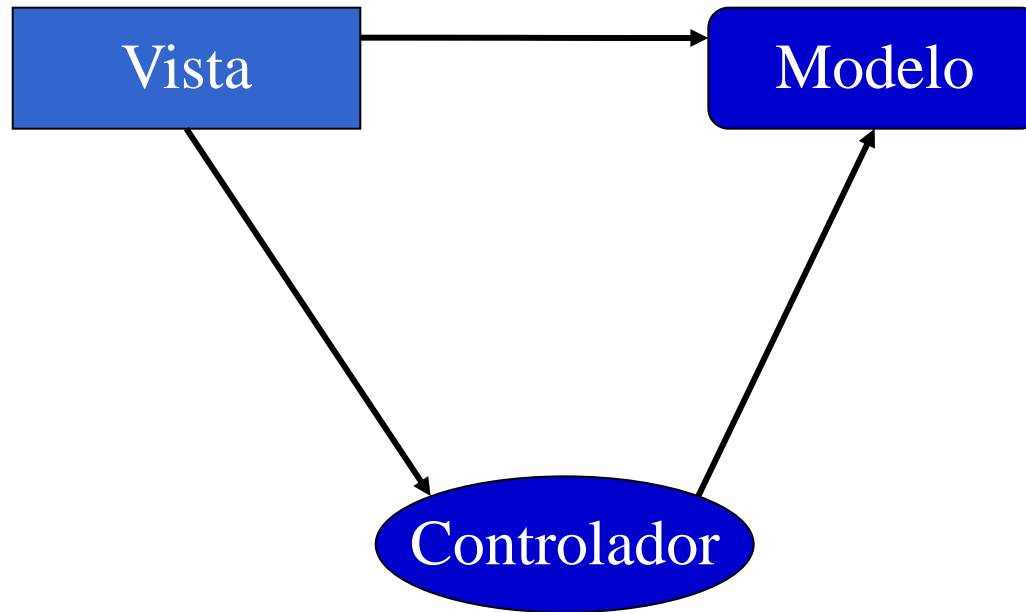
## *Ejemplo del semáforo:*

La vista debe reflejar el estado de los objetos del modelo (el color actual del semáforo, la situación del botón, ...). Por otro lado, para pulsar el botón, el peatón actúa sobre la **vista**, que avisará al **controlador** para que actúe en consecuencia.

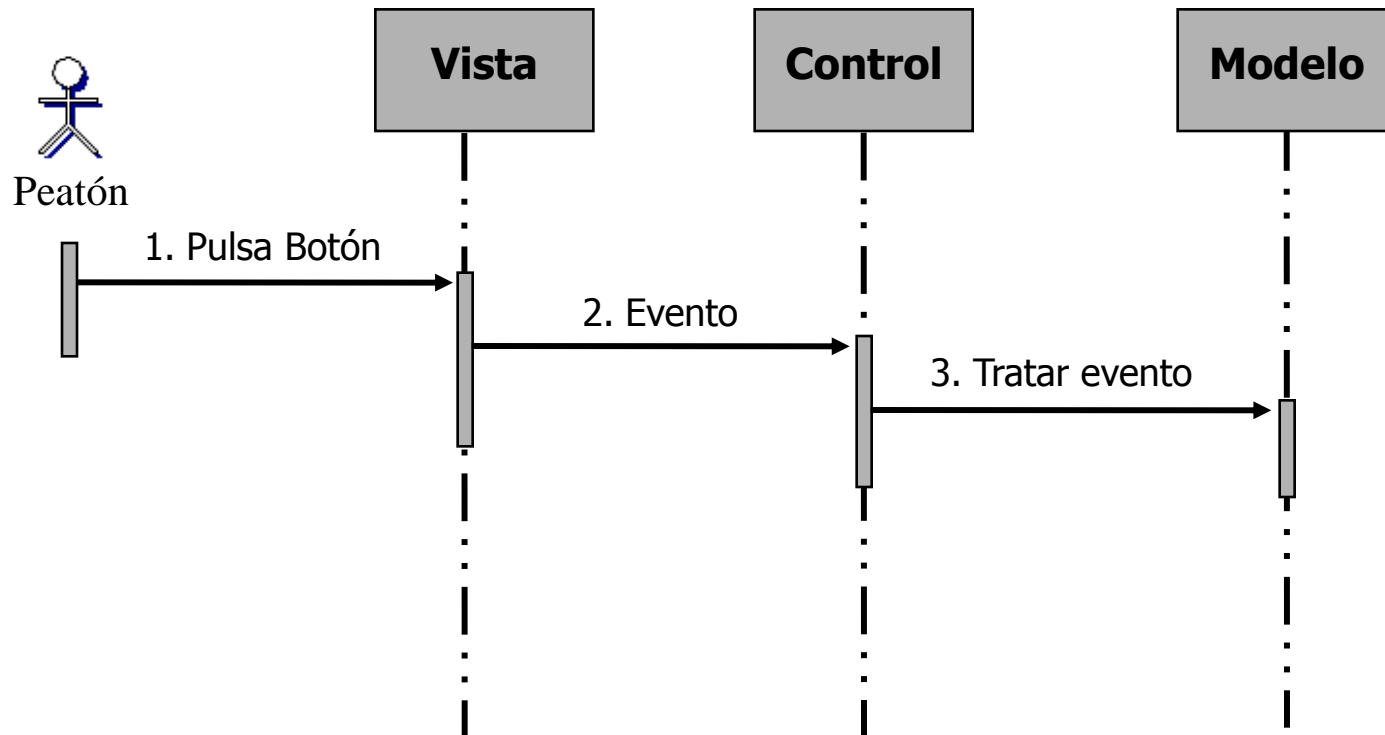


# Dependencias entre *M-V-C*

---



# Funcionamiento de una aplicación MVC



¿Cómo se entera la vista del cambio de color del semáforo?

*Ni el modelo ni el controlador ven a la vista, por lo que no pueden llamar a ninguna clase ni método de ella para que se actualice.*





# Patrón *Observador*

---

## *Solución:*

Es necesario extender el Patrón, pero minimizando las dependencias → nuevos patrones:

- *Observador: Vista.*

Vigila el Modelo, para mostrarlo fielmente.

Reacciona ante los cambios comunicados por el Observable.

- *Observable: Modelo*

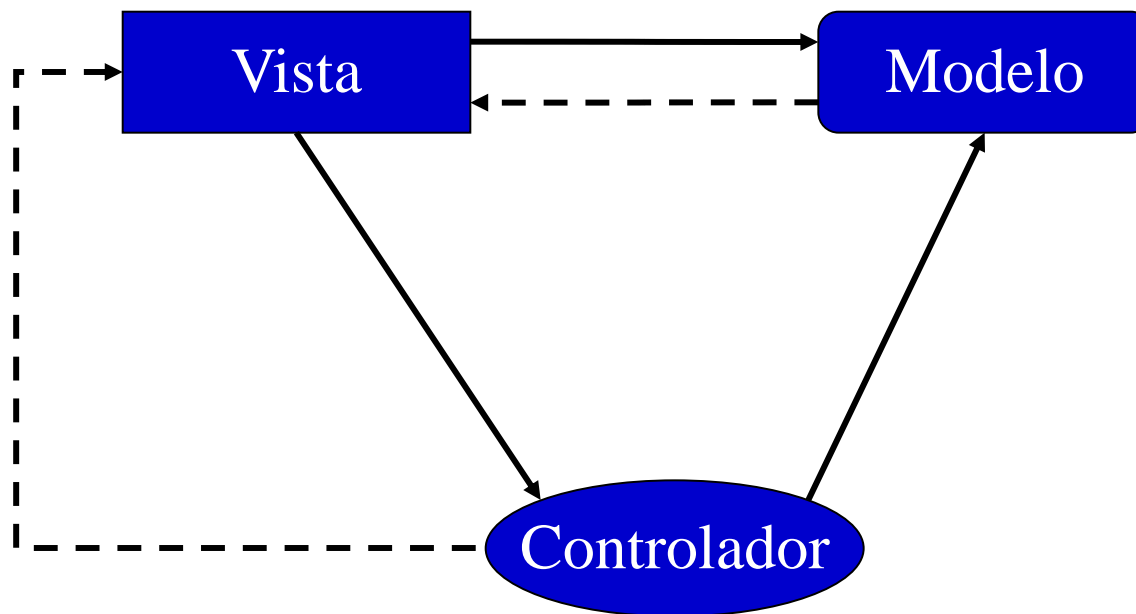
Contiene un registro de *quiénes* están observándolo.

Informa al Observador cuando hay cambios en el Modelo.



# Dependencias entre M-V-C

---

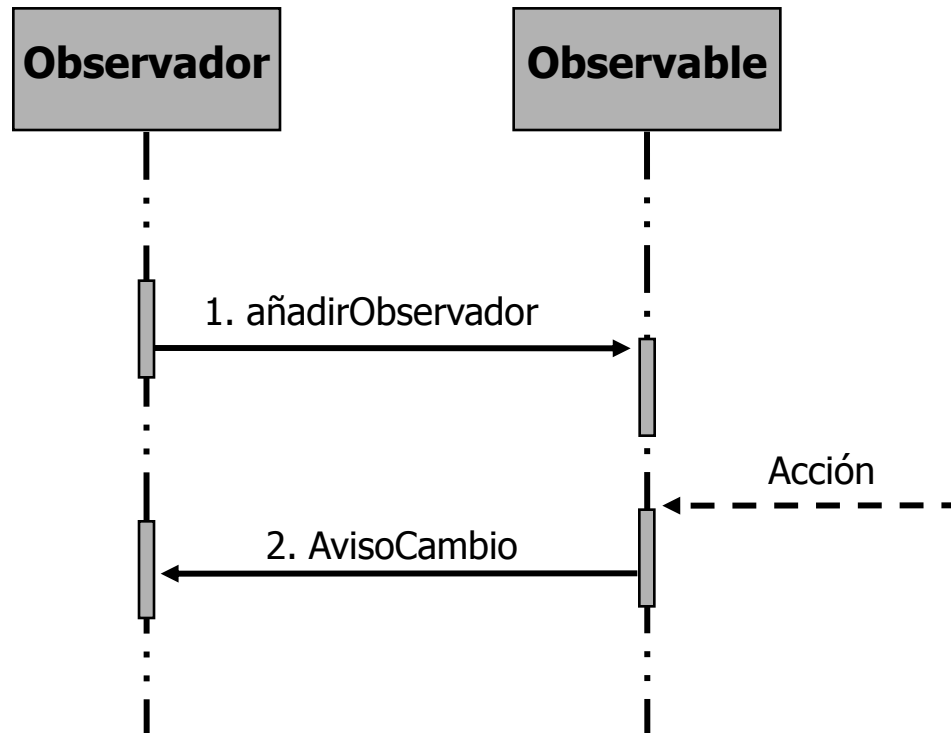


--- ➤ *Comunicación a través del Observador/Observable*

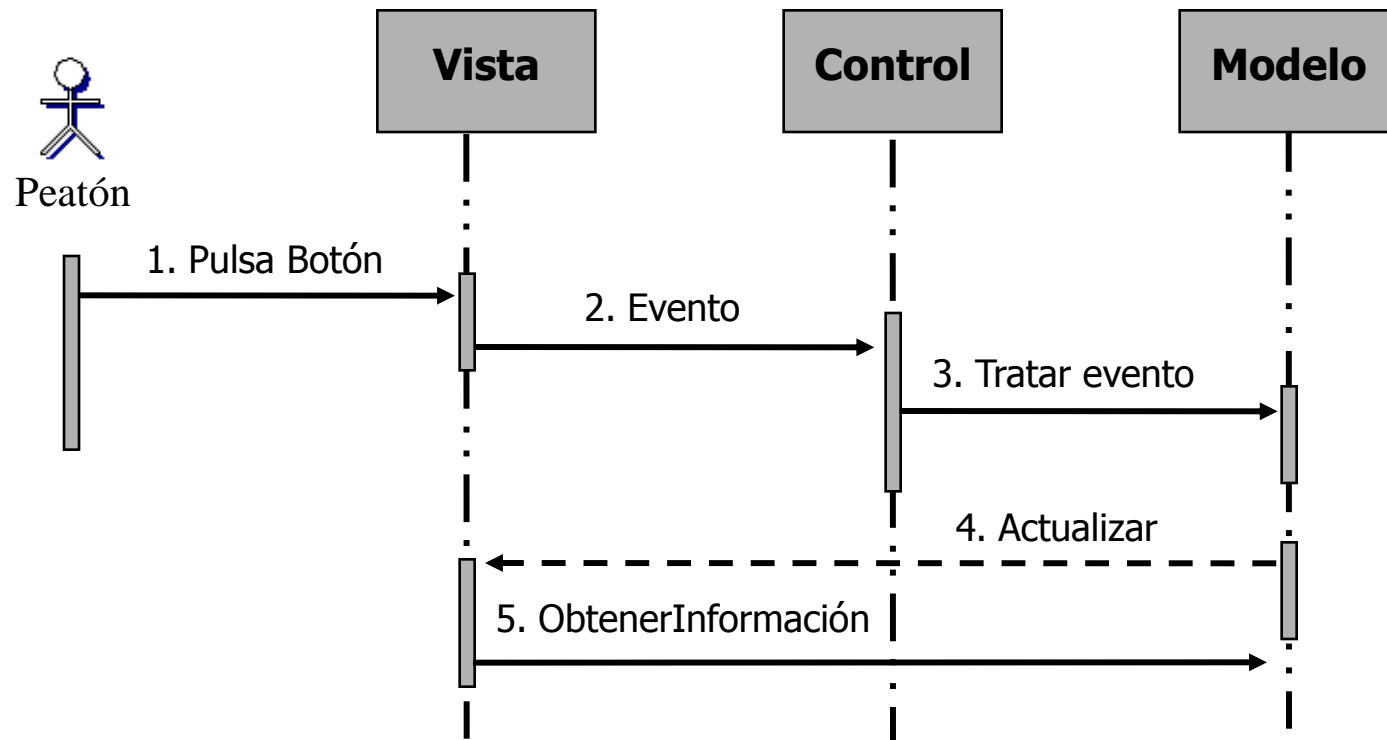


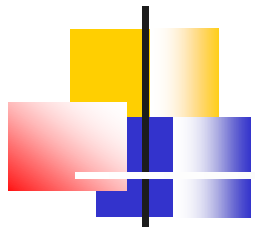
# Patrón *Observador*

---



# Funcionamiento de una aplicación MVC

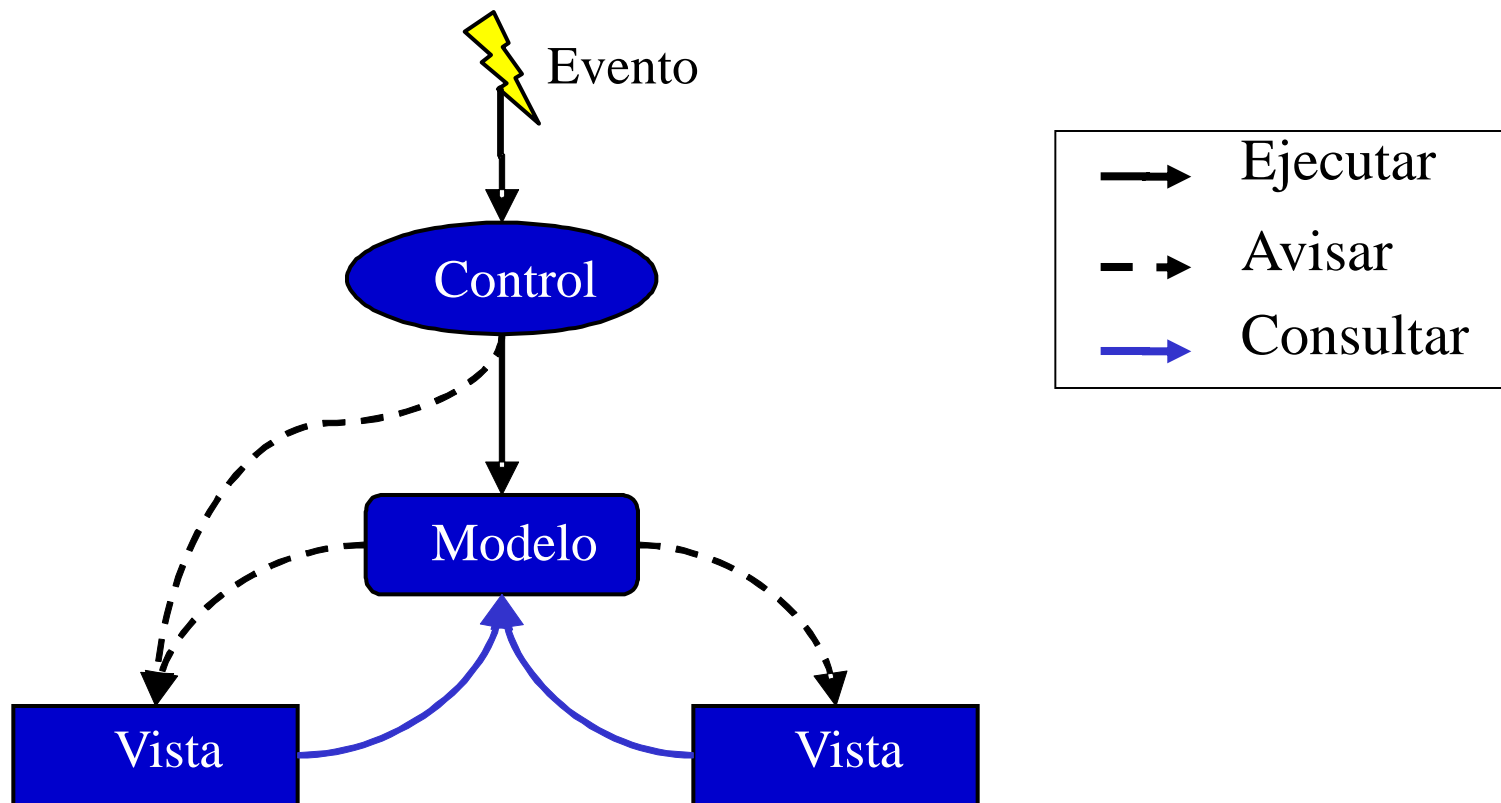




# Dependencias entre *M-V-C*

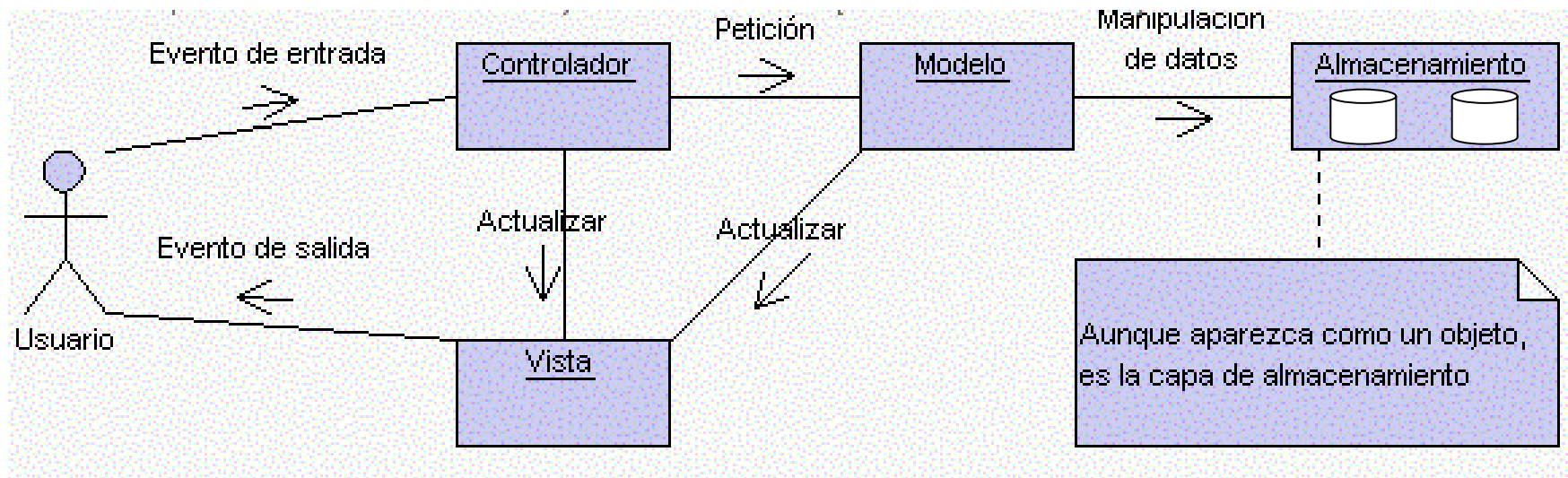
## Un modelo puede tener diversas vistas

**Ejemplo:** la información de una BD se puede presentar de diversas formas: diagrama de tarta, de barras, tabular, etc.



# Funcionamiento de una aplicación MVC

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.

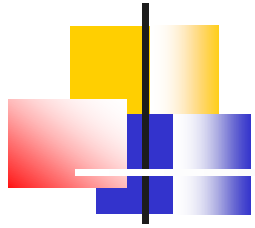




# Ventajas del MVC

---

- Desarrollo independiente de cada uno de los componentes
  - Dependencias identificadas y limitadas
- Modificabilidad
  - Vistas: posibilidad de sustituir las vistas (semáforos) por nuevas versiones (más completas, modernas, ... )
- Múltiples vistas
- Las Vistas pueden observar partes del Modelo
- Aplicable para cualquier tipo de aplicación
  - Herramientas de ayuda
    - Struts (para aplicaciones web)
    - ...



# Ejemplo: Buscaminas

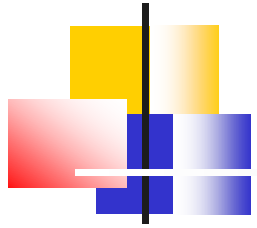
---

## *MODELO*

Constituye el problema que se trata de resolver.

- El modelo contendría una clase (o conjunto de clases) que mantengan una tabla con casillas que pueden contener una mina o un número.
- Las reglas del juego son totalmente independientes de si vamos a dibujar el tablero en 3D o plano y de los recursos gráficos utilizados. Este código también forma parte del modelo.
- Las metodologías orientadas a objeto nos introducen en este tipo de clases, a las que llaman *clases del negocio*.





# Ejemplo: Buscaminas

---


## *VISTA*

Presentación visual que queramos hacer del juego.

Suele ser una interfaz gráfica, pero podría ser de texto, de comunicaciones con otro programa externo, con la impresora, etc.

Estaría constituida por las ventanas del juego:

- ✓ una ventana para recoger la información necesaria para identificar al jugador.
- ✓ una ventana que muestra el tablero del juego. También podría contener botones o menús para finalizar la partida o pedir ayuda durante la realización del juego.



Lista de puntuaciones

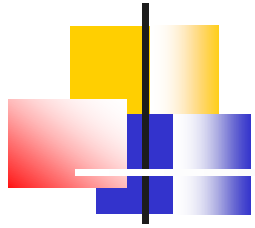
260 <---- Bego

240 <---- Joni

150 <---- Joxean

120 <---- Mikel

OK



# Ejemplo: Buscaminas

---

## *CONTROLADOR*

Es el código que toma decisiones. Este código no tiene que ver con las ventanas visuales ni con las reglas de nuestro modelo.

Formarían parte del **Controlador** los tratamientos de respuesta a los eventos desencadenados cuando el jugador interacciona con la Vista:

- Pulsando algún botón
- Rellenando los campos de Nombre y Nivel
- Pulsando sobre alguna casilla del tablero.