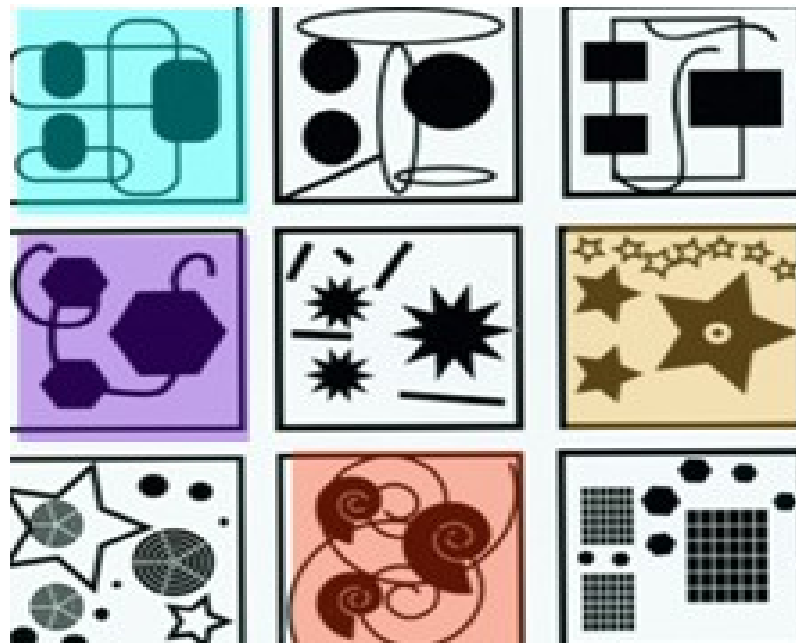


Tema 4



**Conceptos complementarios
en Orientación a Objetos**

Contenidos y bibliografía

Lección	Título	Nº sesiones
4.1	Introducción a los patrones. Patrón Modelo-Vista- Controlador.	1/2
4.2	Manejo de excepciones	1/2
4.3	Colecciones y copia de objetos	1 y 1/2
4.4	Metaclases y reflexión	1/2

http://groups.diigo.com/group/pdoo_ugr



Lección 4.1

Introducción a los patrones.

Patrón

Modelo-Vista-Controlador

Objetivos de aprendizaje



- ◆ Comprender **para qué sirve** un patrón de diseño.
- ◆ Saber qué es un **patrón arquitectónico**.
- ◆ Conocer los **elementos** implicados en la definición del patrón Modelo Vista Controlador.
- ◆ Entender la idea fundamental del patrón MVC y las **ventajas** derivadas de ésta.
- ◆ Saber seguir el **flujo básico** del patrón MVC.
- ◆ Conocer algunos entornos de programación **donde** se puede aplicar dicho patrón de forma asistida.



Contenidos

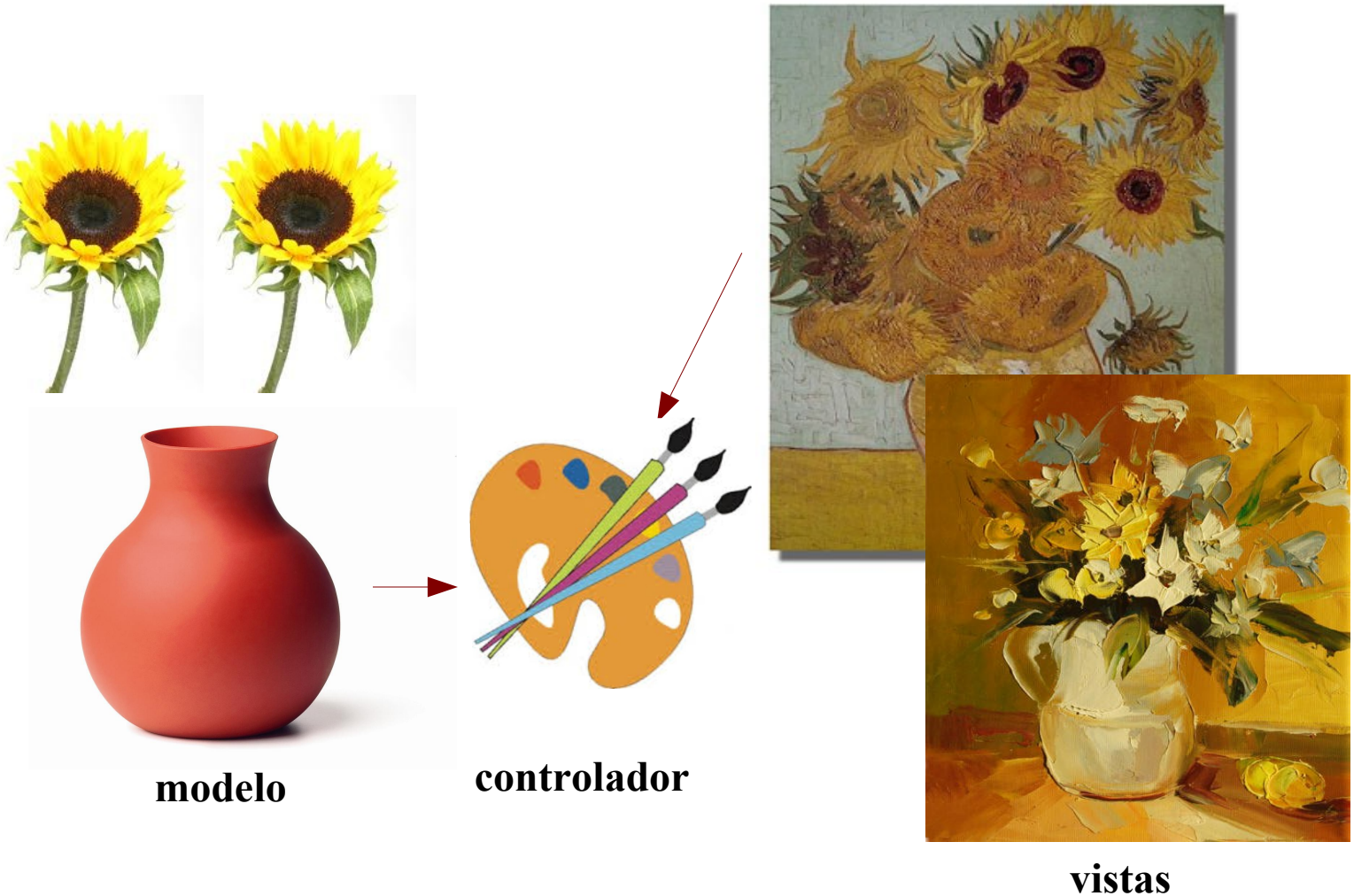


1. Resumen visual del concepto de patrón de diseño
2. Resumen visual del concepto de patrón Modelo
Vista
Controlador
3. Concepto de patrón de diseño
4. Concepto de patrón de arquitectura
5. Modelo Vista Controlador (MVC)

1. Resumen visual del concepto de patrón de diseño



2. Resumen visual del concepto de patrón Modelo Vista Controlador



3. Concepto de patrón de diseño



Christopher Alexander es el padre del lenguaje de patrones (artículo publicado en 1977)

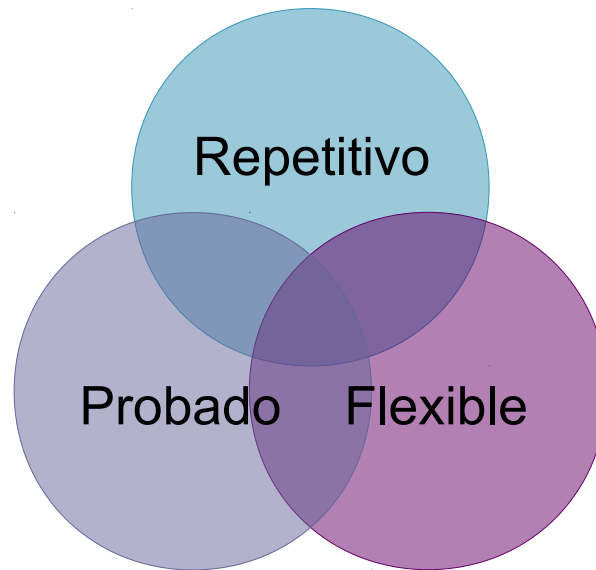
- 253 patrones con conocimiento práctico para la planificación de ciudades y la construcción de edificios (“A Pattern Language: Towns, Buildings, Construction”)

Inspirados en la armonía de las ciudades medievales



3. Concepto de patrón de diseño

- Cada patrón describe un problema que ocurre **una y otra vez** en nuestro entorno, y después describe el núcleo de la solución al problema, de tal manera que tú puedas usar la solución un millón de veces más, sin hacerlo **nunca dos veces de la misma manera** (**Christopher Alexander**)



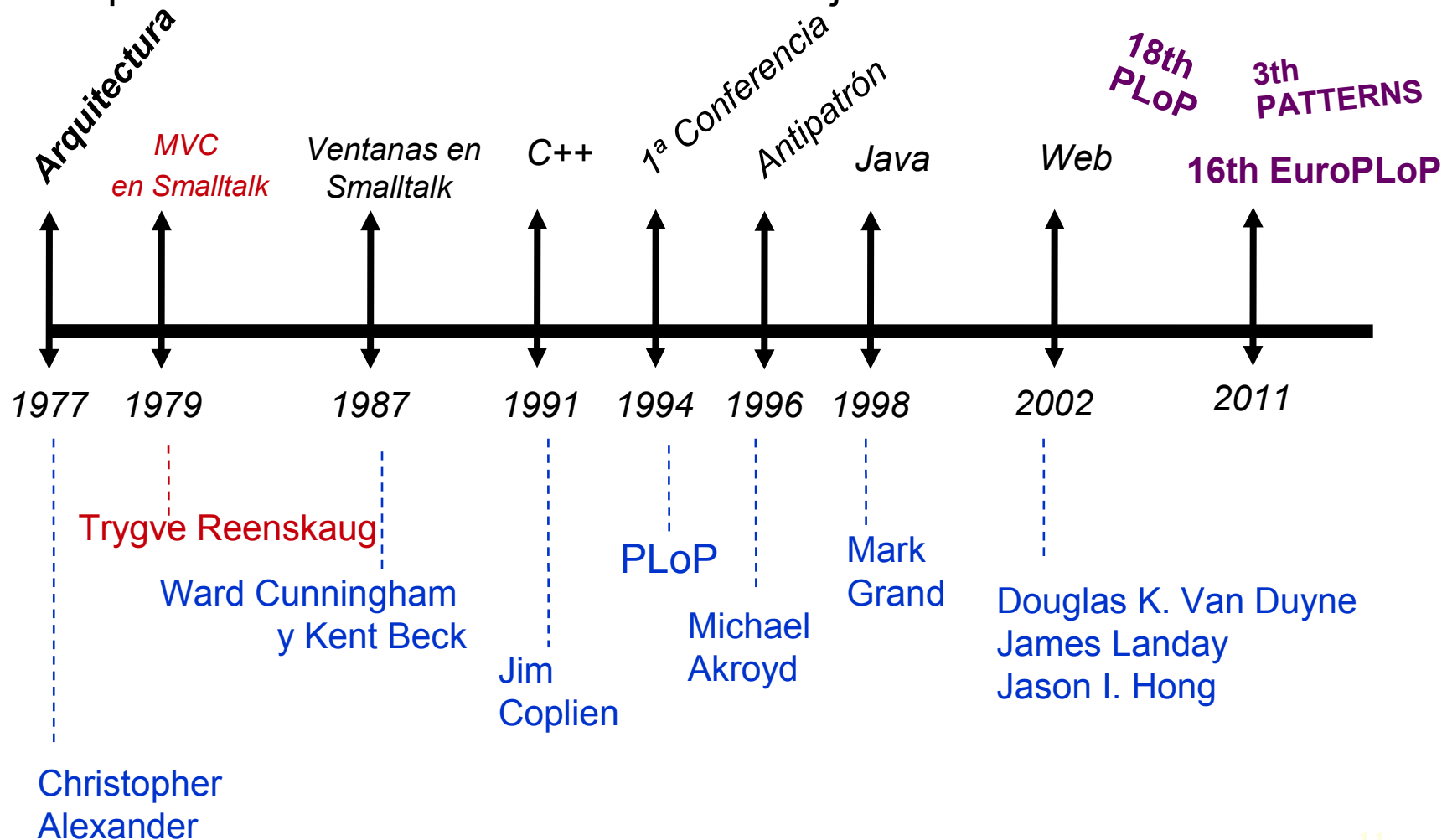
3. Concepto de patrón de diseño

- ◆ La especificación del patrón debe dejar claro: ¿cuándo?, ¿cómo? y ¿por qué? aplicar ese patrón.
- ◆ Elementos esenciales
 - Nombre
 - Problema
 - ◆ Intención
 - ◆ Motivación
 - ◆ Aplicabilidad
 - Solución
 - ◆ Estructura
 - ◆ Ejemplo
 - Consecuencias
 - ◆ Ventajas
 - ◆ Inconvenientes



3. Concepto de patrón de diseño

- El concepto de patrón rápidamente fue aplicado al diseño de aplicaciones software orientadas a objetos.




3. Concepto de patrón de diseño



◆ ¿Por qué usar patrones de diseño en orientación a objetos es útil?

- Hacer un buen diseño O.O. **no es fácil**
 - ◆ Identificar objetos pertinentes
 - ◆ Factorizarlos en clases con granularidad apropiada
 - ◆ Establecer relaciones
 - ◆ Definir una jerarquía de herencia
 - ◆ Definir interfaces
 - ◆ El diseño debe resolver el problema concreto, pero a la vez ser general
 - ◆ El diseño debe ser versátil y fácilmente mantenible
- Una experiencia previa testada y validada será siempre bienvenida

3. Concepto de patrón de diseño

- Un patrón brinda una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.
 - Efectivo
 - Reutilizable
 - Ventajas:
 - Evitar buscar soluciones a problemas frecuentes que ya fueron solucionados
 - Estandarizar la forma de elaborar el diseño de los sistemas de software
 - Formalizar un **vocabulario** común **de diseño**
 - Reutilizar código
- 

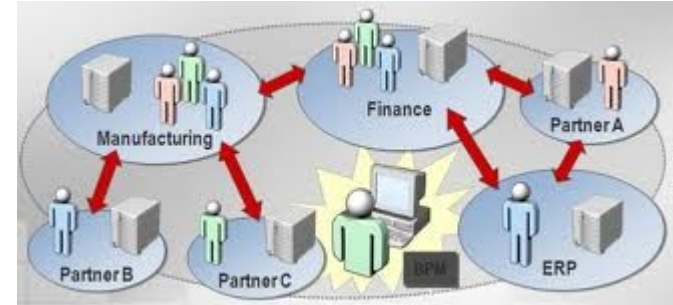


4. Concepto de patrón de arquitectura

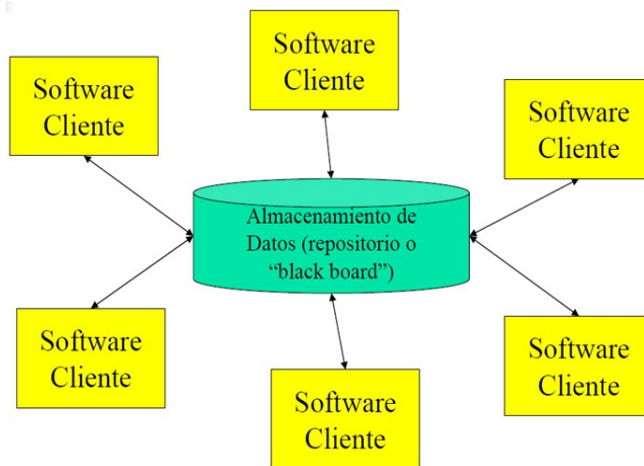
- ◆ Los **patrones de arquitectura**, también llamados patrones arquitectónicos, son patrones de diseño de software que ofrecen **soluciones a problemas de arquitectura de software**.
- ◆ Estos patrones tienen un **alto nivel de abstracción**.
 - Igual que los planos de un edificio, una arquitectura software refleja la estructura, funcionamiento e interacción entre las partes del software.
- ◆ Por lo tanto, este tipo de patrón expresa un **esquema de organización estructural**, que a menudo se divide en subsistemas, para los cuales se indican sus responsabilidades e interrelaciones.

4. Concepto de patrón de arquitectura

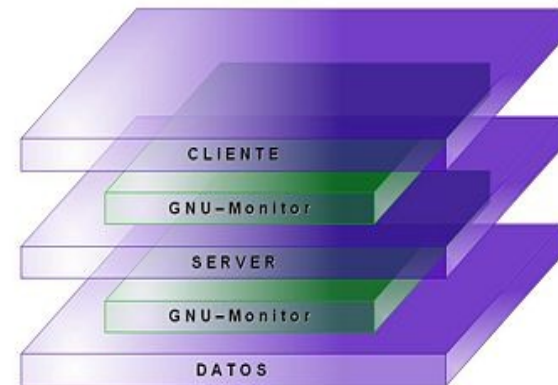
- Algunos **ejemplos** son:
 - Arquitectura en pizarra.
 - Programación por capas.
 - Arquitectura orientada a servicios.
 - Modelo Vista Controlador.**



Arquitectura de servicios



Arquitectura en pizarra

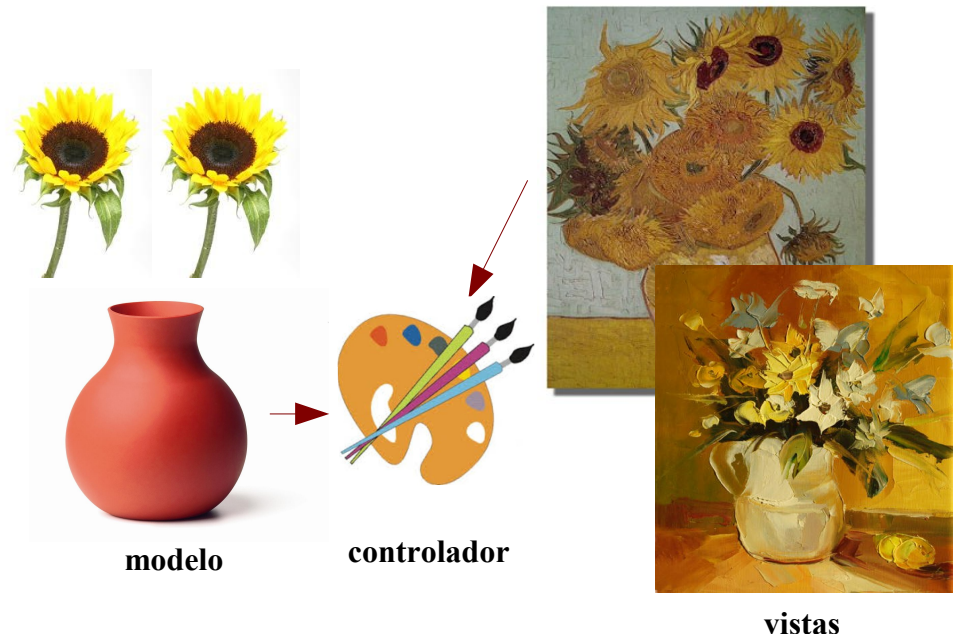


Arquitectura en capas

5. Modelo Vista Controlador (MVC)

- Origen: El patrón MVC fue descrito por primera vez en 1979, por Trygve Reenskaug, para Smalltalk.
 - Artículo original disponible en:
<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

Idea fundamental:
Separar la lógica de negocio (código del funcionamiento de una aplicación) **de la interfaz de usuario** (interfaz gráfica).



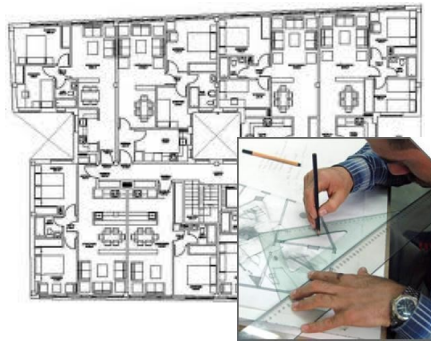
5. Modelo Vista Controlador (MVC)

◆ Elementos del patrón MVC:

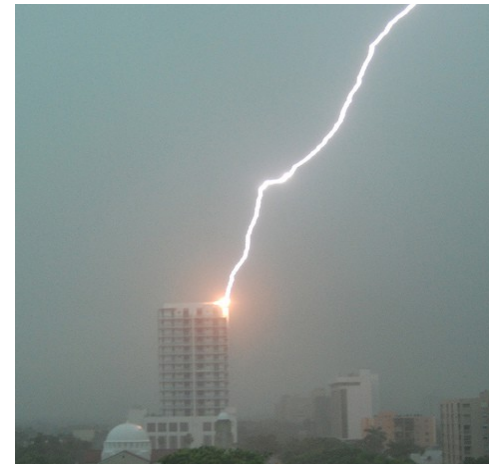
- **Modelo**: Conjunto de clases que representan la lógica de negocio de la aplicación (clases deducidas del análisis del problema). Encapsula la funcionalidad y el estado de la aplicación.
- **Vista**: Representación de los datos contenidos en el modelo. Para un mismo modelo pueden existir distintas vistas.
- **Controlador**: Es el encargado de interpretar las ordenes del usuario. Mapea la actividad del usuario con actualizaciones en el modelo. Puesto que el usuario ve la vista y los datos originales están en el modelo, el controlador actúa como intermediario y mantiene ambos coherentes entre sí.

5. Modelo Vista Controlador (MVC)

- ◆ **Supuesto 1:** Un arquitecto modifica los planos de un edificio.
Dicha alteración en el modelo se debe propagar a la vista.



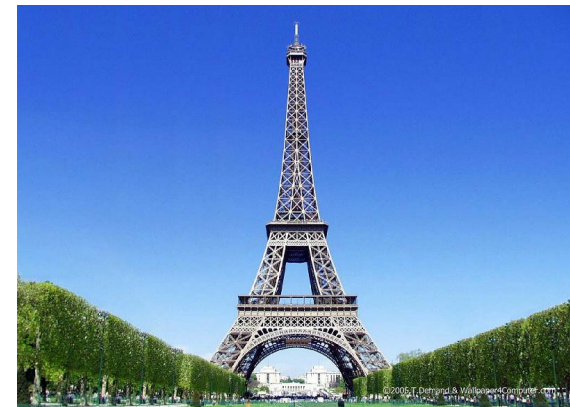
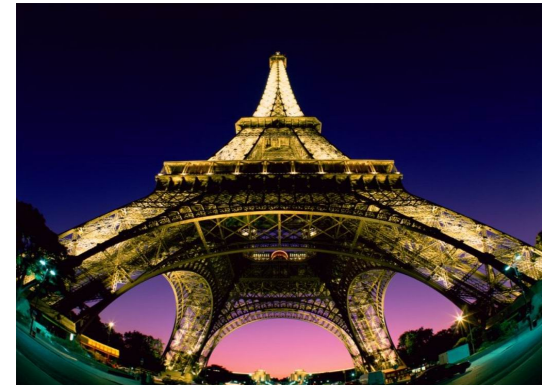
- ◆ **Supuesto 2:** Un rayo modifica la estructura de un edificio.
Dicha alteración mediante la vista debe contemplarse en el modelo.



5. Modelo Vista Controlador (MVC)

Supuesto 3: Se produce un cambio de vista.

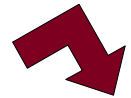
Eso no debe cambiar el modelo pero sí el controlador.



5. Modelo Vista Controlador (MVC)

► Principales beneficios:

- El desarrollo es más sencillo y limpio.
- Facilita la evolución por separado de ambos aspectos.
- ✓ Cualquier modificación que afecte al dominio de la aplicación, implica una modificación sólo en el modelo.
- ✓ Las modificaciones a las vistas no afectan al modelo de dominio, simplemente se modifica la representación de la información, no su tratamiento.
- Incrementa reutilización y flexibilidad.
- La implementación se realiza de forma modular (principio divide y vencerás).
- Las vistas siempre muestran información actualizada.



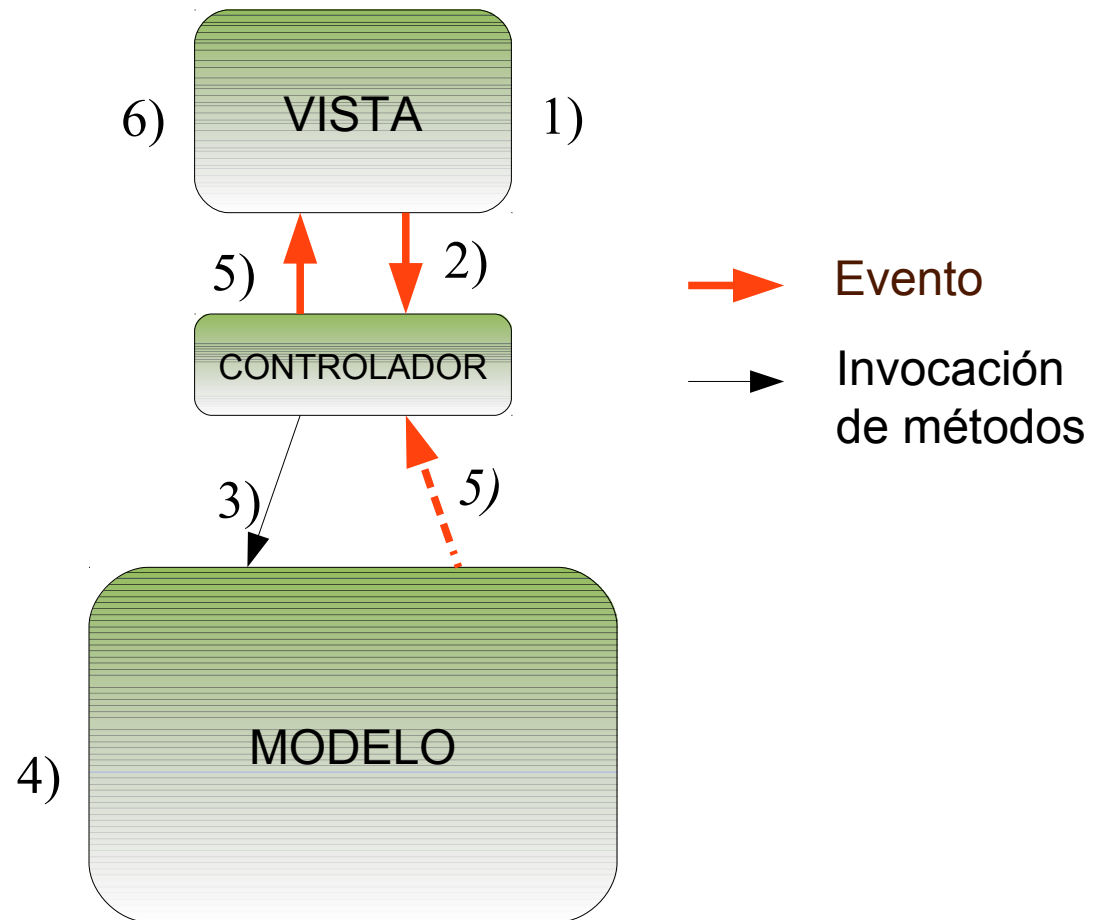
5. Modelo Vista Controlador (MVC)

◆ Flujo de control del MVC:

- 1) El usuario realiza una acción en la interfaz (vista)
- 2) El controlador detecta e interpreta el evento
- 3) Si la acción requiere alguna modificación en el modelo, el controlador inicia el cambio invocando a los métodos apropiados del modelo
- 4) El modelo cambia (notifica la realización del cambio)
- 5) El controlador notifica el cambio a las vistas
- 6) Las vistas afectadas por el cambio se actualizan

5. Modelo Vista Controlador (MVC)

◀ Flujo de control del MVC:



5. Modelo Vista Controlador (MVC)

- ◆ Ejemplo de MVC para un reloj:
 - Modelo: clases Fecha y Hora.
 - Vista: Dos vistas posibles serían un reloj analógico y un reloj digital, con todas las clases necesarias para visualizarse gráficamente.
 - Controlador: Código que asegura que ambos relojes siempre marquen la hora y fecha real, y que permitiría al usuario poner en hora cualquiera de los relojes (cambiando por tanto el modelo y la otra vista).

5. Modelo Vista Controlador (MVC)

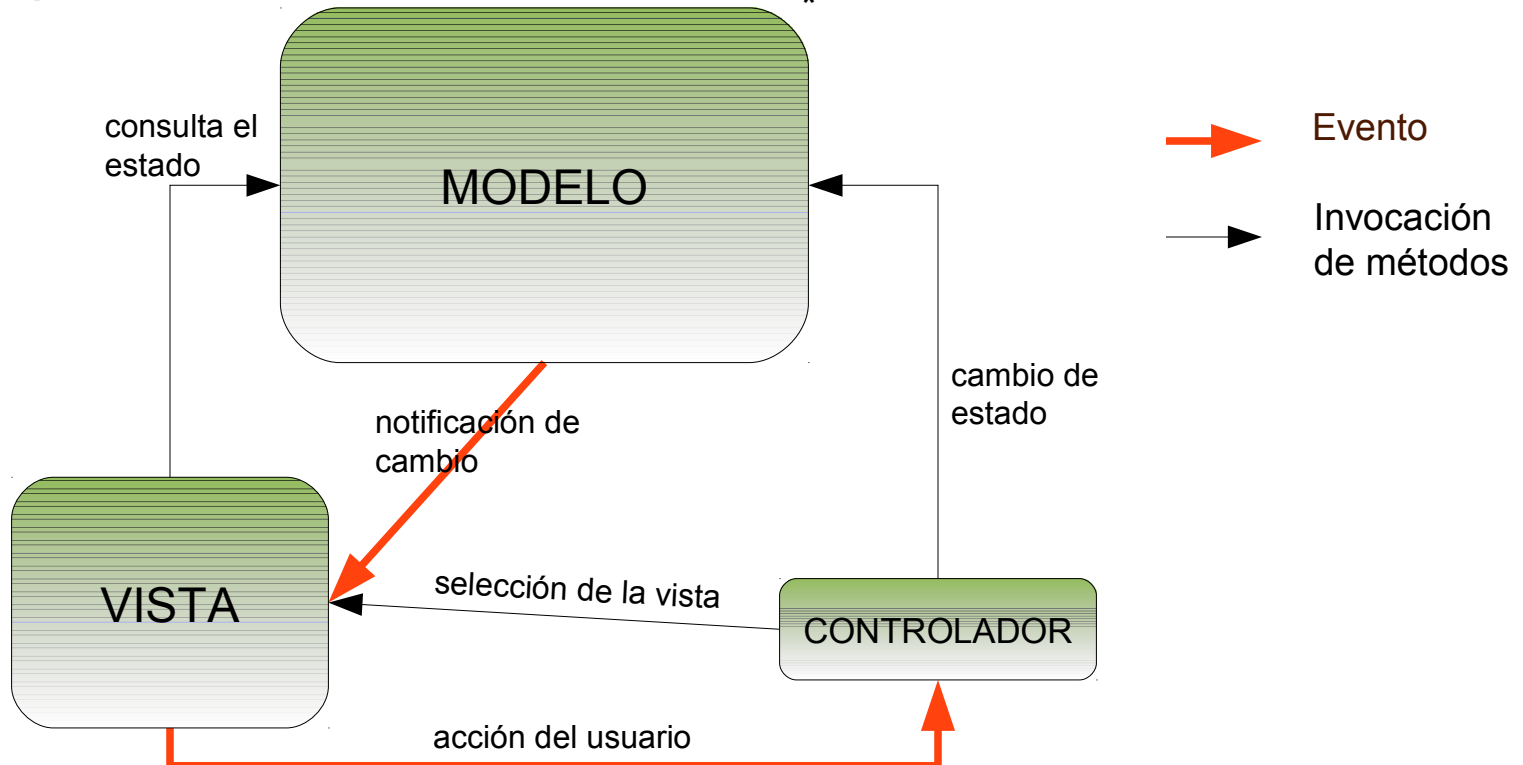
- ◆ Ejemplo de MVC para una página web:
 - Modelo: Información almacenada en el servidor.
 - Vista: Página html visualizada en el navegador.
 - Controlador: Código que procesa dinámicamente la información almacenada en el servidor para generar la página final.

5. Modelo Vista Controlador (MVC)

- ◆ Algunos frameworks donde se utiliza el MVC:
 - Java Swing
 - Java Enterprise Edition (J2EE)
 - Apache Struts (framework para aplicaciones web J2EE)
 - Ruby on Rails (framework para aplicaciones web con Ruby)
 - ASP.NET MVC Framework (Microsoft)
 - Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)

5. Modelo Vista Controlador (MVC)

- ◆ Existe cierta confusión con el MVC, ya que puede ser aplicado de formas ligeramente distintas
- ◆ Por ejemplo, en esta interpretación es el modelo el que notifica a la vista la necesidad de actualizarse



5. Modelo Vista Controlador (MVC)

- En esta otra, usada por Java SWING, se ha pasado del clásico MVC a una **arquitectura de modelo separable** (separable model architecture) en la que el controlador y la vista están fuertemente acoplados en una única clase llamada UI Object o UI Delegate:



- El elemento UI Manager es usado para almacenar información general a la apariencia de todos los componentes (color de fondo por defecto, tipo de letra por defecto, tipos de bordes, etc.), estando todos manejados por el mismo UI Manager.

¿Qué función tendrá una clase con el nombre ActionListener usada dentro de un UI Object?

