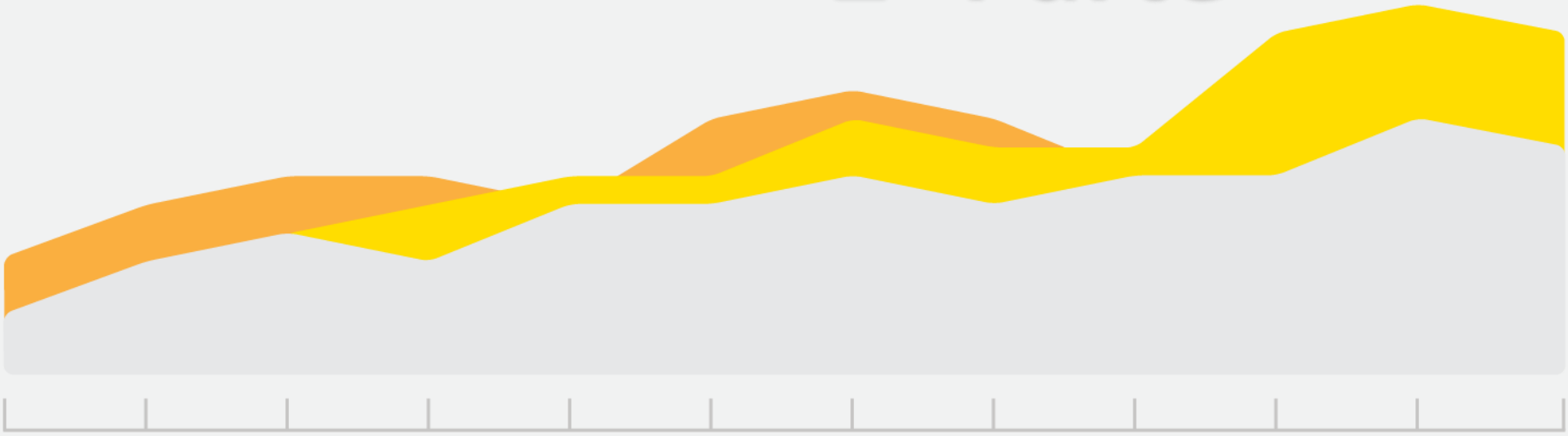
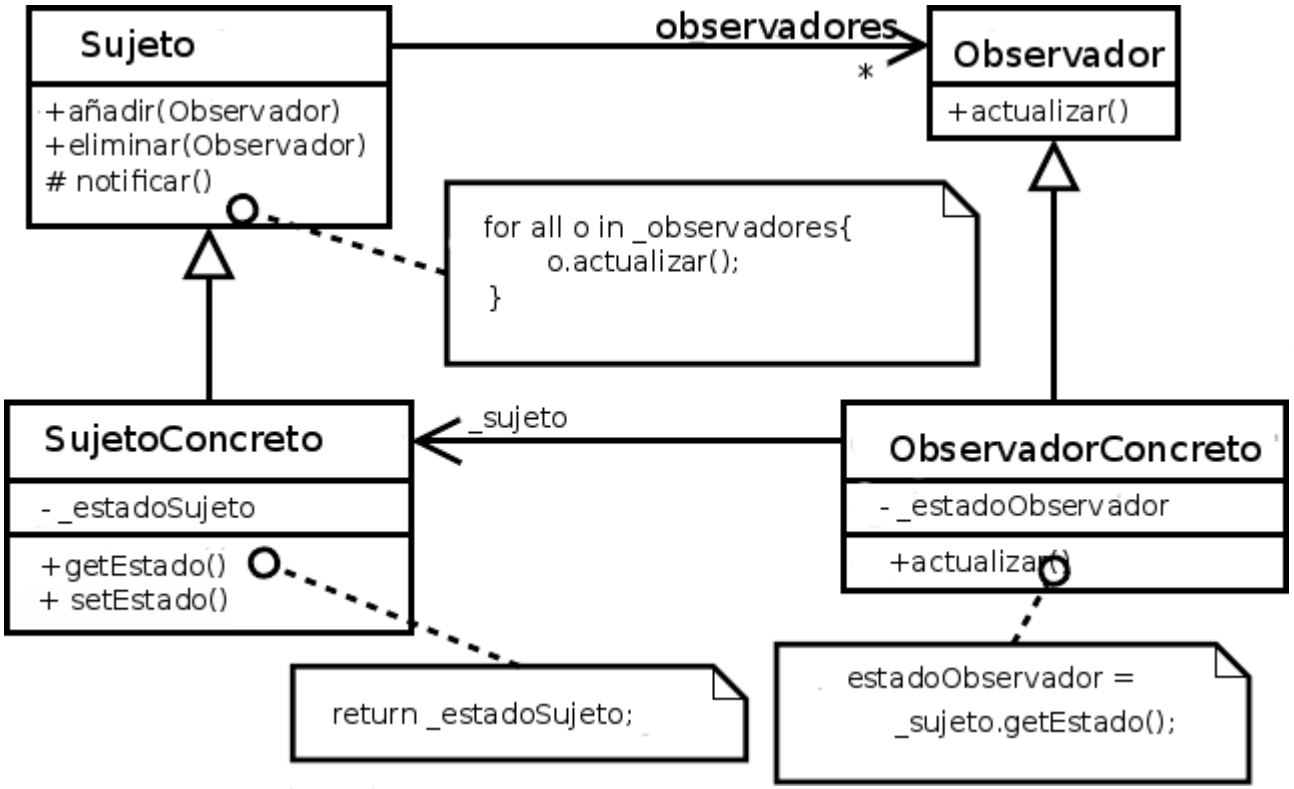
oldemar rodríguez
CONSULTOR en MINERIA DE DATOS

Patrones de Diseño Orientados a Objetos 2° Parte



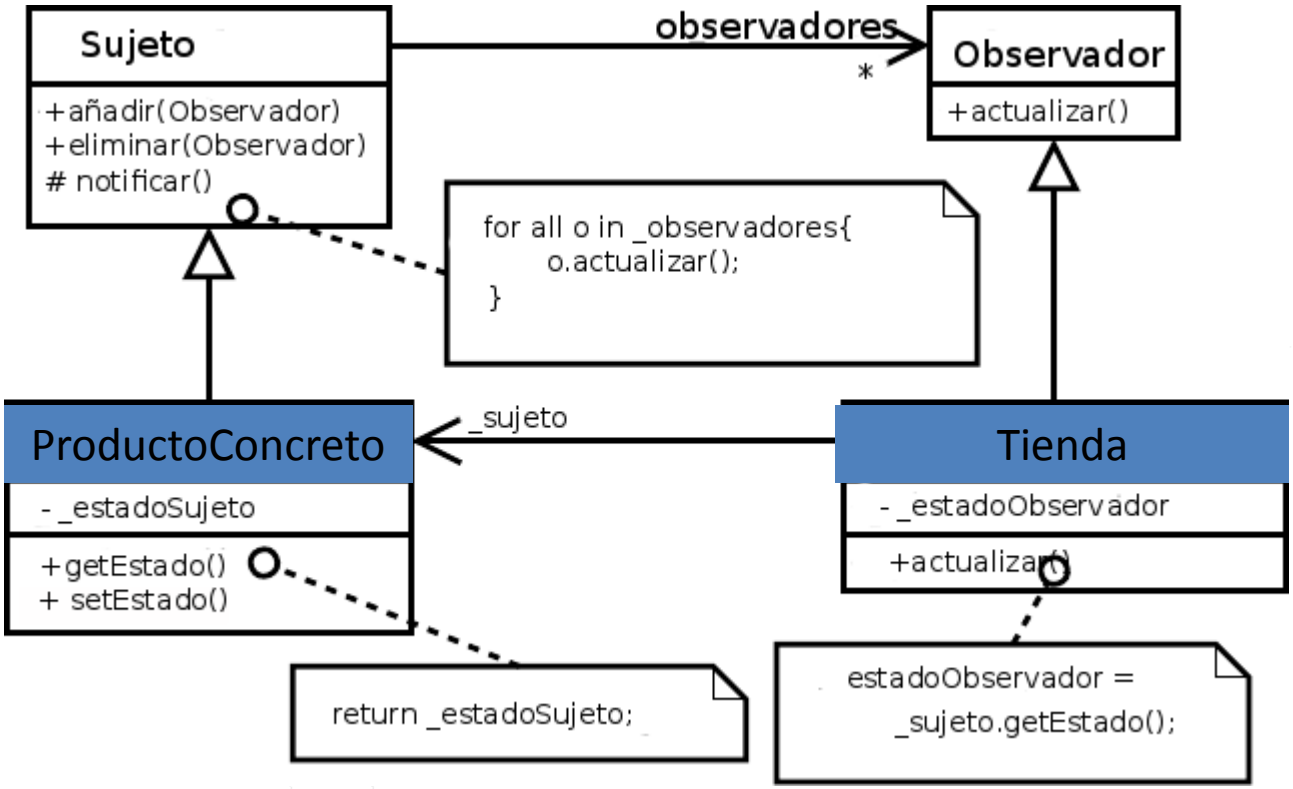
Patrón Observador – Observer (Patrón de Comportamiento)



Patrón Observador – Observer

- ***Observador*** (en inglés: Observer) es un patrón de diseño que define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.
- El patrón “Observer” es la clave del patrón de arquitectura Modelo Vista Controlador (MVC), de hecho el patrón fue implementado por primera vez en Smalltalk's MVC basado en un framework de interfaz.

Ver Ej5.CPP



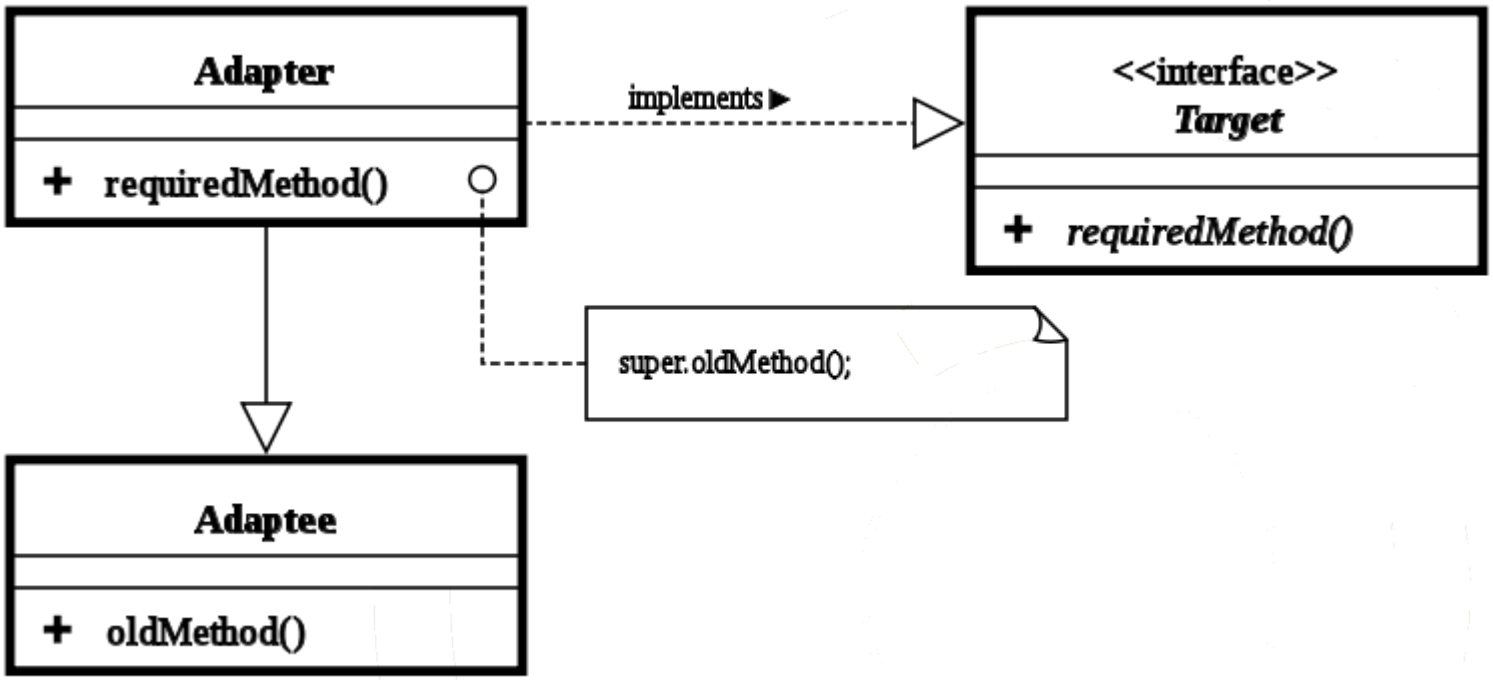
Ejercicio 6: Tarea 8

- Mediante un Patrón Observador programe las clases necesarias para que una Clase Observador denominada “Empleado” que tiene un salario y una anualidad pueda actualizar en todas sus instancias la anualidad mediante una clase denominada “AnualidadConcreta”.

Patrón Adaptador - Adapter

- ***Un Adaptador*** toma un tipo y genera una interfaz para algún otro tipo.
- El patrón Adaptador se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera, haga uso de ella a través de la segunda.
- Es útil cuando se tiene una librería o trozo de código que tiene una interfaz particular, y otra librería o trozo de código que usa las mismas ideas básicas que la primera librería, pero se expresa de forma diferente.

Patrón Adaptador - Adapter



Ver Ej6.CPP

Ejercicio 7: Tarea 8

- Si se tiene una Clase denominada TemperaturaCelcius que tiene un atributo TemperaturaActual en grados Celcuis, mediante un Patrón Adaptador programe las clases necesarias y una clase TemperaturaAdaptador que permita que la temperatura se reciba en grados **Fahrenheit** y sea adaptada adecuadamente.

Patrón Puente - Bridge

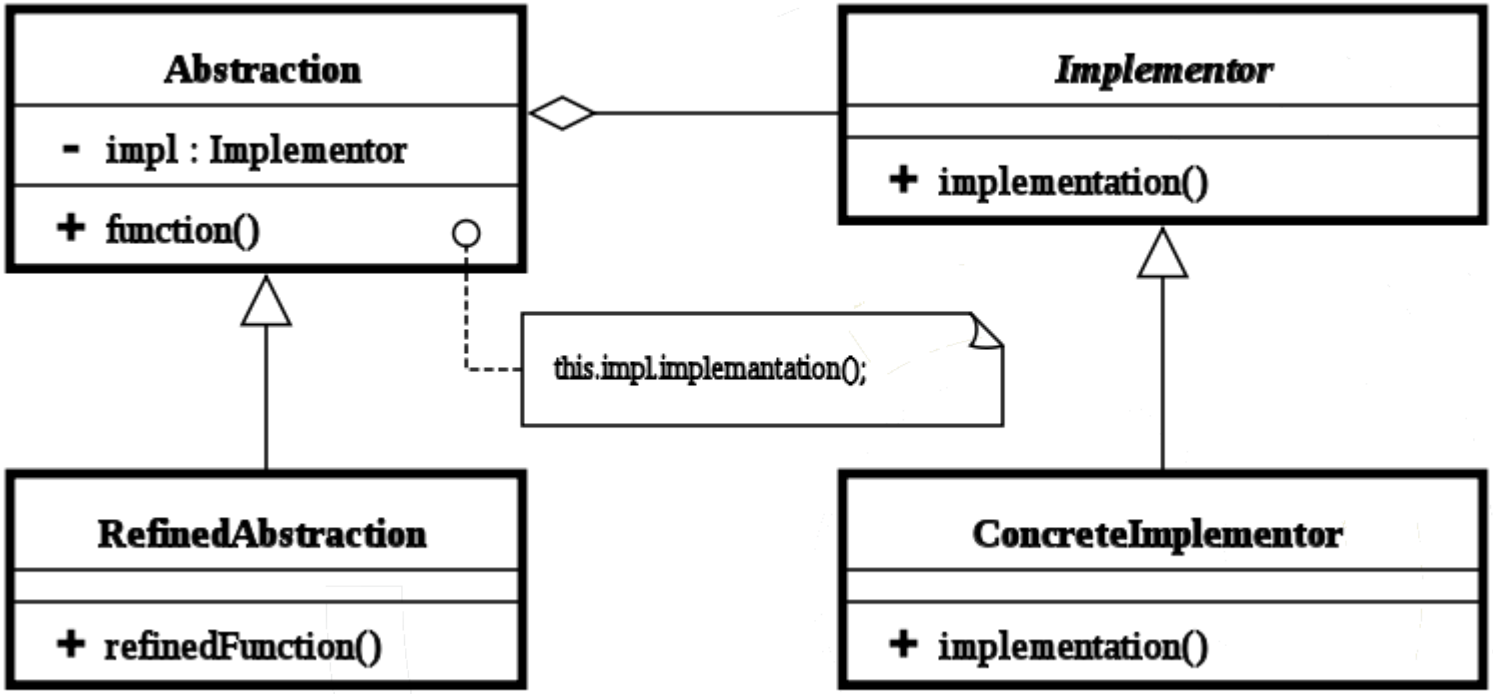
- El patrón Bridge, también conocido como Handle/Body, es una técnica usada en programación para desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente sin necesidad de alterar por ello la otra.

Patrón Puente Aplicabilidad

Se usa el patrón Bridge cuando:

- Se desea evitar un enlace permanente entre la abstracción y su implementación. Esto puede ser debido a que la implementación debe ser seleccionada o cambiada en tiempo de ejecución.
- Tanto las abstracciones como sus implementaciones deben ser extensibles por medio de subclases. En este caso, el patrón Bridge permite combinar abstracciones e implementaciones diferentes y extenderlas independientemente.
- Cambios en la implementación de una abstracción no deben impactar en los clientes, es decir, su código no debe tener que ser recompilado.
- En C++ se desea esconder la implementación de una abstracción completamente a los clientes. En C++, la representación de una clase es visible en la interface de la clase.
- Se desea compartir una implementación entre múltiples objetos (quizá usando contadores), y este hecho debe ser escondido a los clientes.

Patrón Puente - Bridge



Ver Ej7.CPP

Ejercicio 8: Tarea 8

- Mediante el Patrón Puente implemente Clases Abstractas y Clases Implementadoras para las clases Persona, Estudiante y EstCompu vistas en clase (la versión simple sin archivos ni listas).

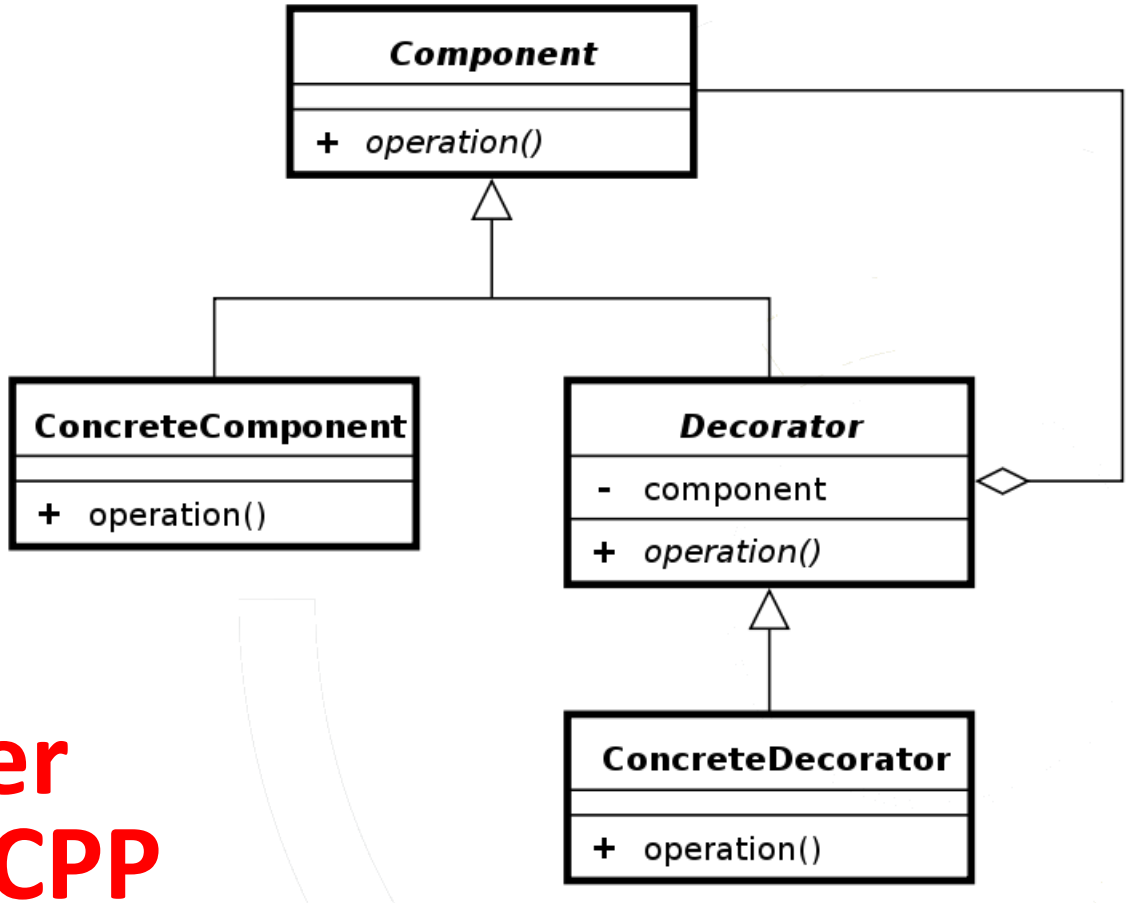
Patrón Decorador - Decorator

- El ***patrón Decorador*** responde a la necesidad de añadir dinámicamente funcionalidad a un Objeto.
- Esto nos permite no tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.

Patrón Decorador - Aplicabilidad

- Añadir objetos individuales de forma dinámica y transparente.
- Responsabilidades de un objeto pueden ser retiradas.
- Cuando la extensión mediante la herencia no es viable.
- Hay una necesidad de extender la funcionalidad de una clase, pero no hay razones para extenderlo a través de la herencia.
- Existe la necesidad de extender dinámicamente la funcionalidad de un objeto y quizás quitar la funcionalidad extendida.

Patrón Decorador - Decorator



Ver Ej8.CPP

Ejercicio 9: Tarea 8

- Suponga tenemos una Clase denominada “DespliegaTexto” que tiene un atributo tipo String para almacenar un texto. Mediante el Modelo Decorador programe las clases necesarias para que esta clase DespliegaClase texto pueda imprimir dicho texto en negrilla y en itálica (por ahora la impresión en negrilla o itálica simplemente significa imprimir entre paréntesis estas palabras al final del texto).

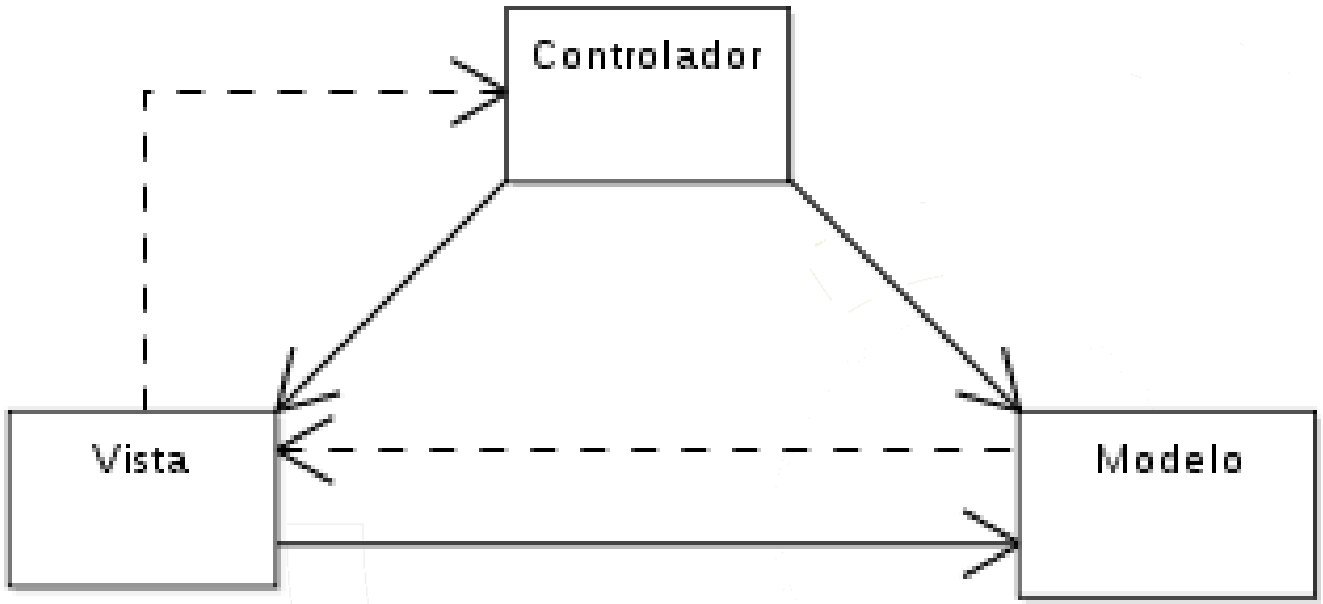
Modelo Vista Controlador (MVC)

- SmallTalk en fue el primer lenguaje de programación que permitió diseñar interfaces de usuario con múltiples “ventanas” desplegadas en una misma pantalla, concepto que después fue aplicado por GEMS, Macintosh, X11, Windows y otras interfaces gráficas de usuario modernas.
- El concepto central detrás de las librerías de interfaz de usuario provistas por SmallTalk está basado en el patrón de diseño MVC, creado por el profesor Trygve Reenskaug

Modelo Vista Controlador (MVC)

- MVC es un patrón de diseño que considera dividir una aplicación en tres módulos claramente identificables y con funcionalidad bien definida: ***El Modelo, las Vistas y el Controlador***.

Modelo Vista Controlador (MVC)



El Modelo

- El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar.
- Así, por ejemplo, un sistema de administración de datos climatológicos tendrá un modelo que representará la temperatura, la humedad ambiental, el estado del tiempo esperado, etc. sin tomar en cuenta ni la forma en la que esa información va a ser mostrada ni los mecanismos que hacen que esos datos estén dentro del modelo, es decir, sin tener relación con ninguna otra entidad dentro de la aplicación.

El Modelo

- En teoría el Modelo desconoce la existencia de las vistas y del controlador. Ese enfoque suena interesante, pero en la práctica no es aplicable pues deben existir interfaces que permitan a los módulos comunicarse entre sí.
- SmallTalk sugiere que el modelo en realidad esté formado por dos sub-módulos: ***El modelo del dominio*** y ***el modelo de la aplica***ción.

Modelo del Dominio

- Se podría decir que el ***modelo del dominio*** (o el modelo propiamente dicho) es el conjunto de clases que un ingeniero de software modela al analizar el problema que desea resolver.
- Así, pertenecerían al modelo del dominio: El cliente, la factura, la temperatura, la hora, etc.
- El modelo del dominio no debería tener relación con nada externo a la información que contiene.

Modelo de la Aplicación

- El ***modelo de la aplicación*** es un conjunto de clases que se relacionan con el modelo del dominio, que tienen conocimiento de las vistas y que implementan los mecanismos necesarios para notificar a éstas últimas sobre los cambios que se pudieren dar en el modelo del dominio.
- El modelo de la aplicación es llamado también ***Coordinador de la Aplicación***.

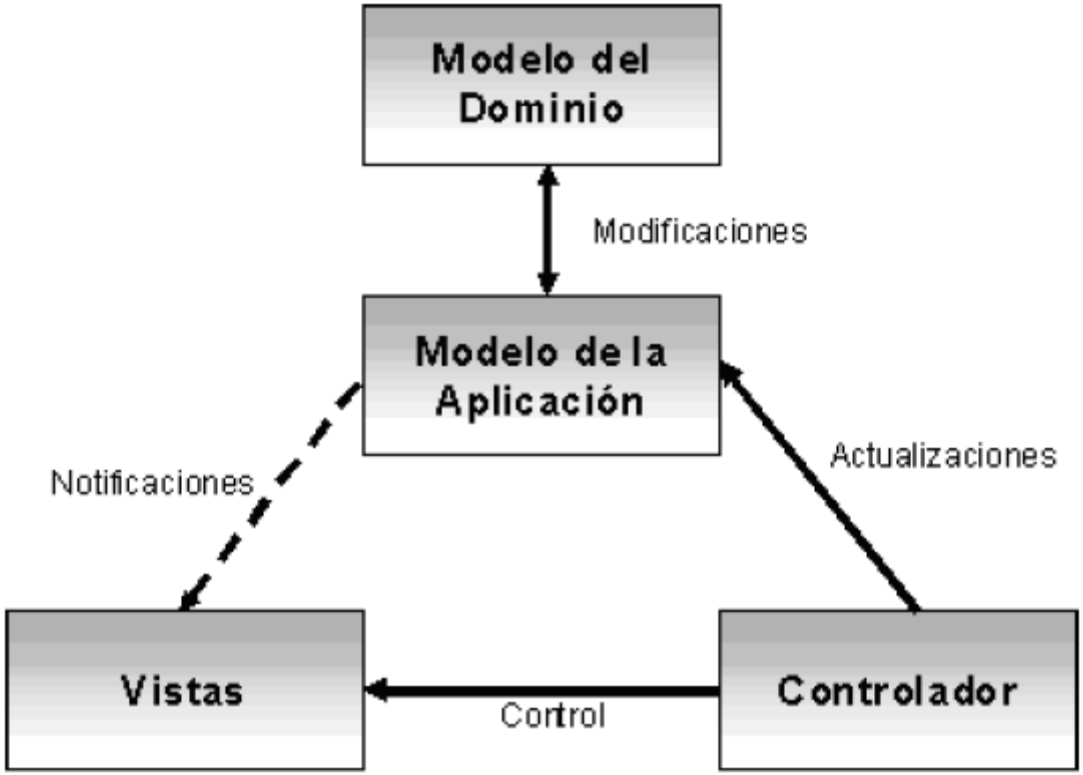
Las Vistas

- ***Las vistas*** son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo.
- Por ejemplo, se puede tener una vista mostrando la hora del sistema como un reloj analógico y otra vista mostrando la misma información como un reloj digital.

El Controlador

- ***El controlador*** es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas.
- El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador.

Modelo Vista Controlador (MVC)
(Programación por Capas)



Un ejemplo

- Tomemos como ejemplo una aplicación hecha para almacenar y procesar los datos de las elecciones.
- ***El modelo*** del dominio sería bastante simple: Un conjunto de votos, un conjunto de mesas y un conjunto de Distritos. Cada voto almacenaría la selección hecha por el votante y la mesa donde emitió su voto. Cada mesa contendría información sobre el lugar de votación y el distrito donde estaría ubicada.

Un ejemplo

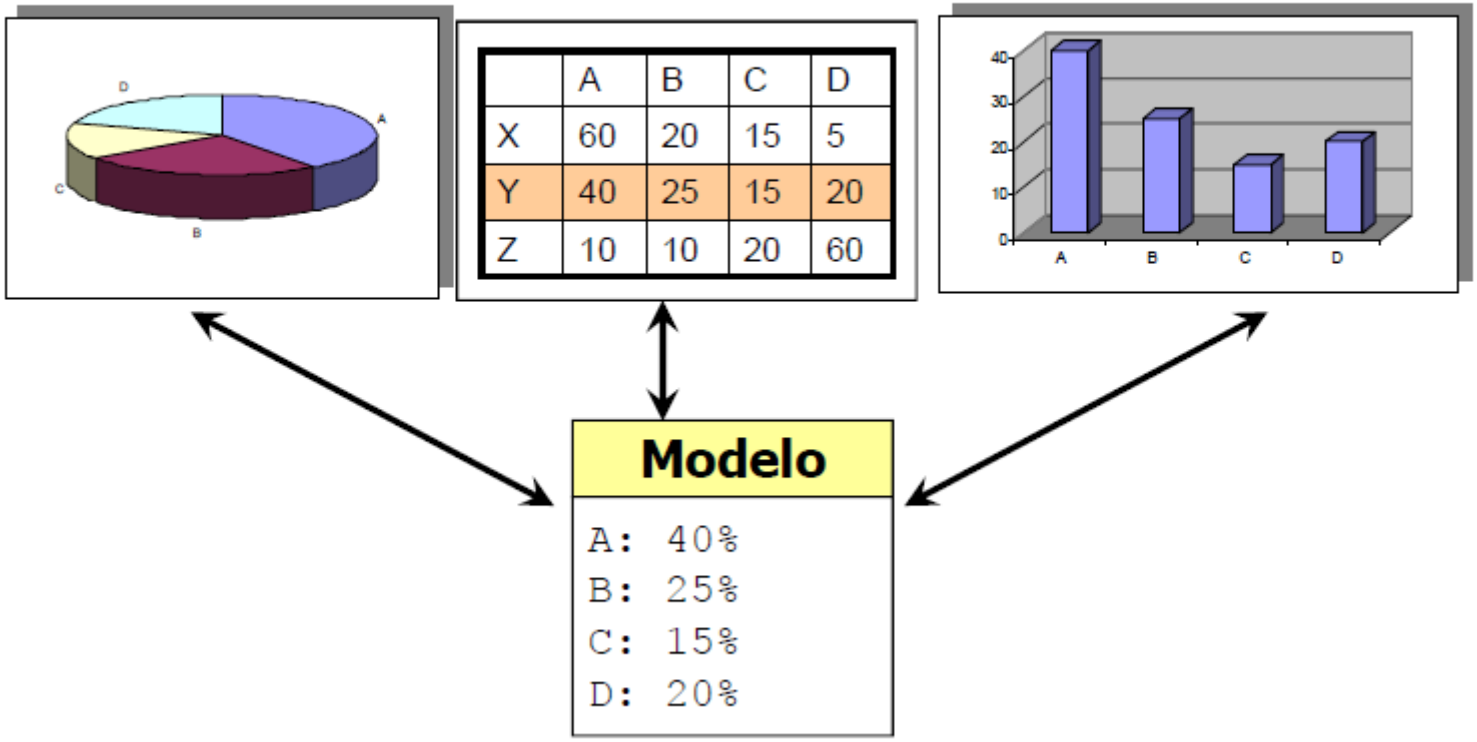
- El ***conjunto de vistas*** sobre el modelo también sería sencillo: Se podría obtener un gráfico estadístico de votos por distrito, cantón o provincia en barras, otro gráfico de votos totales en una tabla, el conjunto de votos totales en barras o en “pie”, etc.
- Como se puede ver, aunque todas las vistas estarían mostrando la información de diferente manera, todas estarían asociadas al mismo modelo del dominio.

Un ejemplo

- ***El controlador*** se encargaría de mostrar las vistas que el usuario desea ver y de permitir al usuario introducir información de votos.
- Si el usuario desea ver una vista, el controlador crea la vista solicitada, esta vista obtendría la información necesaria del modelo y la desplegaría. Si el usuario aumentara la información de votos al sistema, el controlador se encargaría de actualizar la información contenida en el modelo del dominio que, al ser modificado, anunciaría al modelo de la aplicación la existencia de cambios y éste notificaría a todas sus vistas asociadas para que se actualicen.
- De esta manera, las vistas estarían siempre actualizadas mostrando exactamente la misma información contenida en el modelo.

Un Ejemplo

Modelo/Vista/Controlador.



Gracias….

oldemar rodríguez

CONSULTOR en M1NƐR14 DƐ D4T0S