

**UNIVERSIDAD NACIONAL DE INGENIERIA**

**CURSO: Programación Con Java**

**TEMA: PATRON MVC**

**ALUMNA: Montejo Rocca Beatriz Victoria**

**PROFESOR: Coronel Castillo Eric Gustavo**

**- 2020 -**

# **DEDICATORIA**

El presente trabajo está dedicado a aquellas personas que de una u otra manera me brindan su apoyo incondicional. A mis padres y a mi pareja, gracias por su tiempo y cuidados, con lo que me ayudan en cada paso que doy, y al profesor Coronel que nos ha compartido el conocimiento necesario para culminar satisfactoriamente este curso.

# **Índice**

[**DEDICATORIA** 2](#_Toc31046537)

[**Índice** 3](#_Toc31046538)

[**Resumen** 4](#_Toc31046539)

[**Definición** 5](#_Toc31046540)

[**Historia** 6](#_Toc31046541)

[**¿Por qué utilizar MVC?** 7](#_Toc31046542)

[**Componentes** 7](#_Toc31046543)

[**Interacción de los componentes** 9](#_Toc31046544)

[**Tipos de Patrones MVC** 10](#_Toc31046545)

[**MVC y bases de datos:** 11](#_Toc31046546)

[**Frameworks:** 11](#_Toc31046547)

[**Ejemplo de uso del MVC:** 13](#_Toc31046548)

[**Conclusión:** 20](#_Toc31046549)

[**Recomendación:** 20](#_Toc31046550)

[**Referencias:** 21](#_Toc31046551)

# **Resumen**

Las personas o entes dedicados al desarrollo y el mantenimiento de productos de software, constantemente, buscan optimizar las metodologías para obtener buenos resultados. Una de las soluciones para mejorar el proceso de desarrollo ha sido la arquitectura basada en “capas” que separan el código de un programa en función de sus responsabilidades. Esta separación también incluye a los responsables del desarrollo del software, tal es el caso de los diseñadores, quienes podrían no estar familiarizados con determinadas partes de un código que les competen a los programadores, por lo que necesitan separarse de ciertos módulos de programación.

Esto muestra la necesidad de aplicar una arquitectura útil como es el Modelo Vista-Controlador (MVC), que requiere de la separación del código de un programa en capas atendiendo a sus responsabilidades.

En este trabajo tratara de dar a conocer más sobre este tema brindando algunos conceptos y ejemplos.

# **Definición**

El Modell View Controller o Modelo Vista Controlador (MVC) es un patrón de diseño de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos de forma que las modificaciones al componente de la vista, o a cualquier parte del sistema puedan ser hechas con un mínimo impacto en el componente del modelo de datos o en los otros componentes del sistema. Este patrón cumple perfectamente el cometido de modularizar un sistema.

Controlador

Vista

Modelo

*Diagrama sencillo que muestra la relación entre el modelo, la vista y el controlador.*

***Nota****: las líneas sólidas indican una asociación directa, y las punteadas una indirecta.*

"El propósito de este patrón es simplificar la implementación de aplicaciones de acuerdo a las peticiones de los usuarios y los datos a desplegar" [Harrop, 2005].

# **Historia**

El Modelo Vista Controlador (MVC) Surge de la necesidad de crear software más robusto con un ciclo de vida más adecuado, donde se potencie la facilidad de mantenimiento, reutilización del código y la separación de conceptos.

El patrón MVC fue una de las primeras ideas en el campo de las interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones.

Fue descrito por primera vez en 1979 por Trygve Reenskaug, entonces trabajando en Smalltalk en laboratorios de investigación de Xerox. Para los años 70 seguidamente en los años 80 Jim Althoff y otros implementaron una versión de MVC para la biblioteca de clases de Smalltalk-80. Solo más tarde, en 1988, MVC se expresó como un concepto general en un artículo sobre Smalltalk-80.

En esta primera definición de MVC el controlador se definía como "el módulo que se ocupa de la entrada" (de forma similar a como la vista "se ocupa de la salida"). Esta definición no tiene cabida en las aplicaciones modernas en las que esta funcionalidad es asumida por una combinación de la 'vista' y algún framework moderno para desarrollo. El 'controlador', en las aplicaciones modernas de la década de 2000, es un módulo o una sección intermedia de código, que hace de intermediario de la comunicación entre el 'modelo' y la 'vista', y unifica la validación (utilizando llamadas directas o el "observer" para desacoplar el 'modelo' de la 'vista' en el 'modelo' activo).

# **¿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.

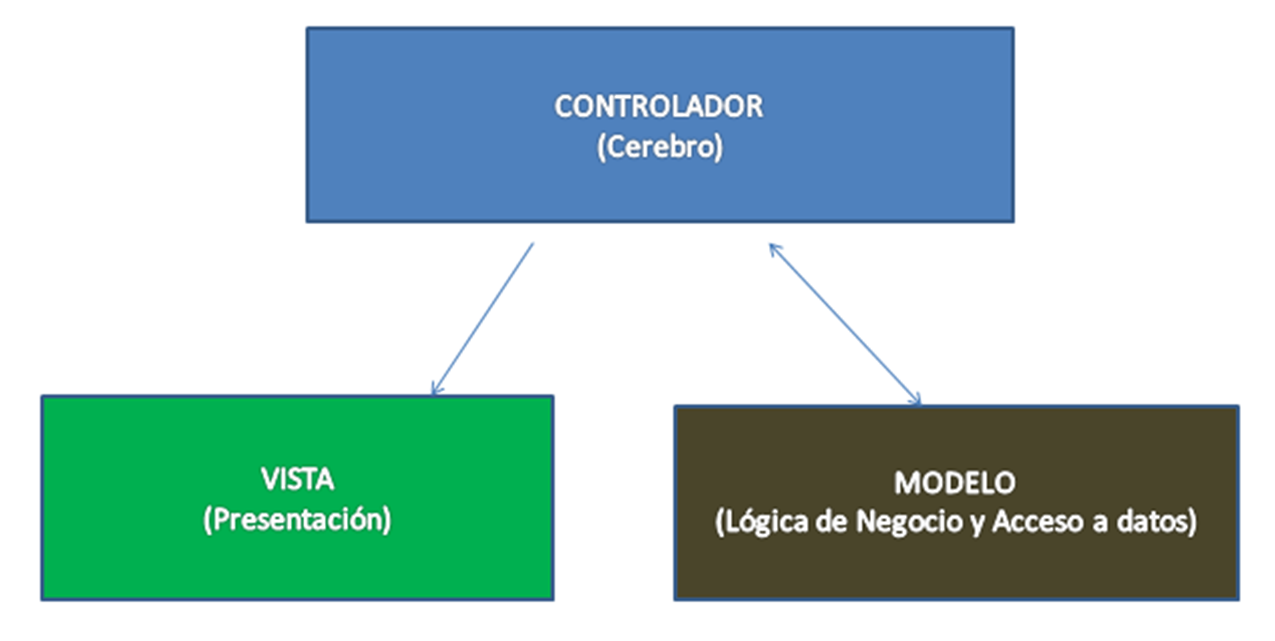
# **Componentes**

Los tres principales componentes del patrón MVC son:

* **Modelo:** Representa los datos que el usuario está esperando ver, en algunos casos el Modelo consiste de Java Beans. 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. 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).

* **Vista:** Se encarga de transformar el modelo para que sea visualizada por el usuario, ya sea un archivo de texto normal o en una página Web (HTML o JSP) que el navegador pueda desplegar. El propósito de la Vista es convertir los datos para que el usuario le sean significativos y los pueda interpretar fácilmente; la Vista no debe trabajar directamente con los parámetros de *request*, debe delegar esta responsabilidad al controlador.
* Controlador: Es la parte lógica que es responsable de procesamiento y comportamiento de acuerdo a las peticiones (*request*) del usuario, construyendo un modelo apropiado, y pasándolo a la vista para su correcta visualización. En el caso de una aplicación Web Java en la mayoría de los casos el Controlador es implementado por un servlet.



*Una típica colaboración entre los componentes de un MVC*

## **Interacción de los componentes**

Se pueden encontrar muchas implementaciones de MVC, pero generalmente el flujo de datos se describe así:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.).
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario. Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo. El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio.
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

# **Tipos de Patrones MVC**

Actualmente existen dos tipos de patrón MVC:

* MVC tipo 1: Las paginas JSP están en el centro de aplicación y contiene tanto la lógica de control como la de presentación. Este tipo de arquitectura funciona de la siguiente manera: el cliente hace una petición o una página JSP, se construye la lógica de la página, generalmente en objetos java y se transforma el modelo para ser desplegado una vez más.

Servidor Web/Aplicación

2

Navegador

JSP

Java Beans

Capa de Servicios

Datos

1. Request

4. Responde

3

* MVC tipo 2: Aquí ya existe una clara separación entre el Controlador y el Vista, ya que ahora es directamente el Controlador quien recibe la petición, prepara el modelo y lo transforma para que sea desplegado en la vista. Esta arquitectura se utiliza para aplicaciones complejas.

3

Navegador

Capa de Servicios

Datos

Controlador

Vista

Modelo: Java Beans

1. Request

4/5 Responde

4. Usa

2. Crea

3. Accesa

Servidor Web/Aplicación

# **MVC y bases de datos:**

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos que debe utilizar la aplicación; en líneas generales del MVC dicha gestión corresponde al modelo. La unión entre capa de presentación y capa de negocio conocido en el paradigma de la Programación por capas representaría la integración entre la Vista y su correspondiente Controlador de eventos y acceso a datos, MVC no pretende discriminar entre capa de negocio y capa de presentación pero si pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el Controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

# **Frameworks:**

Un framework es un diseño re-usable de un sistema, se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta, refiriendo a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación.

Este término es utilizado en la computación en general, para referirse a un conjunto de bibliotecas utilizadas para implementar la estructura estándar de una aplicación. Todo esto es realizado con el propósito de promover la reutilización de código, con el fin de ahorrarse trabajo al desarrollador al no tener que rescribir ese código para cada nueva aplicación que se desea crear. Existen varios Frameworks para diferentes fines, algunos son orientados para aplicaciones web, otros para aplicaciones multiplataforma, sistemas operativos, etc.

Este determina la arquitectura de una aplicación, se encarga de definir la estructura general, sus particiones en clases y objetos, responsabilidades clave, así como la colaboración entre las clases objetos, esto evita que el usuario tenga que definirlo y se pueda enfocar en cosas específicas de su aplicación.

Los Frameworks utilizan un variado número de patrones de diseño, ya que así logran soportar aplicaciones de más alto nivel y que reutilizan una mayor cantidad de código, que uno que no utiliza dichos patrones. "Los patrones ayudan hacer la arquitectura de los Frameworks más adecuada para muchas y diferentes aplicaciones sin necesidad de rediseño". Por esta razón es importante que se documenten que patrones utiliza el FrameWorks para que los que se encuentren familiarizados con dichos patrones puedan tener una mejor visión y poder adentrarse en el FrameWorks más fácilmente.

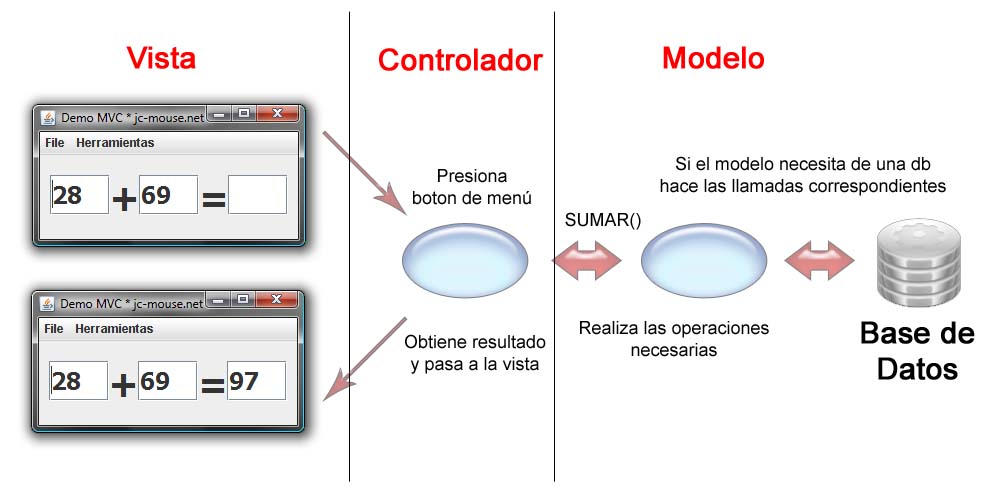
El patrón MVC es utilizado en múltiples Frameworks como:

* Java Swing, Java Enterprise Edition
* XForms (formato XML estándar para la especificación de un modelo de proceso de datos XML e interfaces de usuario como formularios web).
* GTK+ (escrito en C, toolkit creado por Gnome para construir aplicaciones gráficas, inicialmente para el sistema X Window)
* Google Web Toolkit, Apache Struts, Ruby On Rails, entre otros.



# **Ejemplo de uso del MVC:**

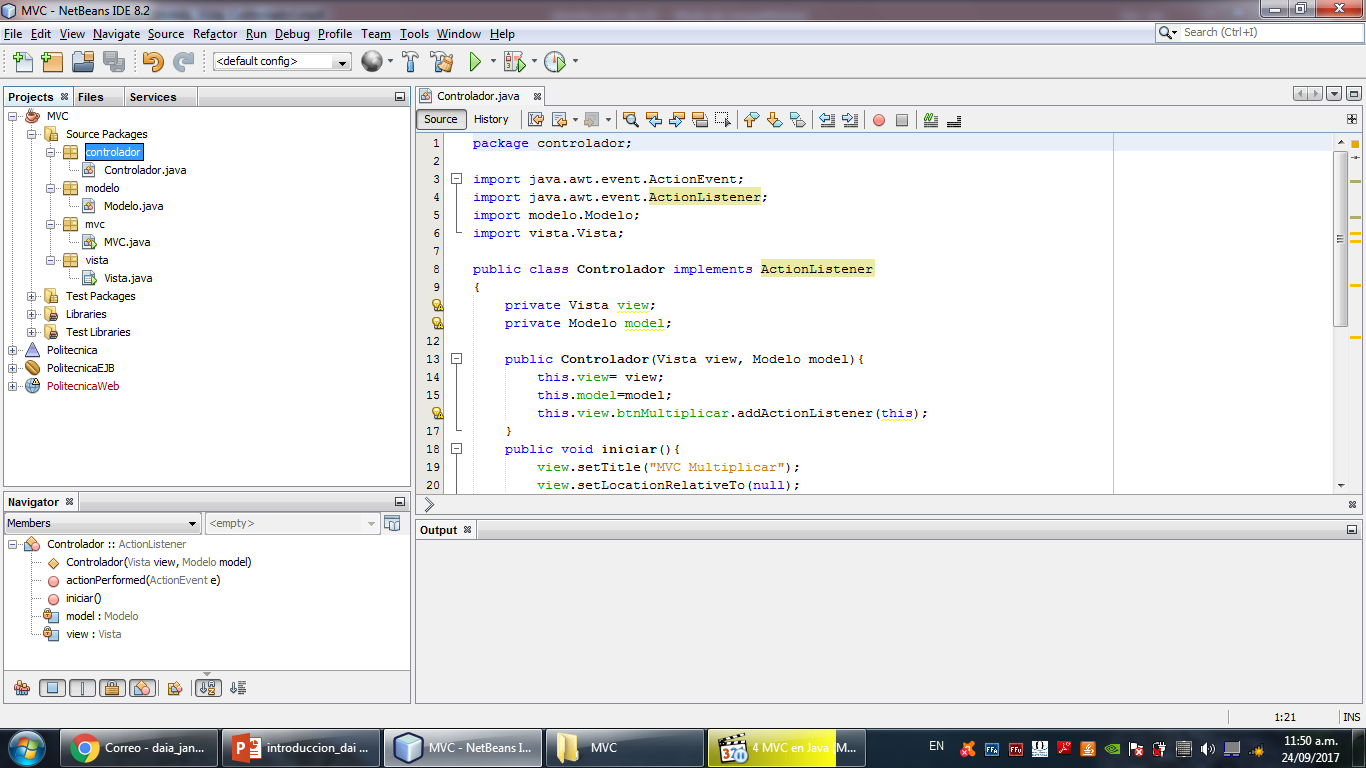
***Calculadora con Java Swing***



* 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:**

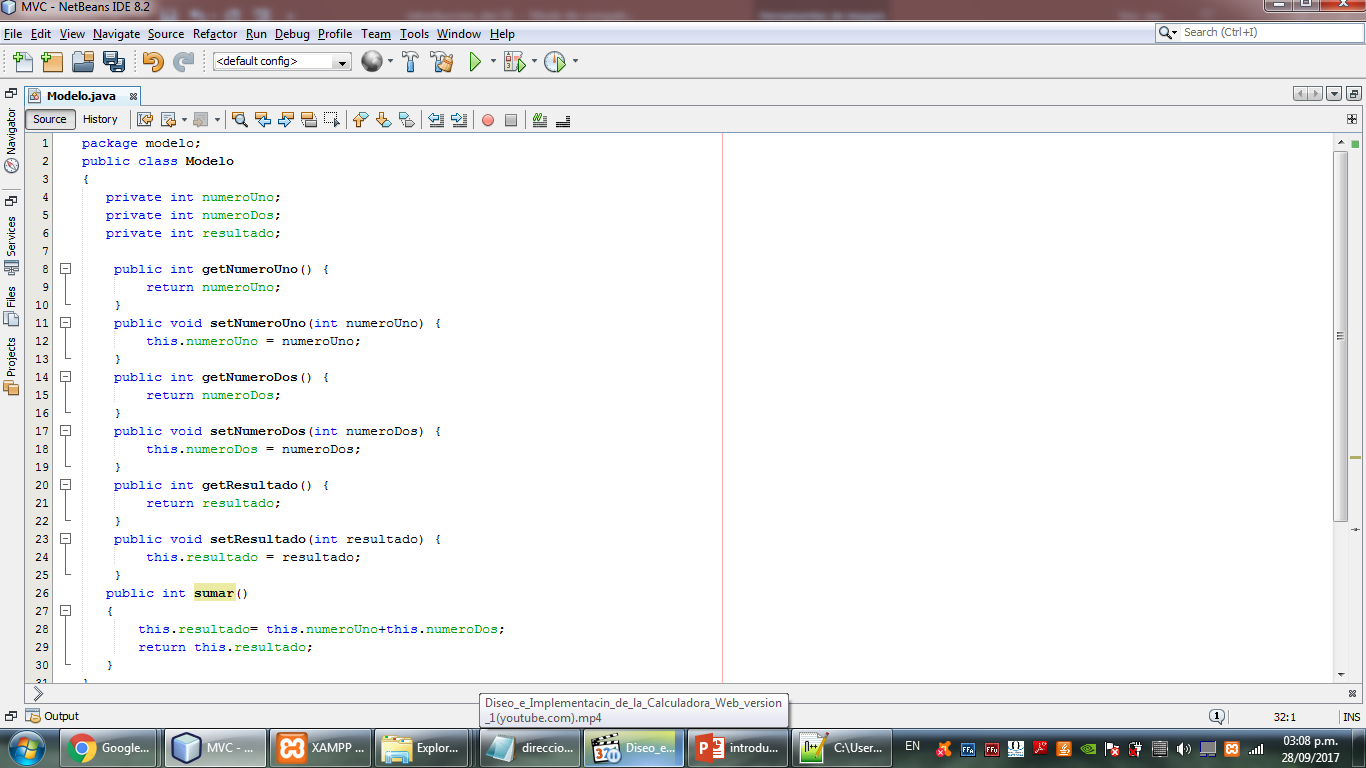
1. Creación del proyecto



Como se puede 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".

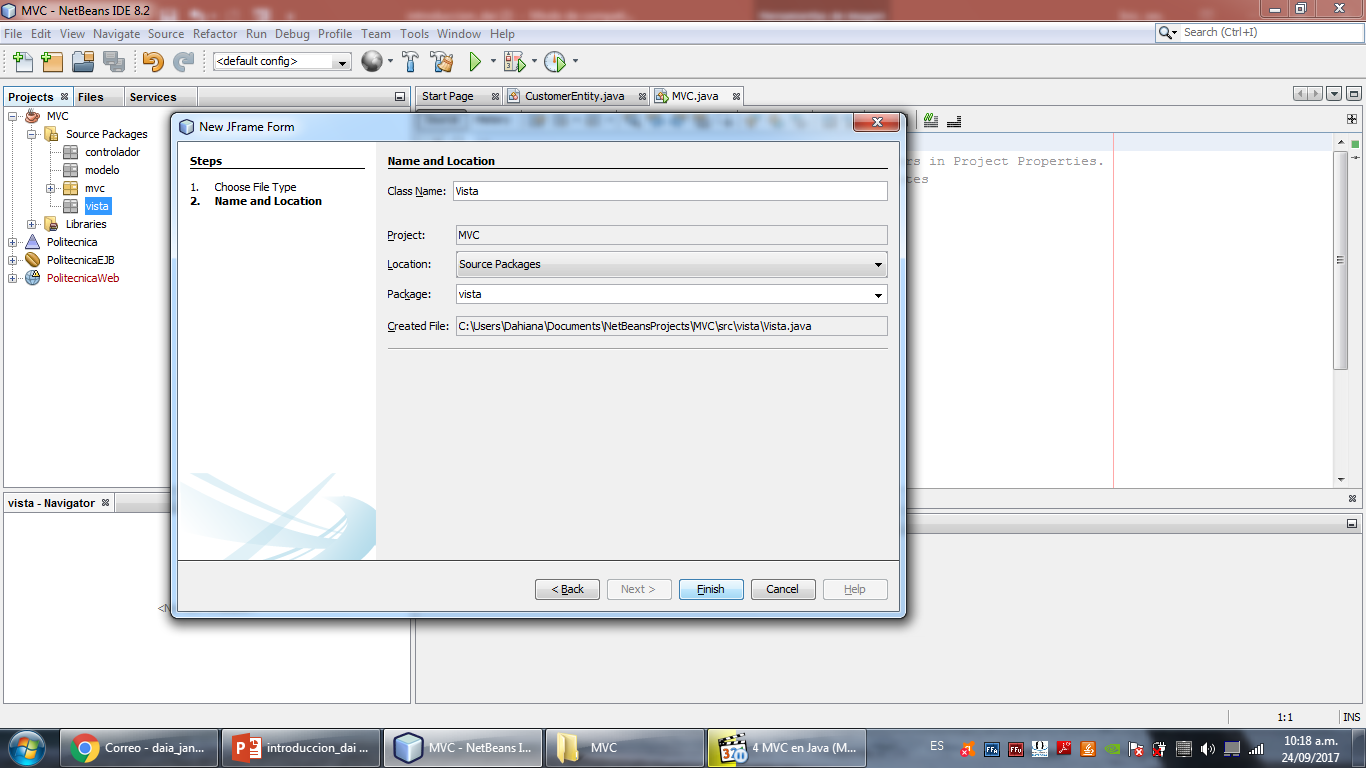
1. Creación del modelo

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



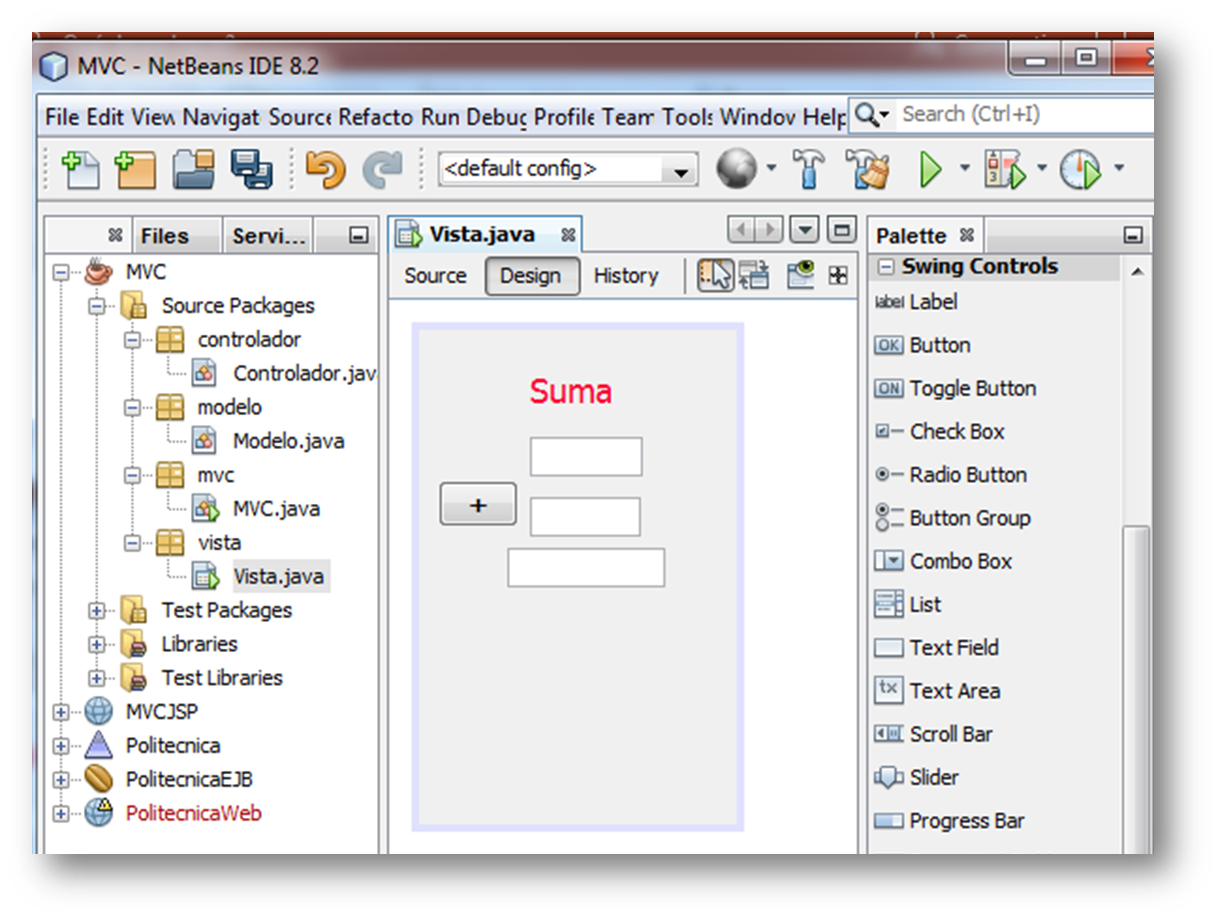
1. Codificación de la clase Vista

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



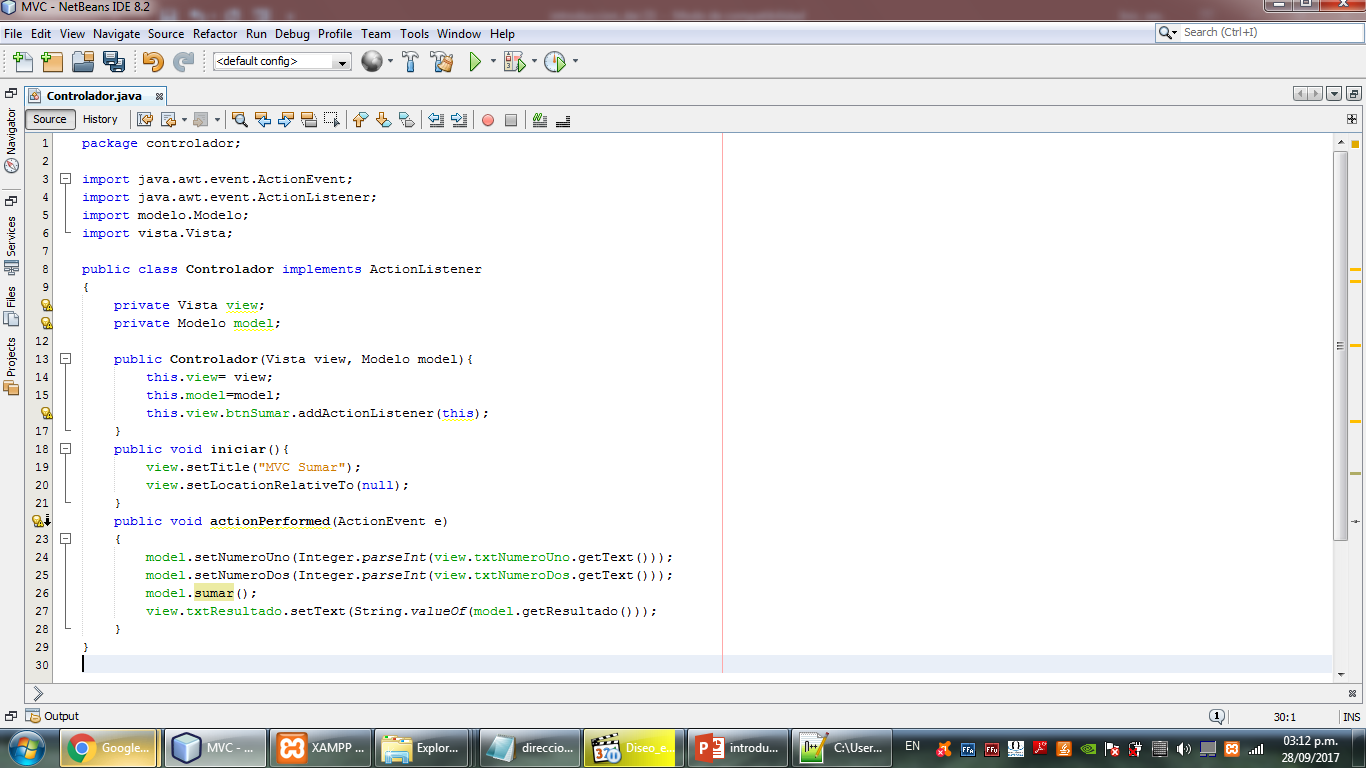
En Java existe una biblioteca gráfica (**Componentes Swing**) la cual incluye widgets para la interfaz gráfica de usuario (cajas de texto, botones, menús entre otros.).

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.



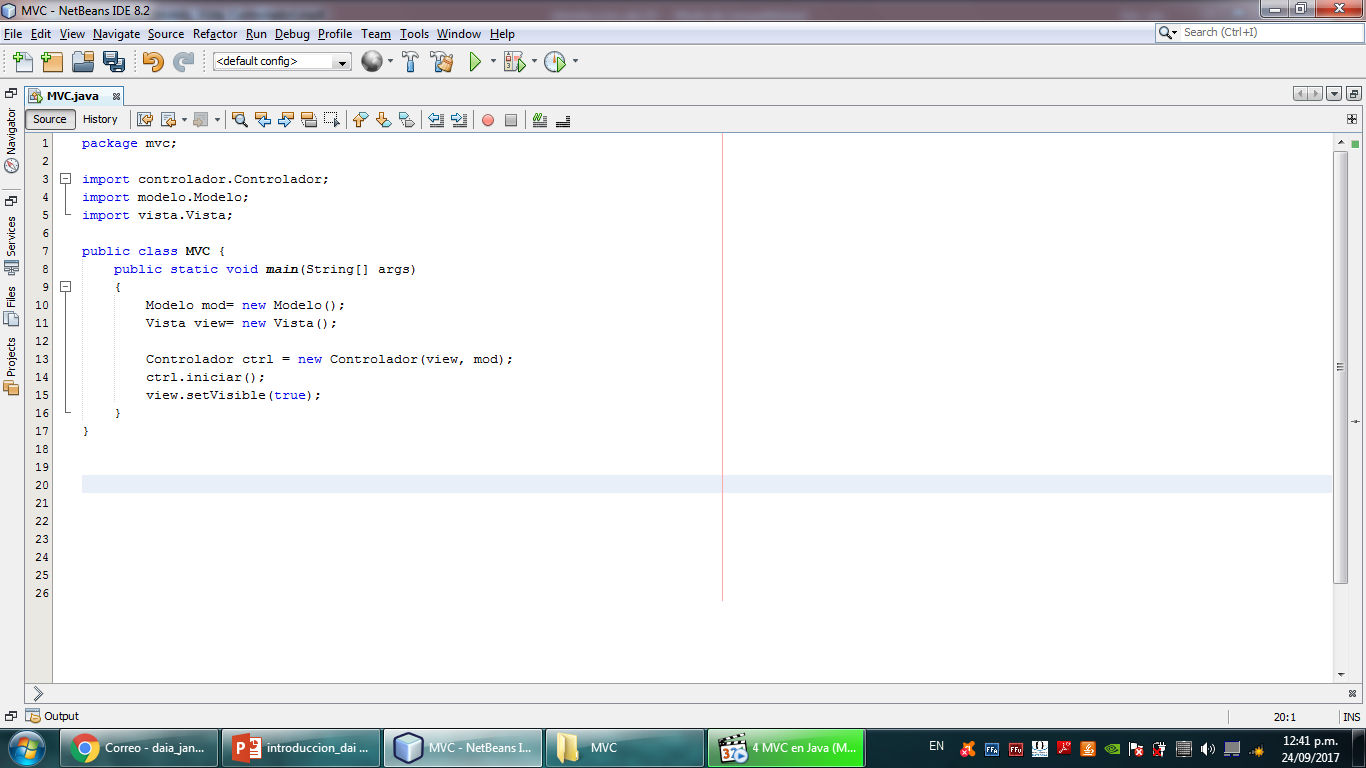
1. Codificación de la clase Controlador

Se crea una nueva clase en el paquete controlador, “Controlador.java" y añade el siguiente código.

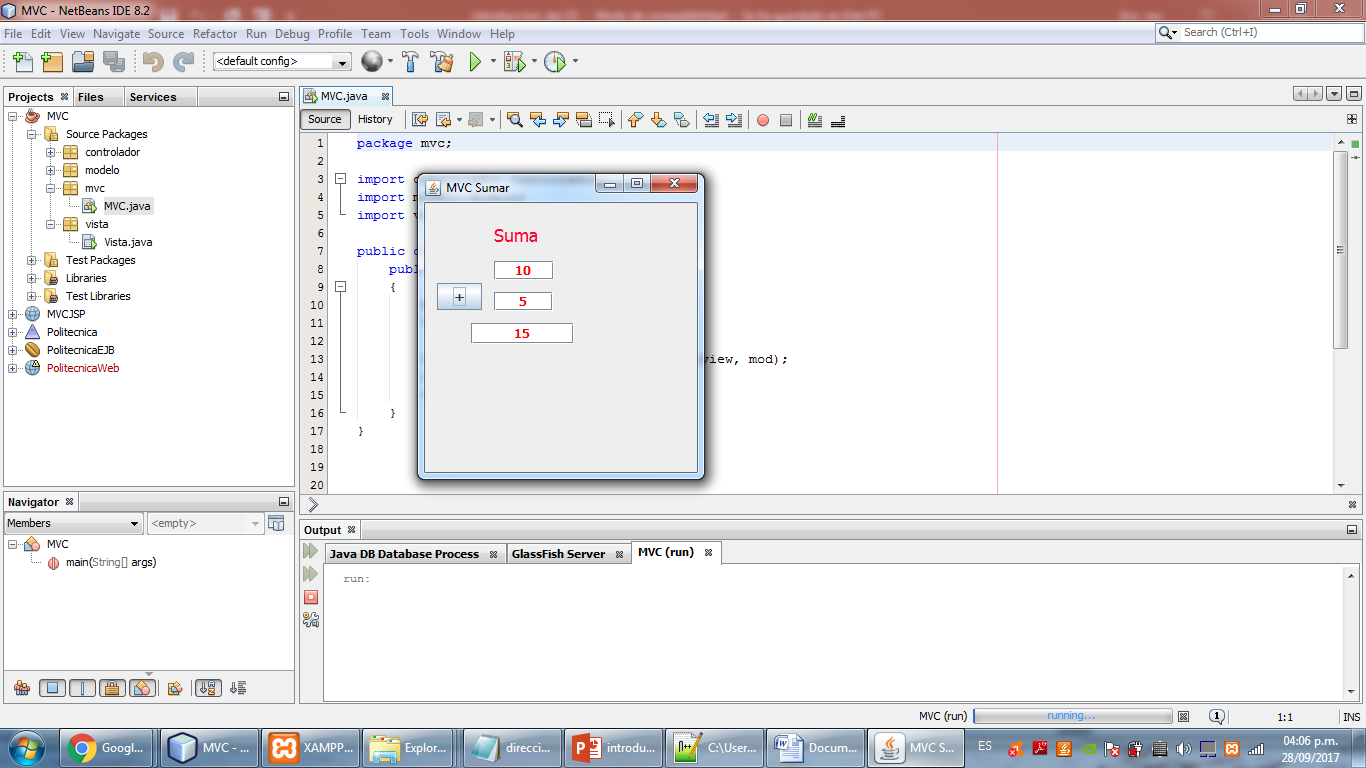


1. Codificación de la clase Principal

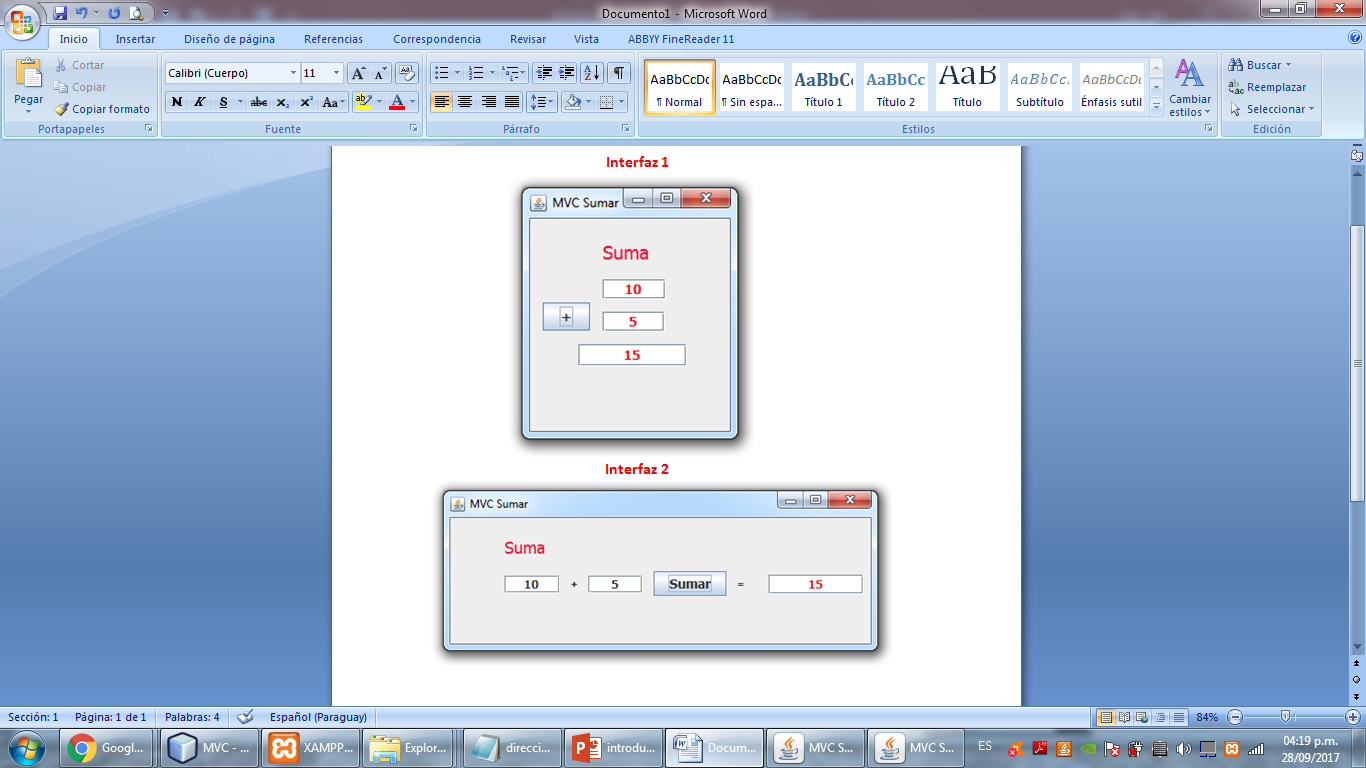
Abrimos la clase principal, creada por defecto, MVC.java y añadimos el siguiente código.



1. Prueba del proyecto MVC



**Resultado**



**Breve explicación:**

1. Nuestra clase controlador, implementa el ActionListener, esto para responder desde esta clase, los eventos realizados desde la interfaz (VISTA).
2. El constructor de la clase pasa como parámetros, la clase VISTA y la clase MODELO.
3. 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.
4. El método action performed captura el evento realizado desde la interfaz.
5. Un CLICK EN EL BOTON **+** (sumar), obtiene los datos correspondientes e invoca al modelo para procesar la información y obtener una respuesta.

# **Conclusión:**

Este patrón para el diseño y desarrollo permite producir software de alta calidad, convirtiendo una aplicación en un módulo fácil de mantener y mejora la rapidez de su desarrollo.

La separación de capas en modelos, vistas y controladores hace que las aplicaciones sean fáciles de entender.

La separación en capas también permite a los responsables, modificar partes del código de la aplicación sin afectar a otras, por ejemplo, si se desea cambiar el diseño de una página o aplicación basta con modificar las vistas y nada más. No hay preocupaciones acerca de “dañar” el modelo (o lógica de negocio) ya que la misma no se modifica.

# **Recomendación:**

Los patrones de diseño son buenas alternativas de solución para problemas comunes y recurrentes que nos encontramos cuando desarrollamos un software, a pesar que nuestra aplicación sea única, esta tendrá algunas partes comunes con otras aplicaciones como, por ejemplo: acceso a datos, creación de objetos con una sola instancia, integraciones con otras aplicaciones, etc. Todo este conjunto de problemas comunes lo podemos solucionar usando los respectivos patrones de diseño, ya que son soluciones probadas que garantizan su funcionamiento.

Aunque los patrones de diseño son muy útiles en el desarrollo de software, no siempre es necesario usarlos y eso no significa que la aplicación esté “mal diseñada”, algunas veces no son necesarios y mucho menos obligatorios, a veces abusar de los patrones de diseño puede traer más problemas de los que resuelve.

# **Referencias:**

* Pavón, Juan. (2009) Estructura de las Aplicaciones Orientadas a Objetos.
* El Patrón Modelo-Vista-Controlador (MVC). Universidad Complutense Madrid.
* Sommerville, Ian (2005). Ingeniera del Software. Pearson Educación.

**Web**

* <http://www.lab.inf.uc3m.es/~a0080802/RAI/mvc.html>
* <http://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>https://desarrolloweb.com/articulos/que-es-mvc.html
* http://jc-mouse.blogspot.com/2011/12/patron-mvc-en-java-con-netbeans.html
* https://es.coursera.org/learn/interfaz-ios/lecture/ACvwq/patron-de-diseno-mvc
* http://www.ecodeup.com/patrones-de-diseno-en-java-mvc-dao-y-dto/
* https://javaweb.osmosislatina.com/curso/mvc.htm