TEMA 2

MANEJO DE **CONECTORES**

CONTENIDO

1 – INTRODUCCIÓN

Acceso a datos: Proceso de recuperación o manipulación de datos extraídos de un origen de datos local o remoto.

Los orígenes de datos pueden provenir de muchas fuentes. Ejemplos: Bases de datos, fichero de texto, hoja de cálculo, etc…

Conectores: Software necesario para poder coonectar el programa y una base de datos.

2 – EL DESFASE OBJETO-RELACIONAL

Bases de datos orientados a objetos: La información se representa mediante objetos como lo que ocurre en la programación orientada a objetos.

Base de datos relacional: Permite establecer unas interconexiones entre los datos. Los datos están estructurados en tablas. Las interconexiones o relaciones nos permiten relacionar los datos de las distintas tablas.

CARACTERÍSTICAS DE LAS BASES DE DATOS RELACIONALES

- Una misma base de datos se compone de distintas tablas y por ello interconexiones.
- No podrán existir dos tablas con el mismo nombre de registro.
- Cada tabla está compuesta por columnas (campos) y filas (registro).

Actualmente van ganando más peso frente a las relacionales ya que son más ágiles, solucionan las necesidades de las aplicaciones más sofisticadas que requieren el tratamiento de elementos más complejos. Ejemplos: Sistemas de información geográfica, Aplicaciones para fabricación y diseño en ingeniería.

Las bases de datos relacionales no están diseñadas para almacenar objetos ya que existe un desfase entre las construcciones típicas de las bases de datos relacionales y las proporcionadas por los ambientes de la programación basada en objetos. Esto es lo que se denomina desfase objeto relacional o desajuste de la impedancia. Con esto nos referimos a los problemas que existen debido a las diferencias entre el modelo de bases de datos relacionales (como tablas) y el de lenguajes de programación orientado a objetos.

Estos dos paradigmas pueden llegar a convivir ("pueden ser amigos pero no novios"). Cada vez que los objetos deban extraerse en una base de datos relacional, se requiere un mapeo entre una librería.

3 – BASES DE DATOS EMBEBIDAS

Cuando desarrollamos pequeñas aplicaciones donde no vamos a tener que almacenar una gran cantidad de datos, no es necesario que utilicemos un sistema gestor de bases de datos como Oracle y MySQL. En su lugar podemos usar una base de datos embebida. En ella el motor está incrustado en la misma aplicación y su uso es exclusivo para ella.

La base de datos se inicia cuando se arranca la aplicación y finaliza cuando se cierra la misma.

Generalmente son de código abierto pero existen algunas de tipo propietario.

3.1 – SQLITE

- Sistema gestor de bases de datos multiplataforma. Está escrito en C y tiene un motor ligero.
- Las bases de datos se guardan como ficheros (en forma de ficheros. Con esto se facilita el traslado de las bases de datos con la aplicación con la que se usa.
- Es un proyecto de dominio público
- Se puede usar desde programas en C/C++, PHP, Visual Basic, Java o Perl.

3.2 – APACHE DERBY

- Nos sirve para bases de datos relacionales de código abierto, implementado totalmente en Java.
- Está disponible bajo licencia Apache 2.0.
- Ventajas:
    - Tamaño reducido
    - Basada en Java y soporta los estándares SQL.
    - Soporta el paradigma cliente-servidor.

Fácil de usar, instalar e implementar.

3.3 – HSQLDB

- Sistema gestor de bases de datos relacionales escrito en Java.
- El paquete Open Office la incluye desde su versión 2.0.
- Puede mantener la base de datos en su memoria o en ficheros en disco.
- Se distribuye bajo licencia BSD (Berkeley Software Distribution).
- Es una licencia muy cercana a dominio público.

3.4 – H2

- Es un sistema gestor de bases de datos relaciones programadas en Java.
- Disponible como software de código abierto bajo la licencia pública de Mozilla.

3.5 –DB4O

- Motor de bases de datos orientado a objetos.
- Está disponible para entornos Java y .NET.
- Posee una licencia dual GPL/comercial.
- Características
    - Se consigue evitar el desfase objeto-relacional.

- No existe un lenguaje SQL para manipular los datos. Para ello se usan métodos delegados.
- Para instalarlo solo hay que añadir una librería.
- Se guarda como un único fichero de base de datos con la extensión ".yap".

3.6 – OTRAS

Veamos otras bases de datos embebidas, algunas de software libre y otras de propietario.

FIREBIRD

Se deriva del código de InterBase 6o de Borland. Es un sistema gestor de bases de datos relacional y de código abierto, es totalmente abierto, se puede usar tanto en aplicaciones comerciales como de código abierto.

- Se presenta en 3 versiones de servidor: SuperServer, Classic y Embedded.
- Es fácil distribuir las aplicaciones ya que no requiere instalación. Se suele usar para crear catálogos en CD-ROM.

Características:

- Completo soporte para procedimientos almacenados y disparadores
- Integridad referencial
- Bajo consumo de recursos
- Lenguaje interno para los procedimientos almacenados y disparadores
- Soporte para funciones externas
- Excasa necesidad de administradores especializados
- Múltiples formas de acceder a la base de datos

SQL SERVER MOBILE

Es una base de datos compacta. Tiene gran variedad de funciones y está diseñada para un gran número de tabletas y de dispositivos inteligentes.

- Ocupa poco espacio
- Es un producto de Microsoft que incluye varias características de las bases de datos relacionales:
  - Tiene un motor de bases de datos compacto y un sólido optimizador de consultas
  - Está permitido el acceso a datos remoto
  - Está integrado con el Microsoft SQL Server 2005
  - Las herramientas de aplicación son: Microsoft SQL Server Management Studio, SQL Server Management Studio Express
  - Está integrado con Microsoft Visual Studio 2005
  - Es compatible con Microsoft ADO.Net
  - Es un conjunto de sintaxis SQL

- Se implementa como una base de datos incrustada en equipos de escritorio, dispositivos móviles y tabletas
- Es compatible con la tecnología de ClickOnce

ORACLE EMBEDDED

Ofrece un conjunto de soluciones al software de desarrollo que aportan ciertas ventajas:

- Tiene una instalación silenciosa (se instala solo, no hace falta hacer nada)
- La configuración es automática
- Pocos costes de implementación, de hardware, licencia, administración…
- Ofrece elementos especializados y una gran libertad de elección a la hora de cubrir las necesidades

La base de datos permite elegir entre 3 elementos:

- Oracle Database 11g para aplicaciones empresariales
- Oracle TimesTen In-Memory Database para aplicaciones que necesitamos tiempos de respuesta muy bajos. Es una base de datos relacional que se ejecuta en memoria, lo cual nos garantiza unos tiempos de respuesta muy bajos. La cache de datos es en tiempo real. Se desarrolla en entornos de misión crítica.
- Oracle Berkeley DB nos proporciona una base de datos embebida que permite varias opciones de almacenamiento
  - Se puede ejecutar en memoria o en disco y nos proporciona un alto rendimiento
  - Tiene un rendimiento extraordinario lo cual elimina los gastos de comunicación entre procesos y SQL
  - La administración es cero, de toda ella se encarga una API, la cual se integra por completo en la aplicación y es invisible para los usuario
  - Para dispositivos móviles ocupa muy poco (<1MB) y el tiempo de ejecución en memoria dinámica es muy corto
  - Bajo coste total porque al ser embebida no gasta en licencias, administrador, hardware, servidor…

4 – PROTOCOLOS DE ACCESO A BASES DE DATOS

NORMAS DE CONEXIÓN A UNA BASE DE DATOS SQL

1) ODBC (Open DataBase Connectivity). Define una API que pueden usar las aplicaciones para abrir una conexión a una base de datos, enviar consultas, actualizaciones y obtener resultados. Las aplicaciones podrán usar esta API para conectarse a cualquier servidor de bases de datos compatible con ODBC.
2) JDBC (Java DataBase Connectivity). Define una API que pueden usar los programas Java para conectarse a los servidores de bases de datos relacionales.

Hay muchos orígenes de datos, no sólo las bases de datos relacionales. Por ejemplo:

Los ficheros planos, los almacenes de correo electrónico, etc.

OLE-DB (OBJECT LINKING AND EMBEDDING FOR DATABASE)

Es una API de C++ de Microsoft. Los objetos son parecidos a ODBC, pero los orígenes de datos no son bases de datos características.

Características:

1) Proporciona estructuras para la conexión con orígenes de datos, ejecución de comandos y devolución de resultados en forma de conjunto de filas.
2) Los programas OLE-DB pueden negociar con los orígenes de datos para averiguar las interfaces que soportan.
3) En ODBC los comandos siempre están en SQL, pero en OLE-DB pueden estar en cualquier lenguaje soportado por el origen de datos.

ADO (ACTIVE DATA OBJECTS)

Es una API creada por Microsoft. Ofrece una interfaz muy sencilla de utilizar con la funcionalidad de OLE-DB.
Puede llamarse desde los lenguajes de guiones como: el VBScript, JScript.

5 – ACCESO A DATOS MEDIANTE ODBC

(Ejercicio de búsqueda)

- *Introducción, para que sirve, licencias*
- *¿Qué pasa con las bibliotecas?*
- *Pasos para usar un ODBC*
- *Lenguajes de programación*
- *Funciones importantes*
- *Manejadores*

5 – ACCESO A DATOS MEDIANTE JDBC

JDBC proporciona una librería estándar para poder acceder a fuentes de datos, principalmente orientada a bases de datos relacionales que usan SQL. No solo proporciona una interfaz sino que también nos define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones Java el acceso a los datos.

JDBC dispone de una interfaz distinta para cada base de datos, es lo que se conoce como conector, driver, controlador.

JDBC consta de un conjunto de clases e interfaces que nos permiten escribir aplicaciones Java.

Esto implica que podamos gestionar ciertas tareas con una base de datos relacional (3).

- Conectarse a la base de datos
- Enviar consultas e instrucciones de actualización a la base de datos
- Recuperar y procesar los resultados recibidos de la base de datos respondiendo a las consultas

ARQUITECTURAS JDBC

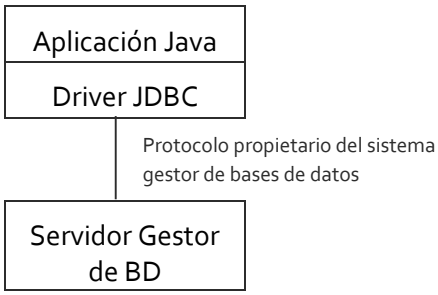
La API JDBC es compatible con los modelos de 2 y de 3 capas para el acceso a la base de datos.

Para el **modelo de 2 capas** un applet o una aplicación Java "habla" se comunica directamente con la base de datos; esto requiere un driver JDBC residiendo en el mismo lugar que la aplicación.

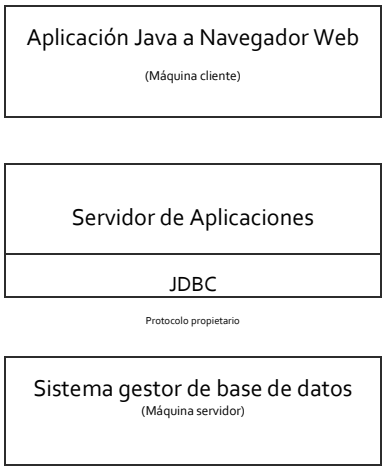
Desde el mismo programa de Java se envían sentencias SQl al sistema gestor de bases de datos para que las procese y los resultados se envían de vuelta al programa.

La base de datos puede encontrarse en otra máquina distinta a la de la aplicación y las solicitudes se hacen a través de la red (arquitectura cliente-servidor).

El driver se encarga de tramitar toda esta información mediante la red siendo transparente para el programa.



En el **modelo de 3 capas** los comandos se mandan a una capa intermedia que se encarga de enviar los comandos SQL a la base de datos y recoger los resultados de la ejecución de las sentencias. Tenemos una aplicación corriendo en una máquina y accediendo a un driver de base de datos situado en otra máquina. En este caso los drivers no tienen que residir en la máquina cliente.



TIPOS DE DRIVERS

Existen cuatro tipos de conectores JDBC:

- **Tipo 1: JDBC-ODBC Bridge**: Permite el acceso a bases de datos JDBC mediante un driver ODBC, exige instalar y configurar ODBC en la máquina cliente.
- **Tipo2: Native (Native-Api-Party-Java Driver)**: Escrito parcialmente en Java y en código nativo. Traduce las llamadas al Api de JDBC Java en llamadas propias del motor de bases de datos. Exige instalar en la máquina cliente código binario propio del cliente y del sistema operativo.
- **Tipo3: Network (JDBC-Net Pure Java Driver)**: Controlador de Java puro, que utiliza un protocolo de red, por ejemplo HTTP, para comunicarse con un servidor de bases de datos. Traduce las llamadas al API de JDBC Java en llamadas propias del protocolo de red. No exige instalación en la máquina cliente.
- **Tipo4: THIN (Native-Protocolo Pure Java Driver)**: Controlador de Java puro con protocolo nativo. Traduce las llamadas a la API de JDBC Java en llamadas propias del protocolo de red usado por el motod de la base de datos. No exige instalación en la máquina cliente.

Los tipos tres y cuatro son los mejores para acceder a bases de datos JDBC.

Los tipos uno y dos se usan cuando no queda más remedio, si el único sistema de acceso final al gestor de bases de datos es ODBC, pero exigen instalación de software en la parte cliente.

Si no existen drivers disponibles para el sistema gestor de bases de datos normalmente se usa el tipo cuatro.

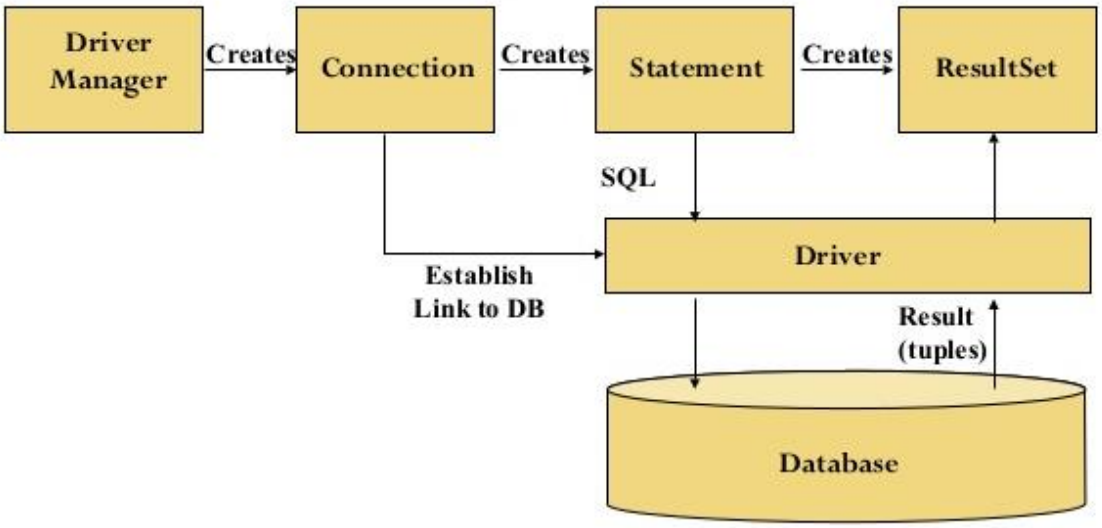
¿CÓMO FUNCIONA JDBC?

JDBC define varias interfaces que permiten realizar operaciones con bases de datos. A partir de estas operaciones se derivan las clases correspondientes. Estas clases están definidas en el paquete java.sql.

Las clases e interfaces más importantes son:

- **Driver**: Permite conectarse a una base de datos. Cada gestor de base de datos requiere un driver distinto
- **DriverManager**: Permite gestionar todos los drivers instalados en el sistema.
- **DriverPropertyInfo**: Proporciona diversa información sobre el driver.
- **Connections**: Representa una conexión con una base de datos. En una aplicación va a poder haber más de una.
- **DatabaseMetadata**: Nos proporciona información sobre la base de datos. Por ejemplo, el número de tablas.
- **Statement**: Permite ejecutar sentencias SQL, pero sin parámetros.
- **PreparedStatement**: Permite ejecutar sentencias SQL pero con parámetros de entrada.
- **CallableStatement**: Permite ejecutar sentencias SQL, pero con parámetros de entrada y de salida, como llamadas a procedimientos almacenados.
- **ResultSet**: Contiene las filas resultantes de ejecutar una orden SELECT.
- **ResultSetMetadata**: Permite obtener información sobre Resultset. Por ejemplo, el número de columnas, sus nombres.

Las cuatro clases principales que usa cualquier programa Java con JDBC son:



El trabajo con JDBC comienza con la clase DriverManager que se encarga de establecer las conexiones con los orígenes de datos a través de los drivers JDBC.

Veamos los pasos del funcionamiento de un programa JDBC:

1) Importar las clases necesarias
2) Cargar el driver JDBC
3) Identificar el origen de datos
4) Crear un objeto Connection
5) Crear un objeto Statement
6) Ejecutar una consulta con el objeto Statement
7) Recuperar los datos del objeto ResultSet
8) Liberar el objeto ResultSet
9) Liberar el objeto Statement
10) Liberar el objeto Connection

En un programa Java de JDBC accediendo a una base de datos todos los import necesarios para manejar la base de datos están incluidos en java.sql.*

Hay que incluir un try/catch ya que casi todos los métodos relativos a bases de datos pueden lanzar la excepción SQLException. Aunque puede lanzar otras excepciones.

Pasos a realizar por el programa:

1) **Cargar el driver**: Con el método forName, de la clase Class, se le pasa un objeto String con el nombre de la clase del driver como argumento. Ejemplo: `Class.forName("com.mysql.jdbc.Driver")` -> Nombre de la clase del driver.
2) **Establecer la conexión**: El servidor tiene que estar arrancado, usamos la clase DriverManager con el método getConnection(). Ejemplo: `Connection conexion = DriverManager.getConnection("jdbc.mysql://localhost/ejemplo", "root", "root");`
    - El 1er parámetro del método representa la URL de conexión a la base de datos.
        (1) jdbc:mysql indica que estamos usando un driver JDBC para MySQL
        (2) localhost indica que el servidor de bases de datos está en la misma máquina en la que se ejecuta el programa. Podremos poner una IP o un nombre que esté en la red
        (3) ejemplo es el nombre de la base de datos a la que vamos a acceder
    - El 2do parámetro es el nombre del usuario que accede a la base de datos
    - El 3er parámetro es la clave del usuario que accede
3) **Ejecutar sentencias SQL**: Se realiza la consulta, para ello recurrimos a la clase Statement para crear sentencias. Para obtener un objeto Statement usamos el método createStatement(). La sentencia que hemos obtenido tiene el método executeQuery() que sirve para realizar una consulta a la base de datos. Ejemplo: `Statement sentencia = conexión.createStatement(); ResultSet resul = sentencia.executeQuery("SELECT * FROM tabla");` El resultado nos lo devuelve como un ResultSet, es un objeto similar a una lista en la que está el resultado de la consulta. Cada elemento de la lista es uno de los registros de la tabla. De primeras ResultSet no contiene todos los datos, los va obteniendo de la base de datos según se van pidiendo. Por ello el método executeQuery() puede ser un poco lento. ResultSet tiene internamente un puntero que apunta al primer registro de la lista. Mediante el método next() el puntero avanza al siguiente

registro. Para recorrer la lista de registros usaremos el método next() dentro de un bucle while que se ejecutará mientras next() devuelva true. Ejemplo: `while(result.next()) => System.out.println(resul.getInt(1) + " " + resul.getString(2) + " " + resul.getString(3));` Los métodos getInt() y getString() nos están devolviendo los valores de los campos de dicho registro. Entre paréntesis ponemos la posición de la columna en la tabla. También se puede poner una cadena que indica el nombre de la columna: `resul.getInt("dept_no")`.

4) **Liberar recursos**: Se liberan todos los recursos y se cierra la conexión. Ejemplo: `resul.close(); sentencia.close(); conexion.close();`

6 - ACCESO A DATOS MEDIANTE EL PUENTE JDBC-ODBC

Hay productos, aunque muy pocos, para los que no hay que conector JDBC pero si hay un conector ODBC. En estos casos se suele utilizar el puente JDBC-ODBC Bridge.

El puente JDBC-ODBC es un controlador JDBC, que implementa operaciones JDBC traduciéndolas en operaciones ODBC, para ODBC aparece como una aplicación normal.

El puente está implementado en Java y usa métodos nativos de Java para llamar a ODBC y se instala automáticamente con el JDK.

Para cada esquema de base de datos es necesario crear un origen de datos.

Ejemplo: Para acceder a una base de datos MySQL usando el puente JDBC-ODBC necesitamos crear un origen de datos o DSN en el panel de control.

7 - PATRÓN MODELO-VISTA-CONTROLADOR (MVC) EN ACCESO A DATOS

El patrón MVC es un patrón de diseño que se utiliza como guía para el diseño de arquitecturas software que ofrecen una gran interactividad con el usuario y donde se requiere una separación de conceptos para que el desarrollo se realice más eficientemente, facilitando la programación en distintas capas de manera paralela e independiente.

Este patrón organiza la aplicación en 3 bloques, cada cual especializado en una tarea:

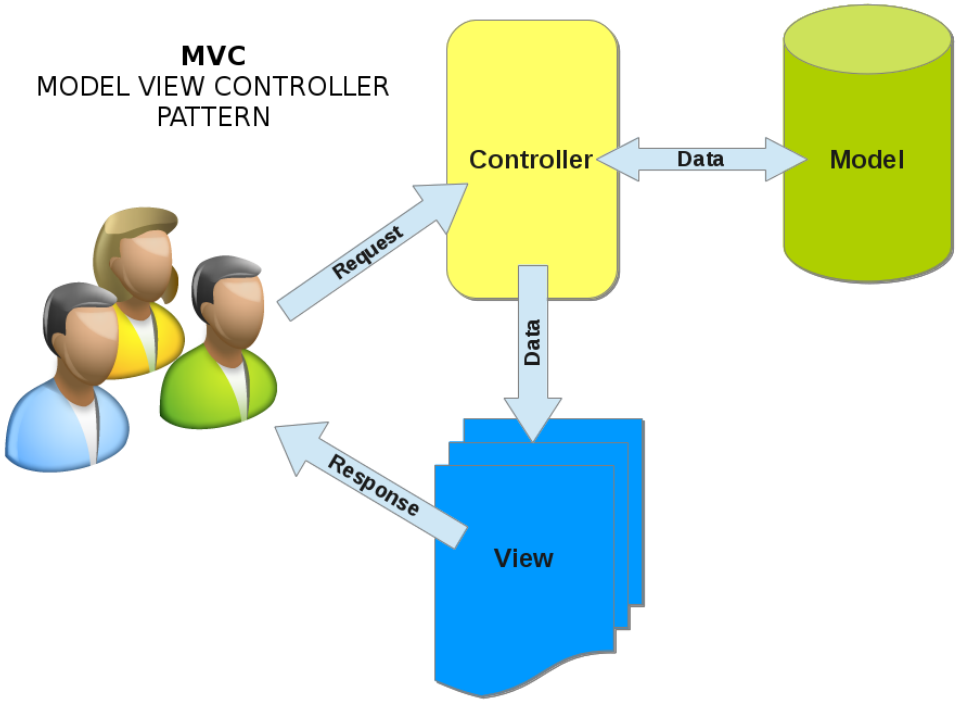
- **Modelo**: Representa los datos de la aplicación y sus reglas de negocio
- **Vista**: Representa el modelo de forma gráfica para poder interactuar con el usuario
- **Controlador**: Interpreta los datos que recibe del usuario, analiza las peticiones y coordina el modelo y la vista para que la aplicación produzca los resultados deseados

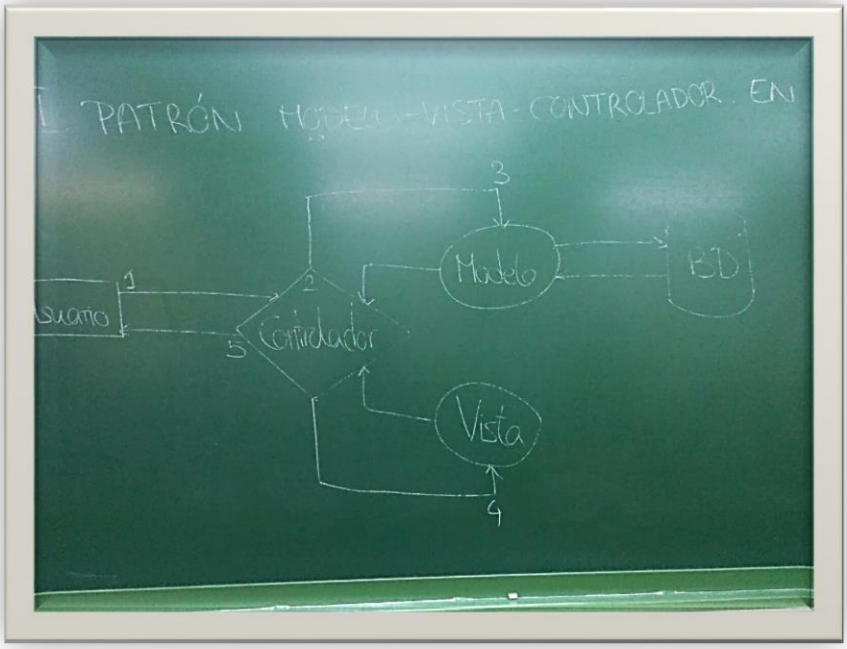
VENTAJAS DE MVC

- Diseño de aplicaciones modulares
- Reutilización de código
- Facilita probar las unidades por separado
- Facilita la detección y manipulación de errores
- Separación de los datos de la representación visual de los mismos

DESVENTAJAS DE MVC

- Al separar nuestro sistema en capas agregamos complejidad
- Hay mayor cantidad de ficheros para desarrollar





1) Un usuario realiza una solicitud a través de la aplicación y esta solicitud es dirigida al controlador.
2) El controlador examina la solicitud y decide que regla de negocio aplicar, es decir, determina el componente de negocio a aplicar para poder procesar la solicitud, este componente de negocio es el modelo.
3) El modelo contiene las reglas de negocio que procesan la solicitud y que dan lugar al acceso a los datos que necesita el usuario. Estos datos se devuelven al controlador.
4) El controlador toma los datos que devuelve el modelo y selecciona la vista en que se van a presentar estos datos al usuario.
5) El controlador devuelve los resultados al usuario una vez procesada su solicitud.