Bases de Datos

Patrón de diseño MVC
Ejemplo de su aplicación en
JAVA y Bases de Datos MySQL

Dr. Diego R. Garcia

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur

Patron de Diseño MVC

- Patrón de diseño de Software que propone separar el código de un programa en 3 responsabilidades (o capas):
  - El Modelo
  - La Vista
  - El Controlador
- Objetivo: Separar la interfaz gráfica del usuario de la lógica del sistema.

El Modelo

- Encargado de la representación de la información con la cual opera el sistema.
- Gestiona todos los accesos a la información, tanto consultas como actualizaciones.
- Implementa la lógica del sistema, también conocida como lógica de negocio.

La Vista

- Encargada de la representación del Modelo en un formato adecuado para interactuar con el usuario (interfaz de usuario)

El Controlador

- Intermediario entre la Vista y el Modelo.
- Encargado de responder a eventos en la Vista (acciones del usuario) e invocar peticiones al Modelo para recuperar información. También puede enviar comandos a su “vista” asociada para que muestre información.

Interacción entre las capas

Usuarios

1. el usuario solicita una acción a la Vista

6. el usuario recibe la la respuesta en la Vista

Vista

2. la Vista captura el evento del usuario y solicita al Controlador que lo resuelva

5. El Controlador responde el pedido de la Vista con los datos devueltos por el Modelo

Controlador

4. La vista responde a la petición del controlador accediendo a los datos

3. el Controlador invoca al Modelo para responder a la petición de información de la vista

Modelo

Modelo: DAO y Beans

- Cuando el patrón MVC se implementa en un lenguaje Orientado a Objetos, en la capa encargada de representar el Modelo se suelen utilizar otras sub-capas o clases auxiliares
- Patrón DAO (Data Access Object): clases encargadas de manejar el acceso a los datos.
- Beans: Es un tipo de objetos que se utiliza para representar datos en el modelo de objetos y poder intercambiar información entre el Modelo, el Controlador y la Vista. En algunos modelos se suelen usar objetos similares a los Beans conocidos como VO (Value Objects) o POJO (Plain Old Data Objects).

Patrón DAO

- El problema que resuelve este patrón es netamente el acceso a los datos. Básicamente tiene que ver con la gestión de diversas fuentes de datos (Bases de Datos, archivos, repositorios en la nube, etc) y además abstraer la forma de acceder a ellos.
- Ventaja: tener una aplicación que no esté ligada al acceso a datos. Por ejemplo, si tenemos un sistema montado con una base de datos MySQL y de pronto lo debemos cambiar a PostgreSQL (o a cualquier otro motor de B.D.) simplemente tenemos que reemplazar la capa DAO, sin necesidad de modificar el Modelo, la Vista o el Controlador.

JavaBean

- Una clase de objetos JavaBean, debe cumplir ciertas convenciones sobre nomenclatura de métodos, construcción y comportamiento:
  - Debe tener un constructor sin argumentos.
  - Sus atributos de clase deben ser privados.
  - Sus propiedades deben ser accesibles mediante métodos get y set que siguen una convención de nomenclatura estándar.
  - Debe ser "serializable" (implementar java.io.Serializable) (Permite convertir un objeto en un montón de *bytes* y luego recuperarlo)

JavaBean: ejemplo

- Clase `BatallaBean` para representar una tupla de la tabla batallas en la B.D. como un objeto dentro del modelo O-O:

```
public class BatallaBean implements Serializable{
 private static final long serialVersionUID = 1L;
//Atributos privados
 private String nombre;
 private java.util.Date fecha;
//setters y getters de los atributos
 public String getNombre() {return nombre;}
 public void setNombre(String nombre) {this.nombre = nombre;}
 public java.util.Date getFecha() {return fecha;}
 Public void setFecha(java.util.Date fecha){
    this.fecha = fecha;}
}
```

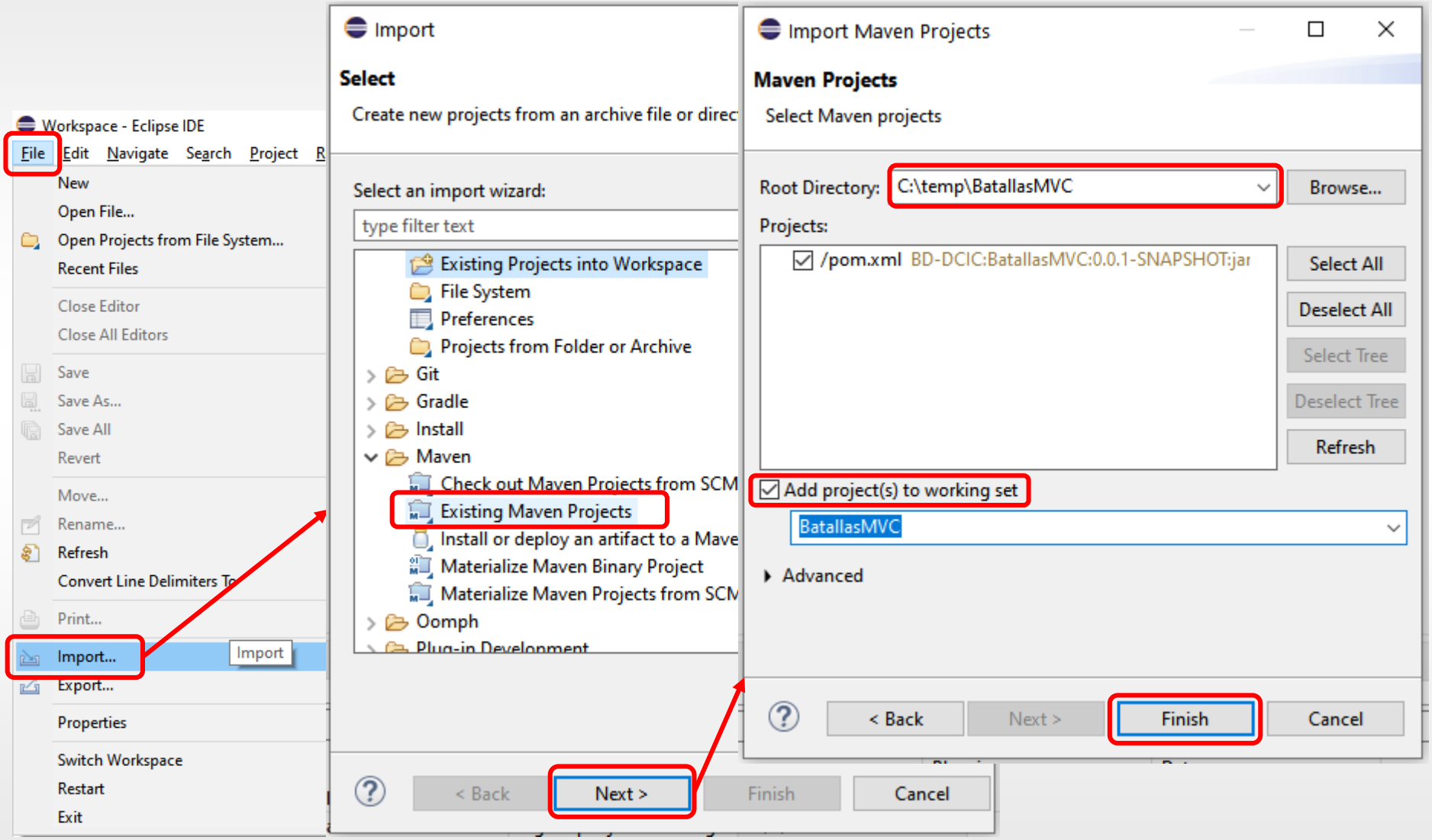
Ejemplo Patrón MVC en Java

- Veremos una aplicación Java implementada siguiendo el patrón MVC, que provee funcionalidades para Crear, Buscar, Actualizar y Eliminar batallas de la tabla batallas de la B.D. homónima (utilizada en el práctico de SQL). Estas operaciones se conocen habitualmente bajo la sigla CRUD (Create, Read, Update, Delete).

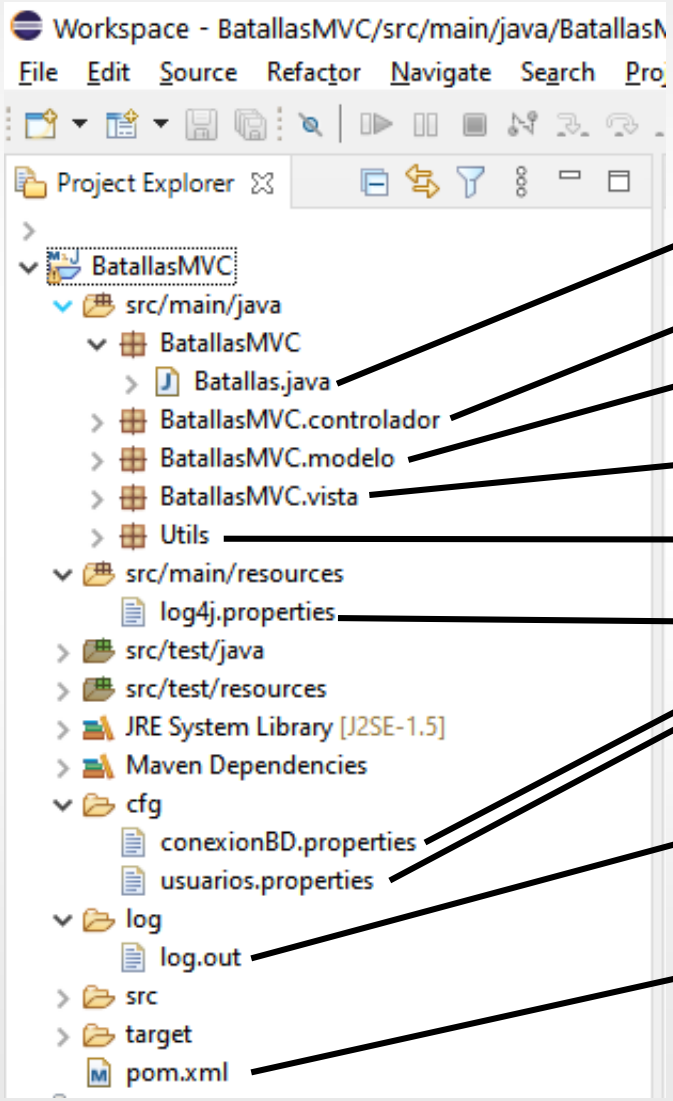
Ejemplo Patrón MVC en Java

- La aplicación fue implementada en el IDE Eclipse como un proyecto Maven.
- Maven es una herramienta de software para la gestión y construcción de proyectos Java basado en el concepto project object model (POM). Entre otras cosas, permite manejar de manera mas simple y automatizada las dependencias a librerías externas (Por ejemplo, el driver JDBC para acceder a la B.D.)

Importar proyecto Maven



Proyecto BatallasMVC



Clase principal

Paquete que implementa el Controlador

Paquete que implementa el Modelo

Paquete que implementa la Vista

Paquete de clases auxiliares

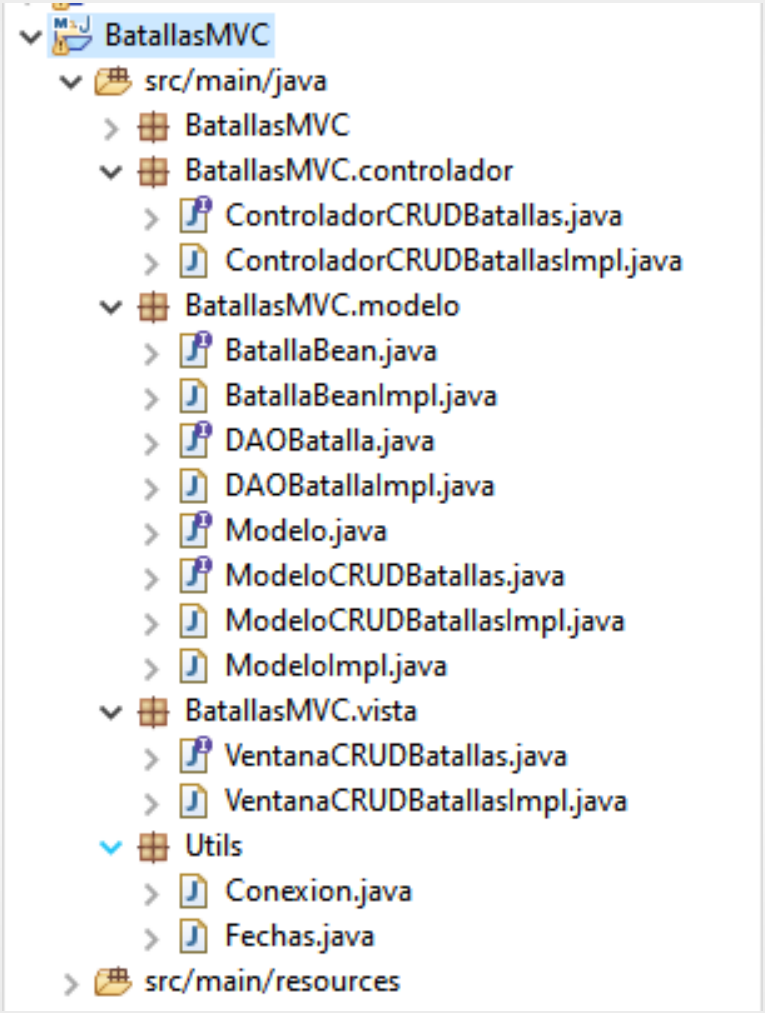
Archivo con la configuración del logger

Archivos de propiedades con la configuración del servidor de B.D. y usuarios

Archivo de log para debugging: documenta toda la actividad y errores de la aplicación

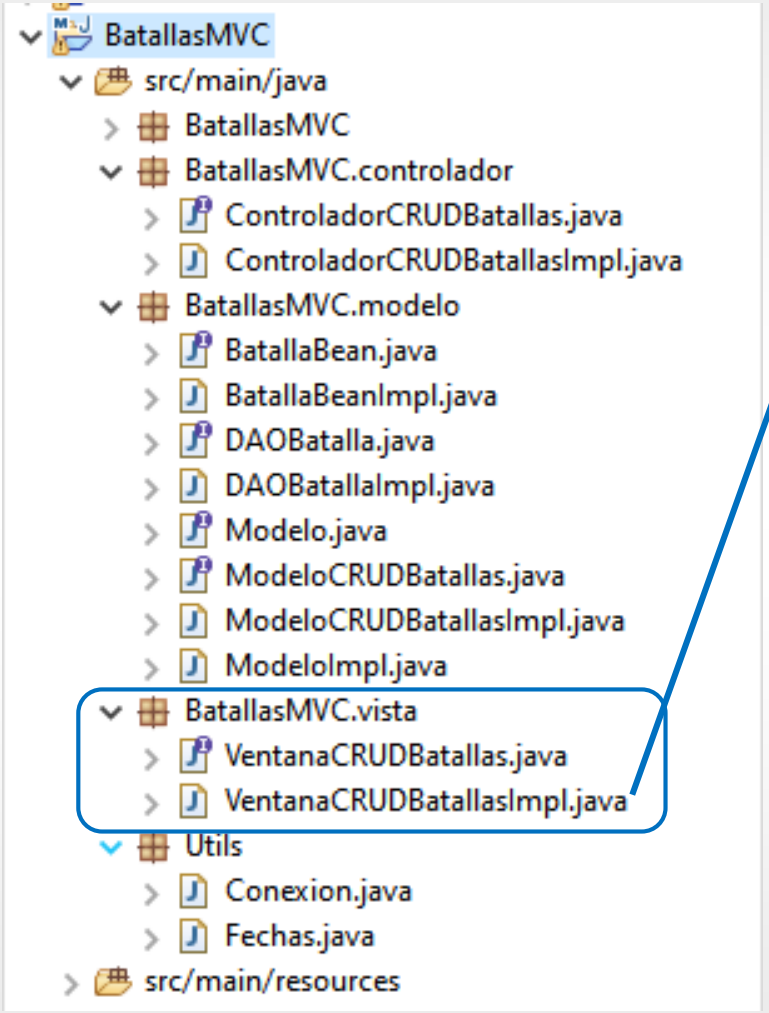
Archivo de configuración de proyecto Maven: setear main class, versión de Java, dependencias externas, etc…

Proyecto BatallasMVC: Paquetes



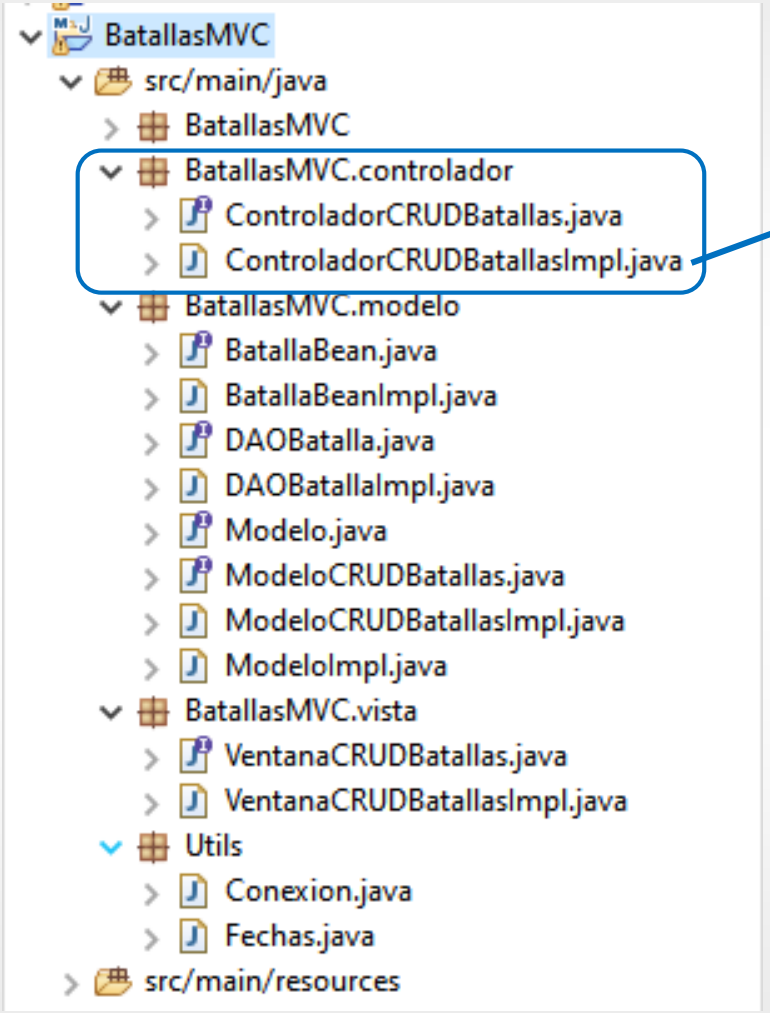
- En todos lo paquetes cada clase (salvo excepciones, por ej. Utils.Conexion y Utils.Fechas) tiene su interfase correspondiente con la signatura de sus métodos públicos.
- Las clases que implementen una interfase tendrán el mismo nombre que la interfase con el agregado de la sigla “Impl” al final. Por ejemplo:
  - BatallaBean.java (interfase)
  - BatallaBeanImpl.java (clase que implementa la interfase BatallaBean)

Proyecto BatallasMVC



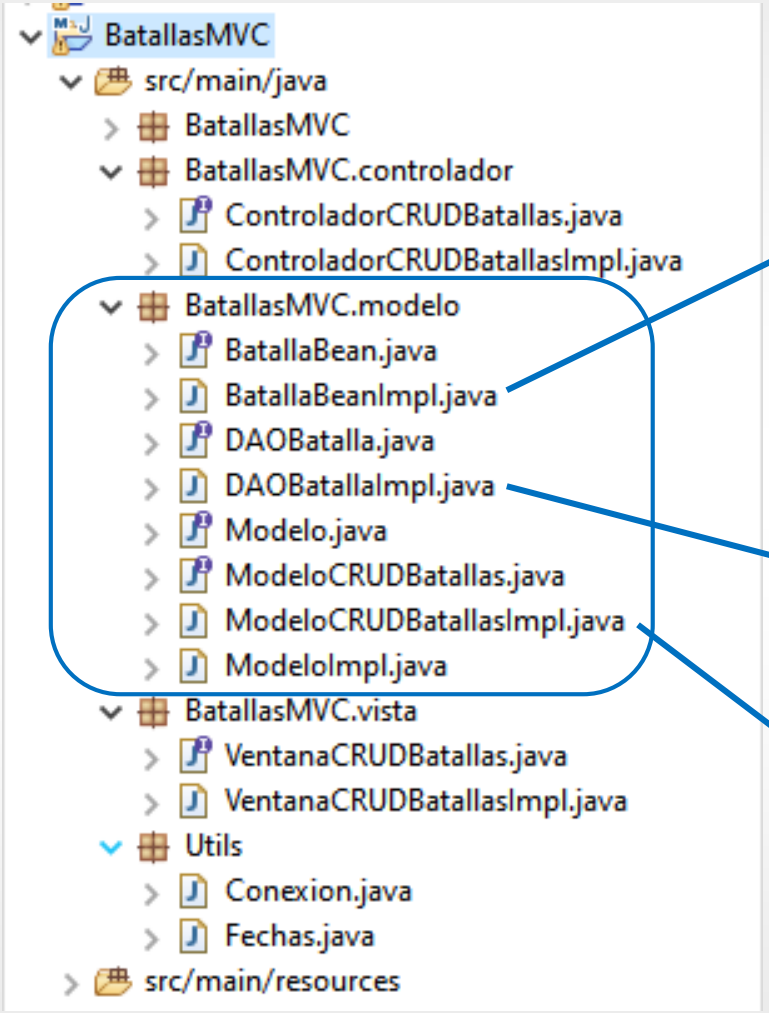
- Paquete BatallaMVC.Vista: implementa la Vista a través de la clase VentanaCRUDBatallasImpl
  - Define componentes: botones, campos de texto y tabla (Jtable) para interactuar con el usuario.
  - Métodos para limpiar campos, mostrar una lista de Batallas en la tabla y seleccionar una batalla.

Proyecto BatallasMVC



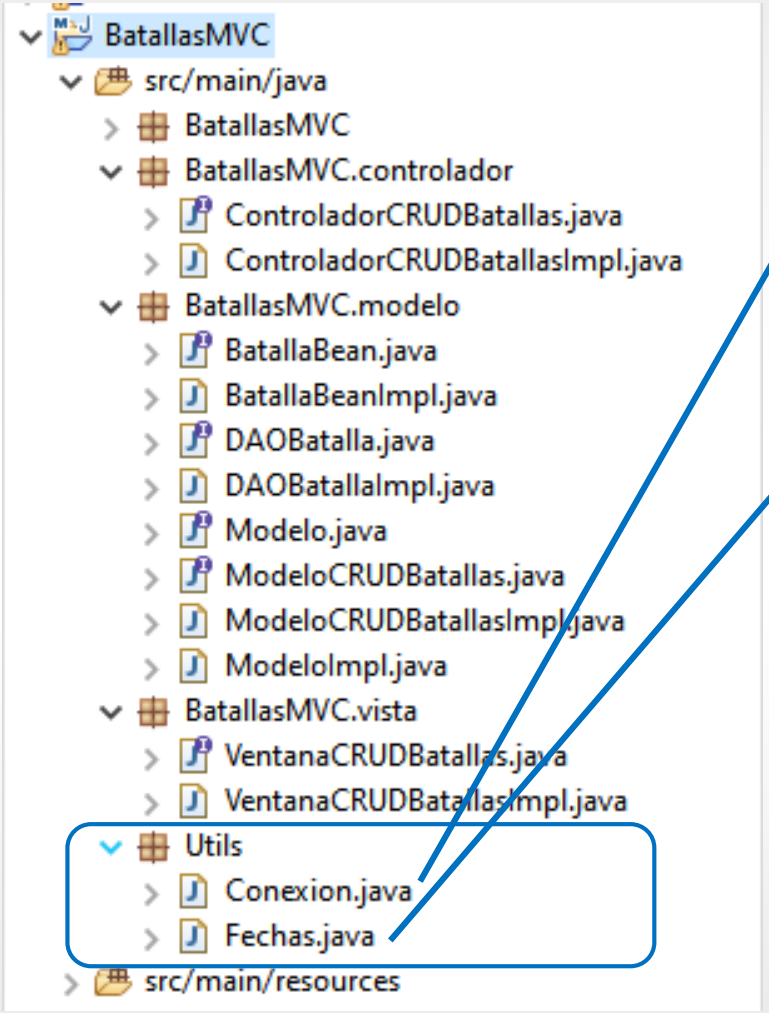
- Paquete BatallaMVC.Controlador: implementa el Controlador a través de la clase controladorCrudBatallasImpl.java
  - Métodos para atender los eventos producidos en la vista
  - Básicamente hay un método por cada botón de la vista que se encargan de crear, buscar, actualizar y eliminar una batalla invocando métodos del modelo que conocen como acceder y manipular los datos.

Proyecto BatallasMVC



- Paquete BatallasMVC.Modelo: implementa el modelo utilizando diferentes clases:
  - BatallaBeanImpl.java: representa una tupla de la tabla batallas como un objeto, para poder intercambiar información entre el Modelo, la Vista y el Controlador.
  - DAOBatallasImpl.java: subcapa del modelo que resuelve solo el acceso al servidor MySQL.
  - ModeloCrudBatallasImpl.java: clase principal que maneja el acceso a los datos y la lógica de la aplicación, extendiendo una clase de modelo genérica (Modelo.java).

Proyecto BatallasMVC



- Paquete Utils: Clases auxiliares
  - Conexión.java: provee métodos estáticos para gestionar una conexión **(java.sql.Connection)** con el servidor MySQL.
  - Fechas.Java: provee métodos estáticos para convertir diferentes formatos de fechas: String, Java.util.Date, Java.sql.Date

Diagrama de eventos: botón Crear