**JSE**

**Objeto ->** propiedades (atributos) – Comportamiento->métodos

**Constructor ->** método da un estado inicial a nuestro objeto

**Modularización ->** separar en diferentes archivos

**Encapsulación ->** private en los atributos

**Getters y Setters ->** acceder a las propiedades de los objetos

**Herencia (reutilización)->** hereda las propiedades y métodos de la clase padre

Clase Coche es superclase (clase padre)

Clase Furgoneta es subclase (clase hijo)

public class Furgoneta extends Coche{}

**Polimorfismo ->**

public String obtenerDetalles(){

return "Nombre: " + nombre + ", sueldo: " + sueldo;

}

//Sobreescribimos el método padre heredado

public String obtenerDetalles(){

//Observamos que para no repetir Código, podemos utilizar

//el método del padre y solo agregar a este método de la clase hija

//esto es invocar un método sobreescrito

return super.obtenerDetalles() + ", departamento: " + departamento;

}

public static void main(String[] args) {

Empleado emp = new Empleado("Juan", 1000);

imprimirDetalles(emp);

Gerente ger = new Gerente("Karla", 2000, "Finanzas");

imprimirDetalles(ger);

}

//Observamos que el tipo que recibe el método es de tipo padre (Empleado)

//Sin embargo al momento de ejecutar el método, se ejecuta el del hijo (Gerente)

//eso es polimorfismo, múltiples formas pero en tiempo de ejecución

public static void imprimirDetalles(Empleado emp){

System.out.println(emp.obtenerDetalles());

}

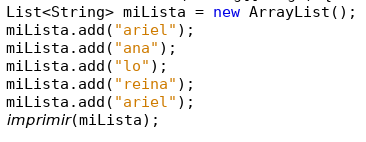
**Clase abstracta ->** Se declara método abstracto en superclase (clase padre) y todas las subclases están obligadas a sobreescribir ese método abstracto.

**Interfaz ->** Similar a clase abstracta, establece los métodos que utilizaran las subclases, sólo contienen métodos abstractos y constantes.

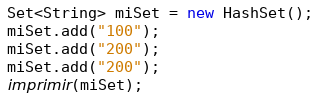
**Excepciones ->** throws/try/catch

**Colecciones ->** (genericas)

**Array**



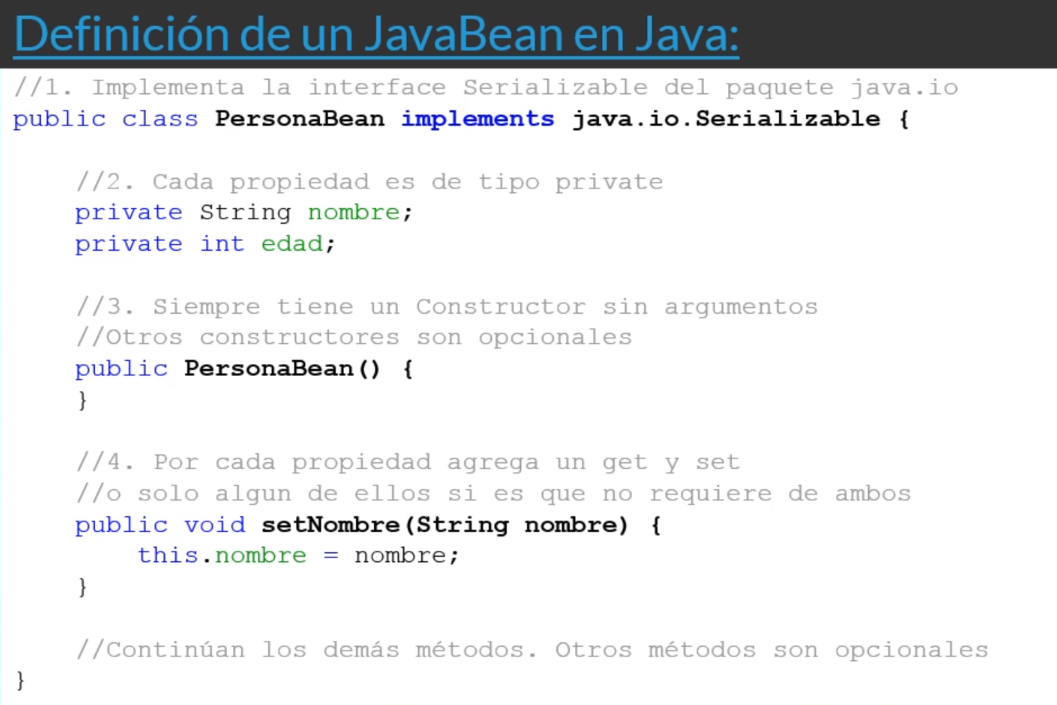
**HashSet**



**HashMap**



**Javabeans->** clase pura de Java para conexión a BBDD



**Colecciones en Java**

Similar al manejo de arreglos, conocido como estructura de datos, permite almacenar información de manera más variada que en los arreglos

**JDBC ->** Java Database Connectivity, API estándar de Java, se utiliza para conectarnos a una BBDD, define un conjunto de interfaces y clases java y deben ser implementadas por cada administrador de BBDD para las consultas

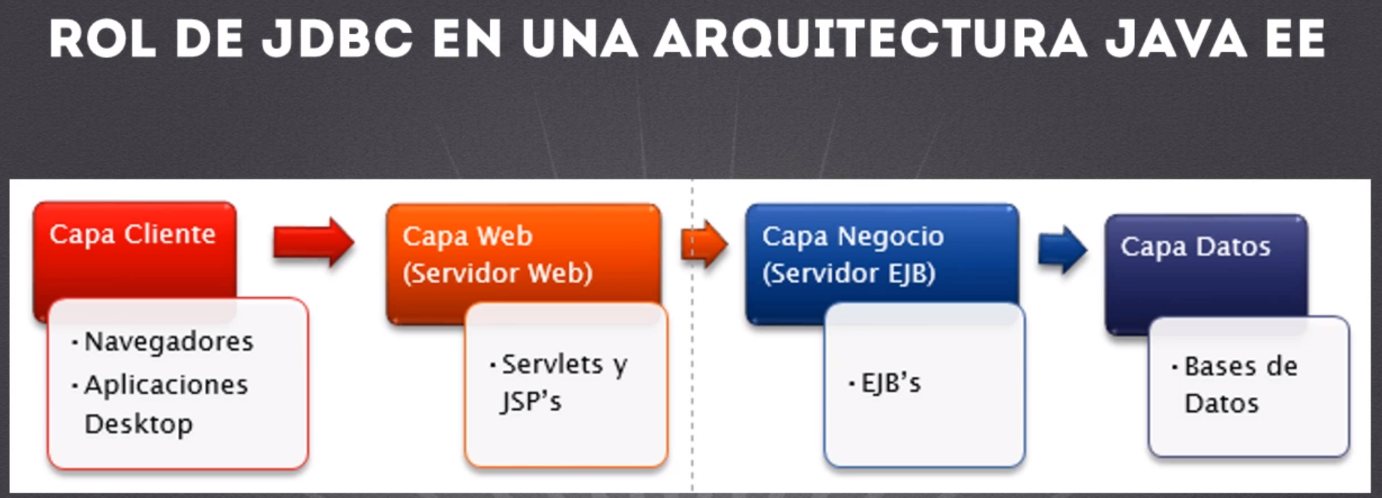
API JDBC

API-> conjunto de clases empaquetadas en un JAR. Tarea específica como conectarse a un BBDD en particular

Nos permite las operaciones del CRUD

Patrón de diseño DAO y DTO

Pool de conexiones para conectividad con la BBDD

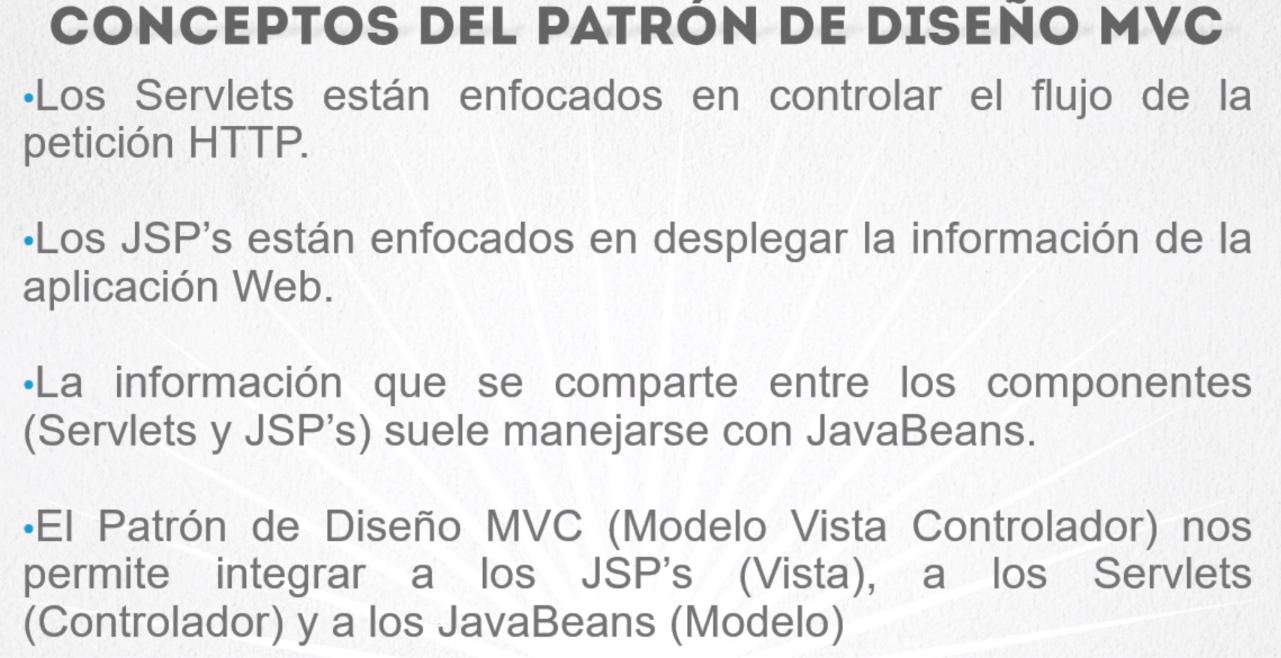


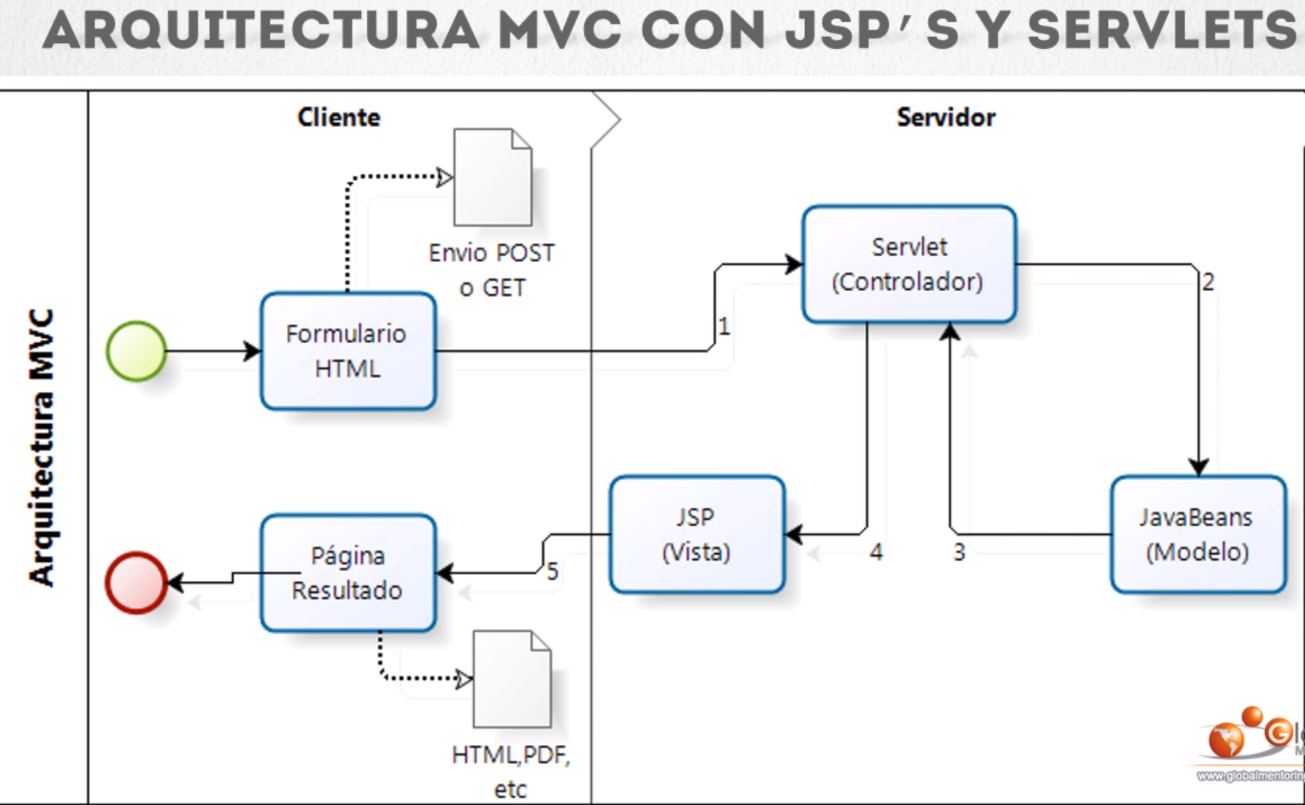
**Servlets y JSPs ->** Patrón de diseño MVC usando Servlets y JSP´s

JSP->Java Server Pages

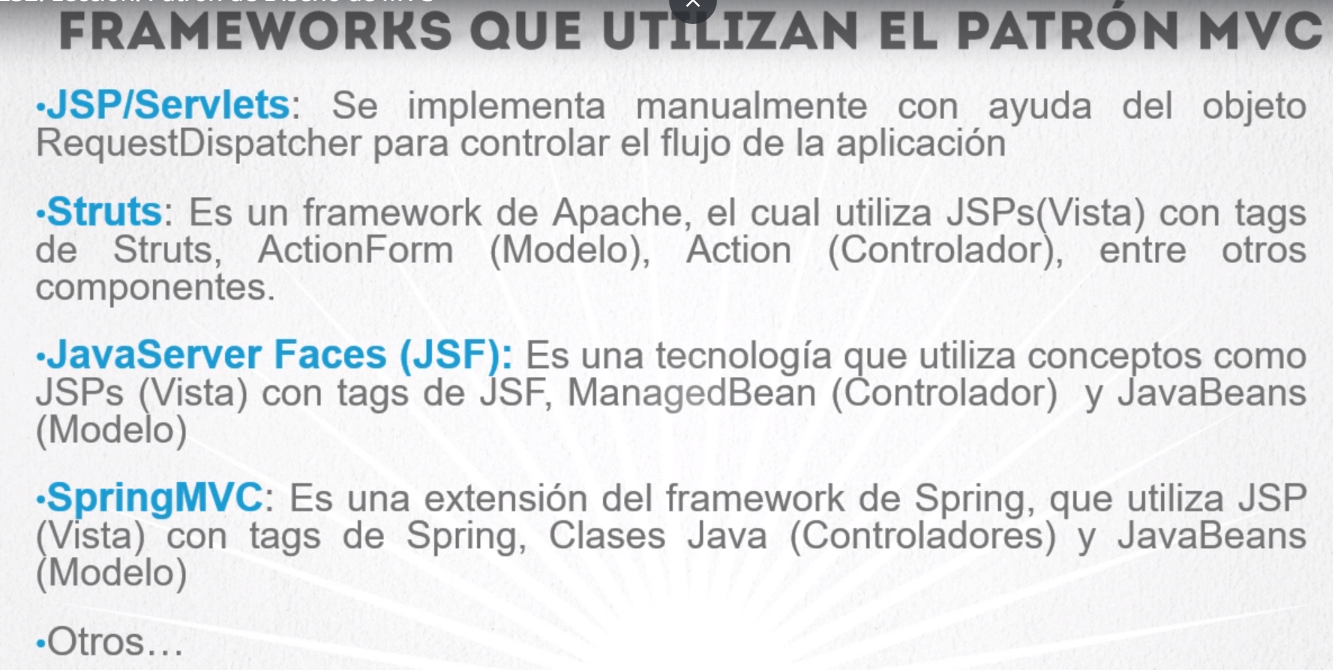
HTML, CSS y Javascript

Con esto ya creamos una aplicación Web









**JavaSever Faces ->** MVC

Managed Beans -> \*\*inyectar dependencias de las managed beans utilizando el estándar de JSF o el estándar de JEE

Facesconfig.xml -> archivo de configuración de JSF

Flujo entre páginas -> Tipos de navegación -> estática o dinámica

**JEE**

**EJB (Enterprise Java Beans) ->** Clases (controlador) que se comunican con las BBDD, son transaccionales, son como servicios dentro del sistema, reutilizables, son remotos.

**JPA (Java Persistence API) ->** Gestión de la BBDD (ABM)

\*\*Integración con Servlets y JSPs

\* Se aplica EJB dentro de los servlets

\*\*Integración con JSF

Capa de Presentación (Vista) -> JSF con Primefaces (extensión de JSF más utilizada)

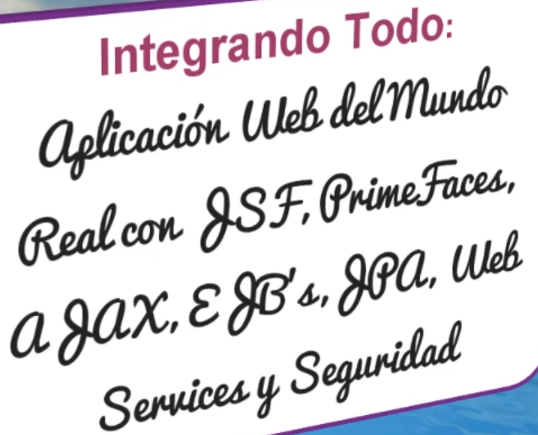
Capa de negocio (Controlador) -> EJB

Capa de datos (Modelo) -> JPA

Intercomunicación –> Web Services – SOAP y REST (estos son APIS)

Seguridad en Java EE -> agregar seguridad en interfaz web, en los EJBs y web services, además de validar cada uno de los clientes

Integrando Todo->



**Spring Framework**

Simplifica el desarrollo JEE

**POJO´s (clases Java Simples) ->** Configuración de servicios y la programación orientada a Interfaces

**Inyección de dependencias ->** Patrón que permite suministrar objetos a una clase que tiene dependencias, en lugar de que la clase las obtenga

**Programación Orientada a Aspectos ->** permite modularizar las aplicaciones y mejorar la separación de responsabilidades entre módulos.

**Módulos de Spring ->**

**Spring Core ->** Fábrica de beans, aplica la inyección de dependencias

**Spring AOP ->** Soporte para la programación orientada a objetos

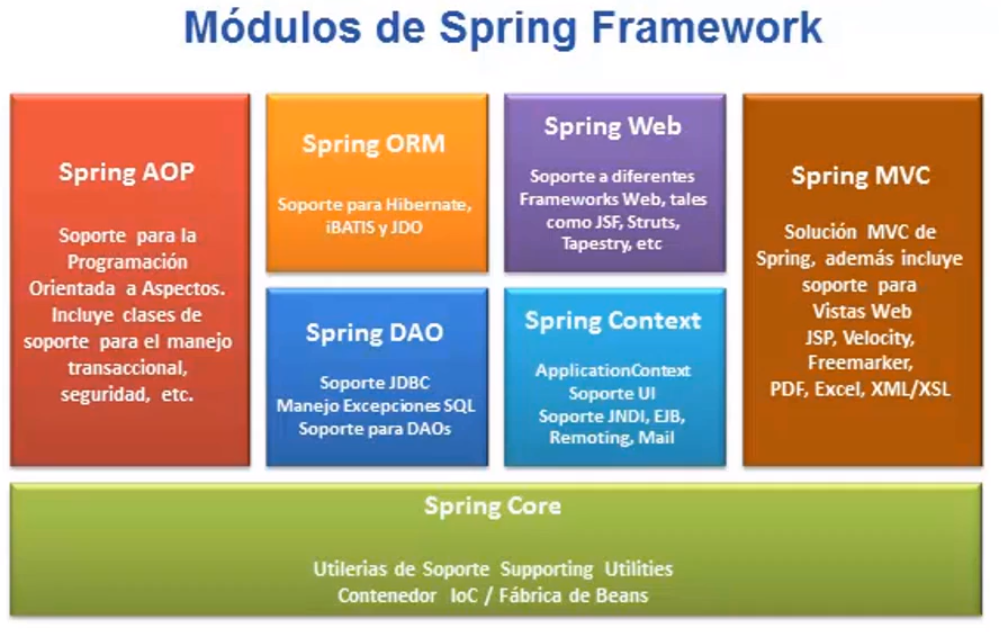
**Spring ORM ->** Permite integrar tecnologías como JPA, Hibernate, iBATIS entre otros frameworks de persistencia

**Spring DAO (Data Access Object) ->** Permite aplicar conceptos de la capa de datos a través de POJO´s, permite crear códigos JDBC

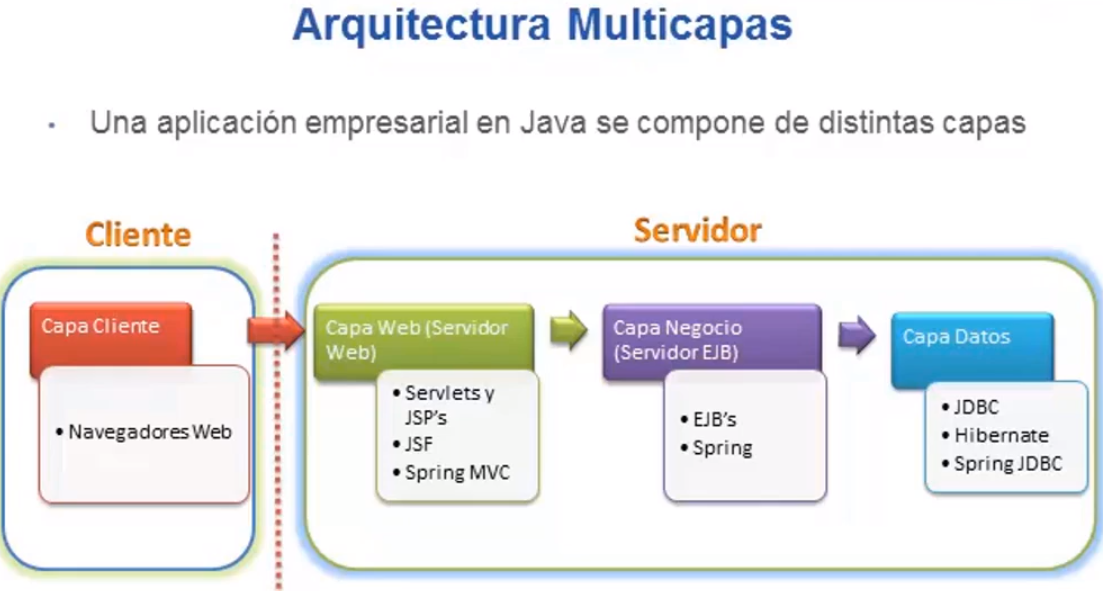
**Spring Context ->** Configuración del framework

**Spring Web ->** Permite la integración de tecnologías web como ser: JSF, Struts

**Spring MVC ->** Implementar MVC



**Spring Arquitectura Multicapas**



**Portafolio de Spring**



**Senapi**

**Usuario Externo**

Vista -> Angular , web services

Modelo -> **JDBC Template** (se implemente fácilmente funciones y procedimientos) más rápido que Hibernate

Se lleva las tablas a objetos, relación de objeto - tabla

Servidor Wildfly

\*\*Fabrica de Beans (Application Context) – busca aquí los métodos

**Usuario Interno**

**Modelo**

Gimodel

JAR – Java Archive

Maven (administrador de librerias) … descargas de .jar automático

pom.xml

**Vista**

giview

WAR – Web Archive

BBDD->Postgress