**¿Cómo funcionan las aplicaciones Java EE?**

Las aplicaciones **Java EE nacen debido a que necesitamos aplicaciones distribuidas**, aplicaciones con muchos componentes que funcionen en entornos separados y al mismo tiempo puedan **estar conectados mediante una red.**

Es por ello que necesitaremos una mente central que una estos componentes, **este será un servidor**, que tenga todos los recursos de velocidad, seguridad y confiabilidad, este debe estar idealmente dedicado a nuestra aplicación.

**En Java EE siempre necesitaremos un servidor, ya sea de pago o gratuito.**

Algunos servidores de aplicaciones famosos son GlassFish, WildFly, Apache etc. Existe otro conocido que no es considerado servidor de aplicaciones sino más bien contenedor de aplicaciones me refiero a **Apache Tomcat**, el cual **esta diseñado para contener aplicaciones Java solamente.**

Un contenedor web es un servidor que se limita solamente a entregar información solicitada por el cliente (solicitud y entrega) y esta funcion es posible gracias a los servlets, normalmente el software que se utiliza es Apache Tomcat.

Un servidor de aplicaciones es un servidor dedicado para dar servicio de una aplicación a una cantidad grande de usuarios (dependiendo de la capacidad de tu servidor) por medio de la red, todas las operaciones que hagas en la aplicacion utiliza recursos del servidor y muy pocos de tu ordenador, es como si manejaras una computadora remota desde tu navegador web, a diferencia de los contenedores web, el software como WebSphere, GlassFIsh gestiona la seguridad y el manejo de usuarios de la aplicación cuando acceses a la red a esta aplicación montada.

Modelo Vista Controlador: es un patrón de arquitectura que facilita el mantenimiento de la aplicación separando los conceptos que en esta habita, esos conceptos son:

* Modelo: Se encarga de gestionar la información que usa la aplicación.
* Vista: Es el medio de entrega de la información ya con un formato legible para el usuario; puede der un jsp, o una consola para nosotros que somos programadores.
* Controlador: Es el intermediario entre la vista y el modelo que se encarga de pedir y entregar información desde el modelo a la vista cuando el usuario solicita nueva información en la aplicación.
* **POJO (Plain Old Java Object)**: Es una instancia de una clase (Objeto) que no extiende ni implementa nada en especial, sirve para enfatizar el uso de clases simples y que no dependen de un framework en especial.

*Por ejemplo, un Controller de Spring tiene que extender de SimpleFormController, e implementar los métodos abstractos de ese tipo: por eso, no es un POJO. Un Servlet, tiene que extender de HttpServlet por lo que tampoco es un POJO. En cambio, si defines una clase Vehículo con atributos y unas cuantas operaciones, tienes un POJO.*

**EJB (Enterprise Java Bean)**: Simplifica la construcción de soluciones empresariales y permite el manejo de transacciones, seguridad, persistencia, etc.

**JavaBean (Bean)**: Es una clase simple en Java que cumple con ciertas normas con los nombres de sus propiedades y métodos.

Servlet: Es una clase que hereda métodos de la clase HttpServlet, una vez heredados tiene la capacidad de manejar cualquier tipo de request, para identificarlo puedes ubicar dentro del código sus métodos doGet o doPost o atraves de su anotación (@WebServlet) + el nombre que le quieras dar como URL.

Bean:

Los **Beans** son similares a los POJO’s solo que **tienen la peculiaridad de que debe implementar la interfaz Serializable**, siempre. Dado que tenemos un proyecto de Java EE y trabajamos sobre una red, necesitamos serializar los objetos para generar transferencias de datos.

Los **EJB** (Enterprise Java Bean) se usan para dar acceso a los servicios del contenedor de aplicaciones (JBoss,WAS,OAS,ETC), estos servicios son seguridad,transacciones,persistencia.etc.

Y otra cosa son los **JavaBean** estos son clases simples en java que debe tener un constructor sin argumentos, tiene declarado todos sus atributos como privados y para cada uno de ellos tiene un método setter y getter.¿Para que sirven? para encapsular varios objetos en un único objeto y así hacer uso de un solo objeto en lugar de varios.

Archivos War:

WAR (Web ARchive): Es un archivo que contiene componentes para una aplicacion web de Java El archivo war. almacenará todos los recursos de la aplicación como archivos jsp, js, css etc. y en cuánto al código sólo almacenará los archivos de bytecode agregando una capa de seguridad.

**Hibernate es un ORM (Object Relational Mapping)**

* Es una herramienta que nos permite mapear una base de datos
* Hibernate esta diseñado para trabajar con bases de datos relacionales.
* La idea de Hiberante es tener una clase equivalente a una tabla de base de datos incluyendo los campos.
* Podremos operar los datos operando no sobre tablas sino sobre objetos
* Busca solucionar el problema de la diferencia entre dos modelos de datos coexistentes en una aplicación
* Podemos definir tipos de datos
* Generar sentencias SQL

Hibernate permite conectarnos a una base de datos a partir de un archivo de configuración llamado **hibernate.cfg.xml**.

**Mapearemos las tablas y campos** de una base de datos a partir de **Anotaciónes @** en las clases

**DAO significa Data Access Object**

El cuál es un patrón de diseño que nos permite crear un API de un Objeto específico.

Un APi significará crear todas las acciones que se pueden realizar hacia un objeto.  
Por ejemplo un CRUD:

* Create
* Read
* Update
* Delete

Generaremos un **DAO de cada entidad** para ello debemos crear una **Interfaz** donde se declaren los métodos de API y **una clase que implemente la interfaz.**

<http://download.jboss.org/jbosstools/oxygen/snapshots/updates/>

**Librería:** conjunto de clases, métodos etc. que son invocadas por el flujo del programa y que posteriormente devuelven el control a este.

**Framework:** Un framework controla el flujo del código, él decide cuándo llamar nuestro código.

* Librería - Programación secuencial.
* Framework - Programación por eventos.

**Inversión de Flujo o de control**

Se da cuando el programador deja de tener el control del flujo del código y en su lugar el framework toma el control del programa, llamando al código cuando suceda un evento específico.

**Spring usa inversión de control, específicamente controlará la creación y destrucción de objetos.**

Un programa tradicional de java tendrá en su método main todas las llamadas a los objetos secuencialmente.

Con Spring framework tendremos las llamadas a los objetos se concentrarán en un contenedor y de este se encargará Spring.

**Podemos decir que la clase A para ser llamada y convertirse en objeto, depende de la clase Principal (la que contiene el método main) la llame. Entonces si se implementa el contenedor de Spring en la clase Principal decimos que Spring está inyectando el objeto A, es decir Spring está inyectando la dependencia.**

Tenemos lo siguiente:

Clase Vehículo - A  
Clase Motor - B

En el ejemplo la clase Vehículo inyecta directa y explícitamente la clase Motor, la clase A inyecta la clase B.

Debemos evitar que la clase B decida qué objetos crear.

**La forma de desacoplar esta dependencia directa es creando una clase más genérica, nos referimos a una interfaz que tenga métodos que puedan ser implementados en varias clases, esta Interfaz será la indicada a inyectar en la clase A (Vehículo) pues al ser genérica se puede crear no solo un tipo de objeto sino varios.**

Recuerda que cuando sobreescribimos métodos y le damos diferente comportamiento a ese comportamiento le llamamos Polimorfismo

Los objetos que son la columna vertebral de tu aplicación y que son administrados por el contenedor Spring se denominan **beans**.

**Un bean** en Spring será un objeto que **tenga consigo un CRUD de acciones**

Los DAO’s son Beans, pero usaremos su interfaz pues es el elemento más abstracto y reutilizable.

También tendremos objetos de servicio y objetos que nos permitan conectarnos con otras partes del sistema como Bases de Datos.

Cuando nuestra interfaz **implementa solo a una clase** usaremos la anotación **@Autowired**  
Cuando la interfaz **implementa a varias clases** usaremos la anotación **@Configuration** y **@Bean**