JRE: es un componente de Java Runtime Environment (JRE). JRE permite que algunas aplicaciones escritas en lenguaje de programación Java se inicien a través de ciertos navegadores. El software Java Plugin no es un programa independiente y no puede instalarse por separado.

JVM: Java Virtual Machine es solo una parte del software de Java que se encarga de ejecutar una aplicación. Java Virtual Machine está incluido en la descarga del software Java, forma parte de JRE y ayuda a ejecutar aplicaciones Java.

Programación orientada a objetos y lenguaje Java

Clases, Objetos y Métodos

Objeto

Es una combinación de unos datos específicos y de las rutinas que pueden operar con esos datos. De forma que los dos tipos de componentes de un objeto son:

- Campos o *atributos*: componentes de un objeto que almacenan datos. También se les denomina variables miembro. Estos datos pueden ser de tipo primitivo (`boolean, int, double, char...`) o, a su vez, de otro tipo de objeto (lo que se denomina *agregación o composición* de objetos). La idea es que un atributo representa una propiedad determinada de un objeto.
- Rutinas o *métodos*: es una componente de un objeto que lleva a cabo una determinada acción o tarea con los atributos. En principio, todas las variables y rutinas de un programa de Java deben pertenecer a una clase. De hecho, en Java no hay noción de programa principal y los subrutinas no existen como unidades modulares independientes, sino que forman siempre parte de alguna clase.

Clase

Representa al conjunto de objetos que comparten una estructura y un comportamiento comunes.

Modificadores de visibilidad:

- **Public**: Un método declarado como public puede ser accedido desde cualquier parte del programa, ya sea dentro de la misma clase, dentro de otra clase en el mismo paquete, o en una clase en un paquete diferente.

- **Private:** Un método declarado como private sólo puede ser accedido desde dentro de la misma clase en la que se ha definido.
- **Protected:** Un método declarado como protected puede ser accedido desde dentro de la misma clase, desde cualquier clase en el mismo paquete, y desde cualquier clase hija (que herede de la clase que define el método).
- **Default:** Un método declarado sin especificar ninguna visibilidad (es decir, sin usar public, private o protected) sólo puede ser accedido desde dentro de la misma clase y desde cualquier clase en el mismo paquete.

Tipos de Datos

- **Primitivos:** Son los tipos de datos por defecto o los de siempre.
- **Referencia:** Se utilizan para crear objetos. Refieren una dirección de memoria en la cual se almacena un objeto real.

Constructor

Método especial que se utiliza para inicializar un objeto recién creado. Es decir, es el método que se ejecuta automáticamente al momento de crear una instancia de una clase mediante la palabra clave "new".

Tiene el mismo nombre que la clase y no tiene tipo de retorno, ni siquiera void. Su objetivo principal es establecer los valores iniciales de los atributos de la instancia de la clase que se está creando.

This:

Es una palabra reservada en Java que se utiliza para referirse a la instancia actual de una clase. Se refiere al objeto actual en el que se está trabajando. Puedes utilizar this() dentro de un constructor para llamar a otro constructor de la misma clase con diferentes argumentos.

Herencia y polimorfismo

Polimorfismo: Cuando se tienen 2 métodos con el mismo nombre y diferentes atributos. Capacidad de los objetos de una clase para tomar diferentes formas y comportamientos en función del contexto en el que se utilicen. En este sentido, el polimorfismo en Java se logra a través de la sobrecarga y la sobreescritura de métodos.

- **Polimorfismo por Sobrecarga:** Capacidad de una clase para tener múltiples métodos con el mismo nombre pero diferentes parámetros. Java

es capaz de determinar cuál método debe ser llamado en función del número y tipo de argumentos que se pasan.
- **Polimorfismo por Sobreescritura:** Permite que las subclases de una clase hereden métodos de su superclase, y luego redefinan o sobre escriban esos métodos en la subclase para adaptarlos a sus necesidades específicas. Cuando se llama a un método sobreescrito en una subclase, Java llama al método de la subclase en lugar del método de la superclase, siempre que el objeto sea una instancia de la subclase.

Herencia

La herencia es un mecanismo fundamental de la programación orientada a objetos que permite que una clase, llamada subclase o clase derivada, herede los campos y métodos de otra clase, llamada superclase o clase base. En Java, la herencia se logra mediante la palabra clave extends.

La subclase hereda los campos y métodos de la superclase y puede agregar nuevos campos y métodos o anular o sobrescribir los campos y métodos heredados según sea necesario. La subclase también puede llamar a los constructores de la superclase para inicializar los campos heredados de la superclase.

En Java, una clase solo puede heredar de una superclase a la vez (es decir, no se admite la herencia múltiple). Sin embargo, una clase puede implementar múltiples interfaces, lo que proporciona una forma de lograr funcionalidad similar a la herencia múltiple.

Interfaces y clases abstractas

Clase Abstracta

Una clase abstracta en Java es una clase que no puede ser instanciada, es decir, no se puede crear un objeto directamente a partir de ella. En cambio, las clases abstractas se utilizan como plantillas o modelos para definir una clase base que se puede extender o heredar en otras clases.

Las clases abstractas se declaran utilizando la palabra clave "abstract" y pueden contener métodos abstractos (es decir, métodos que no tienen implementación y que deben ser implementados por cualquier subclase que herede de la clase abstracta) y métodos concretos (es decir, métodos que tienen una implementación concreta y que se pueden heredar directamente).

Las clases abstractas se utilizan a menudo para definir una clase base que proporciona una funcionalidad común a varias subclases, pero que no tiene sentido instanciar directamente.

Es importante tener en cuenta que, aunque una clase abstracta no se puede instanciar directamente, se puede utilizar para declarar una referencia de objeto y para polimorfismo.

Diferencia entre clase abstracta y herencia

Las clases abstractas sirven como una forma de definir una estructura o un contrato que las clases hijas deben seguir. Cuando una clase es declarada como abstracta, significa que no puede ser instanciada directamente, sino que debe ser heredada por una subclase que la implemente.

Las clases abstractas son útiles cuando se quiere definir un conjunto de métodos que las subclases deben implementar, pero que no necesariamente tienen una implementación específica en la clase abstracta. De esta manera, se puede garantizar que las subclases tengan cierto comportamiento y funcionalidad común, al tiempo que permiten que cada subclase tenga su propia implementación específica de los métodos.

En resumen, las clases abstractas permiten definir una estructura común y obligatoria para las clases hijas, mientras que la herencia permite que las clases hijas hereden atributos y métodos de la clase padre, y los extiendan o los modifiquen según sea necesario.

Interfaz

Es una colección de métodos abstractos (es decir, métodos sin implementación) y constantes (variables que no pueden ser modificadas una vez que se han asignado).

A diferencia de las clases, las interfaces no pueden ser instanciadas directamente, pero pueden ser implementadas por otras clases. Una clase que implementa una interfaz debe proporcionar implementaciones para todos los métodos definidos en la interfaz.

Las interfaces se utilizan principalmente para definir un conjunto de métodos que deben ser implementados por diferentes clases, sin especificar la implementación exacta de estos métodos. Esto permite a diferentes clases proporcionar su propia implementación de estos métodos de acuerdo con sus necesidades específicas, mientras se asegura que se proporciona una interfaz común para trabajar con objetos de estas diferentes clases.

En Java, una clase solo puede heredar de una superclase a la vez (es decir, no se admite la herencia múltiple). Sin embargo, una clase puede implementar múltiples

interfaces, lo que proporciona una forma de lograr funcionalidad similar a la herencia múltiple

Diferencias entre interfaz y clase abstracta Java

Sí, en Java hay algunas diferencias entre una interfaz y una clase abstracta:

1. Una interfaz solo puede contener métodos abstractos, mientras que una clase abstracta puede contener métodos abstractos y métodos concretos (es decir, métodos con implementación).
2. Una clase abstracta puede tener campos (variables de instancia), mientras que una interfaz solo puede tener constantes (variables que no pueden ser modificadas una vez que se han asignado).
3. Una clase puede implementar varias interfaces, pero solo puede extender una clase abstracta.
4. Los métodos de una interfaz son por defecto públicos y abstractos, mientras que los métodos de una clase abstracta pueden tener cualquier nivel de acceso y pueden ser abstractos o no abstractos.
5. En una interfaz, los métodos deben ser implementados por cualquier clase que la implemente. En una clase abstracta, los métodos abstractos deben ser implementados por las subclases, pero los métodos concretos pueden ser heredados y usados tal como están.

En general, se recomienda utilizar una interfaz cuando se quiere definir un conjunto de métodos que deben ser implementados por diferentes clases, y una clase abstracta cuando se quiere proporcionar una implementación base común para un conjunto de subclases relacionadas.

Comparación Herencia, Interfaz y Abstractas

	Intefaz	Clase Abstracta	Clase Padre
¿Se puede instanciar?	No	No	Sí

Extensión	No	Sí	Sí
Implementación	Sí	Sí	No
Métodos	Sólo métodos abstractos	Métodos abstractos y concretos	Métodos concretos
Variables	Sólo variables constantes	Variables constantes y modificables	Variables constantes y modificables
Constructores	No	Sí	Sí
Herencia múltiple	Sí	No	No

Excepciones

En Java, las excepciones son objetos que se utilizan para manejar errores y situaciones excepcionales que pueden ocurrir durante la ejecución de un programa. Cuando ocurre una excepción, se lanza un objeto de excepción que contiene información sobre lo que salió mal y en qué lugar del código ocurrió.

Existen tres tipos de excepciones en Java:

1. Excepciones comprobadas (checked exceptions): son aquellas que el compilador obliga a manejar, ya que pueden ocurrir durante la ejecución del programa. Ejemplos de excepciones comprobadas son IOException, SQLException, etc.

2. Excepciones no comprobadas (unchecked exceptions): son excepciones que no están obligadas a ser manejadas por el compilador. Suelen ser excepciones que ocurren por errores de programación o errores en tiempo de ejecución. Ejemplos de excepciones no comprobadas son NullPointerException, IndexOutOfBoundsException, etc.
3. Errores (errors): son problemas graves que no pueden ser manejados por el programa. Ejemplos de errores son OutOfMemoryError, StackOverflowError, etc.

Colecciones y genéricos

Colección: es un objeto que puede contener un conjunto de elementos del mismo tipo o de tipos diferentes. Las colecciones son utilizadas para almacenar, manipular y procesar datos de manera más eficiente y flexible que los arreglos tradicionales.

Genérico: son una característica que permite definir clases, interfaces y métodos que pueden trabajar con diferentes tipos de objetos, proporcionando mayor seguridad de tipos y reutilización de código.

Threads y concurrencia

Threads:

Es una forma de ejecución concurrente o simultánea de diferentes partes de un programa. La concurrencia se refiere a la capacidad de una aplicación para manejar múltiples tareas simultáneamente. Java proporciona soporte para la concurrencia mediante el uso de threads.

Un thread es un subproceso o subproceso de ejecución independiente que puede ser iniciado y ejecutado en paralelo con otros threads. En Java, un thread se puede crear extendiendo la clase Thread o implementando la interfaz Runnable. La clase Thread proporciona los métodos necesarios para crear, iniciar y controlar un thread, mientras que la interfaz Runnable define el método run(), que se utiliza para escribir el código que se ejecutará en el thread.

La concurrencia se refiere a la capacidad de un programa para manejar múltiples tareas simultáneamente. En Java, se pueden crear threads para lograr la concurrencia en una aplicación. Un thread puede ejecutarse de manera independiente y simultánea a otros threads, lo que permite que una aplicación maneje varias tareas al mismo tiempo.

Sin embargo, la concurrencia también puede presentar desafíos en términos de sincronización y gestión de recursos compartidos. Si varios threads intentan acceder a los mismos recursos al mismo tiempo, puede ocurrir una condición de carrera o un problema de sincronización. En Java, se pueden utilizar diversas

herramientas para abordar estos problemas, como los bloqueos, los semáforos y los monitores.

En resumen, los threads y la concurrencia son una parte importante de la programación en Java y se utilizan para mejorar el rendimiento y la capacidad de respuesta de las aplicaciones, pero también presentan desafíos en términos de sincronización y gestión de recursos compartidos.

Desarrollo web con Java: Spring, Swing, Servlets, JSP y JSF

Herramienta/Framework	Características	Limitaciones
Spring Framework	Amplia gama de funcionalidades para el desarrollo empresarial en Java.	Curva de aprendizaje empinada.
Spring MVC	Framework para el desarrollo de aplicaciones web basadas en Spring.	Puede ser complejo de configurar.
Java Server Faces (JSF)	Proporciona un conjunto de componentes UI predefinidos.	Requiere una gran cantidad de recursos en el lado del servidor.
Servlets	Manejo de peticiones HTTP en el servidor.	Requiere más trabajo manual para el desarrollo que otros frameworks.
JSP	Tecnología para la creación de vistas dinámicas en Java.	Puede ser difícil de mantener en aplicaciones grandes.

Herramienta/Framework	Características	Limitaciones
Swing	Framework para la creación de interfaces de usuario de escritorio en Java.	Solo es adecuado para aplicaciones de escritorio.

Frontend:

1. JSF -> esta es más moderna y utiliza componentes reutilizables
2. JSP -> esta se puede incluir código java junto a html
3. Spring
4. Servelt

Ambas opciones son viables para desarrollar el frontend de tu aplicación en Java. Sin embargo, aquí te presento algunas consideraciones que podrían ayudarte a decidir entre JSF y JSP:

- JSF es un framework más moderno que JSP y se enfoca en la creación de interfaces de usuario utilizando componentes reutilizables. Si te interesa tener una mayor organización en tu código y utilizar componentes para crear tus vistas, JSF podría ser una buena opción.
- JSP, por otro lado, es un lenguaje de programación de servidor que permite incluir código Java en tus páginas HTML. Si prefieres tener un mayor control sobre el código HTML que se genera y no necesitas utilizar componentes reutilizables, JSP podría ser la opción más sencilla.

En resumen, si buscas una solución más moderna y organizada, JSF podría ser una buena opción. Si prefieres algo más sencillo y con mayor control sobre el HTML generado, JSP podría ser tu elección.

Backend:

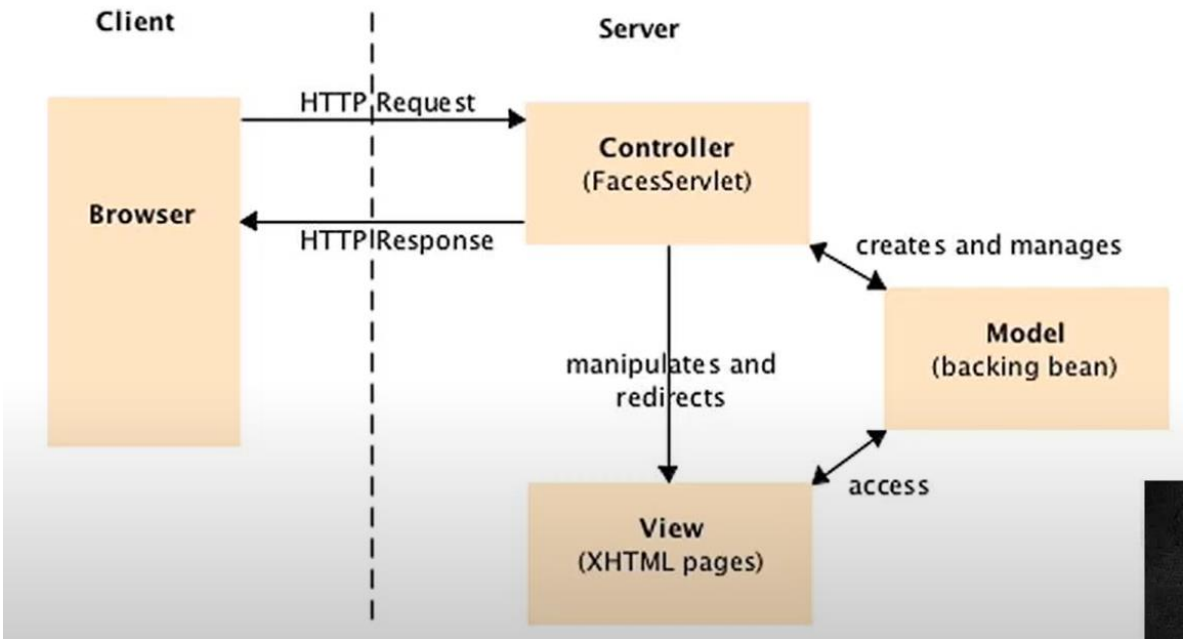
Srping

- Spring: Para el desarrollo del backend, Spring es una excelente opción, ya que proporciona una gran cantidad de funcionalidades y herramientas para la creación de aplicaciones web robustas y escalables.

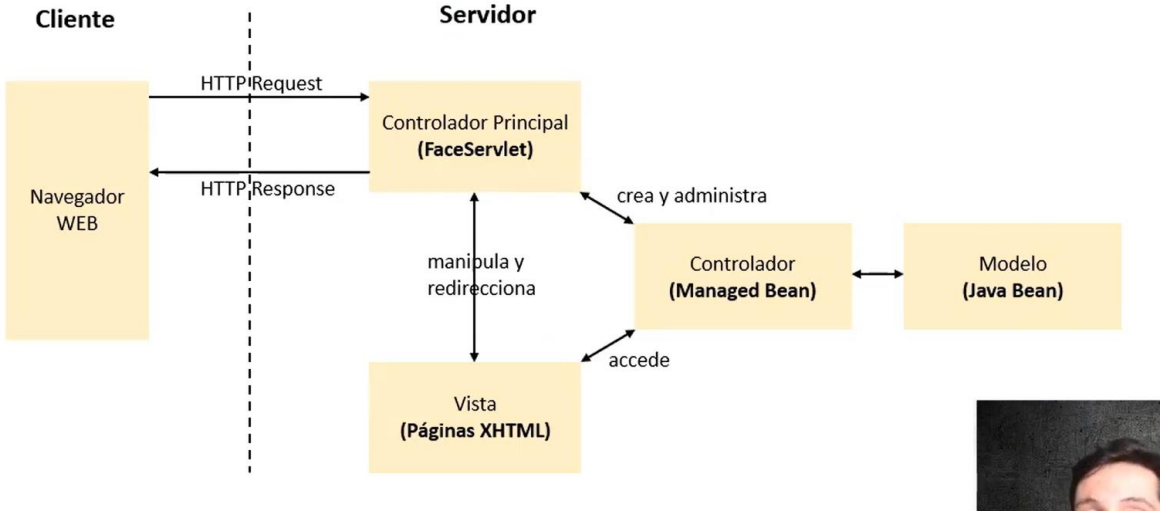
Aplicación de Escritorio:

JSF

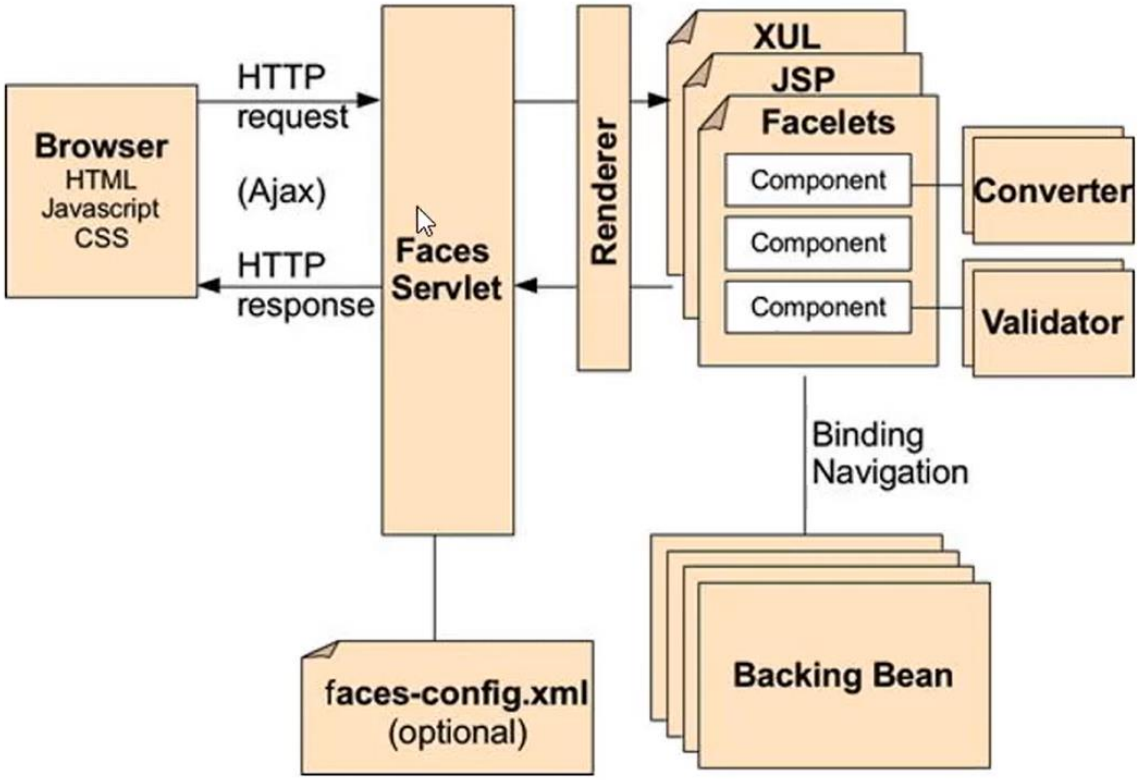
Arquitectura JSF – Model View Controller



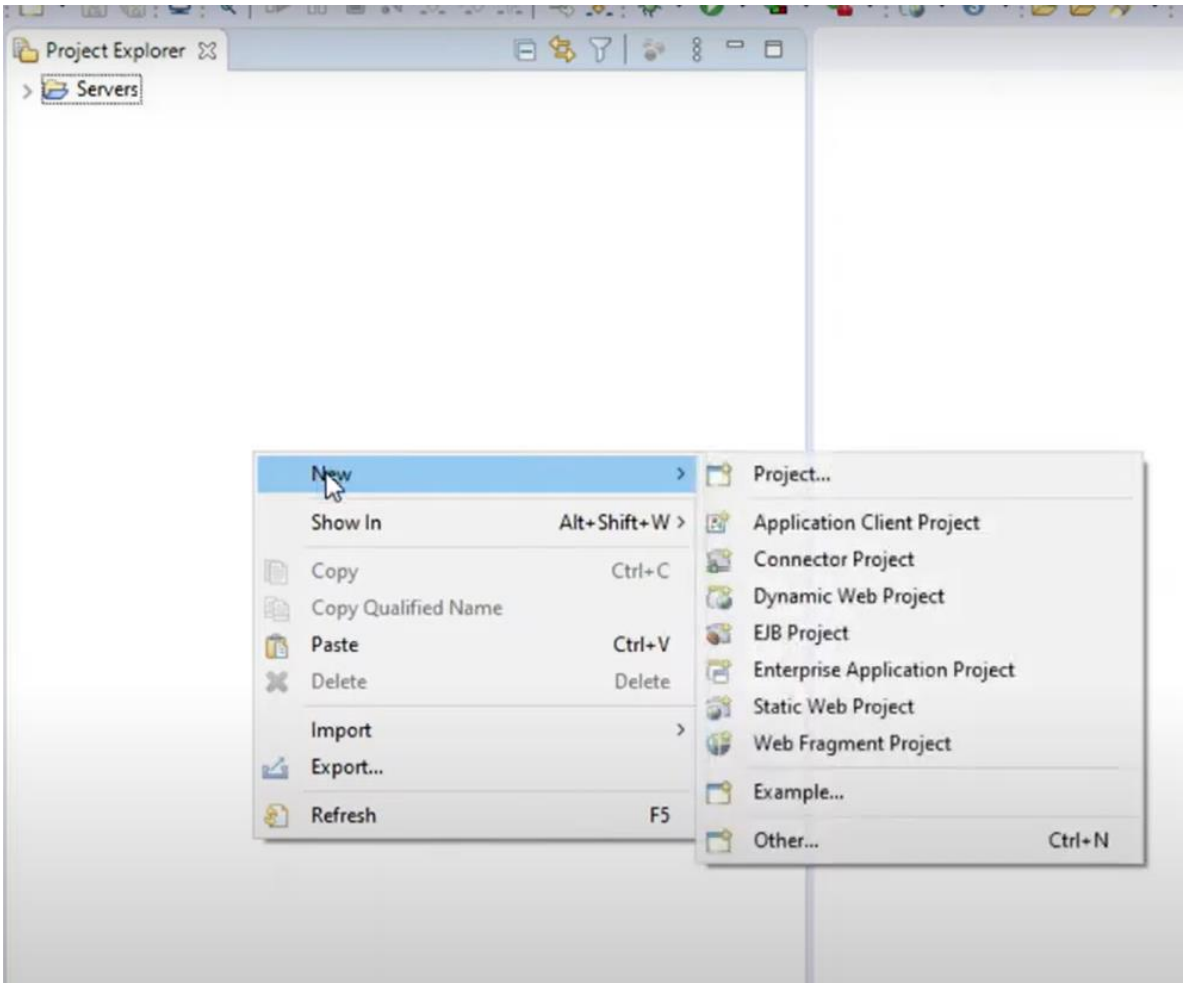
Arquitectura JSF – Model View Controller

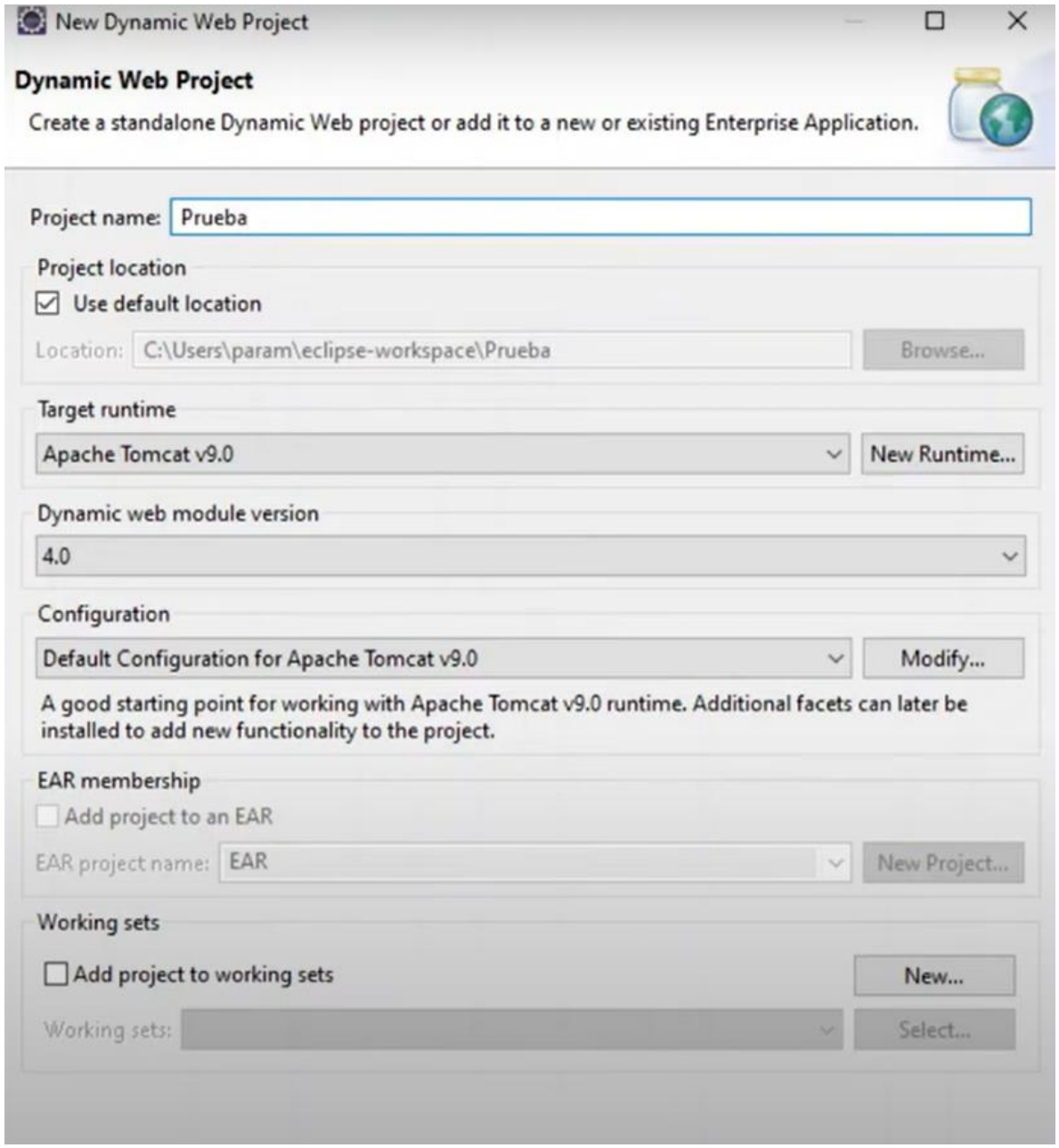


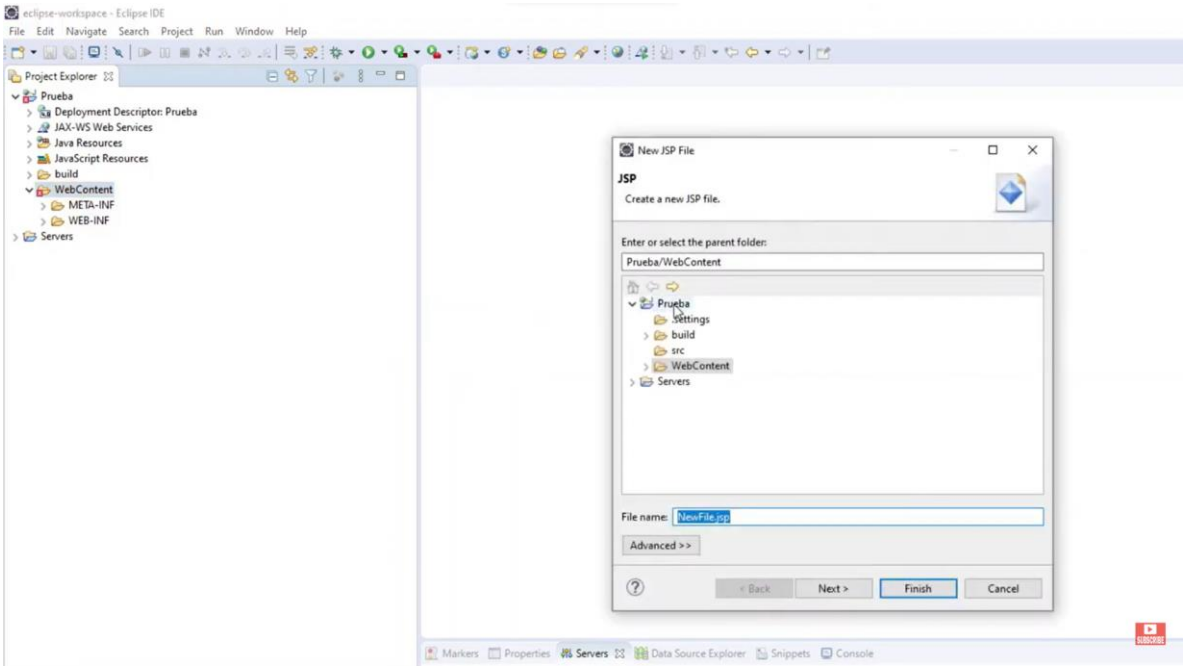
Componentes JSF – Ciclo de Vida

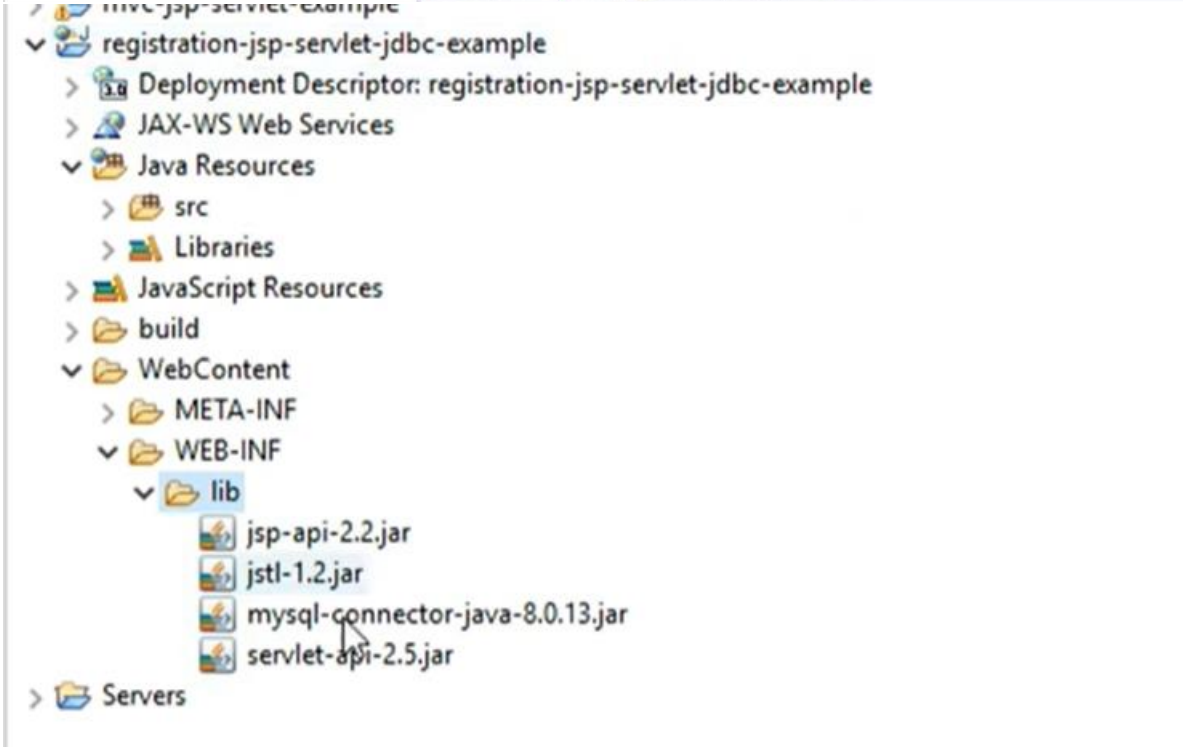


JSP


Project Explorer
Servers
New
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Import
Export...
Refresh F5
Project...
Application Client Project
Connector Project
Dynamic Web Project
EJB Project
Enterprise Application Project
Static Web Project
Web Fragment Project
Example...
Other... Ctrl+N


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Project name: Prueba
Project location
Use default location
Location: C:\Users\param\eclipse-workspace\Prueba
Browse...
Target runtime
Apache Tomcat v9.0
New Runtime...
Dynamic web module version
4.0
Configuration
Default Configuration for Apache Tomcat v9.0
Modify...
A good starting point for working with Apache Tomcat v9.0 runtime. Additional facets can later be installed to add new functionality to the project.
EAR membership
Add project to an EAR
EAR project name: EAR
New Project...
Working sets
Add project to working sets
New...
Working sets:
Select...


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
Prueba
Deployment Descriptor: Prueba
JAX-WS Web Services
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Servers
New JSP File
JSP
Create a new JSP file.
Enter or select the parent folder:
Prueba/WebContent
Prueba
Settings
build
src
WebContent
Servers
File name:
NewFile.jsp
Advanced >>
< Back
Next >
Finish
Cancel
Markers
Properties
Servers
Data Source Explorer
Snippets
Console


registration-jsp-servlet-jdbc-example
Deployment Descriptor: registration-jsp-servlet-jdbc-example
JAX-WS Web Services
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
lib
jsp-api-2.2.jar
jstl-1.2.jar
mysql-connector-java-8.0.13.jar
servlet-api-2.5.jar
Servers

Pasos Para Crear Objetos

1. Creación de la Clase

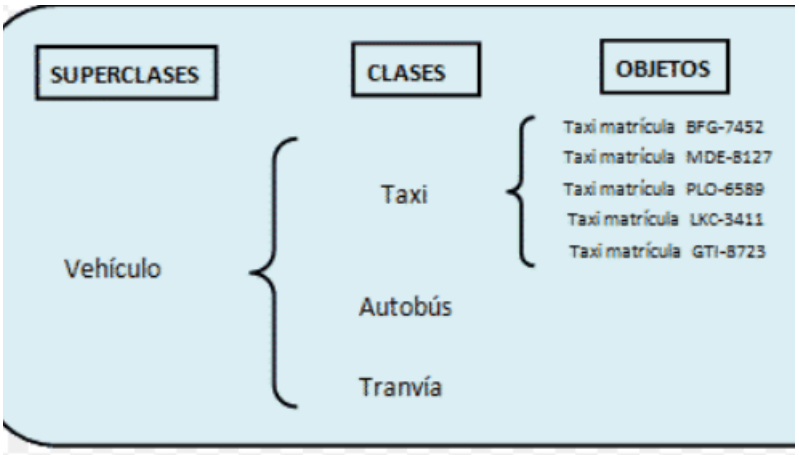
```
class Fecha
{
// declaración de
//variables
//declaración de los
//métodos
}
```

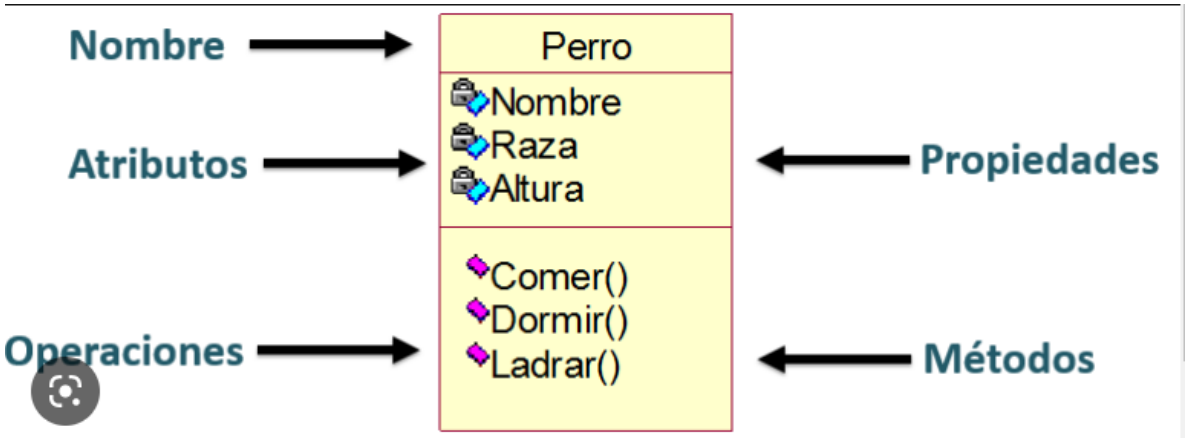
2. Declarar los Objetos

```
Fecha reunion;
```

3. Crear los Objetos

```
reunion = new Fecha ( );
```





https://www.arkaitzgarro.com/java/capitulo-9.html

Links referencia

- https://github.com/Audrie8a/MIA_PROYECTO2/blob/master/Documentacion/DDL_%23201801263.sql
- https://github.com/AAudrie8a/Proyecto_Sopes2/blob/develop/app/frontend/src/Componentes/Menu.jsx
- https://github.com/danielniko/SimpleJspServletDB/tree/master
    - https://www.youtube.com/watch?v=UBOZgNxsiwA&list=PLGRDMO4rOGcOjhFoLV2xOfRQqYjpSVhxt&index=5
- https://www.youtube.com/watch?v=DzYyzmP4m5c&list=PLGRDMO4rOGcOjhFoLV2xOfRQqYjpSVhxt&index=1
    - https://github.com/RameshMF/servlet-tutorial/tree/master
- https://www.youtube.com/watch?v=omppULzvao0
-