www.uniremington.edu.co

UNIREMINGTON®
CORPORACIÓN UNIVERSITARIA REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996

LENGUAJE DE PROGRAMACIÓN III

TRANSVERSAL

FACULTAD DE CIENCIAS BÁSICAS E INGENIERÍA

Vicerrectoría de Educación a Distancia y virtual

2016

AUTOR

Cesar Augusto Jaramillo Henao
Ingeniero de Sistemas
Cesar.jaramillo@remington.edu.co

Nota: el autor certificó (de manera verbal o escrita) No haber incurrido en fraude científico, plagio o vicios de autoría; en caso contrario eximió de toda responsabilidad a la Corporación Universitaria Remington, y se declaró como el único responsable.

RESPONSABLES

Jorge Mauricio Sepúlveda Castaño
Decano de la Facultad de Ciencias Básicas e Ingeniería
jsepulveda@uniremington.edu.co

Eduardo Alfredo Castillo Builes
Vicerrector modalidad distancia y virtual
ecastillo@uniremington.edu.co

Francisco Javier Álvarez Gómez
Coordinador CUR-Virtual
falvarez@uniremington.edu.co

GRUPO DE APOYO

Personal de la Unidad CUR-Virtual
EDICIÓN Y MONTAJE
Primera versión. Febrero de 2011.
Segunda versión. Marzo de 2012
Tercera versión. noviembre de 2015

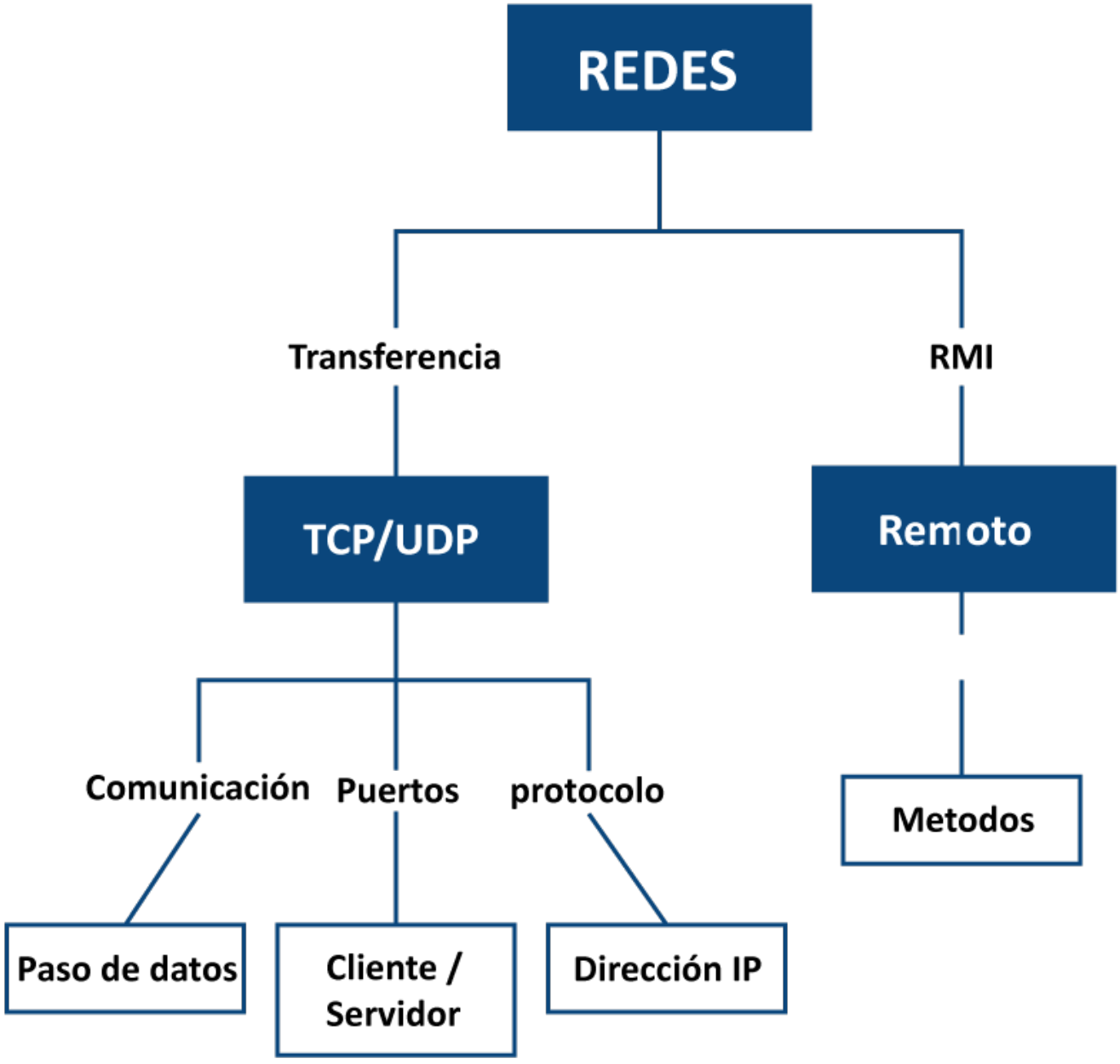
TABLA DE CONTENIDO

Pág.

1 MAPA DE LA ASIGNATURA

2 UNIDAD 1 REDES

2.1.1 RELACIÓN DE CONCEPTOS



Escriba la definición de todos los conceptos plateados en el mapa conceptual

Servidor: Es un aplicativo gestor de la información, provee los recursos que se necesita por parte de un cliente

Cliente: Es un aplicativo que solicita información a un servidor

Dirección IP: Es la ubicación única dentro de una red

TCP: Protocolo de control de transmisión

UDP: Protocolo de nivel de transporte

OBJETIVO GENERAL

Identificar las características de la programación en red, los recursos de comunicación los protocolos y el manejo de los datos.

OBJETIVOS ESPECÍFICOS

- Identificar las características principales para programar en Red
- Identificar los componentes esenciales para la programación en red
- Identificar los comandos más comunes para la programación en red

2.2 TEMA 1 CONCEPTOS BÁSICOS

Muchos de los aplicativos que escribimos están diseñados para ser utilizados en una sola máquina, esto es muy limitante por el crecimiento constante de las empresas y de estar conectado a los distintos recursos que se pueden utilizar.

Para este propósito debemos de familiarizarnos con el manejo de los protocolos, esencialmente dos de ellos que nos permiten realizar esta tarea, TCP (**Transmission Control Protocol**) y UDP (User Datagram Protocol), estos protocolos implementan lo que conocemos como la capa de transporte.

PISTAS DE APRENDIZAJE

Redes: El trabajo en red es fundamental para los requerimientos de la empresa actual

2.3 TEMA 2 TCP / UDP

TCP

> Es un protocolo orientado a conectar dos equipos de manera segura y confiable, cuando la comunicación se establece se crea un canal por medio del cual se puede enviar y recibir información, el protocolo TCP garantiza que los datos que se envían serán íntegros, de lo contrario reportara un error.

La comunicación TCP es análoga a una comunicación telefónica, en que un usuario llama y el otro determina o no atenderlo, cuando decide atenderlo establecen una "conversación" de forma bidireccional.

Dentro de los procesos más comunes de este tipo de protocolo están FTP, Telnet, HTTP, en estos procesos es fundamental respetar el orden de envío de las tareas.

UDP

La comunicación establecida mediante este protocolo no es confiable ni garantizada como en el caso de TCP, esto debido a que UDP no es un protocolo de conexión, en el UDP se envían paquetes de datos llamados datagramas, el envío de estos es comparable con el envió del correo o correspondencia tradicional, en este ejemplo nos encontramos que el envío de una carta no nos preocupa en qué orden llega a su destino.

PUERTO

Los puertos son los mecanismos para hacer llegar la información al aplicativo que lo solicito, cada pc tiene una única conexión física por medio de la cual se recibe la información, los puertos constituyen una dirección interna que direcciona un proceso dentro del equipo de cómputo.

DIRECCIÓN IP

Una dirección IP (Internet Protocol), es un numero de 32 bits que direcciona de manera única a un pc dentro de la red.

APLICACIÓN CLIENTE / SERVIDOR

Es un orden jerárquico de las aplicaciones de una red, una aplicación cliente solicita información a una aplicación servidor, este último proveerá los servicios a un cliente según las características del aplicativo gestor.

SOCKETS

Es conocido como uno de los extremos en una comunicación de programas, es la forma de comunicar un servidor con un cliente, este socket direcciona la información de forma única a la aplicación solicitante.

SERVIDOR

Es un programa que permite la que se conecten los distintos programas clientes, esto se conoce como “escuchar” un cliente”

CLASES COMUNES

ServerSocket: Se utilizar para esperar y escuchar la llegada de los clientes

Socket: Se puede entablar la comunicación cliente/servidor

EJEMPLO DE SERVIDOR

```
public class Servidor {

    public static void main (String args []) throws Exception   {

        ObjectInputStream objectInputStream = null;
        ObjectOutputStream objectOutputStream = null;
        Socket socket = null;
        ServerSocket serverSocket = new ServerSocket(5432);

        while (true)    {

            try {

                socket = serverSocket.accept();
                System.out.print ("\nSe Conectaron desde la IP: " + socket.getInetAddress());
                objectInputStream = new ObjectInputStream (socket.getInputStream());
                objectOutputStream = new ObjectOutputStream (socket.getOutputStream());

                String nombre = (String) objectInputStream.readObject();
                String saludo = "Hola " + nombre + " " + System.currentTimeMillis();

                objectOutputStream.writeObject(saludo);
                System.out.print ("\nSaludo Viajando...");

            }
```

Autoria Propia

```
            catch (Exception ex){

                ex.printStackTrace();
            }
            finally {

                if (objectOutputStream != null)

                    objectOutputStream.close();
                if (objectInputStream != null)

                    objectInputStream.close();
                if (socket != null)

                    socket.close();

                System.out.print ("\nConexion cerrada");

            }
        }
    }
}
```

Autoria Propia

El servidor instancia un **ServerSocket** con un puerto aleatorio que como ejemplo se tendrá el 5432, la instrucción **acept** es la encarda de esperar la conexión de un cliente. La instrucción **getInnetAddress** tomara la **IP** del cliente, el manejo de los datos de la forma tradicional envia solo bytes pero con las clases **ObjectInputStream** y/o *ObjectOutputStream* se procesa como objetos, estas clases leen y escriben objetos por medio de la red.

CLIENTE

```
public class Cliente {

public static void main (String args []) throws Exception   {

        ObjectInputStream objectInputStream = null;
        ObjectOutputStream objectOutputStream = null;
        Socket socket = null;

        try {

            socket = new Socket ("127.0.0.1", 5432);
            objectOutputStream = new ObjectOutputStream (socket.getOutputStream());
            objectInputStream = new ObjectInputStream (socket.getInputStream());

            objectOutputStream.writeObject("Jose");

            String ret = (String) objectInputStream.readObject();
            System.out.print ("\n" + ret);
        }
        catch (Exception ex){

            ex.printStackTrace();
        }
```

Autoria Propia

```
        finally {

            if (objectInputStream != null)

                objectInputStream.close();
            if (objectOutputStream != null)

                objectOutputStream.close();

            if (socket != null)

                socket.close();
        }
    }
}
```

Autoria Propia

Para el cliente comunicarse con el servidor, se tiene un puerto y una dirección IP, para este caso se aplica un servidor local.

RMI

RMI (**Remote Method Invocation**), es una tecnología de invocación remota de métodos, como su nombre lo indica invoca métodos, cuando estos se encuentran en una máquina virtual y **los llama de otra máquina virtual**, esto se conoce como objeto remoto.

El servidor se encarga de instanciar los objetos remotos y los hace disponibles al cliente, esto se ubica en una colección o repositorio de objetos.

Los objetos remotos son los publicados por el servidor a los que se podrán acceder por el cliente remotamente, ambas maquinas utilizan para esta tarea la máquina virtual, a la hora de considerar que un objeto sea remoto deberá heredar la clase **java.rmi. UnicastRemoteObject**.

PISTAS DE APRENDIZAJE

Métodos: Identificar los distintos métodos que componen las tareas de la comunicación en RED

RMI: Herramienta de java para invocar métodos remotos

2.4 TEMA 3 APLICACIÓN

A continuacion se vera un pequeño ejemplo de RMI

Para esto se crea un proyecto llamado RMI, un archivo ObjetoRomoto.java, este archivo es una interfaz y dara caracteristicas generales del proyecto

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ObjetoRemoto extends Remote {

    public String obtenerSaludo (String nombre) throws RemoteException;
}
```

Autoria Propia

El seguinte archivo es una clas tradicional llamada ObjetoRemotoImplementacion.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ObjetoRemotoImplementacion extends UnicastRemoteObject

implements ObjetoRemoto {

    protected ObjetoRemotoImplementacion() throws RemoteException {

        super();
    }

    public String obtenerSaludo(String nombre) throws RemoteException {

        return "Hola RMI" + nombre;
    }
}
```

Autoria Propia

ServidorRMI

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ServidorRMI {

    public static void main (String [] args) throws Exception {

        ObjetoRemotoImplementacion objetoRemotoImplementacion = new ObjetoRemotoImplementacion ();
        Registry registry = LocateRegistry.getRegistry(1099);
        registry.rebind("OBJRemoto", objetoRemotoImplementacion);
    }

}
```

Autoria Propia

ClienteRMI

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ClienteRMI {

    public static void main (String args []) throws Exception {

        Registry registry = LocateRegistry.getRegistry("127.0.0.1", 1099);
        ObjetoRemoto objetoRemoto;
        objetoRemoto = (ObjetoRemoto) registry.lookup("OBJRemoto");
        String saludo = objetoRemoto.obtenerSaludo("Alexandra");
        System.out.print ("\n" + saludo);

    }
}
```

Autoria Propia

PISTAS DE APRENDIZAJE

Remoto: **Proceso que se encuentra en un lugar distinto a la ubicación inicial**

2.4.1 EJERCICIO DE APRENDIZAJE

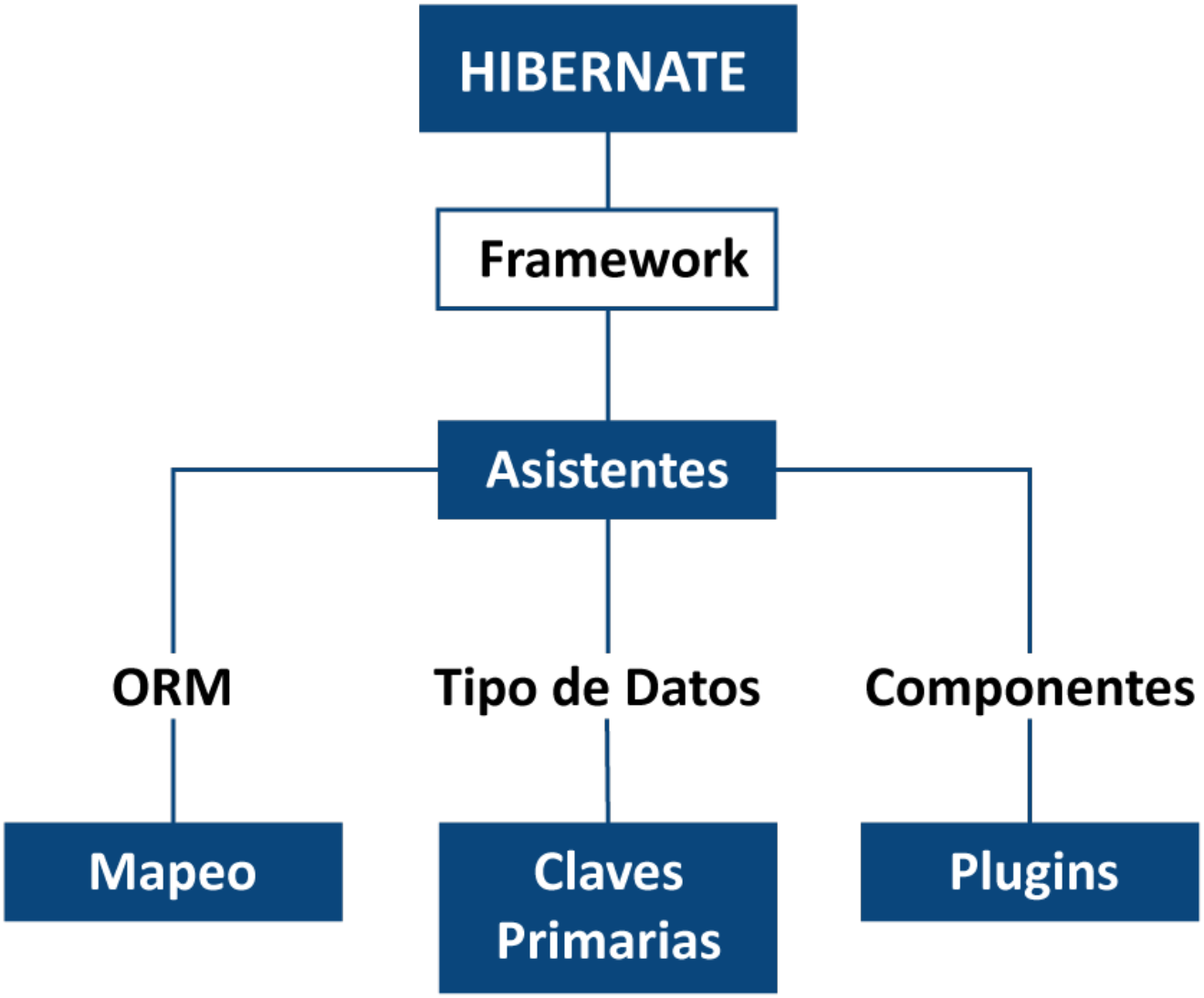
Nombre del taller de aprendizaje: Enviar/Recibir	**Datos del autor del taller:** Cesar augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta: Se puede enviar y recibir información y procesar los datos de forma segura.	
Solución del taller: Si, si se aplican los estándares y las normas apropiadas se podrán crear aplicaciones que enviar y reciban datos de manera optima	

2.4.2 TALLER DE ENTRENAMIENTO

Nombre del taller: Control de Notas	**Modalidad de trabajo:** Individual
Actividad previa: Repase los métodos y procesos de envío y recepción de información	
Describa la actividad: Realice un ingreso de notas de un alumno y el cliente debe de tener la opción de consultar y hacer un reclamo sobre la nota obtenida.	

3 UNIDAD 2 INTEGRACION CON HIBERNATE

3.1.1 RELACIÓN DE CONCEPTOS



Escriba la definición de todos los conceptos plateados en el mapa conceptual

ORM: Es un mapeo de objetos relacionales

Claves Primarias: Elemento principal de una tabla que no permite que se repita información de identificación

Tipos de datos: Elementos que permiten la clasificación de la información.

Asistente: Componente que permite realizar procesos complejos de una forma simple

Framework: Herramienta que permite que la elaboración de un aplicativo se realice de una manera más simple y controlada

OBJETIVO GENERAL

Manejar nuevas herramientas de desarrollo como el Hibernate, conociendo las bondades de este tipo complemento que posibilita la construcción más rápido de un aplicativo tradicional.

OBJETIVOS ESPECÍFICOS

- Identificar las principales características de un **framework**
- Identificar los componentes, sentencias y formas de trabajo con **Hibernate**

3.2 TEMA 1 CONCEPTOS DE ORM

ORM **(Object Relational Mapping),** es una técnica de programación para convertir datos entre el lenguaje de programación orientado a objetos y el sistema de base de datos relacional.

> **El utilizar el ORM, puede proporcionar ciertas ventajas, teniendo en cuenta que es un framework que facilitará el trabajo de los procesos más repetitivos y se podrá invertir un poco más de tiempo a otras áreas del desarrollo.**

VENTAJAS

Rapidez en el desarrollo.

- Abstracción de la base de datos
- Reutilización
- Seguridad
- Mantenimiento del código
- Lenguaje propio para realizar las consultas.

PISTAS DE APRENDIZAJE

ORM: Es una herramienta para el mapeo de objetos relacionales

3.3 TEMA 2 RELACIONES

EL MAPEO RELACIONAL

La ventaja de estos sistemas es la reducción considerable de código necesario para lograr lo que se conoce como persistencia de objetos, esto permite lograr una integración con otros patrones como el **Modelo-Vista-Controlador**.

En general los sistemas de información guardan datos en BD relacionales como **Oracle, mysql, sqlServer**, etc, dentro de los procesos más comunes tenemos que un departamento de una empresa tiene varios empleados, pero un empleado pertenece solo a un departamento.

Hibernate resuelve algunos inconvenientes con la representación de un modelo relacional mediante un conjunto de objetos, en este caso los modelos representan tablas y los atributos de las clases son los campos de las tablas.

Para mapear un modelo relacional se pueden utilizar formatos XML o con anotaciones.

PISTAS DE APRENDIZAJE

Mapeo :Opción de verificar y contralar el aplicativo con la BD que se está vinculando

3.4 TEMA 3 CLAVES PRIMARIAS Y TIPOS DE DATOS

Dentro de las características del hibernate están sus tipos de datos

- integer
- long

- short
- float
- double
- character
- byte
- boolean
- yes_no
- true_false
- string
- date
- time
- timestamp
- text
- binary
- big_decimal
- big_integer

Muchos de ellos muy conocidos por el trabajo de java otros no tanto y más comunes en este tipo de **framework**.

Estos datos tienen una clasificación como

- Fecha y hora
- Date, time y timestamp
- Boolean
- Yes_no, true_false, Boolean
- Texto
- String y text

GENERACIÓN DE CLAVES PRIMARIAS

Hibernate tiene múltiples formas de tratar las claves primarias, la más simple es cuando el desarrollador indica la clave que tendrá el objeto, este proceso se conoce como "**assigned**".

Hibernate Query Language

El **HQL** es el lenguaje de consultas del Hibérnate, este tipo de sentencias tienen algunas características que facilitan el uso de la herramienta, aunque hay que tener presente casos como la sensibilidad de las mayúsculas y minúsculas que en las sentencias como tal no influyen, teniendo presente que puede ser Select, seLect, selecT y no presentaría ningún inconveniente en el trabajo.

Es muy común ver en Hibernate la instrucción **from** sin procesos previos como se está acostumbrado a otras herramientas lo mismo que las uniones con la instrucción join.

Un ejemplo de este tipo de sentencia es

Query = “from empleado order by nombre”

Dentro de las sentencias **Join** se encuentran

inner join

left outer join

right outer join

PISTAS DE APRENDIZAJE

Clave primaria: Forma de identificar un campo que no repite información

Tipos de datos: Conjunto de opciones que permiten clasificar la información.

HQL: Herramienta que utiliza Hibernate para las sentencias sql

3.5 TEMA 4 OBJETOS Y VALIDACIONES

Las validaciones en cualquier tipo de lenguaje se convierten en elementos fundamentales para un trabajo organizado, en hibernate es común encontrar que las validaciones están asociadas a anotaciones

@NotNull

Esta propiedad indica que no puede estar nulo

@Size (min=n, max=m):

Esta propiedad controla que la información no sea nula y que contenga un mínimo de caracteres y un máximo.

Otras validaciones son

- @AssertFalse
- @AssertTrue
- @Digits(integer=n, fraction=m)
- @Future

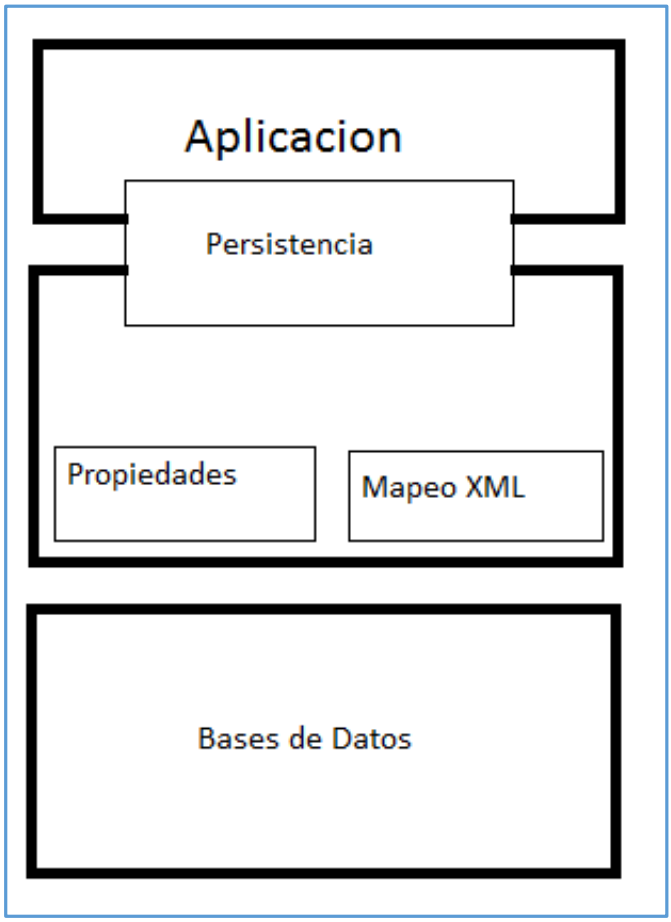
- @Past
- @Max(n)
- @Min(n)
- @NotNull
- @Null
- @Pattern(regexp="r")
- @Size(min=n, max=m).
- @Email
- @NotBlank
- @Valid

PISTAS DE APRENDIZAJE

Validación: Verificar que los datos ingresados cumplan unas condiciones mínimas de funcionamiento.

3.6 TEMA 5 ARQUITECTURA

La arquitectura en términos generales del Hibenate es la siguiente

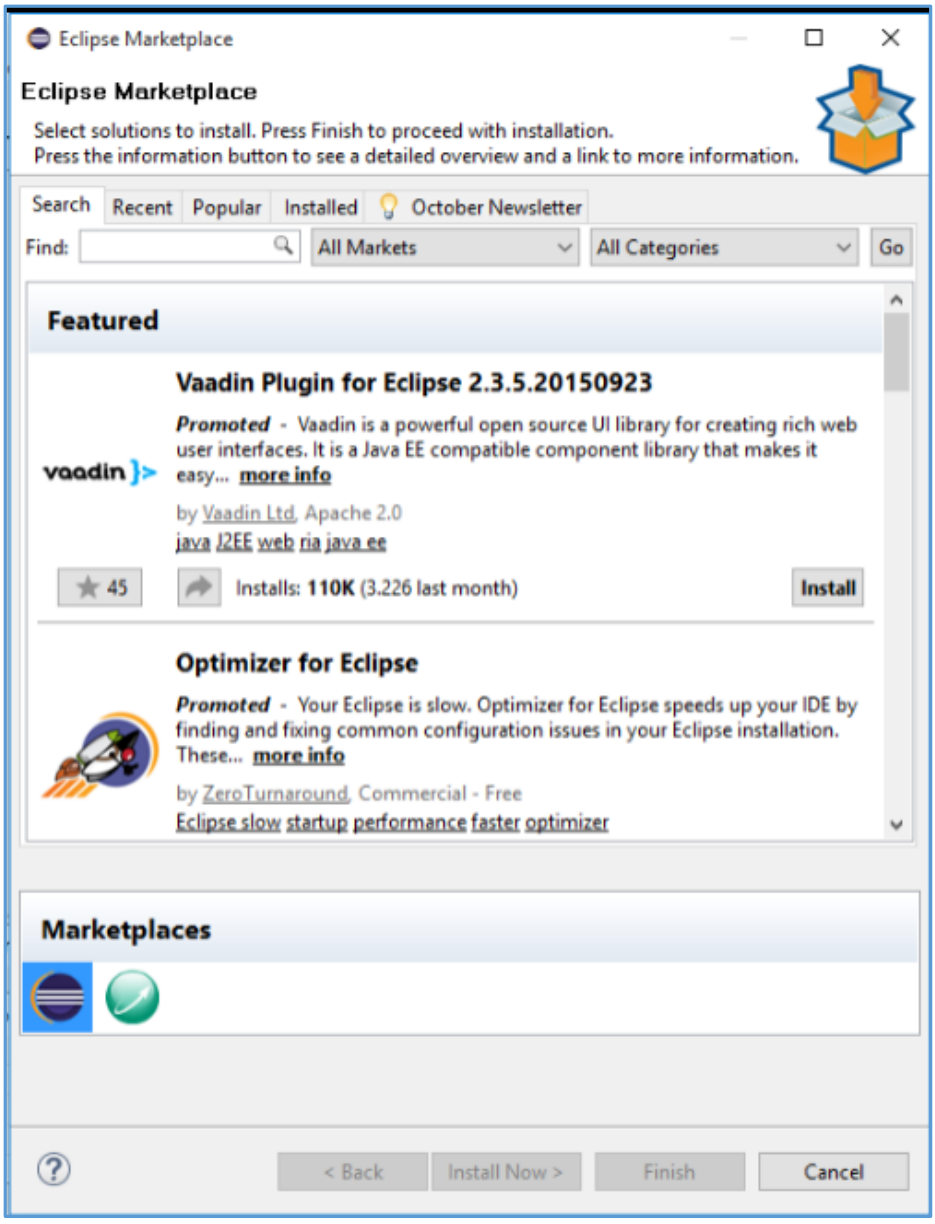


Autoría Propia

Luego de tener una BD organizada procedemos con la configuración inicial

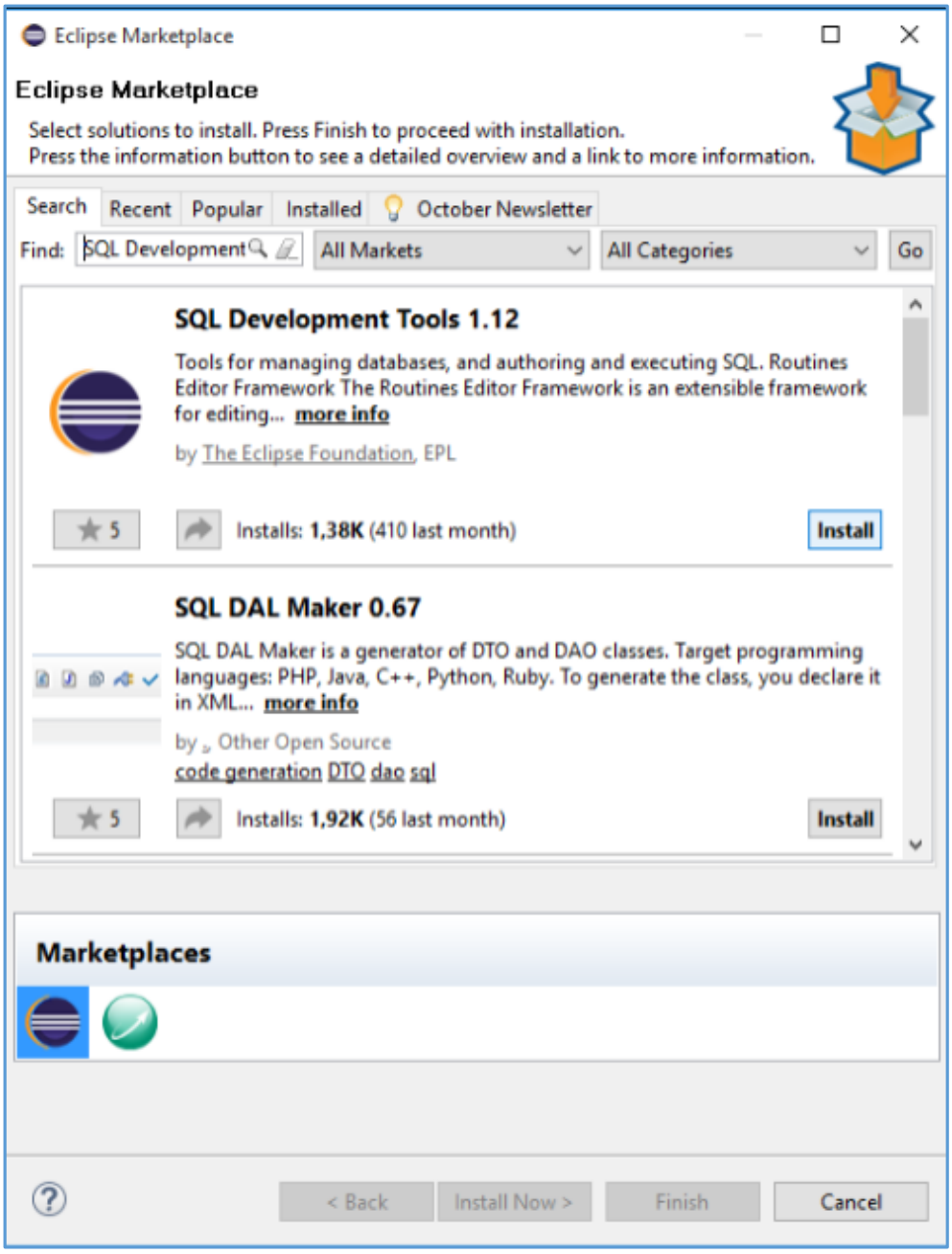
Después de haber ingresado al eclipse y haber creado un proyecto de la forma tradicional se realiza la siguiente configuración, el proyecto tendrá como nombre biblioteca.

En el menú **Help / Eclipse MarketPlace …**



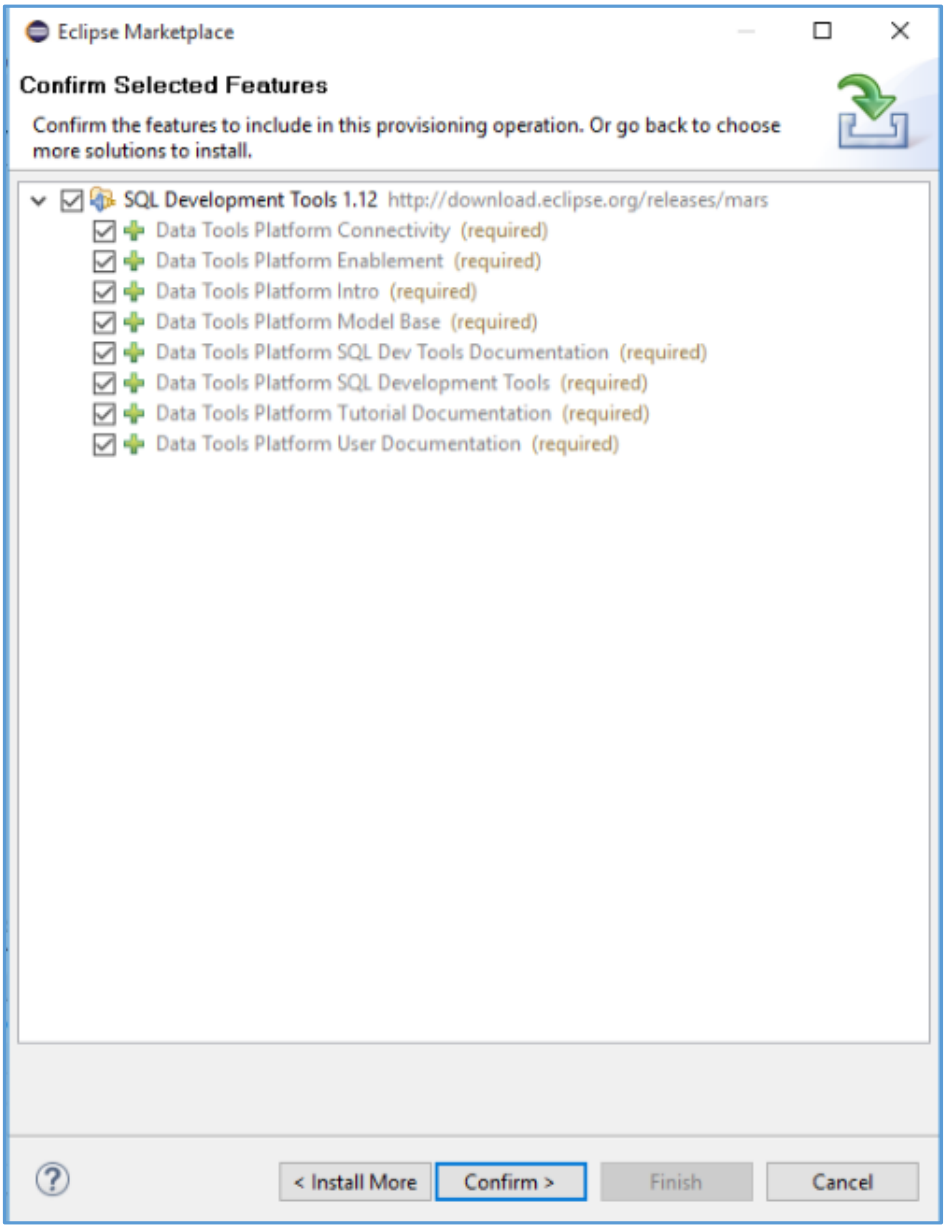
Autoria Propia

Se procede a buscar la instruccion SQL Development tools



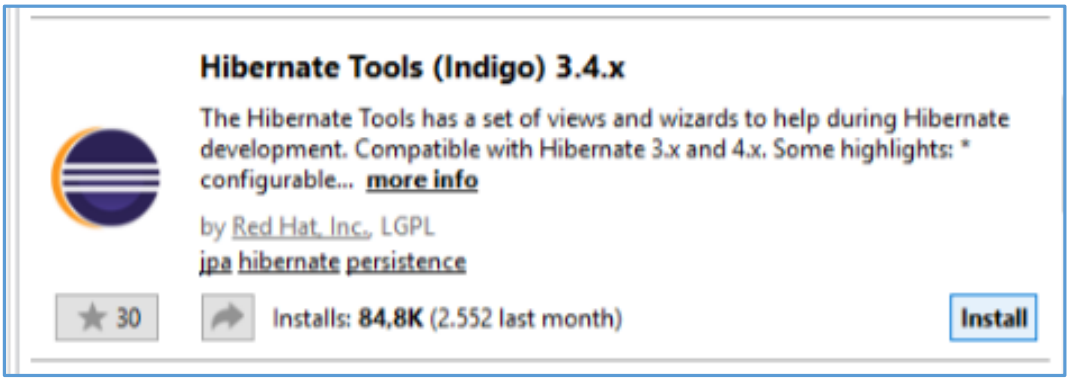
Autoria Propia

Apereceran una serie de opciones, las cuales confirmaremos



Autoria Propia

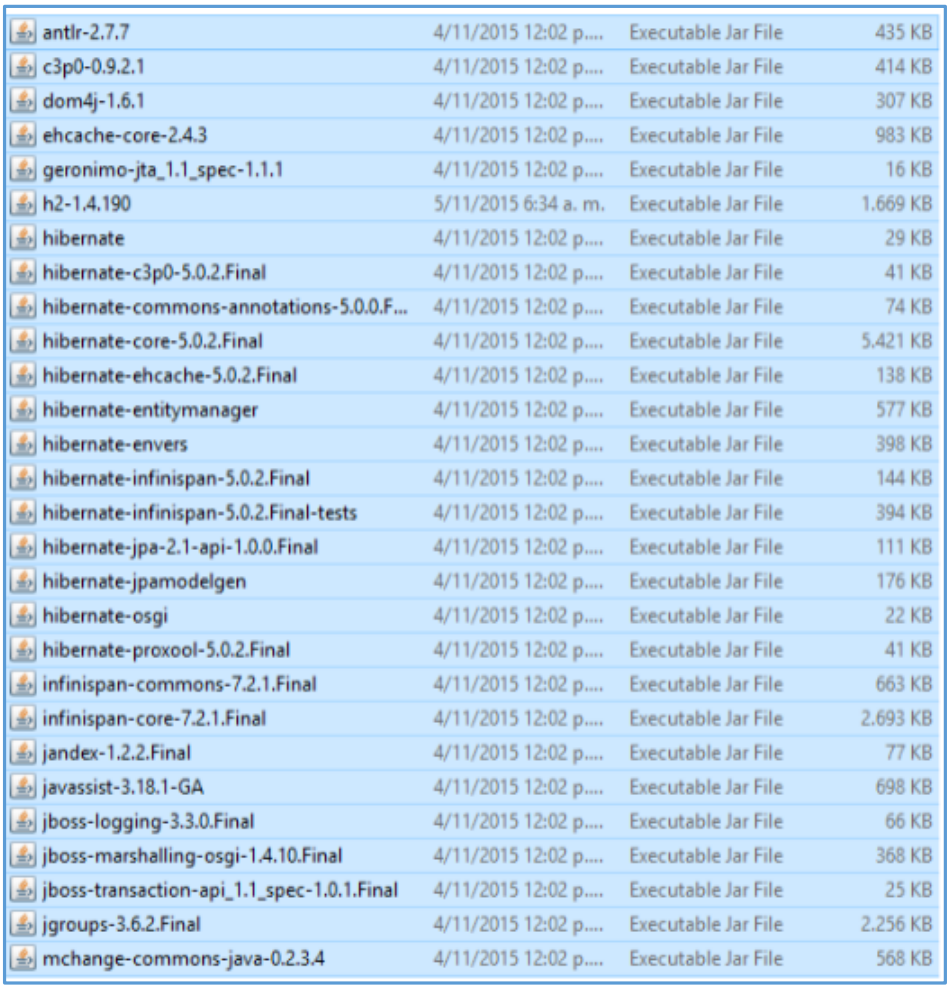
Luego se procede con el proceso de agregar Hibernate y buscamos la informacion de la misma manera en el Help / eclipse marketPlace…



Autoria Propia

Se aceptan los terminos y se finaliza.

Se buscan las librerías

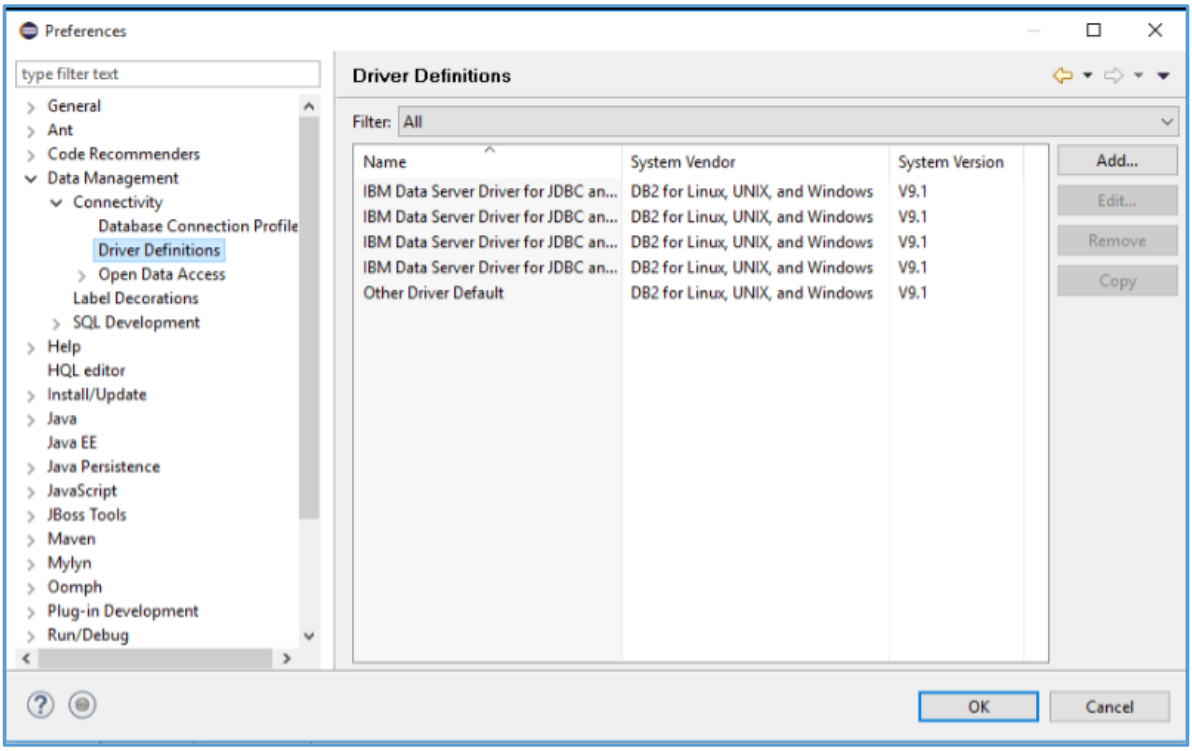


antlr-2.7.7	4/11/2015 12:02 p....	Executable Jar File	435 KB
c3p0-0.9.2.1	4/11/2015 12:02 p....	Executable Jar File	414 KB
dom4j-1.6.1	4/11/2015 12:02 p....	Executable Jar File	307 KB
ehcache-core-2.4.3	4/11/2015 12:02 p....	Executable Jar File	983 KB
geronimo-jta_1.1_spec-1.1.1	4/11/2015 12:02 p....	Executable Jar File	16 KB
h2-1.4.190	5/11/2015 6:34 a. m.	Executable Jar File	1.669 KB
hibernate	4/11/2015 12:02 p....	Executable Jar File	29 KB
hibernate-c3p0-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	41 KB
hibernate-commons-annotations-5.0.0.F...	4/11/2015 12:02 p....	Executable Jar File	74 KB
hibernate-core-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	5.421 KB
hibernate-ehcache-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	138 KB
hibernate-entitymanager	4/11/2015 12:02 p....	Executable Jar File	577 KB
hibernate-envers	4/11/2015 12:02 p....	Executable Jar File	398 KB
hibernate-infinispan-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	144 KB
hibernate-infinispan-5.0.2.Final-tests	4/11/2015 12:02 p....	Executable Jar File	394 KB
hibernate-jpa-2.1-api-1.0.0.Final	4/11/2015 12:02 p....	Executable Jar File	111 KB
hibernate-jpamodelgen	4/11/2015 12:02 p....	Executable Jar File	176 KB
hibernate-osgi	4/11/2015 12:02 p....	Executable Jar File	22 KB
hibernate-proxool-5.0.2.Final	4/11/2015 12:02 p....	Executable Jar File	41 KB
infinispan-commons-7.2.1.Final	4/11/2015 12:02 p....	Executable Jar File	663 KB
infinispan-core-7.2.1.Final	4/11/2015 12:02 p....	Executable Jar File	2.693 KB
jandex-1.2.2.Final	4/11/2015 12:02 p....	Executable Jar File	77 KB
javassist-3.18.1-GA	4/11/2015 12:02 p....	Executable Jar File	698 KB
jboss-logging-3.3.0.Final	4/11/2015 12:02 p....	Executable Jar File	66 KB
jboss-marshalling-osgi-1.4.10.Final	4/11/2015 12:02 p....	Executable Jar File	368 KB
jboss-transaction-api_1.1_spec-1.0.1.Final	4/11/2015 12:02 p....	Executable Jar File	25 KB
jgroups-3.6.2.Final	4/11/2015 12:02 p....	Executable Jar File	2.256 KB
mchange-commons-java-0.2.3.4	4/11/2015 12:02 p....	Executable Jar File	568 KB

Autoria Propia

Se copia y se pegan de la siguiente manera, se crea en el proyecto un folder con el nombre de Lib y dentro de esta se pegan las librerias seleccionadas.

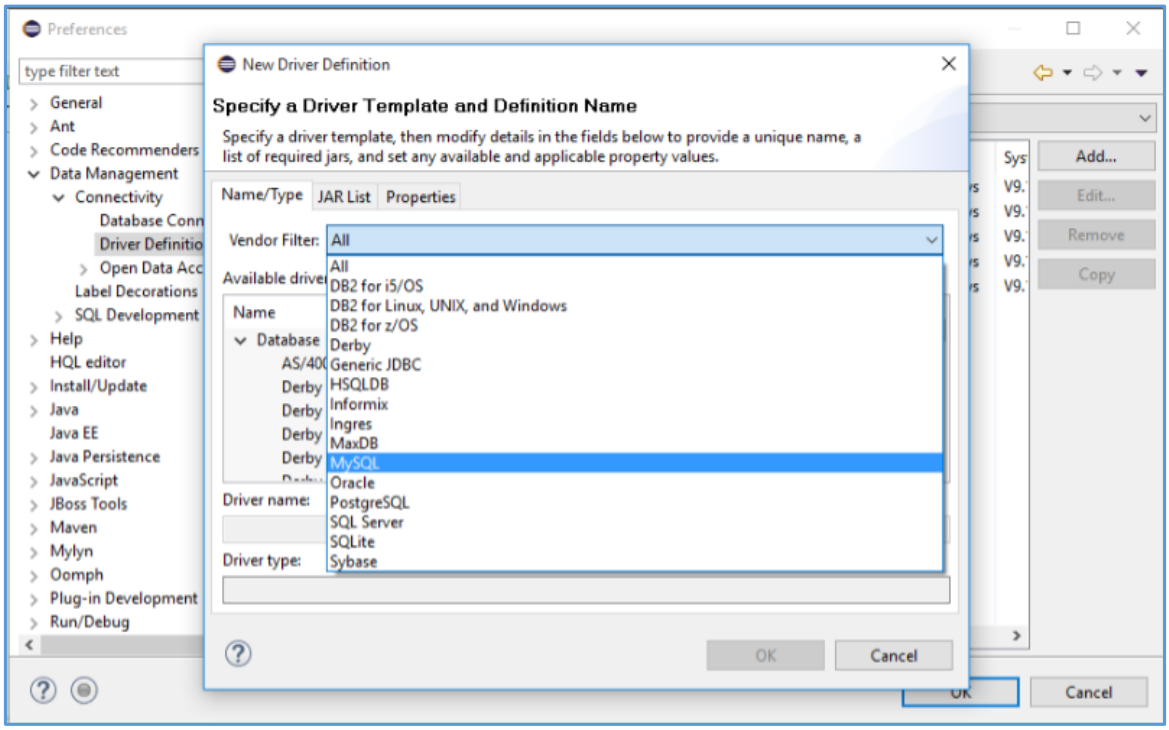
- LUEGO EN EL MENU WINDOWS / PREFERENCES



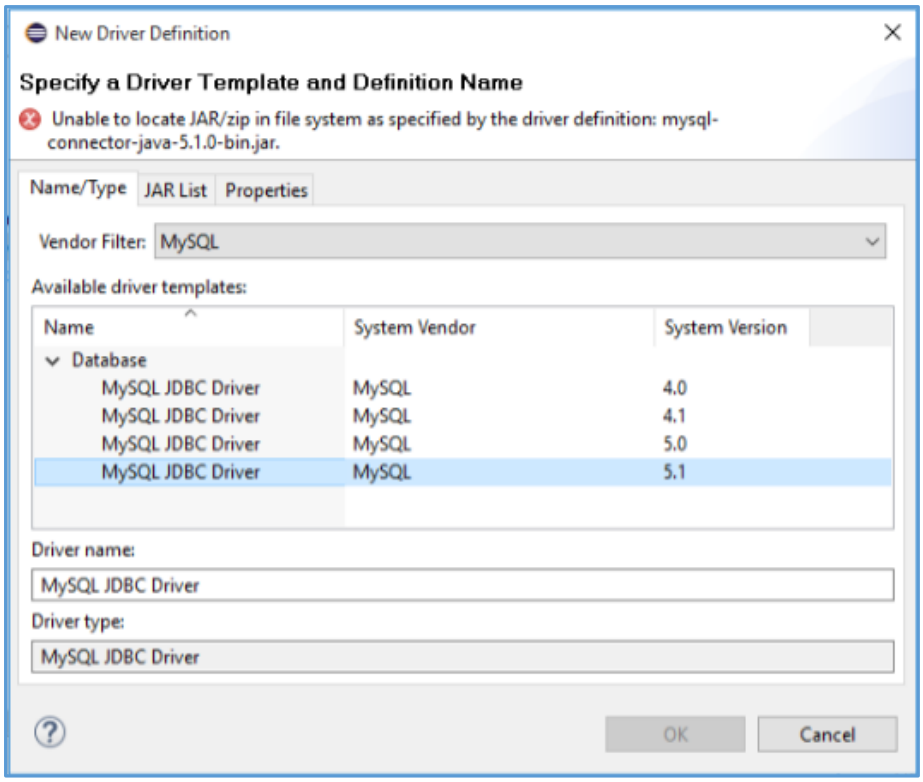
Autoria Propia

Esta es la configuracion de la conexion para el sistema.

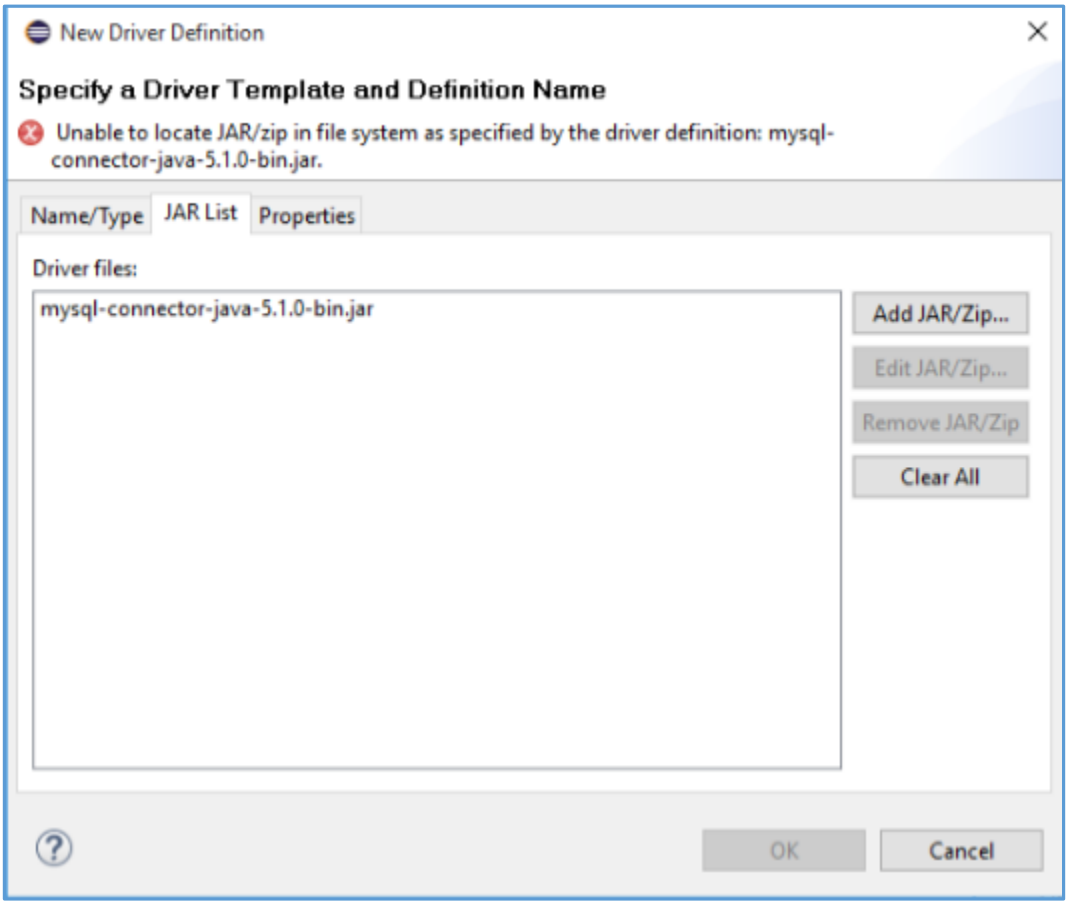
Se selecciona MySQL como herramienta de trabajo



Autoria Propia



Autoria Propia

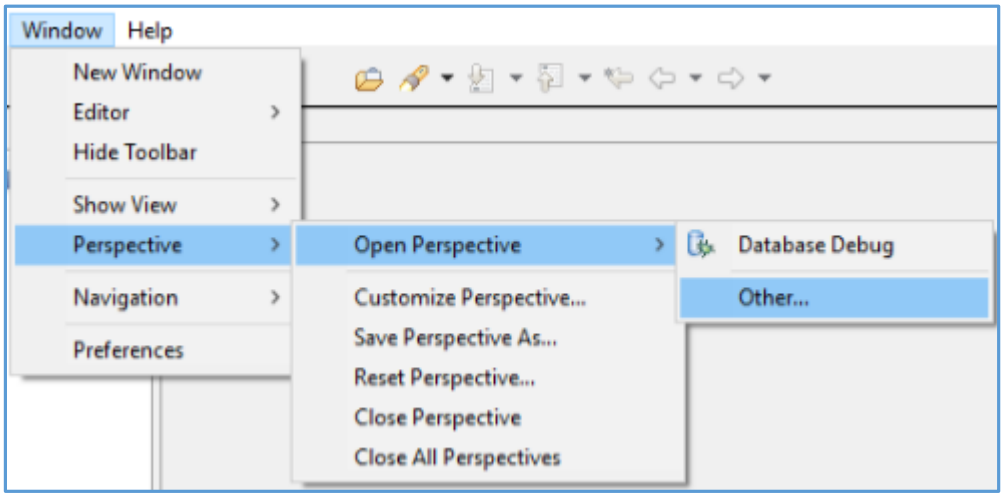


Autoria Propia

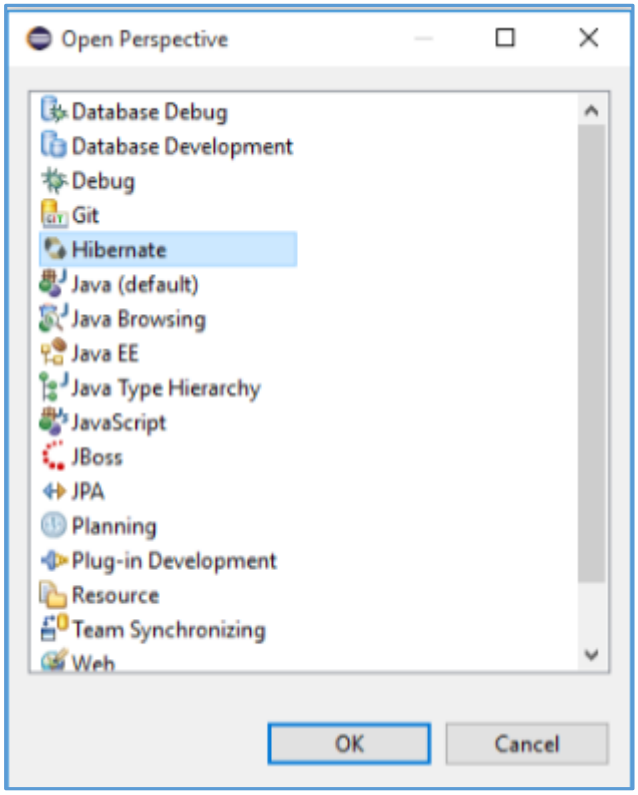
Aca se selecciona el conector que tengamos disponible o lo agregamos si no esta dentro de la lista.

Posterior a esta configuracion basica se continua con las perspecticas

En el menu de windows / perspective

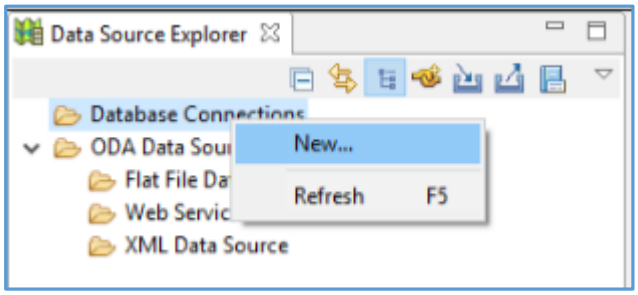


Autoria Propia



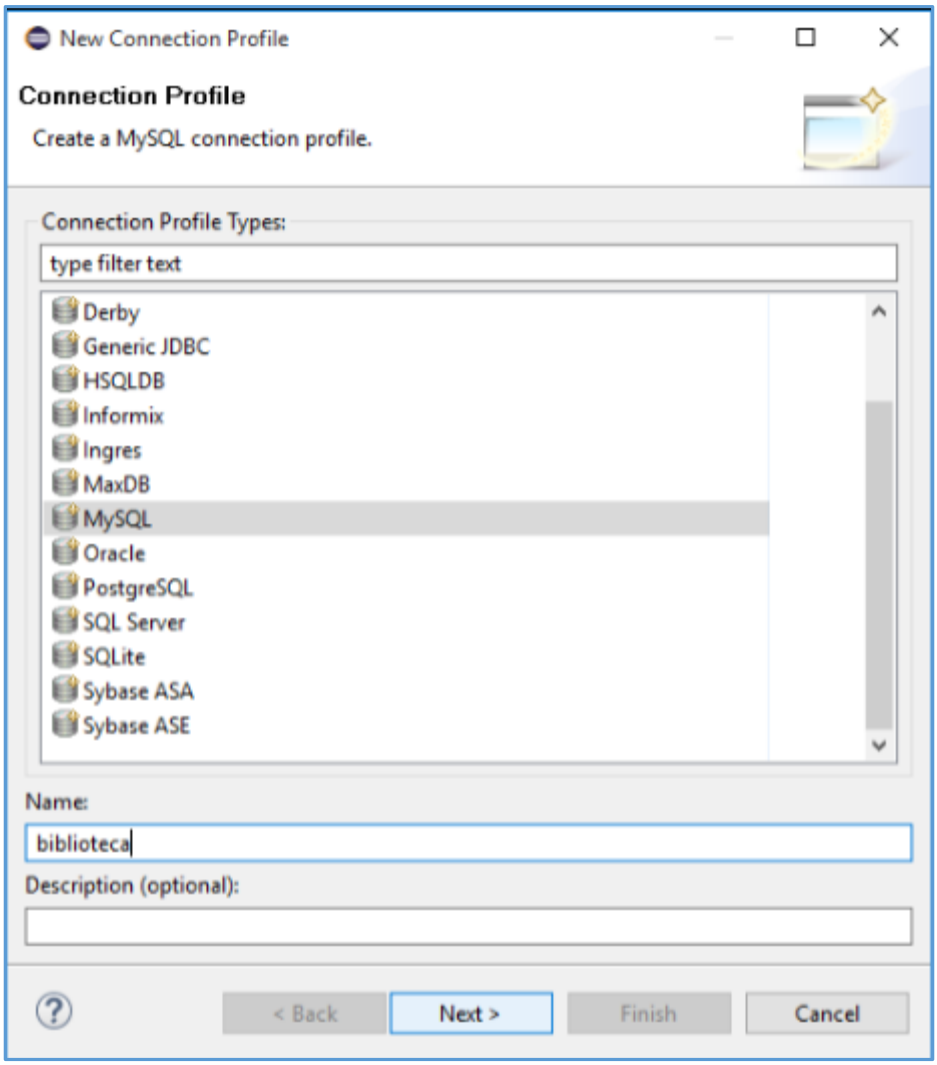
Autoria Propia

Luego de esto en la estructura del programa encontramos una serie de opciones nuevas

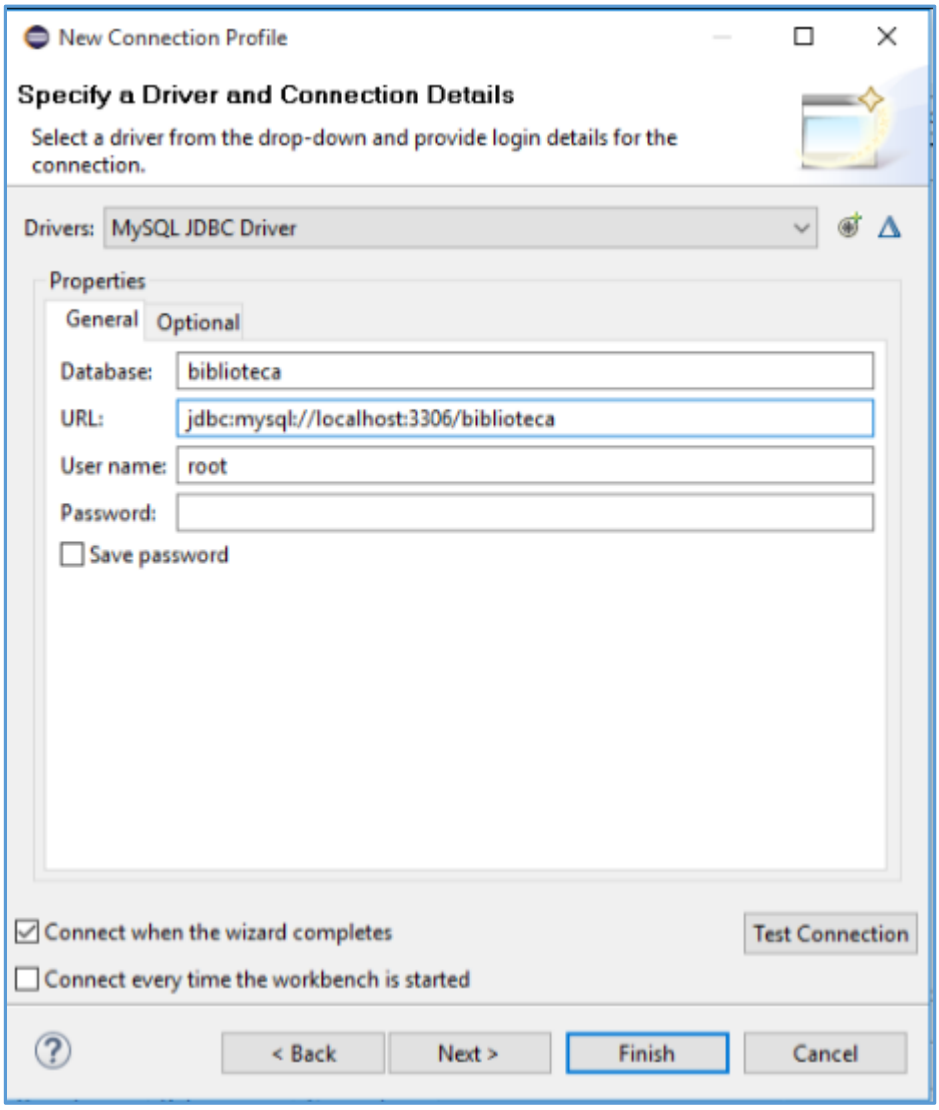


Autoria Propia

Se elecciona boton emergente en DataBase Connections, se selecciona MySQL

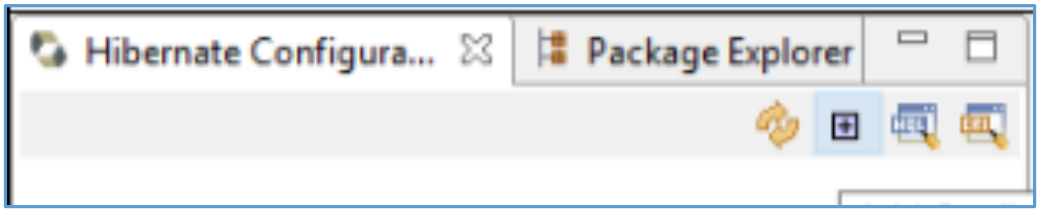


Autoria Propia

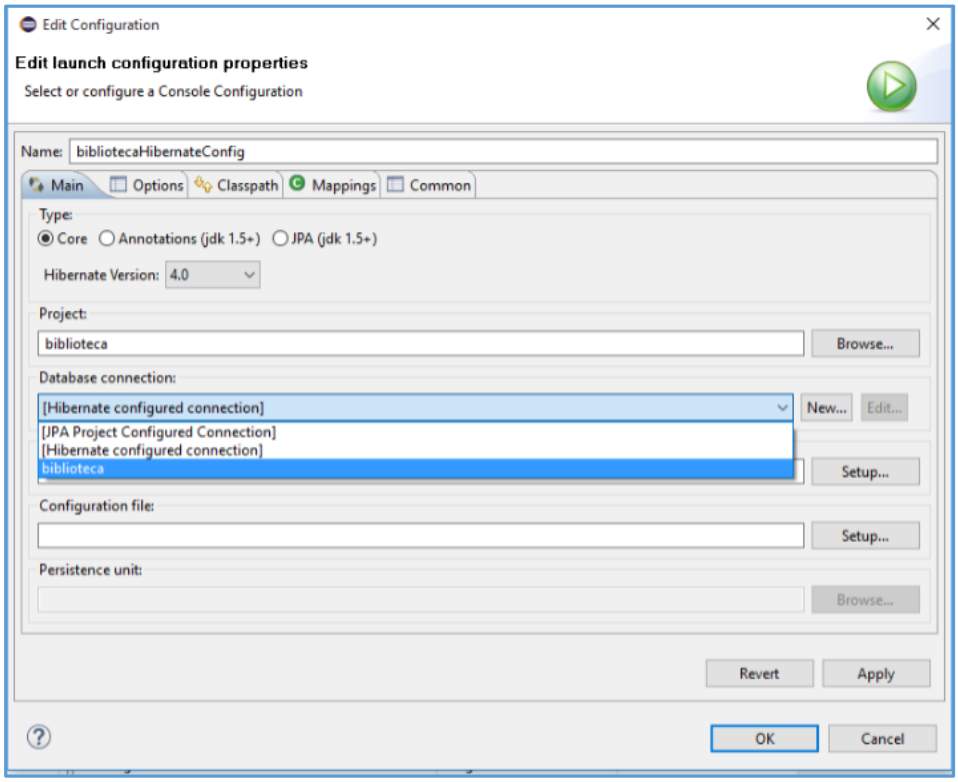


Autoria Propia

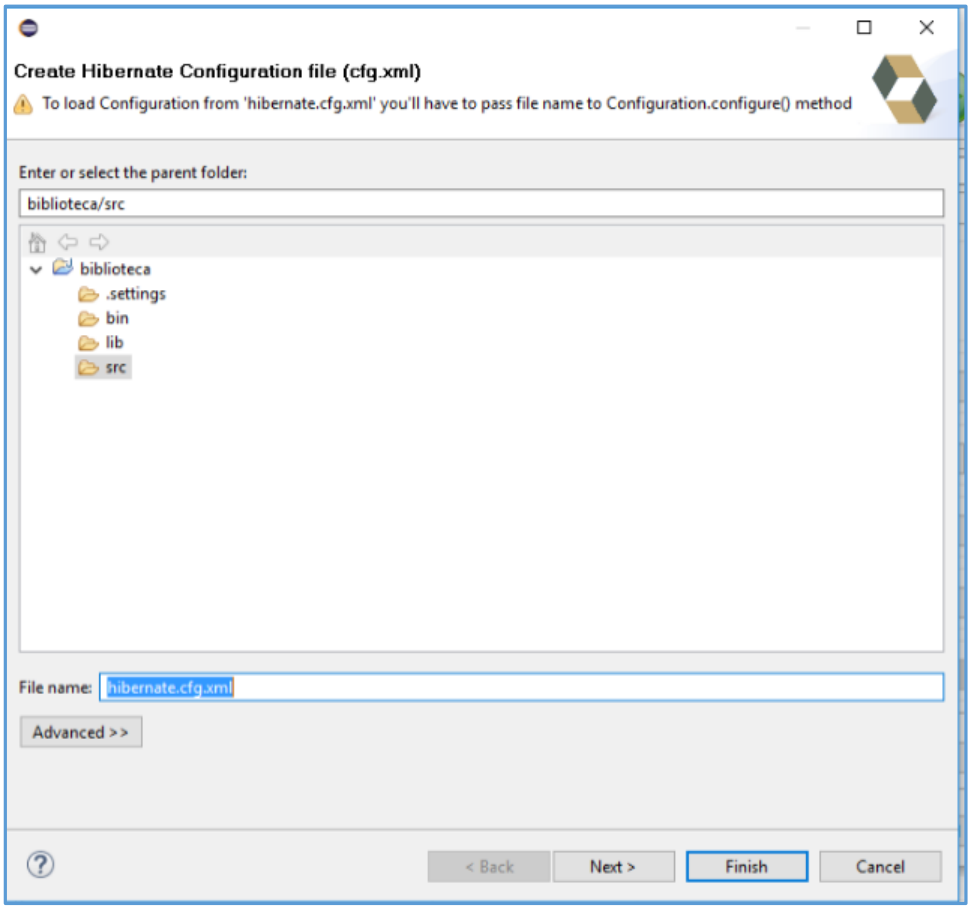
Luego en las opciones de configuracion del eclipse para android realizamos la siguiente tarea



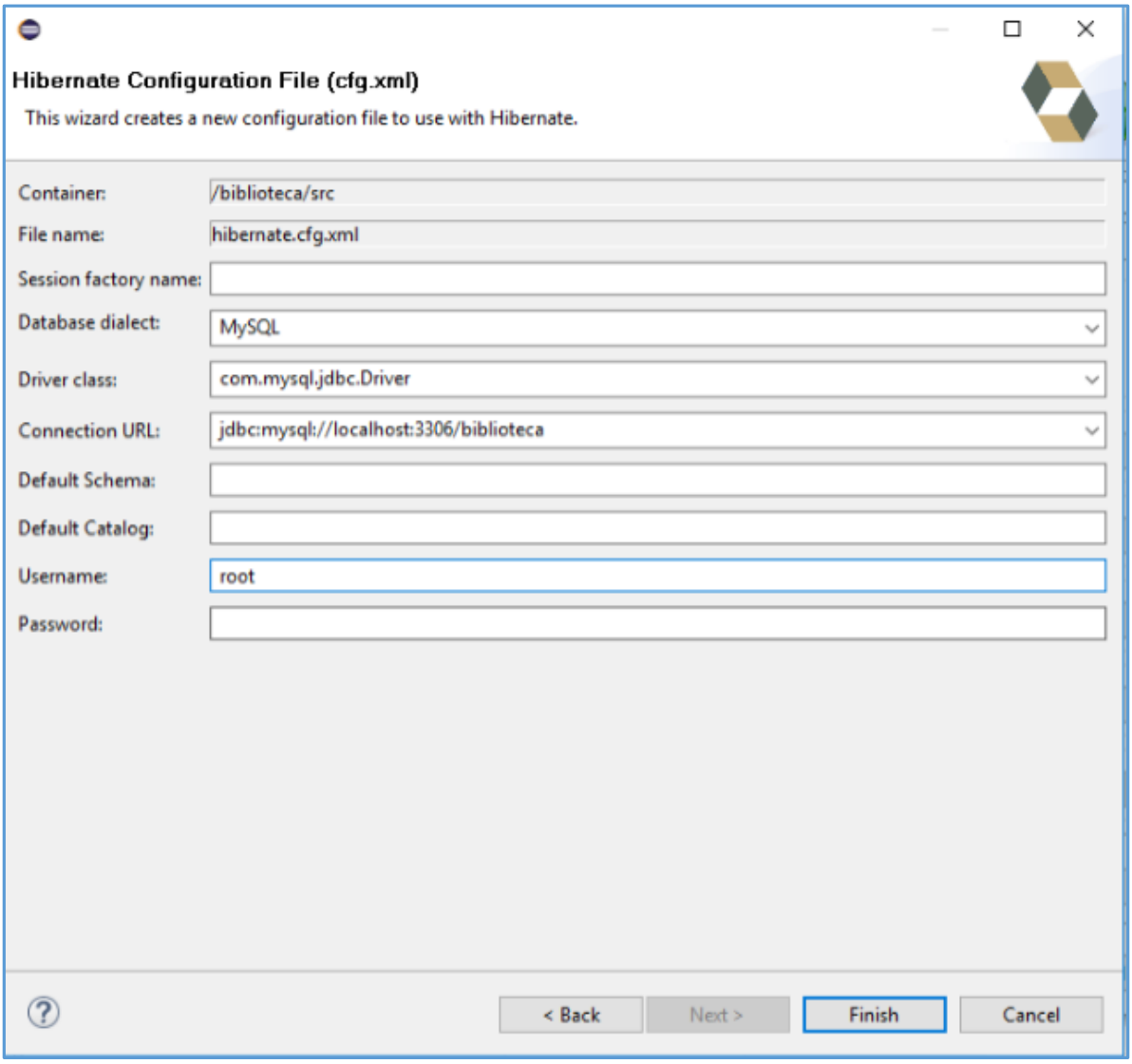
Autoria Propia



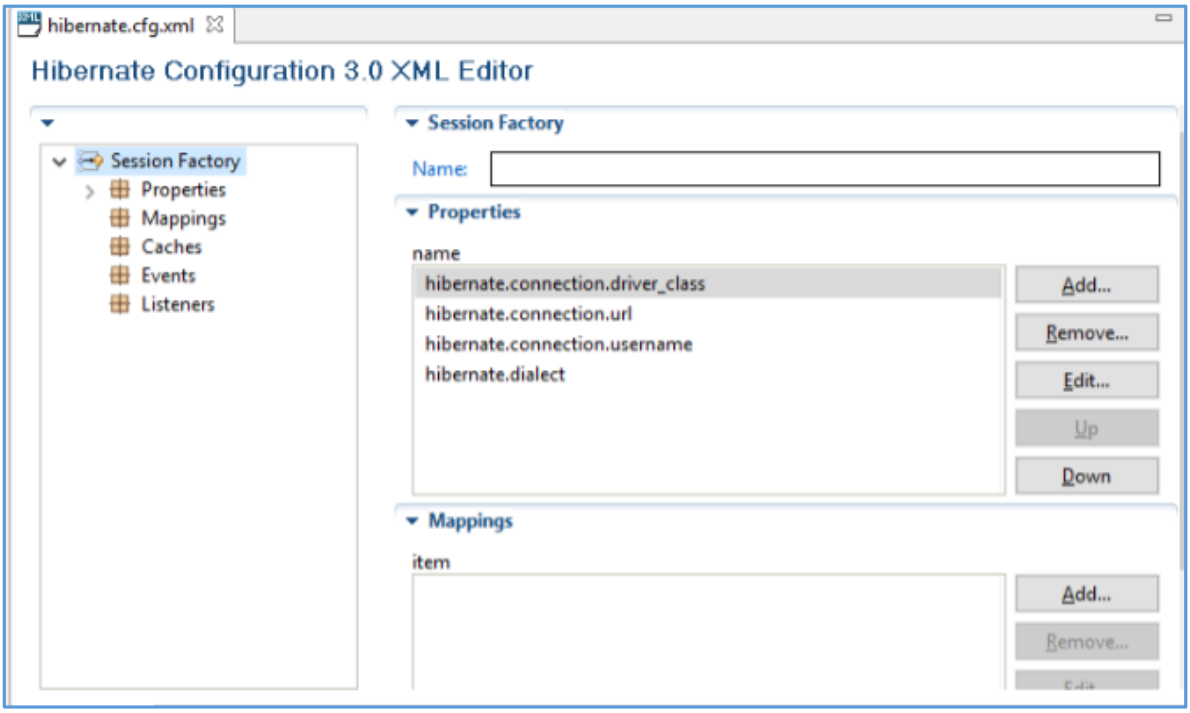
Autoria Propia



Autoria Propia



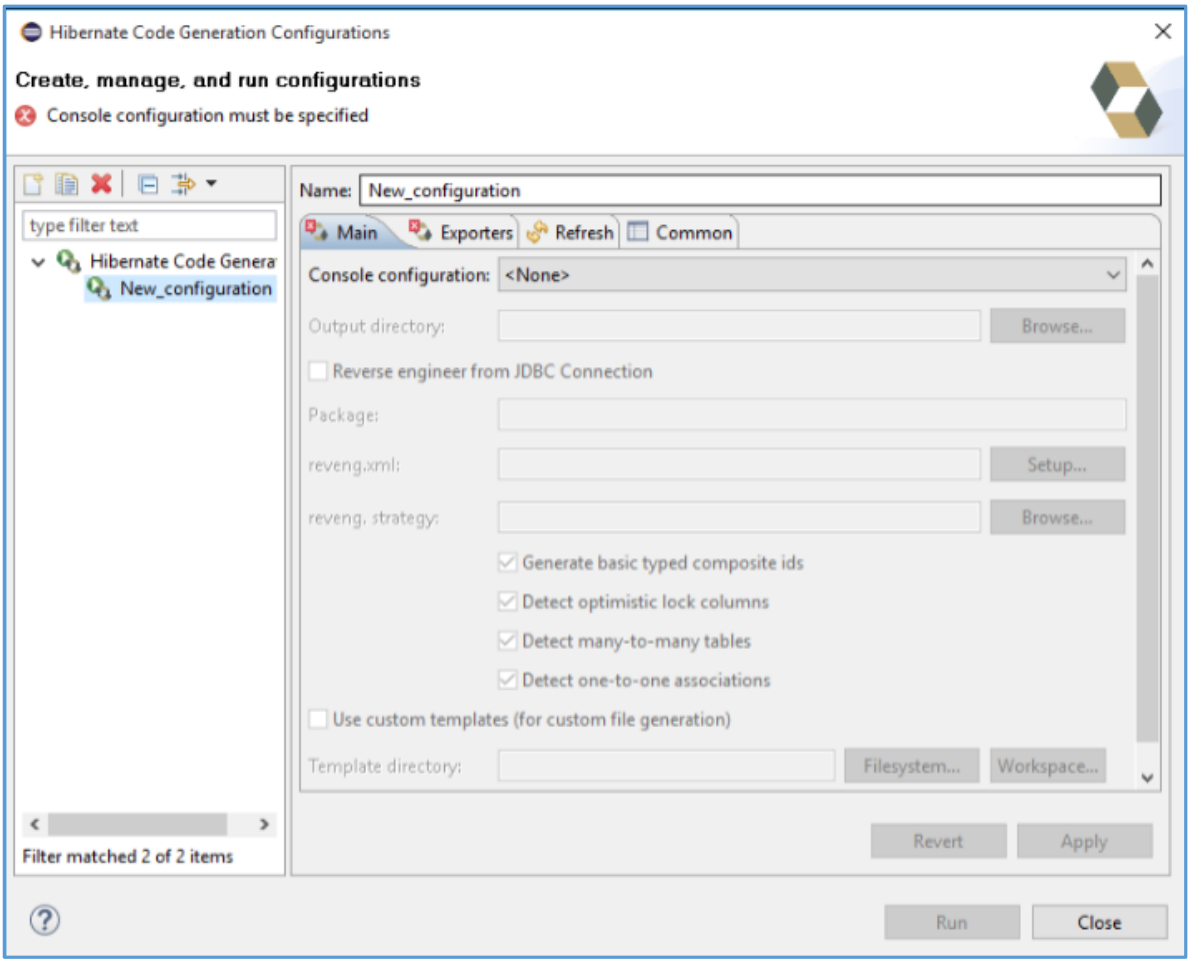
Autoria Propia



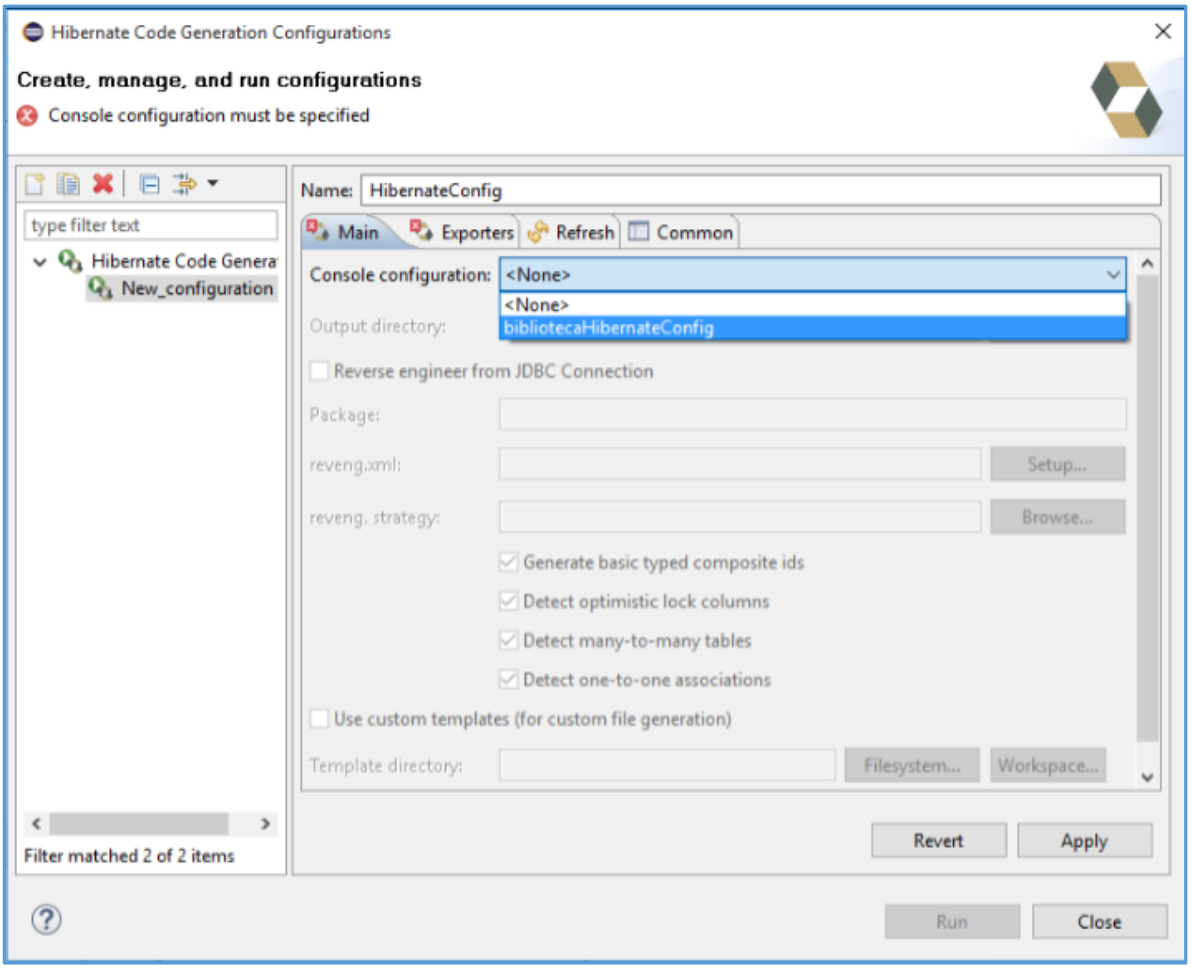
Autoria Propia

Despues de esta configuracion se procede a la activacion de la generacion de codigo por parte de Hibernate

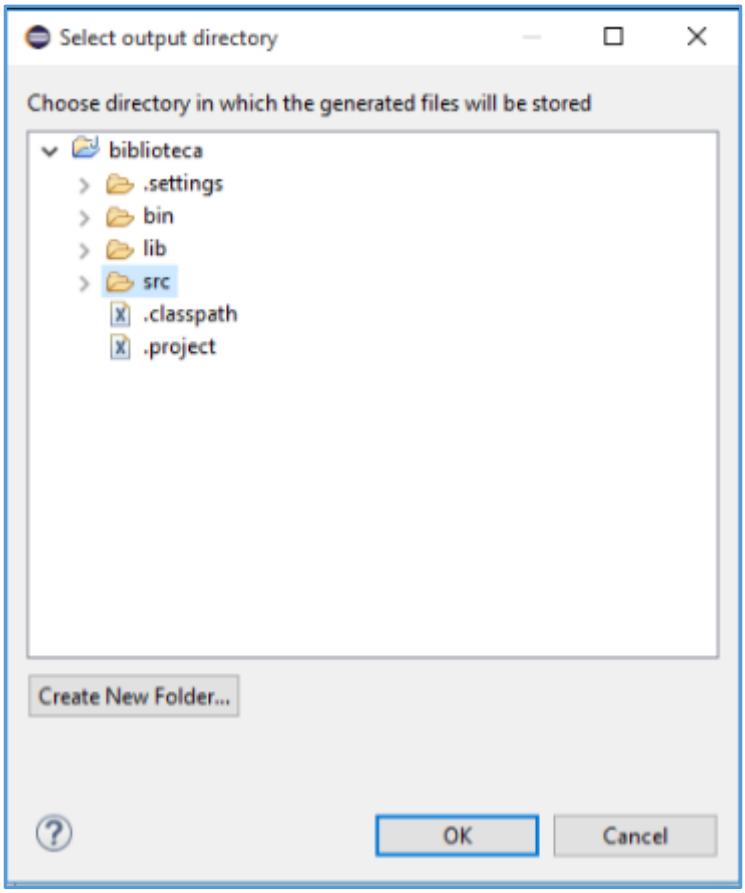
En el menu Run / Hibernate Code Generation / Hibernate Code Generation Configurations…



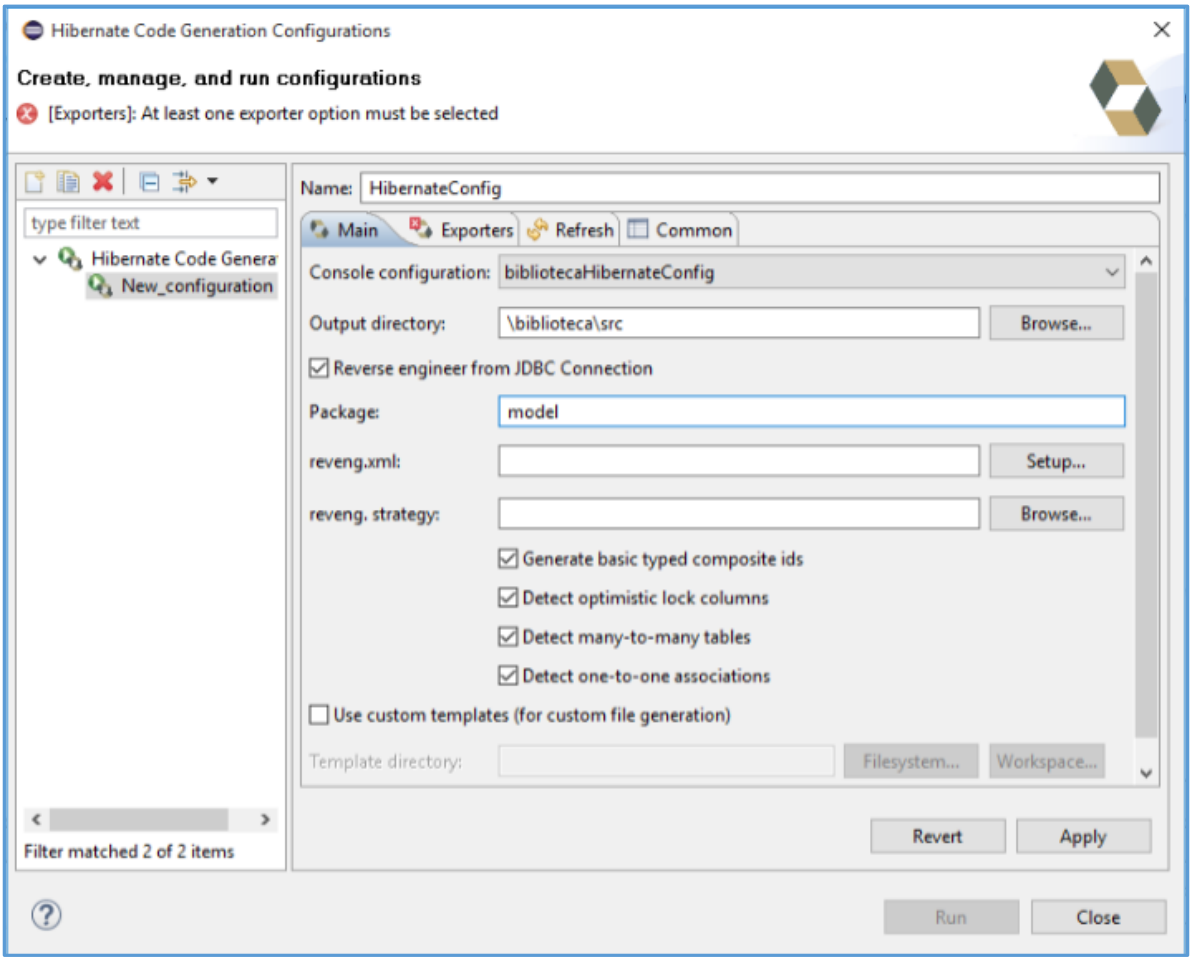
Autoria Propia



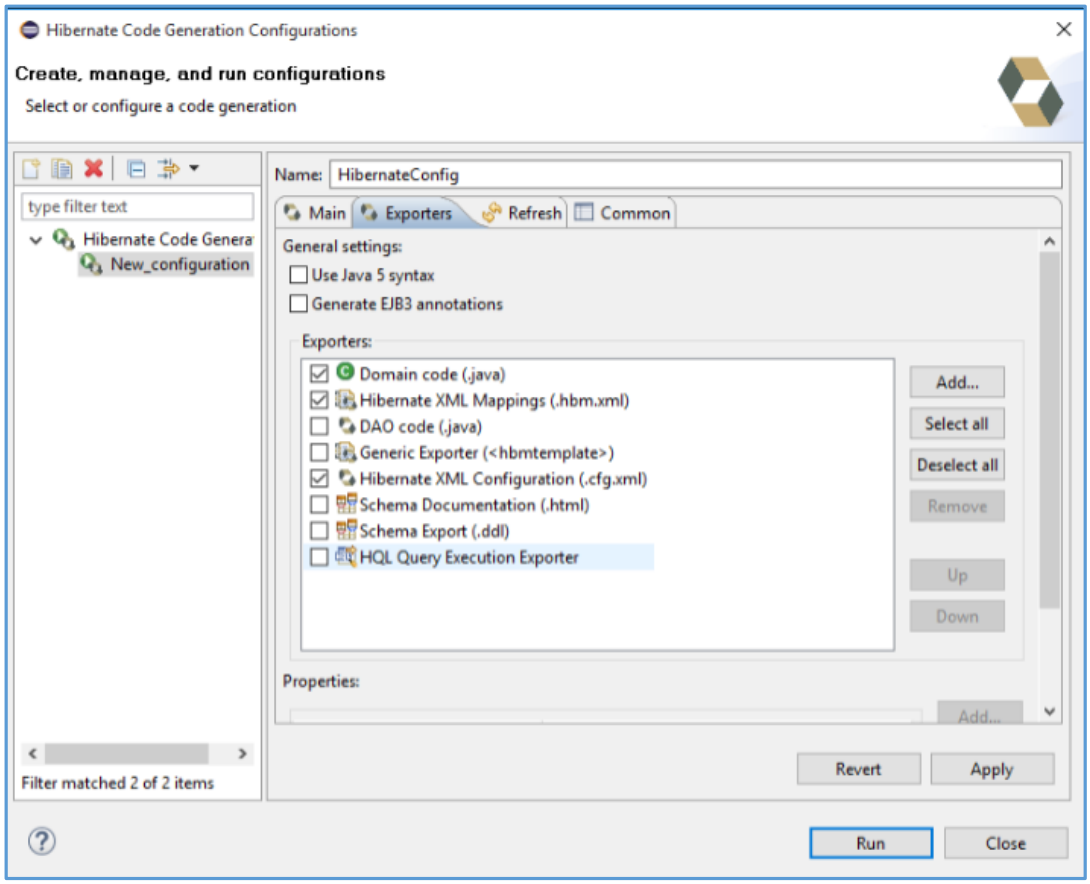
Autoria Propia



Autoria Propia



Autoria Propia



Autoria Propia

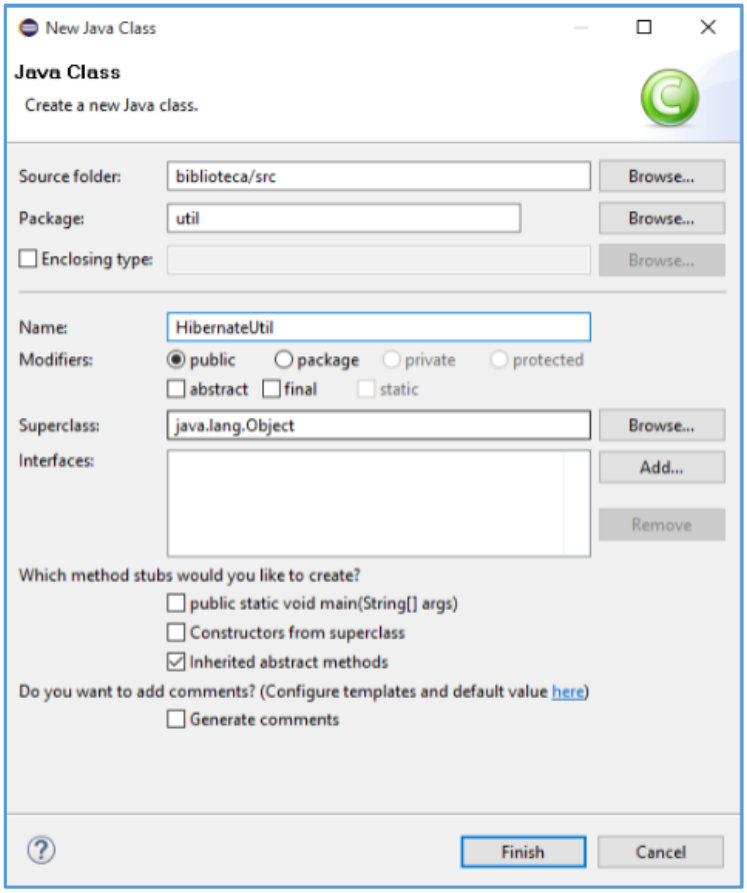
En nuestra BD de biblioteca se agregara una tabla con los campos id, nombre y descripcion

Crearemos una clase Libro.java y se gereral los getters / setters, seguido a esto se crea un archivo libro.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 6/11/2015 04:09:56 PM by Hibernate Tools 4.0.0 -->
<hibernate-mapping>
    <class name="model.Libro" table="libro" catalog="biblioteca">
        <id name="id" type="string">
            <column name="id" length="10" />
            <generator class="assigned" />
        </id>
        <property name="nombre" type="string">
            <column name="nombre" length="25" not-null="true" />
        </property>
        <property name="descripcion" type="string">
            <column name="descripcion" length="45" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

Autoria Propia

Luego de esto se crea un archivo **HibernateUtil** dentro de un paquete util



Autoria Propia

```java
package util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
            throw new ExceptionInInitializerError(e);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Autoria Propia

Complementamos con la creacion de una interfaz

New/ interface y la colocaremos facade dentro del paquete modelo y agregamos el siguiente codigo

```java
package model;

import java.util.List;

public interface Facade<T> {
    public abstract boolean add(T entity);
    public abstract T find(Object id);
    public abstract boolean update(T entity);
    public abstract boolean delete(T entity);
    public abstract List<T> getAll();
}
```

Autoria Propia

A continuacion se crea una clase LibroDAO y se crearan todos los metodos a utilizar

```java
package model;

import java.util.List;

public class LibroDAO implements Facade<Libro> {

    @Override
    public boolean add(Libro entity) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public Libro find(Object id) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public boolean update(Libro entity) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public boolean delete(Libro entity) {
        // TODO Auto-generated method stub
        return false;
    }
```

Autoria Propia

```java
@Override
public boolean add(Libro entity) {
    Session session = null;
    Transaction transaction = null;
    boolean resp;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        session.save(entity);
        transaction.commit();
        resp = true;
    } catch(HibernateException e) {
        transaction.rollback();
        resp = false;
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return resp;
}
```

Autoria Propia

```java
@Override
public Libro find(Object id) {
    Session session = null;
    Transaction transaction = null;
    Libro libro = null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        libro = (Libro)session.get(Libro.class, (Serializable) id);
        transaction.commit();
    } catch(HibernateException e) {
        transaction.rollback();
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return libro;
}
```

Autoria Propia

```java
@Override
public boolean update(Libro entity) {
    Session session = null;
    Transaction transaction = null;
    boolean resp;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        session.update(entity);
        transaction.commit();
        resp = true;
    } catch(HibernateException e) {
        transaction.rollback();
        resp = false;
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return resp;
}
```

Autoria Propia

```java
@Override
public boolean delete(Libro entity) {
    Session session = null;
    Transaction transaction = null;
    boolean resp;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        session.delete(entity);
        transaction.commit();
        resp = true;
    } catch(HibernateException e) {
        transaction.rollback();
        resp = false;
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return resp;
}
```

Autoria Propia

```java
@Override
public List<Libro> getAll() {
    Session session = null;
    Transaction transaction = null;
    List<Libro> list = null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        transaction = session.beginTransaction();
        transaction.begin();
        list = (List<Libro>) session.createQuery("From Libro").list();
        transaction.commit();
    } catch(HibernateException e) {
        transaction.rollback();
    } finally {
        try {
            if(session != null) session.close();
        } catch(HibernateException e) {
            System.out.println(e.getMessage());
            System.out.println(e.getCause());
        }
    }
    return list;
}
```

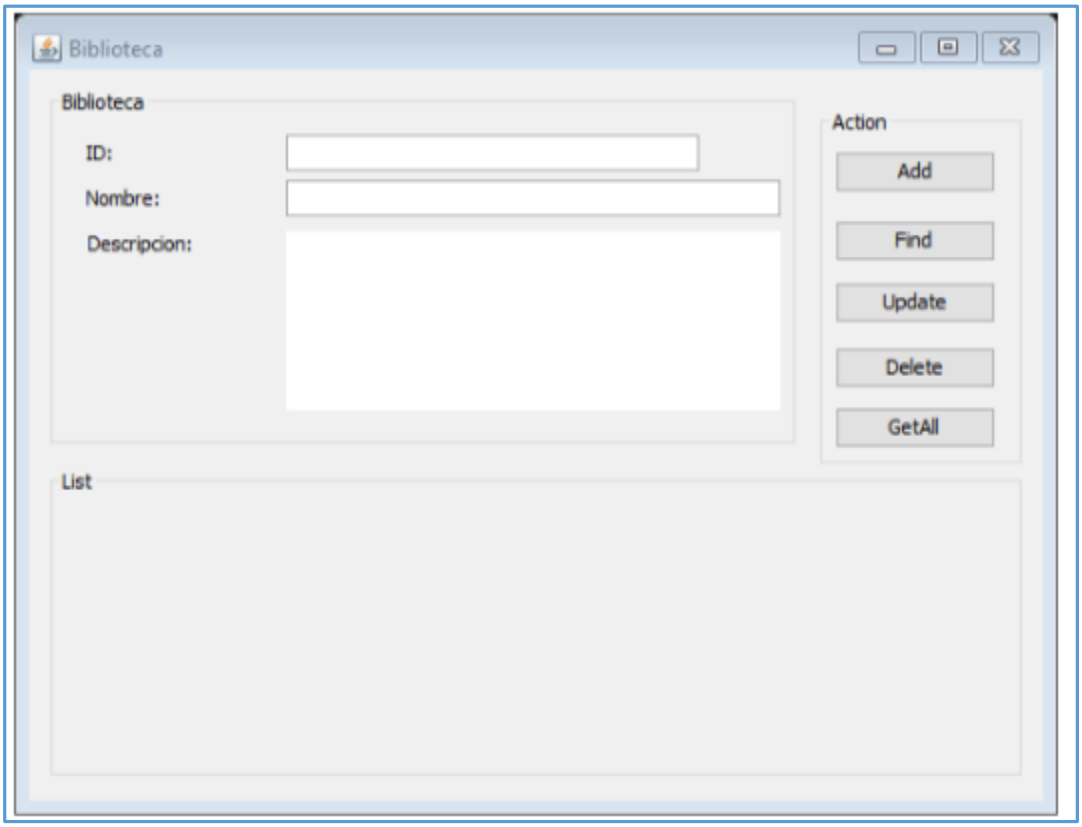
Autoria Propia

Luego se crea una clase Controller.java

```java
package controller;

import java.util.List;
import model.Libro;
import model.LibroDAO;

public class LibroController {
    private Libro libro;
    private LibroDAO libroDAO;

    public LibroController() {
        libro = new Libro();
        libroDAO = new LibroDAO();
    }
    public Libro get() {
        return libro;
    }
    public boolean add() {
        return libroDAO.add(libro);
    }
    public Libro find() {
        libro = libroDAO.find(libro.getId());
        return libro;
    }
    public boolean update() {
        return libroDAO.update(libro);
    }
    public boolean delete() {
        return libroDAO.delete(libro);
    }
    public List<Libro> getAll() {
        return libroDAO.getAll();
    }
```

Autoria Propia

Posterior a este archivo se creara el paquete vista y la clase FrmLibro



Autoria Propia

PROGRAMACION DE LOS BOTONES

```
JButton btnAdd = new JButton("Add");
btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        capturar();
        if(libroController.find() == null) {
            if(libroController.add())
                JOptionPane.showMessageDialog(null, "La informacion se ha agregado correctamente",
                        "Biblioteca", JOptionPane.INFORMATION_MESSAGE);
            else
                JOptionPane.showMessageDialog(null, "La informacion no se ha podido guardar",
                        "Biblioteca", JOptionPane.ERROR_MESSAGE);;
        } else
            JOptionPane.showMessageDialog(null, "La informacion ya existe",
                    "Biblioteca", JOptionPane.INFORMATION_MESSAGE);;
    }
});
```

Autoria Propia

```
JButton btnFind = new JButton("Find");
btnFind.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        libroController.get().setId(jtxtID.getText());
        if(libroController.find() != null) {
            jtxtID.setText(libroController.get().getId());
            jtxtNombre.setText(libroController.get().getNombre());
            jtaDescripcion.setText(libroController.get().getDescripcion());
        }
    }
});
```

Autoria Propia

```
JButton btnUpdate = new JButton("Update");
btnUpdate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        libroController.get().setId(jtxtID.getText());
        libroController.get().setNombre(jtxtNombre.getText());
        libroController.get().setDescripcion(jtaDescripcion.getText());
        if(libroController.update())
            JOptionPane.showMessageDialog(null, "La informacion se ha actualizado correctamente",
                    "Biblioteca", JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null, "La informacion no se ha podido actualizar",
                    "Biblioteca", JOptionPane.ERROR_MESSAGE);;
    }
});
```

Autoria Propia

```
JButton btnDelete = new JButton("Delete");
btnDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        libroController.get().setId(jtxtID.getText());
        libroController.get().setNombre(jtxtNombre.getText());
        libroController.get().setDescripcion(jtaDescripcion.getText());
        if(libroController.delete())
            JOptionPane.showMessageDialog(null, "La informacion se ha eliminado correctamente",
                    "Biblioteca", JOptionPane.INFORMATION_MESSAGE);
        else
            JOptionPane.showMessageDialog(null, "La informacion no se ha podido eliminar",
                    "Biblioteca", JOptionPane.ERROR_MESSAGE);
    }
});
```

Autoria Propia

Con estos procesos el ejemplo quedaria funcional.

PISTAS DE APRENDIZAJE

Configuración: Pasos para la elaboración de una tarea.

3.6.1 EJERICICIO DE APRENDIZAJE

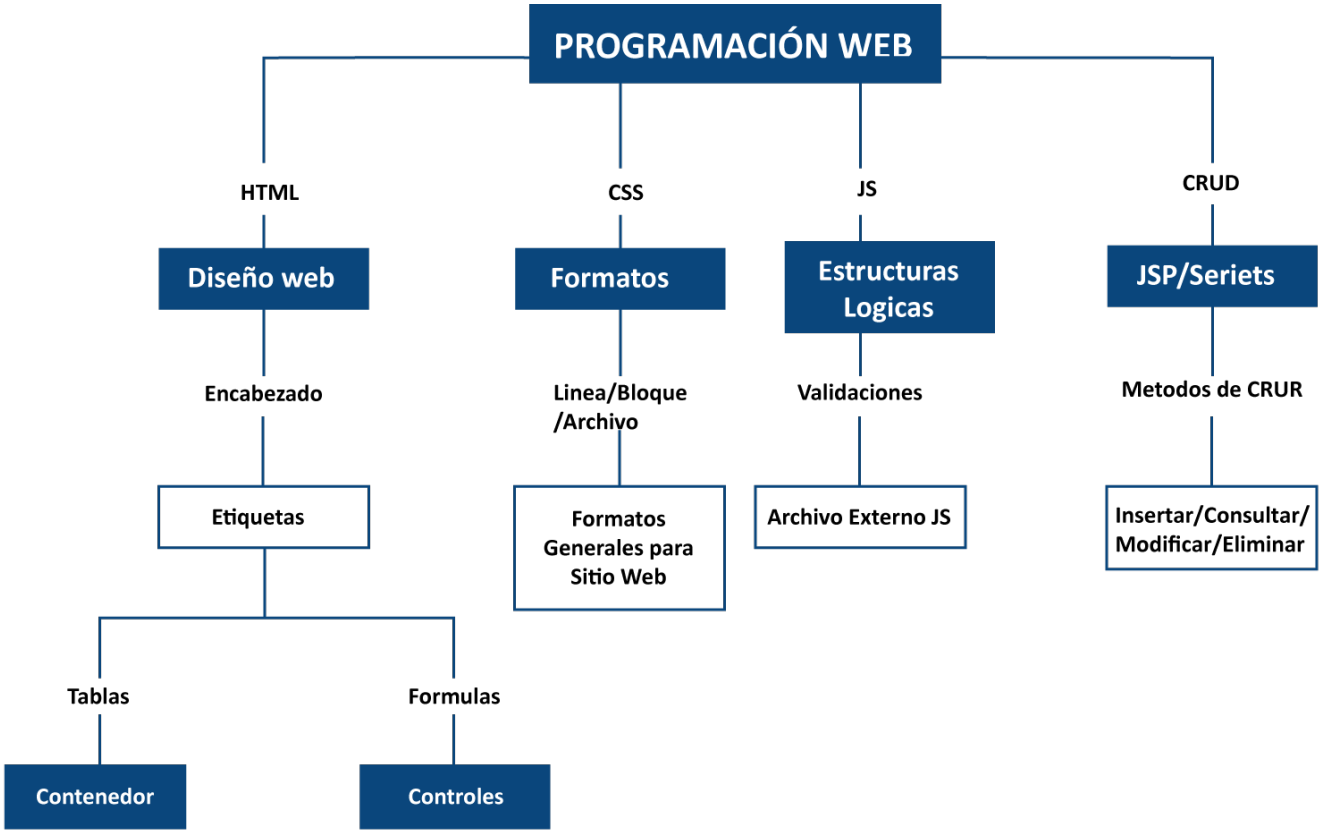
Nombre del taller de aprendizaje: Hibernate	**Datos del autor del taller:** Cesar Augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta: Considera que el hibernate es útil para cualquier tipo de aplicación.	
Solución del taller: Sí, no tiene una limitante o un solo propósito	

3.6.2 TALLER DE ENTRENAMIENTO

Nombre del taller: banco	**Modalidad de trabajo:** Individual
Actividad previa: Realice el trabajo conformado por Nomina	
Describa la actividad: Diseñar un programa en hibernate que cumpla las condiciones mínimas de un banco	

4 UNIDAD 3 INTRODUCCION A LA PROGRAMACION WEB

4.1.1 RELACIÓN DE CONCEPTOS



Contenedor: Herramienta que puede contener otros controles, ejemplo de tablas y formularios

Etiquetas: "comandos" de HTML

Controles: Componentes de un ambiente de programación, cajas de texto, botones, combos, etc.

Formatos: Sentencias que permiten dar presentación, estilos y diseño a un sitio web

Estructuras: Son componentes de un lenguaje de programación tales como ciclos, preguntas, selectores y preguntas

CRUD: Descripción de Crear, Leer, actualizar y Eliminar información.

OBJETIVO GENERAL

Aprender los conceptos básicos de la programación web, las etiquetas básicas, los formatos y las validaciones, así como la construcción de un CRUD

OBJETIVOS ESPECÍFICOS

- Identificar las principales características del HTML en su etapa de diseño para un CRUD
- Aplicar formatos que le den un aspecto menos plano del que se trabaja habitualmente en HTML estándar mediante las herramientas de CSS
- Aplicar las validaciones necesarias para controlar el ingreso de la información dentro un formulario HTML
- Elaborar un CRUD mediante JSP y Servlets

4.2 TEMA 1 HTML / HTML5

HTML

El HTML (**Hyper Text Markup Language**) es el lenguaje con el que se escriben las páginas o estructuras web, un lenguaje que permite colocar texto de forma estructurada, y que está compuesto por etiquetas, también conocidas como **tags** o **marcas**, que indican el inicio y el fin de cada elemento del documento.

> **Un documento de hipertexto no sólo se compone de texto, puede contener sonido, vídeos, imágenes y otros elementos dinámicos, por lo que el resultado puede considerarse como un documento multimedia.**

Los documentos HTML deben tener la extensión HTML o HTM, para que puedan ser visualizados en los navegadores web (Browser), sean estos los más comunes como Internet Explorer, Chrome, Mozilla, Safari, Opera, entre otros.

Los browsers se encargan de interpretar el código HTML de los documentos, y de mostrar a los usuarios las páginas web resultantes del código interpretado.

- ESTRUCTURA BASICA

```
<html>
        <head>

        </head>
        <body>
```

```
        </body>
</html>
```

La gran mayoría de las etiquetas están compuestas por una apertura y un cerrado <html> </html>, la etiqueta que contiene el símbolo slash (/) es la que indica el cerrado, otras etiquetas no se componen por pares y se cierran al final de ella,
 esta es un típico caso.

Dentro de las páginas web existe una estructura como la vista al principio, la etiqueta **<html>** y **</html>** son la primera y la ultima de la página, es la etiqueta que enmarca lo que vamos a realizar, dentro de estas etiquetas se ubicaran dos áreas, la cabecera **(head)** y el cuerpo de la página **(body).**

CABECERA

<head>…</head>

Esta etiqueta alberga el título de la página y permite la invocación de otros elementos como los scripts y las hojas de estilo en cascada, elementos que se verán más adelante.

<title> primera página web **</title>**

```
<html>
        <head>
                <title>primera pagina web </title>
        </head>
```

CUERPO DE LA PÁGINA

<body>…</body>

El cuerpo de la página alberga todo el contenido que se visualizará por parte del usuario, además el <body> podrá tener elementos como muchas otras etiquetas llamados parámetros, estos parámetros permiten darle un diseño o formato adicional

bgcolor=”color de fondo”, este se puede especificar de varias formas, el nombre del color como red, Green, yellow, etc, o se puede trabajar con un formato hexadecimal que nos da una combinación de más 16 millones de colores, este formato se representa así #RRVVAA (Rojo, Verde, Azul), los valores que se utilizan para este caso son números de 0 a 9 y de A a F, en los formatos tradiciones se componen por parejas, las dos RR representan el rojo, GG verde y BB azul, de acá saldrán los 16 millones de colores, #FF0000 nos arroja rojo, #00FF00, verde y #0000FF azul

Background=" imagen de fondo", para el manejo de fondos se podrá usar cualquier formato de imagen como JPG, PNG, GIF, tenga presente el tamaño y la resolución para hacer más agradable es espacio web.

Background=" fondo.jpg"

COMENTARIOS EN HTML

Con mucha frecuencia se requiere hacer comentarios o anular partes del código creado, para esto se utiliza una etiqueta que inhabilita esta área de trabajo

<!- - comentario //-->

SALTOS DE LÍNEA

Es el equivalente a un enter, en HTML lo enter que especifiquemos presionando la tecla o la barra espaciadora no se verá al ejecutar la página para esto existe un grupo específico de etiquetas que presentan estos caracteres

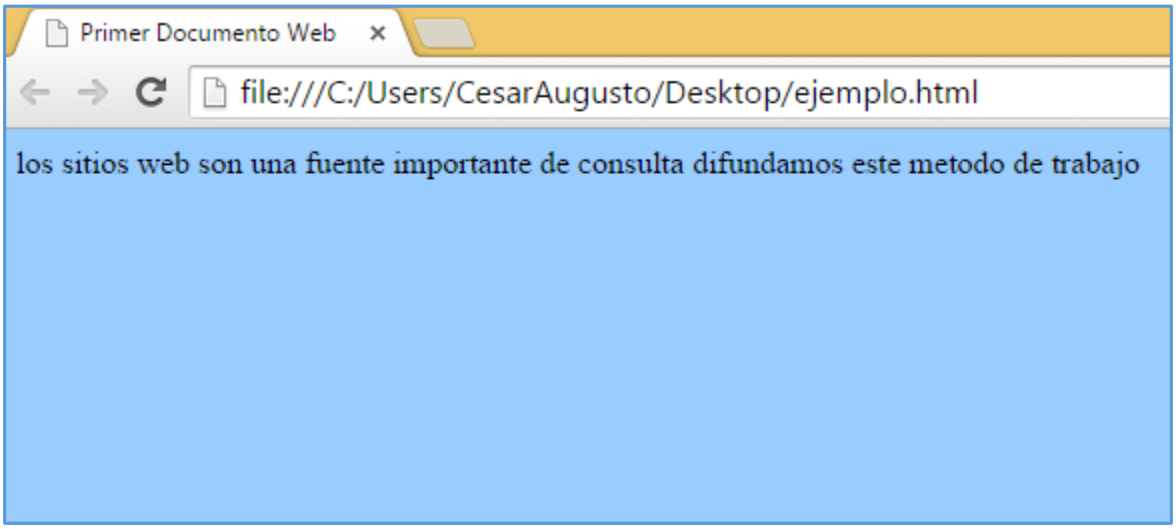
 representara este carácter

Ejemplo de representación

```
<html>
    <head>
        <title>Primer Documento Web</title>
    </head>

    <body bgcolor="#99CCFF">
        los sitios web son una fuente importante de consulta




        difundamos este metodo de trabajo
    </body>
</html>
```

Autoria Propia

Un código con las etiquetas básicas y da como resultado



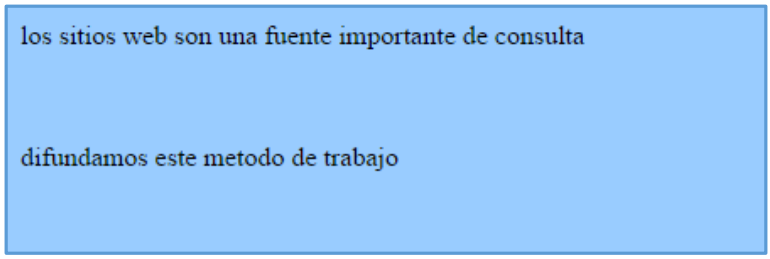
Autoria Propia

Se observa el título "**Primer Documento Web**", además en el cuerpo se ve el fondo azul que se especificó y el texto, pero se puede ver que el texto aparece en la misma línea y en el archivo el texto está separado por espacios, acá entra el funcionamiento de la etiqueta

```
<html>
    <head>
        <title>Primer Documento Web</title>
    </head>

    <body bgcolor="#99CCFF">
        los sitios web son una fuente importante de consulta
        <br />
        <br />
        <br />
        <br />
        difundamos este metodo de trabajo
    </body>
</html>
```

Autoria Propia

Resultado



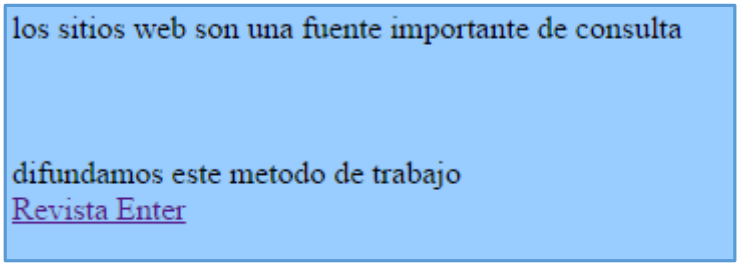
LINK

Una de las razones principales de un sitio web es el manejo de los vínculos o links, con esta herramienta se podrán realizar comunicaciones o llamados con otras páginas o con otros sitios

La etiqueta <a> es la encargada de realizar esta tarea, se acompaña de múltiples parámetros, pero existe uno fundamental que es **href** que indica la dirección o ruta donde se encuentra el archivo o el sitio web a visitar

```
<html>
    <head>
        <title>Primer Documento Web</title>
    </head>

    <body bgcolor="#99CCFF">
        los sitios web son una fuente importante de consulta
        <br />
        <br />
        <br />
        <br />
        difundamos este metodo de trabajo
        <br />

        <a href="http://www.enter.co">Revista Enter</a>
    </body>
</html>
```

Autoria Propia



Autoria Propia

El **hipervínculo** mostrara la palabra “Revista Enter”, pero el llamado es , lo que se ubica en el **href** es una ruta o **url**, después de él se ubica una descripción del texto a llamar y cierra con la etiqueta

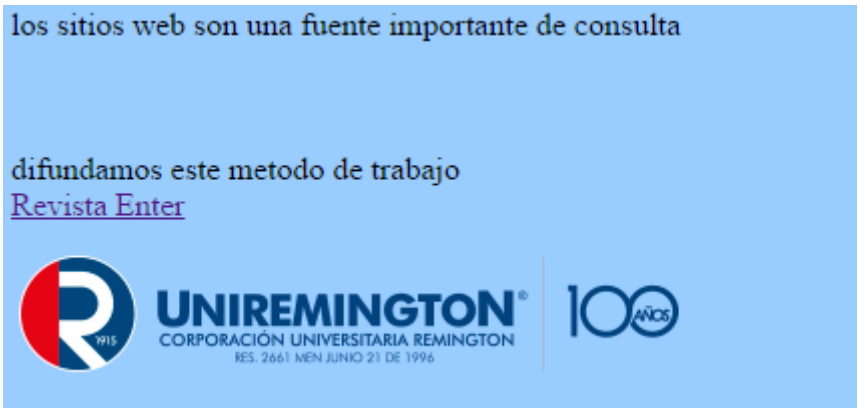
IMÁGENES

El diseño de un sitio web no se limita solo a el texto, los colores o los hipervínculos, las imagines hace parte fundamental de la presentación y de acercarse a las imágenes corporativas de las empresas.

LA ETIQUETA QUE SE UTILIZA ES <img />

Se utiliza como parámetro fundamental src (source o ruta del archivo)

<img src="logo.png">

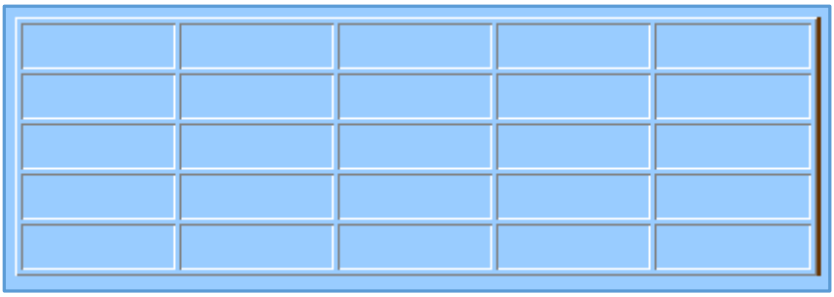


Autoria Propia

Estas imágenes se les pueden agregar bordes, se pueden convertir en hipervínculos.

TABLAS

Las tablas son contenedores, son herramientas que permiten realizar distribución de los elementos y dentro de ellos ubicar texto, imágenes, hipervínculos y otros elementos incluyendo tablas anidadas



Autoria Propia

Este un caso típico de una tabla compuesto por 5 filas y 5 columnas, es una matriz

Para la construcción de ella se requiere de otras etiquetas como son

<tr> </tr> establece el inicio y fin de una fila

<td></td> establece las celdas de la fila

```
<table width="400" border="1">
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
</table>
```

Autoria Propia

Dentro de los parámetros más comunes están width (ancho) y border (grosor del borde)

Existe un carácter especial entre cada <td> </td> este carácter representa un espacio, en este código se mostrarán dos filas y 5 columnas, en el carácter especial hay que tener presente que existen 256 caracteres con este formato, algunos de los que son importantes representan las tildes y caracteres especiales que los browsers no reconocen y que muestran un símbolo que dañaría el formato original.

Existen algunos parámetros adicionales dentro de los **<tr>** y los **<td>,** entre ellos la posibilidad de colocarle formatos como colores e imágenes de fondo y la posibilidad de cambiar filas o columnas

COMBINAR COLUMNAS

Para la combinación de columnas se utiliza el parámetro colspan y el número de columnas

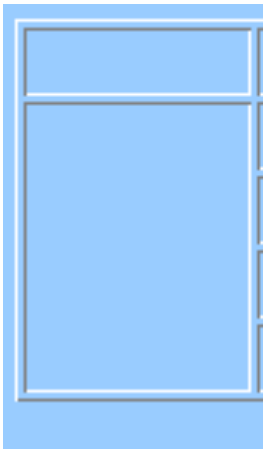
Autoria Propia

```
<table width="400" border="1">
  <tr>
    <td> </td>
    <td colspan="4"> </td>
  </tr>
  <tr>
    <td rowspan="4"> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
```

Autoria Propia

COMBINACIÓN DE FILAS

La combinación de columnas se realiza con la sentencia **rowspan** y el número de filas



Autoria Propia

```
<table width="400" border="1">
  <tr>
    <td> </td>
    <td colspan="4"> </td>
  </tr>
  <tr>
    <td rowspan="4"> </td>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
</tr>
```

Autoria Propia

FORMULARIOS

Los formularios son herramientas que permiten la interacción con el usuario, mediante estos se podrán solicitar datos, hacer cálculos y demás operaciones

Se conforma por la etiqueta <form></form>

Igual que las tablas es un contenedor, puede llevar distintos elementos como cajas de texto, botones, listas, etc., y contiene una serie de parámetros como son

Id: Nombre para identificar el formulario

Name: Nombre para identificar el formulario

Action: Especifica el archivo o la función que se realizara a la hora de enviar

los datos del formulario

Method: Representa la forma de paso de la información, existen dos opciones

tradicionales, POST y GET

elementos de los formularios

- cajas de texto
- áreas de texto
- botones de comando
- botones de radio (botones de opción)
- cajas de chequeo (casilla de verificación)
- lista / menú (comboBox)
- entre otros.

- ESTRUCTURA

- CAJAS DE TEXTO

<input name="caja" type="text" id="caja" size="20" maxlength="10" />

Se crea mediante la etiqueta **input** como muchos de los elementos de entrada de información, pero se especifica mediante el parámetro **type** que es un **text**, **id** y el **name** (nombre) permiten la identificación de la caja, **size** es el ancho que se ve y **maxlength** la cantidad de caracteres máximos que se pueden ingresar.

- ÁREAS DE TEXTO

<textarea name="comentario" id="comentario" cols="45" rows="5"></textarea>

Las áreas de texto son espacios mucho más amplios que las cajas de texto, se compone por id y name para identificarlas, cols para el número máximo de columnas que se mostraran y rows para el número máximo de filas visibles

- BOTONES DE COMANDO

<input type="submit" name="button" id="button" value="Almacenar">

<input type="reset" name="button2" id="button2" value="Restablecer">

Dos de los tipos de botones más comunes son los de envío y los de restablecer, igual de los demás elementos contienes un id y name, valué para mostrar al usuario un resultado y type para determinar que elemento es, en este caso un **submit** para el envío y **reset** para limpiar los elementos del formulario.

- BOTONES DE RADIO (BOTONES DE OPCIÓN)

<input type="radio" name="radio" id="radio" value="radio">

Los botones de radio o de opción permiten seleccionar una de muchas opciones

- **CAJAS DE CHEQUEO (CASILLA DE VERIFICACIÓN)**

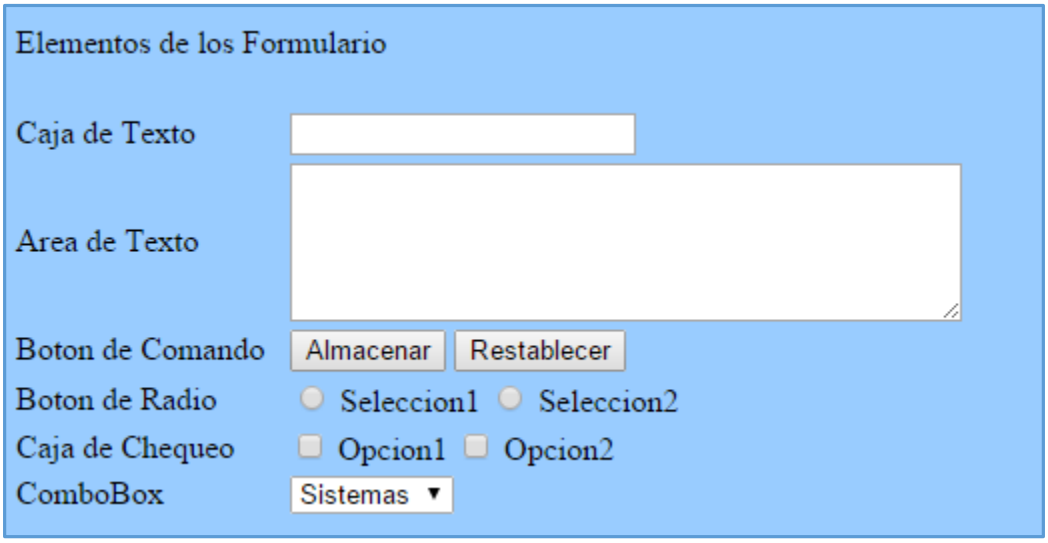
<input type="checkbox" name="checkbox" id="checkbox">

Las cajas de chequeo permiten la seleccione de uno, varios, todos o ningún elemento

LISTA / MENÚ (COMBOBOX)

```
<select name="select" id="select">
        <option value="1">Sistemas</option>
        <option value="2">Medicina</option>
        <option value="3">Derecho</option>
         </select>
```

Los comboBox permiten elegir de una lista de opciones, en value se especifica el valor a pasar y la otra información fuera de la etiqueta es lo que el usuario visualizara



Autoría Propia

HTML5

El HTML5 es una actualización del ambiente que por muchos años a estado al frente del desarrollo web, es probablemente el cambio más significativo que ha tenido el lenguaje, para este capítulo particular se enfocaran los cambios al manejo de formularios, teniendo en cuenta que en otras áreas también se presentaron cambios, pero por efectos de que esta última unidad está enfocada al desarrollo y creación de **CRUD** se enfocara muy particularmente a los controles.

Formulario a diseñar

Creación del formulario y tabla para la ubicación de los elementos

```
<form name="form1" method="post" action="." oninput="range_control_value.value=range_control.valueAsNumber" >
  <table width="550" border="0">
    <tr>
      <td colspan="2">Elementos de Formularios en HTML5</td>
    </tr>
    <tr>
      <td width="224"> </td>
      <td width="316"> </td>
  </tr>
```

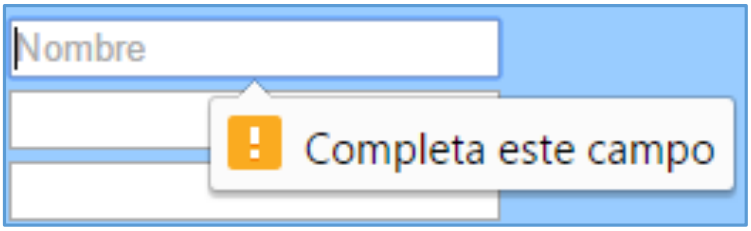
Autoria Propia

CAJA DE TEXTO CON CAMPOS REQUERIDOS, FOCO Y MENSAJE INTERNO

```
    <tr>
      <td>Campos Requeridos y Foco</td>
      <td><input type="text" name="nombre" id="nombre" autofocus required placeholder = "Nombre"/> </td>
  </tr>
```

Autoria Propia

En las cajas de texto de HTML al igual que en este primer ejemplo se utiliza id y **name** para identificar el elemento según el browser, los demás elementos pueden cambiar según el alcance, además el HTML anterior a la versión 5 solo tenía en el **type**, las palabras **text**, **hidden** y **password**, en esta versión nueva encontramos mayor número de alternativas y se verán en los siguientes controles, para este caso particular de campos requeridos se utiliza la sentencia **required**, con esto al momento de procesar la información si la caja de texto estuviera vacía mostraría un mensaje



Autoria Propia

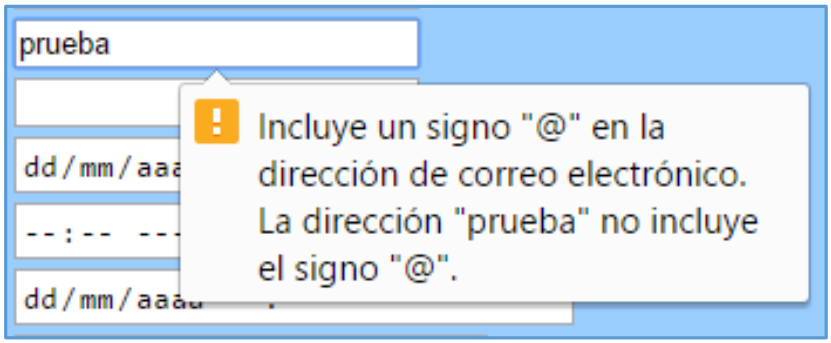
Además de esto se suma la propiedad autofocus, con esta propiedad lleva el cursor a esta caja con el fin de iniciar el proceso de digitación sin la ayuda del mouse, la recomendación para esta propiedad es que solo se utilice en una de las cajas de texto, la última propiedad que se va a trabajar para las cajas de texto tradicional es el **placeholder**, esta opción mostrara un mensaje en el interior de la caja de texto, en el momento de iniciar el ingreso de información esta desaparecerá, es ideal para ahorrar espacio y para dispositivos móviles.

CORREO ELECTRÓNICO

```
  <tr>
    <td>Correo Electronico</td>
    <td><input type="email" name="correo" id="correo" /></td>
</tr>
```

Autoria Propia

Se puede observar que el **type** contiene le valor email, este antes no se podía especificar, solo **text** o **password**, con esta instrucción el sistema validara que la información ingresada concuerde con el formato de un correo electrónico, se puede agregar **required** si se prefiere



Autoria Propia

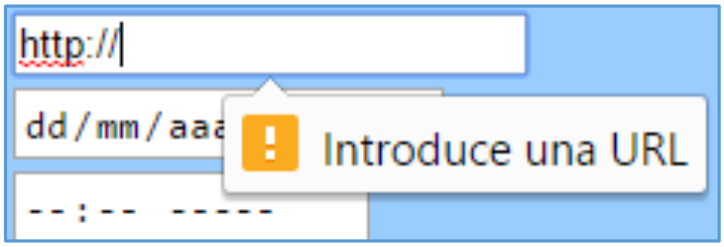
Desde el inicio de la digitación se indicará que incluya una arroba y los demás componentes de un correo electrónico

URL

Las cajas de texto para **url** tendrá esta palabra en el **type**, validaran que la dirección de un sitio sea cumpla las normas mínimas

```
  <tr>
    <td>URL</td>
    <td><input type="url" name="url" id="url" required /></td>
</tr>
```

Autoria Propia



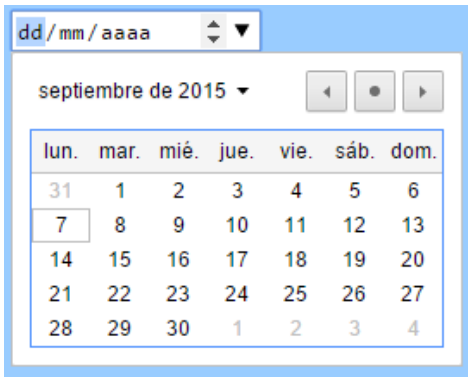
Autoria Propia

FECHAS

En la versión previa de HTML para crear un formato de fechas se recurría a herramientas como JavaScript, con el HTML5 y la instrucción date dentro del type se soluciona este impase.

```
  <tr>
    <td>Fecha</td>
    <td><input type="date" name="fecha" id="fecha" /></td>
</tr>
```

Autoria Propia



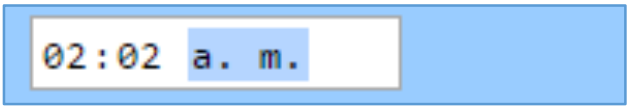
Autoria Propia

HORA

Se agrega la instrucción time dentro del type

```
  <tr>
    <td>Hora</td>
    <td><input type="time" name="hora" id="hora" /></td>
</tr>
```

Autoria Propia



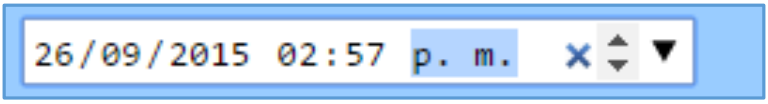
Autoria Propia

FECHA Y HORA

Esta etiqueta mezcla las dos anteriores

```
  <tr>
    <td>Fecha y Hora</td>
    <td><input type="datetime-local" name="fechaHora" id="fechaHora" /></td>
</tr>
```

Autoria Propia



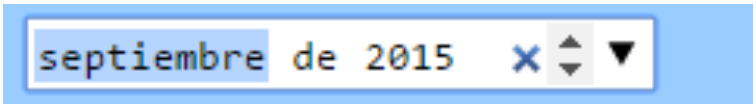
Autoria Propia

MESES

Con la instrucción **month** en el parámetro **type**

```
  <tr>
    <td>Mes</td>
    <td><input type="month" name="mes" id="mes" /></td>
</tr>
```

Autoria Propia



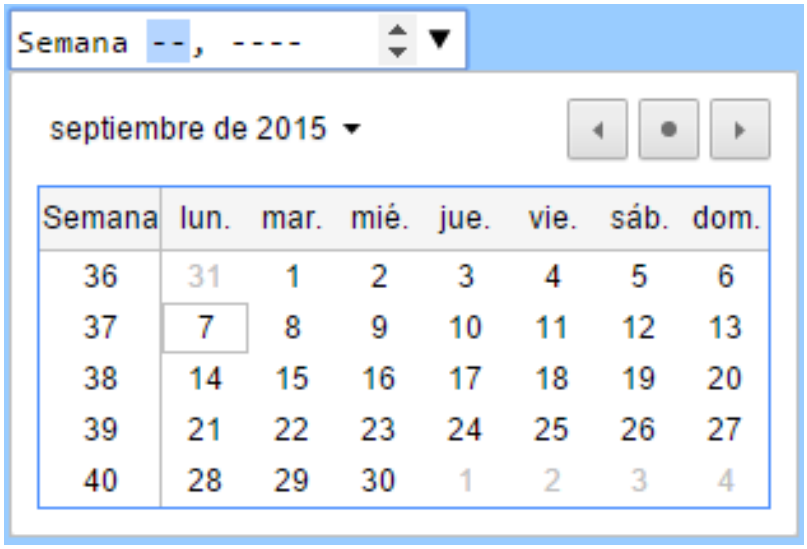
Autoria Propia

SEMANA

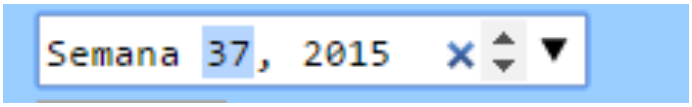
Permite seleccionar el número de semana del año y representarlo en la fecha a la que corresponde

```
  <tr>
    <td>Semana</td>
    <td><input type="week" name="semana" id="semana" /></td>
</tr>
```

Autoria Propia



Autoria Propia

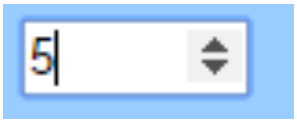


Autoria Propia

RANGO DE NÚMEROS

```
  <tr>
    <td>Rango de Numeros</td>
    <td><input type="number" name="rango" id="rango" min="1" max="50" /></td>
</tr>
```

Autoria Propia



Autoria Propia

INTERVALOS

```
  <tr>
    <td>Intervalo</td>
    <td><input type="range" name="range_control" id="range_control" min="1" max="50" />
    <output for="range_control" name="range_control_value">0</output></td>
</tr>
```

Autoria Propia

Autoria Propia

PISTAS DE APRENDIZAJE

HTML: Lenguaje de marcas de hipertexto

Etiquetas: Comando que permite crear un proceso determinado

Formulario: Contenedor de elementos clase para con contacto con el usuario

4.3 TEMA 2 CSS HOJA DE ESTILO EN CASCADA

Con mucha frecuencia en la construcción de sitios web se presenta que los formatos no son uniformes o que se tienen que aplicar en cada página, cuando el sitio es considerablemente grande este tipo de formatos no son administrables y se puede recurrir a la construcción de un CSS (Cascading Style Sheet) Hojas de Estilo en Cascada.

Esta herramienta permite que de una manera simple se puedan administrar N cantidad de páginas de manera uniforme y con una codificación simple

TIPOS DE CSS

Existen 3 categorías para los CSS

CSS en línea: permite aplicar formatos a una etiqueta particular

CSS en bloque: permite aplicar formatos a una o varias etiquetas dentro del mismo archivo

CSS en archivo: permite la administración de múltiples páginas.

EJEMPLO DE UNA PAGINA SIN FORMATOS

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio ei.

Autoria Propia

Estos bloques de código corresponden a un sitio web que no tiene formato alguno, la estructura es la siguiente

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Formatos CSS</title>
</head>
<body>
    <p>Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber
hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis
reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis
mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores.
Quo no falli viris intellegam, ut fugit veritus placerat per.<p>
    <p>Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu.
No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto
zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam
legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio
 ei.</p>
</body>
</html>
```

Autoria Propia

APLICACIÓN DE FORMATOS EN LÍNEA

CSS se compone de una innumerable cantidad de opciones, algunas de ellas son

Font-family: Especifica la fuente o familia de estas

Font-size: Determina el tamaño de la fuente que se empleara, se puede especificar en pixeles (px), puntos (pt), pulgadas (in), centímetros(cm), milímetros(mm), picas (pc)

Text-align: Alineación del texto a la derecha (right), izquierda (left), centrado (center), justificado (justify)

Font-wieght: Intensidad de la fuente, los valores van entre 100 y 900, bold

Text-transform: Se puede trasformar el texto, **upper** (mayúsculas), **lower** (minúscula), **capitalize**(primera letra en mayúscula)

Color: Especifica el color de fuente, se puede especificar en formato hexadecimal, en formato RGB o con el nombre del color

Background-color: Color de fondo

Background-image: Imagen de fondo

Margin: En este formato se puede crear una margen de contorno, el valor que se especifique aplicara a la derecha, izquierda, arriba y abajo

Margin-left

Margin-right

Margin-top

Margin-button

Son complementos de la anterior

Border: Especifica un borde en contorno

Border-left

Border-right

Border-top

Border-button

Text-decoration: Aplica para colocar subrayados o para quitarlos

Line-heigth: Especifica el espacio entre líneas

Width: Ancho de un elemento

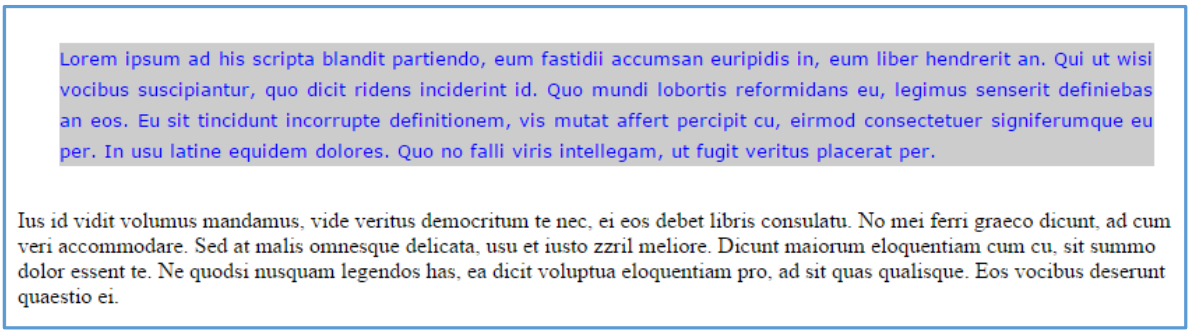
Los formatos en línea solo aplican a la etiqueta que lo requiera

APLICANDO LOS FORMATOS A LA PRIMERA ETIQUETA

```
    <p style="font-family:Verdana, Geneva, sans-serif; font-size:10pt; color:#00F; background-color:#CCC; margin:8mm; line-height
:4mm; text-align:justify">Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit
an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit
definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu
per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.<p>
```

Autoria Propia

Arroja como resultado lo siguiente



Autoria Propia

Los dos párrafos están dentro de la etiqueta ** **, pero solo se aplica el formato en línea a la primera, esta es la forma en la que trabaja el formato lineal.

FORMATO EN BLOQUE

Para la creación de un bloque de estilos se ubica en la cabecera de la página, se especifica la o las etiquetas, están aplicaran el formato a todas las etiquetas que se especifiquen.

```
<head>
<title>Formatos CSS</title>
<style type="text/css">

p    {

    font-family:Verdana, Geneva, sans-serif;
    font-size:10pt;
    color:#00F;
    background-color:#CCC;
    margin:8mm;
    line-height:6mm;
    text-align:justify;
}
</style>
</head>
```

Autoria Propia

La etiqueta ya no tiene formatos en línea y el resultado es el siguiente

Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit tincidunt incorrupte definitionem, vis mutat affert percipit cu, eirmod consectetuer signiferumque eu per. In usu latine equidem dolores. Quo no falli viris intellegam, ut fugit veritus placerat per.

Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto zzril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio ei.

Autoria Propia

En este caso se observa que los dos párrafos tienen la misma distribución y los mismos formatos.

CSS EN ARCHIVO

La limitante de aplicar formatos en bloque consiste en que solo serían formatos para una página, cuando se requiere que estos formatos se den en varias páginas debemos expórtalo

En un archivo independiente con extensión CSS aplicamos los formatos

La invocación de un archivo externo se aplica de la siguiente forma

```
<head>
<title>Formatos CSS</title>
<link rel="stylesheet" type="text/css" href="formatos.css" />
</head>
```

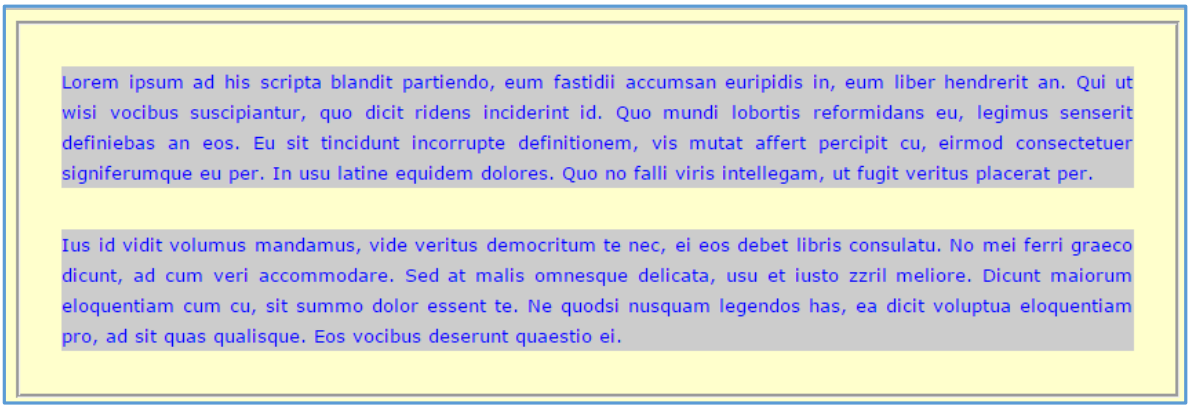
Autoria Propia

En el archivo de ejemplo ya no hay formatos, existe el llamado a un **archivo que contendrá formatos globales** para todas las páginas asociadas. Todas las páginas del mismo sitio que contengan esta línea de código mostrara el mismo formato, si se requiere un cambio de color, de fuente, de márgenes, etc., solo tendrá que ingresar al archivo, cambiarlo y al almacenar y ejecutar cualquiera de los archivos HTML mostrara dicha actualización.

```
p    {

    font-family:Verdana, Geneva, sans-serif;
    font-size:10pt;
    color:#00F;
    background-color:#CCC;
    margin:8mm;
    line-height:6mm;
    text-align:justify;
}

body    {

    font-family:tahoma;
    font-size:9pt;
    background-color:#FFC;
    border:groove;
}
```

Autoria Propia

Resultado



Autoria Propia

FORMATOS EN FORMULARIOS

Formulario sin formatos CSS

Autoria Propia

A este mismo formulario se le aplican algunos formatos de CSS.

Autoria Propia

Los formatos CSS son muchos y muy variados, aplique los mas esenciales que estamos tratando, implemente nuevas alternativas con caracteristicas nuevas de CSS2, CSS3.

PISTAS DE APRENDIZAJE

CSS: **Hoja de estilo en cascada**

Link: **Etiqueta que invoca un archivo CSS externo**

4.4 TEMA 3 JAVASCRIPT

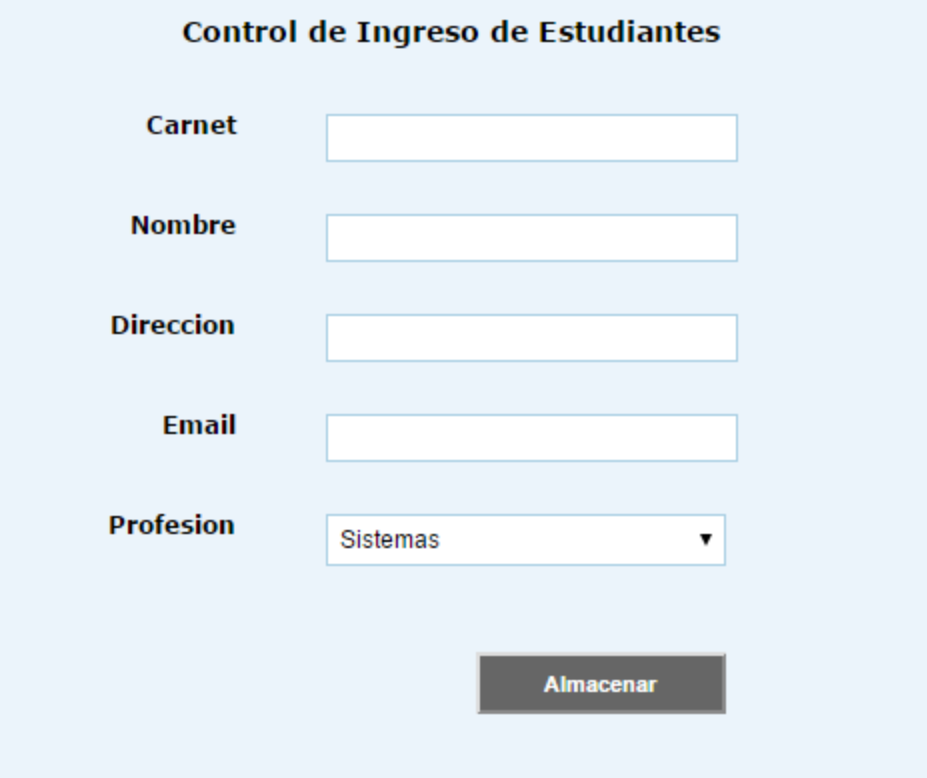
JavaScript es un lenguaje interpretado, tal vez uno de los más conocidos desde hace muchos años por su versatilidad, su gran poder y por dejar una gran herencia con otros ambientes como JQuery, Ajax, JSon, etc., después de muchos años sigue como uno de los principales dentro de la programación web.

ESTRUCTURA

En la estructura no se entrará mucho en detalles dado que tiene una similitud con Java y con C++, la forma de establecer ciclos paras o mientras, condicionales o selectores múltiples tienen las mismas estructuras.

JavaScript **se va a utilizar principalmente como herramienta de validación**, es un tema relativamente corto, pero de gran importación, se aplicará solo este tema por motivos de aplicabilidad en un tema posterior, la utilización del JavaScript (JS) es muy variado y muy amplio, a lo que se invita a seguir leyendo sobre el tema y no dejarlo solo en esta etapa inicial.

FORMULARIO INICIAL



Autoria Propia

Mediante este formulario se aplicarán los conceptos de validar que un campo no este vacío y que cumpla las condiciones mínimas solicitadas.

Para este caso se cuenta con carnet, nombre, dirección, email y profesión, se procederá con la validación de todos los campos que permitan el ingreso de datos.

Para el manejo de JavaScript hay que tener presente que se puede aplicar en condiciones similares a CSS, se pueden crear tareas en línea, en bloque o en archivos independientes, para este caso se aplicaran las validaciones en un archivo externo.

Para este ejemplo se creará un archivo llamado validar.js, para la vinculación de un archivo .js dentro de uno .HTML se procede a realizar la siguiente línea en la cabecera de la pagina

```
<head>
<title>Formatos CSS</title>
<link rel="stylesheet" type="text/css" href="formatos.css" />
<script type="text/javascript" src="validar.js"></script>
</head>
```

Autoria Propia

La instrucción del script especifica el tipo que es texto/ JavaScript y un parámetro **src** que indica la ruta y el archivo donde se encuentra la validación

Se utilizarán expresiones regulares para la validación, esto permitirá que de una forma sencilla y corta se pueda realizar múltiples procesos.

DECLARACIÓN Y ASIGNACIÓN DE LOS CAMPOS DEL FORMULARIO

```
function validacion () {

    var carnet = document.getElementById("carnet").value;
    var nombre = document.getElementById("nombre").value;
    var direccion = document.getElementById("direccion").value;
    var email = document.getElementById("email").value;
```

Autoria Propia

> Al trabajar con un archivo externo para las validaciones en JavaScript se inicia con la sentencia function y el nombre de la función, acompañado de paréntesis que indican que pueden ir parámetros

En las líneas siguientes se declara cada una de las variables que contiene el formulario, y se asigna el valor de las cajas de texto a estas variables con la sentencia document.getElementById("nombredelcontrol").value

Posterior a este paso se procede con la primera validación, especificar que el carnet no este vacío

```
if (carnet == null || carnet == 0 || /^\s+$/.test(carnet)){

    alert ("Digite un numero de Carnet");
    return false;
}
```

Autoria Propia

Se especifica que el campo no sea nulo, no sea 0 y no esté compuesto solo por espacios

La sentencia **alert** muestra una ventana emergente, el **return** en false indica que no se cumplió la condición

- VALIDACIÓN DEL CONTENIDO DEL CARNET

```
else if (!(/^\d{12}$/.test(carnet))) {

    alert ("Carnet no Valido, Ingrese 12 Digitos, solo Numeros");
    return false;
}
```

Autoria Propia

En esta instrucción apreciamos las características de una expresión regular, algunas de sus características son

^ indica el inicio de una cadena

$ indica el final de una cadena

d indica valor entero

{12} indica que solo se pueden ingresar 12 caracteres

VALIDACIÓN DE CAMPO TIPO TEXTO (NOMBRE)

Para la validación de un campo tipo texto se aplica la primera validación con el fin de que el campo no este vacío y luego se aplican las condiciones del campo

```
else if (!(/^[a-z A-Z]{7,40}$/.test(nombre))) {

    alert ("Nombre no Valido, Ingrese entre 7 y 40 Caracteres");
    return false;
}
```

Autoría Propia

En esta instrucción se valida que tenga un rango de letras de la “a” a la “z” tanto en minúscula como en mayúscula, un espacio y que tenga un rango de caracteres entre 7 y 40

VALIDACIÓN DE UNA DIRECCIÓN (CAMPOS CON TEXTO Y NÚMEROS)

```
else if (!(/^[a-z A-Z0-9-]{7,40}$/.test(direccion))) {

    alert ("Direccion no Validad, Ingrese entre 7 y 40 Caracteres");
    return false;
}
```

Autoría Propia

Tiene un contenido similar a el nombre, con la variación de 0-9 que indica que recibe número de cero a nueve y guiones, además de permitir un rango de datos

VALIDACIÓN DE CAMPOS EMAIL

```
else if (!(/^([\da-z_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/.test(email))) {

    alert ("E-Mail no Valido, Ingrese entre 7 y 40 Caracteres");
    return false;
}
else

    return true;
```

Autoría Propia

La validación de este tipo de campos comprenda una mayor cantidad de alternativas, permite números enteros, letras de la "a" a la "z" y **underline** (_) antes de la arroba, después de la arroba tiene una serie de caracteres similares y permite al final entre 2 y 6 caracteres para el domino de la dirección.

En el último **else** se aplica un **return** true, esto indica que cuando cumpla todas las condiciones la validación es valida

Por ultimo en el formulario se agrega el llamado a la función

```
<form id="form1" name="form1" method="post" action="." onsubmit="return validacion()">
```

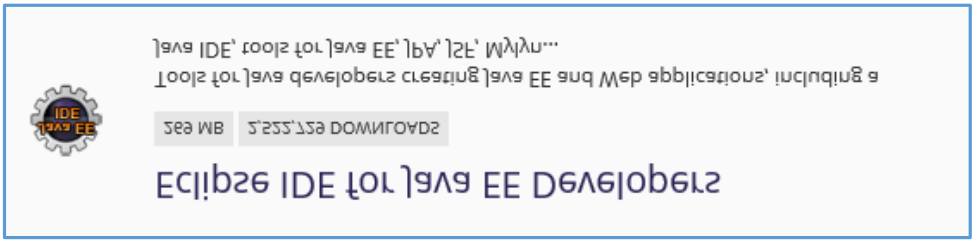
Autoría Propia

Este evento **onsubmit** realiza el llamado de la función cada que se presiona el botón Almacenar, comprende **return** validación, con esta instrucción el sistema **recibe el true o false según la validación de los campos**, el valor que hay dentro del parámetro **action** es opcional, ahí se ubica el archivo que se va a trabajar si las condiciones se cumplen.

4.5 TEMA 4 JSP / SERVLETS

JSP (Java Server Page) es una herramienta complementaria de desarrollo web, la base de todo sitio web es y será HTML, que se complementa con herramientas como CSS, JS, entre otros elementos, JSP hace parte de un selecto grupo de opciones que permiten una mayor interacción, la comunicación y el acceso a las BD, se trabaja del lado del servidor y no del cliente como las otras herramientas.

Para la creación y utilización de un archivo JSP se trabaja con Eclipse EE, se puede descargar del sitio www.eclipse.org



Autoría Propia

También se requiere **tomcat** que es un complemento del Apache y permite la interpretación del sitio diseñado.

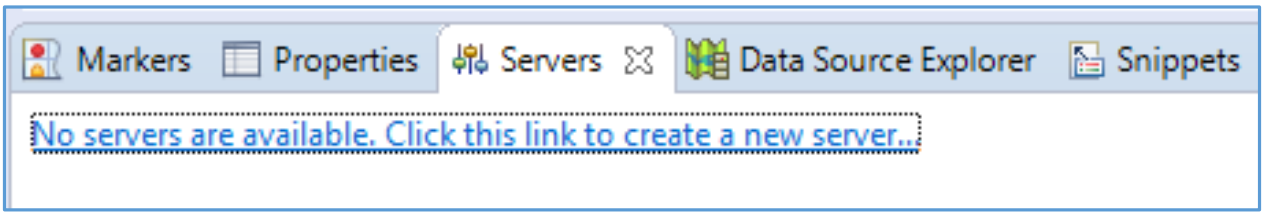
Autoría Propia

Para este capítulo se trabajará con la versión 8

CREACIÓN DEL SERVIDOR

Para este proceso del JSP se requiere crear un servidor con la o las configuraciones necesarias para la interacción e interpretación de la nueva codificación.

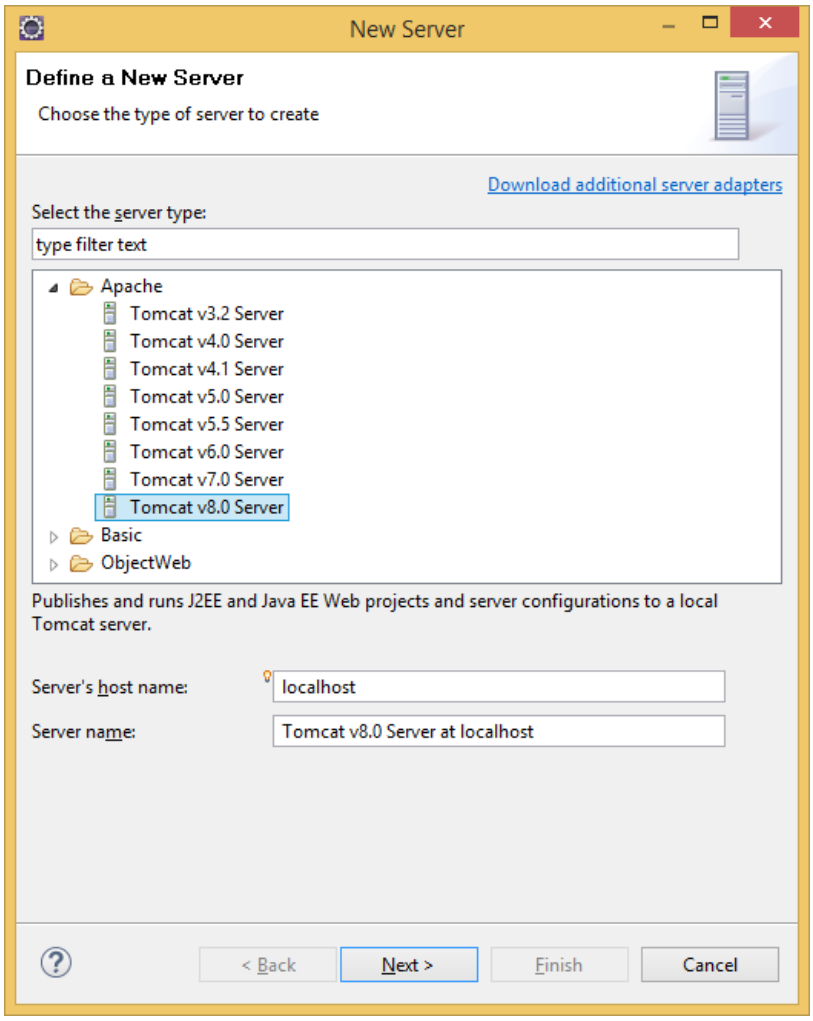
Ubicados en la parte inferior del IDE en la pestana Server



Autoría Propia

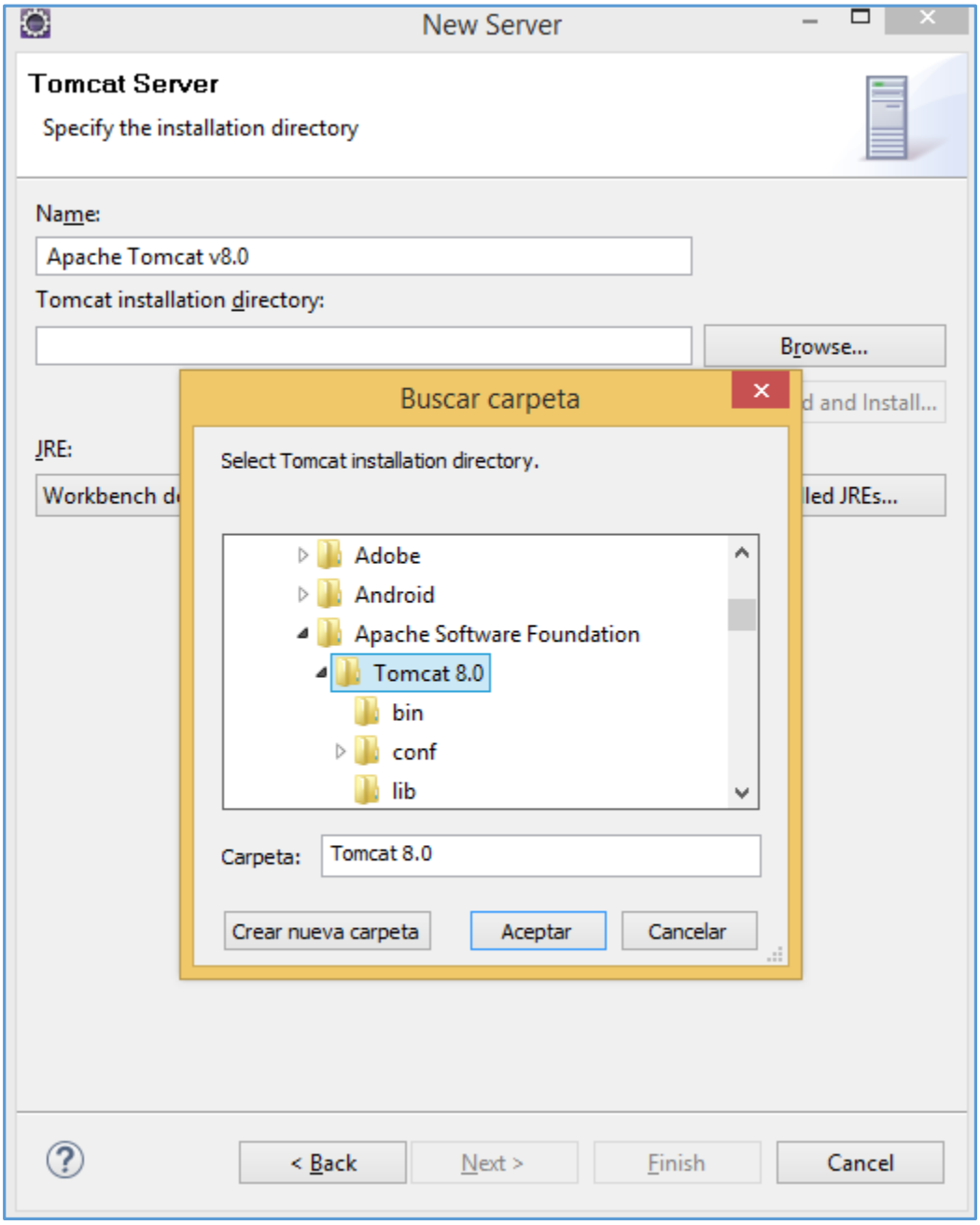
Se selecciona el vínculo **No servers are available. Click this link to create a new server...**

En la ventana contigua se selecciona apache y la versión más reciente que se tenga disponible y / o se haya descargado



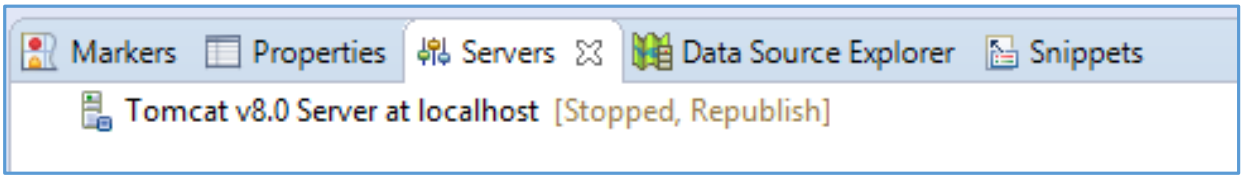
Autoría Propia

Se continua con el botón **Next**,



Autoría Propia

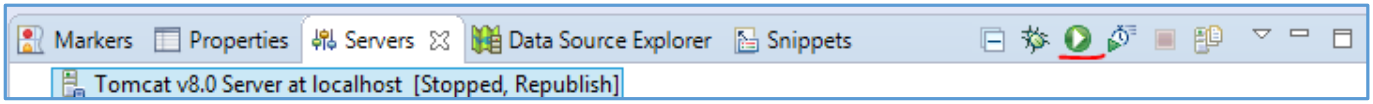
En la opción **Tomcat Installation directory** se busca la ubicación de **tomcat**, habitualmente se encuentra en archivos de programas, **apaches Software Foundation** y se selecciona la versión del **tomcat**, luego se acepta y finaliza



Autoría Propia

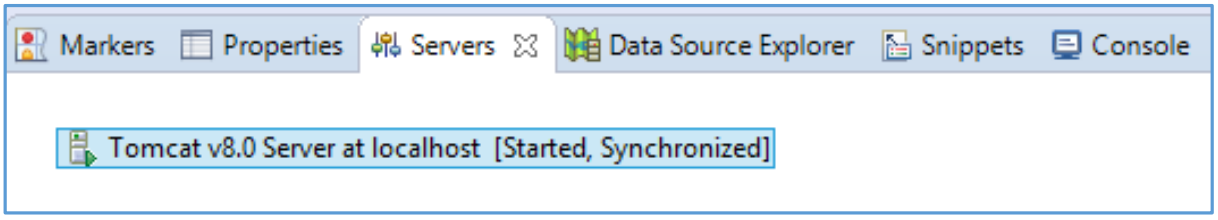
La pestana Servers tiene una configuración en este momento lista para ser usada

PROBAR LA CONFIGURACIÓN



Autoría Propia

Al lado derecho de la opción server se encuentra un icono para la ejecución, verifique el buen funcionamiento del servidor

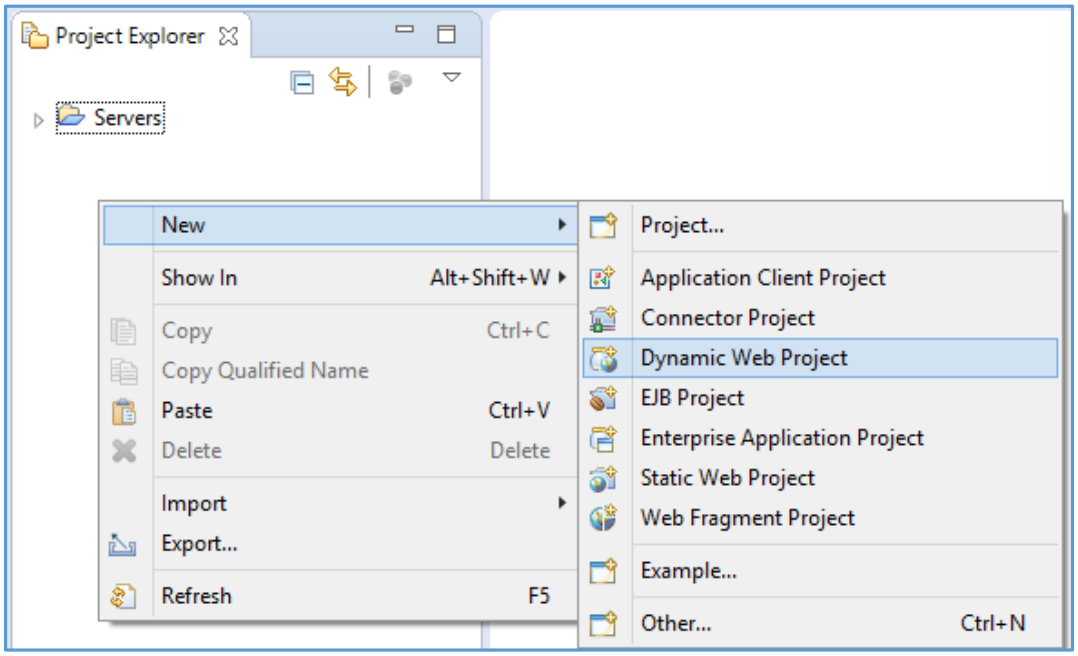


Autoría Propia

Se observa que el estado cambia de **Stopped a Started** y está listo para su uso

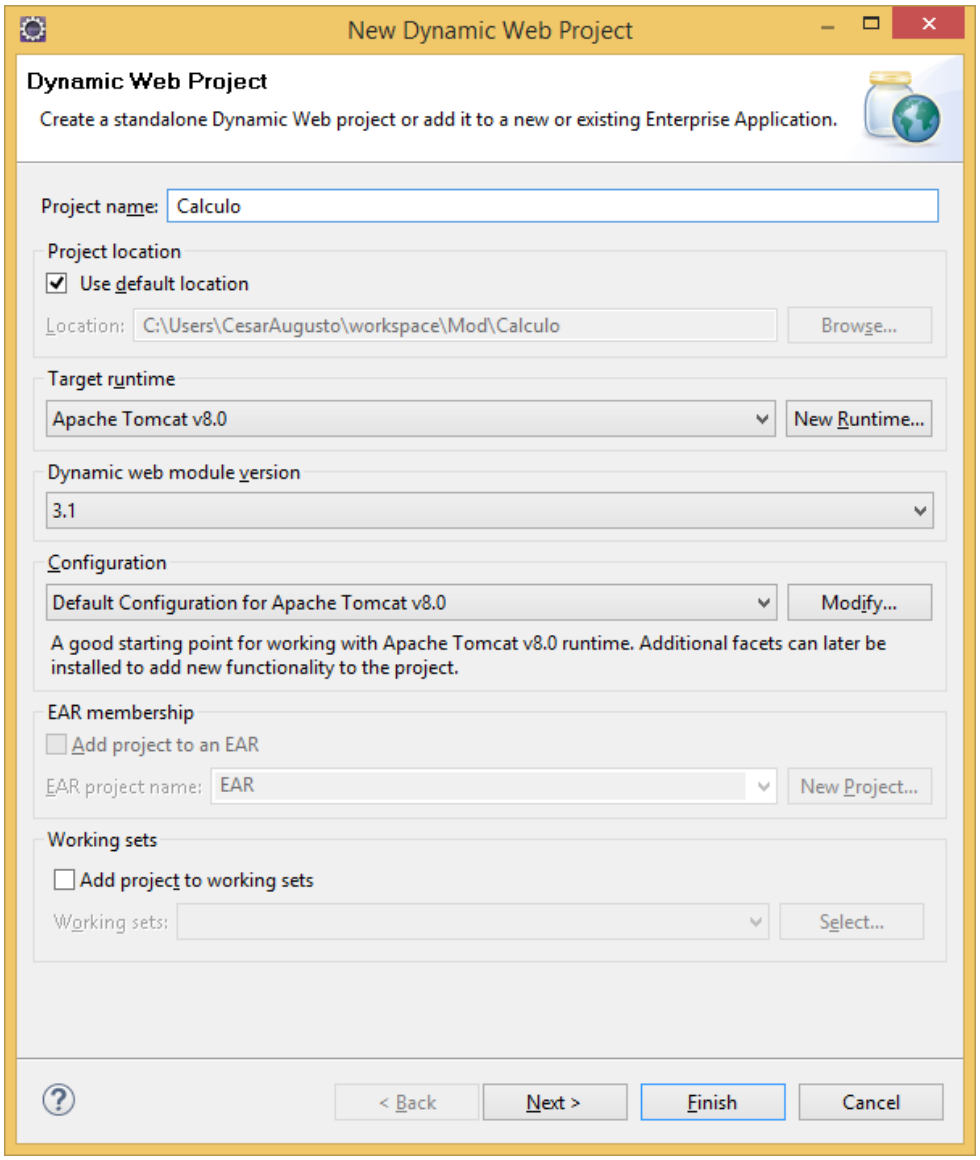
CREACIÓN DEL PRIMER PROYECTO

Ubicados en el Project Explorer y con botón emergente, se selecciona **new / Dinamic Web Project**



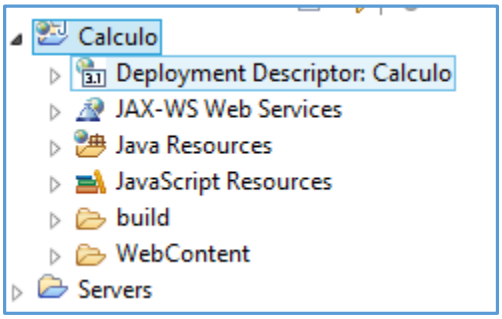
Autoría Propia

Se especifica el nombre del proyecto



Autoría Propia

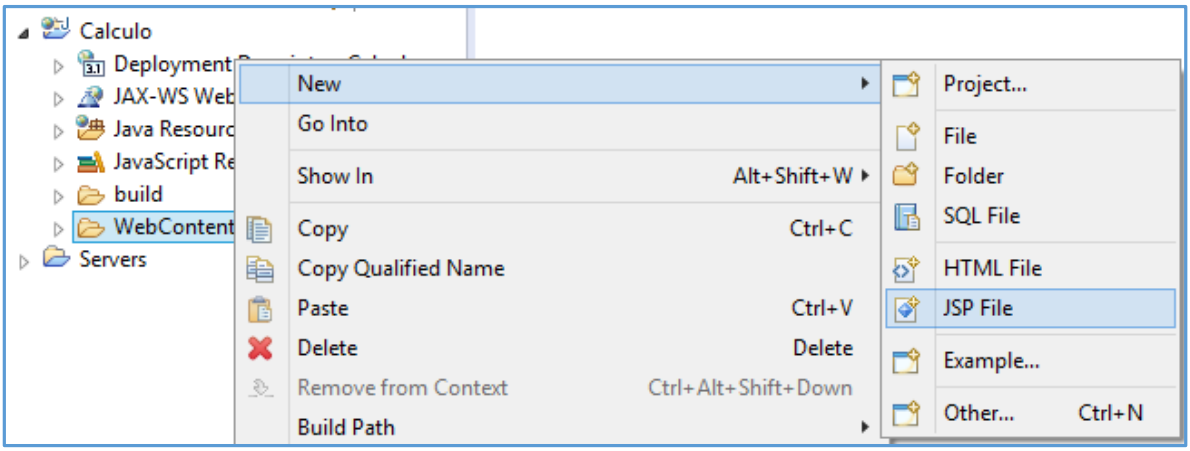
Y se finaliza



Autoría Propia

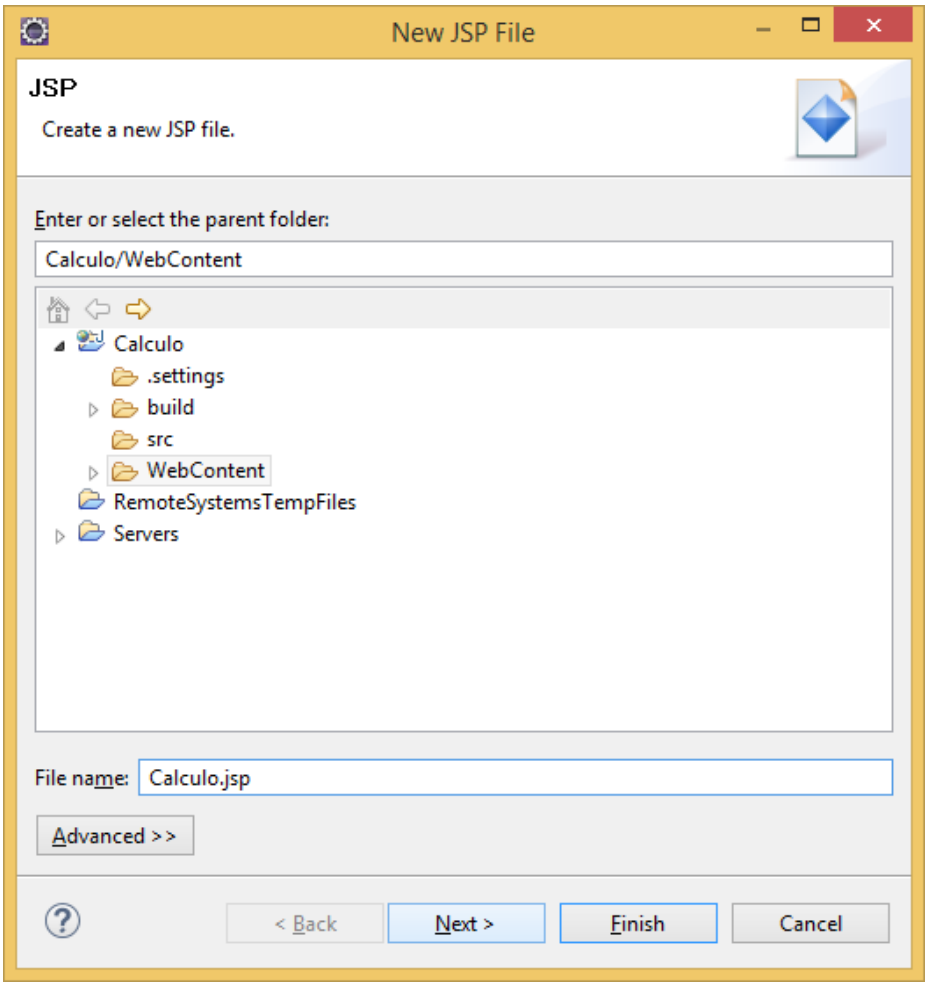
Esta es la estructura de un proyecto nuevo, dos de los aspectos mas comunes e importantes son el **Java Resources** donde se ubicarán los archivos con extensión .java y **WebContent** donde se ubicarán los archivos JSP

CREACIÓN DE UN ARCHIVO JSP



Autoria Propia

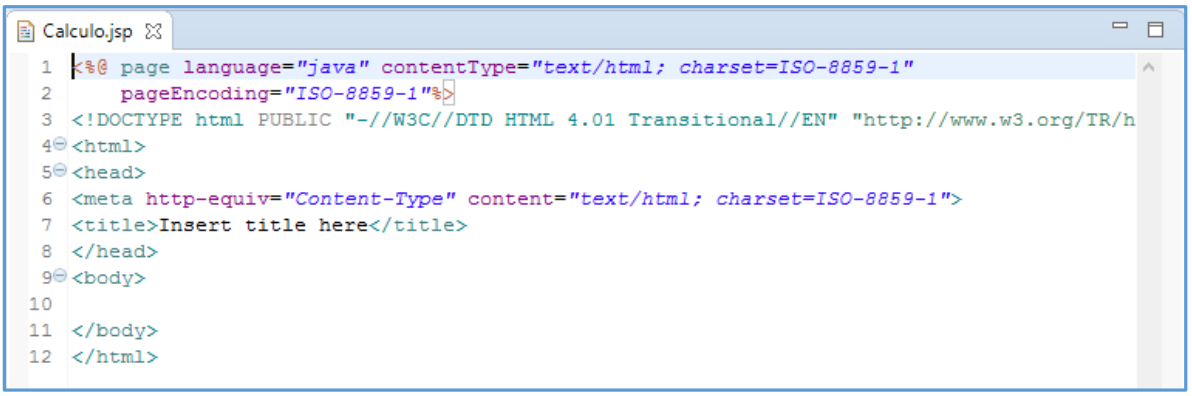
En el WebContent botón emergente, new / JSP File



Autoría Propia

Y finaliza la creación.

APARIENCIA INICIAL



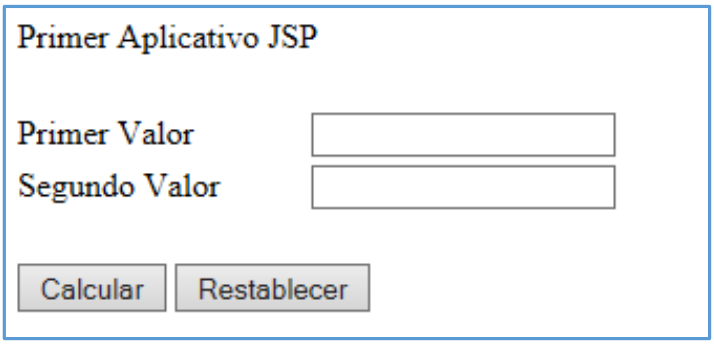
Autoría Propia

Contiene la mayor parte del código HTML pero en la primera línea de código se ve una serie de símbolos propios de JSP

<%@ %>

Estos son los símbolos que representan el trabajo con JSP, indica además el lenguaje, el contexto y una colección como es la ISO.

CREACIÓN DE UN FORMULARIO



Autoría Propia

Este formulario tiene las mismas características de los temas anteriores (creación de formularios), se ubica dentro del **body** del archivo creado, se podrá ejecutar, aunque no arroje ningún resultado.

APLICACIÓN DE LA OPERACIÓN MEDIANTE ARCHIVO JSP

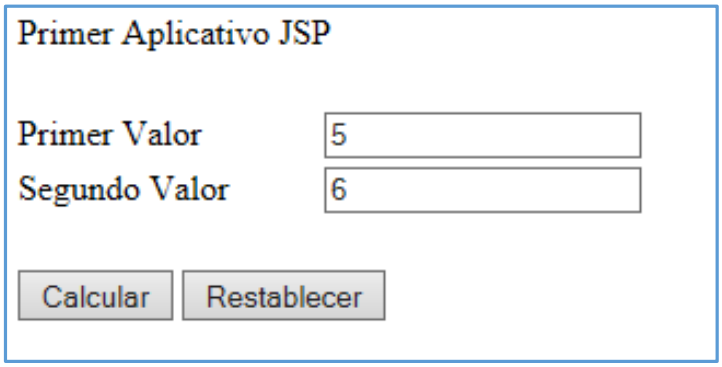
Se crea un archivo nuevo llamado **Resultado.jsp** y se invoca en el formulario anterior en el parámetro action el nombre del archivo y la extensión de este.

```
<form id="form1" name="form1" method="post" action="Resultado.jsp">
```

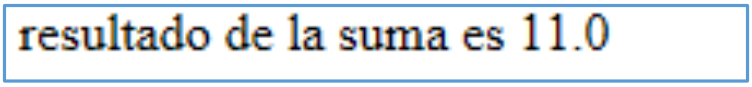
Autoría Propia

```
<body>
    <%
        float pv = Float.parseFloat(request.getParameter("primerValor"));
        float sv = Float.parseFloat(request.getParameter("segundoValor"));
        float total = pv + sv;
        out.print ("resultado de la suma es " + total);
    %>
</body>
```

Autoría Propia



Autoría Propia



Autoría Propia

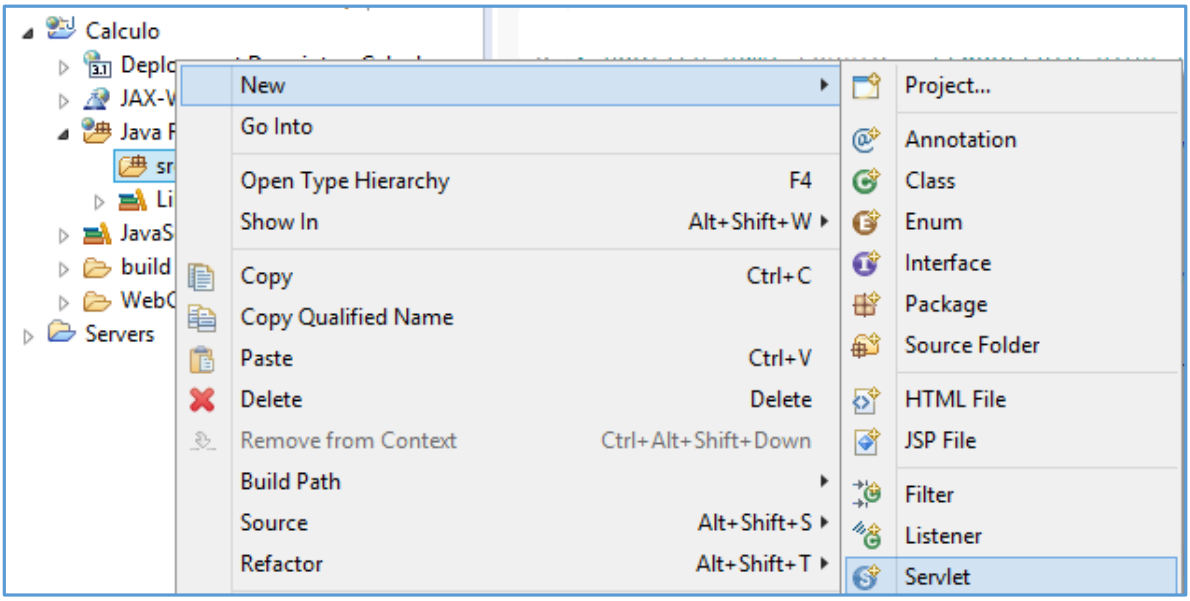
En el formulario creado inicialmente no tiene cambios de codificación salvo la línea de JSP que se establece automáticamente, pero se empieza viendo un parámetro como action que contiene un valor (**Resultado.jsp**), esta acción es el archivo o la función que se desea ejecutar.

Observe en el archivo Resultado.jsp que a una variable float pv se le hace una conversión y dentro de esta aparece la instrucción request. getParameter, esta instrucción toma el contenido de la caja de texto primerValor, lo mismo sucede para la segunda variable, tenga muy presente la escritura de las variables o campos, el java es sensible a mayúsculas y minúsculas.

Aplicación del Mismo formulario mediante **Servlets**

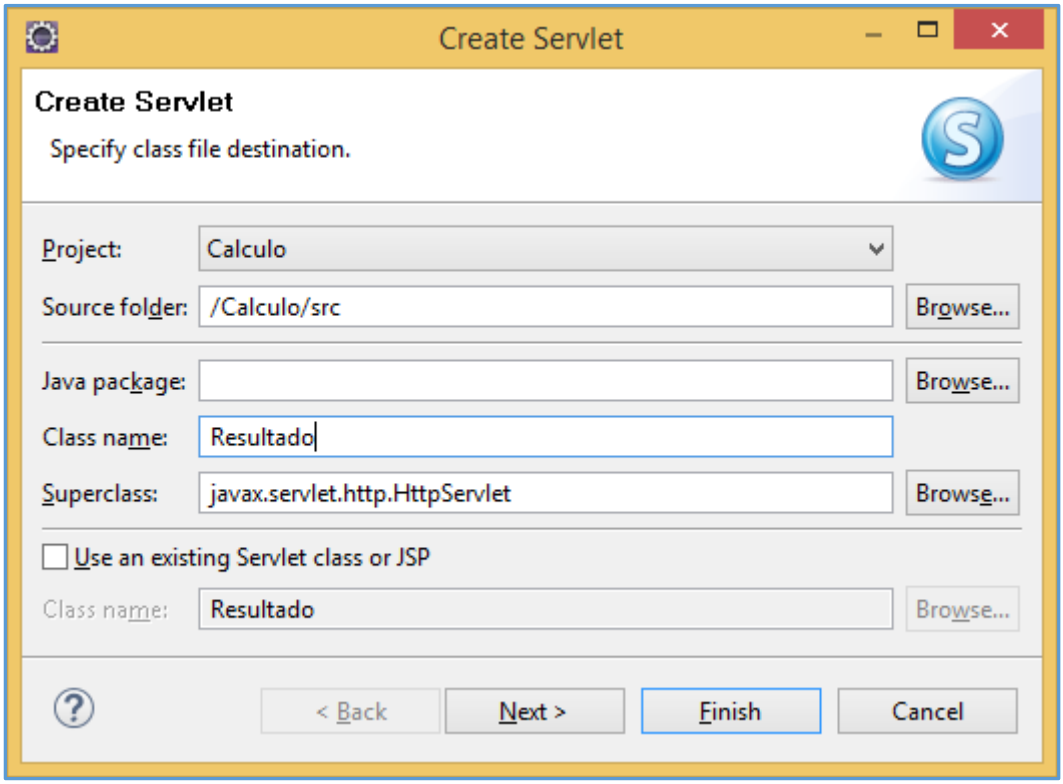
Los servlets son archivos con extensión java diseñados para el manejo de los datos de un formulario mediante métodos como post (doPost) get (doGet)

CREACIÓN DE UN SERVLET



Autoría Propia

Con el botón emergente ubicados sobre el **src** **del Java Resources, new / Servlet**



Autoría Propia

Y se finaliza.

Componentes del Servlet

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class Resultado
 */
@WebServlet("/Resultado")
public class Resultado extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Resultado() {
        super();
        // TODO Auto-generated constructor stub
    }
```

Autoría Propia

Se compone en su parte superior con los paquetes de uso, una anotación fundamental en la ejecución de este @WebServlet, nombre de la clase constructor

METODO DOPOST Y DOGET

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub
}
```

Autoría Propia

Útiles para la recepción de la información, esta llega inicialmente al doGet y al ser procesada pasa al doPost

IMPLEMENTACIÓN

Se hace le llamado desde el **action** del formulario

```
<form id="form1" name="form1" method="post" action="Resultado">
```

Autoría Propia

DESARROLLO DEL SERVLET

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub

    float pv = Float.parseFloat(request.getParameter("primerValor"));
    float sv = Float.parseFloat(request.getParameter("segundoValor"));
    float total = pv + sv;

    System.out.print ("Resultado de la suma :" + total);

}
```

Autoría Propia

Resultado

```
Resultado de la suma :15.0
```

Autoría Propia

4.6 TEMA 5 JAVABEANS

Dentro de los modelos nuevos de desarrollo cada día se encuentras más alternativas, una de ella son los Beans o JavaBeans, este modelo o patrón, cumple la tarea de "**clase principal**", en una clase sin cara (sin diseño gráfico) pero permite el tránsito de la información, todos los procesos pasan por esta clase, la información se actualiza en esta y el proceso que lo requiera siempre y cuando tenga acceso podrá tomarlos procesarlos y devolverlos, así en un ciclo constante la información estará disponible, además de brindar seguridad y que no se tenga que acceder hasta el formulario o a otra clases más restringidas.

CREACIÓN DE UN JAVABEAN

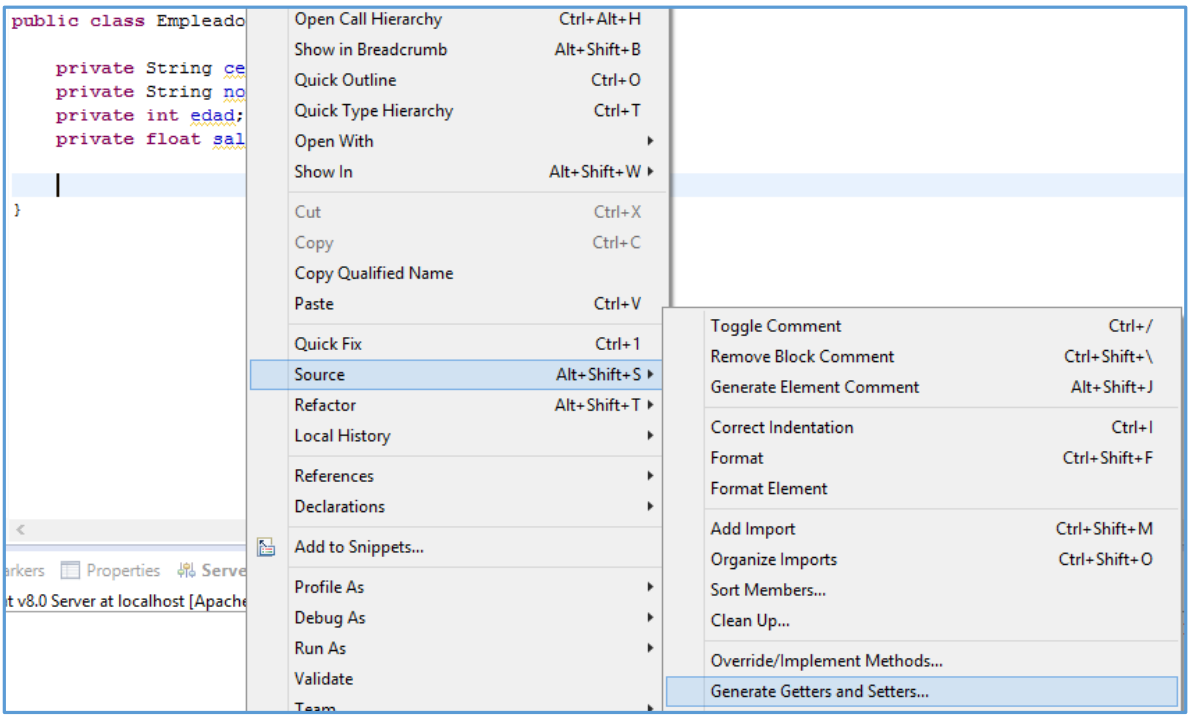
Se crea una clase Empleado.java

```
public class Empleado {

    private String cedula;
    private String nombre;
    private int edad;
    private float salario;


}
```

Autoría Propia

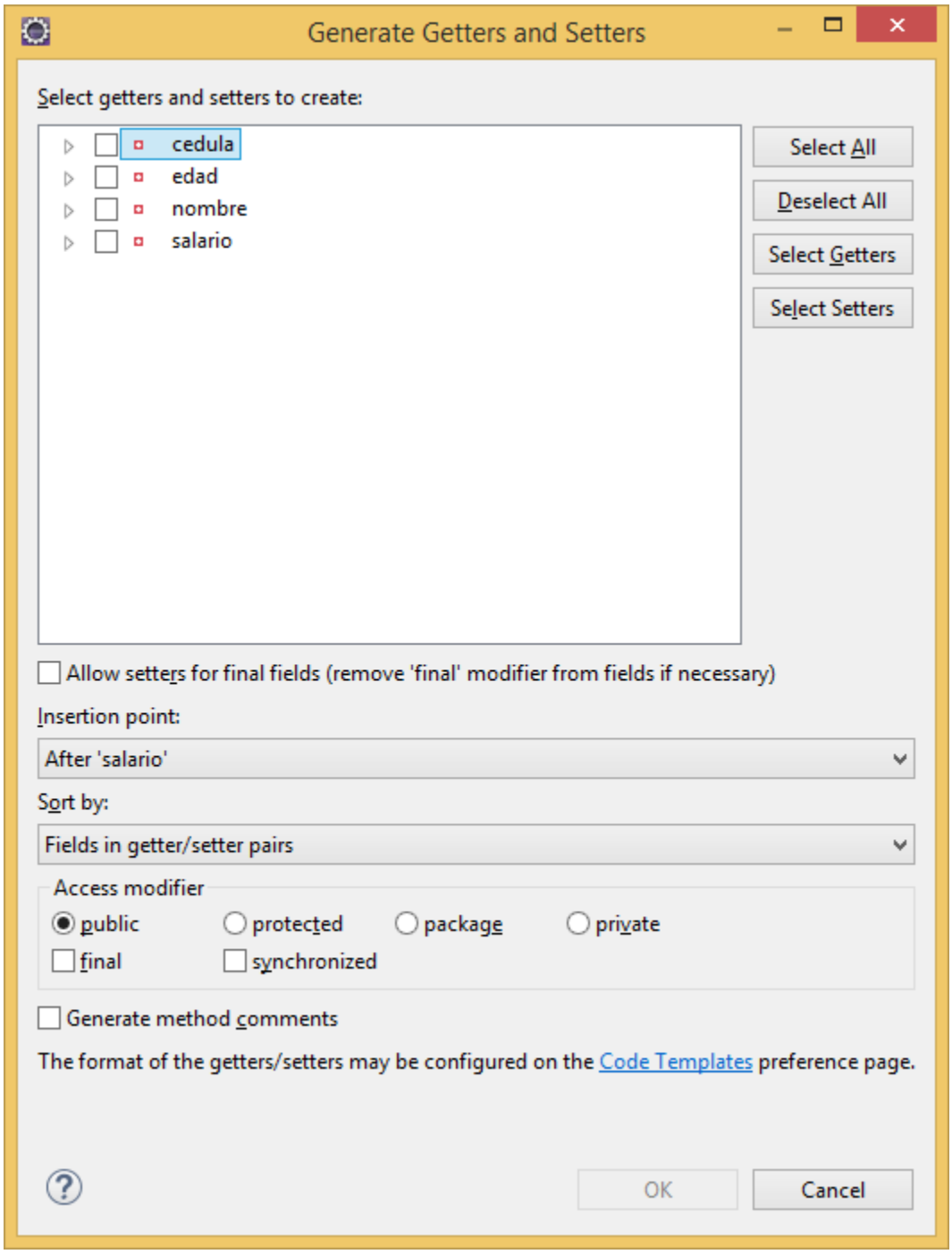
Este es el inicio típico de un **bean en java**, luego de este proceso se procede a crear o a generar los **gettes/setters**

Ubicados en cualquiera de los campos y con el botón emergente realizamos la generación de los getters / setters



Autoría Propia

Después de esto se procede a generar cuales son los campos que se desean incluir



Autoría Propia

Acá se observan algunas de ellas ya generadas

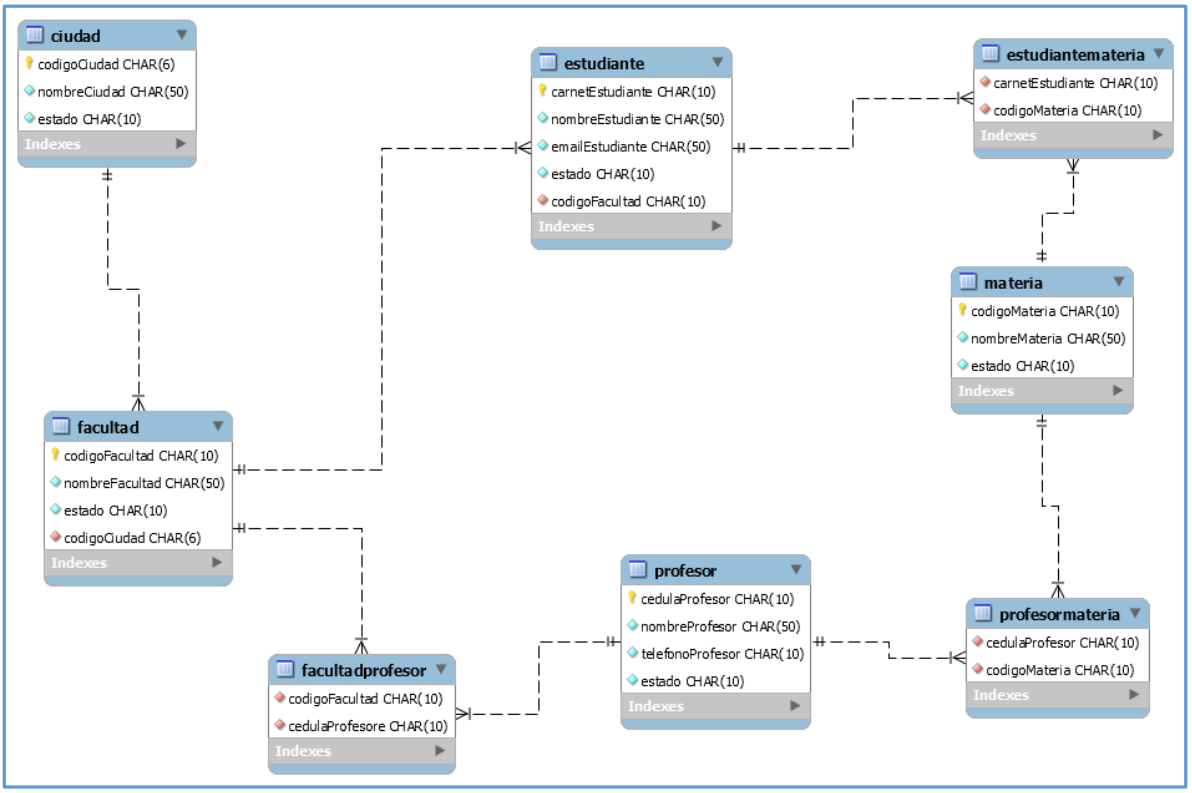
```
private String cedula;
private String nombre;
private int edad;
private float salario;
public String getCedula() {
    return cedula;
}
public void setCedula(String cedula) {
    this.cedula = cedula;
}
public String getNombre() {
    return nombre;
}
public void setNombre(String nombre) {
    this.nombre = nombre;
}
public int getEdad() {
    return edad;
}
public void setEdad(int edad) {
    this.edad = edad;
}
```

Autoría Propia

> En caso de agregar un campo nuevo, el proceso se repite, se selecciona con botón emergente y con source se realiza el proceso de getters / setters.

4.7 TEMA 6 CRUD

La creación de un **CRUD** mediante ambientes de programación siempre cumple unos mínimos requisitos, para este caso se iniciará con un MER (**Modelo Entidad Relación**)



Autoría Propia

Este MER se creará mediante MySQL y se aplicaran procedimientos almacenados para las tareas básicas de insertar, consultar, modificar, eliminar.

Posterior a esto se utilizará el conector de MySQL, este conector debe de ubicarse en la carpeta LIB de Tomcat.

PROYECTO

Se creará un proyecto Universidad

Paquetes de trabajo

Control

Modelo

Utilidad

Utilidad

Se crea el archivo Conexion.java (Class)

```java
package utilidad;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Conexion {

    public static Connection getConnection ()    {

        Connection con = null;

        try {

            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql://localhost/universidad", "root", "admin");
        }
        catch (SQLException | ClassNotFoundException e) {

            System.out.print ("" + e);
        }

        return con;

    }
}
```

Autoría Propia

CREACIÓN DE LOS JAVABEANS

Archivo Ciudad.java

```java
package modelo;

public class Ciudad {

    private String codigoCiudad;
    private String nombreCiudad;
    private String estado;

    public String getCodigoCiudad() {
        return codigoCiudad;
    }

    public void setCodigoCiudad(String codigoCiudad) {
        this.codigoCiudad = codigoCiudad;
    }

    public String getNombreCiudad() {
        return nombreCiudad;
    }

    public void setNombreCiudad(String nombreCiudad) {
        this.nombreCiudad = nombreCiudad;
    }
}
```

Autoría Propia

CREACIÓN DEL ARCHIVO DAO, CIUDADDAO.JAVA

Este archivo contiene los métodos de trabajo para insertar, consultar, modificar y eliminar

Se especificarán los métodos y luego se aplicarán según su necesidad.

Este archivo CiudadDAO.java es una clase tradicional.

INICIO DEL ARCHIVO CIUDADDAO

```java
package control;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

import modelo.Ciudad;
import utilidad.Conexion;

public class CiudadDAO {
```

Autoría Propia

MÉTODO DE INSERTAR

```java
public void insertar (Ciudad ciudad){

    Connection cnn = Conexion.getConnection();
    try{

        PreparedStatement registroCiudad = cnn.prepareStatement("call insertarCiudad (?, ?, ?)");
        registroCiudad.setString (1, ciudad.getCodigoCiudad());
        registroCiudad.setString (2, ciudad.getNombreCiudad());
        registroCiudad.setString (3, "Activo");
        registroCiudad.executeUpdate();
        System.out.print ("\nRegistro Almacenado");
    }
    catch (SQLException e) {

        System.out.print("" + e);
    }
}
```

Autoría Propia

MÉTODO DE LISTADO

```
public ArrayList <Ciudad> listado ()    {

    ArrayList <Ciudad> arrayCiudad = new ArrayList <Ciudad>();
    Connection cnn = Conexion.getConnection();
    Ciudad ciudad;
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call listarCiudad ()");
        ResultSet rs = registroCiudad.executeQuery();
        while (rs.next())   {

                ciudad = new Ciudad ();
                ciudad.setCodigoCiudad(rs.getString("codigoCiudad"));
                ciudad.setNombreCiudad(rs.getString("nombreCiudad"));
                arrayCiudad.add(ciudad);
        }
    }
    catch(SQLException e)   {

        System.out.print(""+ e);
    }

    return arrayCiudad;
}
```

Autoría Propia

> Este método tiene como particularidad la creación de un arrayList, es un arreglo que permite el java y es aplicable para Java SE, EE y Móviles, permite almacenar los datos que cumplan una condición y posteriormente hacerlo visible, en este ejemplo se llena en la instrucción arrayCiudad.add, este método adiciona elemento por elemento y luego este se retorna para ser visualizado (véase JTable de la primera unidad).

ELIMINAR

```
public static void eliminar (String codigoCiudad)    {

    Connection cnn = Conexion.getConnection();
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call eliminarCiudad (?)");
        registroCiudad.setString(1, codigoCiudad);
        registroCiudad.executeUpdate();
    }
    catch(SQLException e) {

        System.out.print("" + e);
    }
}
```

Autoría Propia

CONSULTAR

```
public Ciudad consultar (String codigoCiudad) {

    Connection cnn = Conexion.getConnection();
    Ciudad ciudad = new Ciudad();
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call consultarCiudad(?)");
        registroCiudad.setString(1, codigoCiudad);
        ResultSet rs= registroCiudad.executeQuery();
        if (rs.next())  {

            ciudad.setCodigoCiudad (rs.getString ("codigoCiudad"));
            ciudad.setNombreCiudad (rs.getString ("nombreCiudad"));
        }
    }
    catch(SQLException se)      {

        System.out.print("" + se);
    }

    return ciudad;

}
```

Autoría Propia

MODIFICAR

```
public void modificar (Ciudad ciudad) {

    Connection cnn = Conexion.getConnection();
    try {

        PreparedStatement registroCiudad = cnn.prepareStatement("call modificarCiudad(?,?)");
        registroCiudad.setString(1, ciudad.getCodigoCiudad());
        registroCiudad.setString(2, ciudad.getNombreCiudad());
        registroCiudad.executeUpdate();
    }
    catch(SQLException se) {

        System.out.print("" + se);
    }
}
```

Autoría Propia

Todos estos procesos tienen el mismo origen del CRUD creado en java SE, las sentencias y los comandos son los mismos

FACHADA

La fachada en el ambiente web cambia un poco, tiene una codificación más específica de cada tarea.

Se construirá una Fachada o Facade para la ciudad y se llamara FachadaCiudad.java, este archivo es un servlet.

```java
package control;

import java.io.IOException;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import modelo.Ciudad;

@WebServlet("/FachadaCiudad")
public class FachadaCiudad extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private CiudadDAO ciudadDAO;
    private Ciudad ciudad;

    private static final String LISTAR = "/listarCiudad.jsp";
    private static final String MODIFICAR = "/modificarCiudad.jsp";

    public FachadaCiudad() {
        super();

        ciudadDAO = new CiudadDAO();
        ciudad = new Ciudad ();
    }
```

Autoría Propia

Esta fachada instancia CiudadDAO donde se encuentran los procesos previamente creados, instancia Ciudad que contiene los Bean, posteriormente se encuentran dos líneas que tienen constantes, una de ellas LISTAR y la otra MODIFICAR, hace referencia a sus respectivas URLs y posteriormente un constructor con la instancia definitiva de las dos clases antes mencionadas.

A continuación, se desarrolla el método doget

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

    String opc = request.getParameter("opc");
    String accion = null;

    if (opc.equalsIgnoreCase("listar"))  {

        accion=LISTAR;
        request.setAttribute("listarCiudad", ciudadDAO.listado());
    }
    else if (opc.equalsIgnoreCase("eliminar"))  {

        String codigoCiudad = request.getParameter("codigoCiudad");
        ciudadDAO.eliminar(codigoCiudad);
        accion=LISTAR;
        request.setAttribute("listarCiudad", ciudadDAO.listado ());
    }
    else if (opc.equalsIgnoreCase("consultar")) {

        String codigoCiudad = request.getParameter("codigoCiudad");
        request.setAttribute("listarCiudad", ciudadDAO.consultar(codigoCiudad));
        accion=MODIFICAR;
    }
    else

        accion="modificar";

    RequestDispatcher view = request.getRequestDispatcher(accion);
    view.forward(request, response);
}
```

Autoría Propia

Este método recibe las ordenes según la tarea que se pretende desarrollar, la variable opc recibirá una palabra clave, sea para insertar, consultar, modificar o eliminar, se evalúa cual de las opciones es la correcta y se implementa.

Ejemplo

```
if (opc.equalsIgnoreCase("listar"))  {

    accion=LISTAR;
    request.setAttribute("listarCiudad", ciudadDAO.listado());
}
```

Autoría Propia

Se evalúa si la opción que se recibe es listar, a la variable acción se le asigna una constante que a su vez tiene una URL asignada y por último se invoca el método ciudadDAO.listado, este es el método del archivo DAO, este proceso se lleva a un listar Ciudad que es un arreglo de datos.

Al final de este método se encuentra un par de líneas

```
RequestDispatcher view = request.getRequestDispatcher(accion);
view.forward(request, response);
```

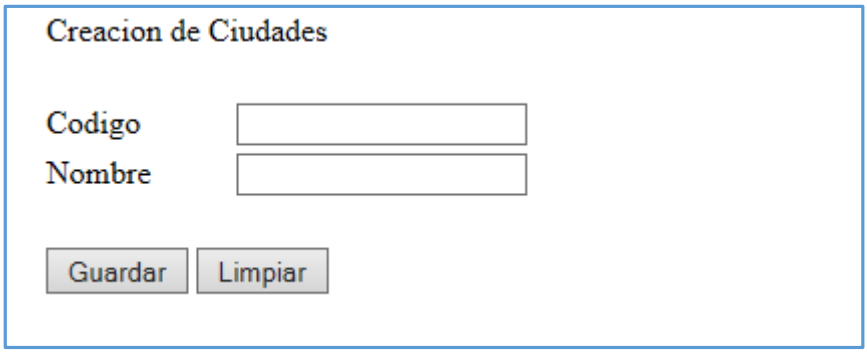
Autoría Propia

Permiten la invocación del proceso que tenga asignado la variable acción, por ejemplo, si se insertar o modifica después de hacer la tarea va a llamar el listado para corroborar que si está funcionando.

Autoría Propia

> En este método se hace recepción de todos los valores que involucran el formulario, se recibe el codigoCiudad, nombre Ciudad y se determina si va a ser una inserción o una modificación

Formulario de trabajo, Ciudad.jsp



Autoria Propia

Este formulario es tradicional a los vistos anteriormente, en el action tiene el siguiente llamado

```
<form id="form1" name="form1" method="post" action="FachadaCiudad?opc=insertar">
```

Autoria Propia

Esta primera parte es funcional, ya almacena información, se va a complementar realizando las demás tareas desde un archivo index.jsp y el listado de información.

INDEX.JPS

El archivo de índex se va a utilizar como medio para llamar las opciones que se desean. Para el ejemplo inicial solo contendrá la inserción y el listado y desde esta última ira la eliminación y modificación.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table width="200" border="0" align="center">
  <tr>
    <td>Menu Principal</td>
  </tr>
  <tr>
    <td> </td>
  </tr>
  <tr>
    <td><a href="ingresoCiudad.jsp">Ingresar Ciudad</a></td>
  </tr>
  <tr>
    <td><a href="FachadaCiudad?opc=listar">Listar Ciudad</td>
  </tr>
</table>
</body>
</html>
```

Autoria Propia

ARCHIVO LISTARCIUDAD.JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>


<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<table width="500" border="0" align="center">
<thead>
  <tr>
    <td colspan="4">Listado General de Ciudades</td>
  </tr>
  <tr>
    <td> </td>
    <td> </td>
    <td> </td>
    <td> </td>
  </tr>
```

Autoria Propia

Este archivo contiene algunas características de configuración, como las 3 primeras líneas en las que se realiza el llamado a la librería jstl, para un correcto funcionamiento del listado.

Los demás procesos son de una tabla tradicional de HTML

```
  <tr>
    <td>Codigo</td>
    <td>Nombre</td>
    <td> </td>
    <td> </td>
  </tr>
  </thead>
   <tbody>
        <c:forEach items="${listarCiudad}" var="lciudad">
    <tr>
      <td> <c:out value="${lciudad.codigoCiudad}"/></td>
      <td> <c:out value="${lciudad.nombreCiudad}"/></td>
      <td> <a href="FachadaCiudad?opc=eliminar&codigoCiudad=<c:out value="${lciudad.codigoCiudad}"/>">Eliminar</a></td>
      <td> <a href="FachadaCiudad?opc=consultar&codigoCiudad=<c:out value="${lciudad.codigoCiudad}"/>">Modificar</a></td>
    </tr>
    </c:forEach>
    </tbody>

</table>
</body>
</html>
```

Autoria Propia

La segunda parte del archivo comprende

```
<c:forEach items="${listarCiudad}" var="lciudad">
```

Autoria Propia

La utilización del arreglo o matriz de datos, esta se especifico desde la fachada Ciudad y se agrega un parámetro var como alias de esta matriz

```
<td> <c:out value="${lciudad.codigoCiudad}"/></td>
```

Autoria Propia

Se hace uso del alias y del campo que se desea visualizar (nombre asignado en la clase principal de **getters / setters**), este proceso se repita para cuantos campos deseamos mostrar en pantalla.

El listar quedaría de esta manera.

Listado General de Ciudades

Codigo	Nombre		
101010	envigado	Eliminar	Modificar
202020	manizales	Eliminar	Modificar
303030	tunja	Eliminar	Modificar

Autoria Propia

MODIFICACIÓN

El archivo de modificacionCiudad.jsp, tiene un diseño similar al de insertarCiudad.jsp, adicionando las siguientes líneas en la parte superior

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
```

Autoria Propia

Y en cada caja de texto se aplicaría lo siguiente

```
<tr>
  <td>Codigo</td>
  <td><input name="codigoCiudad" type="text" id="codigoCiudad" value="${listarCiudad.codigoCiudad}" /></td>
```

Autoria Propia

Este proceso ya es operativo para una tabla maestra.

Tenga presente que el modificar puede ser el proceso más largo, luego de listar información, se consulta y posterior a esto se almacena.

PISTAS DE APRENDIZAJE

JSP: Java Server Page, es el área Web de Java, con todas sus características y un complemento con otras herramientas que permiten un mayor dinamismo y alcance.

Lado del servidor: Existen dos tipos de desarrollo, del lado del cliente y del lado del servidor, el cliente siempre toma los recursos del pc donde se esté trabajando y el servidor siempre toma los recursos del equipo principal que brinda todos los recursos para un sistema en red.

CSS: Es un tipo de formato aplicable para archivos web, generaliza y permite mayor administración del sitio.

JS: Javascript, es un ambiente de desarrollo que permite interactuar con sitios web.

4.7.1 EJERICICIO DE APRENDIZAJE

Nombre del taller de aprendizaje: JSP	**Datos del autor del taller:** Cesar Augusto Jaramillo Henao
Escriba o plantee el caso, problema o pregunta: Para la construcción de un archivo Web con java se tienen que usar los Servlets	
Solución del taller: No, en algunos casos se podría trabajar solo con JSP, los servlets contienen una programación más nativa, y más segura	

4.7.2 TALLER DE ENTRENAMIENTO

Nombre del taller: Veterinaria	**Modalidad de trabajo:** Individual
Actividad previa: Realice completamente el CRUD de jsp y servlets visto en la última unidad, esto dará las bases necesarias para el trabajo posterior.	
Describa la actividad: Cree un proyecto que cubra todos los temas de la unidad 5, aplique formatos y validaciones y un CRUD que comprenda varias tablas incluyendo tablas maestras, referenciales e intermedias.	

5 PISTAS DE APRENDIZAJE

Recuerde que: programación web es un recurso muy amplio que maneja múltiples lenguajes y elementos

Tenga en cuenta: la programación utilizada es basada en java tanto para java SE como para Java EE

Traiga a la memoria: que la mayor parte de comando e instrucciones son los mismos en java SE que en java EE

6 GLOSARIO

- **Java SE:** Es la versión estándar de java, esta versión es la base de todo el trabajo en java
- **Java EE:** Es la versión Enterprise o empresarial, es utilizada para la programación web
- **Eclipse:** Es un IDE de desarrollo que permite facilitar algunas tareas de la programación en Java
- **Proyecto:** Es un conjunto de archivos que componen una aplicación
- **Paquete:** Es un área de trabajo que permite la clasificación de archivos o clases
- **DAO:** Es un modelo de desarrollo o patrón de diseño, standard de trabajo
- **Getters/setters:** Hacen parte de una clase principal que permite accede a la información
- **JSP:** Java Server Page, ambiente de trabajo web
- **HTML:** Lenguaje de Marcas de Hipertexto
- **JavaScript:** Lenguaje similar en estructura a Java que se puede mezclar con aplicaciones web
- **CSS:** Formatos de aplicación de aplicaciones web
- **Método:** Espacio de código que realiza una funciona especifica
- **Façade:** Patrón de diseño que administra un conjunto de clases
- **Hilo:** Herramienta de trabajo que permite realizar una tarea en procesos paralelos
- **Red:** Parte de la programación que permite que varios trabajen con elementos compartidos
- **Hibernate:** FrameWork de java que permite realizar procesos standard o web de una forma mas simplificada
- **Propertie:** Extensión de archivo que permite acceder a recursos fuera de la compilación
- **Conector:** Archivo que contiene los elementos necesarios para vincular un proyecto con un motor de bases de datos
- **Reportes:** Herramienta de visualización de información general o especifica

- Documentación: Herramienta de ayuda para el desarrollador y el control de los procesos realizados en periodos de tiempo.

7 BIBLIOGRAFÍA

- Eckel, Bruce. (2008). Piensa en Java, Madrid. ISBN: 978-84-8966-034-2
- Villalobos, Jorge (2006), Fundamentos de Programación, Bogotá. ISBN: 970-26-0846-5
- Deitel, Paul. (2012), Java, como programar, México. ISBN: 978-607-32-1150-5