

# UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM TEXCOCO  
INGENIERÍA EN COMPUTACIÓN



IMPLEMENTACIÓN DEL PATRÓN MVC EN APLICACIONES WEB CON JAVA  
MEDIANTE LA INTEGRACIÓN DE LOS FRAMEWORK HIBERNATE, SPRING Y  
PRIMEFACES

## TESINA

Que para obtener el Título de  
Ingeniero en Computación

Presenta

Ángela Ríos Luna

Directora de tesis

M. en ISC. Irene Aguilar Juárez

Revisores de tesis

Dr. en Ed. Joel Ayala de la Vega

Dr. en C. Alfonso Zarco Hidalgo

Texcoco, Estado de México, a 2 de julio de 2014

## RESUMEN

En una aplicación desarrollada con un lenguaje orientado a objetos, conviven objetos que representan el dominio del programa y objetos que modifican el estado del dominio del programa.

En 1979 Trygve M.H. Reenskaug separó las aplicaciones en dos secciones: “thing [cosa]” y “tool [herramienta]”; la herramienta es la sección de una aplicación que con sus acciones modifica el estado de la cosa.

Ese mismo año, el propio Reenskaug terminaría por separar el concepto de herramienta en “vista” y “controlador”, desacoplando de manera efectiva el código destinado a la interfaz de usuario del utilizado para realizar acciones. De hecho dos aplicaciones pueden desarrollar exactamente las mismas acciones y tener interfaces graficas diferentes.

El concepto de “Cosa” seria refinado como “Modelo” y representa a la estructura de información que es mostrada al usuario mediante la interfaz gráfica y que es modificada por la lógica de negocios invocada por los controladores.

El concepto de Vista fue necesario porque las interfaces graficas de las aplicaciones web, mostradas a través de un navegador, ni siquiera están programadas en el mismo lenguaje que el resto del programa. Sería muy complejo seguir generando programáticamente contenidos HTML al interior del método “service” de un servlet de Java.

La especialización del concepto de vista derivo en frameworks como JSP, JSF y JSTL que están desarrolladas un nivel por encima de los servlets y en los que las vistas son codificadas como archivos XHTML en los que ciertas etiquetas son remplazadas por el valor que guardan las variables antes de ser mostradas al usuario.

Del mismo modo, los frameworks facilitan que una acción en la interface, por ejemplo presionar un botón, invoquen el método que el programador desea vía la creación de un controlador, sin tener que analizar a nivel de servlets el estado que guarda la aplicación para poder inferir la acción a ejecutar.

La gran mayoría de las aplicaciones web requieren la persistencia, a menudo en una base de datos, de la estructura de la información manejada por una aplicación. El modelo de una aplicación desarrollada con JSF ya está representado como objetos, que pueden ser almacenados y recuperados de manera transparente con el API Hibernate desarrollado por Gavin King en 2001.

Las aplicaciones desarrolladas por un gran número de desarrolladores requieren su separación en módulos y capas altamente especializadas que sean codificadas con estándares internacionales y buenas prácticas cuidando de no aumentar los tiempos de desarrollo y los costos de las aplicaciones.

Cada empresa de software puede definir una arquitectura para las aplicaciones web que desarrollará e invertir en ello tiempo y recursos, o utilizar un framework como EJB (1997),

Struts (2000) o Spring Source (2008) que han sido definidos y refinados durante años por cientos de programadores alrededor del mundo y utilizados por programas que requieren los más altos estándares de calidad y seguridad.

Spring source, uno de los frameworks abordados en este trabajo, ofrece secciones pre-programadas para el acceso a la base de datos. Mensajería, seguridad, programación orientada a aspectos, etc.

Spring ofrece una estructura de aplicación web en la que el desarrollador debe llenar los espacios vacíos. Una aplicación web desarrollada con el framework Spring obtiene de facto sus cualidades de seguridad y calidad.

Más importante aún, Spring es un framework de inyección de dependencias, esto es útil cuando se unen los módulos, capas y aspectos que componen una aplicación empresarial

Este trabajo aborda la integración de Hibernate y Spring en una aplicación web desarrollada con tecnología Java Server Faces con el objetivo de servir de referencia a estudiantes y programadores facilitando su integración a un equipo de trabajo. Las interfaces de JSF son implementadas con Primefaces. Se utiliza el IDE eclipse y el servidor de aplicaciones Tomcat.

# DEDICATORIA

Doy gracias a dios por todo lo que me ha dado, gracias a mi familia ya que ustedes me han demostrado que no existe ningún obstáculo cuando se quiere algo.

# AGRADECIMIENTOS

Arturo Ríos Romero y Ana María Luna Almaraz por todo lo que me han brindado; ustedes forman parte de este trabajo ya que sin ustedes no lo hubiera logrado.

Odette Ríos Luna y Arturo Ríos Luna gracias por formar parte de mi vida, han sido parte fundamental y he compartido momentos de alegría y tristezas.

A mi directora M. en ISC. Irene Aguilar Juárez y mis revisores Doc. en Ed. Joel Ayala de la Vega y el Doc. en C. Alfonso Zarco Hidalgo les doy gracias por todo el apoyo que me brindaron y principalmente su confianza ustedes me han orientado a superarme.

Estoy muy agradecida a la vida por estar rodeadas de personas extraordinarias que me han enseñado y demostrado que todo es posible si uno lo quiere.

# CONTENIDO

Resumen .....	i
Dedicatoria.....	iv
Agradecimientos .....	v
Contenido .....	vi
Índice de Imágenes.....	1
Introducción .....	3
1.1 Problemática .....	4
1.2 Justificación.....	4
1.3 Objetivos .....	4
1.4 Metodología.....	5
Capítulo 2 Marco Teórico .....	6
2.1 Separación de una aplicación en módulos.....	6
2.2 Abstracción.....	8
2.3 Patrones de diseño.....	11
2.4 El patrón MVC .....	12
Capítulo 3 Desarrollo de una aplicación Web con Java Server Faces .....	13
Presentación del Caso de Estudio Genérico .....	13
3.1.1 Java Server Faces.....	19
3.1.2 Etapa de creación del sistema proyecto .....	24
3.1.3 Integración del primefaces .....	29
3.1.4 Implementación de JSF en la aplicación.....	33
3.2 Implementación de persistencia.....	37
3.2.1 Integración del framework hibernate. ....	37
3.2.2 Mapeo objeto relacional con hibernate .....	39
3.2.3 Data Acces Object .....	43
3.2.4 Capa de servicios .....	48
3.2.5 Facades.....	51

3.3	Integración del framework de Spring.....	53
3.3.1	Inyección de dependencias.....	54
3.3.2	Inyección de dependencias con Spring .....	54
3.3.3	Inyección de dependencias en la aplicación web .....	56
3.3.4	Integración del módulo de persistencia a la aplicación web. ....	56
3.4	Seguridad y perfiles de acceso.....	59
3.4.1	Integración de spring security .....	59
Capítulo 4 Resultados .....		62
Conclusiones.....		67
Referencias .....		68
<i>Anexo A</i> Herramientas para el desarrollo.....		70
A.1	El entorno de desarrollo eclipse.....	70
A.2	El servidor tomcat .....	74
<i>Anexo B</i> Crear un proyecto nuevo .....		78
<i>Anexo C</i> Crear un proyecto para librerías .....		81

## ÍNDICE DE IMÁGENES

Ilustración 1 Ejemplo de requerimientos para una aplicación (Elaboración propia) .....	7
Ilustración 2 Alto acoplamiento entre módulos (Elaboración propia) .....	8
Ilustración 3 Bajo acoplamiento entre módulos (Elaboración propia) .....	8
Ilustración 4 Patrones de Diseño <a href="http://www.corej2eepatterns.com/index.htm">http://www.corej2eepatterns.com/index.htm</a> .....	11
Ilustración 5 Ejemplo de un domino (Elaboración propia) .....	13
Ilustración 6 Diseño de la Base de Datos (Elaboración propia) .....	14
Ilustración 7 Diseño de navegación (Elaboración propia) .....	18
Ilustración 8 Etapas de creación de un sistema (Elaboración propia) .....	19
Ilustración 9 Vista que invoca métodos del controlador (Elaboración propia) .....	20
Ilustración 10 Estructura de carpetas (Elaboración propia) .....	25
Ilustración 11 Java Build Path .....	26
Ilustración 12 Carpetas .....	27
Ilustración 13 Carpetas .....	27
Ilustración 14 Módulos .....	28
Ilustración 15 Java Build Path .....	28
Ilustración 16 Componentes de primefaces .....	29
Ilustración 17 Componente de edición de primefaces .....	30
Ilustración 18 Componentes de primefaces .....	31
Ilustración 19 Temas de primefaces .....	32
Ilustración 20 Temas de jQuery .....	32
Ilustración 21 Base de datos del Estudio Genérico .....	40
Ilustración 22 lógica de negocios .....	49
Ilustración 23 Acceso a la Base de Datos .....	50
Ilustración 24 Ejemplo de Facades .....	51
Ilustración 25 Diagrama general del módulo de acceso a la base de datos .....	52
Ilustración 26( <a href="http://docs.spring.io/spring/docs/2.0.x/reference/introduction.html">http://docs.spring.io/spring/docs/2.0.x/reference/introduction.html</a> ) .....	53
Ilustración 27 Vista de autenticación .....	60
Ilustración 28 Página de inicio del sistema .....	62
Ilustración 29 Vista de lista de contactos .....	63
Ilustración 30 Vista de alta de contacto .....	64
Ilustración 31 Vista de lista de proyectos .....	65
Ilustración 32 Alta de un proyecto .....	65
Ilustración 33 Lista de asignaciones .....	66
Ilustración 34 Alta de una tarea .....	66
Ilustración 35 herramienta que permite la búsqueda de un plugin .....	70
Ilustración 36 Descarga de Spring .....	71
Ilustración 37 Instalando SpringSource .....	72
Ilustración 38 Instalando SpringSource .....	72
Ilustración 39 Path de Instalación .....	73



Ilustración 40 Paquetes de instalación .....	73
Ilustración 41 seleccionar el JDK .....	74
Ilustración 42 Zip de Tomcat.....	75
Ilustración 43 Configuración tomcat.....	75
Ilustración 44 Alta de variable de entorno .....	76
Ilustración 45 compilados del tomcat .....	76
Ilustración 46 Página principal del tomcat .....	77
Ilustración 47 Creación de un proyecto Web (Elaboración propia) .....	78
Ilustración 48 Nombre del proyecto Web (Elaboración propia) .....	79
Ilustración 49 Configuración del proyecto (Elaboración propia) .....	79
Ilustración 50 Configuración del proyecto (Elaboración propia) .....	80
Ilustración 51 Vista del código del proyecto (Elaboración propia) .....	80
Ilustración 52 Creación de un proyecto para librerías (Elaboración propia).....	81
Ilustración 53 Creación de un proyecto para librerías (Elaboración propia).....	81
Ilustración 54 Creación de un proyecto para librerías (Elaboración propia).....	82
Ilustración 55 Configuración de librerías .....	85
Ilustración 56 Deployment Assembly .....	86

# INTRODUCCIÓN

En el desarrollo de una aplicación web multicapa es necesario garantizar que las secciones asignadas a cada desarrollador sean codificadas de manera profesional para que puedan acoplarse de manera sencilla y transparente.

Es importante definir la arquitectura con la que todos los integrantes de un equipo desarrollarán una aplicación, pues permitirá que sea más eficiente su trabajo.

La arquitectura de las aplicaciones web ha evolucionado a la par que surgen nuevas tecnologías. Las arquitecturas modernas de aplicaciones separan en capas los niveles de abstracción de una aplicación y en módulos su funcionamiento.

El desarrollo de aplicaciones web es más sencillo con el uso de frameworks y bibliotecas que disminuyan la codificación. Cada día nacen nuevas bibliotecas y tecnologías, la velocidad y éxito de su implementación depende del nivel en el que los programadores entiendan los patrones de diseño modernos.

El uso de un Framework mejora sustancialmente el trabajo de los programadores, ya que dedicarán menos tiempo de desarrollo en la estructura de una aplicación y más en la lógica de negocio.

El objetivo de este trabajo es el desarrollo de un caso genérico a manera de manual que dé a conocer a los desarrolladores web principiantes una forma de integrar tecnologías de punta que actualmente son estándares en la industria.

En este trabajo se explica el funcionamiento del patrón MVC y de las capas típicas de una aplicación WEB de Java Server Faces.

Java Server Faces es una tecnología para aplicaciones web desarrollada por Sun Microsystems en 2004 y que implementa el lenguaje java. Su versión actual es la 2.1 y para el desarrollo de este manual se implementará vía el framework PrimeFaces.

Los tópicos de persistencia se abordan con el framework hibernate, para finalmente integrar los distintos módulos con el API de inyección de dependencias de Spring Source.

El API de persistencia Java (JPA) le permite al programador almacenar estructuras de datos en bases relacionales de forma transparente. El uso de API's de persistencia es cada vez común.

Hibérnate es el API de persistencia más reconocido lo que facilita el acceso a documentación y su aprendizaje. Soporta las bases de datos más usadas y al mismo tiempo puede migrarse de una base de datos a otra fácil y rápidamente.

# Capítulo 1

## PLANTEAMIENTO DEL ESTUDIO

### 1.1 Problemática

La integración de un programador a un equipo de trabajo podría fracasar si su formación no abarcó la implementación de patrones de diseño y buenas prácticas pues son aspectos vitales para aplicaciones altas en complejidad y numero de colaboradores.

Para una empresa, no contar con una estructura arquitectónica y de ingeniería preestablecida dificulta la integración de personal, y puede significar la diferencia entre el éxito o el fracaso del desarrollo colaborativo de software.

Por otro lado, no utilizar API's o Frameworks aumenta la codificación, el tiempo de desarrollo y el uso de recursos, impactando directamente al costo de un producto y minimizando las ganancias de la empresa.

### 1.2 Justificación

La divulgación a manera de manual de los principios arquitectónicos y de ingeniería utilizados en la industria de software será de gran utilidad para la integración de nuevos colaboradores a un equipo de trabajo y para la formación académica de alumnos interesados en el desarrollo de aplicaciones web

### 1.3 Objetivos

Objetivos generales:

Explicar la integración de los framework Hibernate 4.1, Spring 3.1 y Primefaces 3.5 para aplicaciones web que utilizan Java Server Faces, para guiar el desarrollo de aplicaciones web de programadores principiantes, con base en un estudio de casos genérico

Objetivos específicos:

- Analizar un estudio de caso genérico web que tenga acceso a bases de datos.
- Diseñar la arquitectura básica para el caso de estudio genérico.
- Generar un manual de integración que describa la configuración de los distintos frameworks.

## 1.4 Metodología

El desarrollo fácilmente puede dividirse en módulos y cada uno de estos desarrollarse por separado. El diseño de la aplicación y de su funcionamiento es parte importante para procurar que los distintos módulos se integren finalmente con éxito, para ello se utilizarían diagramas de UML y paquetería apropiada.

El equipo de desarrollo tiene el compromiso de generar aplicaciones estandarizadas, que tenga un fácil mantenimiento y que pueda reemplazarse partes del código sin que las dependencias de bibliotecas propietarias lo limiten.

Módulos a Implementar:

- Módulo de acceso a la base de datos.
- Diseño de Interfaz web.
- Inyección de dependencias.

Artefactos:

- Archivos de configuración.
- Manual de integración.
- Diseño de la base de datos.

# Capítulo 2

## MARCO TEÓRICO

Las aplicaciones web son aquellas que son visualizadas por el cliente en un navegador. Son también conocidas como aplicaciones empresariales y tienen un amplio uso en escenarios de recursos compartidos por red.

Mientras que las aplicaciones de escritorio deben instalarse en cada equipo, las aplicaciones web solo se instalan en el servidor y cualquier problema es resuelto solo en el servidor; lo que es particularmente útil cuando se utilizan en empresas o instituciones con varias sucursales.

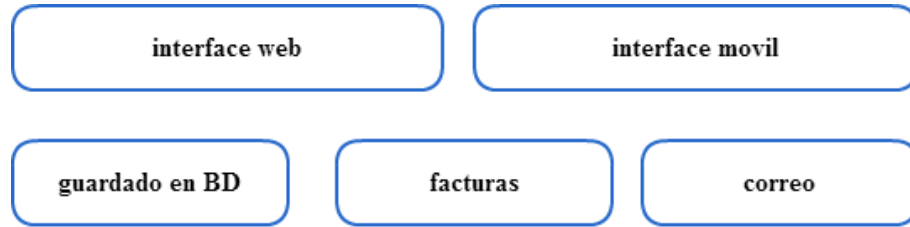
Las aplicaciones web permiten el acceso concurrente a bases de datos comunes. Otra ventaja es que las empresas no necesitan gastar en equipos costosos para los usuarios finales pues el procesamiento lo realiza el servidor.

### 2.1 Separación de una aplicación en módulos.

Cuando comienza el ciclo de diseño de un programa, los requerimientos del cliente serán traducidos a tareas de programación. Una aplicación puede separarse en secciones que reúnan tareas comunes o que cumplan objetivos específicos. Los módulos así definidos permiten dividir la complejidad de un sistema y facilitan la asignación del trabajo a equipos.

Identificar los módulos que compondrán una aplicación incluso facilita la predicción de costos, el tiempo de desarrollo y el número de programadores necesarios. También simplifican el control del avance de la aplicación y la entrega de avances.

Por ejemplo, los objetivos de una aplicación pueden ser: guardar información en una base de datos, generar facturas, enviar correos, realizar una interfaz web, generar una interfaz de la misma aplicación pero para celulares, etc. Todos estos requerimientos se traducen a módulos asignables a distintos programadores (véase en la ilustración 1).



*Ilustración 1 Ejemplo de requerimientos para una aplicación (Elaboración propia)*

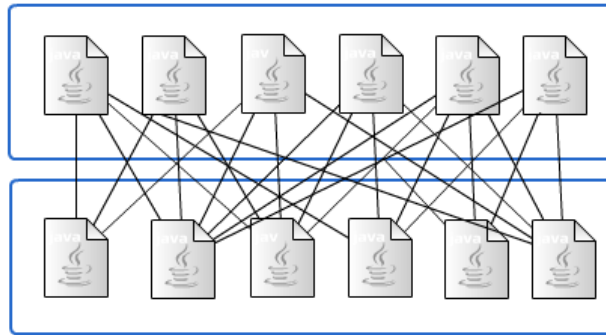
Un módulo debe tener una alta cohesión, es decir, ser autónomo y contener dentro de él todo lo necesario para validar su funcionamiento de forma independiente, aun cuando el resto de la aplicación no haya sido programada.

Las secciones de un programa deben presentar un bajo acoplamiento, es decir, no todas las clases que contiene un módulo deben ser accesibles, pues mientras más puntos de acceso tenga, más difícil será remplazarlo por otro módulo de funcionamiento similar. Es una buena práctica definir interfaces dedicadas exclusivamente a ser puntos de acceso a las funciones de un módulo, de tal manera que pueda ser intercambiable por cualquier otro que satisfaga los requerimientos que expresa esa única interfaz.

Por ejemplo, en lugar de desarrollar un módulo de envío de calificaciones por correo y un módulo de envío de invitaciones por correo, debe desarrollarse un módulo de propósito general que envíe correos y que considere un formato editable. El delegar la responsabilidad del formato del mensaje favorece su reutilización. Al mismo tiempo el programador puede realizar las pruebas unitarias necesarias para descartar errores independientemente del objetivo que se le otorgue en su implementación.

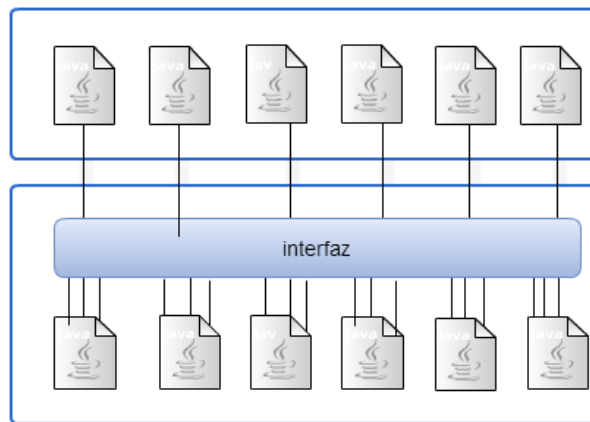
También podría pasar que un módulo diseñado para guardar información en una base de datos sea necesario remplazarlo por otro de un gestor de bases de datos distinto. En ese caso ambos módulos deben cumplir los mismos objetivos. Sin embargo, si las funciones que cumplía el primer módulo se encontraran distribuidas de forma desordenada y en muchos archivos, será difícil para el programador encargado de realizar la nueva versión asegurar que el nuevo diseño no dejó escapar ninguna funcionalidad de su predecesor.

Con alto acoplamiento es difícil realizar una nueva versión, es difícil saber si se cumplen todos los objetivos, difícil de realizar el test de cada funcionalidad (véase en la ilustración 2).



*Ilustración 2 Alto acoplamiento entre módulos (Elaboración propia)*

Con bajo acoplamiento es más fácil realizar una nueva versión, basta con cumplir los requerimientos de la interfaz y el test es más sencillo (véase en la ilustración 3).



*Ilustración 3 Bajo acoplamiento entre módulos (Elaboración propia)*

## 2.2 Abstracción

El termino abstracción se refiere a la identificación de las propiedades de un objeto para centrarse en el estudio de su esencia en un nivel más elevado.

En el lenguaje Java existen dos estructuras que describen el comportamiento y cualidades de un objeto pero que no lo implementan:

- Las clases abstractas en los que algunos métodos están implementados y otros no.

- Las interfaces en las que ningún método se encuentra implementado y son por tanto completamente abstractas, porque solo describen el funcionamiento de la clase.

Supóngase que se tiene una clase Triangulo con un método obtenerArea:

<b>Triangulo</b>
- Altura:float
- Base :float
+ ObtenerArea() :float

Y una clase Cuadrado con el mismo método:

<b>Cuadrado</b>
<u>Altura:float</u>
- <u>Base :float</u>
<u>+ ObtenerArea()</u>
<u>:float</u>

La clase Test para la clase Triangulo se codifica de la siguiente manera:

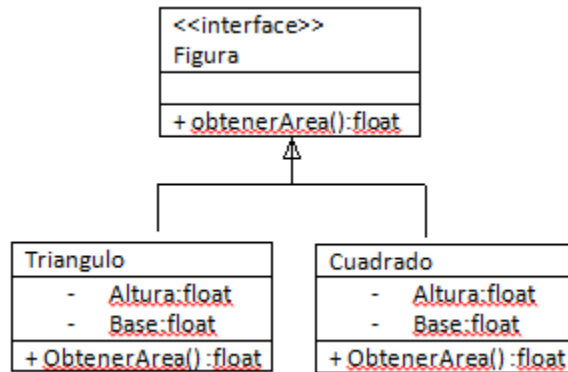
```
Class TestTriangulo{  
    Triangulo triangulo = new Triangulo();  
    Public void run(){  
        System.out.println(triangulo.obtenerArea());  
    }  
}
```

Para cada figura geométrica debe codificarse la clase test de nuevo, y además la clase Test depende de que la figura respectiva ya se encuentre programada:

```
Class TestCuadrado{  
    Cuadrado cuadrado= new Cuadrado();  
    Public void run(){  
        System.out.println(cuadrado.obtenerArea());  
    }  
}
```

Existe otra perspectiva para abordar este conjunto de clases, que abstrae la lógica de las figuras geométricas en una interface.





De tal manera que la clase *Test* puede invocar al método obtener área de la interface *Figura* independientemente de la figura geométrica que lo implemente. En el código siguiente ya no se mencionan ni la clase *Cuadrado* ni la clase *Triangulo*:

```
Class Test{
Figura figura;
Public void run(){
System.out.println(figura.obtenerArea());
}
Public void setFigura(Figura figura){
This.figura=figura;
}
}
```

Por ejemplo, para el cuadrado la clase test se usaría:

```
Test test = new Test();
Test.setFigura(new Cuadrado());
Test.run();
```

Y para la clase *Triangulo*:

```
Test test = new Test();
Test.setFigura(new Triangulo());
Test.run();
```

En este caso la abstracción ofrece varias ventajas:

- Reutilización, ya que solo se necesita una clase *Test* para todas las figuras
- Bajo acoplamiento, que se traduce en que el programador encargado de la clase *Test* puede terminar el trabajo que le fue asignado sin depender de que existan las figuras geométricas [asignadas a otras personas].

## 2.3 Patrones de diseño

El patrón de diseño es el esqueleto para la solución de problemas comunes en el desarrollo de software.

Estos son la base para las búsquedas de soluciones a problemas comunes en el desarrollo de software. Otra de sus características es que deben ser reutilizables (véase en la ilustración 4).



*Ilustración 4 Patrones de Diseño* <http://www.corej2eepatterns.com/index.htm>

## Clasificación de los patrones:

### Tipos

- Patrones de creación de objetos.
- Patrones de comportamiento que coordinan la interacción funcional entre objetos.
- Patrones estructurales que gestionan relaciones estáticas y estructurales entre objetos.
- Patrones de sistema que gestionan la interacción a nivel de sistema.

### Niveles

- Clase única: el patrón se aplica a una única clase.
- Componente: el patrón implica un grupo de clases.
- Arquitectónico: el patrón se utiliza para coordinar las acciones.

Los patrones de diseño representan un gran evolución importante en la abstracción y reutilización del software, la abstracción representa la forma que tienen los desarrolladores para resolver problemas complejos dividiéndolos en otros más simples. La reutilización es igual vital para el desarrollo.

## 2.4 El patrón MVC

Es un patrón de diseño que especifica cómo debe de ser estructurada una aplicación, las capas que van a componer son los siguientes:

- **Modelo:** Es el objeto de aplicación.
- **Vista:** Es su representación de pantalla.
- **Controlador:** Define en modo en que la interfaz reacciona a la entrada del usuario.

Los diseños de interfaz de usuario tendían a agrupar estos objetos en uno solo. MVC los separa para incrementar la flexibilidad y reutilización.

El MVC desacopla las vistas de los modelos estableciendo entre ellos un protocolo de suscripción/notificación. Una vista debe asegurarse de que su apariencia refleja el estado del modelo. Cada vez cambian los datos del modelo, este se encarga de avisar a las vistas dependientes de él. Como respuesta a dicha notificación, cada vista tiene la oportunidad de utilizarse a sí misma. (Gamma, Johnson, & Vlissides, 2003)

## Capítulo 3

# DESARROLLO DE UNA APLICACIÓN WEB CON JAVA SERVER FACES

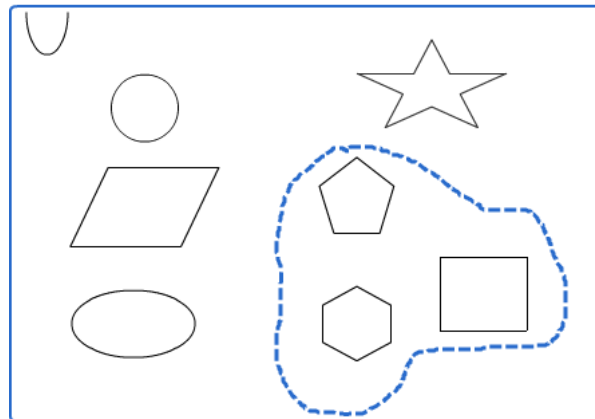
### Presentación del Caso de Estudio Genérico

#### Requerimientos del Sistema:

Sistema que registra la realización de tareas por parte del personal. El sistema permitirá registrar tareas agrupadas por proyecto. Las tareas serán asignadas a los empleados que registrarán la fecha de inicio y término, también almacenará la información personal de los empleados.

#### Dominio de la Aplicación

No solo las aplicaciones, en general cualquier problema a resolver tiene un dominio que se refiere a su campo de acción. Por ejemplo, obtener el área del cuadrado, pentágono y hexágono es un problema matemático que se restringe al dominio de los polígonos regulares y no al universo completo de las figuras geométricas (véase en la ilustración 5).



*Ilustración 5 Ejemplo de un domino (Elaboración propia)*

Del mismo modo cuando se desarrolla una aplicación su dominio es modelado a partir de los requerimientos del cliente, primero como campo semántico, después con el modelo relacional (véase en la ilustración 6).

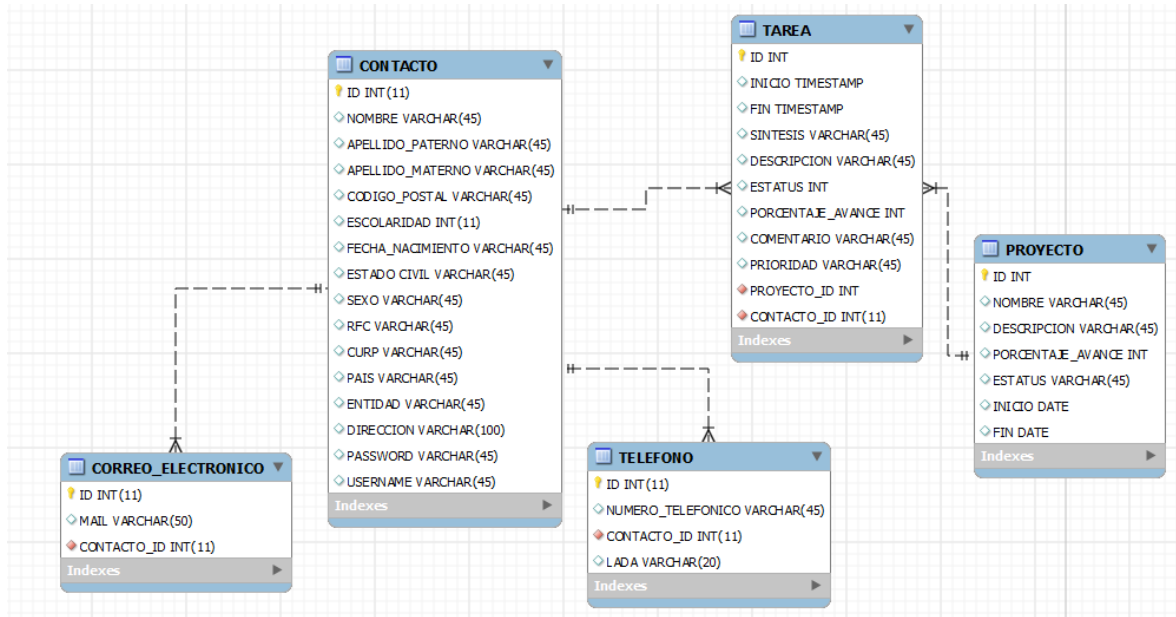


Ilustración 6 Diseño de la Base de Datos (Elaboración propia)

A partir de la base de datos el dominio de la aplicación se codifica como clases que corresponden a cada una de las tablas.

El dominio de la aplicación aquí presentada se restringe a la administración de proyectos y contiene a las entidades Contacto, Tarea, Proyecto.

El primer paso para desarrollar una aplicación de cualquier tipo, es definir aquellas clases que serán la unidad de información manipulada por el programa. Las clases que representan el dominio de una aplicación son POJO's (Plain Old Java Object es similar a un Java bean, con propiedades accesibles mediante los métodos setter y getter) denominados DTO (objeto de transferencia de datos), pues son los paquetes de datos enviados de la interface a la base de datos.

Diagrama de clases del dominio de la aplicación:

Proyecto
<ul style="list-style-type: none"> <li>- <b>idProyecto</b>:Integer</li> <li>- <b>nombre</b>:String</li> <li>- <b>descripción</b>:String</li> <li>▪ <b>estatus</b>:String</li> <li>▪ <b>inicio</b>:Date</li> <li>▪ <b>Fin</b>:Date</li> <li>▪ <b>PorcentajeAvance</b>:Integer</li> </ul>
<< constructores >>
+ Proyecto()
<< métodos >>
+ getters & setters

Telefono
<ul style="list-style-type: none"><li>- <b>idTelefono</b>:Integer</li><li>- <b>lada</b>:String</li><li>- <b>numeroTelefonico</b>:String<ul style="list-style-type: none"><li>▪ <b>contacto</b>:Contacto</li></ul></li></ul>
<< constructores >>
+ Telefono()
<< métodos >>
+ getters & setters

CorreoElectronico
<ul style="list-style-type: none"><li>- <b><u>idCorreoElectronico</u></b>:Integer</li><li>- <b><u>mail</u></b>:String<ul style="list-style-type: none"><li>▪ <b><u>contacto</u></b>:Contacto</li></ul></li></ul>
<< constructores >>
+ <u>CorreoElectronico()</u>
<< métodos >>
+ <u>getters &amp; setters</u>

Tarea
▪ <b>idTarea</b> :Integer
▪ <b>comentario</b> :String
▪ <b>descripción</b> :String
▪ <b>estatus</b> :Integer
▪ <b>fin</b> :Date
▪ <b>inicio</b> :Date
▪ <b>porcentajeAvance</b> :Integer
▪ <b>prioridad</b> :Integer
▪ <b>síntesis</b> :String
▪ <b>proyecto</b> :Proyecto
▪ <b>contacto</b> :Contacto
<<constructores>>
Tarea()
<< métodos>>
+ getters
+ setters

Contacto
▪ <b>idContacto</b> :Integer
▪ <b>apellidoPaterno</b> :String
▪ <b>apellidoMaterno</b> :String
▪ <b>nombre</b> :String
▪ <b>codigoPostal</b> :Integer
▪ <b>curp</b> :String
▪ <b>dirección</b> :String
▪ <b>entidad</b> :String
▪ <b>escolaridad</b> :Integer
▪ <b>estadoCivil</b> :String
▪ <b>fechaNacimiento</b> :Date
▪ <b>país</b> :String
▪ <b>rfc</b> :String
▪ <b>sexo</b> :String
▪ <b>username</b> :String
▪ <b>password</b> :String
▪ <b>tareas</b> :List<Tarea>
▪ <b>teléfonos</b> : List<Telefono>
▪ <b>correoElectronicos</b> :List<CorreoElectronico>
<< constructores>>
Contacto()
<< métodos>>
+ getters
+ setters

Para representar una relación *uno a varios* se utiliza una estructura List o Set. Por ejemplo, el siguiente código crea un objeto proyecto que incluye varias tareas;

```
Proyecto proyecto1 = new Proyecto("Fiesta");  
Tarea tarea1 = new Tarea("Comprar pastel");  
Tarea tarea2 = new Tarea("Comprar piñata"); ...  
Proyecto1.getTareas.add(tarea1);  
Proyecto1.getTareas.add(tarea2)
```

Las clases DTO también contienen el método hashCode y Equals:

- **hashCode:** es un método que genera una clave que define la posición en la que un elemento es insertado en una lista.
- **Equals:** es un método que regresa un booleano especificando si un objeto es similar al evaluado, se utiliza para saber si ya está contenido en una lista.

### **Herramientas para el sistema:**

El sistema estará implementado con el entorno de eclipse y el servidor [véase anexo A]

### **Descripción del sistema:**

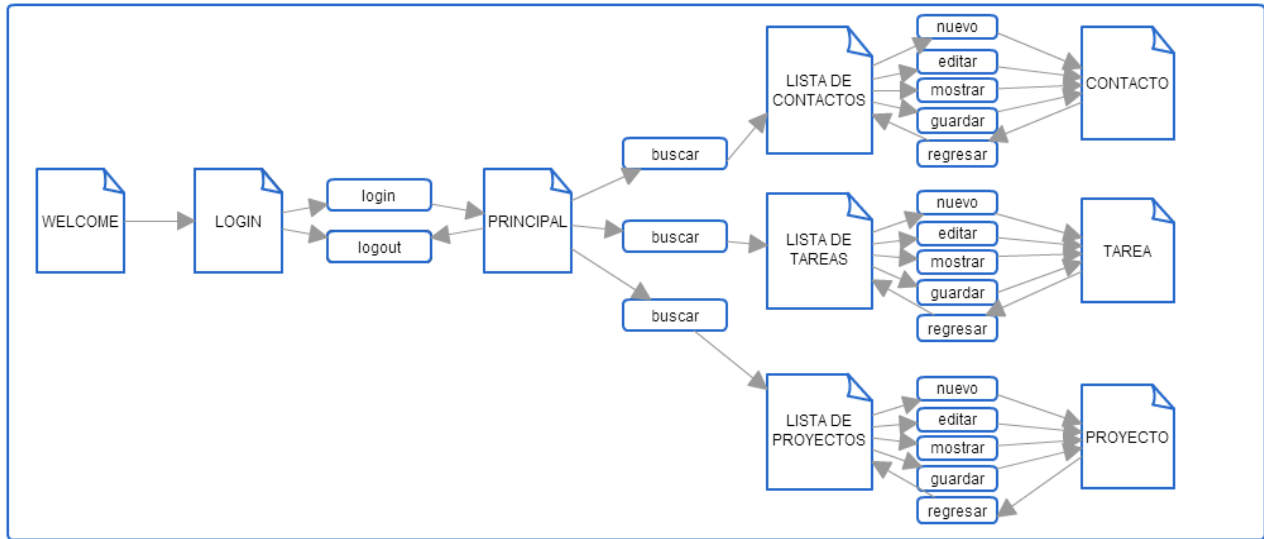
El sistema permite administrar:

- Proyectos
- Tareas
- Contactos



## Diseño de navegación

En base a los requerimientos del sistema se muestra la navegación (véase en la ilustración 7).



*Ilustración 7 Diseño de navegación (Elaboración propia)*

## Roles de Acceso

Los roles que tiene el sistema son los siguientes:

Perfil administrador:

- Administra los proyectos
- Asigna las tareas a los programadores
- Administra el catálogo de programadores

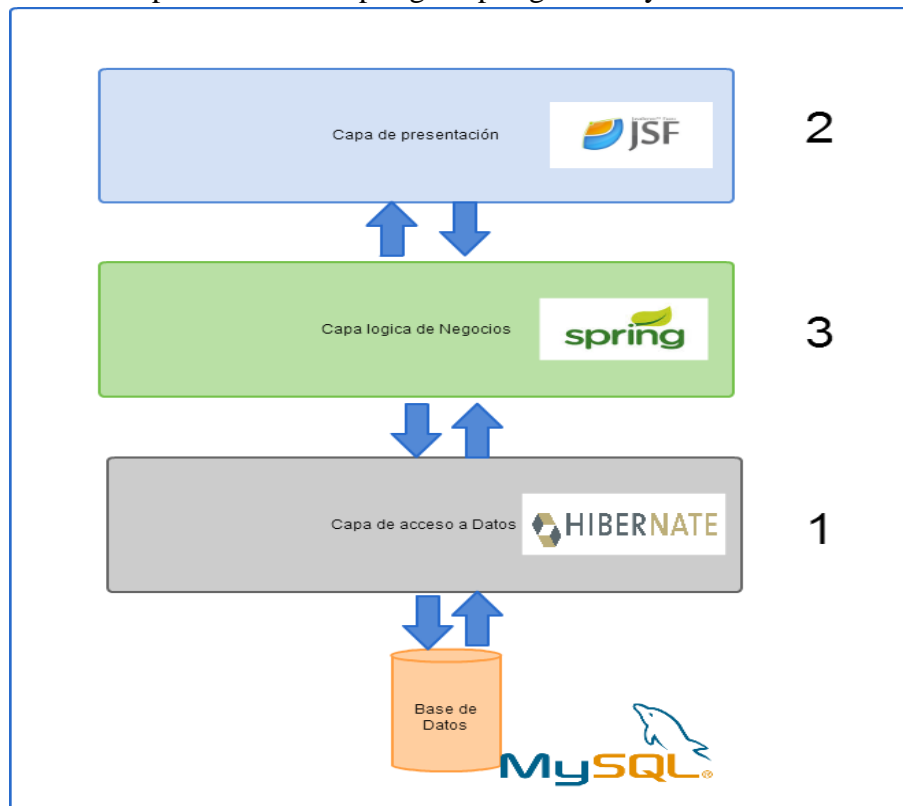
Perfil programador:

- Visualiza la lista de tareas asignadas

## Etapas de Desarrollo

En la ilustración 8 se muestra la secuencia de creación de las etapas del sistema en un entorno de producción:

1. Creación del módulo de acceso a la base de datos con Hibernate.
2. Creación de una aplicación web con JSF.
3. Inyección de dependencias de Spring + Spring Security.



*Ilustración 8 Etapas de creación de un sistema (Elaboración propia)*

El desarrollo de las interfaces gráficas y del guardado en la base de datos puede desarrollarse de manera paralela en tanto que la estructura de datos se halle modelada. En este trabajo, con fines didácticos, se aborda primero el uso de Java Server Faces por ser el tema principal el modelo MVC, una aplicación web que no almacene la información en una base de datos sigue siendo una aplicación MVC.

En primer lugar se explicará el uso de JSF implementándolo con Primefaces, para después explicar el código XHTML de algunas de las ventanas.

### 3.1.1 Java Server Faces

En el modelo vista controlador se detectan tres secciones, las vistas que se refieren a las interfaces graficas normalmente desarrolladas en código HTML, los controladores que se

encargan de direccionar las solicitudes y el modelo que representa el estado en un momento particular de la estructura de datos de una aplicación.

Sin embargo no siempre hay una diferenciación tan marcada entre estos tres conceptos.

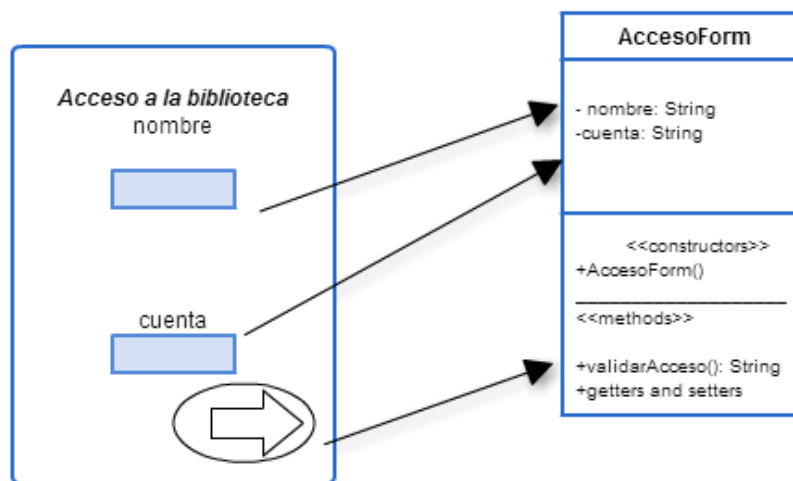
Por ejemplo en la tecnología JSP los objetos de java involucrados en el procesamiento de las solicitudes se encuentran embebidos en el código HTML destinado al diseño de las interfaces; esto implica que aquella persona destinada a desarrollar las interfaces graficas no solo tenga un dominio completo de HTML y hojas de estilo sino también de java y persistencia de objetos; lo mismo sucede en otros lenguajes basados en scripts como PHP.

Incluso el no tener una diferencia marcada entre las tecnologías y secciones de una aplicación incide en el tiempo de capacitación de un empleado, pues no puede empezar a trabajar sin especializarse en todos los aspectos de la aplicación. También afecta el tiempo que se necesita para adecuar una aplicación si un defecto aparece n veces a lo largo de los scripts.

Con JSF sin embargo hay una separación clara entre el diseño de las interfaces con XHTML y CSS, las clases de java que reciben las solicitudes y el resto del programa.

Aunque en una aplicación JSF se involucran otros aspectos, puede decirse que hay tres tipos de documento principales en una aplicación JSF (véase en la ilustración 9).

- 1.- Las interfaces codificadas en XHTML
- 2.- Clases java que ejecutan las peticiones de las vistas (Controladores).
- 3.- Clases java que representan el dominio del programa (Modelo).



*Ilustración 9 Vista que invoca métodos del controlador (Elaboración propia)*

Los métodos invocados por los botones de las vistas retornan la URL de la interfaz a la que debe direccionarse la navegación. Una cadena vacía hará que el usuario permanezca en la misma interfaz.

## Controladores

Los controladores son las clases que reciben las peticiones de las interfaces. Estas clases deben darse de alta en el archivo faces-config.xml presente en todas las aplicaciones JSF:

<u>mx.uaemex.proyecto.beans.LoginController</u>
• <u>Username:String</u>
• <u>Password:String</u>
<u>&lt;&lt; constructor &gt;&gt;</u>
<u>LoginController()</u>
<u>&lt;&lt; métodos&gt;&gt;</u>
<u>+ login():String</u>
<u>+ logout():String</u>
<u>+ getUsername():String</u>
<u>+ setUsername(String):void</u>
<u>+ getPassword():String</u>
<u>+ setPassword(String):void</u>

## Archivo WebContent/WEB-INF/Faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee;
  http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd"
  version="2.1">
  <managed-bean>
    <managed-bean-name>LoginController</managed-bean-name>
    <managed-bean-class>
      mx.uaemex.proyecto.beans.LoginController
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

Cuando la aplicación JSF se ejecute, esta creará un objeto llamado LoginController del tipo mx.uaemex.proyecto.beans.LoginController. El parámetro *managed-bean-scope* establece que se creará un objeto de este tipo por cada usuario que ocupe la aplicación.

Tipos de scope (ambito, vida de los beans):

- Request: el objeto se crea por cada petición de las interfaces graficas

- Session: el objeto se crea por cada sesión de usuario
- Application: el objeto se crea una sola vez para toda la aplicación

Cuando el objeto es dado de alta, las interfaces graficas pueden utilizarlo con la siguiente nomenclatura:

```
"#{Controlador.metodo}"  
"#{Controlador.variable}"
```

Por ejemplo, el método login de la clase LoginController se invocaría con

```
"#{LoginController.login}"
```

### Botón en Login.xhtml

```
<h:commandButton  
    id="btnLoginId"  
    value="Login"  
    action="#{LoginController.login}" >  
</h:commandButton> ...
```

La cadena capturada en la caja de texto para nombre y password, se envían a las variables nombre y password del controlador automáticamente, solo hay que asociar los componentes de la interface gráfica con los campos correspondientes del objeto.

### Cajas de texto en Login.xhtml

```
... usuario ...  
<h:inputText  
    id="userName"  
    value="#{LoginBean.userName}" >  
</h:inputText>  
... password ...  
<h:inputSecret  
    id="password"  
    value="#{LoginBean.password}" >  
</h:inputSecret>
```

Los tres componentes deben ponerse entre etiquetas `<h:form>` y `</h:form>` que serán procesadas cuando un botón se accione. Puede haber varias secciones form en una misma interface gráfica.

Cuando el método login sea ejecutado, este validará el nombre de usuario y contraseña y retornará una cadena de texto con la URL a la que se dirigirá la navegación.

Más adelante, en la sección de Spring Security se ahonda en el tema de sesiones de usuario.

### Sintaxis del lenguaje de expresiones JSF

La sintaxis JSF es similar a la utilizada en el lenguaje de expresión JSP 2.1. Sin embargo, con las siguientes excepciones. El Delimitador, debe ser el `# {y}`.

Ejemplos de expresión:

- `# {Foo}`

- # {} Foo.bar
- # {} Foo.bar.baz
- # {Foo [bar]}
- # {Foo ["bar"]}
- # {Foo [3]}
- # {Foo [3]. Bar}
- # {Foo.bar [3]}
- # {Foo.bar == "Hello World"}
- # {(Foo.bar) / 5 \* 3}
- Oferta válida desde # {offer.validFromDate} a {} offer.validToDate

El valor debe ser un identificador que coincide con el nombre del método en el objeto en la expresión.

Operadores:

- Aritmética: + , - (binario), \* , / y div , % y mod , - (unario)
- Lógico: y , && , o , | , no , !
- Relación: == , EQ , != , ne , < , lt , > , gt , GE , >= , le , <=
- Condiciones: ? A, B: C . B o C Basándose en los resultados de la evaluación de A.

Ejemplo: mostrar la calificación de un alumno en un outputLabel de color rojo si es menor a 6.

```
<h:outputLabel
value="calificación : #{AlumnoBean.alumno.calificacion}"
style="#{AlumnoBean.alumno.calificacion lt 6 ? 'redStyle' : null}"
/>
```

## Validadores

JSF puede validar la información capturada en un componente. Ejemplo que valida que el texto escrito sea un correo electrónico:

```
<h:inputText value="#{AlumnoBean.alumno.mail}" >
<f:validateRegex
pattern="[\w\.-]*[a-zA-Z0-9_]@[\w\.-]*[a-zA-Z0-9]\.[a-zA-Z][a-zA-Z\.-]*[a-zA-Z]"
/>
</h:inputText>
```

Otros validadores:

- f:validateBean
- f:validateDoubleRange
- f:validateLength
- f:validateLongRange
- f:validateRegex
- f:validateRequired

También pueden hacerse validadores propios, sobrescribiendo la clase: javax.faces.validator.Validator y dando de alta el validador en el faces-config.xml.

## ValidatorNumerosRomanos.java

```
public class ValidatorNumerosRomanos implements Validator{
    public void validate(FacesContext context, UIComponent component, Object value)
        throws ValidatorException {
        String valor = (String) value;
        valor = valor.trim();
        char []cad = valor.toCharArray();
        if(cad.length <= 0 || valor.equals("true")){
            throw new ValidatorException(new FacesMessage( FacesMessage.SEVERITY_ERROR, "",
            "Este campo no puede estar vacio."));}
        for(int ch:cad){
            switch(ch){
                case 73://I
                case 108://i
                case 105://V
                case 67://v
                case 86://X
                case 99://x
                case 118://L
                case 68://l
                case 88://C
                case 100://c
                case 120://D
                case 77://d
                case 76://M
                case 109:break;//m
                default:throw new ValidatorException(new
                FacesMessage(FacesMessage.SEVERITY_ERROR, "", "Este campo solo acepta los numeros romanos:
                I,V,X,L,C,D,M,i,v,x,l,c,d,m.")); } } }
```

## Alta de validador en faces-config.xml

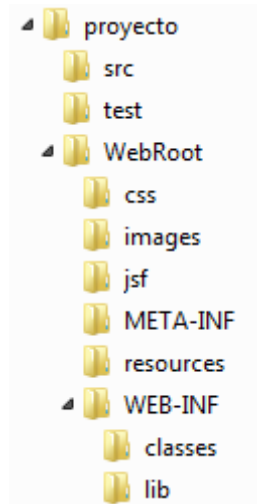
```
<validator>
<description> Validador de los números romanos </description>
<validator-id> ValidatorNumerosRomanos</validator-id>
<validator-class> mx.uaemex.proyecto.validators.ValidatorNumerosRomanos </validator-class>
</validator>
```

## 3.1.2 Etapa de creación del sistema proyecto

### 1. Estructura de creación de proyecto web.

Para iniciar, debe crearse con eclipse una aplicación de tipo Dinamic Web Project [véase anexo B]

Un proyecto de JSF desarrollado con eclipse tiene la siguiente estructura de carpetas (véase en la ilustración 10).



*Ilustración 10 Estructura de carpetas (Elaboración propia)*

En el editor, para comodidad de los programadores, la estructura de carpetas que es visible no es igual.

Hay tres carpetas principales:

- La carpeta src que contiene el código fuente del programa
- La carpeta test que contiene las pruebas unitarias
- La carpeta WebRoot que agrupa las vistas.

Dentro de la carpeta WebRoot se encuentran:

- La carpeta css con las hojas de estilo
- La carpeta images que contiene las imágenes y logos.
- La carpeta resources que puede guardar archivos de texto y multimedia referenciados por una URL
- La carpeta JSF con los archivos XHTML que conforman las vistas
- La carpeta META-INF, que en su archivo MANIFEST.MF guarda la ruta del CLASSPATH y otras configuraciones, en aplicaciones de escritorio guarda la ruta de la clase principal.
- La carpeta WEB-INF que contiene las clases de java ya compiladas y las bibliotecas .jar usadas por el proyecto.

En la carpeta META-INF también se almacenan los archivos XML de configuración de JSF, Spring e Hibernate. Esta carpeta no es accesible desde el directorio "test" así que cuando se desea desarrollar pruebas unitarias que involucren a los archivos de configuración estos deberán estar en la carpeta SRC o en algún lugar visible para el CLASSPATH.



## 2. Configuración de librerías.

Agregar bibliotecas a la aplicación [véase anexo C]

## 3. Agregar carpetas fuentes.

El editor eclipse nos permite agregar a nuestros proyectos:

- Folder: carpeta para guardar recursos tales como imágenes o códigos HTML.
- Package: carpeta para almacenar clases de java relacionadas.
- Source folder: carpeta que agrupa varios paquetes relacionados.

Una aplicación de java tendrá por default el “Source-folder” **SRC** que contendrá los paquetes y clases. En el caso del módulo de acceso a la base de datos es conveniente agregar el “source-folder” **TEST** cuyas clases no son del interés del resto de la aplicación y por ello pueden encontrarse separadas.

Para ello en la sección “java build path “ de las propiedades de la aplicación, seleccionar la pestaña “Source” y con el botón “add folder” agregar la carpeta de fuentes TEST (véase en la ilustración 11).

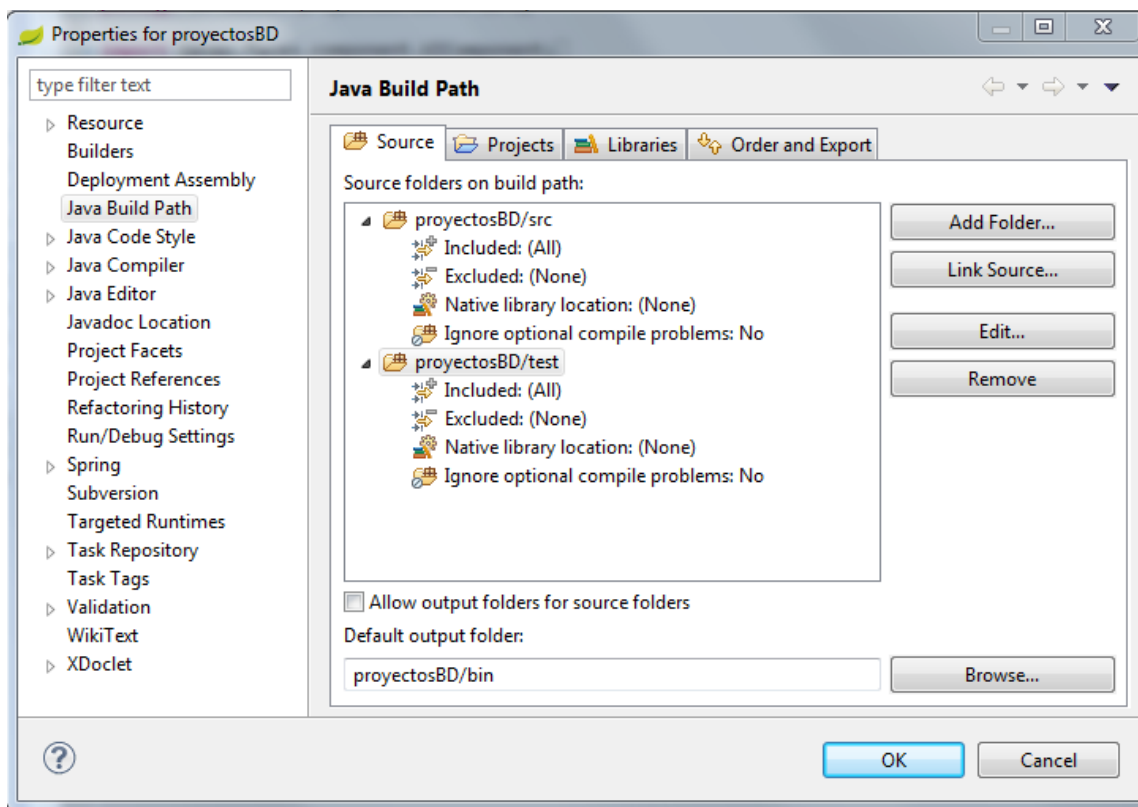


Ilustración 11 Java Build Path

A la carpeta de fuentes SRC se le agregarán los paquetes de la aplicación y a la carpeta de fuentes test se agrega el paquete que contendrán las pruebas unitarias. (véase en la ilustración 12).

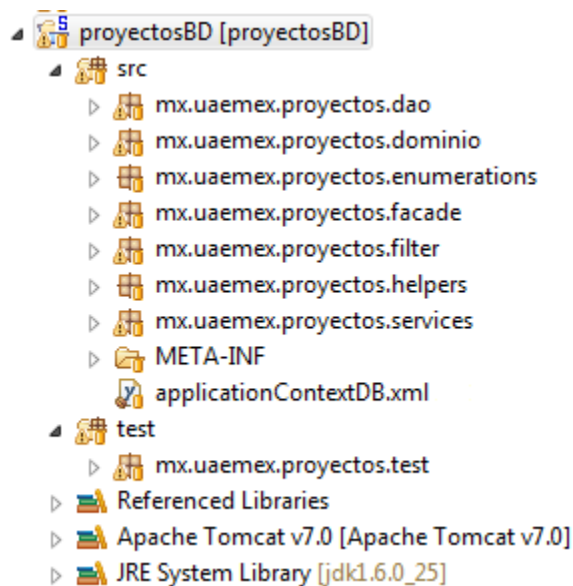


Ilustración 12 Carpetas

Por su parte el módulo de la interface gráfica contiene el paquete de fuentes src que contendrá las clases de java correspondientes a los controladores y converters. Además contiene un conjunto de carpetas que contendrán las vistas, imágenes, hojas de estilo y archivos XML de la aplicación (véase en la ilustración 13).

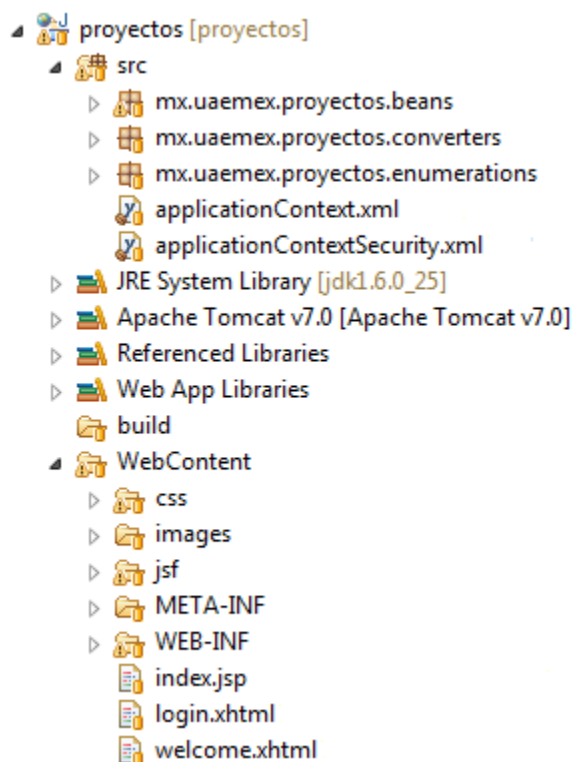


Ilustración 13 Carpetas

#### 4. AGREGAR MODULOS

La aplicación desarrollada en este trabajo está dividida en el módulo “proyectos” que es la interface gráfica y “proyectosDB” que es el módulo de acceso a la base de datos. Si en el futuro las interfaces deben ser migradas a una nueva versión de Primefaces o de otro framework, tales como IceFaces, MyFaces o RichFaces no será necesario rehacer la aplicación completa, solo el módulo de la interface gráfica (véase en la ilustración 14).

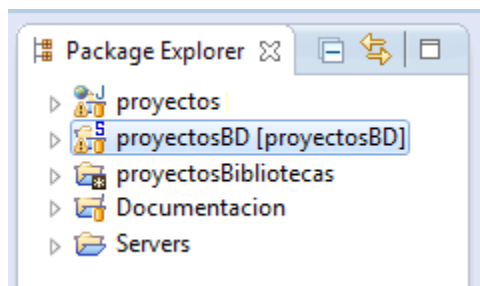


Ilustración 14 Módulos

Debe establecerse en la sección “projects” de las propiedades de “proyectos” que utilizará a “proyectosDB”. Las dependencias son agregadas como a continuación con el botón “add” (véase en la ilustración 15).

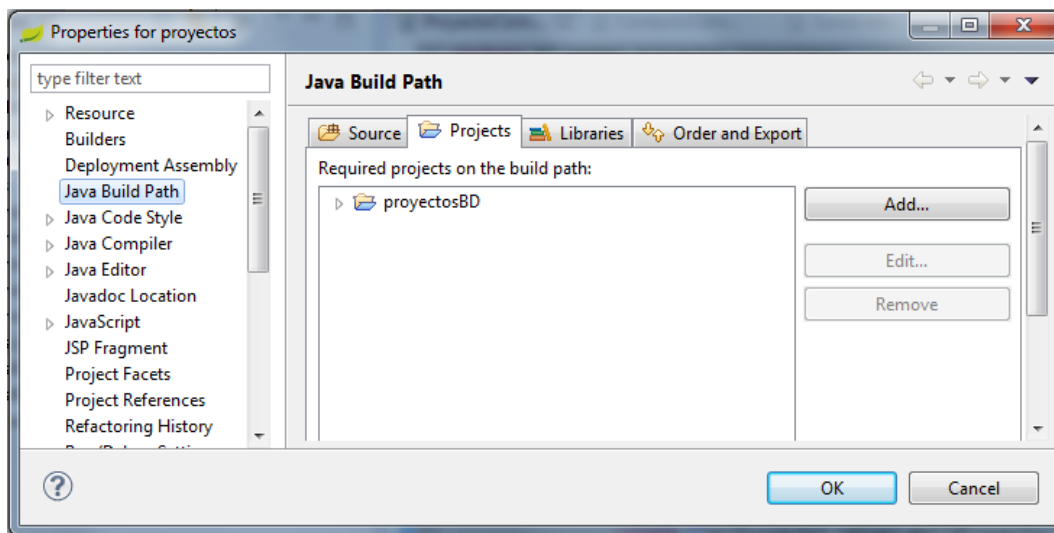


Ilustración 15 Java Build Path

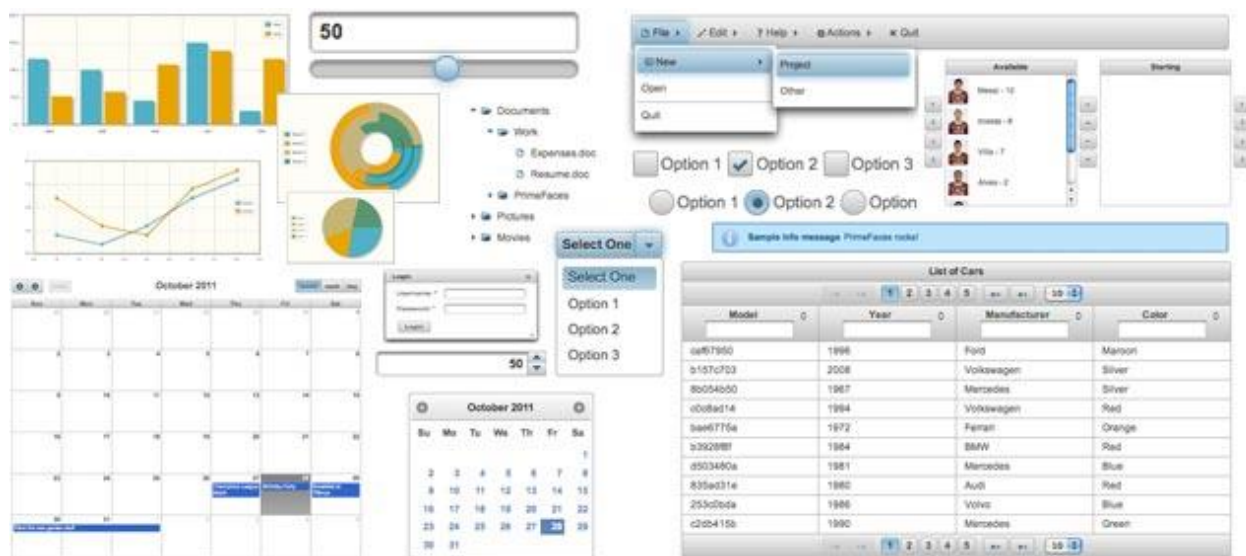
El proceso es equivalente a generar un archivo jar de la aplicación “proyectosBD” y agregarlo como biblioteca a “proyectos”.

De esta manera pueden agregarse, eliminarse e intercambiarse fácilmente módulos de una aplicación.

### 3.1.3 Integración del primefaces

Primefaces es un conjunto de componentes JSF open source con amplias extensiones unas de ellas son:

- Cuenta con un conjunto de componentes (HtmlEditor, Dialog, AutoComplete, Charts... etc véase en la ilustración 16)
- Primefaces es ligero, sin configuraciones ni dependencias y se distribuye en un único archivo jar.
- Incluye componentes para dispositivos móviles.
- Más de 35 temas listos para usar.
- Ampliamente documentado.



*Ilustración 16 Componentes de primefaces*

## Descarga

PrimeFaces es creado y mantenido por “premier Teknoloji”, una empresa desarrolladora de software de Turquía especializada en Java EE.

Primefaces es distribuido como un único archivo .jar que puede ser descargado junto con su documentación en su sitio oficial:

(<http://www.primefaces.org/>)

## Requerimientos

Primefaces funcionará en implementaciones de JSF 2 con JRE 5 o superior. Para utilizar ciertas funcionalidades como exportar los datos de una tabla a formato PDF requerirá algunas dependencias, de lo contrario es suficiente con el jar de la biblioteca.

## Configuración

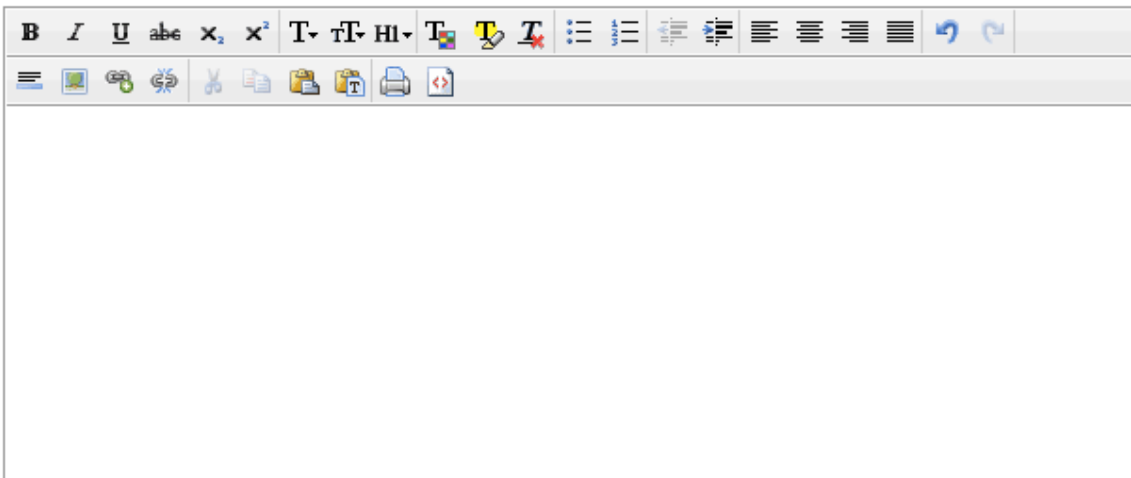
Solo se requiere agregar la biblioteca al CLASSPATH de la aplicación, no se requiere la modificación de ningún archivo de configuración.

## Uso de Primefaces

Una vez que se haya agregado la biblioteca al proyecto, se necesita agregar el namespace de primefaces (`xmlns:p="http://primefaces.org/ui"`) a la etiqueta HTML de cada interface que la utilice. Con ello cualquier componente que inicie con la letra P será reconocido como un componente de primefaces.

El siguiente ejemplo incluye el namespace de primefaces con lo que la etiqueta editor es traducida a un editor de texto de primefaces. Text es la variable tipo String del bean que recibirá el texto escrito en el editor cuando el formulario sea procesado. La variable text debe tener sus getters y setters (véase en la ilustración 17).

```
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">
  <h:head>
  </h:head>
  <h:body>
    <p:editor value="#{bean.text}" />
  </h:body>
</html>
```



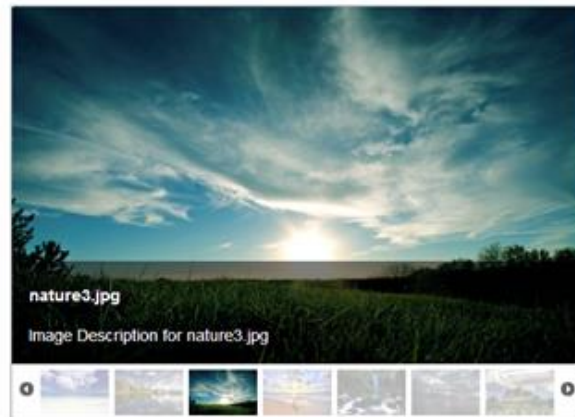
*Ilustración 17 Componente de edición de primefaces*

**Otros componentes de primefaces** (véase en la ilustración 18).

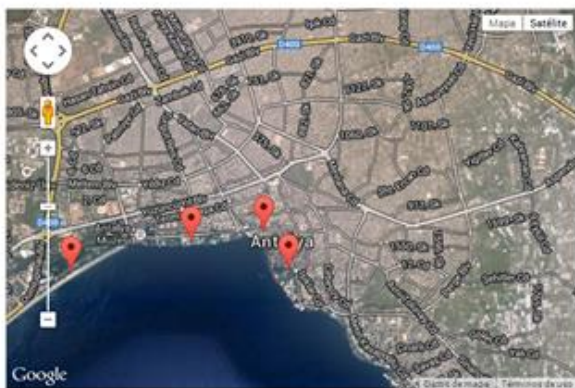
Calendar



Galleria



Gmap



DataTable

(1 of 5)			
Id	Year	Brand	Color
5bd18713	2001	Volvo	Brown
ef31e8f1	1999	Volvo	Black
32fb0ad2	1969	Audi	Brown
6dd7b399	1998	BMW	Red
728991d5	1990	Honda	Orange
cee4caf2	1963	Honda	White
32282f0b	2004	Volvo	Yellow
ab709fd4	1962	Renault	White
4400e0ab	2004	Jaguar	Maroon
fcf33a00	1963	Ford	Blue
(1 of 5)			

*Ilustración 18 Componentes de primefaces*



## Temas

Primefaces pone a disposición de los programadores más de 35 temas listos para usarse. Para utilizarlos hay que seleccionar uno y descargarlo en <http://www.primefaces.org/themes.html> (véase en la ilustración 19).

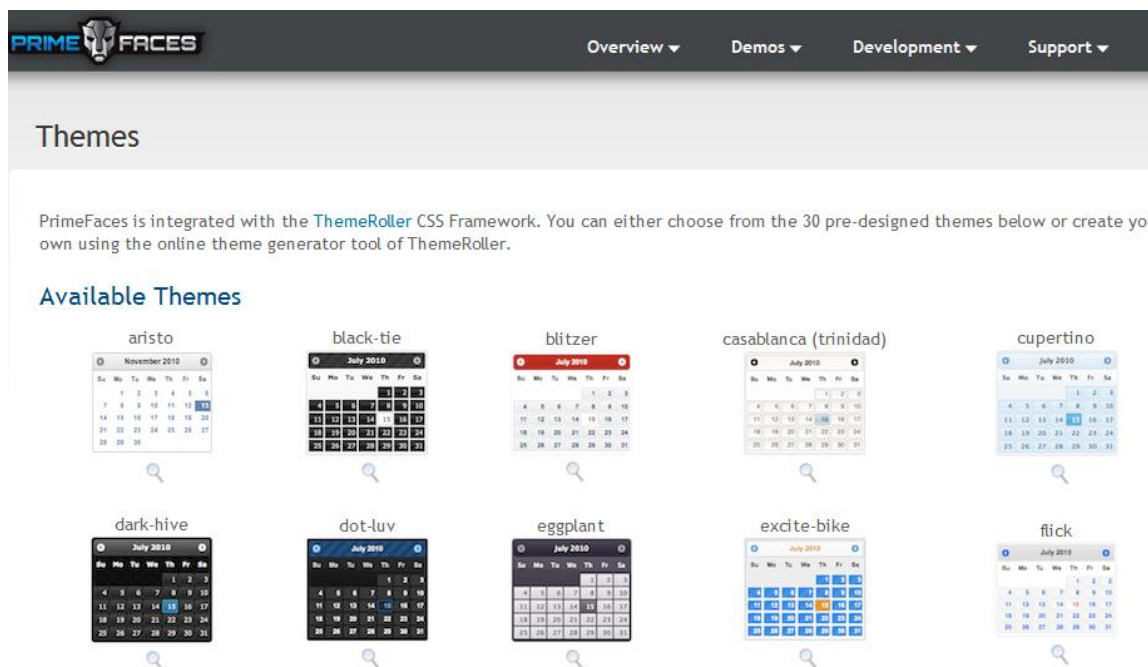


Ilustración 19 Temas de primefaces

Se pueden establecer estilos personalizados en <http://jqueryui.com/themeroller/> (véase en la ilustración 20).

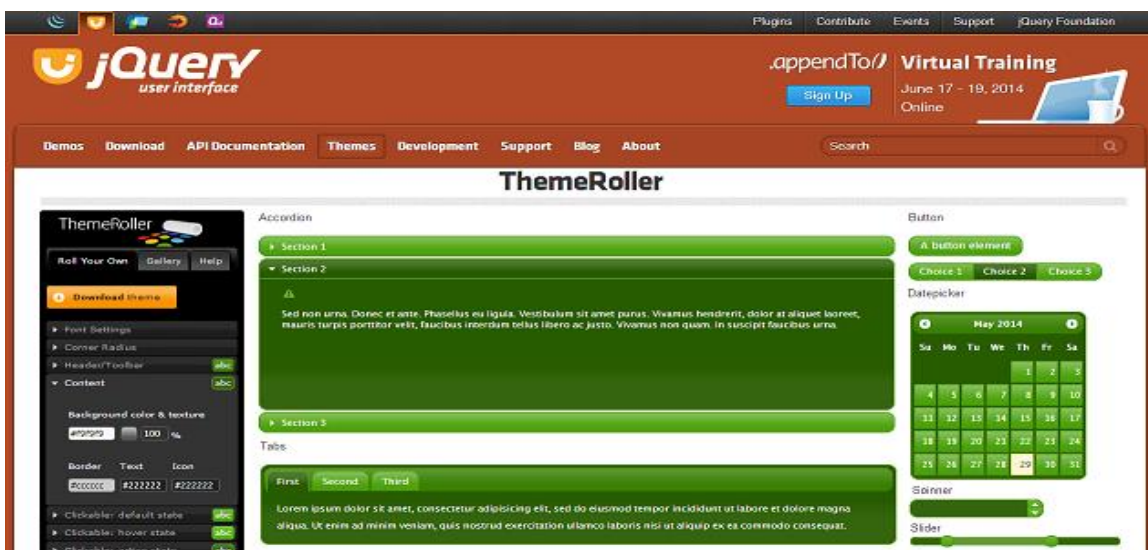


Ilustración 20 Temas de jQuery

Una vez descargado el archivo jar del tema hay que agregarlo como biblioteca al proyecto.

Finalmente hay que agregar en el archivo web.xml la etiqueta que hace referencia al nombre del tema:

```
<context-param>
  <param-name>primefaces.THEME</param-name>
  <param-value>bootstrap</param-value>
</context-param>
```

Código XHTML de la barra de menús desarrollada con primefaces:

```
<p:menubar >
  <p:menuitem value="Home" icon="ui-icon-home" url="/jsf/principal.jsf" />
  <p:submenu label="Catálogos" icon="ui-icon-contact" >
    <p:menuitem value="Contactos" icon="ui-icon-person" url="/jsf/catalogos/ContactoLista.jsf"/>
    <p:menuitem value="Proyectos" icon="ui-icon-calculator" url="/jsf/catalogos/ProyectoLista.jsf"/>
  </p:submenu>
  <p:submenu label="Asignaciones" icon="ui-icon-note" >
    <p:menuitem value="Tareas" icon="ui-icon-clock" url="/jsf/catalogos/TareaLista.jsf"/>
  </p:submenu>
  <p:menuitem
    value="Cerrar sesión: "
    actionListener="#{LoginBean.doLogout}"
    url="/j_spring_security_logout"
    ajax="true"
    icon="ui-icon ui-icon-close"/>
</p:menubar>
```

### 3.1.4 Implementación de JSF en la aplicación.

Las ventanas involucradas en el inicio de sesión se explican más adelante cuando se aborde el tema de Spring Security. El resto de la aplicación está compuesta por los catálogos. Cada catalogo tiene dos ventanas, la de búsqueda y la captura de nuevos registros (que es reutilizada para la edición).

#### Template

El template de la aplicación es la sección que permanece igual para todas las ventanas, esto es el encabezado (incluyendo la barra de menús) y el pie de página. Pueden conseguirse Templates en varios sitios de internet o el cliente puede tener uno propio que hay que implementar.

La creación de un template significa que las vistas están repartidas en varios archivos. JSF mediante el API facelets provee herramientas para unir los archivos que componen una interfaz.

En uno o varios archivos XHTML pueden establecerse secciones que son nombradas con un alias, en el siguiente ejemplo la etiqueta composition se refiere al documento y la etiqueta “define” a una sección que es bautizada con el identificador “contenido”:



```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    template="/jsf/layout/template.xhtml">
    <ui:define name="contenido">
        <!-- Código HTML →
    </ui:define>
</ui:composition>
```

El template por su parte utilizará la etiqueta insert para llamar a la sección “contenido” del archivo anterior:

```
<html
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui"
    xml:lang="es"
    lang="es">
<h:head>
</h:head>
<h:body>
<h:form>
<!-- ENCABEZADO HTML →
    <div>
        <ui:insert name="contenido"></ui:insert>
    </div>
    <!--PIE DE PAGINA →
</h:form>
</h:body>
</html>
```

## Catálogo de proyectos

El catálogo de proyectos inicia con la definición de la sección “contenido”. Esta rodea al resto del código. La ventana contiene 2 componentes principales, un panel en la parte superior con los campos de búsqueda y una tabla en la parte inferior con los resultados de la búsqueda.

Un panel de primefaces se crea con la etiqueta p:panel del que el atributo header señala el título que se muestra en la parte superior izquierda del componente:

```
<p:panel header="Proyectos">
</p:panel>
```

El campo de texto que permite establecer el nombre del proyecto a buscar es un p:inputText. Este componente tiene un atributo **value** que establece la variable de tipo String que recibirá el valor escrito por el usuario cuando el formulario se procese, es decir, hay un controlador de nombre ProyectoBean con una variable de nombre proyectoFiltro que tiene una variable “nombre” de tipo string:

```
<p:inputText
    id="idProyecto"
    value="#{ProyectoBean.proyectoFiltro.nombre}"
    title="Escribe"
    size="30"
    maxlength="45"
/>
```

### Código completo del catálogo de proyectos:

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui"
    template="/jsf/layout/template.xhtml" lang="es">
    <ui:define name="contenido">
        <p:panel header="Proyectos">
            <table style="border-collapse: separate; border-spacing: 5px; width: 100%">
                <tr>
                    <td align="right" style="width: 15%">
                        <h:outputLabel value="Nombre del proyecto:" for="idProyecto" />
                    </td>
                    <td style="width: 25%">
                        <p:inputText
                            id="idProyecto"
                            value="#{ProyectoBean.proyectoFiltro.nombre}"
                            title="Escribe"
                            size="30"
                            maxlength="45"
                        />
                        <p:watermark for="idProyecto" value="Introduce" />
                    </td>
                    <td align="right" style="width: 25%">
                        <h:outputLabel
                            value="Estatus:"
                            for="idEstatus"
                        />
                    </td>
                    <td style="width: 35%">
                        <p:selectOneMenu
                            id="idEstatus"
                            value="#{ProyectoBean.proyectoFiltro.estatus}"
                            style="width: 100px;" >
                            <f:selectItem itemValue="" itemLabel="---SELECCIONAR---" />
                            <f:selectItems value="#{ProyectoBean.estatusesProyecto}" var="e"
                                itemLabel="#{e.valor}" itemValue="#{e.clave}" />
                        </p:selectOneMenu>
                    </td>
                </tr>
                <tr>
                    <td colspan="4">
                        <p:commandButton
                            id="buscar"
                            value="Buscar"
                            title="Busqueda"
                        />
                    </td>
                </tr>
            </table>
        </p:panel>
    </ui:define>
</ui:composition>
```

```

        update="listado"
        action="#{ProyectoBean.onBuscar}"
        icon="ui-icon-search"
        process="@this , formEverything"
    />
    <p:commandButton
        id="btnAddNew"
        value="Agregar nuevo"
        title="Agregar nuevo contacto"
        action="#{ProyectoBean.onNuevo}"
        icon="ui-icon-plusthick"
    />
</td>
</tr>
</table>

</p:panel>
<br/>
<p:dataTable
    id="listado"
    var="proyecto"
    value="#{ProyectoBean.proyectos}"
    rows="10"
    paginator="true"
    paginatorTemplate="{CurrentPageReport} {FirstPageLink}
{PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {RowsPerPageDropDown}"
    rowsPerPageTemplate="10,20,30"
>
    <p:columnGroup type="header">
        <p:row>
            <p:column headerText="Nombre" />
            <p:column headerText="Descripción" />
            <p:column headerText="Estatus" />
            <p:column headerText="Fecha de inicio" />
            <p:column headerText="Fecha de término" />
            <p:column headerText="Porcentaje de avance" />
            <p:column headerText="Acciones" />
        </p:row>
    </p:columnGroup>
    <p:column style="text-align:left;">
        <h:outputText value="#{proyecto.nombre}" />
    </p:column>
    <p:column style="text-align:left;">
        <h:outputText value="#{proyecto.descripcion}" />
    </p:column>
    <p:column style="text-align:left;">
        <h:outputText value="#{proyecto.estatus}" />
    </p:column>
    <p:column style="text-align:left;">
        <h:outputText value="#{proyecto.inicio}" />
    </p:column>
    <p:column style="text-align:left;">
        <h:outputText value="#{proyecto.fin}" />
    </p:column>
    <p:column style="text-align:center;">
        <h:outputText value="#{proyecto.porcentajeAvance}" />
    </p:column>
    <p:column style="text-align:center;">

```

```
<p:commandLink action="#{ProyectoBean.onEditar}" >
  <h:outputText value="editar" />
  <f:setPropertyActionListener
    target="#{ProyectoBean.proyecto}"
    value="#{proyecto}" />
</p:commandLink>
<p:spacer width="5" height="1" />
<p:commandLink action="#{ProyectoBean.onMostrar}" >
  <h:outputText value="ver" />
  <f:setPropertyActionListener
    target="#{ProyectoBean.proyecto}"
    value="#{proyecto}" />
</p:commandLink>
<p:spacer width="5" height="1" />
<p:commandLink action="#{ProyectoBean.onEliminar}" >
  <h:outputText value="borrar" />
  <f:setPropertyActionListener
    target="#{ProyectoBean.proyecto}"
    value="#{proyecto}" />
</p:commandLink>
</p:column>

</p:dataTable>

</ui:define>
</ui:composition>
```

## 3.2 Implementación de persistencia

Que es Persistencia:

Significa existencia o duración por largo tiempo. La persistencia permite guardar un objeto lo que puede ser en una base de datos o archivo, para gestionarlo más adelante.

### 3.2.1 Integración del framework hibernate.

Los Framework están de moda en la actualidad en el mundo de la Ingeniería de Software, son entornos de desarrollo que hacen más fácil alguna tarea y que tienen internamente mucha programación y una estructura sólida bien definida. Para trabajar con un Framework es indispensable conocer su metodología de trabajo de la cual extiende o saber el uso de las librerías o paquetes de la cual se compone. Al momento de crear una nueva aplicación de software, los programadores tienen diferentes alternativas para escoger, es decir, buscar que Framework les sirve más para desarrollar su trabajo.

ORM “Object-Relational mapping” que es Mapeo de Objeto-Relacional que el código que se realiza para guardar el valor de nuestra clase en una base de datos relacional es decir:

“Framework de persistencia de nuestra base de datos relacional”.

La idea de Hibernate la tuvo Gavin King, Ingeniero del actual grupo JBoss, el cual cansado de la ineficiencia y complejidad de los sistemas de persistencia de la época, ideó un sistema base que fue apoyado por un inmenso grupo de programadores alrededor del mundo. Esta idea de King se

transformó en un proyecto robusto, con licencia libre que actualmente es el más utilizado, en cuanto a Framework dedicados a persistencia se refiere.

(<http://www.mundoblogweb.com/programadores-java/framework-hibernate.html>, 2012)

Tradicionalmente, el guardado en la base de datos se realiza mediante el Puente JDBC-ODBC, por ejemplo el siguiente método almacena los datos de un alumno:

```
public void guardarEnBD(String nombre, Integer calificacion) throws Exception {  
    Class.forName("com.mysql.jdbc.Driver");  
    String URL = "jdbc:mysql://localhost:3307/escuela"  
    Connection connection = DriverManager.getConnection(URL, "root", "myPass");  
    String insertQuery = "insert into alumno(NOMBRE,CALIFICACION) values (?,?)";  
    PreparedStatement preparedStatement = connection.prepareStatement(insertQuery);  
    preparedStatement.setString(1, nombre);  
    preparedStatement.setInteger(2, calificacion);  
    preparedStatement.executeUpdate();  
}
```

En cambio HIBERNATE simplifica las cosas al tratar las transacciones directamente en términos de objetos, en el siguiente ejemplo la sentencia SQL de guardado es creada automáticamente por el framework:

```
Public void guardarEnBD (Alumno alumno){  
    Session session = getSession();  
    Session.beginTransaction();  
    Session.save(alumno);  
    Session.getTransaction().commit();  
    Session.close();  
}
```

Algunos métodos de hibernate:

- save(object)
- update(object)
- saveOrUpdate(object)
- delete(object)
- List<Object> resultado = session.createQuery(query).list();

Sin hibernate, cuando una aplicación cambia de motor de bases de datos, todo el módulo de acceso a la base de datos debe reescribirse por que las consultas son diferentes entre distintas bases de datos. Con hibernate, el mismo modulo sirve para cualquier base de datos pues las consultas se modifican para cada motor automáticamente, el programador ya no escribe consultas en SQL propietario sino en HQL (Hibernate Query Language) que es neutro.

Características:

- Licencia LGPL.
- Ofrece su propio lenguaje de consulta HQL.

- Implementado con XML o Anotaciones (JPA).
- Amplia Documentación (<http://hibernate.org/orm/>).

### Ventajas

- Productividad: Evita mucho del código confuso de la capa de persistencia, permitiendo centrarse en la lógica de negocio.
- Mantenimiento: Por tener pocas líneas de código permite que el código sea más claro. Al dividir la capa de persistencia se puede identificar los errores muy fácilmente.
- Rendimiento: Existe la tendencia a pensar que una solución “manual” es más eficiente que una “automática”. Hibérnate tiene un buen desempeño pero todo depende realmente de cómo se realicen las consultas y como se configure el Framework.  
(<http://www.mundoblogweb.com/programadores-java/framework-hibernate.html>, 2012)

De todo el conjunto de Apis de Hibérnate, existen varias clases que permiten el trabajo básico con el Framework. Entre estas se encuentran:

- Session: Corresponde con un objeto que representa una unidad de trabajo con la base de datos (transacción). Además representa el gestor de persistencia, ya que dispone de la API básica para poder cargar y guardar objetos.
- Transaction: La API de Hibérnate contiene utilidades para demarcar la transaccionalidad de operaciones de manera programática.
- Query: Este interfaz permite crear consultas y enlazar argumentos a parámetros de la consulta (binding). Permite definir consultas en HQL (Hibernate Query Language) o en SQL.
- SessionFactory: Es una factoría de sesiones. Proporciona objetos Session. Es thread-safe. Permite concurrencia.

### 3.2.2 Mapeo objeto relacional con hibérnate

A continuación se muestra la base de datos de la que se va a mapear cada una de las tablas (véase en la ilustración 21)

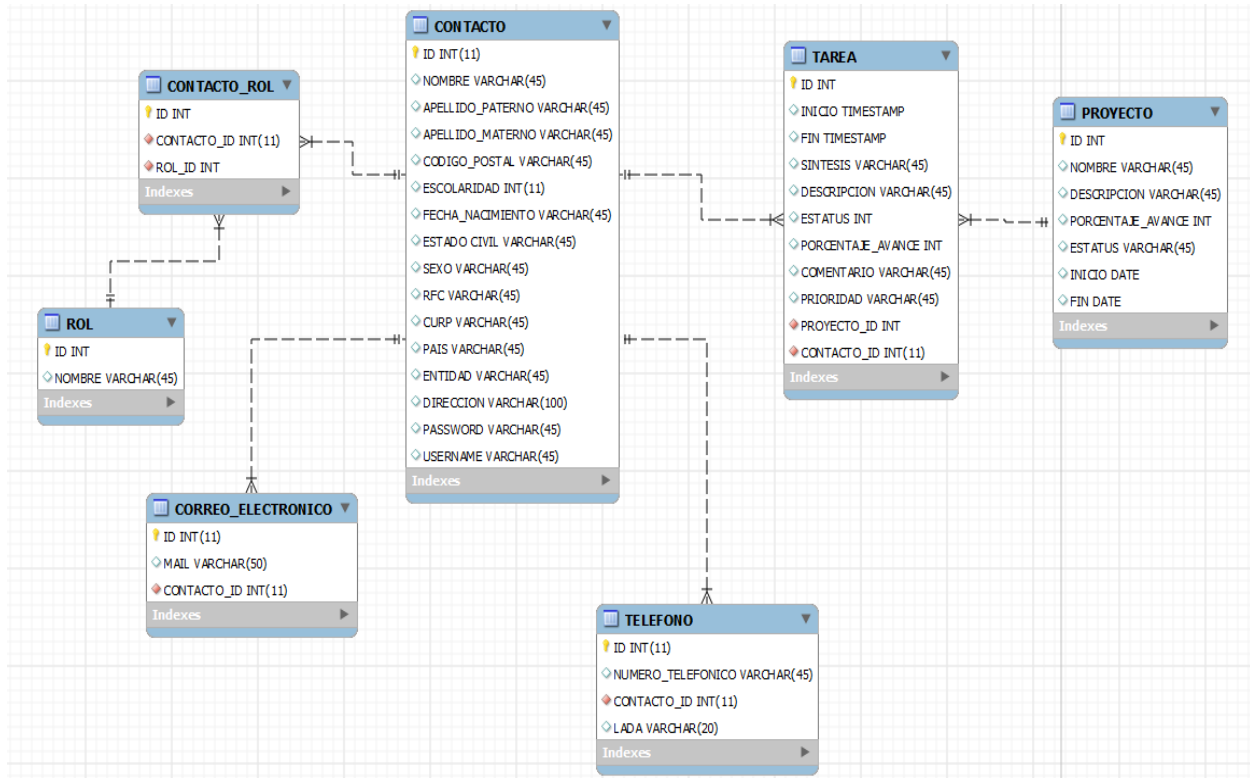


Ilustración 21 Base de datos del Estudio Genérico

En la sección introductoria, se habla del **dominio** de una aplicación, que se refiere a la codificación como clases de las entidades descritas por la base de datos. Al hecho de escribir una clase para cada tabla de la base de datos se le denomina Mapeo. Cada clase incluye una variable por cada campo de la base de datos

Hibernate sabe la tabla que corresponde a cada clase por que se le agrega una anotación “*table*”:

```
@Table(name = "NOMBRE_TABLA" , catalog="NOMBRE_BASE")
```

Hibernate sabe la columna en la tabla a la que corresponde cada variable de la clase porque se les agrega una anotación “*column*”:

```
@Column(name="NOMBRE_COLUMA", unique=true, nullable=false)
```

Para el mapeo de la tabla “TAREA” se crea una clase llamada Tarea, la cual está formada por sus atributos, constructores, métodos accesorios (getters y setters) y los métodos equal y hashCode. En el código siguiente se observa la necesidad de establecer las etiquetas de mapeo, las cuales permiten la relación de un atributo de la clase con un atributo de la tabla:

```
package mx.uaemex.proyectos.dominio;
import java.io.Serializable;
import javax.persistence.*;

import java.util.List;
import java.util.Date;
```

```
package mx.uaemex.proyecto.dominio;
import java.io.Serializable;
import javax.persistence.*;

import java.util.List;
import java.util.Date;

@Entity
@Table(name = "TAREA", catalog = "AGENDA")
public class Tarea implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name = "ID", unique = true, nullable = false)
    private Integer id;

    @Column(name="COMENTARIO")
    private String comentario;

    @Column(name="DESCRIPCION")
    private String descripcion;

    @Column(name="ESTATUS")
    private Integer estatus;

    @Column(name="FIN")
    private Date fin;

    @Column(name="INICIO")
    private Date inicio;

    @Column(name="PORCENTAJE_AVANCE")
    private Integer porcentajeAvance;

    @Column(name="PRIORIDAD")
    private String prioridad;

    @Column(name="SINTESIS")
    private String sintesis;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name = "PROYECTO_ID", nullable = false)
    private Proyecto proyecto;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name = "CONTACTO_ID", nullable = false)
    private Contacto contacto;

    public Tarea() {
    }

    ... GETTERS & SETTERS ....

    @Override
```



```
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((id == null) ? 0 : id.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Tarea other = (Tarea) obj;
    if (id == null) {
        if (other.id != null)
            return false;
    } else if (!id.equals(other.id))
        return false;
    return true;
}

@Override
public String toString() {
    return "Tarea [id=" + id + ", fin=" + fin + ", inicio=" + inicio + ", sintesis=" + sintesis + "];"
}
}
```

En el ejemplo anterior se observa que hay una relación de varios a uno con la tabla proyecto y contacto. Es decir, cada tarea está asociada a un proyecto y a un Contacto. Programáticamente, si quisiéramos saber el contacto a la que una tarea está asignada usaríamos:

```
System.out.println("la tarea "+tarea.getDescripcion() +" está asignada al contacto "+tarea.getContacto().getNombre());
```

Por su parte la clase Contacto tiene una relación inversa de uno a varios hacia la clase tarea, es decir, cada contacto tiene varias tareas asignadas:

```
@OneToMany(
    mappedBy = "contacto", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
private List<Tarea> tareas;
```

Si quisiéramos imprimir las tareas asignadas a un contacto escribiríamos:

```
System.out.println("las tareas asignadas al usuario "+usuario.getNombre()+" son:");
For(Tarea tarea:contacto.getTareas()){
    System.out.println("tarea:"+tarea.getDescripcion());
}
```

Hibernate intentara asignar una columna a cualquier método escrito en un DTO; en el caso de requerir un método que no corresponde a un campo de la base de datos, se excluye de hibernate con la anotación @transient. En el siguiente ejemplo se utiliza la anotación transient porque no hay una columna NOMBRE\_COMPLETO en la tabla CONTACTO

```
@Transient
Public String getNombreCompleto(){
Return nombre+" "+ paterno+" "+ materno;
}
```

### 3.2.3 Data Acces Object

Una vez que las tablas están mapeadas es hora de escribir un DAO (Data Acces Object) por cada una de ellas. Los DAO son los encargados de realizar consultas a la base de datos.

Implementar la estructura DAO para realizar el acceso a la base de daos permite que:

- El modulo puede desarrollarse por varias personas, cada una para ciertos DAO.
- El diagnóstico de errores sea más rápido, si por ejemplo el método de la capa de servicios que agrega contactos falla, entonces el error está en ContactoDAO.
- Pueda utilizarse algún generador de código porque es un estándar.

Por ejemplo, la clase ContactoDAO incluirá el siguiente método:

```
public Contacto nuevo(Contacto contacto){
    Session session = getCurrentSession();
    session.beginTransaction();
    session.save(contacto);
    session.getTransaction().commit();
    session.close();
    return contacto;
}
```

La clase ProyectoDAO incluirá el siguiente método:

```
public Proyecto nuevo(Proyecto proyecto){
    Session session = getCurrentSession();
    session.beginTransaction();
    session.save(proyecto);
    session.getTransaction().commit();
    session.close();
    return proyecto;
}
```

Se observa que en ambas clases el método es el mismo, solo cambia el tipo de objeto utilizado. Esto significa que se puede hacer una clase generica de la que hereden ProyectoDAO y ContactoDAO:

```
public class GenericDAO<T> implements Serializable{

    public T nuevo(T t){
        Session session = getCurrentSession();
        session.beginTransaction();
        session.save(t);
        session.getTransaction().commit();
        session.close();
        return t;
    }
}
```

Ahora cualquier DAO tendrá acceso al método “nuevo” con solo heredar de GenericDAO :

```
public class ProyectoDAO extends AbstractDAO<Proyecto>
```

El método buscar es diferente porque la consulta depende de cada tabla, así que no puede implementarse del todo en la clase genérica:

```
public List<T> buscar(AbstractFiltro filtro){
    where=false;
    String query = select ¿? From ¿? // los campos y las tablas dependen de la clase que herede este método
    Session sesion = getCurrentSession();
    sesion.beginTransaction();
    Query queryHibernate = sesion.createQuery(query);
    List<T> result = queryHibernate.list();
    sesion.getTransaction().commit();
    sesion.close();
    return result;
}
```

Para solucionarlo se establece que el método que crea la consulta sea sobrescrito por cada clase que herede de esta.

```
protected String construirQueryBusqueda(Filtro filtro) {};
protected void agregarParametrosBusqueda(Query query, Filtro filtro) {};
```

El método buscar queda:

```
public List<T> buscar(Filtro filtro){
    where=false;
    String query = construirQueryBusqueda(filtro); // ahora la query es responsabilidad de quien herede
    Session sesion = getCurrentSession();
    sesion.beginTransaction();
    Query queryHibernate = sesion.createQuery(query);
    agregarParametrosBusqueda(queryHibernate, filtro); // prepared statement
    List<T> result = queryHibernate.list();
    sesion.getTransaction().commit();
    sesion.close();
    return result;
}
```

Sobre escritura en la clase ProyectoDAO de los métodos construirQueryBusqueda y agregarParametrosBusqueda:

```
protected String construirQueryBusqueda(AbstractFiltro abstractFiltro) {
    ProyectoFiltro filtro =(ProyectoFiltro)abstractFiltro;
    StringBuilder qsb = new StringBuilder("from Proyecto as u where");

    if(filtro.getId()!=null){
        HelperDAO.addQuery(qsb,"u","idProyecto");
    }
    if(filtro.getNombre()!=null){
        if(!filtro.getNombre().isEmpty()){
            HelperDAO.addQuery(qsb,"u","nombre");
        }
    }
    if(filtro.getDescripcion()!=null){
```

```
        if(!filtro.getDescripcion().isEmpty()){
            HelperDAO.addQuery(qsb,"u","descripcion");
        }
    }
    if(filtro.getEstatus()!=null){
        if(!filtro.getEstatus().isEmpty()){
            HelperDAO.addQuery(qsb,"u","estatus");
        }
    }
}

if (qsb.toString().contains("and")) {
    return qsb.toString().substring(0, qsb.toString().lastIndexOf("and"));
} else {
    return qsb.toString().substring(0, qsb.toString().lastIndexOf("where"));
}
}

protected void agregarParametrosBusqueda(Query query, AbstractFiltro abstractFiltro) {
    ProyectoFiltro filtro =(ProyectoFiltro)abstractFiltro;
    addParameter(query, "idProyecto", filtro.getId());
    addParameter(query, "nombre", filtro.getNombre());
    addParameter(query, "descripcion", filtro.getDescripcion());
    addParameter(query, "estatus", filtro.getEstatus());
}
```

El código anterior utiliza un objeto de tipo ProyectoFiltro que es equivalente al DTO Proyecto pero esta formulado para realizar búsquedas. Tiene los mismos campos que la clase Proyecto pero también otras variables y métodos que solo son del interés de JSF y no de Hibernate.

La query creada por estos métodos podría ser:

```
From Proyecto as u where u.nombre =:nombre and u.descripcion = :descripción
```

Nótese que las consultas de HQL se realizan en términos de clases y campos y no de tablas y columnas. “*Proyecto*” está escrito con P mayúscula por que hace referencia a la clase del mismo nombre y “*u.nombre*” utiliza el operador punto para hacer referencia a una variable escrita con minúsculas.

Los términos que tienen dos puntos como prefijo son remplazados posteriormente por el valor contenido en el campo correspondiente del filtro, es decir, la query es un prepared statement.

```
protected void addParameter(Query query, String id, Integer valor) {
    if (valor != null) { if(valor>0){query.setInteger(id, valor)} }
}
```

Finalmente, los métodos construirQueryBusqueda y agregarParametrosBusqueda pueden ser sobrescritos por la clase que los hereda, pero no están obligados a sobrescribirlos. Para forzar la sobre escritura deben ser declarados como abstractos en la clase base (véase abstracción):

### Código completo de la clase abstractDAO (versión abstracta y genérica)

```
public abstract class AbstractDAO<T> implements Serializable{

    protected SessionFactory sessionFactory;
    private static final ThreadLocal<Session> threadLocal = new ThreadLocal<Session>();
```

```
private Class<T> type;
boolean where;

public AbstractDAO(Class<T> type) {
    this.type=type;
}

public T nuevo(T t){
    Session session = getCurrentSession();
    session.beginTransaction();
    session.save(t);
    session.getTransaction().commit();
    session.close();
    return t;
}

public T actualizar(T t){
    Session session = getCurrentSession();
    session.beginTransaction();
    session.update(t);
    session.getTransaction().commit();
    session.close();
    return t;
}

public T buscarPorId(Integer id){
    Session session = getCurrentSession();
    session.beginTransaction();
    T t = (T)session.get(type, id);
    llenarObjetoBusqueda(t);
    session.getTransaction().commit();
    session.close();
    return t;
}

public boolean borrar(T t){
    boolean succes = true;
    Session session;
    session = getCurrentSession();
    try{
        session.beginTransaction();
        session.delete(t);
        session.getTransaction().commit();
    }catch(Exception e){
        e.printStackTrace();
        succes = false;
    }
    session.close();
    return succes;
}

public List<T> buscar(AbstractFiltro filtro){
    where=false;
    String query = construirQueryBusqueda(filtro);
    Session session = getCurrentSession();
    session.beginTransaction();
    Query queryHibernate = session.createQuery(query);
```

```
agregarParametrosBusqueda(queryHibernate, filtro);
List<T> result = queryHibernate.list();
session.getTransaction().commit();
session.close();
return result;
}
protected void llenarObjetoBusqueda(T t){}
protected abstract String construirQueryBusqueda(AbstractFiltro filtro) ;
protected abstract void agregarParametrosBusqueda(Query query, AbstractFiltro filtro) ;

public Session getCurrentSession(){
try{
Session session = sessionFactory.getCurrentSession();
return session;
}catch(Exception e){
Session session = (Session) threadLocal.get();
if (session == null || ! session.isOpen()) {
session = (sessionFactory != null) ? sessionFactory.openSession() : null;
threadLocal.set(session);
}
return session;
}
}

public SessionFactory getSessionFactory() {return sessionFactory;}
public void setSessionFactory(SessionFactory sessionFactory) {
this.sessionFactory = sessionFactory;}

protected void addParameter(Query query, String id, Integer valor) {
if (valor != null) {
    if(valor>0){
        query.setInteger(id, valor);
    }
}
}

protected void addParameter(Query query, String id, String valor) {
    if (valor != null) {
        if (!valor.isEmpty()) {
            query.setString(id, valor);
        }
    }
}
}
```

### Código completo de la clase ProyectoDAO.java

```
public class ProyectoDAO extends AbstractDAO<Proyecto> {

public ProyectoDAO() {
    super(Proyecto.class);
}

protected String construirQueryBusqueda(AbstractFiltro abstractFiltro) {
ProyectoFiltro filtro =(ProyectoFiltro)abstractFiltro;
StringBuilder qsb = new StringBuilder("from Proyecto as u where");
}
```

```
        if(filtro.getId()!=null){
            HelperDAO.addQuery(qsb,"u","idProyecto");
        }

        if(filtro.getNombre()!=null){
            if(!filtro.getNombre().isEmpty()){
                HelperDAO.addQuery(qsb,"u","nombre");}
        }

        if(filtro.getDescripcion()!=null){
            if(!filtro.getDescripcion().isEmpty()){
                HelperDAO.addQuery(qsb,"u","descripcion");
            }
        }

        if(filtro.getEstatus()!=null){
            if(!filtro.getEstatus().isEmpty()){
                HelperDAO.addQuery(qsb,"u","estatus");
            }
        }
    }

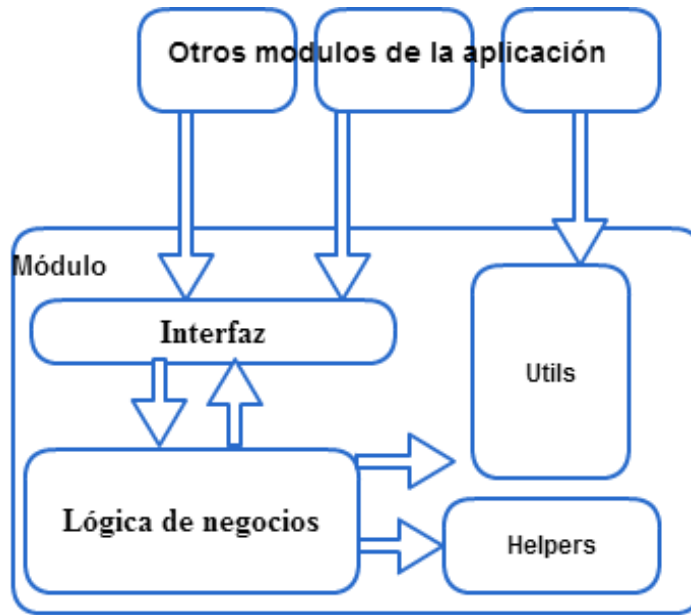
    if (qsb.toString().contains("and")) {
        return qsb.toString().substring(0, qsb.toString().lastIndexOf("and"));
    } else {
        return qsb.toString().substring(0, qsb.toString().lastIndexOf("where"));
    }
}

protected void agregarParametrosBusqueda(Query query, AbstractFiltro abstractFiltro) {
    ProyectoFiltro filtro =(ProyectoFiltro)abstractFiltro;
    addParameter(query, "idProyecto", filtro.getId());
    addParameter(query, "nombre", filtro.getNombre());
    addParameter(query, "descripcion", filtro.getDescripcion());
    addParameter(query, "estatus", filtro.getEstatus());
}
}
```

### 3.2.4 Capa de servicios

Los métodos de los módulos de una aplicación desarrollada con Java pueden agruparse en aquellos que son estáticos y los que no son estáticos. Otros módulos de la aplicación pueden hacer uso de ambos tipos de métodos. Los métodos estáticos que un módulo ofrece se denominan utilerías. Los métodos estáticos que solo el modulo utiliza y que no son del interés del resto de la aplicación se denominan ayudantes.

Por otro lado, el resto de la lógica de negocios es realizada con Clases que solo interesan al módulo, por lo que estas permanecerán ocultas para el resto de la aplicación. De esta sección solo será accesible una interfaz que redirección las peticiones al resto de las clases del módulo (véase en la ilustración 22).



*Ilustración 22 lógica de negocios*

Esta interfaz de acceso, presente en todos los módulos subyacentes al módulo de la interfaz gráfica se denomina capa de servicios.

Para la aplicación a desarrollar se necesitan los siguientes métodos en el módulo de guardado en base de datos.

- Guardar nuevos registros
- Editar registros
- Eliminar registros
- Búsqueda

Para empezar a construir el módulo de acceso a la base de datos el programador asignado genera una interfaz que lista los métodos que este módulo abarcará. Deberá ponerse de acuerdo con quien desarrolla la interfaz gráfica para satisfacer sus requerimientos.

Interfaz de la capa de servicios:

```
import java.util.List;
import mx.uaemex.proyectos.filter.ContactoFiltro;
import mx.uaemex.proyectos.filter.ProyectoFiltro;
import mx.uaemex.proyectos.filter.RolFiltro;
import mx.uaemex.proyectos.filter.TareaFiltro;
import mx.uaemex.proyectos.dominio.Contacto;
import mx.uaemex.proyectos.dominio.Proyecto;
import mx.uaemex.proyectos.dominio.Rol;
import mx.uaemex.proyectos.dominio.Tarea;

public interface ServiciosAgenda {
```



```
// *****CONTACTO*****  
public Contacto nuevoContacto(Contacto contacto);  
public Contacto actualizarContacto(Contacto contacto);  
public Contacto getContacto(Integer id);  
public boolean borrarContacto(Contacto contacto);  
public List<Contacto> getContactos(ContactoFiltro contactoFiltro);  
// *****PROYECTO*****  
public Proyecto nuevoProyecto(Proyecto proyecto);  
public Proyecto actualizarProyecto(Proyecto proyecto);  
public Proyecto getProyecto(Integer id);  
public boolean borrarProyecto(Proyecto proyecto);  
public List<Proyecto> getProyectos(ProyectoFiltro proyectoFiltro);  
// *****TAREA*****  
public Tarea nuevoTarea(Tarea tarea);  
public Tarea actualizarTarea(Tarea tarea);  
public Tarea getTarea(Integer id);  
public boolean borrarTarea(Tarea tarea);  
public List<Tarea> getTareas(TareaFiltro tareaFiltro);  
}
```

Una vez que el contrato está definido, el programador asignado a desarrollar la interfaz gráfica puede empezar a desarrollar su parte del programa con la promesa de que esos métodos con exactamente esos parámetros serán implementados por la persona asignada a desarrollar el acceso a la base de datos.

La clase de java que implemente la interfaz direccionará las solicitudes hacia las clases que creen la conexión con la base de datos y que ejecuten las consultas SQL.

Específicamente para un módulo de acceso a la base de datos, la capa de servicios invocara los métodos que ofrecen los Data Acces Object, que son las clases encargadas de realizar acciones en cada una de las tablas de la base de datos (véase en la ilustración 23).

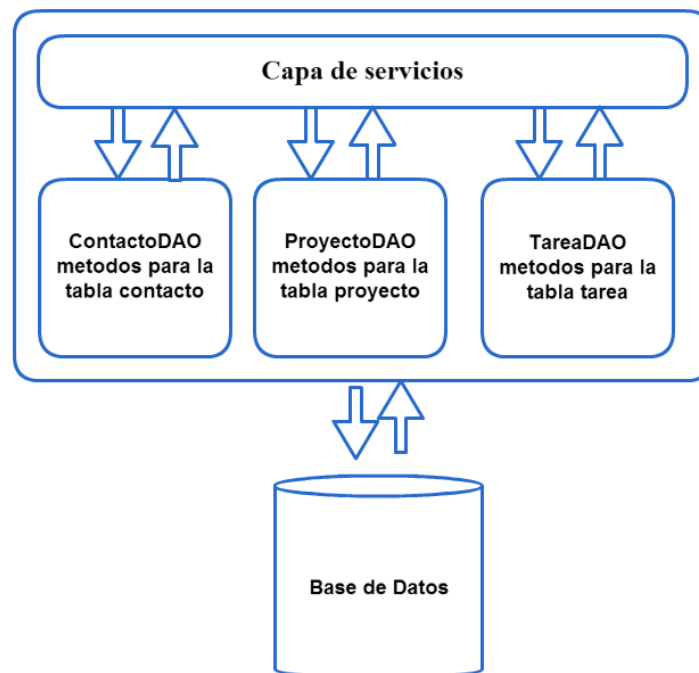


Ilustración 23 Acceso a la Base de Datos

### 3.2.5 Facades

Algunas veces un solo método de la capa de servicios puede requerir invocar a más de un DAO. También puede pasar que por ejemplo requiera realizar tareas intermedias o en cascada que no estén directamente relacionadas con el acceso a base de datos.

Por ejemplo, en un módulo de acceso a la base de datos de jugos de frutas, podría ser necesario enviar un correo a los proveedores cuando alguna fruta se termine.

- 1.- El proceso de envío de este correo no se puede poner en la interface gráfica porque si cambia la interfaz no debe cambiar la lógica.
- 2.- El proceso no se puede poner en el DAO por que enviar correos no es una tarea de acceso a bases de datos
- 3.- El proceso no se puede poner en la capa de servicios por que esta solo debe encargarse de direccionar las solicitudes, si se pusiera en la capa de servicios y el módulo de guardado cambiara se perdería esta lógica.

Por ello como paso intermedio, cuando la situación lo amerite, se encuentra la capa Facades. La capa facade puede requerir afectar a varias tablas. Por ejemplo, en la base de datos de jugos de frutas, al realizarse un jugo de fresas debe disminuirse en 5 el número de fresas, al hacerse un licuado de plátano debe usarse el DAO de los plátanos para disminuir en uno su cantidad, pero en el caso del licuado de fresas con plátano, el mismo método en la capa de servicios debe llamar a un método en la capa facade que disminuya tanto el número de plátanos como de fresas (véase en la ilustración 24).

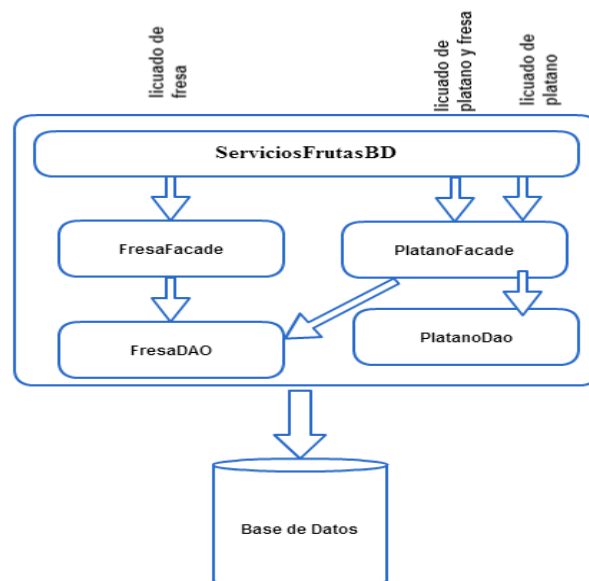
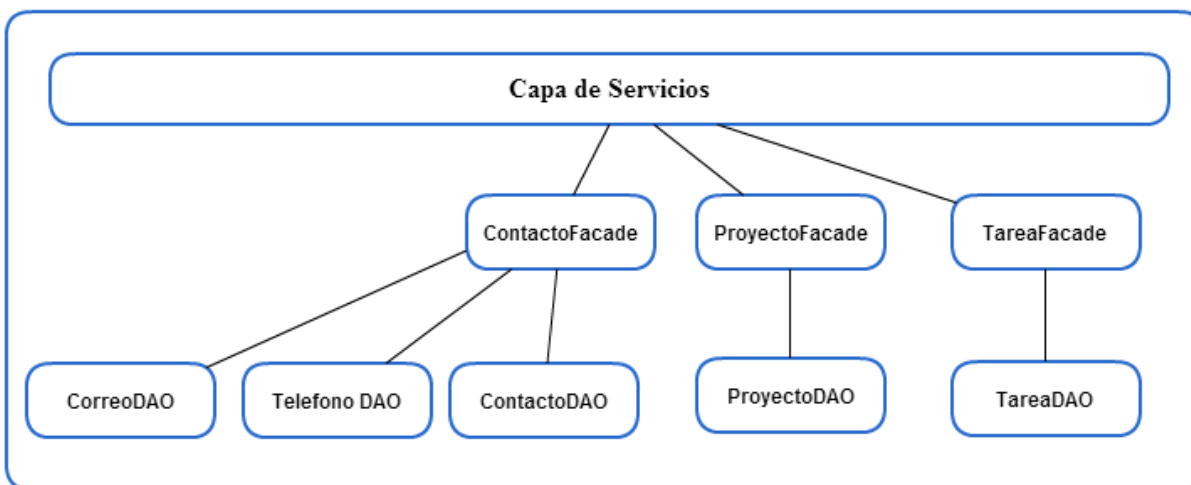


Ilustración 24 Ejemplo de Facades

Finalmente se muestra así (véase en la ilustración 25).



*Ilustración 25 Diagrama general del módulo de acceso a la base de datos*

### 3.3 Integración del framework de Spring

Tanto framework's como API's se componen de varias clases concentradas en archivos jar, su diferencia se centra en quien establece la arquitectura de una aplicación.

En el caso de las API's el programador establece la estructura de su aplicación y ciertas rutinas que no quiere o no puede realizar las delega a una biblioteca.

Por otro lado cuando se utiliza un framework se debe seguir la arquitectura que este establece y llenar los espacios que la herramienta no puede inferir

Además de la inyección de dependencias Spring ofrece Inversión de control, es decir, un programador puede invertir tiempo y esfuerzo en inventar todas las secciones que componen una aplicación o dejar el control de la arquitectura de su aplicación al framework y solo implementar aquello que es específico a su aplicación.

El framework Spring le provee al programador herramientas listas para usarse de las que solo hay que escribir su implementación. Una aplicación que aprovecha un framework incorpora fácil y rápidamente módulos desarrollados durante años por cientos de programadores y cuyo funcionamiento y seguridad están validados.

Secciones del framework Spring (véase en la ilustración 26).

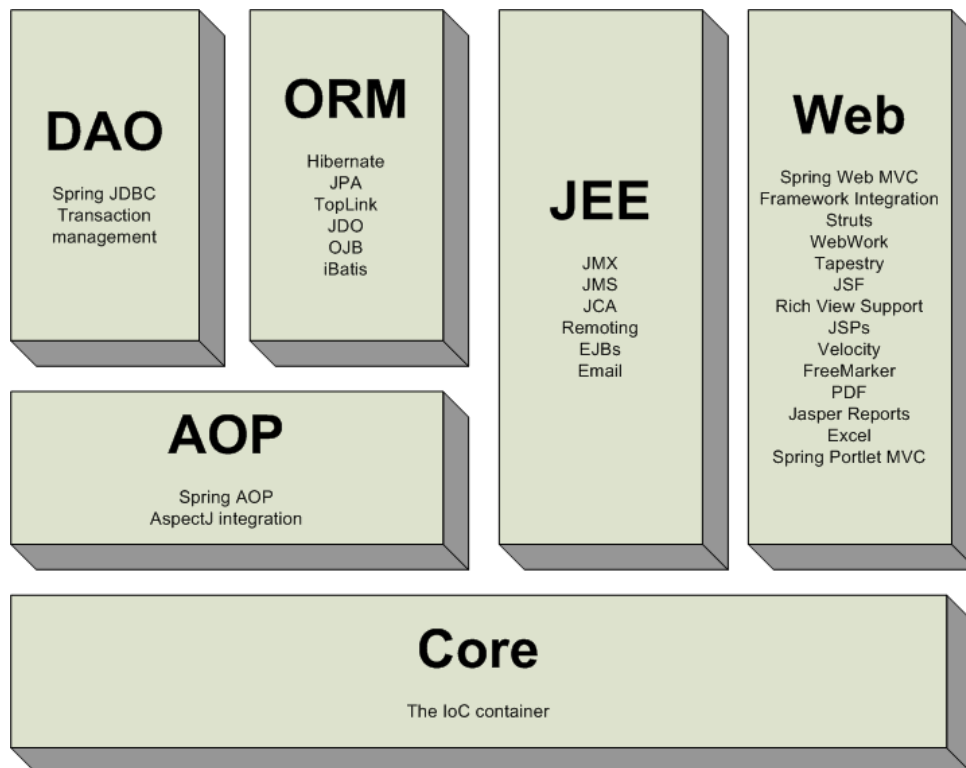


Ilustración 26(<http://docs.spring.io/spring/docs/2.0.x/reference/introduction.html>)

A diferencia de otros frameworks, Spring:

- Utiliza las API líderes en el mercado.
- No es necesario implementar la estructura completa sino solo la sección deseada.
- Puede incorporarse y quitarse fácilmente pues se utiliza a través de la inyección de dependencias configuradas mediante un archivo XML.
- Interactúa con otros frameworks como struts.
- Es el framework para aplicaciones web más utilizado lo que permite que haya versiones de editores especializadas en Spring.
- Es gratuito y de código abierto

### 3.3.1 Inyección de dependencias

La segunda versión de la clase test:

Test
- figura: Figura
+ run()
+ setFigura(Figura figura)

Depende de que una implementación de la interface Figura le sea puesta a disposición, ya sea una instancia de Triangulo o Cuadrado. Se le llama inyección de dependencias al establecimiento de los objetos que una clase necesita para funcionar.

### 3.3.2 Inyección de dependencias con Spring

A la clase Test se le puede establecer la dependencia Triangulo programáticamente:

```
Triangulo miTriangulo = new Triangulo();  
Test.setFigura(miTriangulo);
```

O con el framework Spring mediante un archivo XML de configuración en el que se crea un bean de la clase Triangulo llamado *miTriangulo* que se incluye (inyecta) en la variable *figura* del objeto *miTest*:

```
<bean id="miTriangulo" class="Triangulo" />  
<bean id="miTest" class="Test">  
  <property name="figura" ref="miTriangulo" />  
</bean>
```

Spring es un framework de inyección de dependencias que permite abstraer aplicaciones enteras como un conjunto de capas. También permite inyectar módulos enteros dentro de otros. Otro

aspecto es que la generación de los beans es dinámica, es decir, se crean hasta que son solicitados por primera vez lo que ahorra memoria.

Un módulo acompañado de su archivo de inyección de dependencias está listo para ser incluido dentro de otro con una simple sentencia:

```
<import resource="otroArchivoSpring.xml"/>
```

Los beans de java server faces también se establecen mediante un archivo XML, las dependencias expresadas en el contexto de Spring pueden inyectarse en el contexto de los beans de la aplicación web.

1.- declaración de beans de Spring en el ***applicationContext.xml*** :

```
<bean id="miObjetoDeSpring" class="ClaseSpring" />
```

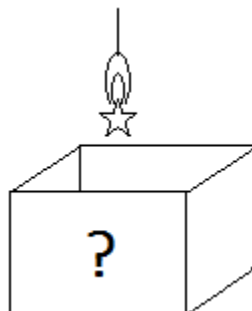
2.- llamar el archivo xml de Spring desde el web.xml:

```
<context-param>  
  <param-name>contextConfigLocation</param-name>  
  <param-value>  
    classpath: applicationContext.xml  
  </param-value>  
</context-param>
```

3.- A un bean de java server faces se le inyecta un bean de Spring desde el faces-config.xml:

```
<managed-bean>  
  <managed-bean-name>miObjetoJSF</managed-bean-name>  
  <managed-bean-class>ClaseJSF</managed-bean-class>  
  <managed-bean-scope>session</managed-bean-scope>  
  <managed-property>  
    <property-name>miDependencia</property-name>  
    <value>#{miObjetoDeSpring}</value>  
  </managed-property>  
</managed-bean>
```

El contexto de Spring puede considerarse una caja negra del que se extraen Beans por su identificador



Programáticamente:

Genera un objeto ApplicationContext a partir del XML de Spring:

```
String [] archivos = {"applicationContext.xml"};  
ApplicationContext contexto= new ClassPathXmlApplicationContext(archivos);
```

Se le solicitan beans al contenedor:

```
Estrella estrellita= (Estrella) applicationContext.getBean("miEstrella");
```

### 3.3.3 Inyección de dependencias en la aplicación web

En una aplicación de escritorio o web la clase que se encarga del acceso a la base de datos debe compartirse para todos los controladores:

Controller1
- servicios:ServiciosBD
+Controller1()

Pero si la capa de servicios es creada en el constructor, entonces cada controlador tendrá su propia capa de servicios lo que creará múltiples conexiones a la base de datos. Con spring la misma capa de servicios es compartida e inyectada para todos los controladores que la necesiten mediante el método setServicios:

Controller2
- servicios:ServiciosBD
+ setServicios(serviciosBD servicios)

### 3.3.4 Integración del módulo de persistencia a la aplicación web.

Por ejemplo, Spring contiene una sección dedicada a Hibernate desarrollada durante años por un gran número de programadores del más alto nivel que suple la necesidad del HibernateUtils y de la administración del pool de conexiones a la base de datos, su configuración se realiza mediante etiquetas XML que crean un DataSource que se inyecta a los diferentes DAO

1.- DataSource con los parámetros de conexión a la base de datos:

```
<bean id="DataSource"  
    class="com.mchange.v2.c3p0.ComboPooledDataSource"  
    destroy-method="close">  
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>  
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3307/miBase"/>  
    <property name="user" value="root"/>  
    <property name="password" value="miPassword"/>  
</bean>
```

2.- Se crea un SessionFactory al que se le inyecta el dataSource. El sessionFactory requiere la ubicación de las clases de persistencia:

```
<bean id="SessionFactory"
      class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
  <property name="dataSource" ref="DataSource" />
  <property name="annotatedClasses">
    <list>
      <value>mx.uaemex.proyectos.dominio.Contacto</value>
      <value>mx.uaemex.proyectos.dominio.CorreoElectronico</value>
      <value>mx.uaemex.proyectos.dominio.Proyecto</value>
      <value>mx.uaemex.proyectos.dominio.Tarea</value>
      <value>mx.uaemex.proyectos.dominio.Telefono</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
```

3.- finalmente el sessionFactory es inyectado a un DAO:

```
<bean id="ContactoDAO" class="mx.uaemex.proyectos.dao.ContactoDAO">
  <property name="sessionFactory" ref="SessionFactory"/>
</bean>
```

Archivo ApplicationContext.xml del módulo de acceso a la base de datos completo:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:p="http://www.springframework.org/schema/p"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context/spring-context-3.0.xsd"
  xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <bean id="DataSource"
    class="com.mchange.v2.c3p0.ComboPooledDataSource"
    destroy-method="close">
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>
    <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/base"/>
  </bean>
```



```
<property name="user" value="root"/>
<property name="password" value="miPassword"/>
<property name="maxPoolSize" value="140"/>
<property name="maxStatements" value="0"/>
<property name="minPoolSize" value="5"/>
</bean>
<bean id="SessionFactory"
  class="org.springframework.orm.hibernate4.LocalSessionFactoryBean" >
  <property name="dataSource" ref="DataSource" />
  <property name="annotatedClasses">
    <list>
      <value>mx.uaemex.proyectos.dominio.Contacto</value>
      <value>mx.uaemex.proyectos.dominio.CorreoElectronico</value>
      <value>mx.uaemex.proyectos.dominio.Proyecto</value>
      <value>mx.uaemex.proyectos.dominio.Tarea</value>
      <value>mx.uaemex.proyectos.dominio.Telefono</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <props>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
    </props>
  </property>
</bean>
<tx:annotation-driven transaction-manager="txManager"/>
<bean id="txManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
  <property name="sessionFactory" ref="SessionFactory"/>
</bean>
<!-- DAO -->
<bean id="ContactoDAO" class="mx.uaemex.proyectos.dao.ContactoDAO">
  <property name="sessionFactory" ref="SessionFactory"/>
</bean>
<bean id="ProyectoDAO" class="mx.uaemex.proyectos.dao.ProyectoDAO">
  <property name="sessionFactory" ref="SessionFactory"/>
</bean>
<bean id="TareaDAO" class="mx.uaemex.proyectos.dao.TareaDAO">
  <property name="sessionFactory" ref="SessionFactory"/>
</bean>
<!-- FACADE -->
<bean id="ContactoFacade" class="mx.uaemex.proyectos.facade.ContactoFacade">
  <property name="contactoDAO" ref="ContactoDAO"/>
</bean>
<bean id="ProyectoFacade" class="mx.uaemex.proyectos.facade.ProyectoFacade">
  <property name="proyectoDAO" ref="ProyectoDAO"/>
</bean>
<bean id="TareaFacade" class="mx.uaemex.proyectos.facade.TareaFacade">
  <property name="tareaDAO" ref="TareaDAO"/>
</bean>
<!-- CAPA DE SERVICIOS -->
<bean id="ServiciosAgenda" class="mx.uaemex.proyectos.services.ServiciosAgendaImpl">
  <property name="contactoFacade" ref="ContactoFacade" />
  <property name="proyectoFacade" ref="ProyectoFacade" />
  <property name="tareaFacade" ref="TareaFacade" />
</bean>
</beans>
```

### 3.4 Seguridad y perfiles de acceso

Los servidores suelen recibir peticiones de muchos clientes diferentes, y sólo son conscientes de un cliente en particular hasta que deben satisfacer una solicitud con una respuesta. Después de eso, los servidores se olvidan del cliente y pasan a cumplir con las peticiones de otros clientes. HTTP no proporciona mecanismos para distinguir un cliente de otro así que si sucede que dos peticiones cualesquiera vienen desde el mismo cliente, el servidor no es consciente de ello. En este sentido HTTP se conoce como un protocolo sin estado (Jose Annunziato, 2001)

El control de sesiones de usuario solía realizarse mediante el almacenamiento de cookies (pequeños paquetes de datos sobre el usuario) en el equipo cliente. También puede agregarse un identificador a las URL que un usuario visita.

A partir de la introducción de JSP se creó el concepto de creación de objetos por sesión del lado del cliente y del lado del servidor. Un objeto de tipo Session es incluido dentro del parámetro HttpServletRequest de los servlets para llevar el control de los datos del cliente y de los objetos creados.

#### 3.4.1 Integración de spring security

Para java server faces la implementación de sesiones de usuario ya no es explícita, sino que el contexto de la aplicación crea automáticamente una sesión cuando el usuario visita un sitio. Los datos de la sesión y del usuario son accesibles mediante un conjunto de palabras reservadas.

“j\_username” es el nombre de usuario y “j\_password” la contraseña que escribe, así que desde que un usuario accede a una aplicación de JSF se crea una sesión aun cuando el usuario no se halla logueado. Para el login se requiere un formulario JSP con dos elementos inputText asociados a j\_username y j\_password .

```
<h:inputText id="j_username" required="true" />
```

Spring security es una sección del framework spring lista para usarse que simplifica el proceso de administración de sesiones de usuario. Para usarlo es necesario incluir las bibliotecas:

1. springSecurity/spring-security-config-3.1.2.RELEASE.jar
2. springSecurity/spring-security-core-3.1.2.RELEASE.jar
3. springSecurity/spring-security-web-3.1.2.RELEASE.jar

y agregar las etiquetas correspondientes al applicationContext.

Para la aplicación desarrollada en este trabajo se establecen tres roles de usuario:

- ROLE\_ADMIN: captura los contactos, proyectos y asigna las áreas
- ROLE\_EMPLEADO: solo puede ver las áreas que le fueron asignadas.
- IS\_AUTHENTICATED\_ANONYMOUSLY: Rol predeterminado de spring que representa a los usuarios que no están logueados.

El funcionamiento de Spring security es muy sencillo, basta con establecer en un archivo xml las rutas de navegación y los roles que tienen permitido visitarlas. Un usuario puede visitar todas las secciones marcadas como "IS\_AUTHENTICATED\_ANONYMOUSLY" sin la necesidad de loguearse pero en cuanto quiera acceder a una URL protegida, aparecerá la página encargada de solicitarle sus credenciales.

La página de bienvenida incluye un link **"login"** hacia una página que solo deben ver usuarios registrados.

```
<a href="http://localhost:8080/proyectos/jsf/principal.jsf" >login</a>
```

Cada que se quiera acceder a la ruta protegida `http://localhost:8080/proyectos/jsf/principal.jsf` spring security revisa si el usuario está logueado (véase en la ilustración 27).

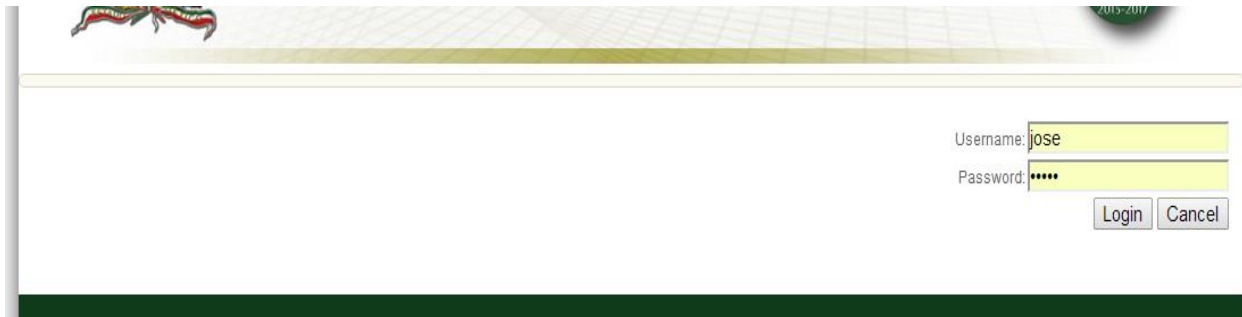


Ilustración 27 Vista de autenticación

Cuando un usuario ya esté registrado su navegación ya no se interrumpirá. El splash que pide el usuario y la contraseña está disponible por default pero puede suplirse por una vista personalizada

Para establecer el perfil de usuario que tiene permitido visitar una URL se utiliza la etiqueta `intercept-url` que recibe como parámetro `pattern` una expresión que simboliza a la ruta y `Access` que establece los roles:

```
<intercept-url pattern="/jsf/catalogos/**" access="ROLE_ADMIN, ROLE_EMPLEADO" />
```

Cabe recordar que las hojas de estilo e imágenes deben estar disponibles también para los usuarios anónimos.

```
<intercept-url pattern="**/*.css" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
Archivo ApplicationContextSecurity.xml completo:
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans
xmlns="http://www.springframework.org/schema/security"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/security
```

```
http://www.springframework.org/schema/security/spring-security-3.1.xsd">
<global-method-security secured-annotations="enabled"/>
<http auto-config="true" >
<intercept-url pattern="/index.jsp" access="IS_AUTHENTICATED_ANONYMOUSLY" />
<intercept-url pattern="/welcome.jsf" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/login.jsf" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/jsf/layout/**" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/jsf/layout/resources/**"
access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/images/**" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/css/**" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<intercept-url pattern="/jsf/principal.jsf"
access="ROLE_ADMIN, ROLE_EMPLEADO"/>
<intercept-url pattern="/jsf/catalogos/**"
access="ROLE_ADMIN, ROLE_EMPLEADO" />
<intercept-url pattern="***.css" access="IS_AUTHENTICATED_ANONYMOUSLY"/>
<http-basic />
</http>
<authentication-manager >
<authentication-provider>
<password-encoder hash="md5"/>
<user-service>
<user name="jose"
password="b3bf679735ce2ca3d433343c1e422198"
authorities="ROLE_ADMIN" />
<user name="ana"
password="55dd7e933d0e114fe7c5f3f29ac4ea3b"
authorities="ROLE_ADMIN" />
</user-service>
</authentication-provider>
</authentication-manager>
</beans:beans>
```

En el archivo se observa en la parte final la creación de un Bean authentication-manager y entre estas etiquetas la definición de los usuarios *jose* con rol `ROLE_ADMIN` y *ana* con rol `ROLE_EMPLEADO` que tendrán permitido visitar las secciones protegidas de la aplicación. El password se muestra codificado en formato MD5 de tal manera que aun cuando alguien acceda al código fuente no podrá inferir las contraseñas

Para el usuario *jose* la contraseña es *j0s3* cuya representación MD5 es *b3bf679735ce2ca3d433343c1e422198* y para el empleado *ana* la contraseña es *4n4* que se representa como *55dd7e933d0e114fe7c5f3f29ac4ea3b* en formato md5.

La codificación MD5 le aplica un algoritmo HASH a una cadena de texto. Un algoritmo HASH es una función matemática que genera un resumen de un documento de manera unidireccional, es decir, a partir de la cadena que resulta es muy difícil inferir el parámetro que la origino, aun sabiendo la formula aplicada. Por ejemplo, dos números sumados dan como resultado 567654435; El algoritmo MD5 secciona una cadena de texto en segmentos de 32 bits a los que aplica funciones matemáticas a la vez que las suma.

Para saber si un usuario ingresó la contraseña correcta Spring no necesita saber el texto original, sino que convierte a MD5 la cadena ingresada y la compara con el MD5 establecido en el XML.

## Capítulo 4

# RESULTADOS

En este trabajo se logró una descripción detallada del desarrollo de un proyecto básico, usando primefaces, spring e hibernate. En el capítulo tres se describe con detalle cada paso para usar los frameworks que permiten la implementación del MVC. El sistema ha sido desarrollado para llevar acabo la administración de proyectos, así como también la asignación de tareas a usuarios. Este sistema cuenta con dos perfiles de usuario:

- Perfil administrador: Cuenta con la administración de los catálogos de contactos y proyectos, y la asignación de tareas a un contacto.
- Perfil empleado: El usuario puede visualizar las tareas a realizar como también agregar tareas que realizo.

A continuación se muestran las vistas del sistema:

Página principal del sistema donde se encuentra el menú de Catálogos y Asignaciones (véase en la ilustración 28).



*Ilustración 28 Página de inicio del sistema*

Vista donde se muestra la lista de contactos, como también se puede buscar por los campos que se encuentran en la parte de arriba; basta con capturar y seleccionar el botón buscar (véase en la ilustración 29).

Logo of the Universidad Autónoma del Estado de México (UAEM) and the text "Universidad Autónoma del Estado de México".

Navigation bar: HOME, CATÁLOGOS, ASIGNACIONES, CERRAR SESIÓN: JOSE

Section: Catalogo de contactos

Search fields:

- Nombre: nombre, paterno, materno
- CURP:
- RFC:
- Entidad Federativa: ---SELECCION
- CP:

Buttons: Buscar, Agregar nuevo

Table header:

Nombre	RFC	CURP	Fecha nacimiento	Estado	Acciones
--------	-----	------	------------------	--------	----------

Table content:

(1 of 1) 10

No records found.

(1 of 1) 10

Ilustración 29 Vista de lista de contactos

Vista donde se da de alta un nuevo contacto capturando la información necesaria (véase en la ilustración 30).

The screenshot displays the contact registration interface of the Universidad Autónoma del Estado de México (UAEM). The header features the UAEM logo, the text "Universidad Autónoma del Estado de México", and a circular seal on the right. Below the header is a navigation bar with links: HOME, CATÁLOGOS, ASIGNACIONES, and CERRAR SESIÓN: JOSE. The main form is divided into three sections: "Datos personales", "Acceso al sistema", and "Contactos".

**Datos personales**

Nombre:  Nombre  Apellido paterno  Apellido materno

RFC:  Escribe su direccion  CURP:  Introduce el CURP del ciudadano

Sexo:  ---SELECCIONAR---  Entidad Federativa:  ---SELECCIONAR---

Fecha de nacimiento:  CP:  Escriba su codigo postal

Dirección:

**Acceso al sistema**

Usuario:  identificador de acceso al sistema

**Contactos**

Correo electrónico:  Escribe el correo personal de la persona

**Correos Electrónicos**

Teléfono:  lada  Teléfono de contacto

**Teléfonos**

**Footer:**

Home: HOME

Catálogos: CONTACTOS, PROYECTOS

Asignaciones: TAREAS

Ilustración 30 Vista de alta de contacto

Visita donde se visualizan la lista de proyectos, cabe mencionar que también se puede realizar búsquedas y agregar un nuevo proyecto (véase en la ilustración 31).

Proyectos

Nombre del proyecto:  Estatus:

(1 of 1)



Vista donde se filtran las asignaciones a un proyecto (véase en la ilustración 33).

UAEM | Universidad Autónoma del Estado de México

HOME CATÁLOGOS ASIGNACIONES CERRAR SESIÓN: JOSE

**Tareas**

Descripción de la tarea:

Proyecto:

Estatus:

Prioridad:

(1 of 1) 1-4 < > 10

Proyecto	Persona	Tarea	inicio	termino	Acciones
No records found.					

(1 of 1) 1-4 < > 10

Home Catálogos Asignaciones

Ilustración 33 Lista de asignaciones

Alta a las asignaciones de una tarea en específico para un proyecto (véase en la ilustración 34).

UAEM | Universidad Autónoma del Estado de México

HOME CATÁLOGOS ASIGNACIONES CERRAR SESIÓN: JOSE

**Tareas**

Proyecto:

Contacto:

Prioridad:

Fecha de inicio de la tarea:

Fecha de termino de la tarea:

Síntesis:

Descripción de la tarea:

Porcentaje de avance:

Comentario:

Ilustración 34 Alta de una tarea

## CONCLUSIONES

Implementar el patrón MVC permite dar un mantenimiento fácil a las aplicaciones web debido a una clara separación entre las capas de presentación, lógica de negocio y acceso a datos.

Ventajas del MVC:

- Facilita agregar nuevos tipos de datos según sea requerido por la aplicación ya que son independientes del funcionamiento de las otras capas.
- La separación permite que sea reutilizable.
- Un mantenimiento fácil se traduce en ahorro de tiempo.
- Facilita el mantenimiento en caso de errores.
- Ofrece maneras más sencillas para probar el correcto funcionamiento del sistema.
- Permitir la sustitución de las interfaces de usuario.
- Diseñar vistas simultáneamente del mismo modelo.

La ventaja de utilizar un Framework es que al reducir la codificación se disminuye el tiempo de desarrollo disminuyendo los costos. En el caso de hibernate, este nos permite elegir la base de datos relacional con la queremos trabajar y en dado momento podemos cambiar nuestro motor de base de datos sin que afecte los demás módulos, otra de las ventajas es que varios editores cuentan con un plugin que genera automáticamente el código del mapeo objeto-relacional para ahorrar tiempo.

El uso de patrones de diseño permite separar una aplicación en módulos asignables paralelamente a varias personas.

El patrón MVC garantiza que las secciones asignadas a cada desarrollador sean codificadas de manera profesional para que puedan acoplarse de manera sencilla y transparente y para poder incluir Frameworks y API's que aumente la seguridad y calidad de una aplicación.

## REFERENCIAS

Gamma, E. H., Johnson, R., & Vlissides, J. (2003). *Patrones de diseño*. Madrid: PEARSON EDUCACION S.A. .

<http://docs.spring.io/spring/docs/2.0.x/reference/introduction.html>. (s.f.).

<http://jcalderon.wordpress.com/2008/01/04/instalacion-y-configuracion-de-apache-tomcat-60-en-windows-xp/>. (s.f.).

<http://www.mundoblogweb.com/programadores-java/framework-hibernate.html>. (2012).  
*Framework Para el mapeo de Objetos en Java*.

Jose Annunziato, D. a. (2001). *Sams teach yourself java server pages in 24 hours*.

Olvan Maassen, S. S. (2003). *Patrones de diseño aplicados a Java*. Madrid: Pearson Educacion, S.A.

Prime, O. (s.f.). USER'S GUIDE Primefaces.

# **ANEXOS**

## Anexo A HERRAMIENTAS PARA EL DESARROLLO

### A.1 El entorno de desarrollo eclipse

Requerimientos para la instalación:

JDK: última versión del JDK 6 o 7

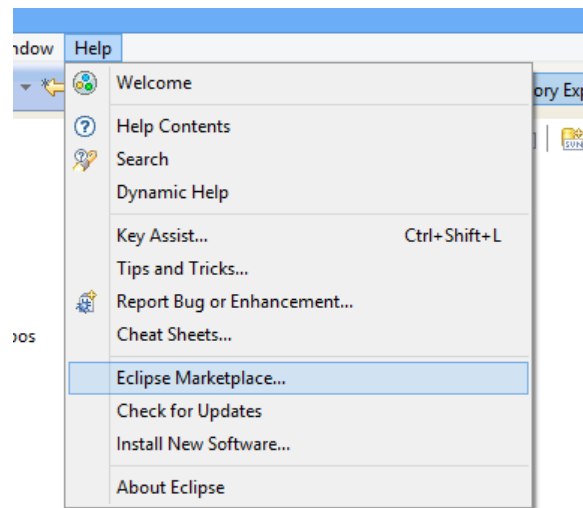
Sistema operativo: Microsoft Windows XP/Vista/7, Apple Mac OS X, Linux.

Hardware: 500MB de espacio libre en disco duro y memoria de 2GB.

Existen 2 formas de descargar la herramienta de Spring- tool-suite 3.2.0 a continuación se listan:

#### A. Instalación a través del Eclipse marketplace

1. Si ya tienes instalado alguna herramienta de eclipse deberás de ir a la sección de Help y posteriormente a eclipse Marketplace (véase en la ilustración 35).



*Ilustración 35 herramienta que permite la búsqueda de un plugin*

2. Posteriormente capturamos la herramienta a buscar (SpringSource tool Suite) y seleccionamos en botón de Go (véase en la ilustración 36).

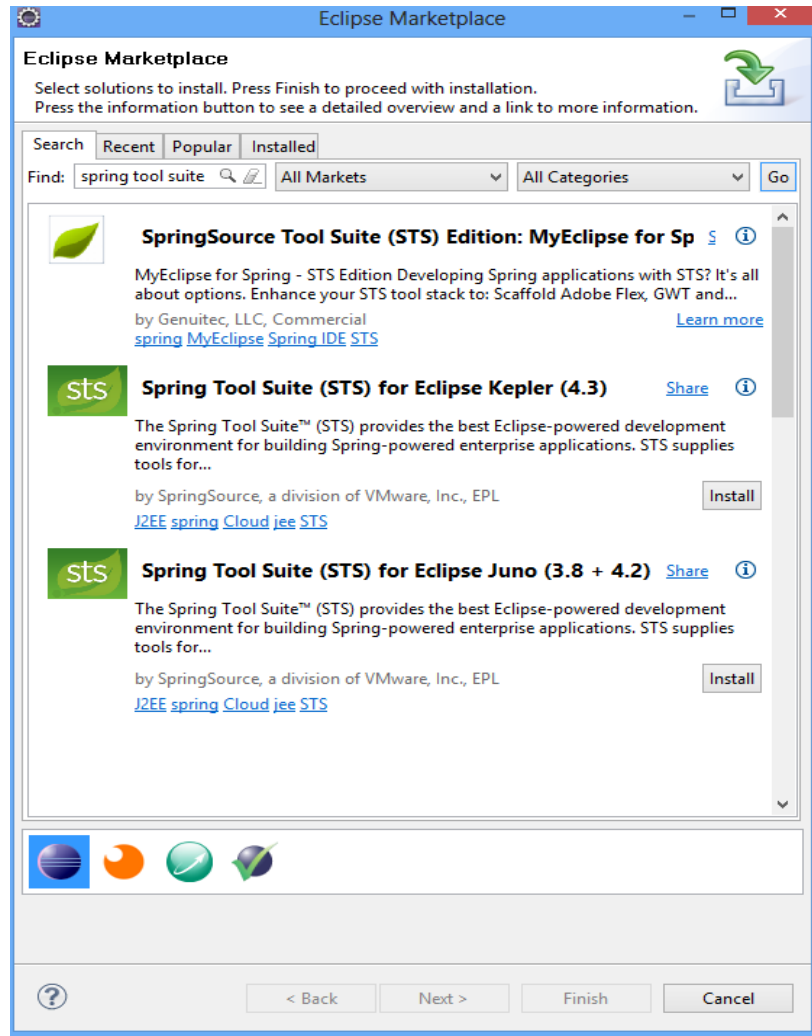


Ilustración 36 Descarga de Spring

3. Seleccionamos botón de Install
- B. Instalar desde los repositorios
1. Ingresar a la página oficial de descargas <http://www.springsource.org/spring-tool-suite-download>.
  2. Descargar springsource-tool-suite 3.2.0 o posterior

## Instalación de Spring-tool-suite

1. Seleccionar Spring tool Suite seleccionar el botón de next (véase en la ilustración 37).

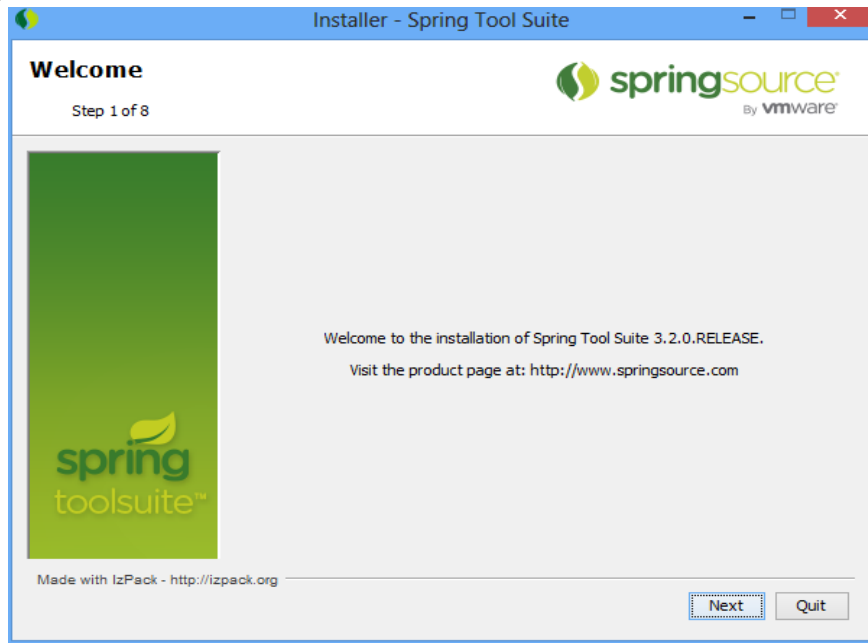


Ilustración 37 Instalando SpringSource

2. Aceptar términos de la licencia, seleccionar botón de next (véase en la ilustración 38).

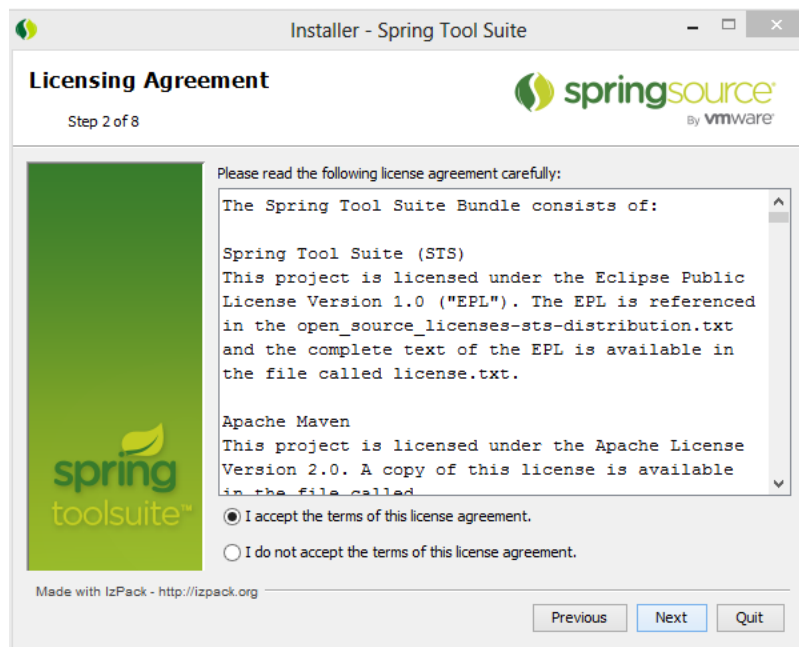


Ilustración 38 Instalando SpringSource

3. Seleccionar el path y posteriormente dar clic en next (véase en la ilustración 39).

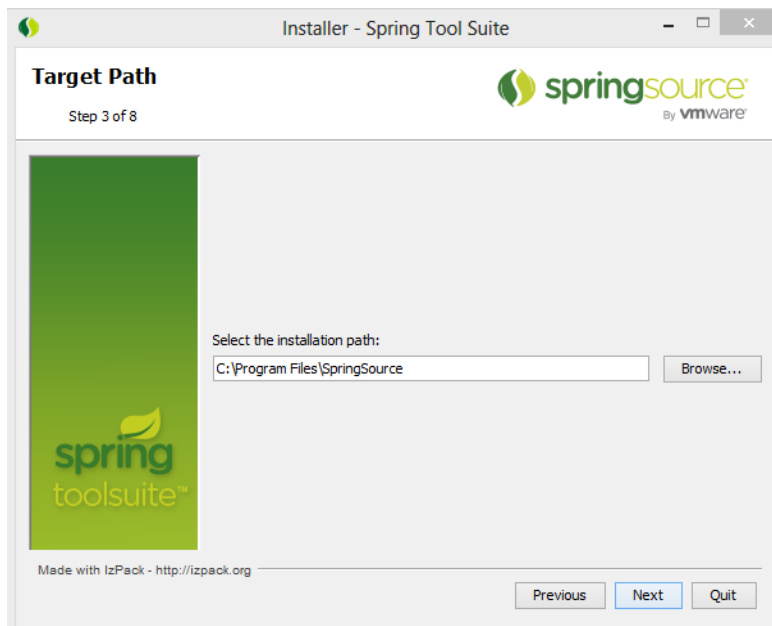


Ilustración 39 Path de Instalación

4. Seleccionar el paquete que vamos a ocupar para el estudio de caso genérico (véase en la ilustración 40).

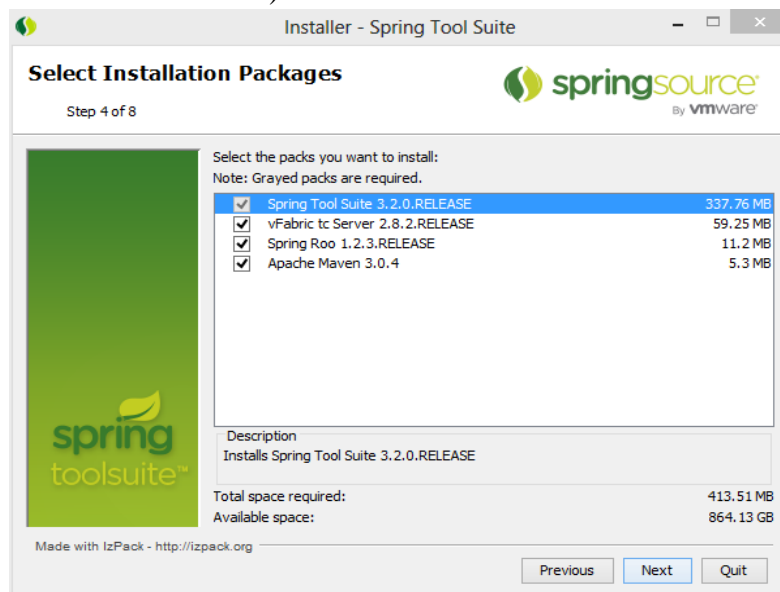
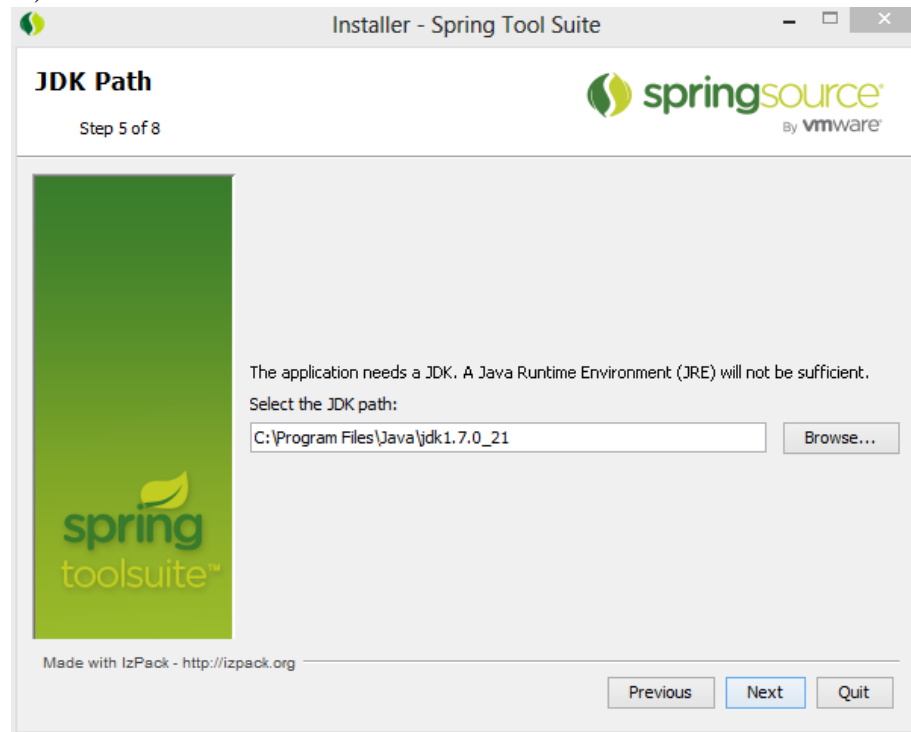


Ilustración 40 Paquetes de instalación



5. Seleccionar el JDK que se va a ocupar y seleccionar next (véase en la ilustración 41).



*Ilustración 41 seleccionar el JDK*

6. Seleccionar botón de next y posteriormente Finish

## A.2 El servidor tomcat

Es un servidor web (http) y funciona como un contenedor de servlets. Es la implementación de referencia de las especificaciones de servlets 2.5 y de Java Server Pages (JSP) 2.1, especificaciones para Java Community Process, usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Apache Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Download:

Java Platform, Standard Edition 6 Development Kit (JDK 6). Debe instalarse previo a la instalación de apache-tomcat.

Apache-tomcat-6.0.14. No descargar la versión que lo instala como un servicio de Windows si se va a usar para desarrollar. Recomiendo Descargar la versión que se instala manualmente descomprimiendo un zip (Windows) o un tar.gz (Linux).

(<http://jcalderon.wordpress.com/2008/01/04/instalacion-y-configuracion-de-apache-tomcat-60-en-windows-xp/>)

Descomprimir el archivo apache-tomcat. (véase en la ilustración 42).

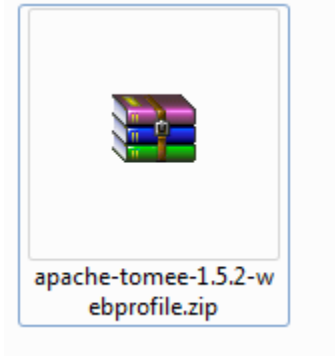


Ilustración 42 Zip de Tomcat

Acceder al directorio donde se ha descomprimido (véase en la ilustración 43).

Nombre	Fecha de modifica...	Tipo	Tamaño
apps	26/09/2013 11:46 a...	Carpeta de archivos	
bin	04/11/2013 03:43 ...	Carpeta de archivos	
conf	25/09/2013 02:06 ...	Carpeta de archivos	
endorsed	20/03/2013 03:14 a...	Carpeta de archivos	
lib	20/03/2013 03:14 a...	Carpeta de archivos	
logs	12/12/2013 11:43 a...	Carpeta de archivos	
temp	12/12/2013 11:43 a...	Carpeta de archivos	
webapps	26/12/2013 03:06 ...	Carpeta de archivos	
work	25/09/2013 02:06 ...	Carpeta de archivos	
LICENSE	20/03/2013 03:14 a...	Archivo	92 KB
NOTICE	20/03/2013 02:51 a...	Archivo	4 KB
RELEASE-NOTES	12/02/2013 09:45 ...	Archivo	9 KB
RUNNING.txt	12/02/2013 09:45 ...	Documento de tex...	17 KB

Ilustración 43 Configuración tomcat

CATALINA\_HOME = Representa la raíz donde se ha instalado apache-tomcat.

Cada uno de los directorios

/bin – arranque, cierre, y otros scripts y ejecutables

/temp – archivos temporales

/conf – archivos XML y los correspondientes DTD para la configuración de apache-tomcat el mas importante es server.xml.

/logs – archivos de registro (log) de apache-tomcat.

/webapps – directorio que contiene las aplicaciones web

/work – almacenamiento temporal de ficheros y directorios

El siguiente paso quizás es uno de los más importantes y es la creación de la variable de entorno JAVA\_HOME.

JAVA\_HOME = directorio del JDK.

Para acceder a la creación de la variable de entorno: Windows + Pausa>Opciones Avanzadas>Variables de Entorno>Nueva (véase en la ilustración 44).

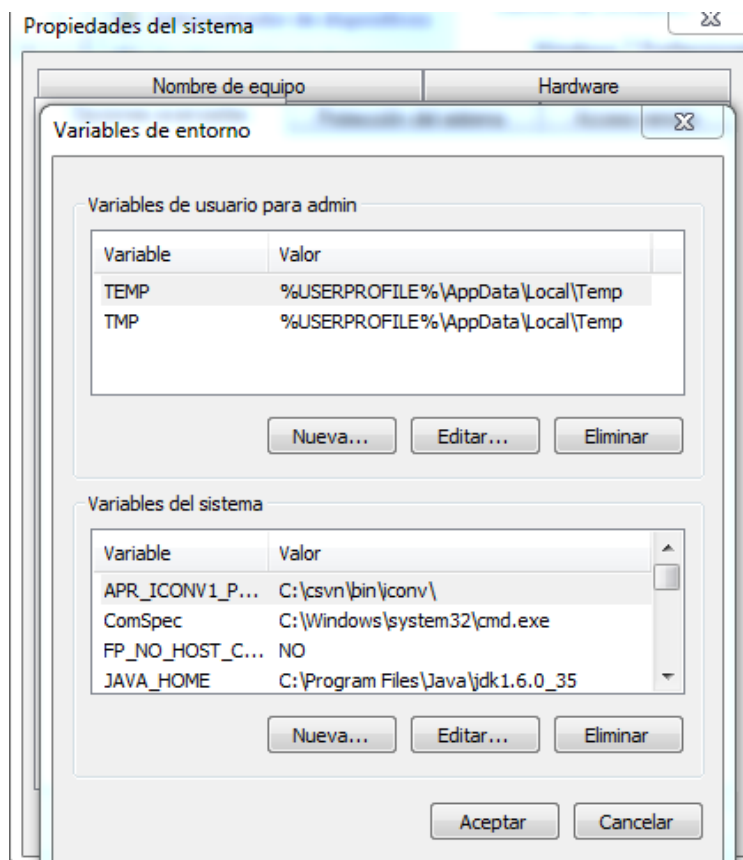


Ilustración 44 Alta de variable de entorno

Existen dos archivos sumamente importantes los cuales nos permitirán iniciar y parar apache-tomcat se encuentra en el siguiente directorio (véase en la ilustración 45).

\$CATALINA\_HOME/bin/startup = para iniciar o arrancar

\$CATALINA\_HOME/bin/shutdown = para parar o detener






	shutdown.bat	12/02/2013 09:45 ...	Archivo por lotes ...	3 KB
	shutdown.sh	12/02/2013 09:45 ...	Archivo SH	2 KB
	startup.bat	12/02/2013 09:45 ...	Archivo por lotes ...	3 KB
	startup.sh	12/02/2013 09:45 ...	Archivo SH	2 KB
	tomcat-iuli.ia	12/02/2013 09:45	Executable Jar File	38 KB

Ilustración 45 compilados del tomcat

Seguidamente abrimos un navegador web y escribimos en el URL.


`http://{host}:{port}/` = donde `{host}``{port}` representa el hostname y el puerto donde corre apache-tomcat, entonces quedaría `http://localhost:8080/` y aparecerá la página de bienvenida de apache-tomcat (véase en la ilustración 46).

Home Documentation Configuration Examples Wiki Mailing Lists Find Help

## Apache Tomcat (TomEE)/7.0.37

The Apache Software Foundation  
<http://www.apache.org/>

If you're seeing this, you've successfully installed Tomcat. Congratulations!

 Recommended Reading:

- [Security Considerations HOW-TO](#)
- [Manager Application HOW-TO](#)
- [Clustering/Session Replication HOW-TO](#)

TomEE Gui  
Server Status  
Manager App  
Host Manager

### Developer Quick Start

- [Tomcat Setup](#)
- [Realms & AAA](#)
- [Examples](#)
- [Servlet Specifications](#)
- [First Web Application](#)
- [JDBC DataSources](#)
- [Tomcat Versions](#)

#### Managing Tomcat

For security, access to the `manager.webapp` is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 7.0 access to the manager application is split between different users. [Read more...](#)

- [Release Notes](#)
- [Changelog](#)
- [Migration Guide](#)
- [Security Notices](#)

#### Documentation

[Tomcat 7.0 Documentation](#)  
[Tomcat 7.0 Configuration](#)  
[Tomcat Wiki](#)

Find additional important configuration information in:

```
$CATALINA_HOME/RUNNING.txt
```

Developers may be interested in:

- [Tomcat 7.0 Bug Database](#)
- [Tomcat 7.0 JavaDocs](#)
- [Tomcat 7.0 SVN Repository](#)

#### Getting Help

##### FAQ and Mailing Lists

The following mailing lists are available:

- [announce@tomcat.apache.org](mailto:announce@tomcat.apache.org)  
Important announcements, releases, security vulnerability notifications. (Low volume).
- [users@tomcat.apache.org](mailto:users@tomcat.apache.org)  
User support and discussion
- [taglibs-user@tomcat.apache.org](mailto:taglibs-user@tomcat.apache.org)  
User support and discussion for [Apache Taglibs](#)
- [dev@tomcat.apache.org](mailto:dev@tomcat.apache.org)  
Development mailing list, including commit messages

#### Other Downloads

- [Tomcat Connectors](#)
- [Tomcat Native](#)
- [Taglibs](#)

#### Other Documentation

- [Tomcat Connectors](#)
- [mod\\_jk Documentation](#)
- [Tomcat Native](#)

#### Get Involved

- [Overview](#)
- [SVN Repositories](#)
- [Mailing Lists](#)

#### Miscellaneous

- [Contact](#)
- [Legal](#)
- [Sponsorship](#)

#### Apache Software Foundation

- [Who We Are](#)
- [Heritage](#)

Ilustración 46 Página principal del tomcat

## Anexo B CREAR UN PROYECTO NUEVO

Para crear un proyecto nuevo es necesario que identifiques la ruta de almacenamiento, el cuadro de dialogo te permite explorar el archivo de directorios para ubicar tu proyecto en la dirección de tu elección

File/Other/Web /Dynamic web (véase en la ilustración 47).

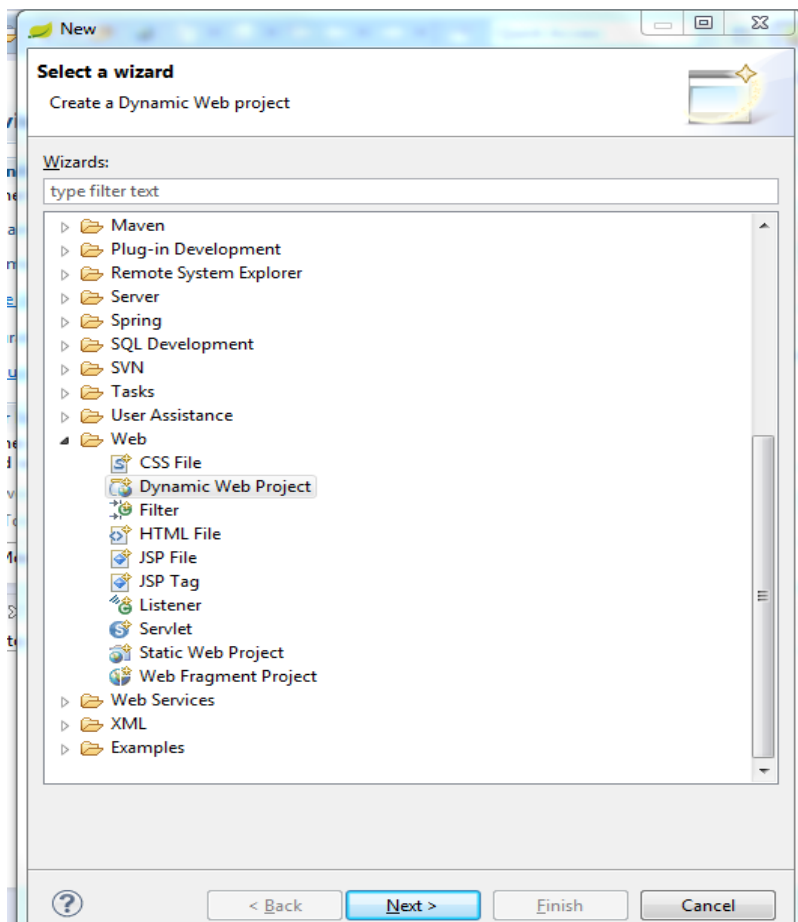


Ilustración 47 Creación de un proyecto Web (Elaboración propia)

Nombre del proyecto y configuración (véase en la ilustración 48).

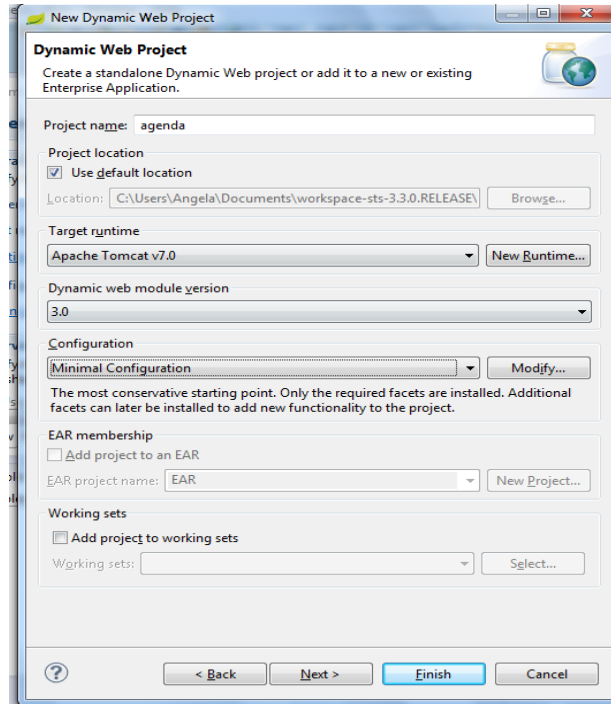


Ilustración 48 Nombre del proyecto Web (Elaboración propia)

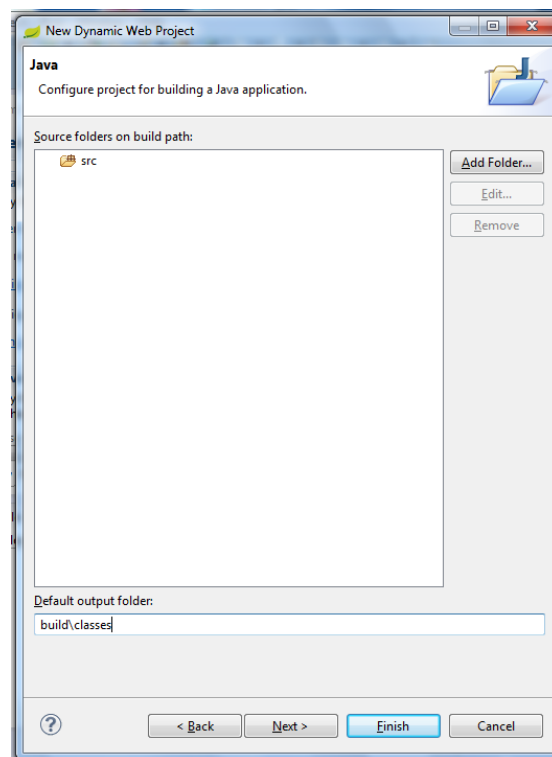
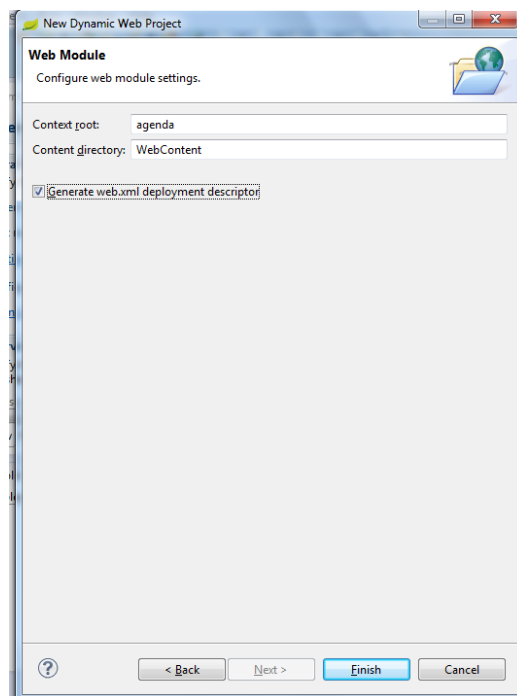
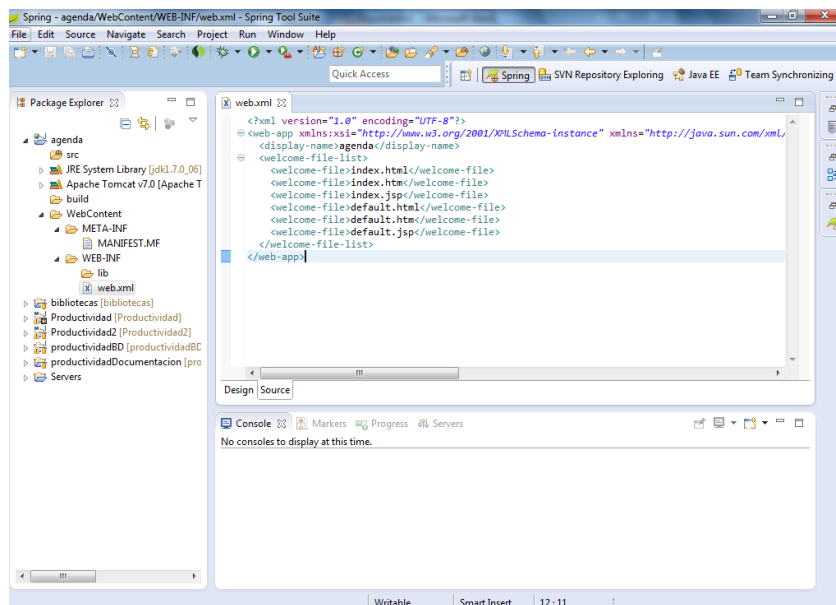


Ilustración 49 Configuración del proyecto (Elaboración propia)



*Ilustración 50 Configuración del proyecto (Elaboración propia)*

Proyecto creado por default (véase en la ilustración 51).



*Ilustración 51 Vista del código del proyecto (Elaboración propia)*

## Anexo C CREAR UN PROYECTO PARA LIBRERÍAS

Creación de un proyecto para guardar las librerías (véase en la ilustración 51).

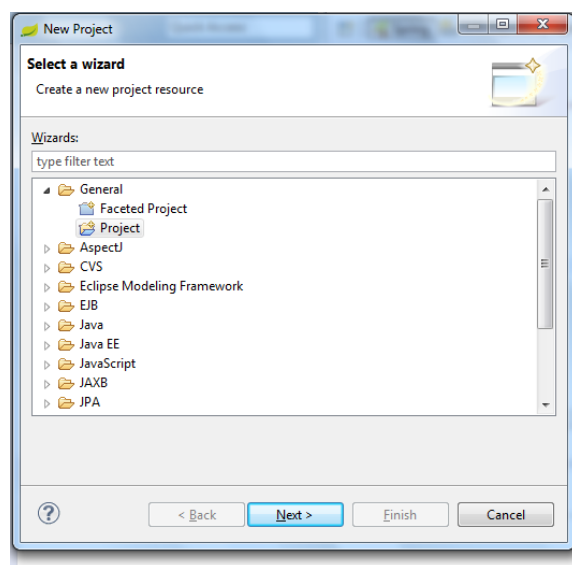


Ilustración 52 Creación de un proyecto para librerías (Elaboración propia)

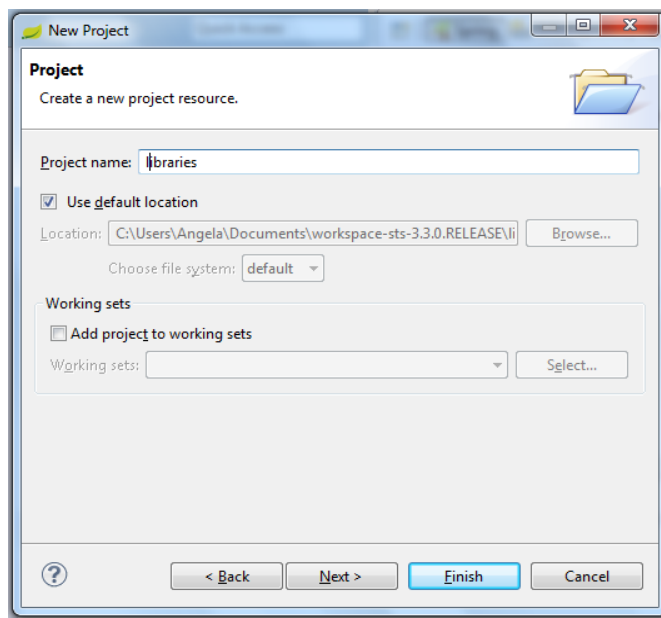


Ilustración 53 Creación de un proyecto para librerías (Elaboración propia)



Para generar el war necesitamos importar las librerías (véase en la ilustración 54).

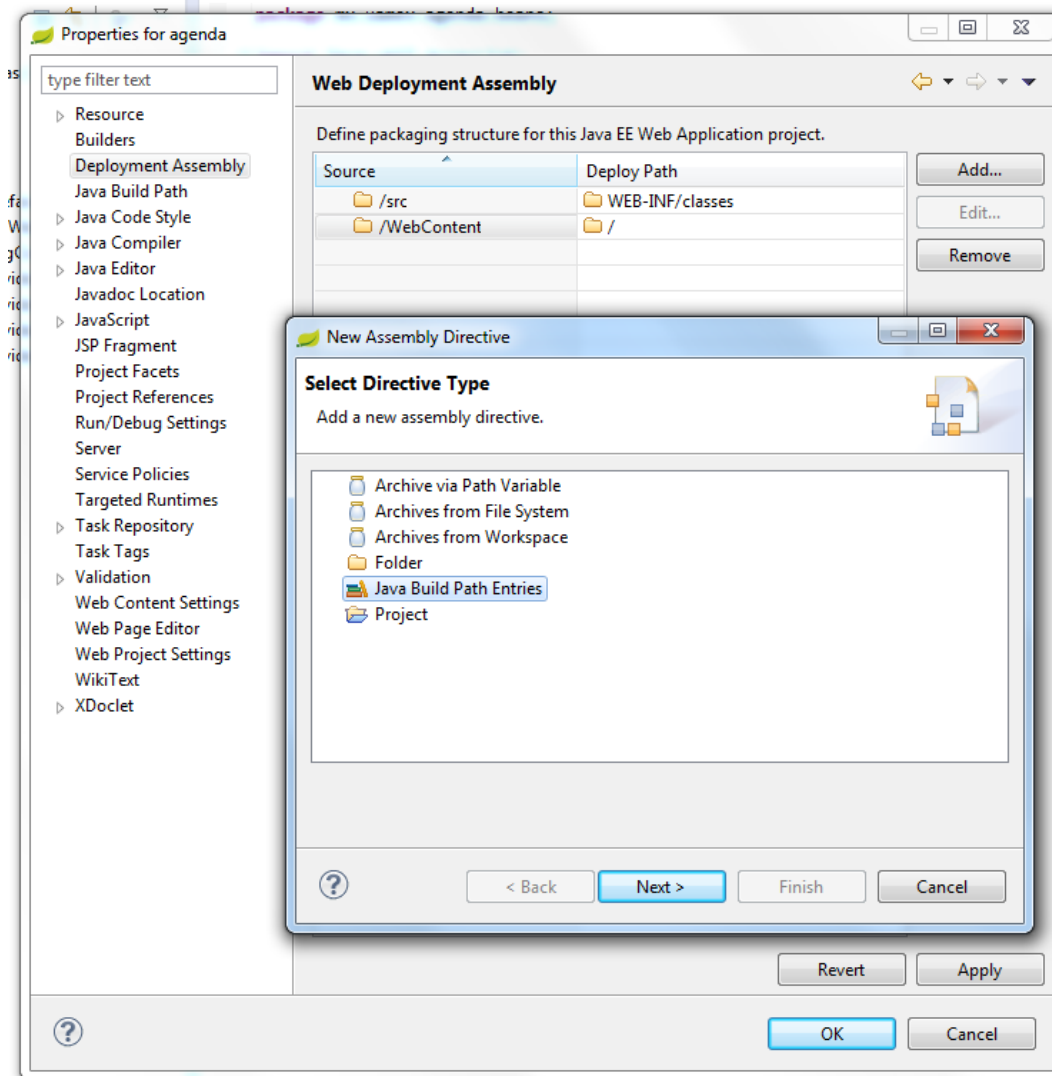


Ilustración 54 Creación de un proyecto para librerías (Elaboración propia)

Lista de bibliotecas requeridas para el módulo de interface grafica

4. el-api-2.2.jar
5. el-impl-2.2.jar
6. el-ri-1.0.jar
7. jsf-api-2.1.13.jar
8. jsf-impl-2.1.13.jar"
9. jsp-api-2.1.jar
10. jstl-1.2.jar
11. servlet-api-2.5.jar
12. javax.inject-1.jar
13. Primefaces/primefaces-3.5.jar
14. SpringWeb/aopalliance-1.0.jar
15. SpringCore/commons-logging-1.1.1.jar
16. springSecurity/aspectjweaver-1.6.10.jar Primefaces/southStreet/south-street-1.0.10.jar
17. spring\_3\_1\_3/org.springframework.aop-3.1.3.RELEASE.jar
18. spring\_3\_1\_3/org.springframework.asm-3.1.3.RELEASE.jar
19. spring\_3\_1\_3/org.springframework.aspects-3.1.3.RELEASE.jar
20. spring\_3\_1\_3/org.springframework.beans-3.1.3.RELEASE.jar
21. spring\_3\_1\_3/org.springframework.context-3.1.3.RELEASE.jar
22. spring\_3\_1\_3/org.springframework.context.support-3.1.3.RELEASE.jar
23. spring\_3\_1\_3/org.springframework.core-3.1.3.RELEASE.jar
24. spring\_3\_1\_3/org.springframework.expression-3.1.3.RELEASE.jar
25. spring\_3\_1\_3/org.springframework.instrument-3.1.3.RELEASE.jar
26. spring\_3\_1\_3/org.springframework.instrument.tomcat-3.1.3.RELEASE.jar
27. spring\_3\_1\_3/org.springframework.jdbc-3.1.3.RELEASE.jar
28. spring\_3\_1\_3/org.springframework.jms-3.1.3.RELEASE.jar
29. spring\_3\_1\_3/org.springframework.orm-3.1.3.RELEASE.jar
30. spring\_3\_1\_3/org.springframework.test-3.1.3.RELEASE.jar
31. spring\_3\_1\_3/org.springframework.transaction-3.1.3.RELEASE.jar
32. spring\_3\_1\_3/org.springframework.web-3.1.3.RELEASE.jar
33. spring\_3\_1\_3/spring-tx-3.1.3.RELEASE.jar
34. springSecurity/spring-security-config-3.1.2.RELEASE.jar
35. springSecurity/spring-security-core-3.1.2.RELEASE.jar
36. springSecurity/spring-security-web-3.1.2.RELEASE.jar
37. hibernate\_4\_1\_9/optional/c3p0/c3p0-0.9.1.jar"
38. hibernate\_4\_1\_9/required/antlr-2.7.7.jar
39. hibernate\_4\_1\_9/required/dom4j-1.6.1.jar
40. hibernate\_4\_1\_9/required/hibernate-commons-annotations-4.0.1.Final.jar
41. hibernate\_4\_1\_9/required/hibernate-core-4.1.9.Final.jar
42. hibernate\_4\_1\_9/required/hibernate-jpa-2.0-api-1.0.1.Final.jar

- 43. hibernate\_4\_1\_9/required/javassist-3.17.1-GA.jar
- 44. hibernate\_4\_1\_9/required/jboss-logging-3.1.0.GA.jar
- 45. hibernate\_4\_1\_9/required/jboss-transaction-api\_1.1\_spec-1.0.0.Final.jar
- 46. mysql/mysql-connector-java-5.1.19-bin.jar

Lista de bibliotecas requeridas para el módulo de acceso a la base de datos

- 1. hibernate\_4\_1\_9/required/antlr-2.7.7.jar
- 2. hibernate\_4\_1\_9/required/dom4j-1.6.1.jar
- 3. hibernate\_4\_1\_9/required/hibernate-commons-annotations-4.0.1.Final.jar
- 4. hibernate\_4\_1\_9/required/hibernate-core-4.1.9.Final.jar
- 5. hibernate\_4\_1\_9/required/hibernate-jpa-2.0-api-1.0.1.Final.jar
- 6. hibernate\_4\_1\_9/required/javassist-3.17.1-GA.jar
- 7. hibernate\_4\_1\_9/required/jboss-logging-3.1.0.GA.jar
- 8. hibernate\_4\_1\_9/required/jboss-transaction-api\_1.1\_spec-1.0.0.Final.jar
- 9. hibernate\_4\_1\_9/optional/c3p0/c3p0-0.9.1.jar
- 10. spring\_alliance aop/com.springsource.org.aopalliance-1.0.0.jar
- 11. spring\_3\_1\_3/org.springframework.aop-3.1.3.RELEASE.jar
- 12. spring\_3\_1\_3/org.springframework.asm-3.1.3.RELEASE.jar
- 13. spring\_3\_1\_3/org.springframework.aspects-3.1.3.RELEASE.jar
- 14. spring\_3\_1\_3/org.springframework.beans-3.1.3.RELEASE.jar
- 15. spring\_3\_1\_3/org.springframework.context-3.1.3.RELEASE.jar
- 16. spring\_3\_1\_3/org.springframework.context.support-3.1.3.RELEASE.jar
- 17. spring\_3\_1\_3/org.springframework.core-3.1.3.RELEASE.jar
- 18. spring\_3\_1\_3/org.springframework.expression-3.1.3.RELEASE.jar
- 19. spring\_3\_1\_3/org.springframework.instrument-3.1.3.RELEASE.jar
- 20. spring\_3\_1\_3/org.springframework.instrument.tomcat-3.1.3.RELEASE.jar
- 21. spring\_3\_1\_3/org.springframework.jdbc-3.1.3.RELEASE.jar
- 22. spring\_3\_1\_3/org.springframework.jms-3.1.3.RELEASE.jar
- 23. spring\_3\_1\_3/org.springframework.orm-3.1.3.RELEASE.jar
- 24. spring\_3\_1\_3/org.springframework.test-3.1.3.RELEASE.jar
- 25. spring\_3\_1\_3/org.springframework.transaction-3.1.3.RELEASE.jar
- 26. spring\_3\_1\_3/spring-tx-3.1.3.RELEASE.jar
- 27. mysql/mysql-connector-java-5.1.19-bin.jar
- 28. commons/commons-logging-1.1.1.jar

## Agregar bibliotecas a un proyecto

Click derecho en el proyecto y seleccionar la opción propiedades. Seleccionar **“java Build Path”** (véase en la ilustración 55).

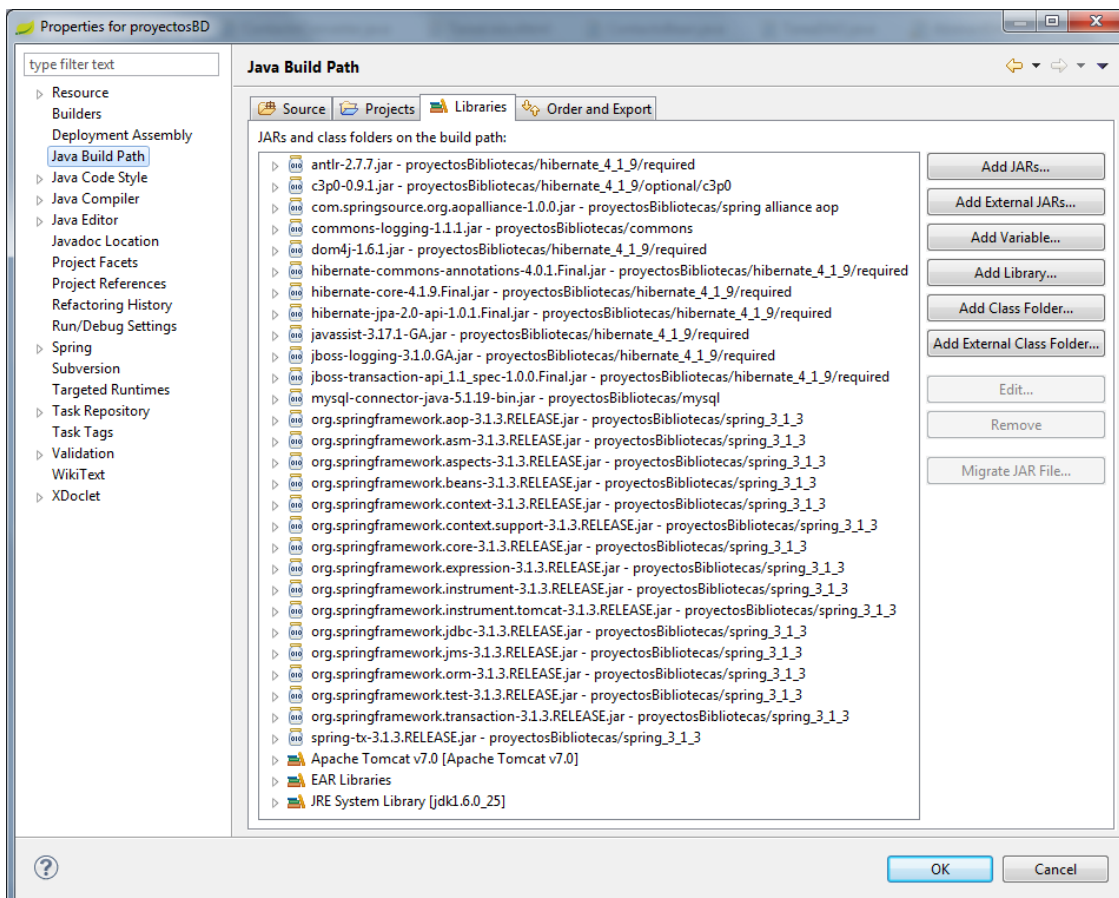


Ilustración 55 Configuración de librerías

En esta ventana se listan las bibliotecas agregadas al proyecto. Con el botón Add Jar´s se agregan los archivos.

Las bibliotecas agregadas de este modo son visibles para el editor lo que permite utilizar sus clases en nuestro programa, pero no son incluidas cuando la aplicación terminada es exportada como archivo jar o war. Esto debido a que si varias aplicaciones de escritorio utilizan las mismas bibliotecas estas pueden agregarse al CLASSPATH de la máquina virtual y compartirse lo que disminuye el tamaño de las aplicaciones.

En el caso de las aplicaciones web tanto el servidor “Tomcat” como “Glassfish” ya incluyen las bibliotecas de java EE e igualmente se pueden compartir bibliotecas para ahorrar espacio. Esto se vuelve relevante cuando una aplicación es cargada en un servidor con memoria limitada o en el que el costo se incrementa en función a la memoria utilizada en el servidor.

Sin embargo, cuando el tamaño de las aplicaciones no representa una limitante, se establece que los archivos jar sean incluidos dentro del proyecto en la sección “**Deployment assembly**” de las propiedades de la aplicación (véase en la ilustración 56).

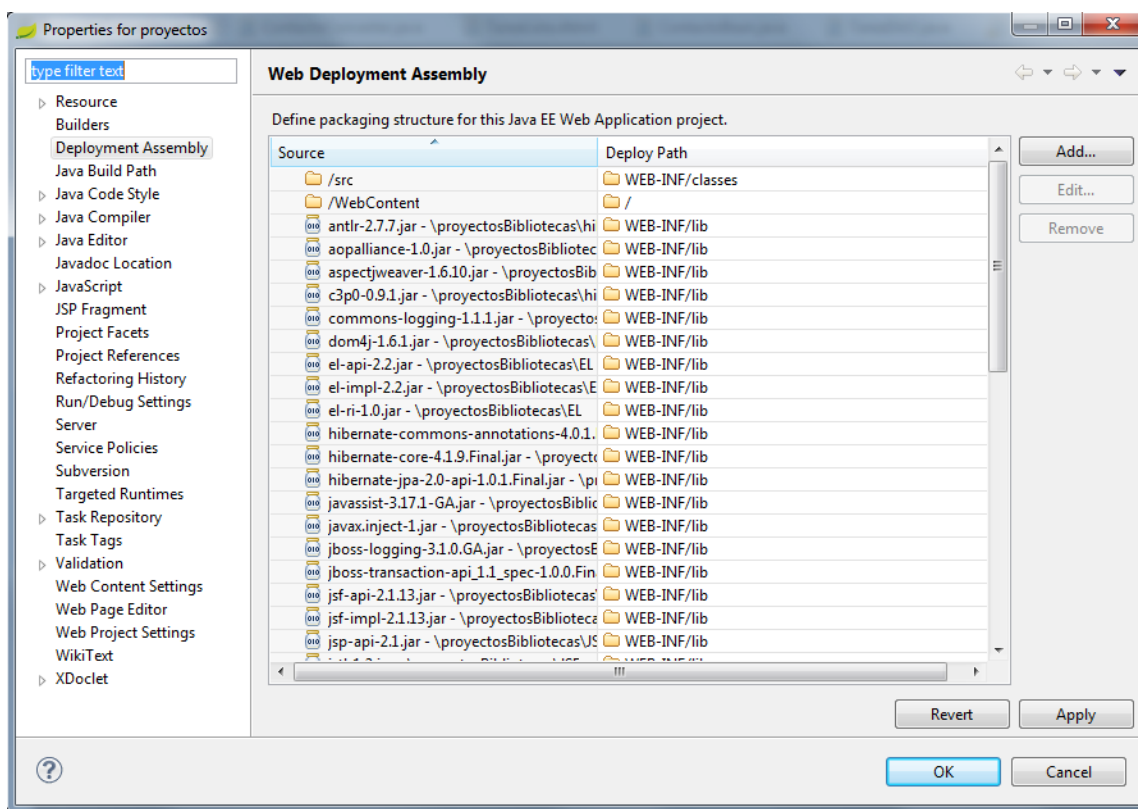


Ilustración 56 Deployment Assembly

En esta sección se agregan los archivos que serán exportados dentro de la aplicación con el botón “add”.

Las bibliotecas agregadas y exportadas se reflejan en el archivo “classpath” que se encuentra en el directorio raíz de la aplicación.