UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

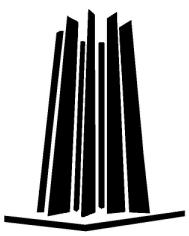
FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

"DESARROLLO DE SISTEMAS PARA PEQUEÑAS Y MEDIANAS EMPRESAS UTILIZANDO EL FRAMEWORK STRUTS"

T R A B A J O E S C R I T O

EN LA MODALIDAD DE SEMINARIOS
Y CURSOS DE ACTUALIZACIÓN Y
CAPACITACIÓN PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A :

M A N U E L S A L A Z A R
C A R R I L L O



ASESOR: M. EN I. ARCELIA BERNAL DÍAZ

MÉXICO, 2009.

DESARROLLO DE SISTEMAS PARA PEQUEÑAS Y MEDIANAS EMPRESAS UTILIZANDO EL FRAMEWORK STRUTS

INTRODUCCIÓN

En la actualidad existen pequeñas y medianas empresas (PyMES) dedicadas a la prestación de bienes y servicios diversos, cada una con diferentes necesidades específicas para llevar acabo de una forma más eficiente el control de las diferentes áreas de las cuales está compuesta una empresa.

Dentro de estas áreas tenemos el área de abastecimiento, que es la encargada de todo lo que se refiere a la obtención productos e insumos para transformarlos en nuevos bienes o para su redistribución posterior, también es encargada del almacenamiento de estos productos e insumos y del control de inventario del almacén donde se guardan dichos productos e insumos.

Las empresas actuales para poder sobresalir están en constante desarrollo, siempre buscando la forma de ser más productivas, automatizando sus áreas con nueva maquinaria, equipo de cómputo y ayuda de tecnologías de información. Como ejemplo tenemos a la empresa denominada Herrajes Benítez S.A. de C.V., la cual se dedica a la fabricación de piezas de lámina para el ensamblado de literas y cabeceras de camas. Dentro de los productos que elaboran están las virolas que sirven de adorno a las literas, esquinas que sirven para ensamblar las literas con las cabeceras, las grapas que son utilizadas para fijar las mallas de las literas con sus bases, entre otros productos. Para llevar un control más eficiente de su almacén donde guardan sus productos y los materiales necesarios para su elaboración, desean un sistema que les permita registrar entradas y salidas de productos y materiales; así como también un control de inventario que les permita saber de forma rápida y eficiente cuanto tienen en su almacén o cuanto a salido de él.

Actualmente en la empresa Herrajes Benítez S.A. de C.V. no cuenta con ningún tipo de control de inventario lo que ocasiona pérdidas de material y de productos, por lo tanto se requiere un sistema que permita tener un control de entradas y salidas donde se registren los productos, cantidad y quien registra la entrada o la salida del almacén, también tenemos el control de inventario que servirá para saber qué cantidad y qué tipo de productos fueron introducidos o sacados en un cierto lapso de tiempo.

Para resolver el problema de la empresa denominada Herrajes Benítez S.A. de C.V., se realizará un sistema de almacén que permitirá controlar entradas/salidas de productos y materiales; además de llevar a cabo el inventario de forma automática.

Para realizar este sistema de almacén recurriremos a las tecnologías de información de la cuales forma parte el lenguaje de programación Java. Este lenguaje de programación es muy versátil y flexible y con una gran cantidad de herramientas que nos permite hacer cualquier tipo de aplicación para resolver diversos problemas.

Este sistema de almacén podrá ser accesado en red desde diferentes máquinas al mismo tiempo lo cual se denomina "aplicación multiusuario"; para conseguir esto se utilizará una computadora con un servidor de aplicaciones Web conocido como Tomcat que permitirá tener levantado el sistema para que sea accedido por los usuarios; además para guardar los usuarios, productos y registros de entradas y salidas necesitaremos una base de datos y un manejador de base de datos (DBMS), por lo tanto utilizaremos para este sistema el DBMS MySQL[1].

Otra tecnología que se utilizará es el marco de trabajo (Framework) STRUTS que permite desarrollar aplicaciones Web con tecnología Java de una forma más eficiente y rápida, este marco de trabajo es muy popular ya que utiliza la metodología de desarrollo MVC[2] (Modelo-Vista-Controlador), donde hay una separación entre la vista que se le presenta al usuario y la lógica de negocio que se necesita implementar en el sistema, esto hace a los sistemas

[1] Véase DBMS MySQL pag.23
[2] Véase MVC pag. 41

susceptibles a modificaciones más rápidas por no existir tanta dependencia entre los elementos que conforman el sistema.

A través de todas estas herramientas mencionadas se pretende realizar un sistema integral, práctico y útil para la empresa Herrajes Benítez S.A. de C.V. que cubrirá con sus necesidades de automatización y control de su almacén.

En resumen necesitamos desarrollar un sistema que nos permita controlar las entradas y salidas de productos y materiales así como la realización de inventarios automatizados para la empresa denominada Herrajes Benítez S.A. de C.V.

Esto lo conseguiremos obteniendo los requisitos funcionales, analizando y diseñando un sistema de información que permita a la empresa cubrir sus necesidades de automatización de almacén a través del Lenguaje Unificado de Modelado (UML).

Desarrollando un sistema eficiente y útil que permita controlar entradas, salidas e inventarios del almacén de la empresa utilizando el lenguaje Java.

Para cubrir la necesidad de un sistema accesible por diferentes usuarios al mismo tiempo integraremos el lenguaje Java y las tecnologías Servlet y JSP. Todo esto dentro del FrameWork STRUTS que implementa el MVC, creando de esta forma un sistema que podrá ser modificable y extensible.

Este trabajo está estructurado en cinco capítulos, donde el capitulo número uno abarca la principal herramienta que nos permitirá modelar el sistema la cual es el lenguaje unificado de modelado (UML), este lenguaje nos permitirá realizar los distintos diagramas que servirán en la construcción del sistema.

El capitulo dos abarca el lenguaje de programación Java, el cual es la base de las tecnologías Servlet y JSP.

En el capitulo tres se da una breve descripción de las tecnologías Servlet y JSP que son la base para el manejo del FrameWork Struts

Dentro del capítulo cuatro se expone el funcionamiento del Framework Struts y la construcción de los diversos componentes que integran la lógica del Modelo-Vista-Controlador en el cual está basado este FrameWork.

Dentro del capítulo cinco esta la documentación del sistema Product Warehouse, donde se ponen en práctica todos los conocimientos expuestos en los capítulos anteriores.

Finalmente se incluyen las conclusiones del uso de estas tecnologías para el desarrollo de sistemas y la bibliografía consultada en la elaboración de este trabajo.

1.- EL LENGUAJE UNIFICADO DE MODELADO (UML).

1.1.- Introducción a UML

El Lenguaje Unificado de Modelado (*UML - Unified Modeling Language por sus siglas en ingles*) es un conjunto de diagramas que nos ayudan a modelar sistemas de información. Estos diagramas cubren los tres niveles en los que puede representarse un sistema de información: el conceptual, de especificación y de implementación.

Los dos últimos niveles, el de especificación y el de implementación tienen una fuerte conexión con el código fuente, de hecho se pretende que los diagramas de especificación lleguen a convertirse en código fuente, los diagramas del nivel de especificación y de implementación tienen una notación muy formal evitando así ambigüedad en los conceptos que representan, por otra parte los diagramas de nivel conceptual representan el dominio del problema humano y tienen una nomenclatura menos formal o estricta, esto provoca cierta ambigüedad, pero a través del refinamiento de los diagramas subsecuentes se pueden diseñar sistemas de una forma muy eficiente.

1.2.- Diagramas de contexto

El diagrama de contexto es el más general de todos, en él se delimita el sistema que se quiere diseñar y se identifican los actores que intervendrán en él, de esta forma se sabe que alcance tendrá el sistema, quienes interactuarán con el sistema y las vías de comunicación existentes hacia él.

Los actores son los elementos externos al sistema que se desea diseñar, consisten en usuarios humanos, administradores, bases de datos, impresoras y hasta otros sistemas que podrían interactuar con el nuestro, *ver Figura 1.1.*

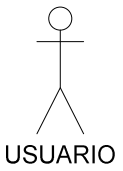


Figura 1.1
Representación Grafica de un actor en el lenguaje UML

El sistema se puede representar como una caja negra que iremos diseñando con los siguientes diagramas, sirviendo esta caja para delimitar nuestro sistema y ver que actores interactuarán con ella, *ver Figura 1.2.*

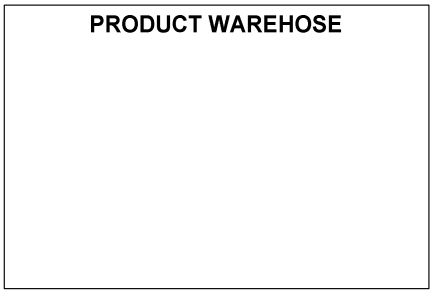


Figura 1.2
Representación Grafica de la delimitación de un sistema en el lenguaje UML

Las vías de comunicación pueden ser simples líneas que representen la unión al sistema ya que el sentido de comunicación entre el sistema y los actores puede ser bidireccional, *ver Figura 1.3*

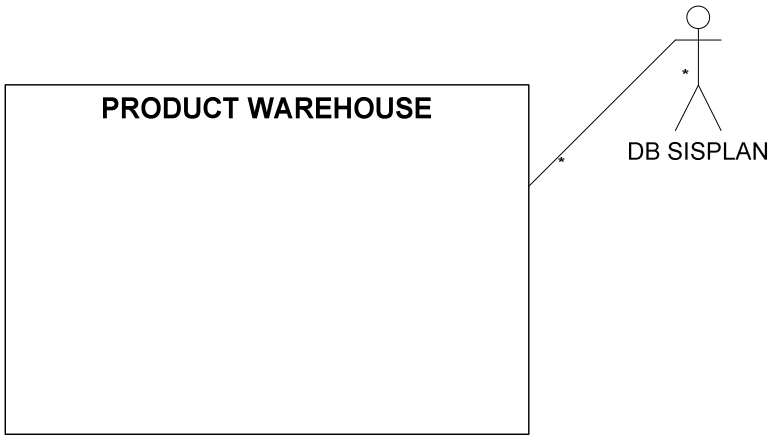


Figura 1.3
Ejemplo de conexión de un sistema con un actor en el lenguaje UML

El siguiente ejemplo de diagrama de contexto delimita el sistema "Product WareHouse" que consiste en un sistema de almacén. En este diagrama se muestran los diversos actores que interactuarán con él, *ver Figura 1.4.*

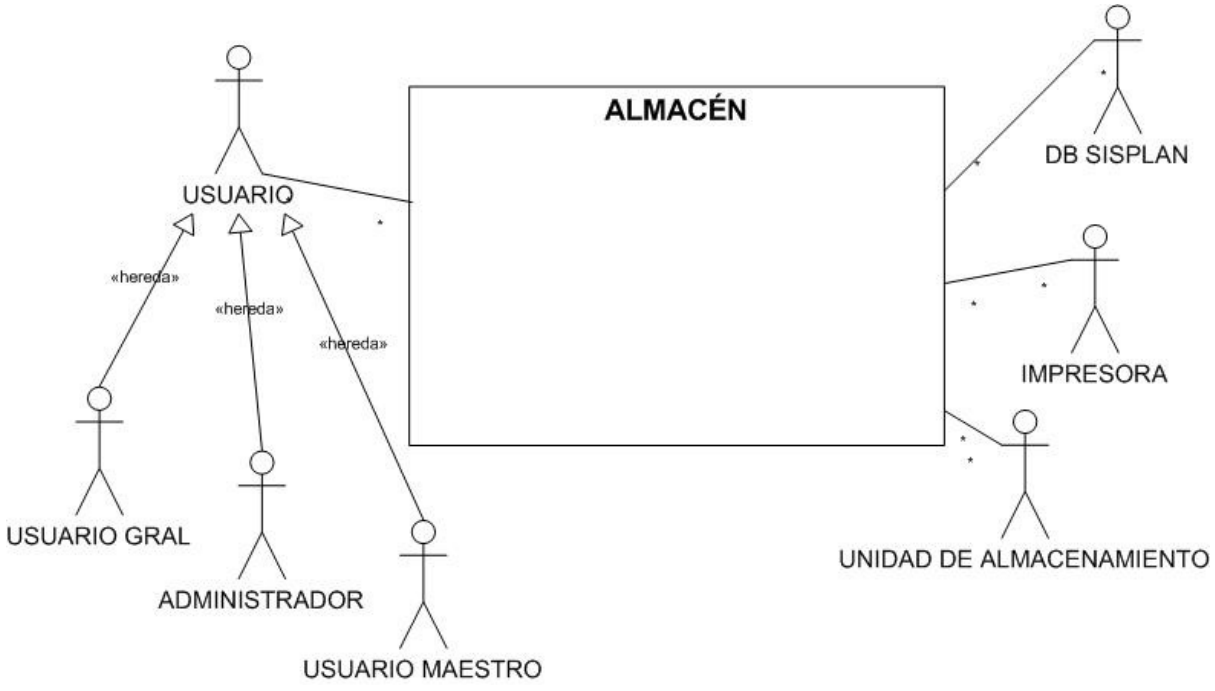


Figura 1.4

Diagrama de contexto para el sistema "PRODUCT WAREHOUSE" utilizando el lenguaje UML

1.3.- Diagramas de casos de uso

Los diagramas de caso de uso son la base del sistema, ya que en estos diagramas se describe la funcionalidad del sistema, como un diagrama de estado donde el usuario inicia una acción o una tarea, esta pasa de un estado a otro realizando otra tarea y así sucesivamente siguiendo el flujo principal o los sub-flujos provocados por tareas adicionales o errores.

Todo este proceso es visto, desde el punto de vista del usuario, aquí no se describe el comportamiento interno de los procesos, solo se describe el flujo de acciones que puede seguir el sistema a partir de un estímulo proporcionado por un actor.

Como en el diagrama de contexto, los actores son los elementos externos al sistema que se desea diseñar y consisten en usuarios humanos, administradores, bases de datos, impresoras e incluso otros sistemas que podrían interactuar con el nuestro, *ver Figura 1.5.*

Figura 1.5

Representación Grafica de un actor en el lenguaje UML

Los casos de uso representan acciones o procesos que debe realizar el sistema a través de un estímulo, todo desde el punto de vista del actor, *ver Figura 1.6.*

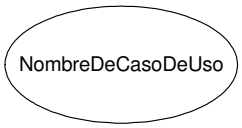


Figura 1.6

Representación Grafica de un caso de uso en el lenguaje UML

La comunicación entre los casos de uso y los actores se hace mediante líneas simples ya que existe una comunicación bidireccional, *ver Figura 1.7.*

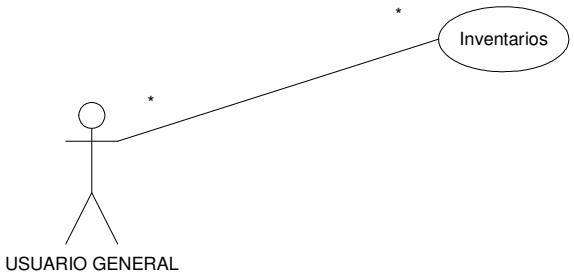


Figura 1.7

Ejemplo de conexión de actor con un caso de uso en el lenguaje UML

La conexión entre un caso de uso se describe con líneas y se pone en la parte de arriba que tipo de relación tiene, existen dos tipos de relaciones entre casos de uso, la inclusión y la extensión.

La extensión es la forma en la que un caso de uso se inserta en otro para agregarle funcionalidad extra, el caso de uso al que se inserta funcionalidad es aquel que forma parte del flujo principal de acciones iniciadas por un actor u otro caso de uso, *ver Figura 1.8.*

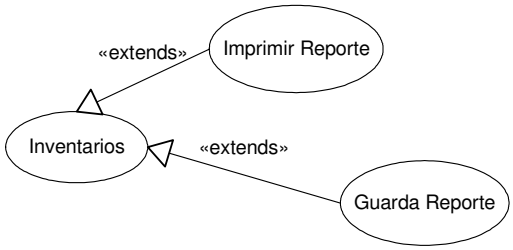


Figura 1.8

Ejemplo de conexión de extensión entre casos de uso en el lenguaje UML

La relación de inclusión permite la reutilización de casos de uso que dentro del sistema tienden a repetirse, de esta forma este caso de uso es llamado cada que es requerido por algún otro caso de uso, *ver Figura 1.9.*

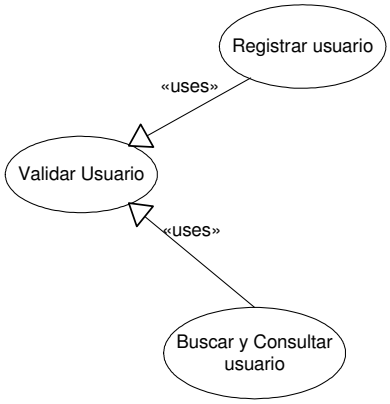


Figura 1.9

Ejemplo de conexión de inclusión entre casos de uso en el lenguaje UML

Los diagramas de caso de uso son muy útiles para describir las actividades que debe de realizar el sistema, su flujo principal y sus sub-flujos, manteniéndolo en un nivel de complejidad bajo se le considera más útil, por otra parte teniendo nuestro diagrama de casos de uso completo hay que elaborar la documentación de cada caso de uso, describiendo que actores intervienen cual es su flujo principal, sus sub-flujos y excepciones.

Un ejemplo de un diagrama de casos de uso es el del sistema "Product WareHouse". *Ver Figura 1.10.*

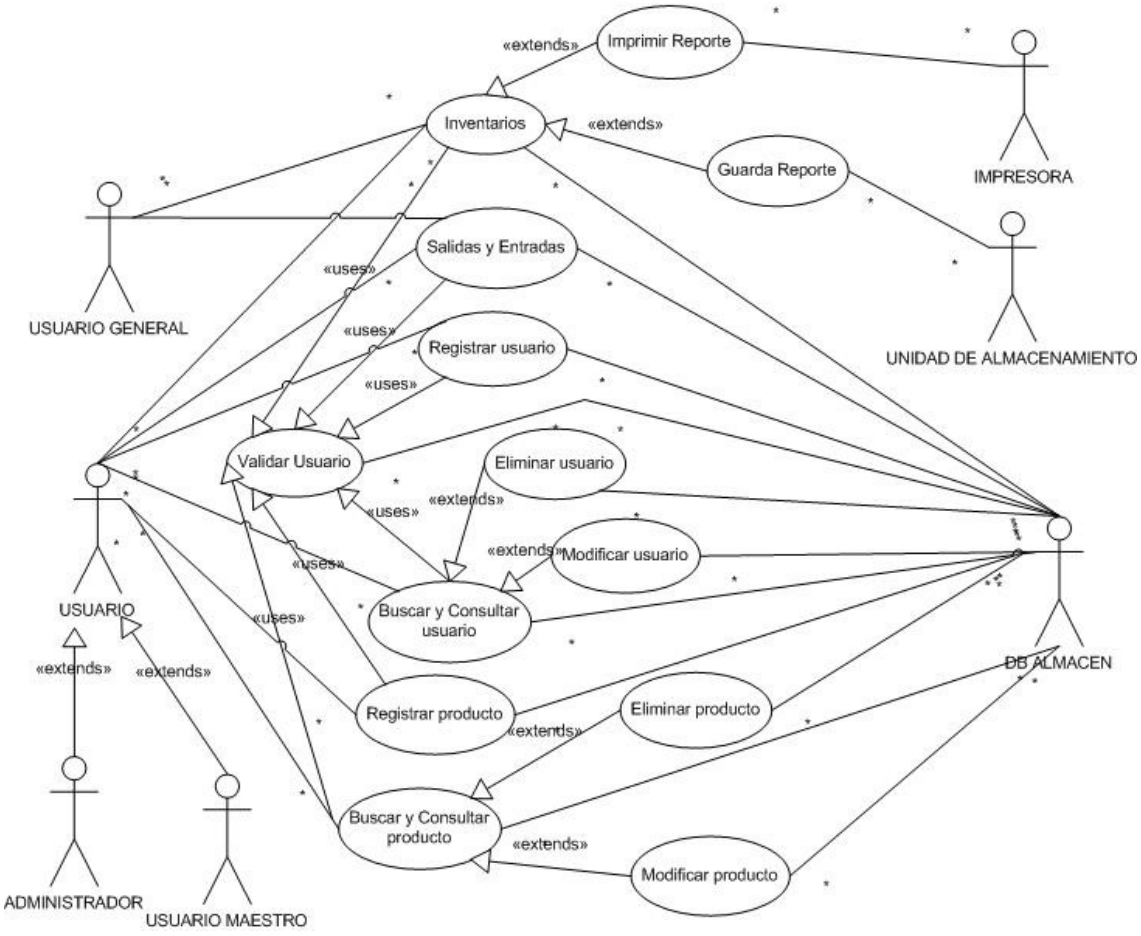


Figura 1.10

Ejemplo de diagrama de casos de uso para el sistema "PRODUCT WAREHOSE" utilizando el lenguaje UML

1.4.- Diagramas de clases

Los diagramas de clases con todo el detalle que se puede lograr con ellos describen en forma estática como van a interactuar las clases entre si dentro del sistema, estos diagramas se obtienen a partir de la documentación de los casos de uso y de la descripción de nuestro problema, este tipo de diagramas con todos sus elementos nos dan la información suficiente para convertirlas en código fuente, dejando la comunicación entre las clases a los diagramas de secuencia que se verán más adelante.

Un diagrama de clases se compone de varios elementos pero la forma más sencilla de representar una clase es mediante su nombre dentro de un recuadro, *ver Figura 1.11.*

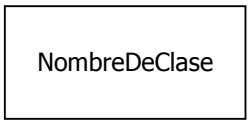


Figura 1.11
Representación básica de una clase utilizando el lenguaje UML

A las clases puede agregársele un estereotipo el cual nos indicará qué función tiene nuestra clase, mediante la utilización del modelo-vista-controlador (MVC)[1] podemos diferenciar entre tres tipos de clases: entidad o modelo, vista y control, *ver Figura 1.12.*

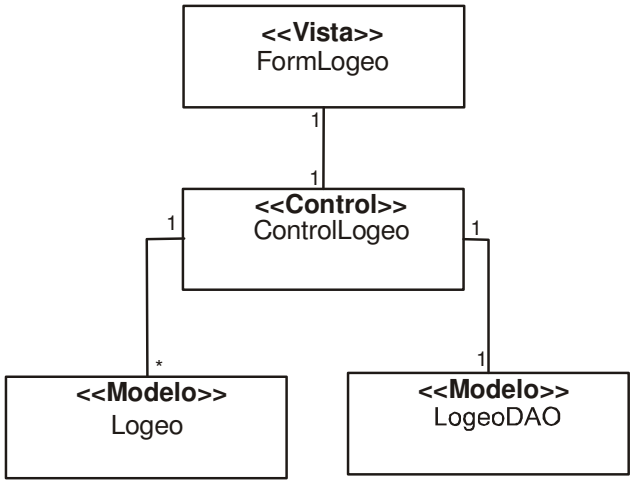


Figura 1.12
Representación básica de una clase utilizando el lenguaje UML

Otros dos elementos que se le pueden agregar a los diagramas de clase son los compartimentos para agregar las variables de la clase, que van justo debajo del recuadro del nombre de la clase y el compartimiento para agregar los métodos de la clase, que van debajo del recuadro de las variables.

El carácter delante de las variables y de las funciones indica el nivel de visibilidad, donde: (-) indica que es privado, (#) indica protegido y (+) indica que es público, *ver Figura 1.13.*

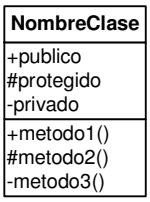


Figura 1.13

Representación una clase con atributos y métodos utilizando el lenguaje UML

[1] Véase MVC pag: 41

Para comunicar las clases entre sí tenemos varias formas, las más importantes son la asociación simple y la herencia.

La asociación representa variables en instancias que mantienen una referencia a otros objetos donde se puede indicar el número de referencias existentes en el caso de (*) significa muchos, *ver Figura 1.14.*

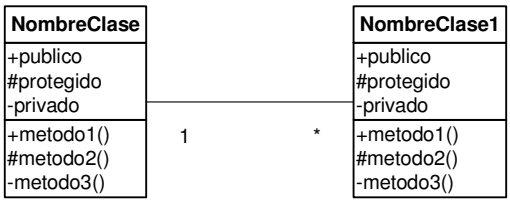


Figura 1.14

Ejemplo de asociación entre clases utilizando el lenguaje UML

La herencia por otra parte se representa con una flecha indicándole que clase hereda de cual, *ver Figura 1.15.*

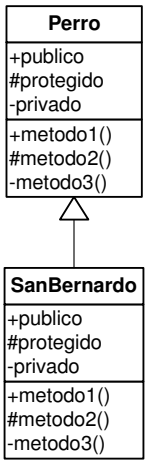


Figura 1.15

Ejemplo de asociación entre clases utilizando el lenguaje UML

1.5.- Diagramas de secuencia

Los diagramas de secuencia representan un modelo dinámico de las clases interactuando entre sí y con los actores, mandándose mensajes para la comunicación, así se puede describir el flujo que tendrán nuestras clases a través del sistema. Los eventos que se manejan dentro de este diagrama serán los eventos que se manejarán en el código final, como se mencionó anteriormente teniendo los diagramas de clase y los diagramas de secuencia se puede programar un sistema.

Los diagramas de secuencia representan una especie de línea de tiempo donde interactúan las clases mandándose mensajes que pueden o no regresar una respuesta. La nomenclatura de los actores y de las clases es la misma de los diagramas de clase con estereotipos solo que se le agrega una línea vertical donde se van agregando los eventos o mensajes que una clase o un actor le manda a otro, *ver Figura 1.16.*

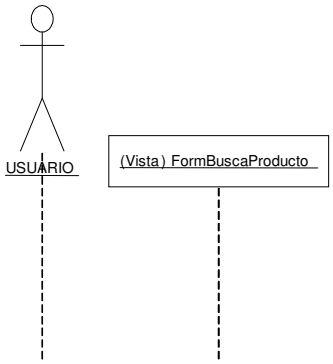


Figura 1.16

Ejemplo de actores y clases para diagramas de secuencia utilizando el lenguaje UML

La comunicación entre clases y actores es mediante flechas que en la parte de arriba tiene el número de secuencia o evento y la acción que se realiza, también para representar el tiempo transcurrido desde que inicia un proceso y se recibe una respuesta se utiliza un rectángulo vertical sobre las líneas de la clase, *ver Figura 1.17.*

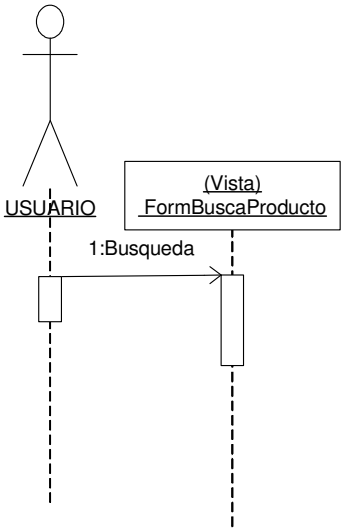


Figura 1.17

Ejemplo de comunicación entre actores y clases utilizando el lenguaje UML

Un ejemplo de un diagrama de secuencia que representa un caso de uso, es el de buscar y consultar producto del sistema "PRODUCT WAREHOUSE", *ver Figura 1.18.*

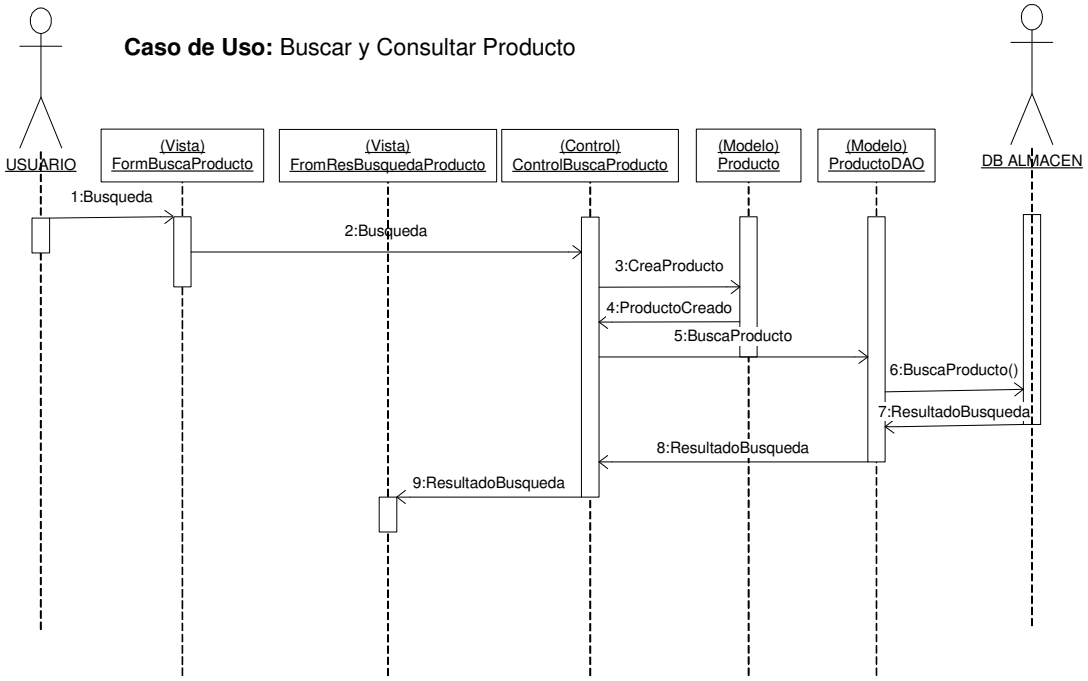


Figura 1.18
Ejemplo de diagrama de secuencia del sistema "PRODUCT WAREHOSE" utilizando el lenguaje UML

2.- LENGUAJE JAVA

2.1.- Introducción a la programación orientada a objetos.

Anteriormente se utilizaba la programación estructurada que consistía en seguir un programa desde su inicio línea por línea hasta su fin en la última línea, ejerciendo control con sentencias de bifurcación y de ciclos de repetición, así como llamadas a métodos fuera del programa principal. Esta práctica era muy eficaz, ya que con estas sentencias de control de flujo se podían realizar aplicaciones de todo tipo.

El paradigma de la programación orientada a objetos es una forma de programar más parecida a la forma de pensar de las personas, ya que asocia tipos de objetos (clases) que tienen propiedades, características y acciones en común. La forma en la que estos tipos de objetos se representan en forma concreta es a través de objetos, esto permite que la programación sea más fácil de entender y más fácil de diseñar; ya que en cierta forma está basada en la forma en la que nuestro cerebro concibe y asocia las ideas. También se le puede considerar como una evolución de la programación estructurada ya que dentro de los "métodos" que forman parte de las clases se sigue utilizando este tipo de programación y se deja la lógica en la que interactúan los elementos a la programación orientada a objetos.

En los siguientes ejemplos podemos ver un clásico Hola Mundo programado tanto en la programación estructurada usando C y el mismo Hola Mundo con programación orientada a objetos.

Programación Estructurada:

```
#include <stdio.h>

 int main() {
    printf("¡Hola, mundo!\n");
    return 0;
 }
```

En la programación estructurada tenemos que nuestro programa comienza con main() que es el bloque principal de nuestro programa después se ejecutará la siguiente sentencia que presentará en pantalla el mensaje hola mundo.

Programación Orientada a Objetos:

```
public class HolaMundo
 {
        public static void main(String[] args)
        {
           System.out.println("¡Hola, mundo!");
        }
 }
```

En la programación orientada a objetos tenemos como primer línea la declaración de una clase llamada HolaMundo esta clase contiene el método main() que nos indica que esta clase debe de ejecutarse como clase principal, la sentencia después del main "System.out.println("¡Hola, mundo!");", utiliza la clase System.out y el método println() de esta clase para presentar en pantalla el mensaje "¡Hola, mundo!".

2.2.- Conceptos fundamentales

La programación orientada a objetos se conforma de varios elementos fundamentales que permiten que este paradigma de programación funcione, estos elementos son las clases, objetos, métodos, eventos y mensajes

Es importante definir los elementos mencionados anteriormente ya que en ellos está basado el paradigma de programación orientado a objetos.

Clase: Define todas las propiedades (atributos) y acciones de un objeto concreto, donde instanciar un objeto es la creación e inicialización de este, con las características de la clase definida.

Objeto: Es la entidad compuesta de propiedades y atributos (datos), también posee comportamiento y funcionalidad (métodos), se representaría como un objeto del mundo real, como lo sería una mesa, una silla, un perro, o cualquier otro objeto que posee características definidas, funcionalidad y propiedades.

Método: Es un algoritmo dentro de la definición de una clase, donde al instanciarse un objeto de este tipo, el objeto será capaz de hacer una modificación en los datos de él mismo, o de otro objeto, regresando algún resultado o un mensaje. A través de los métodos se describe que funcionalidad puede tener un objeto para ir interactuando con otros y así resolver problemas más complejos.

Evento: Es un suceso que desencadena un usuario al interactuar con la máquina enviando un mensaje a un objeto para que realicé cierta acción a través de un manejador de eventos el cual recibe el evento y lo envía al objeto que lo procesará.

Mensaje: Es la comunicación entre objetos. Donde un objeto le manda a otro órdenes para que ejecute cierto método y le regrese un resultado o simplemente para cambiar el estado del objeto, el mensaje puede enviar parámetros para que el objeto que lo recibe pueda ejecutar el método con estos nuevos parámetros.

2.3.- Características principales de un lenguaje OO (Orientado a Objetos).

Para que un lenguaje orientado a objetos pueda ser considerado como tal debe cumplir con las siguientes propiedades:

Abstracción: Es la forma en la que los objetos son concebidos a partir de sus clases que sirven como modelo para que el objeto adquiera todas las propiedades de la clase como son los atributos y los métodos pero, no solo los objetos tienen esta propiedad, también los mismos métodos, las funciones y los procesos pueden ser abstraídos.

Encapsular: Significa reunir a todos los elementos que solo interactúan a un nivel de abstracción esto quiere decir que solo pertenecen y son utilizados en esa entidad, permitiendo tener mayor control y seguridad sobre los datos que maneja la entidad y que pueden considerarse pertenecientes solo a esta entidad. Con esta propiedad existen varios beneficios entre ellos está la posibilidad de separar componentes para su corrección o ampliación.

Polimorfismo: Son comportamientos diferentes de los elementos que conforman un programa donde pueden ser asociados a entidades distintas, que pueden tener el mismo nombre; ejemplos de esta característica son la colecciones genéricas que pueden almacenar diferentes tipos de objetos los cuales serán modificados en forma dinámica.

Herencia: Es una relación de clases donde una clase obtiene las características de otra, como son métodos y atributos dando funcionalidad a la clase, como una relación padre-hijo donde el hijo puede obtener las propiedades del padre para ampliar su funcionalidad e ir definiendo aun más las clases hijas dándoles un comportamiento más especializado.

2.4.- Introducción a Java

Java es un lenguaje orientado a objetos basado en C++, desarrollado por Sun Microsystems este lenguaje a tenido un gran auge por su versatilidad, ya que se basa en una máquina virtual que interpreta el byte-code y teniendo la máquina virtual del sistema operativo específico el mismo código puede ser ejecutado en diferentes plataformas sin realizar cambios.

Otro punto fuerte de Java es su integración con WEB a través de los Servlets[1] y JSP´s[2] que permite tener sistemas en línea multiusuario, por otra parte el desarrollo en marcos de trabajo (FrameWork) permite desarrollar de una forma más rápida este tipo de aplicaciones, siendo Struts[3] uno de los frameworks más utilizados.

2.5.- Identificadores y tipos primitivos

Los identificadores sirven para diferenciar nombres de variables, funciones, clases y objetos, en Java los identificadores pueden empezar con una letra, con un guión bajo (_) o con símbolo de pesos ($), lo caracteres subsecuentes pueden ser letras o dígitos, en Java existe la distinción entre mayúsculas y minúsculas siendo hola y Hola nombres de variables diferentes y no existe una longitud máxima establecida para los identificadores. La declaración de una variable es de esta forma:

```
Tipo identificador [ = valor][, identificador [= valor] . . . ];
```

En Java existen 5 tipos primitivos de datos: enteros (byte, short, int y long), reales en coma flotante (float, double), booleanos (boolean), caracteres (char) y cadenas (String, StringBuffer).

2.6.- Operadores

Dentro de Java existen un gran número de operadores que nos permiten hacer operaciones sobre uno o dos operandos. Existen operadores unarios que actúan con solo un operando, binarios que actúan sobre dos operandos y en Java existe un operador ternario especial que se verá más adelante.

Operadores aritméticos

Los operadores aritméticos actúan sobre valores enteros y de coma flotante, en Java tenemos los siguientes operadores binarios +, -, *, / y %.

La forma de utilizar los operadores binarios es la siguiente:

Operando *operador* operando

[1] Véase Sevlets en pag. 31
[2] Véase JSP en pag. 31
[3] Véase STRUTS pag. 47

En el caso del símbolo +, también puede ser usando para concatenar cadenas como en el siguiente ejemplo:

"hola"+"mundo"+"como estas?"

En el caso de los operadores unarios que soporta Java tenemos:

+	para indicar un valor positivo en un operando
-	para indicar un valor negativo de un operador
++	incrementa en 1 un operando, puede ser utilizado como sufijo o prefijo
--	decremento en 1 un operando, puede ser utilizado como sufijo o prefijo

Operadores relacionales y condicionales

Los operadores relacionales en Java son los siguientes:

>	el operando de la izquierda es mayor que el de la derecha
>=	el operando de la izquierda es mayor o igual al de derecha
<	el operando de la izquierda es menor que el de la derecha
<=	el operando de la izquierda es menor o igual que el de la derecha
==	el operando de la izquierda es igual al de la derecha
!=	el operando de la izquierda es diferente al de la derecha

Combinando los operadores relacionales con los condicionales se pueden obtener evaluaciones de expresiones mucho más complejas. Los operadores condicionales son los siguientes:

&& Las expresiones izquierda y derecha son verdaderas
|| O la expresión izquierda o la expresión derecha son verdaderas
! La expresión de la derecha es falsa

Operadores de asignación

El operador de asignación principal es (=), que se utiliza para asignar el valor del operando de la izquierda al de la derecha, pero también existen una serie de operadores que permiten hacer operaciones y asignación al mismo tiempo. Los operadores de asignación con operación son los siguientes:

+= -= *= /= %= &= |=

Operador ternario if-then-else

Este operador especial de Java es muy útil para ahorrar código para hacer una operación del tipo if-then-else donde primero se evalúa una condición y dependiendo del resultado se realiza una operación u otra. Su forma general es la siguiente:

expresión ? sentencia1 : sentencia2

Un ejemplo para dejar más claro el uso de este operador, es una división, donde se puede evaluar si es una división entre cero y así evitar un error:

cociente = denominador == 0 ? 0 : numerador / denominador

Moldeo de operadores

Es conocido como casting la conversión de un tipo de dato a otro, Java convierte automáticamente de un tipo a otro cuando es pertinente, pero también puede ser forzado. Cualquier tipo de dato primitivo puede ser promovido con excepción de los booléanos.

A veces es necesario hacer un moldeo al tipo genérico de datos conocido como Object para poder manipularlo dentro de algún método o colección y posteriormente se puede volver a moldear para regresar el objeto al tipo que le corresponde, como en el siguiente ejemplo.

```
Miclase clase= new Miclase();
(Object)clase;
….
(Miclase)clase;
```

2.7.- Variables y expresiones

El uso de variables en Java y en todos los lenguajes sirve para almacenar datos que van a cambiar durante la ejecución del programa, para declarar una variable es necesario declarar el tipo y el nombre de la variables con un identificador válido.

Una expresión es un conjunto de operadores y operandos que en combinación forman una sentencia para obtener un resultado en particular, es importante considerar que la precedencia de los operadores puede variar el resultado, para eso se puede utilizar los paréntesis como separador de expresiones si estas son muy complejas.

2.8.- Arreglos, cadenas y tipos genéricos

Los arreglos conocidos en inglés como *Arrays* son un set de variables del mismo tipo referidas por posiciones secuenciales, dentro de Java se hace una comprobación exhaustiva de la correcta manipulación para evitar la corrupción de memoria al sobrepasar los límites establecidos del arreglo.

Un array debe ser declarado antes de utilizarse y después declarar su tamaño, la declaración de un array se muestra a continuación:

```
tipoDeElementos[] nombreDelArray;
```

Y utilizando el operador new podemos proporcionar el tamaño del array:

```
tipoDeElementos[] nombreDelArray = new
   tipoDeElementos[tamañoDelArray];
```

Los arrays pueden ser declarados de cualquier tipo, desde tipos primitivos hasta clases completas, también existen los arrays bidimensionales conocidos como arrays de arrays, un ejemplo de estos es el siguiente:

```
int  arreglo1 [] [] = new  int  [10] [5];
```

Las cadenas conocidas como *String* es un ejemplo especial de un arreglo formado por caracteres, Java pone un énfasis especial en los *Strings* y tiene varias clases para su manipulación, más adelante se hace un estudio más detallado de estas clases y de su importancia en el lenguaje Java.

Los tipos genéricos son muy útiles para guardar e interactuar con una gran cantidad de objetos complejos, objetos como entidades que contienen más de un tipo primitivo.

Dentro de las colecciones existen las de tipo *Vector*, *ArrayList* y *HashMap* que pueden almacenar cualquier tipo de objeto y ofrecen varios métodos para su manipulación y almacenamiento.

Un ejemplo es la clase *Vector*, que se declara y se instancia de la siguiente forma:

```
Vector<tipoDeObjeto> v = new Vector <tipoDeObjeto>();
```

Con el método *add* se pueden agregar elementos a la colección como se muestra en el siguiente código:

```
Vector<Integer> v = new Vector < Integer>();
v.add(1);
v.add(3);
v.add(10);
```

2.9.- Control de Flujo

Las sentencias de control de flujo dentro de un lenguaje de programación nos ayudan a cambiar el control de nuestra aplicación dependiendo de los valores que vayan tomando nuestros datos en tiempo de ejecución, existiendo las bifurcaciones que ramificarán el flujo de la aplicación y los bucles que harán tareas repetitivas. Dentro de Java tenemos las siguientes sentencias:

Sentencias de bifurcación o salto

if/else

```
if(expresión-booleana){
bloque de sentencias;
}

[else {
 bloque de sentencias;
}]
```

El *if/else* es una sentencia que permite evaluar una expresión booleana que regresará verdadero o falso (*true-false*) ejecutando el bloque de sentencias dentro del *if* cuando la expresión sea verdadera y saltándose ese bloque cuando sea falsa, siendo el *else* opcional cuando la expresión evaluada es falsa se ejecutará el bloque contenido dentro de él.

Se puede hacer una construcción *if/else* anidada para ir capturando casos en los que una variable toma distintos valores aunque una mejor herramienta para este caso sería una sentencia *switch*.

switch

```
switch ( expresión ) {
    case valor1:
        sentencias;
        break;
    case valor1:
        sentencias;
        break;
   [default :
       Sentencias;]
    }
```

Como se mencionó anteriormente la sentencia *switch* tiene como finalidad evaluar una variable y dependiendo del valor que esta tome, se ejecutará un bloque de sentencias, el uso de la sentencia *break* rompe el ciclo *switch*, sin esta sentencia se ejecutarían todos los casos subsecuentes al primer case donde entró el programa, la sentencia *default* es opcional y permite capturar todos los valores que no están contemplados en algún *case*.

Sentencias de bucle

Dentro de Java existen dos tipos principales de bucles con sus respectivas variaciones, el *for* y *while*, que nos permiten hacer un ciclo de instrucciones una y otra vez mediante una variable de control y una expresión que será la condición de término del ciclo.

for

El *for* permite desde un inicio declarar toda la información necesaria dentro de su encabezado, donde podemos controlar la inicialización de la variable de control, la condición de término y el incremento de la variable de control, su sintaxis es la siguiente:

```
for( inicialización; terminación; iteración){
bloque de sentencias;
}
```

Otra forma de utilizar el *for* introducida en el JDK (*Java Development Kit*) 1.5 es la siguiente:

```
for (parámetro: expresión)
sentencias;
```

Esta forma de utilizar la sentencia *for* nos permite recorrer mediante un iterador que es denominado como parámetro todas las posiciones de una colección, arreglo o vector de forma automática.

while

```
[inicialización;]
while( condición de terminación){
 sentencias;
[iteración]
}
```

El *while* es un bucle más sencillo que el *for*, en este mientras se cumpla la condición booleana se seguirán ejecutando las sentencias dentro del ciclo, la inicialización e iteración son opcionales y nos permiten llevar el control de la condición de término.

Una variación de la sentencia *while* es el *do/while* que funciona de la misma forma pero sin importar la condición de termino el ciclo se ejecuta por lo menos una vez.

do/while

```
[inicialización;]
do {
 sentencias;
[iteración]
} while( condición de termino);
```

2.10.- Clases de Java

Java ofrece un sin número de paquetes y de clases que tienen usos específicos de hecho este es uno de los puntos más fuertes de Java, ya que constantemente se están actualizando, extendiendo así la funcionalidad del lenguaje, existen clases para casi todo desde manejo de imágenes 2D y 3D hasta comunicación con equipos remotos. A continuación veremos algunas de las clases básicas que son usadas frecuentemente dentro de la programación Java.

La clase String y StringBuffer

Dentro de Java es muy importante el manejo de cadenas, existen objetos tipo **String** y *StringBuffer* un objeto *String* es una cadena de caracteres alfanumérica de valor fijo que no puede ser cambiado después de haber sido declarado, por el contrario un objeto de la clase *StringBuffer* puede variar su tamaño, esta diferencia hace los objetos *String* mucho más baratos para el sistema que los *StringBuffer*, dentro de Java existe la posibilidad de que cualquier objeto puede ser convertido en su representación en cadena con el método *toString* y también con el operador + que concatena cadenas y convierte a *String* cualquier objeto para ser concatenado a una cadena.

La clase *StringBuffer* posee diferentes métodos específicos para cambiar la longitud de la cadena:

```
int lenght(); //Devuelve la longitud de la cadena StringBuffer
char charAt( int index ); /*devuelve el carácter de la posición
                            indicada*/
void setLenght( int newLeght ); //cambia la longitud de la cadena
void setCharAt ( int index, char ch );/* introduce un carácter en la
                                        posición especificada*/
int capacity(); //devuelve la capacidad del StringBuffer
```

La clase System

La clase *System* permite tener acceso a la consola donde se está ejecutando la máquina virtual, de esta forma podemos acceder a los recursos del sistema con solo tener en cuenta las peculiaridades de este, sin tener que modificar el código.

Entrada/Salida estándar

Cuando se ejecuta un programa en Java, automáticamente se abren tres canales para la comunicación con el sistema, uno para entrada de datos (teclado), para salida de datos (pantalla) y para mensajes de error (pantalla), de esta forma se pueden pedir y presentar datos en la consola del sistema.

Para acceder a los métodos de la clase *System* se utiliza el operador punto (.) ya que la clase es final y todos sus contenidos son privados.

Un ejemplo del uso de la clase *System* para la salida de datos es:

```
System.out.println( "Hola Mundo" );
```

La clase Math

La clase *Math* contiene todas las funciones matemáticas contenidas en cualquier lenguaje de programación, esta clase tiene un constructor privado pero la clase es pública y estática para ser utilizada solo exportando la librería.

Las clases estándar de *Math* son las siguientes:

```
Math.abs( x );   //Devuelve el valor absoluto
Math.sin( double a ); //  Devuelve el seno del valor
Math.cos(double a );   //Devuelve el coseno del valor
Math.tan( double a );  // Devuelve la tangente del valor
Math.asin(double a );  // Devuelve el ángulo del seno de a
Math.acos( double a ); //  Devuelve el ángulo del coseno de a
Math.exp(double x );  // Devuelve e elevado a x
Math.sqrt( double b );  // Devuelve la raíz cuadrada de b
Math.pow(double x, double y );  // Devuelve y elevado a la x
Math.PI;   //Devuelve el valor matemático de PI aprox. 3.14
```

La clase File

El manejo de ficheros dentro de Java se hace utilizando la clase *File* esta tiene los siguientes constructores:

```
File miArchivo;

/*1.-*/ miArchivo = new File("/home/msc");

/*2.-*/ miArchivo = new File("/home" , "msc");

/*3.-*/ miArchivo = new File("/home");
        miArchivo = new File ( directorio , "msc")
```

Cuando se requiere un solo fichero en una aplicación el primer constructor es el más adecuado, por el contrario cuando se manejan muchos archivos dentro de un directorio específico el segundo es mejor y el tercero sirve cuando el nombre del archivo es una variable.

A continuación se mencionan algunos métodos útiles en el manejo de archivos.

```
miArchivo.delete(); // Borra un archivo descrito por su path

miArchivo.mkdir(); // Crea un directorio descrito por su path

miArchivo.exist();//Verifica si el archivo descrito por su path existe

miArchivo.getName(); // Devuelve el nombre del archivo descrito por su
                     // path
```

2.11.- JDBC

Bases de datos y el DBMS MySQL

Las bases de datos nos permiten guardar en un medio permanente la información que necesitemos para su recuperación posterior, esta definición podría ser la de un archivo que era lo que se utilizaba antes de las bases de datos, la diferencia entre uno y otro es que en la base de datos existe una lógica que permite eliminar los principales problemas que existían con los archivos como la redundancia e inconsistencia en los datos, dificultad de acceso a los datos, aislamiento de los datos, anomalías de acceso concurrente y falta de seguridad, esta lógica aplicada en los datos se hace mediante los modelos de datos, existen varios modelos como entidad-relación, el orientado a objetos, de red, jerárquico y relacional.

En la actualidad uno de los modelos más usados por las empresas e instituciones para las bases de datos de sus sistemas computacionales es el modelo relacional.

El modelo relacional como su nombre lo indica introduce relaciones lógicas entre los datos contenidos en una columna con otra columna de otra tabla, existiendo así columnas con nombres únicos, que relacionan las tablas subsecuentes.

Ejemplo:

Nombre	Calle	Ciudad	Número
Lowery	Mapple	Queens	900
Shiver	North	Bronx	556
Shiver	North	Bronx	647

Tabla 1. Ejemplo 1 de tabla de una base de datos relacional

Saldo	Número
100	900
2400	556
500	647

Tabla 2. Ejemplo 2 de tabla de una base de datos relacional

De esta forma podemos relacionar el campo "Número" de la *Tabla 1* con el campo "Número" de la *Tabla 2* y así obtener una separación de los datos personales de la *Tabla 1* con el saldo del cliente de la *Tabla 2*, pero existiendo una relación entre estas tablas por medio del campo "Número".

Para manejar de una forma más eficaz nuestros datos, existen los DBMS (*Data Base Manager System*) en español manejador de bases de datos, estos programas nos ayudan a administrar nuestras bases de datos, tablas y registros, permitiendo manejar una gran cantidad de información, su lógica aplicada y la seguridad de los datos.

Existen diferentes DBMS como el SQL Server, Oracle, Informix, MySQL y muchos más, todos los DBMS utilizan el lenguaje SQL estándar más pequeñas variaciones dependiendo del DBMS, así que conociendo el lenguaje SQL estándar solo hay que ver la documentación de cada DBMS para ver los cambios.

Para este proyecto utilizaremos el DBMS MySQL el cual es fácil de usar y es de uso libre.

La documentación completa del MySQL está en la página oficial del producto *http://www.mysql.com/*, aquí mostraremos algunas de las sentencias básicas para la creación y manejo de las bases de datos y tablas.

El lenguaje SQL (*Structured Query Language*) o en español Lenguaje de Consulta Estructurado es el que nos permite darle las instrucciones a nuestro DBMS para que este pueda manipular los datos, las sentencias en SQL se clasifican en dos las DDL y las DML.

Las sentencias DDL (*Data Definition Lenguaje*) en español lenguaje de definición de datos, que nos permiten definir los objetos de una base de datos donde dichos objetos son guardados en el diccionario de datos, algunas sentencias de este tipo son *create, revoke, grant y alter* entre otras.

Las sentencias DML (*Data Manipulation Lenguaje*) en español lenguaje de manipulación de datos, que se utilizan para manejar los datos de una base, algunas sentencias de este tipo son *select, insert, update y delete*.

Comandos y Sentencias de MySQL

-Entrar al servidor MySQL desde la línea de comandos:

```
mysql (-h host) -u user -p (base de datos)
```

-Salir del servidor

```
QUIT, \q
```

-Ver datos del servidor

```
SELECT VERSION(), CURRENT_DATE, USER(), NOW();
```

-Salir del tipeo de comandos

```
\c
```

-Mostrar bases de datos

```
show databases
```

-Seleccionar una base

```
use nombrebase (sin ;)
```

-Crear una base de datos

```
CREATE DATABASE nombreBase;
```

-Dar permisos de uso a un usuario y a la vez crearlo

```
-GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER,INDEX,FILE ON
nombrebase.* TO llama@localhost IDENTIFIED BY 'camel'; (password)

-GRANT ALL ON nombrebase.* TO 'su_nombre_mysql'@'su_host_cliente';
```

-Ver tablas de la base seleccionada

```
SHOW TABLES;
```

-Crear tablas

Ejemplo:

```
CREATE TABLE nombretabla(campo1 TIPO (NOT NULL), owner VARCHAR(20),
      species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

Otro ejemplo:

```
CREATE TABLE CLIENTE (id INT NOT NULL,
RSocial VARCHAR (15) NOT NULL,
RFC VARCHAR (15) NOT NULL,
CalleNum VARCHAR (30) NOT NULL,
Colonia VARCHAR (20),
Municipio INT NOT NULL,
Estado INT NOT NULL,.
```

```
Cp INT NOT NULL,
 PRIMARY KEY(id)
)
```

-Ver los campos de una tabla

```
DESCRIBE nombretabla;
```

-Cargar datos desde un archivo de texto

```
LOAD DATA LOCAL INFILE '/path/nombrearchivo.txt' INTO TABLE
nombretabla; LINES TERMINATED BY '\r\n'; (opcional en windows)
```

(Ejemplo de columna para cargar el archivo de texto)

```
name         owner       species    sex   birth            death
Whistler     Gwen         bird      \N    1997-12-09       \N
```

-Insertar registros uno por uno

Ejemplo:

```
INSERT INTO pet
VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

-Para seleccionar información de una o más tablas

```
SELECT (DISTINCT) seleccionar_Esto \\distinc para seleccionar no
FROM desde_tabla                    \\ repetidos
WHERE condiciones;
order by nombrecampo1, nombrecampo2, ... (DESC)
condiciones del where
    (and, or)
```

-Borrar contenido de las tablas

Para borrar todos los registros:

```
DELETE FROM nombretabla;
```

-Actualizar un registro

Ejemplo:

```
UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
```

Conectividad JDBC

JDBC (Java Data Base Conectivity) es una especificación para conexión a base de datos en un programa Java, esta especificación fue creada por Sun Microsystems y cada fabricante de DBMS crea controladores que se adecuarán a la especificación JDBC.

JDBC es un traductor que se encarga de mandar mensajes entre el DBMS y la API (Application Programming Interface) o en español Interfaz de Programación de Aplicaciones, JDBC está compuesto por dos partes, una es la API que contiene las clases necesarias para la manipulación de datos y el Driver que establece la conexión con el DBMS.

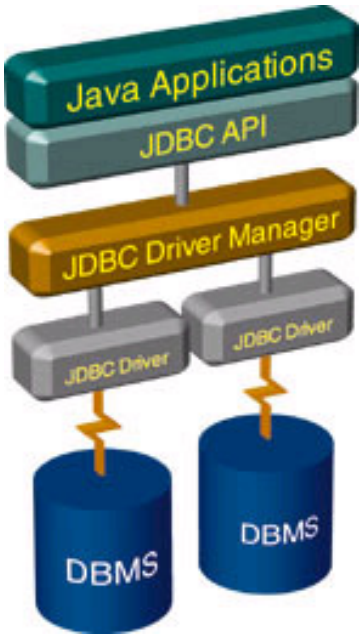


Figura 2.1

Diagrama que describe la arquitectura principal de una conexión JDBC

En la *Figura 2.1* podemos observar como nuestras aplicaciones se encuentran en la parte superior de este diagrama, después tenemos el API de JDBC que está escrito en lenguaje JAVA, el siguiente nivel es el manejador del driver JDBC que tiene conexión con el Driver JDBC que sirve como traductor de las sentencias Java a código SQL y por último tenemos el DBMS que maneja nuestras bases de datos.

El Driver es una colección de archivos .class comprimidos en un archivo JAR que permiten la conexión y manipulación de la base de datos, cada fabricante de DBMS distribuye su propia API en un archivo JAR.

Pasos para crear conexión

Para establecer una conexión con la base de datos hay que seguir los siguientes pasos:

1.- Cargar el Driver

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      try{
         Class.forName("com.mysql.jdbc.Driver"); //con esta sentencia
cargamos el driver
            …
      …
```

```
      }catch(Exception e){System.out.println("Problema\n"+e);}
   }
}
```

2.- Définir URL

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      String host;
      String dbName;
      try{
         Class.forName("com.mysql.jdbc.Driver");
      host=“jdbc:mysql://127.0.0.1/”;  //Aqui definimos el DBMS y el
      dbName=“curso”                  // host, despues definimos el
      …                               // nombre de nuestra BD
      …


      }catch(Exception e){System.out.println("Hubo un problema con la
conexión\n"+e);}
   }
}
```

3.- Establecer la conexión

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      String host;
      String dbName;
      try{
         Class.forName("com.mysql.jdbc.Driver");
      host=“jdbc:mysql://127.0.0.1/”;
      dbName=“curso”

Connection db =
DriverManager.getConnection
host+dbName,“msc”,“dnf"); /*en estas líneas hacemos la conexión
                           a la DB introduciendo el host, el
                           nombre de la base, el login y password */

      }catch(Exception e){System.out.println("Problema\n"+e);}
   }
}
```

4.- Crear un objeto Statement

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      String host;
      String dbName;
      try{
         Class.forName("com.mysql.jdbc.Driver");
      host=“jdbc:mysql://127.0.0.1/”;
      dbName=“curso”

      Connection db     =DriverManager.getConnection
(host+dbName,“msc",“dnf");
```

```
      Statement st = db.createStatement(); //creamos un objeto Stament
      …                                     //que nos permitirá hacer
      …                                    //las consultas a la BD
      }catch(Exception e){System.out.println("Problema\n"+e);}
   }
}
```

5.- Ejecutar manipulaciones y consultas

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      String host;
      String dbName;
      try{
         Class.forName("com.mysql.jdbc.Driver");
      host=“jdbc:mysql://127.0.0.1/”;
      dbName=“jmaster1”

      Connection db
      =DriverManager.getConnection(host+dbName,“jm1",“jmaster");
      Statement st = db.createStatement();
         ResultSet rs =
         st.executeQuery("SELECT  *  FROM cliente"); // esta línea
      … // ejecuta un sentencia de query para regresarnos un resultado
      …


      }catch(Exception e){System.out.println("Problema\n"+e);}
   }
}
```

6.- Procesar resultados

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      String host;
      String dbName;
      try{
         Class.forName("com.mysql.jdbc.Driver");
      host=“jdbc:mysql://127.0.0.1/”;
      dbName=“jmaster1”

      Connection db
      =DriverManager.getConnection(host+dbName,“jm1",“jmaster");
      Statement st = db.createStatement();
         ResultSet rs = st.executeQuery("SELECT  *  FROM cliente");

      /* Al obtener el resultado del query podemos recorrer el
      ResulSet para procesar los registros obtenidos*/

      if (rs.wasNull()) System.out.println("NADA\n");
      while(rs.next())
         System.out.println(“Razon Soc= "+ rs.getString(1) + " RFC= "
+  rs.getString(2) );

      …
      …
      }catch(Exception e){System.out.println("Problema\n"+e);}
   }
}
```

7.- Cerrar

```
import Java.sql.*;
public class TestMysql {
   public static void main(String[] argv){
      String host;
      String dbName;
      try{
         Class.forName("com.mysql.jdbc.Driver");
      host=“jdbc:mysql://127.0.0.1/”;
      dbName=“jmaster1”

      Connection db
      =DriverManager.getConnection(host+dbName,“jm1",“jmaster");
      Statement st = db.createStatement();
         ResultSet rs = st.executeQuery("SELECT  *  FROM alumno");

      if (rs.wasNull()) System.out.println("NADA\n");
      while(rs.next())
         System.out.println(“Nombre= "+ rs.getString(3) + " NC= " +
rs.getString(2) );

/*la sentencia DBObject.close permite cerrar los objetos que
utilizamos durante un consulta a una BD*/

 rs.close();
         st.close();
         db.close();

      }catch(Exception e){System.out.println("Problema\n"+e);}
   }
}
```

Clases para manipular, ejecutar consultas y procesar resultados

Esta tabla muestra las clases más utilizadas en JDBC para establecer una conexión, ejecutar consultas, procesar los resultados y obtener información de la base de datos.

Clase/Interface	Descripción
Driver	Permite conectarse a una base de datos: cada gestor de base de datos requiere un driver distinto
DriverManager	Permite gestionar todos los drivers instalados en el sistema
DriverPropertyInfo	Proporciona diversa información acerca de un driver
Connection	Representa una conexión con una base de datos. Una aplicación puede tener más de una conexión a más de una base de datos
DatabaseMetadata	Proporciona información acerca de una Base de Datos, como las tablas que contiene, etc.
Statement	Permite ejecutar sentencias SQL sin parámetros
PreparedStatement	Permite ejecutar sentencias SQL con parámetros de entrada
CallableStatement	Permite ejecutar sentencias SQL con parámetros de entrada y salida, típicamente procedimientos almacenados
ResultSet	Contiene las filas o registros obtenidos al ejecutar un SELECT
ResultSetMetadata	Permite obtener información sobre un ResultSet, como el número de columnas, sus nombres, etc.

3.- SERVLETS Y JSP´s

Dentro del mundo de los sitios web existen dos clases de páginas, las estáticas que solo permiten ver información de un fichero HTML mandado por un servidor a una máquina cliente haciendo que la interacción con el usuario sea solo unidireccional ya que el usuario no puede enviar información al servidor para interactuar con él, esta práctica a caído en desuso debido a que los portales web actuales requieren el llenado de formularios o registro y autentificación de usuarios esto se logra mediante páginas dinámicas que implementan código HTML, scripts o aplicaciones que se ejecutan en el servidor y regresan una respuesta.

La utilización de los scripts en lenguajes como Perl y PHP permite generar páginas HTML construidas dinámicamente, que son independientes del navegador donde ese ejecuten, debido a que se encuentran dentro del servidor que además ofrece una respuesta mucho más rápida a ciertos procesos complejos.

Los scripts contenidos dentro de un servidor son conocidos como CGI (Common Gateway Interface - Interfaz de Entrada Común), que fueron la respuesta a la necesidad de hacer más interactivas las páginas web, pero esta tecnología tiene un gran problema el cual consiste en que lanza una copia del script por cada respuesta provocando una sobrecarga en el servidor cando este tiene un gran número de peticiones simultaneas.

La respuesta de Sun Microsystems a la tecnología CGI fueron los servlets. Los servlets tienen diversas características que superan a los CGI, una característica y tal vez la más importante es que un servlet no sobrecarga el servidor ya que levanta solo una vez el proceso sin importar las peticiones, queda en memoria para su reutilización y una misma instancia responde a todas la peticiones concurrentes resolviendo así el problema principal de los CGI, por otro lado el hecho de que los servlets estén escritos en código Java nos da la independencia entre plataformas y la seguridad en el manejo de memoria que contiene este lenguaje de programación.

3.1.- API Servlet

El API servlet contiene las clases necesarias para la generación y publicación de servlets en diferentes servidores que contienen la norma servlet, siendo el Jakarta-Tomcat el motor de servlets oficial de Sun Microsystems desarrollado por la Apache Software Foundation. Debido a la flexibilidad del API servlet existen diversos servidores capaces de interpretar servlets tales como JBOSS, ServletExec, WEBLogic, etc.

Para crear un servlet hay que crear una clase que herede de HttpServlet, que debe ser guardada en un directorio con una forma específica para que nuestro motor de servlets pueda hacer una instancia de la clase, la forma del directorio es /WEB-INF/*clases*.

La clase que hereda de HttpServlet hereda también la interfaz Servlet que contiene 5 métodos:

service().- Este método es de los más importantes ya que a través de los objetos que recibe como parámetros (ServletResquest y ServletResponse) se hace la interacción entre el servidor y el cliente, ServletRequest se encarga de encapsular las peticiones del cliente y ServletResponse contiene el código que regresará la respuesta a la petición hecha.

init().- Permite inicializar un servlet, este método será solo llamado una vez durante la carga del servlet y puede recibir como parámetro un objeto del tipo ServletConfig, que contendrá información de configuración del servidor.

destroy().- Este método permite liberar un servlet para volver a ser cargado por el servidor en caso de que el servlet sea actualizado, este método solo es llamado una sola vez dentro del ciclo de vida del servlet.

getServletConfig().- Obtiene un referencia hacia el objeto ServletConfig para poder obtener la configuración del servlet.

log().- Se utiliza para mandar mensajes a los archivos log del servidor para saber en qué estado se encuentra nuestra aplicación.

getServletContext().- Obtiene una referencia a un objeto ServletContext que sirve para interactuar con otros objetos de nuestra aplicación web como pueden ser JSP (Java Server Pages), Java Beans o incluso otros servlets.

La razón por la que los servlets tienen un rendimiento superior frente a otras tecnologías es que una vez creada e inicializada una instancia de un servlet esta se mantiene latente en el servidor durante el tiempo que se ejecute nuestra aplicación web, permitiendo así procesar información en cualquier momento que sea requerido, otra ventaja es que la llamada del método service() puede ser ejecutada por diferentes tareas al mismo tiempo.

La clase HttpServlet implementa la interfaz Servlet que contiene todos los métodos necesarios para crear y manipular nuestros servlets, básicamente Httpservlet permite procesar formularios Html, el protocolo por el cual se comunica esta clase es el HTTP (*HyperText Transfer Protocol*) que implementa una relación petición-respuesta y es usado para la comunicación entre un cliente y un servidor web, estas peticiones y respuestas están compuestas por dos partes, una cabecera que contiene información de configuración del navegador o del servidor y el cuerpo que contiene los datos que se desean procesar o los resultados de este procesamiento.

El método service() que implementa la clase HttpServlet utiliza los métodos estándar utilizados en el protocolo HTTP/1.1 que nos sirven para establecer una comunicación petición-respuesta, estos métodos son GET, POST, PUT, HEAD, TRACE, OPTIONS y DELETE, en Java todos estos métodos son llamados de la forma doxxx (doGet(), doPost, doTrace, etc.) y reciben dos parámetros HttpServletResponse y HttpServletRequest para respuesta y petición respectivamente.

La sobrecarga de algunos de estos métodos está permitida ya que dentro de esta nueva implementación se estarán procesando los datos del cliente para enviarle una respuesta, los métodos doOptions() y doTrace() son los dos únicos métodos que no se pueden sobrecargar y el método HEAD devuelve la cabecera que se puede obtener al igual que con el método doGet() solo que sin el cuerpo, así que los métodos a sobrescribir son doGet(), doPost(), doDelete() y doPut(), donde se necesita sobrecargar al menos uno de ellos para darle la funcionalidad necesaria a nuestra aplicación web.

Los argumentos que se le pasan a estos métodos son, en el caso de la petición el HttpServletResquest que pertenece a una superclase llamada ServletRequest y las respuesta que envían estos métodos está contenida dentro de HttpServletResponse que es una subclase de ServletResponse.

Para el envió de datos de un formulario se pueden utilizar los métodos GET y POST, el método GET envía los parámetros dentro de la URL y el método POST los envía en el cuerpo de la página. Por otra parte los métodos PUT y DELETE son utilizados para descargar y borrar recursos en un servidor respectivamente.

3.2.- POST

El método POST nos permite enviar los datos a través de nuestra aplicación web pero solo por medio de un formulario html, con la etiqueta html <FORM> que tiene como parámetros el método ACTION que es la URL hacia donde se redirige la página para su tratamiento y METHOD que indica cómo se va a tratar ya sea como GET o POST.

3.3.- GET

El método GET funciona tanto con formularios como con hiperenlaces siendo mucho más útiles con estos, el método GET enviará en la URL los datos para su procesamiento en la siguiente página donde evidentemente no es conveniente enviar información sensible como pueden ser nombres de usuarios, passwords o números confidenciales.

3.4.- HttpServletRequest y HttpServletResponse.

A continuación se muestran todos los métodos para estas interfaces y su uso

HttpServletRequest

Esta clase al implementar la interfaz ServletRequest utiliza dos métodos que son de los más importantes para el flujo de datos dentro de nuestra aplicación web.

getParameter(String name)

Este método nos permite recuperar los parámetros de la sesión para ser procesados por nuestras páginas a través del flujo de nuestra aplicación web.

setAttribute(String nombre, Object obj)

Con este método podemos nombrar atributos para luego ser recuperados por otra página web especificando el nombre del atributo y el objeto.

A continuación se muestran los demás métodos propios de la clase HttpServletRequest.

getAuthType()

Regresa el nombre del esquema de autentificación usado para proteger el servlet, por ejemplo, "BASIC" o "SSL", o NULL si el servlet no está protegido.

getContextPath()

Regresa la porción de la URI pedida que indica el contexto de la petición.

getCookies()

Regresa un arreglo que contiene todos los objetos Cookie que el cliente envió en esa petición.

getDateHeader(String name)

Regresa el valor de la cabecera de la página web específica como un valor long que representa un objeto Date.

getHeader(String name)

Regresa el valor de la cabecera de la página web específica pedida como un String.

getHeaderNames()

Regresa una Ennumeration de todos los nombres de las cabeceras que contiene esa petición.

getHeaders(String name)

Regresa todos los valores de la cabecera específica de la petición con un Ennumeration de objetos String.

getIntHeader()

Regresa el valor de la petición específica como un Int.

getMethod()

Regresa el nombre de método HTTP con el cual fue hecha la petición, por ejemplo, GET, POST o PUT.

getPathInfo()

Regresa alguna información del path extra asociado con la URL que el cliente manda cuando hace una petición.

getPathTranslated()

Regresa alguna información del path extra después del nombre del servlet pero antes de la cadena de query y lo traduce en un path real.

getQueryString()

Regresa la cadena query que está contenida en la URL de la petición después del path.

getRemoteUser()

Regresa el login del usuario que está haciendo esa petición, si el usuario esta autentificado o NULL si el usuario no ha sido autentificado.

getRequestedSessionId()

Regresa el ID de la sesión especificado por el cliente.

getServletPath()

Regresa la parte de la URL pedida que llamó el servlet.

getSession()

Regresa la sesión actual asociada con la petición o si la petición no tiene una sesión la crea.

getUserPrincipal()

Regresa un objeto Java.security.Principal que contiene el nombre del actual usuario autentificado.

isRequestedSessionIdFromCookie()

Checa si al ID de sesión de la petición viene como una cookie.

isRequestedSessionIdFromUrl()

Checa si al ID de sesión de la petición viene como una URL.

isRequestedSessionIdValid()

Checa que el Id de sesión de la petición sigue siendo válido.

isUserInRole(String role)

Regresa un booleano indicando que el usuario autentificado este incluido en el rol lógico específico.

HttpServletResponse

addCookie(Cookie cookie)

Agrega la cookie específica a la respuesta.

addDateHeader (String name, long date)

Agrega una cabecera de respuesta junto con el nombre y el valor de la fecha.

addHeader(String name, String value)

Agrega una cabecera de respuesta con el nombre y el valor dado.

addIntHeader(String name, int value)

Agrega una cabecera de respuesta con el nombre y el valor entero dado.

containsHeader(String name)

Regresa un booleano indicando si la nombrada cabecera de respuesta ha sido llenada.

encodeRedirectURL(String url)

Codifica la URL específica para ser usada en el método sendRedirect o si la codificación no se necesita regresa la URL sin cambios.

encodeURL(String url)

Codifica la URL específica incluyendo el ID de sesión en el o si la codificación no se necesita regresa la URL sin cambios.

sendError(int sc)

Envía un error de respuesta al cliente usando el estatus específico.

sendError(int sc, String msg)

Envía un error de respuesta al cliente usando el código de estatus específico y el mensaje que lo describe.

sendRedirect(String location)

Envía una respuesta de redirección temporal al cliente utilizando la URL de redirección especificada.

setDateHeader(String name, long date)

Nombra la cabecera de respuesta con el nombre y la fecha dada.

setHeader(String name, String value)

Nombra la cabecera de respuesta con el nombre y el valor dado.

setIntHeader(String name, int value)

Nombra la cabecera de respuesta con el nombre y el valor entero dado.

setStatus(int sc)

Nombra el código de estatus para esa respuesta.

En el siguiente ejemplo de servlet, se procesa una petición para agregar un alumno a una base de datos, el procesamiento de los datos se hace en el método *processRequest()* que sin importar que haya llegado por el método GET o POST será procesado ya que se sobrescribieron estos métodos para ser re-direccionados hacia el *processRequest()*.

La sentencia que nos permite dar como salida código HTML y que este sea interpretado por el navegador es la sentencia *out.println("");.*

En este ejemplo se hace uso de los métodos de la interfaz ServletRequest y ServletResponse para obtener los datos que provienen de otro servlet y re-direccionar según el resultado del procesamiento de nuestros datos.

```
/*
 * agregaAlumno.Java
 *
 * Created on 24 de octubre de 2007, 20:27
 */

package controlador;

import Java.io.*;
import Java.net.*;

import Javax.servlet.*;
import Javax.servlet.http.*;
import modelo.Alumno;
import modelo.dao.AlumnoDAO;

/**
 *
 * @author Administrador
 * @version
 */
public class agregaAlumno extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and
<code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
```

```
    protected      void      processRequest(HttpServletRequest      request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        String nc=request.getParameter("numCta");
        String n=request.getParameter("nombre");
        String ap=request.getParameter("apPaterno");
        String am=request.getParameter("apMaterno");
        Alumno ca = new Alumno(0,nc,n,ap,am);

               if  (AlumnoDAO.insertAlumno(ca,"root","manuelsalazar")){

response.sendRedirect(request.getContextPath()+"/exito.html");
               }else{

response.sendRedirect(request.getContextPath()+"/fallo.html");
               }
        /* TODO output your page here
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet agregaAlumno</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet agregaAlumno at " + request.getContextPath
() + "</h1>");
        out.println("</body>");
        out.println("</html>");
         */
        out.close();
    }
        // <editor-fold defaultstate="collapsed" desc="HttpServlet methods.
Click on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected  void  doGet(HttpServletRequest  request,  HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected  void  doPost(HttpServletRequest  request,  HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>
```

3.5.- Cookies

Las cookies son elementos que permiten almacenar información de forma temporal en la parte del cliente, esto permite cargar menos al servidor y enviarle información al cliente que en una visita posterior será recuperable, aunque una desventaja de las cookies es que no se puede enviar información sensible.

Una cookie consta de un nombre y valor y los servlets tienen métodos para su manipulación

El constructor que nos permite crear objetos del tipo cookie es el siguiente:

```
new Cookie(name, value)
```

Donde name es el identificador del cookie y value su valor, ambos parámetros deben ser del tipo String que después pueden ser moldeados para convertirlos en algún tipo primitivo.

Los métodos que permiten leer y especificar los atributos de una cookie son los siguientes:

getComment/setComment
Obtiene/Selecciona un comentario que está asociado a este cookie.

getDomain/setDomain
Obtiene/Selecciona el dominio al que se aplica el cookie.

getMaxAge/setMaxAge
Obtiene/Selecciona el tiempo dado en segundos que debe pasar para que la cookie expire, si no se específica la cookie solo vive durante la sesión actual.

getName/setName
Obtiene/Selecciona el nombre del cookie. Al recorrer el arreglo de cookies obtenemos el nombre de la cookie solamente con getName y para obtener el valor se utiliza el método getValue.

getPath/setPath
Obtiene/Selecciona el path al que se aplica este cookie. Si no especificamos esto el cookie se devuelve para todas las URL´s del mismo directorio que la página actual.

getSecure/setSecure
Obtiene/Selecciona el valor booleano indicando si el código sólo debería enviarse sobre conexiones encriptadas (SSL).

getValue/setValue
Obtiene/Selecciona el valor asociado con el cookie.

getVersion/setVersion
Obtiene/Selecciona la versión del protocolo del cookie la versión 0 es por defecto.

Para utilizar las cookies deben de ser agregadas a la cabecera de respuesta esto se hace mediante el método *addCokkie* de un objeto *HttpServletResponse* como se muestra a continuación:

```
Cookie userCookie = new Cookie("user", "uid1234");
response.addCookie(userCookie);
```

Después los métodos que nos permiten procesar las cookies son *getCookies* que nos devuelve el arreglo de todas las cookies enviadas en la cabecera, con el método *getName* recuperamos la cookie que nos interesa y con *getValue* obtenemos el valor que será procesado.

3.6.- Sesiones

Las sesiones permiten dar seguimiento a un usuario y a las acciones que realice en el sistema, se pueden almacenar valores que irán pasando de una página a otra para su posterior utilización, el API que nos permite realizar acciones de sesión es *HttpSession*.

Los pasos que se realizan para hacer seguimiento de una sesión son los siguientes:

1.-Buscar un objeto *HttpSession* que este asociado a la petición actual, esto se logra mediante el método *getSession* del *HttpServletRequest*, en el caso de no existir una sesión se crea una automáticamente mandando true como parámetro, como en el siguiente ejemplo:

```
HttpSession session = request.getSession(true);
```

2.-Buscar la información asociada con una sesión, esto se logra mediante la recuperación de la sesión y recuperar un valor asociado a la sesión previamente incluido en ella mediante el método *setAtributte(name, object)* donde "name" es el nombre del atributo y "object" el objeto que se desea agregar a la sesión, la recuperación se hace mediante el método *getAtributte(name)* que regresa el objeto asociado con ese nombre solo que para procesarlo hay que hacer un moldeo que permita especificar de qué tipo de objeto se trata como en el siguiente ejemplo:

```
HttpSession session = request.getSession(true);
ShoppingCart previousItems =
(ShoppingCart)session.getAtributte("previousItems");
. . .
```

Otros métodos para obtener información de una sesión son los siguientes:

getAtributteNames

Este método nos sirve para conocer todos los nombres de los atributos guardados en una sesión y que nos devuelve un objeto **Enumeration**.

getId

Este método devuelve un identificador único generado para cada sesión.

isNew

Este método devuelve **true** si el cliente nunca ha visto la sesión, esto sucede cuando la sesión acaba de ser creada y devuelve **false** para una sesión preexistente.

getCreationTime

Devuelve la hora, en milisegundos desde 1970, en la que se creó la sesión. Para obtener un valor útil para impresión, pasamos el valor al constructor de **Date** o al método **setTimeInMillis** de **GregorianCalendar**.

getLastAccessedTime

Esto devuelve la hora, en milisegundos desde 1970, en que la sesión fue enviada por última vez al cliente.

getMaxInactiveInterval

Devuelve la cantidad de tiempo, en segundos, que la sesión debería seguir sin accesos antes de ser invalidada automáticamente. Un valor negativo indica que la sesión nunca se debe desactivar.

3.- Asociar información con una sesión, esto se hace mediante el método *getAtributte* para recuperar información y luego con el método *putValue* que reemplaza cualquier valor anterior esto sirve cuando queremos recuperar un valor y luego modificarlo, como en el siguiente ejemplo:

```
HttpSession session = request.getSession(true);
session.putValue("referringPage", request.getHeader("Referer"));
ShoppingCart previousItems =
(ShoppingCart)session.getAtributte("previousItems");
if (previousItems == null) {
previousItems = new ShoppingCart(...);
}
String itemID = request.getParameter("itemID");
previousItems.addEntry(Catalog.getEntry(itemID));

session.putValue("previousItems", previousItems);
```

3.7.- JSP (Java Server Pages)

JSP es la evolución de los servlets, estos sirven para construir páginas web dinámicas separando la vista de la lógica de la aplicación y la respuesta de Sun Microsystems para la tecnología ASP de Microsoft.

Dentro de un servlet común tenemos las partes estáticas y dinámicas de una página HTML juntas al igual que la lógica de negocio provocando que los programas y componentes no sean reutilizables, difíciles de mantener y de modificar.

Las páginas ASP (Active Server Pages) son lo más cercano a los JSP, donde su principal desventaja es que solo tienen soporte para los productos de Microsoft, sin dar la libertad de utilizar otro servidor web que no sea el de Microsoft.

Otras desventajas de los ASP´s frente a los JSP´s son las siguientes:

- Los ASP´s están escritos con VBScript y los JSP están escritos con leguaje Java haciéndolo independientes de la plataforma.

- Estas dos tecnologías se basan en etiquetas que generan el código dinámicamente, los JSP´s tienen una característica que permite que el programador cree sus propias etiquetas personalizadas y las utilice en sus aplicaciones, característica con la que no cuentan los ASP´s.

- Otra diferencia es la velocidad, ya que las páginas ASP siempre son interpretadas a diferencia de las páginas JSP que son compiladas para ser convertidas en Servlets y al ser llamadas la primera vez quedarán cargadas en memoria para su posterior utilización.

- Como ya se mencionó antes, la utilización de las ASP´s compromete a utilizar aplicaciones de Microsoft como Microsoft IIS (Internet Information Services – Servicios de Información de Internet) y Personal Web Server, por el contrario de los JSP´s que no dependen de un servidor web especifico ni de una plataforma.

3.8.- MVC (Modelo-Vista-Controlador)

Al utilizar las etiquetas o tags de JSP podemos separar los componentes de nuestra aplicación utilizando un modelo como el modelo-vista-controlador que consiste en separar la vista que se le presenta al usuario con el modelo de datos, siendo el controlador el encargado de refrescar los datos del modelo en la vista, *ver Diagrama 3.1*

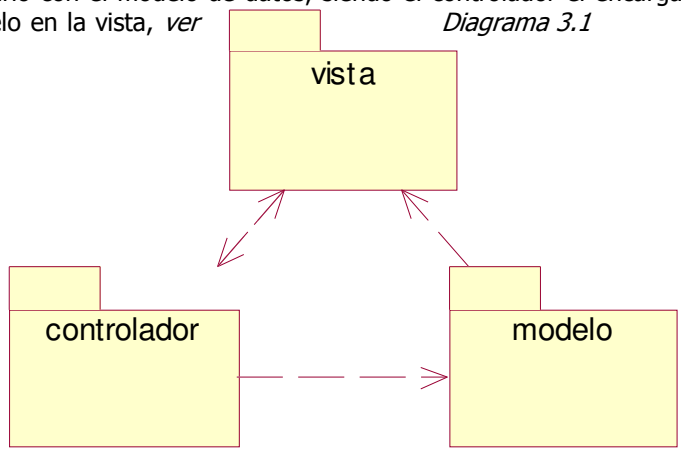


Diagrama 3.1. Este diagrama describe el flujo de comunicación entre los distintos tipos de objetos.

3.9.- Scripts

Son etiquetas especiales que sirven para agregar código Java a nuestras páginas, existen los:

Scriptlets (<% %>)

Esta etiqueta permite insertar código Java dentro del método servlet que quedará intacto al construirse la página, su sintaxis general es:

<% Código Java %>

Al utilizar los Scriplets podemos implementar lógica para la presentación de nuestra página ya que se pueden dejar abiertas las sentencias Java para afectar alguna parte del código estático:

```
<% if (Math.random() < 0.5) { %>
Have a <B>nice</B> day!
<% } else { %>
Have a <B>lousy</B> day!
   <% } %>
```

Que al construirse nuestra página obtendremos:

```
if (Math.random() < 0.5) {
out.println("Have a <B>nice</B> day!");
} else {
out.println("Have a <B>lousy</B> day!");
}
```

Finalmente se puede observar que el equivalente al scriplet en código XML es:

```
<jsp:scriplet>
   Código Java
</jsp:scriplet>
```

Expresiones (<%= %>)

Las expresiones JSP nos permiten introducir valores directamente en la salida de nuestro JSP y tienen la siguiente forma:

```
<%= expresión Java %>
```

La expresión contenida dentro de esta etiqueta es evaluada para obtener su valor, convertida a cadena y presentada en la página.

Existen una gran cantidad de variables ya establecidas que podemos usar o que son de uso común, algunas de ellas son:

- **request**, el **HttpServletRequest**.
- **response**, el **HttpServletResponse**.
- **session**, el **HttpSession** asociado con el **request.**
- **out**, el **PrintWriter** (una versión con buffer del tipo **JspWriter**) usada para enviar la salida al cliente.

Un ejemplo podría ser cuando queremos obtener el Path de donde nos encontramos en ese momento:

```
<%=request.getContextPath()%>
```

Que nos podría imprimir algo como esto "http://localhost:8080/aplicacionEnStruts/"

Por último, el equivalente en código XML de las expresiones es:

```
<jsp:expression>
Expresión Java
</jsp:expression>
```

Declaraciones (<%! %>).

Las declaraciones en JSP nos permiten declarar variables y métodos que no generarán una salida, pero en conjunción con las expresiones o scriplets nos permiten ingresar código Java dinámico en nuestros JSP´s.

Las declaraciones en XML se escriben de la siguiente forma:

```
<%! Código Java%>
```

Un ejemplo de una combinación de scriplets y declaraciones es el siguiente:

```
<%! private int accessCount = 0; %>
 <%= ++accessCount %>!--cuenta el numero de accesos cada que se
reinicia
                      !-- el servidor
```

Y el equivalente en tags XML para las declaraciones es:

```
<jsp:declaration>
Código
</jsp:declaration>
```

Comentarios (<%-- --%>).

Este marcador sirve para agregar algún comentario a nuestro código para describirlo

3.10.- Directivas *<%@ %>*

Las directivas afectan directamente la estructura de nuestra clase servlet, su forma general es:

```
<%@  directive attribute=”value” %>
```

Donde también se puede combinar varios atributos en una sola directiva:

```
<%@  directive attribute1=”value”
      attribute2=”value”
      attribute3=”value”
      …..
      attributeN=”value” %>
```

Las directivas pueden ser page, include y taglib, que a continuación se describen.

Page

La directiva page nos permite importar las clases que vamos a utilizar dentro de ese JSP y sus atributos son los siguientes:

-import=*"package.class" o* ***import****="package.class1,...,package.classN".*

Sirve para especificar los paquetes y las clases que puede soportar nuestro JSP. Por ejemplo:

```
<%@ page import="Java.util.*" %>
```

El atributo import es el único que puede aparecer múltiples veces.

-contentType*=” MIME-Type” o "MIME-Type; charset=Character-Set"*

Esto específica el tipo MIME (*Multipurpose Internet Mail Extensions - Extensiones de Correo de Internet Multipropósito*), de la salida. El valor por defecto es text/html. Este valor de la directiva puede ser también especificado por el scriplet:

```
<% response.setContentType("text/plain"); %>
```

-isThreadSafe*="true|false".*

Un valor de true (por defecto) indica un procesamiento del servlet normal, donde múltiples peticiones pueden procesarse simultáneamente con un sólo ejemplar del servlet. Un valor de false indica que el servlet debería implementarSingleThreadModel, con peticiones enviadas serialmente o con peticiones simultáneas siendo entregadas por ejemplares separados del servlet.

-session*="true|false".*

Un valor de true (por defecto) indica que la variable predefinida session (del tipo HttpSession) debería unirse a la sesión existente si existe una, si no existe se debería crear una nueva sesión para unirla. Un valor de false indica que no se usarán sesiones, y los intentos de acceder a la variable session resultarán en errores en el momento en que la página JSP sea traducida a un servlet.

-buffer*="sizekb|none".*

Esto específica el tamaño del buffer para el JspWriter out. El valor por defecto es específico del servidor, debería ser de al menos 8kb.

***-autoflush**="true/false".*

Un valor de true (por defecto) indica que el buffer debería descargarse cuando esté lleno. Un valor de false, raramente utilizado, indica que se debe lanzar una excepción cuando el buffer se sobrecargue. Un valor de false es ilegal cuando usamos buffer="none".

***-extends**="package.class".*

Esto indica la superclase del servlet que se va a generar.

***-info**="message".*

Define un String que puede usarse para ser recuperado mediante el método getServletInfo.

-errorPage="url"

Específica una página JSP que se debería procesar si se lanzará cualquier Throwable que no fuera capturada en la página actual.

-isErrorPage="true|false".

Indica si la página actual actúa o no como página de error de otra página JSP. El valor por defecto es false.

-language="Java".

En algunos momentos, esto está pensado para especificar el lenguaje a utilizar. Por ahora, no debemos preocuparnos por él ya que Java es tanto el valor por defecto como la única opción legal.

La sintaxis XML para definir directivas es:

```
<jsp:directive.TipoDirectiva atributo=valor />
```

Include

La directiva include permite insertar dentro de nuestra página JSP código JSP o HTML guardado dentro de otro archivo. Su forma general es la siguiente:

```
<%@ include file="url relativa" %>
```

La URL especificada normalmente se interpreta como relativa a la página JSP a la que se refiere, pero, al igual que las URLs relativas en general, podemos decirle al sistema que interpreta la URL relativa al directorio home del servidor Web empezando la URL con una barra invertida. Los contenidos del fichero incluido son analizados como texto normal JSP, y así pueden incluir HTML estático, elementos de script, directivas y acciones.

Esta directiva es de mucha utilidad cuando se tiene código repetido que se debe de mostrar en todas nuestras páginas web, así no es necesario copiar y pegar, el código mejor se guarda en un archivo y luego es mandado llamar con esta directiva en cada página.

Taglib

La directiva taglib sirve para incluir en nuestra página el uso de acciones personalizadas o pertenecientes a un tercero, tal es el caso de los frameworks que incluyen sus propias bibliotecas de tags para realizar acciones.

La forma general de taglib es:

```
<%@taglib uri=”valor” prefix=”valor”>
```

Donde el parámetro uri (*Uniform Resource Identifier, identificador uniforme de recurso*) indica la librería que se va a utilizar y prefix es el alias con el que se van a manejar los tags.

3.11.- Acciones

Las acciones tienen una estructura de sintaxis XML que nos permite modificar el comportamiento de un servlet ya sea para usar componentes JavaBeans, hacer re-direccionamiento a otra página o insertar ficheros dinámicamente.

Las acciones utilizadas en JSP son las siguientes:

<jsp:include>

Esta acción permite insertar código HTML guardado en otro fichero es muy parecida a la directiva include solo que a diferencia de la directiva que inserta el fichero al momento de ser construido el servlet, esta acción lo inserta al ser mandada a llamar la página, esta directiva permite insertar cambios en el contenido de la página sin modificar la estructura de la misma.
Su sintaxis general es la siguiente:

```
<jsp:include page=" URL relativa" flush="true" />
```

El atributo page indica la URL relativa donde se encuentra el fichero que se va a insertar y el atributo flush específica si el flujo de salida de la página principal debería ser enviado al cliente antes de enviar el de la página incluida.

<jsp:forward>

Esta acción permite re-direccionar una petición a otra página donde el único atributo que contiene es page y en él se específica la página de destino, esta puede ser una dirección estática o dinámica generada en el momento de la petición.
En los siguientes ejemplos se muestra la forma estática y dinámica de la re-dirección:

```
<jsp:forward page="/utils/errorReporter.jsp" />
<jsp:forward page="<%= someJavaExpression %>" />
```

<jsp:plugin>

Esta acción permite el uso de elementos OBJECT o EMBED diciéndole al navegador que se va a ejecutar un applet utilizando el plug-in de Java.

<jsp:useBean>

Esta acción permite el uso de un Java Bean dentro de una página JSP, de esta forma pueden utilizarse las propiedades de los Beans o sus métodos dándole mayor soporte a una página JSP. La forma más sencilla de declarar un Beans es:

```
<jsp:useBean id="name" class="package.class" />
```

Las propiedades más importantes para declarar un Bean son id que le da un nombre específico al Bean y class que indica que clase se va a ejemplificar.

Otro atributo es scope que permite especificar el alcance de los Beans este puede tomar 4 valores posibles: page, request, session y application.

<jsp:setProperty>

Esta acción permite obtener e ir guardando valores de las propiedades de los Beans o algún otro parámetro que queramos ir arrastrando durante la ejecución de nuestra aplicación web, se puede utilizar en dos contextos uno es desde fuera de una acción useBean:

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName"
property="someProperty" ... />
```

La opción anterior guarda la propiedad sin importar si se ejemplificó el objeto, otro uso es para algún otro valor que quiera ser arrastrado durante nuestra aplicación.

Y la otra es dentro de un useBean:

```
<jsp:useBean id="myName" ... >
...
<jsp:setProperty name="myName"
property="someProperty" ... />
      </jsp:useBean>
```

De esta forma no se creará la propiedad si no crea el objeto

<jsp:getProperty >

Sirve para recuperar un valor que fue declarado como una propiedad con la acción setProperty perteneciente a un Bean y al mandarlo llamar lo muestra como una cadena en la salida del JSP, el siguiente ejemplo muestra su uso:

```
<jsp:useBean id="itemBean" ... />
...
<UL>
<LI>Number of items:
<jsp:getProperty name="itemBean" property="numItems" />
<LI>Cost of each:
<jsp:getProperty name="itemBean" property="unitCost" />
         </UL>
```

4.- STRUTS

4.1.- Instalación y configuración de un servidor de aplicaciones Apache TOMCAT

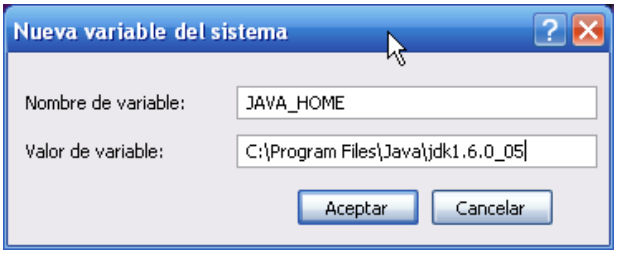
Antes de empezar a hablar acerca del framework STRUTS tenemos que hacer un pequeño espacio para hablar sobre un servidor de aplicaciones web que nos permita poder visualizar nuestras aplicaciones ya sean servlets, JSP´s o sistemas basados en el FrameWork STRUTS, para esto utilizaremos el servidor Apache TOMCAT el cual cumple con la especificación oficial de Sun MicroSystems, este servidor web puede descargarse de la página oficial de la Apache Foundation *http://tomcat.apache.org/* para este proyecto utilizaremos la versión binaria comprimida en archivo zip para Windows **version** 6.0.16 que cumple con las especificaciones **Servlet** 2.5 y **JSP** 2.1.

Una vez descargado y descomprimido el Apache TOMCAT en la ruta que prefiramos, esta carpeta será la raíz y nos referiremos a ella como *$CATALINA_HOME*, debemos de asegurarnos que este instalado el JDK (Java Development Kit), que puede ser descargado de *http://java.sun.com/javase/downloads/index.jsp* aquí utilizaremos el JDK 6 Update 5.

La estructura interna del directorio $CATALINA_HOME es la siguiente:

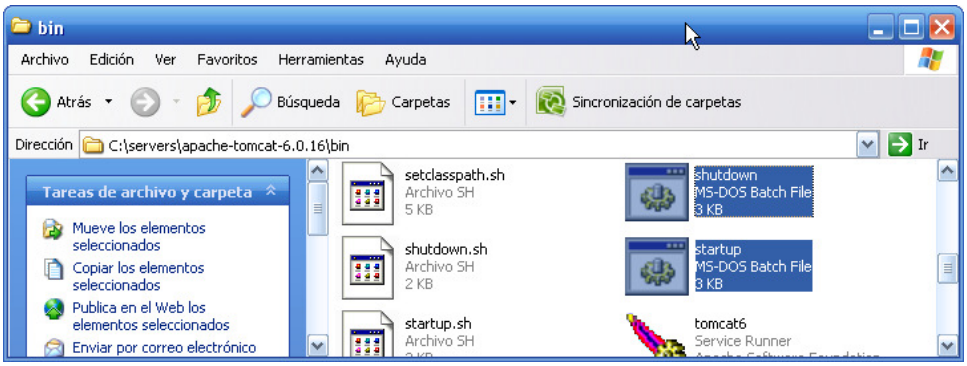
/bin - arranque, cierre, y otros scripts ejecutables
/temp – archivos temporales
/conf – archivos XML para la configuración de apache-tomcat siendo el más importante server.xml.
/logs – archivos de registro (log) de apache-tomcat.
/webapps - directorio que contiene las aplicaciones web
/work - almacenamiento temporal de ficheros y directorios

Como primer paso hay que crear la variable de entorno JAVA_HOME donde indicaremos el directorio donde se encuentra instalado nuestro JDK por ejemplo:

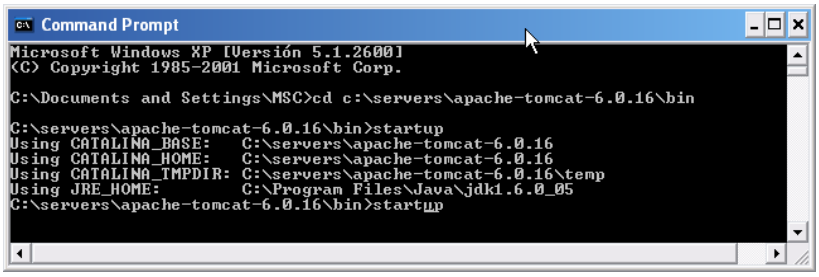


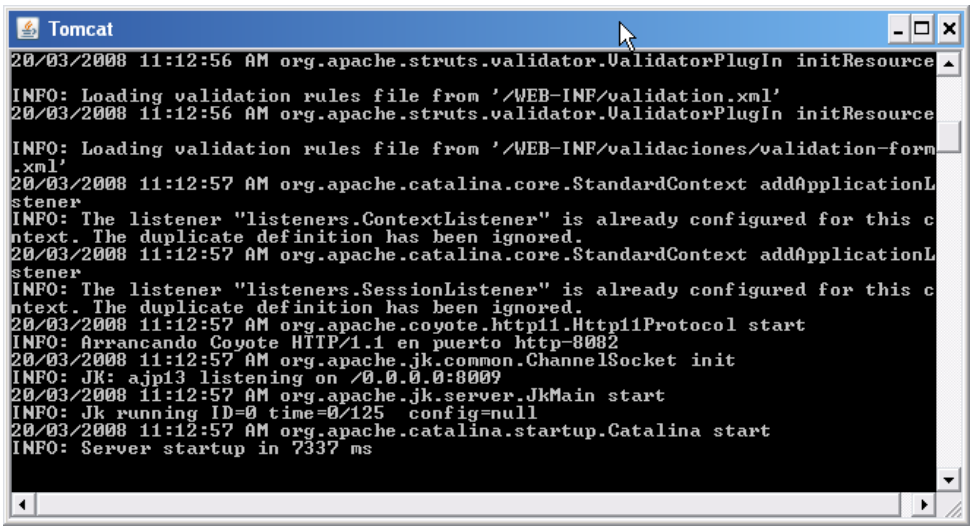
Para acceder a la creación de las variables de entorno entramos a las propiedades del sistema con Windows+Pausa>Opciones Avanzadas>Variables de Entorno>Nueva

El siguiente paso es localizar los archivos startup y shutdown que son archivos por lotes que permiten iniciar y detener el servidor respectivamente y estan dentro de $CATALINA_HOME\bin:



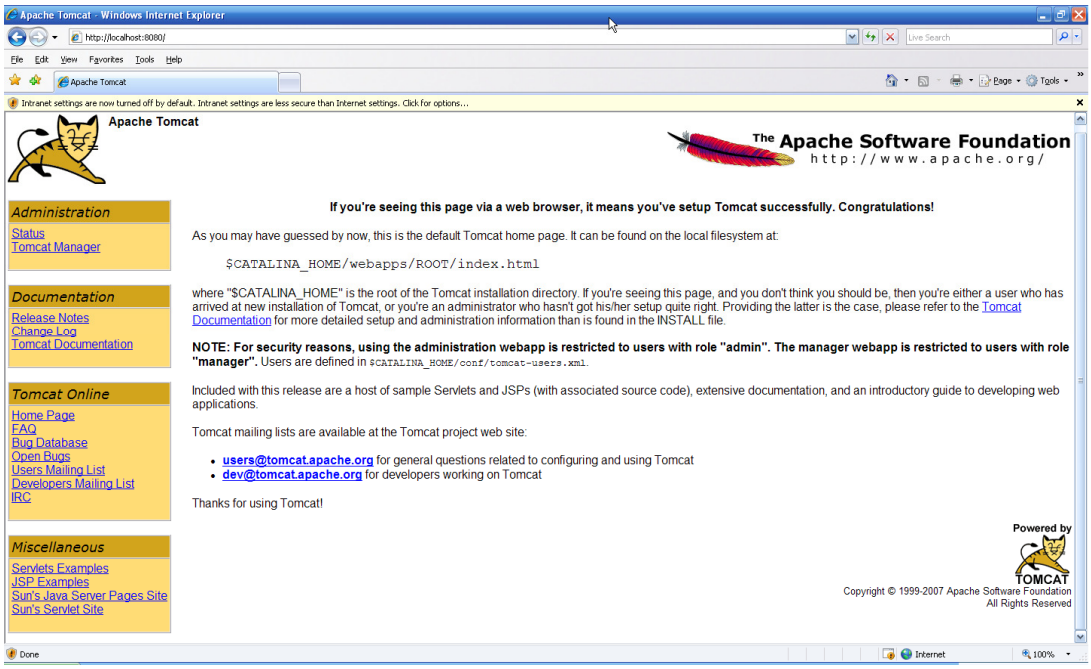
Una vez ejecutado startup empezará correr nuestro servidor:





Para tener acceso al servidor solo hay que abrir nuestro navegador de páginas web debemos teclear la URL que tendría la siguiente forma:

http://{host}:{port}/ = donde {host}{port} representa el hostname y el puerto donde corre apache-tomcat, entonces para nuestro caso quedaría http://localhost:8080/ que nos mostrará la página de bienvenida de apache-tomcat:

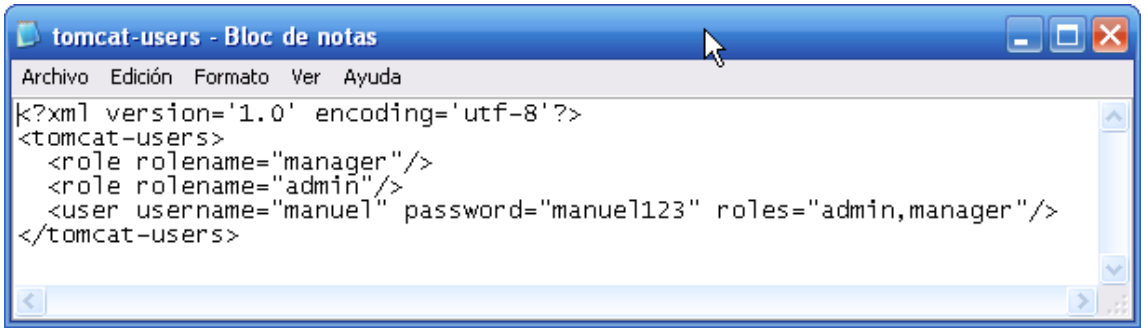


Para acceder a la gestión de aplicaciones tenemos que crear un usuario con los permisos necesarios, esto se hace editando el archivo tomcat-users.xml que se encuentra en $CATALINA_HOME\conf

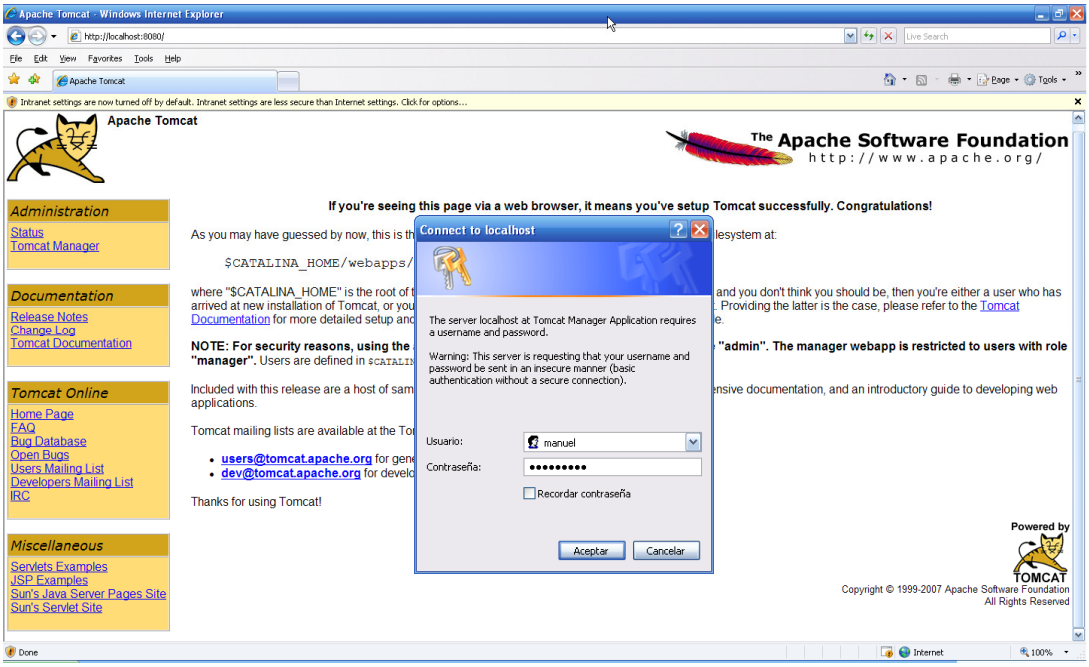
Agregando las líneas:

```
<role rolename="manager"/>

<role rolename="admin"/>

<user username="manuel" password="manuel123" roles="admin,manager"/>
```

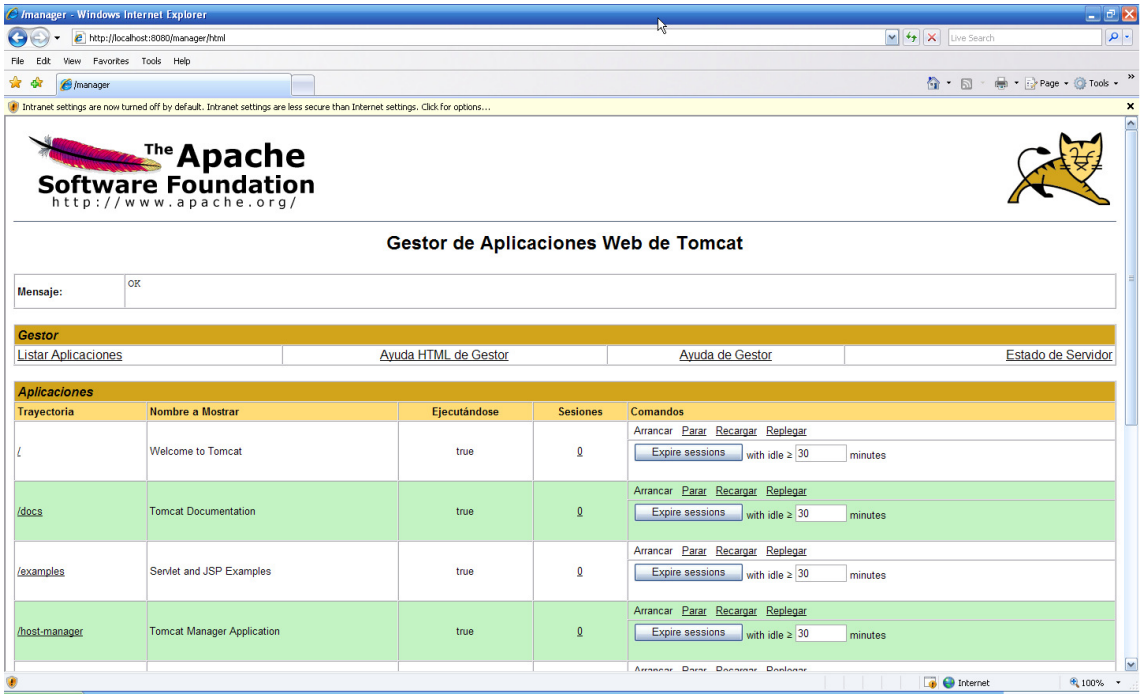
Podemos crear un usuario con rol de manejador y administrador:



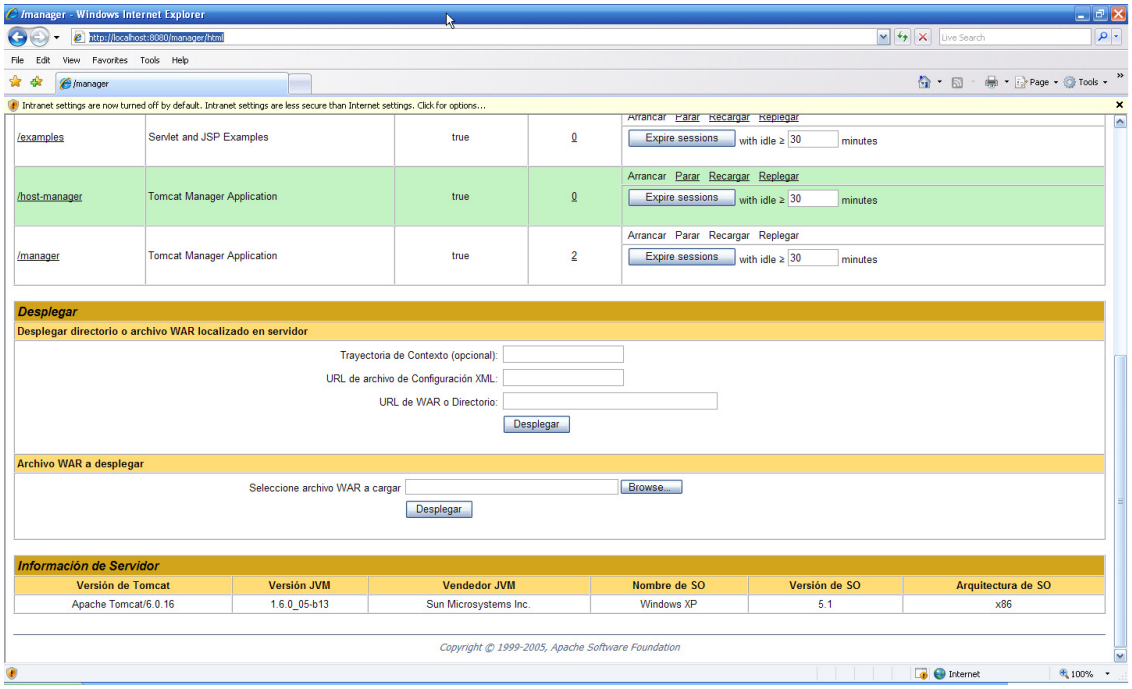
Guardamos los cambios y reiniciamos el servidor entramos a la página de bienvenida y entramos en el Tomcat Manager introduciendo el login y el password:



Al entrar en el manager podemos Arrancar, Parar, Recargar o Replegar nuestras aplicaciones web, así como poner el tiempo de expiración de una sesión y ver información como cuantas sesiones están
abiertas en ese momento y si nuestra aplicación se está ejecutando:

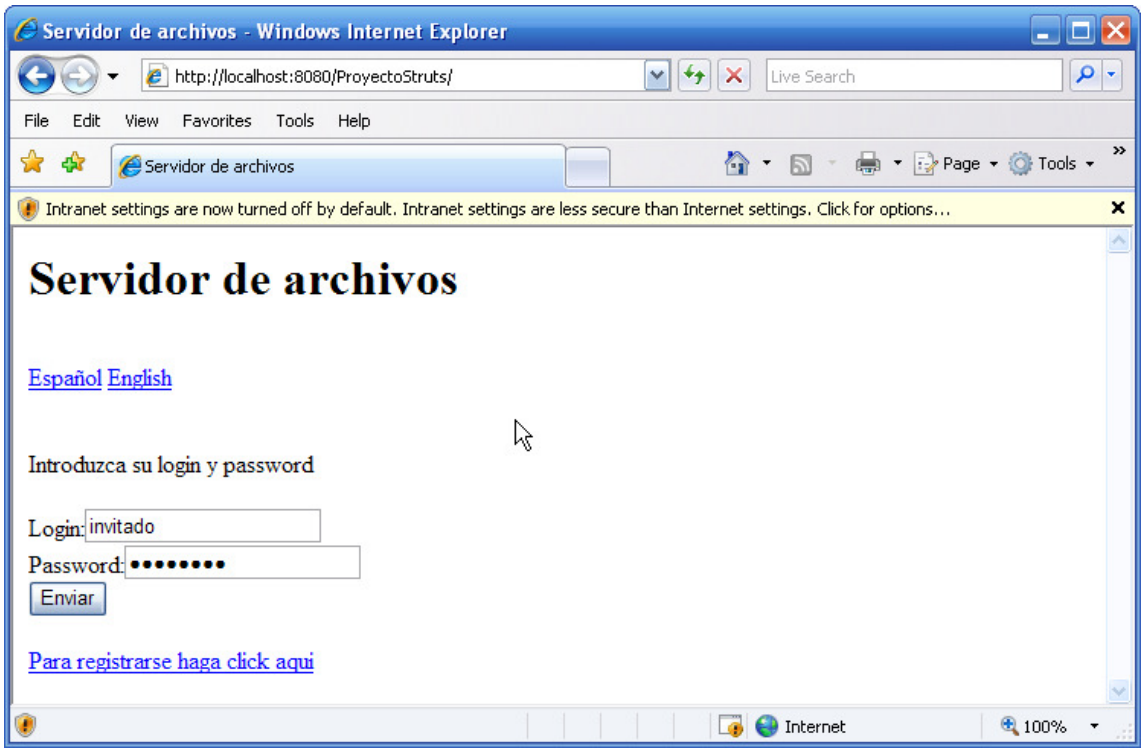


La forma más fácil de subir una aplicación web a nuestro servidor es utilizando la pestaña de desplegar de nuestro manager que nos permite introducir la dirección donde se encuentra la carpeta que contiene nuestro proyecto o archivo .war o la opción que nos permite explorar el disco en busca de un archivo war para luego ser desplegado en el servidor:



Por último para entrar a nuestra aplicación debemos introducir el host, el puerto y la ruta de nuestro proyecto por ejemplo:

http://localhost:8080/ProyectoStruts/



4.2.- Introducción al Framework STRUTS

Antes que nada necesitamos definir la palabra framework (marco de trabajo), este es una estructura que nos sirve para realizar de una forma más eficaz y en base a un modelo, el desarrollo de un proyecto de software, esto pude incluir soporte de programas o bibliotecas para conseguir su fin, el uso de un framework permite a un programador dedicar más tiempo a la identificación de requerimientos que a la programación de detalles.

El frameworks STRUTS se podría considerar como el siguiente paso de los servlets y JSP´s para el desarrollo de aplicaciones web. STRUTS es un framework que integra el uso del MVC (Modelo-Vista-Controlador) que como se mencionó en capítulos anteriores, permite la separación de componentes en objetos de vista, de control y modelo de datos, esta separación permite que los componentes de nuestra aplicación puedan ser modificables y extensibles de forma eficiente.

Analizando a profundidad el MVC tenemos que el controlador es el manejador del flujo de nuestra aplicación, este puede delegar a un sub-controlador adecuado el manejo de una solicitud, estos manejadores están ligados a un modelo que se encarga de representar una entidad y tiene los métodos apropiados para el manejo de la misma, el controlador también se encarga de refrescar los datos que contiene el modelo y que se presentan en la vista para que el usuario pueda interactuar con ellos.

El FrameWork STRUTS integra los tres tipos de objetos, los de control representados por servlets y que son proporcionados por el propio STRUTS, la vista, formada de JSP´s y la lógica de negocio o modelo, representada generalmente por Java Beans.

El servlet controlador se encarga de direccionar todas las solicitudes HTTP de los demás objetos como las JSP´s y los objetos Action que pertenecen a la clase org.apache.struts.action.Action proporcionada por el propio STRUTS. Al inicializar el controlador se analiza el archivo de configuración de recursos que define el mapeo con la clase

org.apache.struts.action.ActionMapping, el mapeo se utiliza para convertir solicitudes HTTP en acciones de aplicación.

Un ActionMapping está compuesto por:

- Un path solicitado (o "URI").
- El tipo de objeto (subclase de Action) para actuar sobre la solicitud.
- y otras propiedades según se necesite.

El objeto Action es el encargado de responder a la solicitud del cliente que normalmente es un navegador web y también se encarga de re-direccionar la solicitud hacia el lugar a donde debe de ser procesada.

Los objetos del tipo Action pueden utilizar los métodos de servlet controlador esto permite que se puedan enviar otro tipo de objetos de forma indirecta como pueden ser Java Beans.

Los objetos Action pueden pasar el control de una colección de datos a otro mapeo lo que permite que estas colecciones puedan ser mostradas en una JSP, de esta forma el usuario que levante una sesión dentro del sistema tendrá sus propios elementos con sus respectivos valores mientras esté viva su sesión, los Action interactúan con los Java Beans y sus propiedades que se encargan de modelar la lógica de negocio, la forma en la que un Action manda llamar las propiedades del Java Bean sin saber cómo funciona esté, permite encapsular la lógica de negocio en el JavaBean, permitiendo al Action concentrarse en el manejo de errores y en el flujo de la aplicación.

Los Java Beans también pueden ser utilizados en los formularios ligándolos a una clase del tipo org.apache.struts.action.ActionForm con esto podemos almacenar los valores de un formulario de entrada para validar los datos y que estos no se pierdan al recargar una página, este Bean se graba dentro de las colecciones estándar y puede ser posteriormente usado por un objeto Action para su procesamiento. El mismo STRUTS incorpora dentro de los Action un mecanismo que puede validar campos y mostrar mensajes de error.

Todos los Beans de formulario necesitan estar declarados en la configuración de STRUTS donde se introduce el nombre del fichero del Bean y el ActionMapping con el que se quiere enlazar. Cuando se manda llamar a un Action que utiliza un Bean de formulario, nuestro controlador principal crea una instancia del Bean y se lo pasa al Action que checa el contenido del Bean antes de que este se muestre junto con los mensajes que va a manejar el Bean del formulario. Después el Action devuelve el control con un reenvió y lo muestra en una JSP.

Otra ventaja que tenemos dentro de STRUTS es el uso de etiquetas personalizadas que rellenan automáticamente los campos de un formulario solo con identificar los nombres de los campos apropiados.

Las etiquetas personalizadas también sirven para la internacionalización de nuestras aplicaciones mediante el uso de ficheros que contienen todos los mensajes y etiquetas de nuestra aplicación en diferentes idiomas que luego, dependiendo del idioma o país del cliente se cargará el recurso correcto.

Para algunas aplicaciones simples se puede integrar la lógica de negocio dentro del Action, pero la forma correcta es que un Action llame a un Java Bean para realizar la lógica de negocio real. Con esto el Action solo se encarga del manejo de errores y del control de flujo; debe de existir una separación de los JavaBeans que contienen la lógica de negocio ya que si se quiere reutilizar este código es necesario que los JavaBeans no hagan referencia a ningún objeto web, para esto el Action debería traducir los detalles necesario para que solo se manden a llamar las variables de la lógica de negocio como variables normales.

Un ejemplo de lo anterior es una consulta a una base de datos para obtener información, primero el Java Bean se encarga de conectar con la base y consultar la información requerida, el mismo Bean se encarga de devolver el resultado al objeto Action y el objeto Action almacenará el resultado en un Bean de formulario que está ligado a una JSP que mostrará el resultado en un formulario HTML.

La separación de componentes permite que el Action y el JSP no necesiten saber de dónde viene el resultado, solo necesitan saber cómo manejarlo y como mostrarlo respectivamente.

4.3.- Modelo

El modelo dentro de un sistema MVC puede dividirse en dos componentes, los "conceptos" que representan el estado interno del sistema y las "acciones" que pueden realizarse para cambiar el estado. Estos estados y valores manejados dentro de los JavaBeans junto con las acciones encierran la lógica de negocio de nuestra aplicación.

En una aplicación tenemos un estado interno del sistema representado por varios JavaBeans y a través de los valores que tomen estos JavaBeans sabremos con detalle el estado de nuestra aplicación. Estos JavaBeans pueden ser más complejos o menos complejos dependiendo del tamaño de nuestra aplicación, estos JavaBeans pueden contener los métodos necesarios para hacer persistencia de datos y guardar su estado actual, este es el caso menos complejo, en algunos casos en los que nuestra aplicación es mucho más compleja es recomendable separar los Beans de estado y los Beans de acceso a datos.

Otra forma de integrar la lógica de negocio en aplicaciones de menor escala es dentro de los Action que forman parte del controlador, este método puede utilizarse cuando la lógica de negocio no va ser reutilizable y aunque Struts permite esto, lo más recomendable es hacer la separación de elementos pertinente.

4.4.- Vista

La vista dentro de una aplicación web esta generalmente construida con tecnología JSP o servlet, en el caso de Struts la vista es compuesta de JSP´s que utilizan tanto código HTML estático como código dinámico insertado a través de las etiquetas estándar de JSP, o con nuestras propias etiquetas personalizadas que podemos programar y reutilizar en varios proyectos.

Uno de los puntos más fuertes de Struts son las extensas librerías de etiquetas personalizadas (Custom Tags) que permiten la creación de interfaces graficas a través de elementos que conviven con los Beans ActionForm que a su vez forman parte del modelo.

El uso de las etiquetas de Struts permite tener asociado un ActionForm, este ActionForm tiene asociado a su vez un JavaBean, con esto cada que recarguemos nuestra página los valores contenidos en el JavaBean serán mostrados en la JSP.

4.5.- Controlador

El controlador dentro del modelo MVC es el encargado de recibir las solicitudes del cliente para luego distribuirlas al objeto que se encargará de aplicar la lógica de negocio y después pasando la responsabilidad al siguiente elemento de la vista según sea el caso. Dentro del marco de trabajo Struts, el componente principal es el controlador, este controlador está compuesto por un objeto ActionServlet. Este servlet define un conjunto de objetos ActionMaping, estos a su vez definen un path que se comparará con la URI (Unifom Resource Identifier – Identificador de recurso uniforme) de la solicitud entrante y normalmente apunta hacia una clase Action. Todos los objetos Action son subclases de la clase org.apache.struts.action.Action que son los

encargados de encapsular la lógica de negocio, interpretar la salida y pasar el control al objeto vista correspondiente.

Otra característica interesante de Struts es la habilidad de soportar objetos del tipo ActionMapping que permiten mapear otro tipo de valores asociados a la vista y almacenarlos para su procesamiento posterior. Otra característica de Struts es el uso de nombres lógicos para los controles, de esta forma se puede preguntar por una página llamada "main" y así ocultar su verdadero nombre y ubicación; permitiendo separar el control de la vista.

4.6.- Construcción de los Componentes del Modelo

En el levantamiento de requerimientos a veces hay un gran enfoque en la parte de la vista, dejando un poco de lado la construcción de nuestro modelo, lo cual es un error ya que nuestro modelo tiene una fuerte conexión con la información que va a ser procesada en cada solicitud, el desarrollo del modelo se hace mediante la construcción de clases JavaBeans que permitirán dar la funcionalidad a nuestra aplicación, la construcción de estos JavaBeans va de la mano con los requerimientos de la aplicación, estos JavaBeans pueden clasificarse en varias categorías que se describirán más adelante. Para adentrarnos en la clasificación de los JavaBeans primero tenemos que hacer un repaso al concepto de "ámbito" que relaciona los Beans y JSP´s.

Los JavaBeans y el ámbito

En una aplicación web que hace uso de JavaBean tenemos reglas sobre el tiempo de vida que tendrán los datos contenidos en estos objetos y la visibilidad de estos, esta regla se conoce como ámbito. Dentro de la especificación JSP tenemos los siguientes ámbitos:

- **page** – Hace a los Beans visibles solo dentro de la página JSP donde es declarado, para el tiempo de vida de la solicitud actual.
- **request** - Hace a los Beans visibles dentro de la página JSP donde es declarado, así como en cualquier página o servlet que esté incluido en esta página, o reenviado por esta página.
- **session** – Hace que los Beans sean visibles para todas las páginas JSP y servlets que participan en una sesión de un usuario en particular, a través de una o más solicitudes.
- **application** – Hace que los Beans sean visibles para todas las páginas JSP y los servlets que forman parte de una aplicación Web.

Dentro de las aplicaciones Web, que se conforman de JSP´s y servlets, podemos incluir nuestros JavaBeans almacenándolos como un atributo request como se muestra a continuación:

```
Autor aut = new Autor(...);
request.setAttribute("Autor", aut);
```

Este Bean puede ser visible en la siguiente página JSP utilizando la etiqueta estándar <jsp:useBean> como se muestra en el siguiente codigo:

```
<jsp:useBean id="Autor" scope="request"
class="com.myAplication.Autor"/>
```

Beans "ActionForm"

Un Bean del tipo "ActionForm" está ligado a la vista todavía más que al modelo mismo. En el marco de trabajo Struts, normalmente se definen Beans "ActionForm" que no son otra cosa que una clase Java que extiende de la clase ActionForm perteneciente al marco de trabajo Struts, esta clase es utilizada para ser ligada a un formulario de entrada de datos en nuestra aplicación, también se les conoce como "Beans de formulario", estos Beans al ser declarados

dentro de nuestro archivo de configuración ActionMapping[1], permiten que el servlet controlador de Struts realice las siguientes acciones antes de ser mandado llamar el Action correspondiente:

- Verifica que dentro de una sesión de usuario exista un ejemplar de un Bean con la clase y clave apropiada.
- Si el Beans no existe o no está disponible dentro de una sesión de usuario, crea uno nuevo y lo añade a dicha sesión.
- Cuando requerimos mandar a llamar un parámetro de un Bean, automáticamente se mandará llamar a su método set() para obtener el valor.
- El Bean ya actualizado será pasado al método perform() de la clase Action para que los valores estén disponibles inmediatamente.

Al momento de codificar nuestros Bean ActionForm debemos tener en cuenta estos principios:

- Una clase ActionForm solo debe de tener métodos set y get para las propiedades que se requieren en el formulario, no es necesario emplear ningún otro método que implemente lógica de negocio.
- Un objeto del tipo ActionForm tiene un mecanismo de validación para las propiedades que contiene, esto se hace mediante la sobre escritura del método "stub" donde también implementaremos los mensajes de error del recurso, una vez hecho esto Struts validará automáticamente nuestro formulario, usando el método que sobrescribimos. Aunque para validar usuarios también podemos utilizar un objeto Action, como se verá más adelante.
- Al definir una propiedad dentro del ActionForm debemos siempre generar a cada propiedad su método get y set con el nombre de la propiedad según las convenciones usuales. Por ejemplo una propiedad que se llama nombreAutor deberá tener el método setNombreAutor() y getNombreAutor().
- También debemos de tener en cuenta que todas las propiedades de nuestro Bean ActionForm tenga valores razonables, esto mediante método validate(), ya que si el Bean falla en la validación no será tomado en cuenta en el Action.
- Otra cosa a considerar es la propiedad de poder anidar Beans dentro de nuestro Bean ActionForm, por ejemplo podemos tener dentro de nuestro ActionForm un Bean llamado dirección con los campos calle. numero y colonia, donde en nuestra página JSP accederemos a la propiedad del Bean dirección mediante el operador "." Por ejemplo direccion.calle, que mandará llamar al método setCalle().

Cuando nos referimos a un formulario, nos referimos a un conjunto de propiedades con algo en común, este formulario no necesariamente necesita estar contenido en la vista en una sola página JSP, pudiéndose extender a través de múltiples páginas, cuando este es el caso Struts aconseja que todas las propiedades sean definidas en un solo ActionForm y que sean reenviadas a la misma clase Action. Con esto podemos cambiar la presentación de los campos en diferentes páginas web sin modificar el ActionForm.

Beans de Estado del Sistema

Para representar el estado actual en que se encuentra el sistema se utilizan una serie de clases JavaBean, en donde definimos las propiedades que nos ayudarán a representar el estado actual del sistema y de los usuarios que hayan levantado una sesión dentro del él. Un ejemplo podría ser un Bean que represente un carrito de compras en una tienda virtual, donde se incluirán todos los ítems que se vayan seleccionando, también podemos tener otros Beans para guardar la información del perfil del usuario o para tener el catálogo de artículos disponibles y su nivel de inventarío.

[1] Véase Construir los Componentes del Controlador pag: 62

Cuando tenemos una aplicación pequeña, o para información que no necesita estar presente durante toda la aplicación, podemos utilizar algunos Beans que puedan contener todos los detalles sobre el estado del sistema o como en el caso más frecuente, los Beans del estado del sistema guardarán información almacenada en un medio permanente de datos y serán creados o eliminados según convenga.

Beans de Lógica de Negocio

Los Beans de la lógica de negocio son los encargados de encapsular la lógica funcional de nuestra aplicación, mediante llamadas a métodos, estos métodos podrían estar dentro de las mismas clases que se utilizan para el estado del sistema o estar en clases separadas exclusivamente dedicadas a la lógica, cuando es el caso anterior necesitamos pasarle los Beans de estado como parámetro para que sean procesados por los métodos de la lógica.

Para realizar la separación entre los elementos de nuestra aplicación y con esto poder reutilizar el código, los Beans de la lógica de negocio deben ser implementados de tal forma que no sepan que están siendo utilizados en una aplicación web. En el momento que necesitamos una clase contenida dentro del API Servlet, estamos ligando el Bean a una aplicación web, teniendo esto en cuenta, podemos reutilizar nuestra lógica de negocio en otros ámbitos que no sean web.

La complejidad de nuestra aplicación nos lleva a la utilización de simples Beans con métodos de la lógica de negocio que interactúan con Beans del estado del sistema que son pasados como argumentos o JavaBeans que acceden a medios permanentes mediante llamadas JDBC en su caso menos complejo. Para grandes aplicaciones, podemos utilizar Enterprise JavaBeans (EJB´s) en lugar de simples Beans.

Acceder a Bases de Datos Relacionales

El marco de trabajo Struts permite tener un pool de conexiones JDBC y definir las propiedades de las fuentes de datos dentro de un archivo de configuración estándar.

Después de definir las fuentes de datos para nuestra aplicación necesitamos definir la conexión dentro del método perform de la clase Action, como se muestra a continuación:

```
public ActionForward
       perform(ActionMapping mapping,
 ActionForm form,
 HttpServletRequest request,
 HttpServletResponse response)
{
 try {
   Javax.sql.DataSource dataSource =
     servlet.findDataSource(null);
   Java.sql.Connection myConnection =
     dataSource.getConnection();

   //Aquí va el código para manipular nuestros datos

 } catch (SQLException sqle) {
   getServlet().log("Connection.process", sqle);
 } finally {

   //Una vez finalizadas nuestras consultas debemos de asegurarnos de
   // cerrar la conexión
   try {
     myConnection.close();
   } catch (SQLException e) {
```

```
      getServlet().log("Connection.close", e);
    }
  }
}
```

El pool de conexiones que ofrece Struts es un opcional, ya que para aplicaciones a gran escala lo más conveniente es usar otros almacenes de conexiones o incluso un Framework como Hibernate que ofrecen un mejor rendimientos en sistemas con muchas transacciones.

4.7.- Construcción de los Componentes de la Vista

Introducción

A continuación veremos como el marco de trabajo Struts crea la **Vista** de una aplicación, esto se hace mediante la tecnología JSP. Struts tiene una serie de componentes que nos permite crear y ligar nuestras interfaces de usuario de una forma muy sencilla, además de ofrecer la internacionalización de nuestras aplicaciones.

Mensajes Internacionalizados

Anteriormente las aplicaciones solo eran accesadas por los residentes de un país, por lo tanto solo contaban con uno o dos idiomas, pero con la aparición de las aplicaciones web que se despliegan en internet, se ha abierto el acceso a todo el mundo, por lo cual se ha hecho necesaria la creación de las aplicaciones con soporte de internacionalización y localización.

Struts permite construir aplicaciones internacionalizadas y localizadas, mediante la utilización de diversas clases Java, algunas de estas son:

- **Locale**- La clase Java que nos permite la internacionalización es `Java.util.Locale`. Cada objeto `Locale` guarda la información de la elección del idioma, la ubicación y opciones de formateo para fechas y números.
- **ResourceBundle** - La clase `Java.util.ResourceBundle` se utiliza para dar el soporte de mensajes en varios idiomas.
- **PropertyResourceBundle**– Es una de las implementaciones estándar de `ResourceBundle` para definir varios archivos de recursos con el mismo nombre para guardar todos los mensajes que utilizaremos en una aplicación web.
- **MessageFormat** - La clase `Java.text.MessageFormat` permite reemplazar partes de un string con los argumentos guardados en un archivo de propiedades, esto permite crear sentencias dinámicas sustituyendo el contenedor {0}, por el primer argumento, {1}, por el segundo argumento y así sucesivamente.
- **MessageResources** - La clase `org.apache.struts.util.MessageResources` sirve para declarar un recurso de base de datos y también permite establecer cuál será el idioma que se mostrará en una localidad en particular sin importar la localidad por defecto del servidor.

Para crear una aplicación con internacionalización hay que seguir los siguientes pasos:

Tenemos que crear dentro del paquete donde se encuentra nuestro código fuente por ejemplo: `com.mycompany.mypackage`, dentro de este paquete hay que crear un paquete de recursos `com.mycompany.mypackage.MyResources`, por ejemplo:

- **MyResources.properties** – Aquí definimos todos los mensajes del idioma que queremos que sea el idioma por default de nuestra aplicación.

- **MyResources_xx.properties** – Tenemos los mismos mensajes que en el otro archivo utilizando el código de idioma ISO "xx" cambiando los mensajes del archivo de propiedades.

Para utilizar estos archivos de propiedades debemos declararlos dentro del servlet controlador en el descriptor de despliegue, donde tenemos que definir como parámetro la ruta completa de nuestro archivo de propiedades, como se muestra a continuación:

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-
class>
  <init-param>
    <param-name>application</param-name>
    <param-value>com.mycompany.mypackage.MyResources</param-value>
  </init-param>
  <.../>
</servlet>
```

Interacciones de Forms y FormBean

Otra capacidad que ofrece Struts para construir formularios basados en código HTML es el seguimiento de errores en el llenado de los campos del formulario, que además de mostrar los errores permite que el usuario no tenga que llenar de nuevo todo el formulario, tan solo tiene que llenar los campos donde se produjeron errores.

Utilizar solo tecnología JSP puede crear confusión en los programadores HTML que no están adiestrados en la programación Java, en el siguiente ejemplo que muestra como declarar un campo de entrada llamado "username" con tecnología JSP:

```
<input type="text" name="username"
      value="<%= loginBean.getUsername() %>"/>
```

Struts simplifica este código de la siguiente forma:

```
<html:text property="username"/>
```

Con este código podemos referirnos al campo con un alias sin tener que mandar llamar el JavaBean, esto lo hará automáticamente el marco de trabajo. Este código es más claro para un programador HTML.

Otra propiedad de Struts es el manejo de formularios que van a través de varias páginas conocidos como multiparte, este tipo de formularios son más difíciles de manejar en código HTML pero con Struts estos formularios son manejados como formularios normales.

Construir Formularios con Struts

A continuación se mostrará un ejemplo completo de un formulario utilizando los tags de Struts, este formulario es un ejemplo de login que utiliza varias de las características que puede ofrecer el marco de trabajo.

```
<%@ page language="Java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"
        prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-Bean.tld"
```

```
        prefix="Bean" %>
<html:html>
<head>
<title>
  <Bean:message key="logon.title"/>
</title>
<body bgcolor="white">
<html:errors/>
<html:form action="/logon" focus="username">
<table border="0" width="100%">
  <tr>
    <th align="right">
      <html:message key="prompt.username"/>
    </th>
    <td align="left">
      <html:text property="username"
       size="16"/>
    </td>
  </tr>
  <tr>
    <th align="right">
      <html:message key="prompt.password"/>
    </th>
    <td align="left">
      <html:password property="password"
           size="16"/>
    </td>
  </tr>
  <tr>
    <td align="right">
      <html:submit>
        <Bean:message key="button.submit"/>
      </html:submit>
    </td>
    <td align="right">
      <html:reset>
        <Bean:message key="button.reset"/>
      </html:reset>
    </td>
  </tr>
</table>
</html:form>
</body>
</html:html>
```

A continuación describiremos las características más importantes en la creación de formularios utilizando Struts basados en el ejemplo anterior.

- La directiva `taglib` indica al compilador donde debe de buscar la librería que contiene las etiquetas que se usarán en esa página, para no utilizar toda la nomenclatura de la librería se utiliza un prefijo que nos ayudará a identificarla en este caso es la palabra "Bean" para la librería `struts-Bean`, y "html" para la librería `struts-html`. Aunque es posible utilizar cualquier prefijo que queramos.
- También se hace uso de los mensajes internacionalizados guardados en los `MessageResources` las claves utilizadas para acceder a estos mensajes son:
  - **logon.title** - Título de la página de login.
  - **prompt.username** - Un string para pedir el "Username:"
  - **prompt.password** - Un string para pedir la "Password:"
  - **button.submit** - Etiqueta para el botón "Submit"
  - **button.reset** - Etiqueta para el botón "Reset"

Cuando entramos a la aplicación el objeto Locale es el encargado de seleccionar el archivo de propiedades apropiado para mostrar los mensajes en ese idioma y tan solo con cambiar el objeto Locale almacenado, cambiaremos el idioma de nuestros mensajes.

- Utilizando las banderas de error podemos mostrar todos los mensajes de error que se hayan suscitado al validar los campos o por un componente de la lógica de negocio.
- Con la etiqueta `form` que será renderizada en un elemento `<form>` HTML, dentro de esta etiqueta asociamos todos los campos con los atributos del `FormBean` con ámbito de sesión, este Bean se almacena bajo la clave `logonForm`.
- Otra etiqueta que se utilizó en este ejemplo es "text" que se convertirá en un elemento `<input>` de HTML del tipo "text". Cuando ejecutamos esta página, el campo asociado a la propiedad `username` del Bean correspondiente tendrá el valor devuelto por el método `getUsername()`.
- La etiqueta `password` funciona de igual forma que la anterior solo que muestra el contenido con asteriscos para teclear el password.
- Las etiquetas `submit` y `reset` crean los botones correspondientes automáticamente con los archivos de mensajes utilizados para internacionalización.

Tipos de Campos de Entrada Soportados

El marco de trabajo **Struts** tiene un gran número de etiquetas disponibles para la creación de formularios basados en HTML para todos estos tipos de campos de entrada:

- Checkboxes
- Campos hidden
- Campos de entrada password
- Botones de radio
- Botones de reset
- Listas select con opciones embebidas o ítems de opciones.
- Option
- Options
- Botones submit
- Campos de entrada de texto text
- Textareas

Para utilizar todos estos elementos deben de estar contenidos dentro de una etiqueta `form`, de esta forma podemos ligar los campos con los Beans correspondientes.

Otras Etiquetas Útiles de Presentación

Además de las etiquetas para la creación de formularios, tenemos algunas más para presentar la información en nuestras páginas web, algunas de estas se muestran a continuación:

- **[logic] itérate**.- Permite imprimir cada elemento de una colección como un Enumeration, un Hastable, un Vector o un array de objetos dentro de la página web.
- **[logic] present.-** Sirve para especificar un atributo en la etiqueta y si este está presente en dentro de esta hará cierta operación.
- **[logic] notPresent.-** Esta etiqueta funciona del mismo modo que la de no present solo que la condición es que el elemento no esté presente en la etiqueta.
- **[html] link.-** Genera un link basado en HTML (<a>)
- **[Bean] parameter.-** Sirve para recuperar el valor del parámetro solicitado.

Validación Automática de Formularios

Dentro del marco de trabajo Struts tenemos la oportunidad de validar campos de un formulario de forma automática, mediante la sobre escritura del método `validate()` que se encuentra en la clase `ActionForm`:

```
public ActionErrors
  validate(ActionMapping mapping,
    HttpServletRequest request);
```

Este método `validate()` es mandado llamar por nuestro servlet controlador una vez que son llenadas las propiedades de un Bean y antes de que se ejecute el método `perform()` de la clase Action. Dentro del método `valídate()` tenemos las siguientes opciones:

- Se ejecutan las validaciones del método y no se encuentra ningún problema. Cuando esto sucede se devuelve un valor `null` u objetos del tipo `Action Errors` con valor de 0 y después se mandará llamar el método `perform()` para realizar las operaciones necesarias.
- Se ejecutan las validaciones de los campos y se encuentran errores. Se devuelve un objeto `ActionErrors` con las claves de mensajes de error que están guardados dentro de los MessageResources para que sean mostrados. Después estos mensajes se almacenan como un array, el servlet controlador lo almacena como un atributo y lo imprime al ejecutar el tag <html:errors> y por ultimo devuelve el control al formulario para corregir los errores.

Otras Técnicas de Presentación

Utilizando todo el potencial de las tecnologías JSP y las librerías de Struts podemos construir cualquier aplicación web, pero implementando algunas técnicas podemos mejorar la calidad, la re-utilización de los componentes y la reducción de errores. A continuación veremos algunas de estas técnicas.

Etiquetas Personalizadas Específicas de la Aplicación

Otro aspecto importante de Struts es la posibilidad de crear nuestras propias etiquetas para una aplicación. Dentro de Struts tenemos un ejemplo de estas etiquetas, el código fuente está en el directorio, src/example, en el paquete org.apache.struts.example, junto con otras clases Java que son usadas por esta aplicación. Las etiquetas personalizadas que encontraremos en esta aplicación son:

- **checkLogon** – Sirve para verificar que exista una sesión de usuario levantada, de no existir reevia el control a una página para logearse, esto puede suceder cuando un usuario intente entrar a una página mediante un bookmark para evitar la seguridad de la aplicación.
- **linkSubscription** – Sirve para generar un link que nos llevará a una página para una suscripción, llevando atributos asociados al usuario.
- **linkUser** – Sirve para generar un link que nos llevará a una página que contiene la información del usuario.

Composición de Páginas con Includes

Otra técnica que nos permite diseñar páginas web de una forma más eficiente es mediante la construcción de una página mediante archivos JSP que se insertarán dinámicamente por ejemplo, en una página principal podemos insertar páginas o elementos con diferentes funcionalidades, como:

- Acceso a un motor de búsqueda.

- Uno o más displays "alimentadores de noticias" con los tópicos de interés.
- Acceso a tópicos de discusión relacionados con este portal.
- Un indicador de "mail esperando" si nuestro portal proporciona cuentas gratuitas de correo.

Desarrollando estos componentes por separado, podemos incluirlos mediante la etiqueta `include` que proporciona Struts. Existen 3 tipos de `include`, que dependiendo cuando queremos que ocurra la salida, utilizaremos uno u otro:

- La directiva `<%@ include file="xxxxx" %>`, con ella podemos declarar cualquier tipo de archivo Java o etiquetas de JSP, e incluso utilizar variables utilizadas en la página JSP.
- El include de **action** (`<jsp:include page="xxxxx" flush="true" />`), este código se procesa al momento de la solicitud, permitiendo incluirla dentro de una etiqueta condicional para mostrar, o no su contenido.
- La etiqueta "Bean:include" toma un argumento "forward" que representa un alias de un Bean que será impreso en la salida de la página.

4.8.- Construcción de los Componentes del Controlador

Introducción

Después de haber definido la forma en la que se construyen los componentes del Modelo y la Vista en una aplicación web, podemos pasar a ver los componentes del Controlador.

El marco de trabajo Struts tiene una clase servlet encargada del mapeo de una solicitud URI a una clase Action. Para implementar un controlador necesitamos ya tener establecidas las siguientes acciones:

- Haber declarado una clase `Action` por cada solicitud lógica que podría ser recibida, esta clase debe de ser extendida de la clase `org.apache.action.Action`.
- Configurar nuestro archivo de mapeo `struts-config.xml`, el cual está escrito en código XML, poniendo dentro de la etiqueta `ActionMapping`, cada una de nuestras solicitudes lógicas.
- Actualizar el fichero del descriptor de despliegue de la aplicación Web que también está escrito con código XML para que nos permita utilizar los componentes necesarios del marco de trabajo Struts.
- Añadir los componentes Struts apropiados a nuestra aplicación.

Clases Action

La clase `Action` define los métodos que podrían ser ejecutados dependiendo de nuestro entorno servlet, en el siguiente código podemos ver como se declara un método perform() y que parámetros recibe de la petición:

```
public ActionForward perform(ActionMapping mapping,
 ActionForm form,
 ServletRequest request,
 ServletResponse response)
  throws IOException, ServletException;
```

Un objeto del tipo Action tiene como finalidad el procesar una solicitud, mediante el método perform(), después el objeto Action regresará un objeto ActionForward que indica donde se debe reenviar el control de la aplicación, generalmente es una JSP que seguirá con el proceso o

una JSP con un mensaje de error. Siguiendo el patrón del MVC, una clase Action sigue los siguientes pasos dentro de una aplicación web:

- Validar el estado actual de la sesión del usuario, si la clase detecta que la sesión ha expirado o que el usuario intenta entrar de forma ilegal a la mitad de la aplicación por medio de la url u otro método, se le regresará a la página de inicio donde se le pedirá que se vuelva a logear.
- Si la validación del formulario no se ha completado, valida las propiedades del FormBean y si encuentra un problema almacena las claves de los mensajes de error dentro de la petición y le delega de nuevo el control al formulario para que pinte los mensajes de error y estos sean corregidos.
- Realizar el procesamiento de los datos que se obtuvieron de la petición, esto se logra mediante métodos escritos dentro del mismo Action o de la forma más apropiada, mediante métodos de los Beans de la lógica de negocio, un ejemplo seria el grabar un registro en una base de datos.
- Actualizar los objetos de estado del sistema que se encuentran del lado del servidor, que serán usados para crear la siguiente página de la interface de usuario, estos pueden tener un ámbito de sesión o de solicitud.
- Devolver un objeto ActionForward que identificará la página JSP que se usará para generar la respuesta una vez que los Beans de estado han sido actualizados.

Algunos problemas que se pueden suscitar cuando no diseñamos y creamos clases Action de forma correcta son:

- Debemos de codificar nuestra clase Action para que opere en un entorno multi-proceso, parecido a un método service() en un servlet. Esto debido a que el servlet controlador de Struts solo crea un ejemplar de la clase Action y este se usa para todas las solicitudes.
- Cuando los Beans que creamos para representar nuestro Modelo lanzan excepciones debido a algún problema como puede ser el acceso a una base de datos, se recomienda atrapar esas excepciones dentro del método perform() y guardarlas en un archivo log, mediante el `método servlet.log("Mensaje de excepción", exception)`
- Debemos de procurar solo mantener los datos necesarios para el flujo de nuestra aplicación dentro de una misma sesión de usuario, cerrando las conexiones a base de datos que ya no utilicemos o liberando los Beans que ya no sean necesarios, antes de reenviar la información de un Action al siguiente componente de la vista.

Otro punto muy importante a considerar, es la extensión de nuestra clase Action. Debemos de procurar que esta sea lo más corta posible, evitando codificar dentro de ella métodos de la lógica de negocio, esto hace que la clase Action quede en código duro, que hará difícil su comprensión y su re-utilización en aplicaciones posteriores, ya que estará dentro de un entorno web. La forma más recomendable es crear Beans de la lógica de negocio sin ninguna referencia hacia un entorno web y mandarlos llamar dentro de nuestro Action de forma independiente. El único caso en que se puede embeber la lógica de negocio dentro de un objeto Action es cuando nuestra aplicación es muy pequeña.

La Implementación de ActionMapping

Para que podamos ligar nuestros Beans, Actions y Beans de Formulario necesitamos configurar el archivo struts-config.xml que hace referencia a una interface Java llamada ActionMapping que contiene la información necesaria para mapear las URI. Las propiedades más importantes que debemos incluir son:

- **type** – Nombre completo de la clase Action, que usaremos en el mapeo. Debemos de incluir también el paquete donde está la clase, por ejemplo: "com.acciones.MyAccion".

- **name** - El nombre del Bean de formulario que fue definido en el mismo archivo struts-config.xml, con el que distinguiremos los distintos Beans de formulario.
- **path** – Es el path como se reconocerá la URI solicitada en este mapeo.
- **unknown** – Puesto en true, tomará este Action para que maneje todas las solicitudes que no tengan definido un Action, esta propiedad solo puede ser declarada para un solo Action dentro de la aplicación
- **validate** – Estando en true nos indica si debe de ser llamado el método validate() para hacer la validación de campos del Action asociado al mapeo.
- **forward** – Indica el path de la URI a donde será enviado el control de la aplicación una vez terminada de procesar la información por el Action.

Archivo de Configuración de los Mapeos de Action

El marco de trabajo Struts incluye un modulo llamado Digester que se encarga de leer código XML y crear los objetos necesarios en la aplicación web, el archivo XML de donde el Digester toma la configuración de nuestra aplicación web es nombrado struts-config.xml, que es responsabilidad del programador incluirlo dentro del directorio WEB-INF de la aplicación, el formato que debe de seguir este archivo está definido en el documento "**struts-config_1_0.dtd**", que puede ser consultado en http://jakarta.apache.org/struts/dtds/.

El archivo struts-config.xml, tiene como elemento principal la etiqueta `<struts-config>`, donde a partir de esta tenemos dos elementos importantes que son usados para describir nuestras acciones:

- **<form-Beans>**
  Dentro de esta etiqueta podemos definir los Beans que vamos a utilizar en nuestra aplicación. Utilizando el elemento <form-Bean> por cada Bean de formulario que queramos declarar, esta etiqueta tiene los siguientes atributos:
  - **name**: Identificador único para el Bean, este nombre sirve para diferenciarlo cuando los ligamos a un Action.
  - **type**: Es el nombre completo de la clase Java a la pertenece nuestro Bean de formulario, incluyendo el paquete donde está contenida.
- **<action-mappings>**
  Aquí es donde podemos definir nuestras acciones, mediante la etiqueta <action>, una por cada acción que vayamos a utilizar en nuestra aplicación. Los atributos de de esta etiqueta son:
  - **path**: El path de la clase A**ction** con la que nos referiremos en nuestra aplicación.
  - **type**: El nombre de la clase a la que pertenece este Action, incluyendo el paquete donde se encuentra contenida
  - **name**: El nombre del Bean de formulario que estará ligado con esta acción.

A continuación se muestra un ejemplo de un archivo struts-config.xml, donde podemos observar la declaración de un Bean de formulario y como se liga a una acción.

```
<struts-config>
    <form-Beans>
<form-Bean name="LoginForm" type="formulario.LoginForm"/>
  </form-Beans>
<action-mappings>
<action input="/jsp/index.jsp"
                name="LoginForm"
                path="/login"
                scope="request"
                type="acciones.Enlace">
         <forward name="paso" path="/jsp/menu.jsp"/>
         <forward name="nopaso" path="/jsp/index.jsp"/>
```

```
        <forward name="pasoAnonimo" path="/jsp/menuA.jsp"/>
</action>
</action-mappings>
```

Esta configuración es para un logeo de usuario, donde dependiendo de la autentificación se dejará pasar o no al usuario, teniendo también un caso en el que exista una cuenta de usuario invitado para que pueda accesar con funcionalidad limitada.

Primero definimos un Bean de formulario que llamaremos "LoginForm" que pertenece a la clase formulario.LoginForm, este nombre se usará como nombre de atributo de sesión o solicitud para el Bean de formulario.

Después en la acción utilizamos como página de entrada de datos index.jsp, le asociamos el nombre del Bean de formulario "LoginForm" con el atributo name, el path con el que accederemos a esta acción será "/login", su ámbito será de petición y el tipo de la clase Action será acciones.Enlace y por último los forwards que tendrán las cadenas que se compararán con el resultado del Action al procesar la información, para saber hacia dónde debe de redirigirse el control de la aplicación.

Otra etiqueta que tiene el archivo struts-config.xml es la de <global-forwards> que permite crear mapeos de páginas JSP que serán mandadas llamar con frecuencia dentro de un Action como en el siguiente código:

```
actionMappingInstace.findForward("welcome");
```

Las páginas globales se declaran dentro del struts-config de la siguiente forma:

```
<global-forwards>
        <forward name="welcome"  path="/Welcome.do"/>
</global-forwards>
```

Utilizar dentro de la etiqueta forward el atributo nombre y path permite que dentro de uno o más Actions que requieran esa página JSP podamos referirnos a ella mediante un alias y en el caso que necesitemos cambiar el nombre de esta página no tendremos que modificar cada Action que la mencione, solo hay que modificar el atributo path en la etiqueta forward.

El archivo struts-config.xml también permite a través de la etiqueta <data-sources>, especificar una fuente de datos externa como una base de datos para que acceda a ella nuestra aplicación, como en el siguiente ejemplo:

```
<struts-config>
  <data-sources>
    <data-source
      autoCommit="false"
     description="Example Data Source Description"
     driverClass="org.postgresql.Driver"
maxCount="4"
minCount="2"
password="mypassword"
     url="jdbc:postgresql://localhost/mydatabase"
    user="myusername"/>
  </data-sources>
</struts-config>
```

Descriptor de Despliegue de la Aplicación Web

Para terminar nuestra configuración es necesario agregar en nuestro descriptor de despliegue que está guardado en la carpeta WEB-INF con el nombre de web.xml los componentes de Struts necesarios, a continuación veremos que modificaciones hay que hacerle al archivo de despliegue.

-Configurar el Ejemplar de Action Servlet

Añadimos una entrada definiendo el propio servlet action, junto con los parámetros de inicialización apropiados. Dicha entrada se podría parecer a esto:

```
 <servlet>
   <servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-
class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name >debug</param-name>
            <param-value >2</param-value>
        </init-param>
        <init-param>
            <param-name >detail</param-name>
            <param-value >2</param-value>
        </init-param>
        <load-on-startup >2</load-on-startup>
</servlet>
```

El mínimo de atributos que debemos declarar dentro de la etiqueta <servlet> son:

<servlet-name>.- Específica el nombre con el que identificaremos el servlet, para el ejemplo anterior es: "action".

<servlet-class>.- específica a que clase pertenece el servlet, para el ejemplo anterior es: "org.apache.struts.action.ActionServlet"

Después dentro de la etiqueta <init-param> declaráramos las propiedades que queremos inicializar con la etiqueta <param-name> y su valor con la etiqueta <param-value>. Los valores que puede tomar la etiqueta <param-name> son:

- **config**.- Indica el path donde está guardado el archivo de configuración (/WEB-INF/struts-config.xml).

- **debug** .- Indica el nivel de detalle de depuración para este servlet, que dice cuanta Información tendremos en nuestro archivo log.

- **detail** .- Indica el nivel de depuración que tendrá el Digester, el cual muestra su salida a través del método System.out.

Existen más propiedades para el init-param pero con estas podemos arrancar nuestras aplicaciones.

-Configurar el Mapeo del Servlet Action

El servlet controlador mapeará todas las URLs que tengan un prefijo y extensión predeterminada que describiremos con el siguiente código:

```
   <servlet-mapping>
     <servlet-name>action</servlet-name>
     <url-pattern >/execute/*.do</url-pattern>
   </servlet-mapping>
```

De esta forma tendremos uniformidad en las direcciones que se muestran y podemos ocultar el path verdadero de la página JSP o servlet controlador que estemos utilizando, el prefijo es opcional y la extensión por default es .do de esta forma nuestra URL quedaría:

```
http://www.mycompany.com/myapplication/welcome.do
```

Donde /myapplication es el path de contexto con el que identificamos a nuestra aplicación.

-Configurar la Librería de Etiquetas de Struts

Dentro de nuestro descriptor de despliegue también tenemos que agregar las librerías de struts donde se encuentran nuestras etiquetas que darán funcionalidad a nuestras páginas JSP, actualmente existen 4 librerías estándar: struts-Bean, struts-html, struts-logic y struts-template.

La librería struts-Bean contiene etiquetas que nos permiten el manejo de Beans, como acceder a ellos, a sus propiedades o definir nuevos Beans de clases del API Java, creadas por el programador o incluso con el valor de una Cookie.

La librería struts-html provee etiquetas para crear formularios de entrada y otros componentes útiles para la creación de interfaces de usuario basados en componentes HTML.

La librería struts-logic provee de etiquetas para el manejo de bucles sobre colecciones de objetos para representar su contenido en forma de texto por medio de bucles, también provee de etiquetas para el manejo condicional de salida de texto y el control de flujo dentro de una página JSP.

La librería struts-template provee etiquetas para generar plantillas.

En el siguiente código podemos observar cómo se declaran las librerías en el descriptor de despliegue.

```
<jsp-config>
    <taglib>
         <taglib-uri>/WEB-INF/struts-Bean.tld</taglib-uri>
         <taglib-location>/WEB-INF/struts-Bean.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
        <taglib-location >/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri >/WEB-INF/struts-logic.tld</taglib-uri>
        <taglib-location >/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri >/WEB-INF/struts-nested.tld</taglib-uri>
        <taglib-location >/WEB-INF/struts-nested.tld </taglib-
location>
    </taglib>
    <taglib>
        <taglib-uri >/WEB-INF/struts-tiles.tld</taglib-uri>
        <taglib-location >/WEB-INF/struts-tiles.tld </taglib-location>
```

```
    </taglib>
</jsp-config>
```

Con este código indicamos a nuestras páginas JSP donde pueden encontrar las librerías con las etiquetas que utilizaremos a lo largo de la ejecución de nuestra aplicación.

-Añadir Componentes Struts a nuestra Aplicación

Para poder accesar a las librerías donde se encuentran las etiquetas que vamos a emplear en nuestra aplicación web, necesitamos copiar los archivos con extensión .tld, copiar el archivo struts.jar y todos los archivos commons-*.jar dentro del directorio WEB-INF/lib.

5.- SISTEMA "PRODUCT WAREHOUSE"

En este capítulo nos enfocaremos a resolver la necesidad que tiene la empresa Herrajes Benítez S.A. de C.V. que planteamos en un inicio de este trabajo y consiste en un sistema de almacén al que nombraremos "PRODUCT WAREHOUSE". Para resolver esta necesidad emplearemos las metodologías y las tecnologías que describimos en los capítulos anteriores, utilizándolas para resolver una necesidad real y beneficiar de esta forma a la empresa.

Para elaborar este sistema seguiremos el proceso de desarrollo en cascada, que consiste en las siguientes etapas:

Requisitos.- Descripción del problema por parte del área usuaria para el levantamiento de requerimientos, identificación de actores y roles así como el alcancé del sistema.

Análisis.- Una vez obtenidas los requerimientos, alcance del sistema, actores y roles pasaremos al modelado del sistema mediante el diagrama de casos de uso, para describir el sistema y definir las actividades que se realizarán dentro de él, describiendo cada caso de uso y su flujo dentro del sistema. También desglosaremos cada caso de uso para identificar las clases necesarias para ser representado mediante un diagrama de clases con estereotipos.

Diseño.- Describiremos en función a las clases encontradas, sus atributos y métodos mediante el diagrama de clases completo y mediante los diagramas de secuencia veremos la interacción entre las clases y los actores así como los eventos que son disparador para realizar una actividad. De igual forma haremos nuestro diagrama relacional para la Base de Datos que utilizaremos para guardar de forma permanente la información del sistema.

Desarrollo.- Desarrollaremos nuestro sistema con las herramientas mencionadas en los capítulos anteriores.

Pruebas.- Haremos las pruebas necesarias a nuestro sistema antes de liberarlo para encontrar algún defecto en él y ver que cumple con los requisitos que planteo en un principio el usuario.

Implementación.- Una vez de cerciorarnos que el sistema no tenga defectos y que cumpla con lo requerido por el área usuaria este se liberará para ser instalado en la empresa y comenzar a utilizarlo.

Documentación: El sistema estará documentado mediante los diagramas mencionados en las etapas anteriores los cuales se incluyen en este trabajo.

5.1.- Requisitos del sistema "PRODUCT WAREHOUSE"

Descripción del problema

Actualmente en la empresa Herrajes Benítez S.A. de C.V. no se cuenta con ningún tipo de control de almacén ni de inventario lo que ocasiona pérdidas de material y de productos, otro problema es que no se sabe con certeza cuándo hacer los pedidos a los proveedores, por lo tanto se requiere un sistema que permita tener un control de entradas y salidas donde se registren los productos, su cantidad y quien registra la entrada o la salida del almacén así como en el control de inventario poder saber qué cantidad y qué tipo de productos fueron introducidos o sacados en un cierto lapso de tiempo o en caso de no tener existencia del producto llamar al proveedor para vuelva a surtir.

Por lo tanto se necesita una base de datos donde se guarde la información para su posterior consulta o modificación, el sistema debe de entregar toda esta información en un reporte impreso o guardado en un archivo de texto para que los jefes puedan consultarlo. Se desea que el sistema sea utilizado por varios usuarios ubicados en diferentes partes de la empresa por lo tanto deberá ser accesado en red y se desea que dentro de estos usuarios podamos distinguir entre empleados que registren solo entradas, encargados que puedan registrar entradas y salidas, permitiéndoles a los jefes que puedan tener acceso total al sistema. Se desea que el sistema pueda correr en el equipo con el que se cuenta actualmente, que consiste en 5 PC´s con procesador Pentium D a 3.2 Ghz, 1 GB de RAM y 120GB de DD.

Módulos Identificados:

A.- Administración de usuarios
B.- Administración de productos
C.- Inventarios
D.- Registro de Entradas y Salidas

Requerimientos Funcionales

Módulo: Administración de usuarios.

No. de requerimiento:	A001					
Nombre:	Poder administrar los usuarios del sistema según el perfil del usuario logeado					
Descripción:	Permitir que el usuario con los permisos necesarios puedan crear, modificar, eliminar y consultar el catalogo de usuarios.					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja
Documento anexo:						
Comentarios:						

Módulo: Administración de productos.

No. de requerimiento:	B001					
Nombre:	Poder administrar los productos del sistema según el perfil del usuario logeado					
Descripción:	Permitir que el usuario con los permisos necesarios pueda crear, modificar, eliminar y consultar el catalogo de productos.					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja

Documento anexo:	
Comentarios:	

Módulo: Inventarios.

No. de requerimiento:	C001					
Nombre:	Permitir generar inventario					
Descripción:	Según el perfil del usuario logeado permitir generar el inventario de los productos ya sea por semana, quincena, mes, semestre o año.					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja
Documento anexo:						
Comentarios:						

No. de requerimiento:	C002					
Nombre:	Poder guardar e imprimir el inventario					
Descripción:	Una vez generado el inventario, permitir al usuario guardarlo en un archivo o imprimirlo					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja
Documento anexo:						
Comentarios:						

Módulo: Registro de Entradas y Salidas.

No. de requerimiento:	D001					
Nombre:	Permitir registrar entradas					
Descripción:	El usuario podrá registrar la entrada de productos al almacén					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja
Documento anexo:						
Comentarios:						

No. de requerimiento:	D002					
Nombre:	Permitir registrar salidas					
Descripción:	El usuario pondrá registrar salida de productos del almacén					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja
Documento anexo:						
Comentarios:						

No. de requerimiento:	D003

Nombre:	Registrar movimientos de entras y salidas					
Descripción:	El sistema debe de registrar cada movimiento que se haga en el almacén ya sea una entrada o salida, describiendo el producto, la cantidad la fecha y nombre del usuario que realizó el movimiento					
Prioridad:	X	Alta		Media		Baja
Tipo de requerimiento:	X	Nuevo		Cambio		Defecto
Estabilidad:	X	Alta		Media		Baja
Documento anexo:						
Comentarios:						

Requerimientos no funcionales

Requerimientos de performance

El sistema debe ser realizado de manera que garantice la estabilidad e Integridad de la información capturada por el personal de la empresa de herrajes Benítez S.A. de C.V. la cual se guardará en una base de datos ubicada en la misma empresa.

Requerimientos de usabilidad

La interfaz del usuario debe ser fácil de utilizar y amigable, tanto como sea posible, para que en un corto tiempo de capacitación, un usuario pueda cumplir sin impedimentos las funciones que se le han asignado dentro del sistema "PRODUCT WAREHOUSE", para ello se deben considerar aspectos como espacio en pantalla, tiempo de respuesta, diseño y usabilidad, para darle al usuario una mejor perspectiva del sistema.

Requerimientos de configuración y tecnológicos

El sistema será desarrollado en un ambiente Web para tener acceso a él desde cualquier lugar vía Intranet o Internet.

Para operar el Sistema "PRODUCT WAREHOUSE ", se requiere un equipo de cómputo con las siguientes características:

.

Software

- Navegador Internet Explorer 6.0 o superior, Mozilla Firefox 1.5 o superior.
- Acrobat Reader versión 5.0 ó superior.
- Windows XP con SP2.

Hardware

- Procesador Pentium III ó superior.
- 512 Mb de memoria RAM (mínima recomendada).
- Conexión a Internet.
- Impresora a color.

Requerimientos de seguridad

Se designarán permisos de acuerdo a los siguientes tipos de usuarios:

Súper Usuario (SU):
Los usuarios que pertenezcan a este grupo podrán visualizar y administrar toda la información del sistema y crear todo tipo de usuarios.

Administrador (Adm):

Los usuarios que pertenezcan a este grupo podrán visualizar y administrar la información correspondiente a su perfil y crear Usuarios Normales.

Usuario Normal (UN):
Los usuarios que pertenezcan a este grupo podrán visualizar e interactuar con el sistema de forma limitada y no podrán crear usuarios.

USUARIOS "PRODUCT WAREHOUSE"	
Tipo de Usuario	**Permisos en el modulo:**
Súper Usuario	**Administración de usuarios:** - Altas - Bajas - Consultas - Modificaciones **Administración de productos** - Altas - Bajas - Consultas - Modificaciones **Inventarios** - Generar - Ver registros E/S - Imprimir - Guardar a archivo **Registro de Entradas y Salidas** - Registrar Entrada - Registrar Salida
Administrador	**Administración de usuarios:** - Altas (solo Usuarios Normales) - Bajas (solo Usuarios Normales) - Consultas - Modificaciones (solo Usuarios Normales) **Administración de productos** - Altas - Bajas - Consultas - Modificaciones **Inventarios** - Generar - Ver registros E/S - Imprimir - Guardar a archivo **Registro de Entradas y Salidas** - Registrar Entrada - Registrar Salida
Usuario Normal	**Registro de Entradas y Salidas** - Registrar Entrada

	- Registrar Salida

Identificación de actores y roles

Worker	Tipo de usuario
Dueño de la empresa	Súper usuario
Gerente de la empresa	Administrador
Empleados	Usuario Normal

5.2.- Análisis del sistema "PRODUCT WAREHOUSE"

Diagrama de límite del sistema

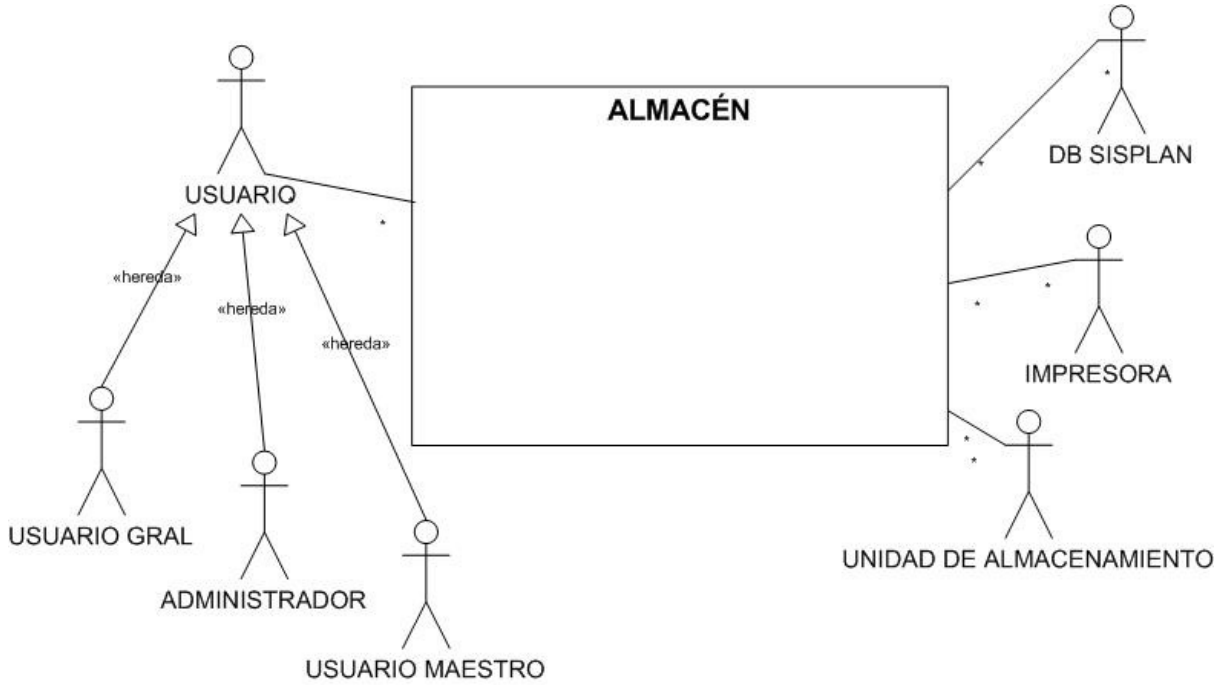
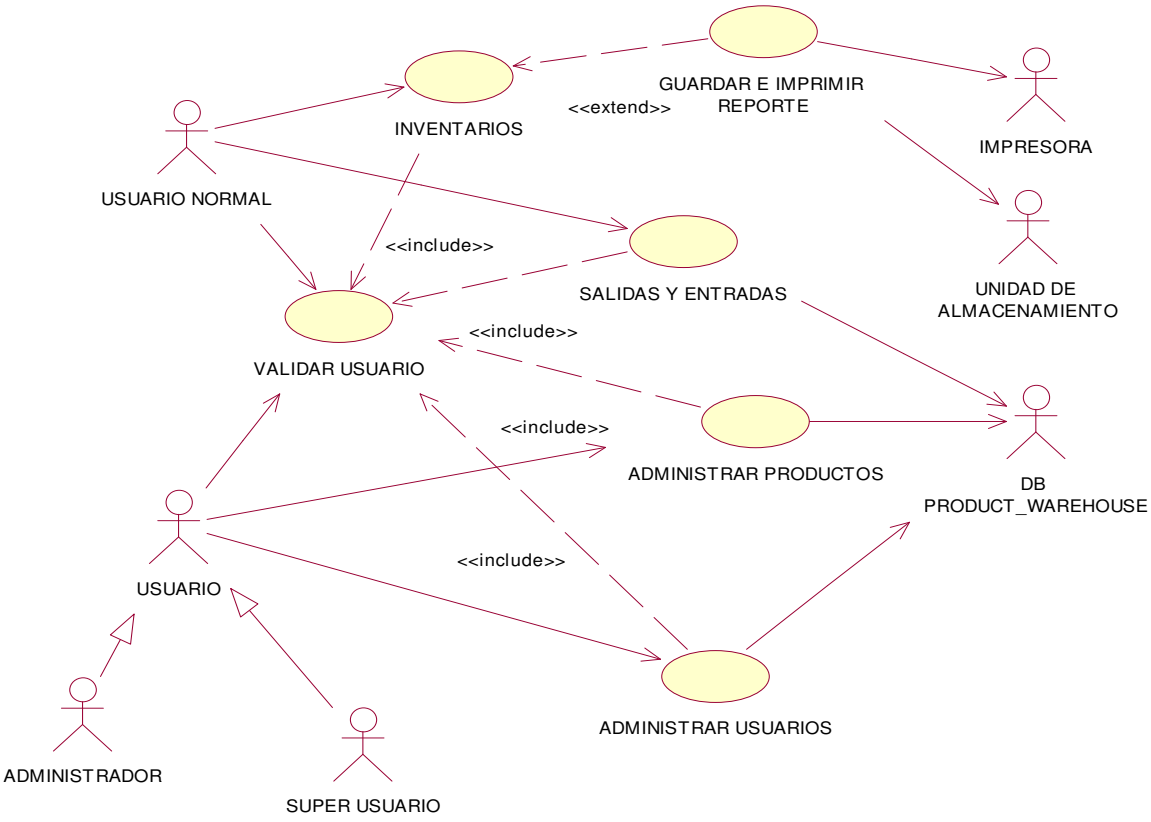


Diagrama de casos de uso



Documentación de casos de Uso

Caso de Uso:	VALIDAR USUARIO
Actores:	USUARIO NORMAL, SÚPER USUARIO Y ADMINISTRADOR
Propósito:	Ingresar al sistema con el rol correspondiente.
Resumen:	Este caso de uso servirá para tener control en las actividades que realicé el usuario y asignar los permisos correspondientes para que estos interactúen con el sistema.
Precondiciones:	ninguna
Flujo Principal:	El usuario podrá insertar su login y password pudiendo pasar a (E-1). - Ingresará a la pantalla de bienvenida que contiene el menú principal. - Elegirá la tarea que desee realizar en el sistema.
Subflujos:	ninguno
Excepciones:	(E-1).- Login y/o password incorrectos. En caso de que el login y/o el password que el usuario ha introducido no se encuentre en la BD.

Caso de Uso:	ADMINISTRAR USUARIOS
Actores:	SÚPER USUARIO Y ADMINISTRADOR
Propósito:	Poder administrar el catalogo que contendrá la información de los usuarios que interactuarán con el sistema.
Resumen:	Este caso de uso podrá dar de alta, baja o modificación a un usuario y podrá hacer búsquedas y consultas con los parámetros indicados.
Precondiciones:	Haber sido logeado en el caso de uso "VALIDAR USUARIO" como SÚPER USUARIO O ADMINISTRADOR.
Flujo Principal:	El usuario podrá insertar un nuevo registro llenando los campos indicados pudiendo pasar a (E-1). - Inserta el nuevo registro y pasa a una apantalla de éxito en la operación - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación. El usuario podrá buscar usuarios en la BD de acuerdo a los parámetros dados pudiendo pasar a (S-1) ó (E-2).
Subflujos:	(S-1).- Pasa a un pantalla con los resultados de la búsqueda, donde al seleccionar un registro se abre otra pantalla donde se puede pasar a (S-2) ó (S-3). (S-2).- Eliminar el registro, donde se podrá eliminar un usuario pudiendo pasar a (E-3). - Se muestra una pantalla donde se indica que se tuvo éxito en la operación. - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación. (S-3).- Modificar Registro, donde se pueden modificar los datos del registro para ser guardados posteriormente, se puede pasar a (E-1). - Se muestra una pantalla donde se indica que se tuvo éxito en la operación. - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación.
Excepciones:	(E-1).- Datos incorrectos o faltantes. En caso de que el usuario no llene los campos requeridos o que estos no tengan el formato correcto. (E-2).- Demasiados registros ó ningún registro. En caso de que existan más de 100 registros con esos parámetros ó en caso de que no exista ningún registro que coincida con los parámetros dados. (E-3).- No es posible eliminar registro. En caso de que este registro tenga alguna relación con otros datos del sistema de acuerdo a la BD.

Caso de Uso:	ADMINISTRAR PRODUCTOS
Actores:	SÚPER USUARIO Y ADMINISTRADOR
Propósito:	Poder administrar el catalogo que contendrá la información de los productos que interactuarán con el sistema.
Resumen:	Este caso de uso podrá dar de alta, baja o modificación a un producto y podrá hacer búsquedas y consultas con los parámetros indicados.
Precondiciones:	Haber sido logeado en el caso de uso "VALIDAR USUARIO" como SÚPER USUARIO O ADMINISTRADOR.

Flujo Principal:	El usuario podrá insertar un nuevo registro llenando los campos indicados pudiendo pasar a (E-1). - Inserta el nuevo registro y pasa a un apantalla de éxito en la operación - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación. El usuario podrá buscar usuarios en la DB de acuerdo a los parámetros dados pudiendo pasar a (S-1) ó (E-2).
Subflujos:	(S-1).- Pasa a una pantalla con los resultados de la búsqueda, donde al seleccionar un registro se abre otra pantalla donde se puede pasar a (S-2) ó (S-3). (S-2).- Eliminar el registro, donde se podrá eliminar el usuario pudiendo pasar a (E-3). - Se muestra una pantalla donde se indica que se tuvo éxito en la operación. - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación. (S-3).- Modificar Registro, donde se pueden modificar los datos del registro para ser guardados posteriormente, se puede pasar a (E-1). - Se muestra una pantalla donde se indica que se tuvo éxito en la operación. - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación.
Excepciones:	(E-1).- Datos incorrectos o faltantes. En caso de que el usuario no llene los campos requeridos o que estos no tengan el formato correcto. (E-2).- Demasiados registros ó ningún registro. En caso de que existan más de 200 registros con esos parámetros ó en caso de que no exista ningún registro que coincida con los parámetros dados. (E-3).- No es posible eliminar registro. En caso de que este registro tenga alguna relación con otros datos del sistema de acuerdo a la DB.

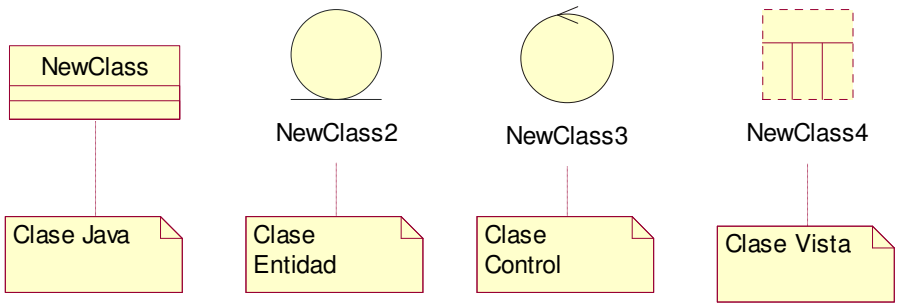
Caso de Uso:	SALIDAS Y ENTRADAS
Actores:	USUARIO NORMAL, SÚPER USUARIO Y ADMINISTRADOR
Propósito:	Poder introducir al sistema los datos de una entrada o salida de producto según el perfil logeado.
Resumen:	En este caso de uso se podrá ingresar una entrada o salida de producto en el almacén, donde se registrarán todos los datos del producto así como la fecha, hora y el usuario que hizo este movimiento.
Precondiciones:	Haber sido logeado en el caso de uso "VALIDAR USUARIO" como USUARIO NORMAL, SÚPER USUARIO O ADMINISTRADOR.
Flujo Principal:	El usuario podrá llenar un formulario donde ya se tienen registrados todos los productos existentes, se indicará que tipo de operación de trata (entrada o salida) y la cantidad de producto, donde se puede pasar a (E-1), (E-2) ó (E-3). - Se muestra una pantalla donde se indica que se tuvo éxito en la operación. - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación.

Subflujos:	ninguno
Excepciones:	(E-1).- Datos incorrectos o faltantes. En caso de que el usuario no llene los campos requeridos o que estos no tengan el formato correcto. (E-2).- No hay suficiente producto. En caso de que se desee hacer una salida de producto y que en la BD no exista suficiente cantidad para realizar la operación. (E-3).- No esta autorizado para realizar esta operación. En caso de que se desee hacer una salida y el perfil logeado sea de usuario normal.

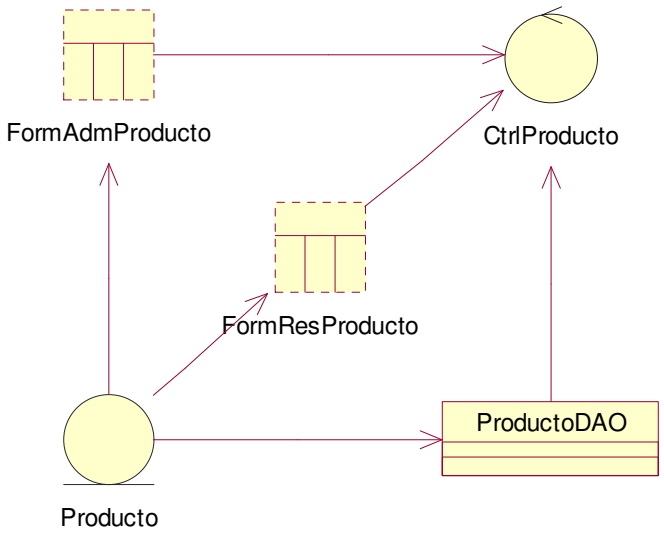
Caso de Uso:	INVENTARIOS
Actores:	USUARIO NORMAL, SÚPER USUARIO Y ADMINISTRADOR
Propósito:	Poder generar el inventario del almacén.
Resumen:	En este caso de uso se generarán los inventarios del almacén donde se podrán obtener todo los movimientos hechos en un lapso de tiempo (semana, quincena, mes año).
Precondiciones:	Haber sido logeado en el caso de uso "VALIDAR USUARIO" como USUARIO NORMAL, SÚPER USUARIO O ADMINISTRADOR.
Flujo Principal:	El usuario podrá llenar un formulario donde se especificarán los parámetros para generar el inventario, donde se puede pasar a (E-1). - Se mostrará el reporte con el inventario generado para que sea consultado por el usuario donde se puede pasar a (S-1). - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación.
Subflujos:	(S-1).- Se pasa al caso de uso "GUARDAR E IMPRIMIR"
Excepciones:	(E-1).- Datos incorrectos o faltantes. En caso de que el usuario no llene los campos requeridos o que estos no tengan el formato correcto.

Caso de Uso:	GUARDAR E IMPRIMIR
Actores:	USUARIO NORMAL, SÚPER USUARIO Y ADMINISTRADOR
Propósito:	Poder guardar y/o imprimir los inventarios generados.
Resumen:	En este caso de uso se podrán imprimir y/o guardar los inventarios generados en el caso de uso "INVENTARIOS" en una unidad de almacenamiento.
Precondiciones:	Haber seleccionado la opción de guardar y/o imprimir inventario del caso de uso "INVENTARIOS".
Flujo Principal:	El usuario podrá seleccionar que acción desea realizar (IMPRIMIR, GUARDAR o AMBOS) donde puede pasar a (S-1), (S-2) o (S-3). - Muestra la opción de regresar a la pantalla anterior ó muestra el menú para hacer otra operación.
Subflujos:	(S-1).- Se muestra la vista preliminar del inventarío generado para su impresión. (S-2).- Se selecciona la ruta donde se desea guardar el archivo de inventarios. (S-3).- Se selecciona la ruta donde se desea guardar el archivo y después se muestra la vista preliminar para impresión del inventario.
Excepciones:	ninguna

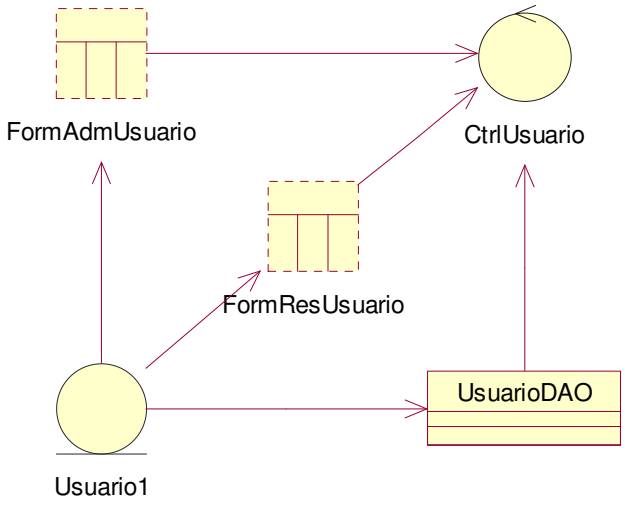
Diagramas de clases con estereotipos



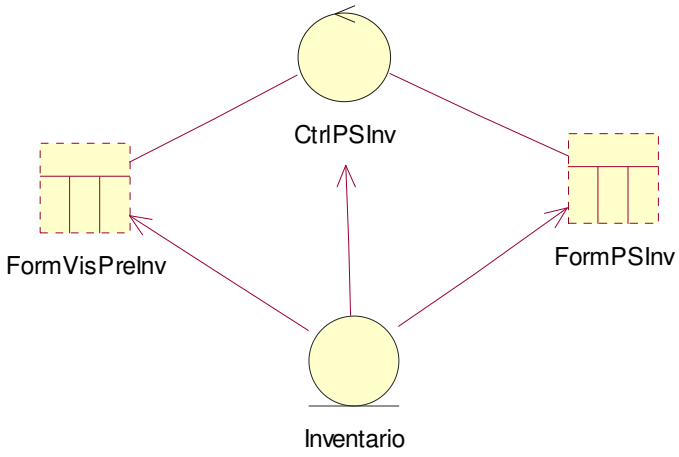
Administrar productos



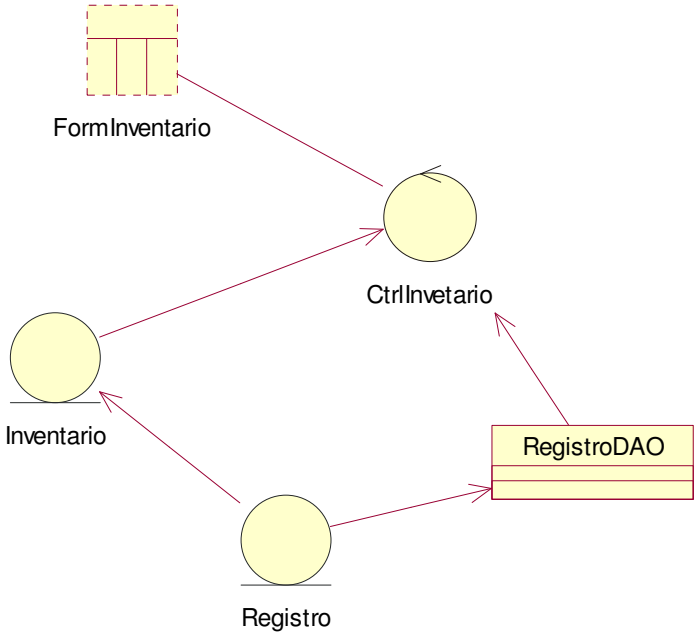
Administrar usuarios



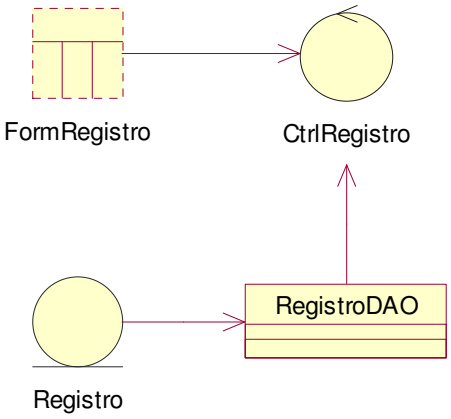
Guardar e imprimir reporte



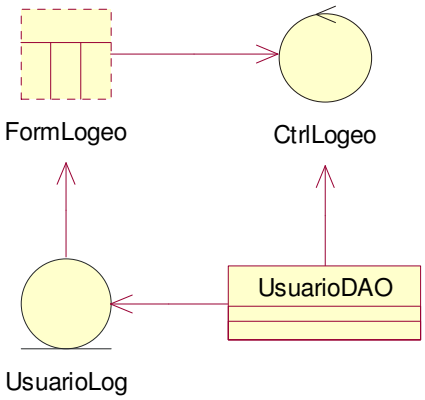
Inventarios



Salidas y entradas



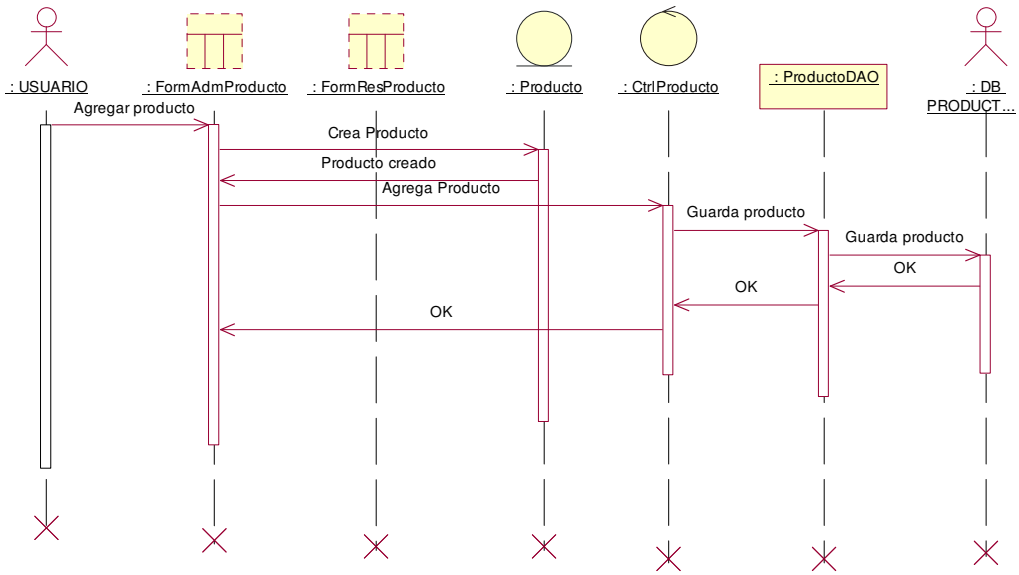
Validar usuario

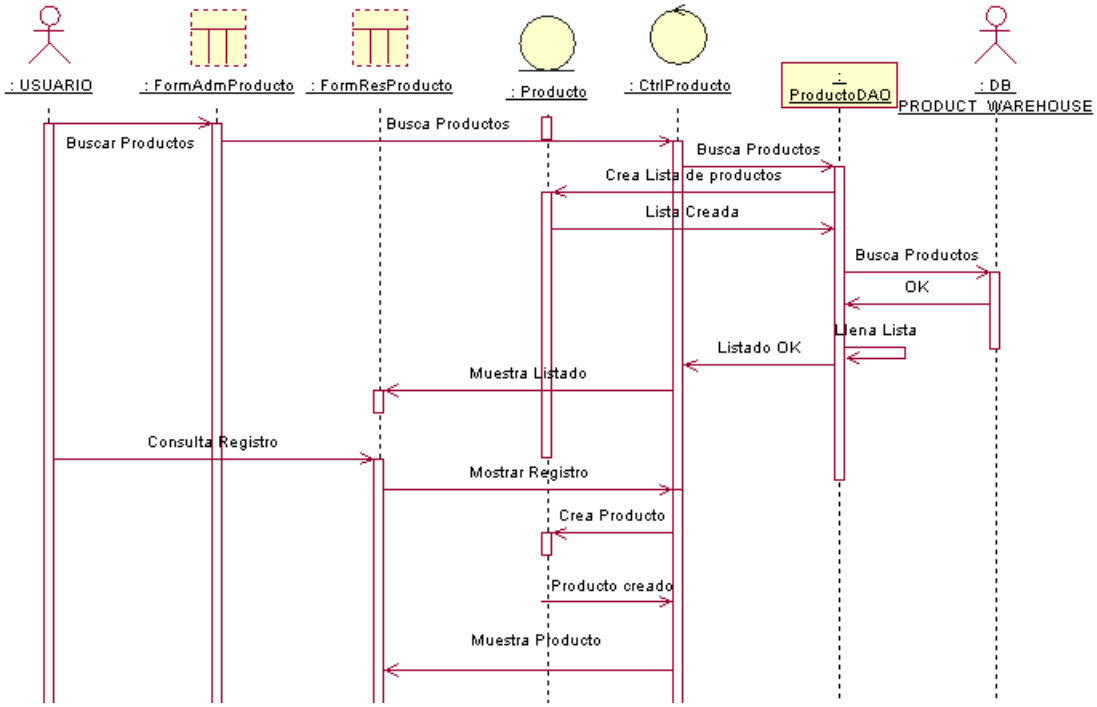


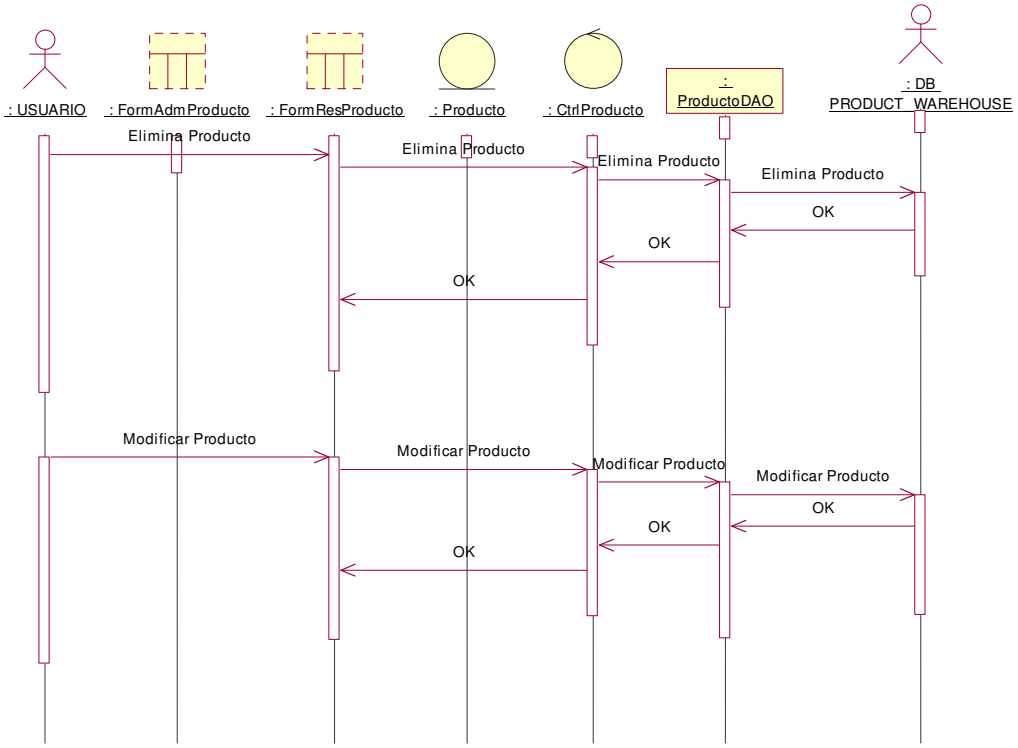
5.3.- Diseño del sistema "PRODUCT WAREHOUSE"

Diagramas de secuencia

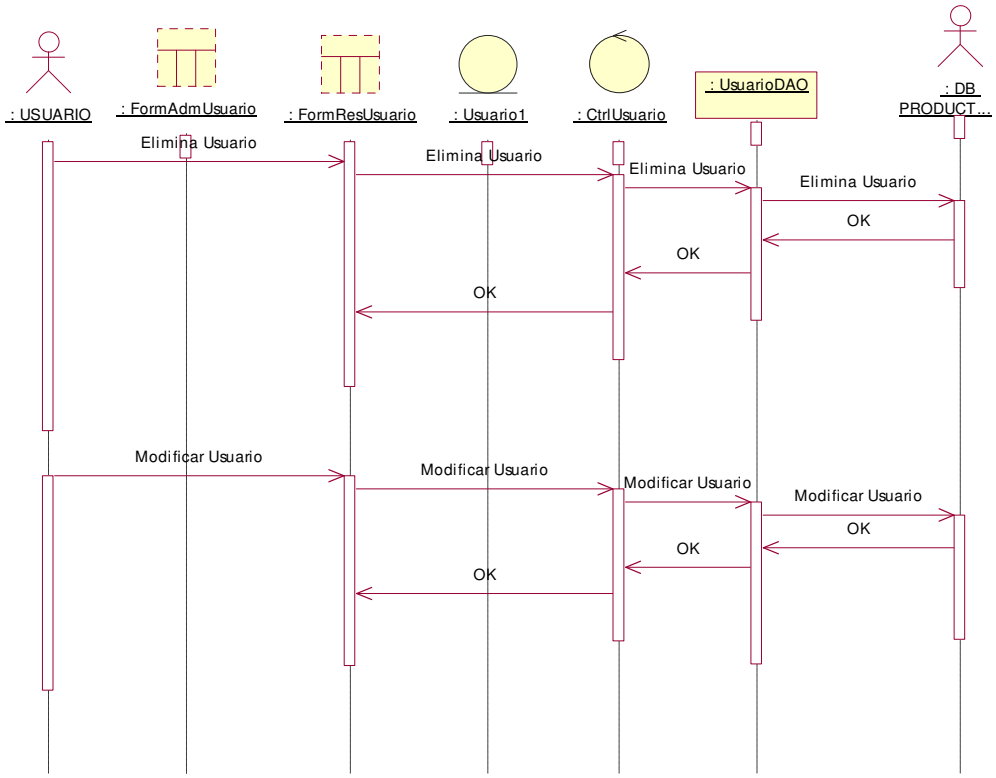
Administrar productos

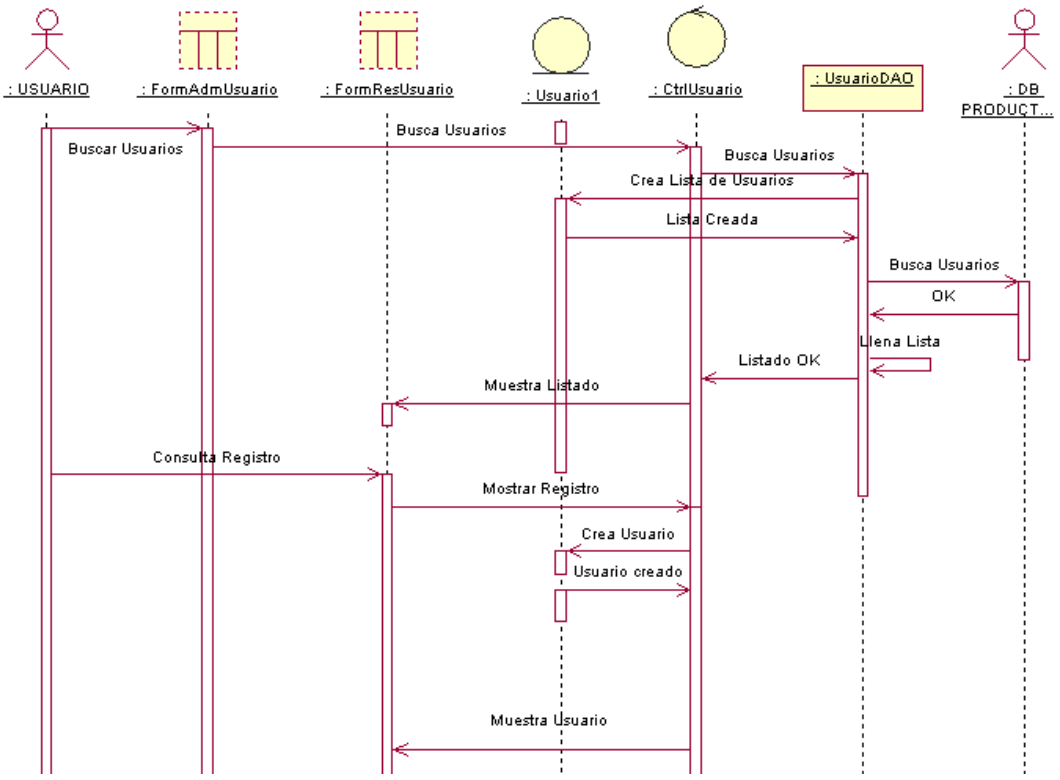


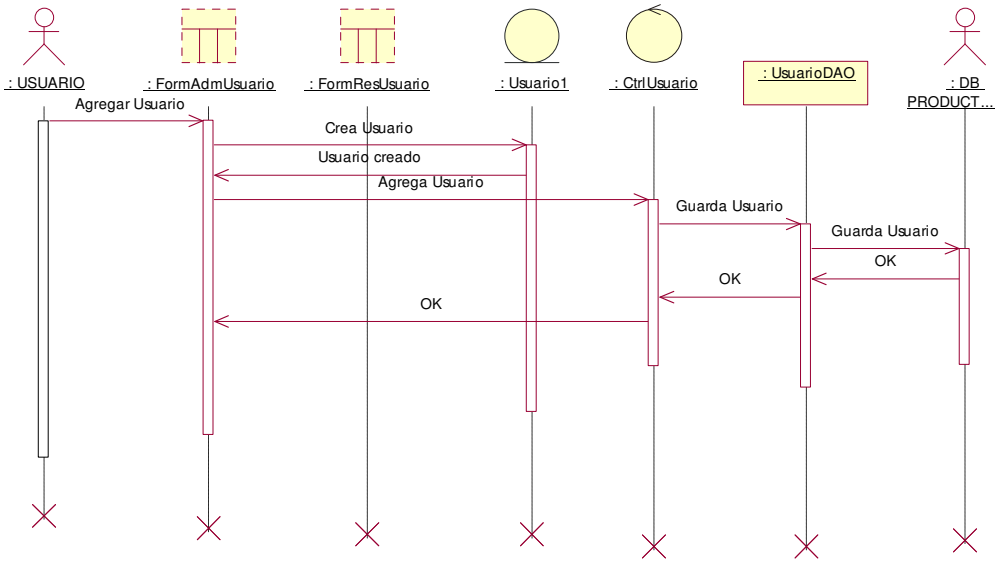




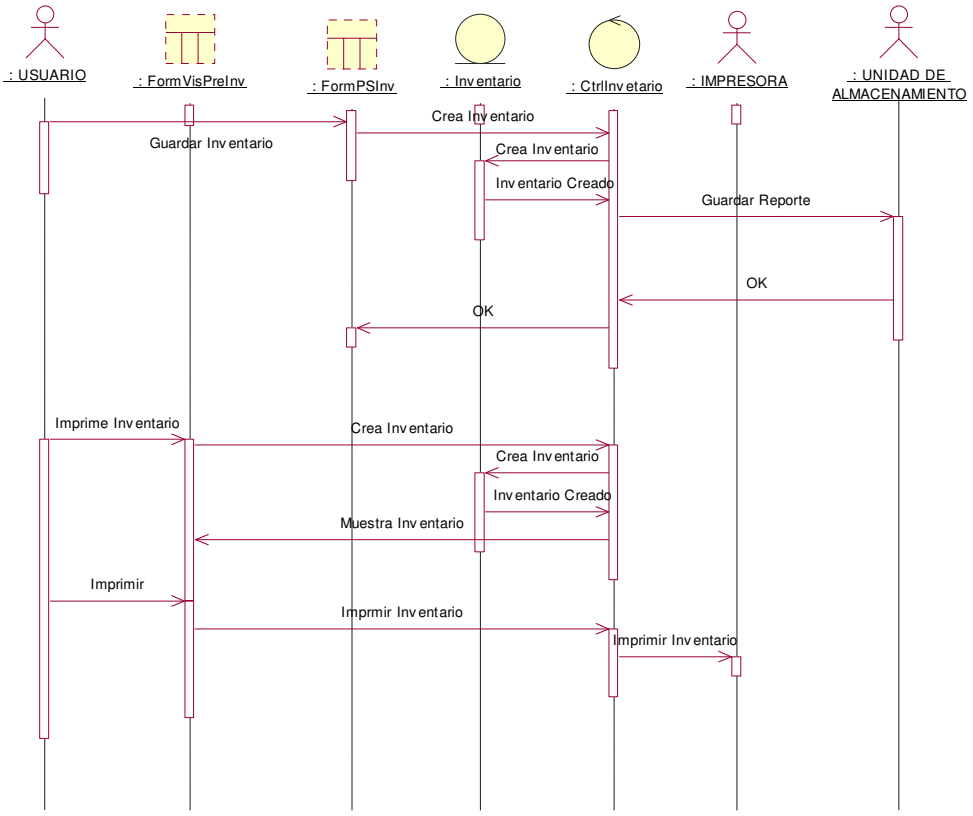
Administrar usuarios



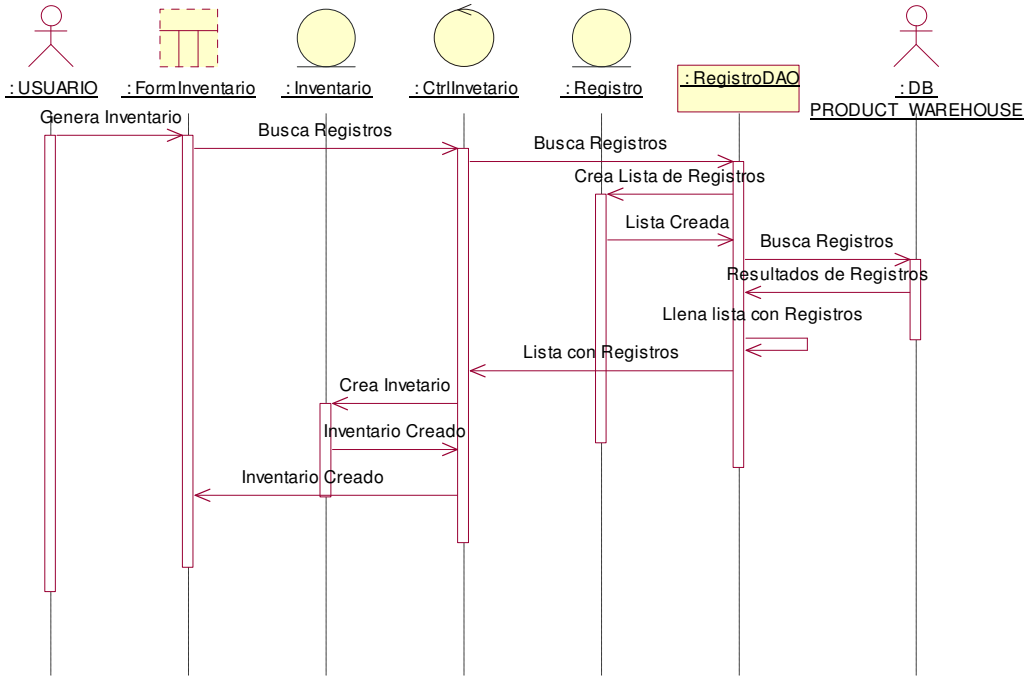

: USUARIO
: FormAdmUsuario
: FormResUsuario
: Usuario1
: CtrlUsuario
: UsuarioDAO
: DB PRODUCT...
Buscar Usuarios
Busca Usuarios
Busca Usuarios
Crea Lista de Usuarios
Lista Creada
Busca Usuarios
OK
Llena Lista
Listado OK
Muestra Listado
Consulta Registro
Mostrar Registro
Crea Usuario
Usuario creado
Muestra Usuario


: USUARIO
: FormAdmUsuario
: FormResUsuario
: Usuario1
: CtrlUsuario
: UsuarioDAO
: DB PRODUCT...
Agregar Usuario
Crea Usuario
Usuario creado
Agrega Usuario
Guarda Usuario
Guarda Usuario
OK
OK
OK

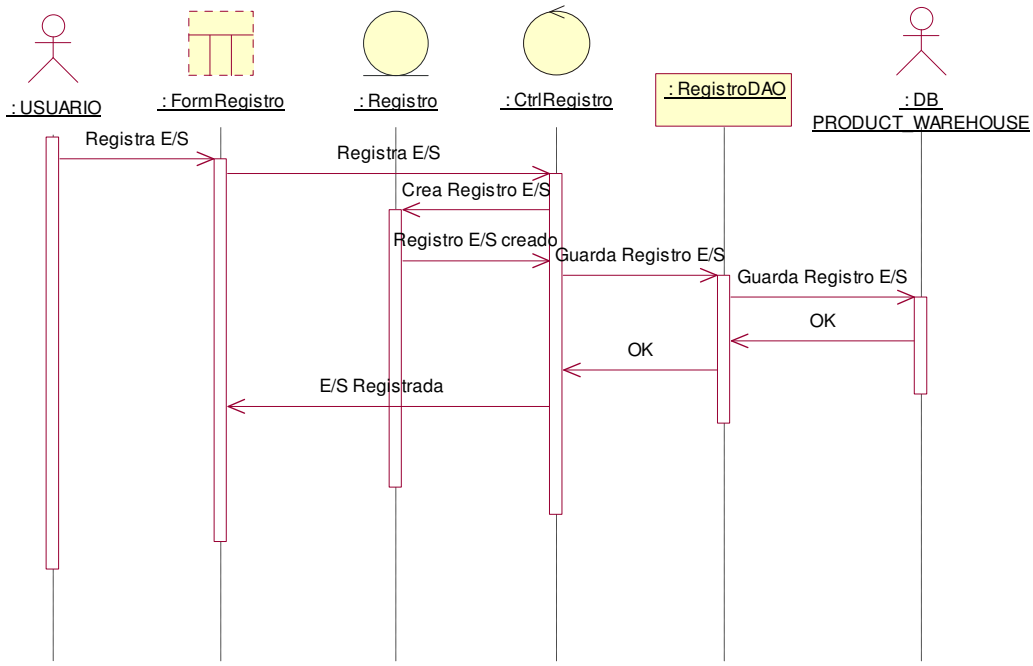
Guardar e imprimir reporte



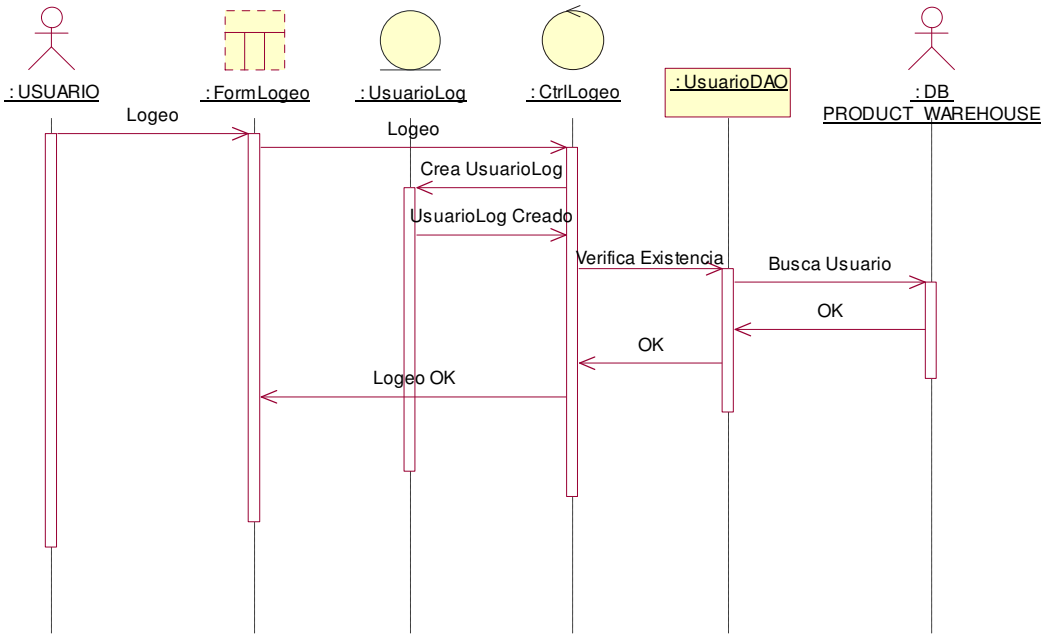
Inventarios



Salidas y entradas

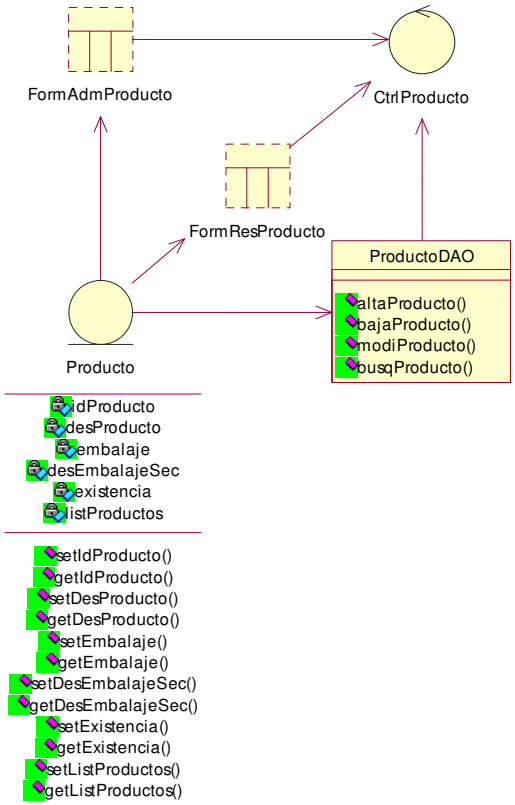


Validar usuario

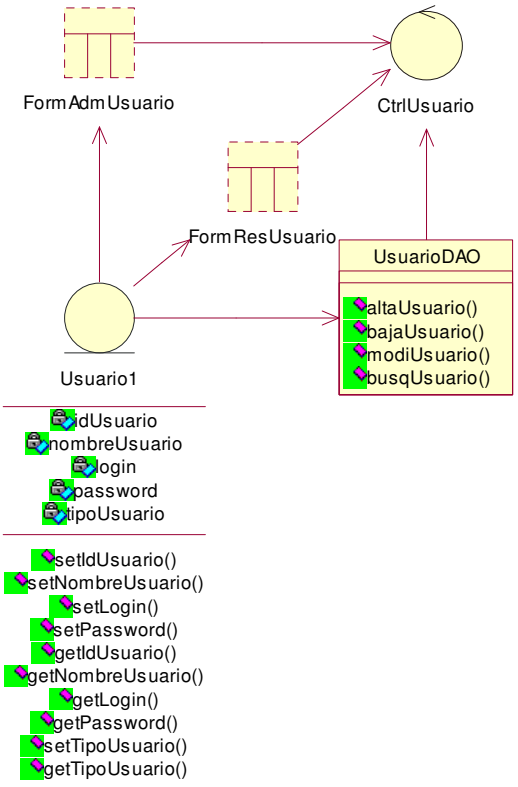


Diagramas de clase

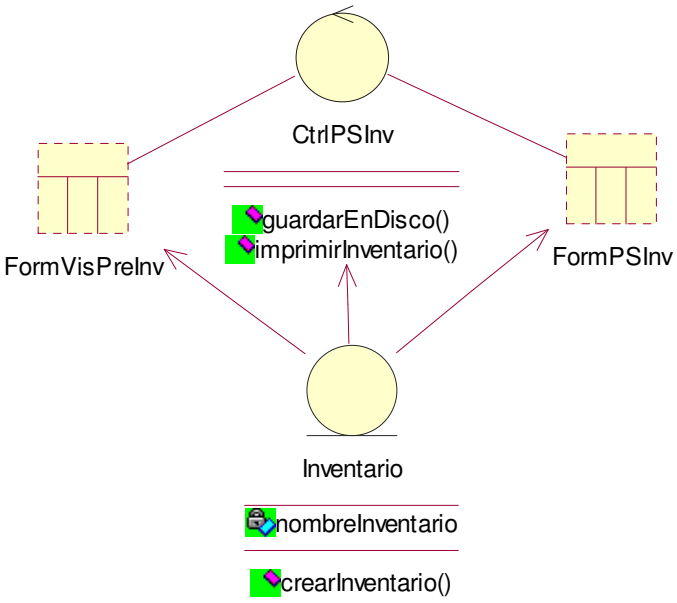
Administrar productos



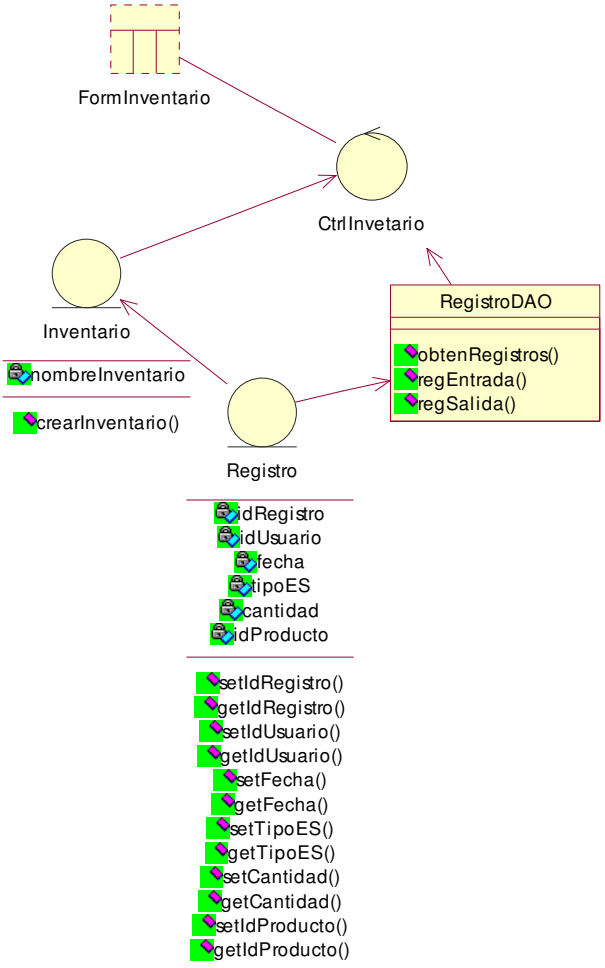
Administrar usuarios



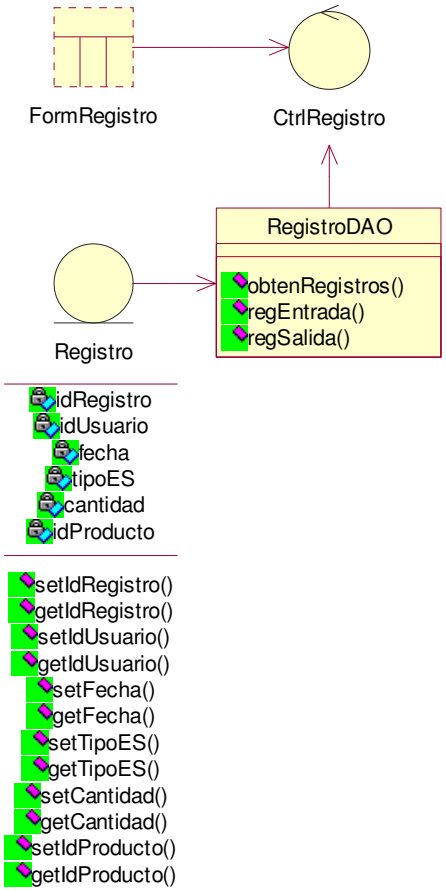
Guardar e Imprimir Inventario



Inventarios



Salidas y Entradas



Validar usuario

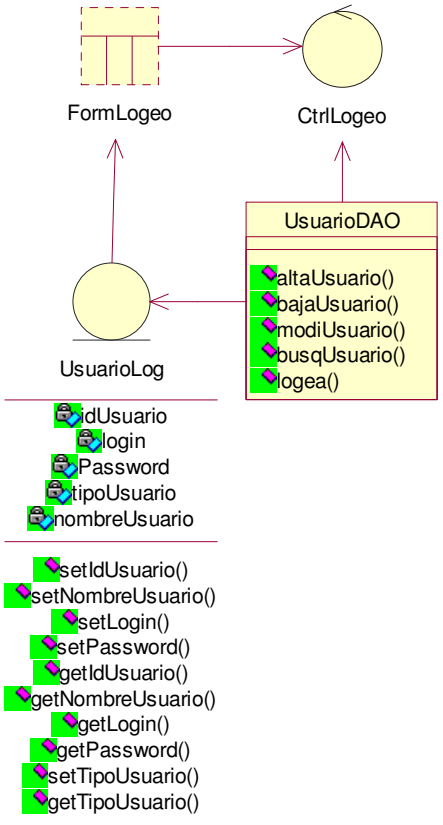
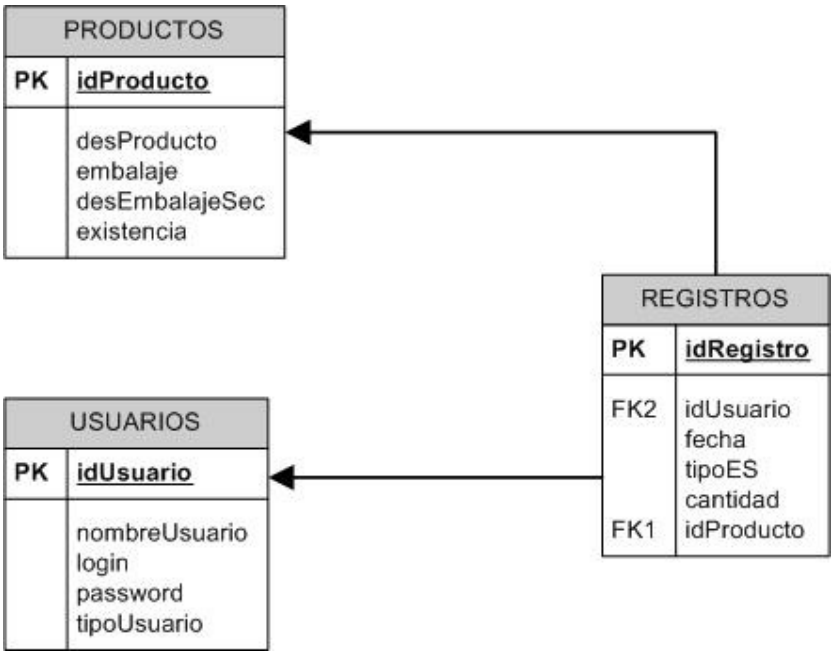


Diagrama relacional para la base de datos "PRODUCT WAREHOUSE"



CONCLUSIONES

Después de haber Desarrollado e Implementado el sistema "PRODUCT WAREHOUSE" para satisfacer la necesidad de la empresa Herrajes Benítez S.A. de C.V. se llegó a la conclusión de que a través de las herramientas computacionales actuales podemos automatizar los diferentes procesos de una PyME, para que esta pueda administrar sus tiempos y así obtener una mayor productividad. Usando de forma específica las tecnologías JSP, Servlet, STRUTS, MySQL y TOMCAT, podemos crear aplicaciones de calidad empresarial a un bajo costo en cuestión de licencias de software, ya que todas estas tecnologías son software libre. Bajar los costos permite que las PYMES puedan adquirir sistemas informáticos y lograr una mejor competitividad frente a sus rivales.

La automatización de procesos es hoy en día una necesidad para no quedar rezagado en el mundo empresarial y con la ayuda de los sistemas informáticos podemos conseguir esto, todas la PYMES que quieran crecer y convertirse en grandes empresas deben contar con estas herramientas.

El realizar el sistema "PRODUCT WAREHOUSE" es un ejemplo de como a través de aplicaciones web con software libre podemos satisfacer distintas necesidades para las PyMES, como pueden ser un almacén con inventario (como en este caso) o un sistema para el manejo de una abarrotera, para el control de clientes en un gimnasio, el control de medicamentos en una farmacia o un catalogo de libros para una biblioteca etc., la versatilidad que tienen este tipo de aplicaciones y el bajo costo en las herramientas de desarrollo, permite hacer aplicaciones a la medida de cada empresa.

El lenguaje de programación JAVA tiene varias ventajas respecto a otros lenguajes de programación las cuales ya fueron mencionados en los capítulos expuestos anteriormente siendo algunas de las más importantes, la orientación objetos y la portabilidad. Estas características del lenguaje de programación JAVA lo convierten en la mejor opción para realizar aplicaciones de todo tipo.

La tecnología Servlet que ha sido ampliamente usada en aplicaciones web, nos ha permitido integrar contenido dinámico a nuestra aplicación, dándole más funcionalidad y parecido a una aplicación stand-alone.[1]

[1] Independiente. Separado de un conjunto. Autónomo. Caracteriza a un terminal o un puesto de trabajo que está conectado directamente al ordenador o a un concentrador; es decir, que tiene una sola conexión a través de una línea externa.

Las JSP (JAVA SERVER PAGES) que son la evolución de los servlets permiten editar de una forma más rápida y estructurada el contenido web dinámico. Debido a sus prestaciones y rapidez de desarrollo se ha utilizando esta tecnología para la vista de nuestro proyecto con los resultados deseados.

Por otra parte la utilización STRUTS como framework de desarrollo, nos permitió enfocarnos en las fases iníciales y a veces las más importantes de un proyecto como son el análisis y el diseño, esto debido a la facilidad de programación que ofrece STRUTS.

Otras herramientas indispensables para el desarrollo de este proyecto fueron MySQL y TOMCAT, siendo MySQL un DBMS muy poderoso y de libre uso que ofrece una gran cantidad de herramientas útiles para el manejo de datos. Por otro lado TOMCAT nos permitió administrar nuestras aplicaciones web y desplegarlas para que estén disponibles a través de un navegador web.

El conjunto de todas estas herramientas (JAVA, JSP, SERVLETS, STRUTS, MySQL y TOMCAT) permiten desarrollar de una forma eficiente y rápida una aplicación web, en especial el Framework STRUTS que tiene una gran cantidad de librerías con diversos componentes que permiten realizar la labor de programación de una forma más rápida.

BIBLIOGRAFÍA

UML (UNIFIED MODELING LANGUAJE)

UML PARA PROGRAMADORES JAVA
Martín, Robert C.
Pearson Educación, 2004

EL LENGUAJE UNIFICADO DE MODELADO
Addison Wesley
Grady Booch, James Rumbaugh, Ivar Jacobson, 1999

EL LENGUAJE UNIFICADO DE MODELADO. MANUAL DE REFERENCIA
Addison Wesley
James Rumbaugh, Ivar Jacobson, Grady Booch, 2000

EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE
Addison Wesley
Ivar Jacobson, Grady Booch, James Rumbaugh, 2000

MySQL

MySQL
Gutiérrez Gallardo, Juan Diego
Anaya Multimedia-Anaya Interactiva, 2004

MySQL AVANZADO
Zawodny, Jeremy D.
Anaya Multimedia-Anaya Interactiva, 2004

MySQL PARA WINDOWS Y LINUX
Pérez, César
Ra-Ma, Librería y Editorial Microinformática, 2004

JAVA

INGENIERÍA DE SOFTWARE ORIENTADA A OBJETOS TEORÍA Y PRÁCTICA CON UML Y JAVA
Dr. Alfredo Weitzenfeld
Departamento Académico de Computación
División Académica de Ingeniería ITAM, 2001

JAVA 2, MANUAL DE USUARIO Y TUTORIAL 3ª Ed. AMPLIADA Y ACTUALIZADA A LA VERSIÓN J2SE
Froufe, Agustín
Alfaomega-Rama

THINKING IN JAVA
Bruce Eckel
3ra edición, Prentice Hall, 2002

COMO PROGRAMAR EN JAVA , 5ED
Harvey M. Deitel.

PROBLEMAS RESUELTOS DE PROGRAMACIÓN EN LENGUAJE JAVA
Jesús Carretero Pérez, Félix García Carballeira, José Manuel Pérez Lobato, José María Pérez Menor.

JAVA 2. CURSO DE PROGRAMACIÓN, 3ª EDICIÓN.
Ceballos, F.J.

JAVA SE 6
Moldes, F. Javier.

J2EE. MANUAL DE REFERENCIA
Keogh J.

JAVA 2 INICIACIÓN Y REFERENCIA
Jesús Sánchez Allende.

TUTORIAL OFICIAL DE JAVA

http://java.sun.com/docs/books/tutorial/

SERVLET Y JSP

JAVA SERVLET AND JSP COOKBOOK
O'reilly & Associates

JSP
Urbaneja Fan, Javier.

DESARROLLO WEB CON JSP
Jayson Falkner, Ben Galbraith, Romin Irani.

BEGINNING JAVASERVER PAGES
Chopra Vivek, Eaves Jon, Jones Rupert, Li Sing, Bell John T.

JSP
EJEMPLOS PRÁCTICOS
Andrew Patzer.

SERVLETS Y JAVA SERVER PAGES: GUÍA PRACTICA
Hall, Marty

CORE SERVLETS AND JAVA SERVER PAGES (JSP)
Brown Larry, Hall Marty,
Prentice Hall

SERVLETS Y JAVASERVER PAGES GUÍA PRACTICA
Hall Marty
Prentice-Hall

PROFESSIONAL JAVA XML PROGRAMMING WITH SERVLETS AND JSP
Alexander Nakhimovsky, Tom Myers

CORE SERVLETS UND JAVA SERVER PAGES.
Larry Brown

STRUTS

JAKARTA STRUTS
O´Reilly
1ra edición, Anaya-Multimedia, 2005

APACHE STRUTS (PROGRAMACION)
DORAY, ARNOLD
1ª Edición, Anaya Multimedia-Anaya Interactiva, 2007

PÁGINA OFICIAL DE APACHE STRUTS

http://struts.apache.org/