UNIVERSIDAD NACIONAL AUTONÓMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
"ARAGÓN"

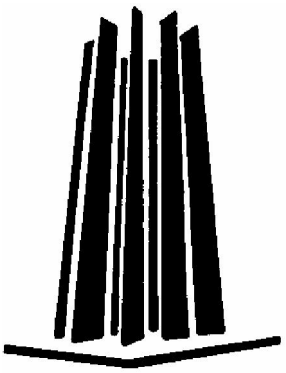
"PROYECTO DE MIGRACIÓN DEL SITIO DE LA RED INALÁMBRICA UNIVERSITARIA (RIU) EN TECNOLOGÍA BASADA EN JAVA"

BAJO LA MODALIDAD DE
SEMINARIOS Y CURSOS DE ACTUALIZACION Y CAPACITACION PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

PRESENTA:

ÁNGEL ANDRÉS SÁNCHEZ GUZMÁN



ASESOR:
ING. SILVIA VEGA MUYTOY

MÉXICO D.F. 2009

Este trabajo es la consecución de una meta en mi vida, más no significa que sea el final del camino. Me da una inmensa alegría poder compartirlo con la gente que más quiero, aprecio y respeto…

Mi gente

GRACIAS

A Dios,
porque sé que nunca me ha descuidado,
siempre ha velado por mi felicidad
y la de los que me rodean…

A mis padres,
los principales motores de mi vida, quienes
han procurado inculcarme los mejores
principios y el espíritu de lucha y coraje para
hacer que todo sueño se haga realidad.

A mis hermanos,
porque han sido el impulso que me ha permitido
llegar hasta donde estoy ahora. Sin duda, buena
parte de lo que soy es por ustedes, *manos*.

GRACIAS

A la familia Sánchez Sánchez,
quienes siempre han velado por mi bienestar
y de quien nunca me ha faltado
su consejo en los momentos difíciles.

A la familia Guzmán Baena,
quienes a pesar de la distancia siempre me han
apoyado y dado ánimos para nunca
darme por vencido.

GRACIAS

A mis amigos de Prepa 9*, Los Pamboleros*:
Diego Pérez, Norman Álva, Christian Federico y Xchel Vázquez,
porque a pesar de que nos unió algo tan simple como
un balón de futbol, queda claro que una amistad
puede perdurar por siempre a pesar de las distancias.

A mis compañeros de FES Aragón:
Alejandro Rubalcava, Mario Martínez, Jorge Gómez,
Gian Carlo Nuño, Christian Castillo, Araceli Martínez y Yanira Meneses,
ya que aunque pasamos buenos y malos momentos
me enseñaron la importancia de defender aquello en lo que creemos.

Pero sobre todo, a mis mejores amigos:
Israel Vázquez, Omar Salvador, Diana Jiménez,
Arturo Nava y David Hernández, porque cada momento con ustedes se
ha convertido en una enseñanza que no tiene precio, y su amistad
ha sido indispensable en mi vida para poder
seguir siendo una mejor persona.

GRACIAS

A la Ing.Yazmín Diana Reyes Torres,
porque siempre me brindó su confianza y de quien admiro
su dedicación, profesionalismo y liderazgo.
Pero sobre todo, por ofrecerme una ventana a su amistad,
la cual ha resultado invaluable.

Al Ing. Adolfo Quintana Teruel,
porque su amistad, experiencia y calidad humana han sido
importantes en las decisiones más importantes
y determinantes.

Al Staff del Centro de Atención a Usuarios de DGSCA,
porque han sido como mi segunda familia y de
quien siempre he recibido las mejores atenciones.

GRACIAS

A la familia Delgado Vallejo,
porque desde siempre han tenido las
mejores atenciones y un gran respeto tanto
para mí como para mi familia…

… en especial a **Thalía**, porque a pesar de que ya no te
encuentras físicamente con nosotros, esta es una pequeña
manera de hacerte saber que en nuestras vidas aún sigues
presente y que tienes que ver en gran medida con la consecución
de este logro.

INDICE GENERAL

INTRODUCCIÓN

Cuando se habla del tratamiento y persistencia de la información, el desarrollo de los Sistemas de Información cobra en la actualidad una importancia muy marcada. La confiabilidad que deben ofrecer a los usuarios finales al momento de ser planificados, desarrollados y liberados debe ser tan óptima como para que se puedan encontrar otras características deseables tales como eficiencia, robustez y portabilidad, por mencionar algunas.

Sin embargo, para alcanzar estos objetivos hoy en día se cuentan con herramientas, lenguajes y técnicas administrativas que aparte de ser novedosas, poderosas y relativamente sencillas de utilizar, que ayudan a tener un acercamiento más próximo a estas prioridades.

El lenguaje de programación Java es un ejemplo claro de lo anterior, ya que en la actualidad ha tenido una importante aceptación por parte de los programadores tanto principiantes como experimentados por su sencillez encaminada el desarrollo en base a la programación Orientada a Objetos (POO), y la enorme cantidad de librerías existentes que puede ayudar a simplificar procesos complejos.

Por otro lado, independientemente del lenguaje, plataforma o herramienta que se desee utilizar para el desarrollo de un Sistema de Información en particular, existe una situación clave que es importante tomar en cuenta con el objeto de garantizar el éxito en la realización de un sistema: la planeación. Siempre que el tiempo, los recursos y la disposición sean invertidos por parte del equipo de trabajo en forma racional e inteligente, se puede garantizar el éxito del proyecto.

El presente trabajo pretende como principal objetivo amalgamar el manejo del lenguaje de programación Java en el desarrollo de Sistemas de Información eficientes, orientado a las necesidades fundamentales de los usuarios finales, en este caso, el personal del Centro de Atención a Usuarios de la DGSCA, encargado de realizar el proceso de asignación de cuentas de acceso a los usuarios que requieran el servicio de la Red Inalámbrica Universitaria a través del sitio https://www.riu.unam.mx.

Los conocimientos teóricos adquiridos sobre la administración de proyectos, así como de todos los elementos propios del lenguaje Java para el desarrollo del sistema fueron adquiridos en el Diplomado en Java Máster, Segunda Generación, impartido en las instalaciones de la FES Aragón.

El diplomado se mantiene estructurado en dos capítulos principales. El primer capítulo engloba los conocimientos básicos adquiridos en el Diplomado Java Máster y que abarcan desde los conceptos teóricos más importantes referentes a la administración de proyectos hasta los tópicos básicos, intermedios y avanzados sobre del lenguaje de programación Java. El panorama general de cada uno de los módulos es presentado a continuación:

- Módulo 1: Conceptos básicos, metodologías y técnicas más importantes en la administración de proyectos.
- Módulo 2: Introducción al Lenguaje de Programación Java. Conceptos más importantes.
- Módulo 3: Clases Avanzadas y Colecciones. Entorno gráfico (Java Swing).

- Módulo 4: Conexión a bases de datos con JBBC y tópicos básicos del entorno Enterprise Java beans (EJB).
- Módulo 5: Páginas JSP y Servlets.
- Módulo 6: Estudio del Framework Struts.
- Módulo 7: Estudio del Framework Java Server Faces (JSF).

El segundo capítulo muestra el desarrollo del proyecto de migración del sitio de la Red Inalámbrica Universitaria a tecnología respaldada por el lenguaje Java. En esta parte será necesario contemplar el análisis de los problemas más comunes que el propio departamento presenta para dar de alta a sus usuarios, los objetivos básicos que esta nueva propuesta debe incluir y el desarrollo detallado de cada una de las acciones realizadas, visualizando las diferencias existentes entre el sistema original y la solución propuesta.

Cabe mencionar que para la realización del sistema mencionado anteriormente, se tuvo la necesidad de contar con información adicional a la proporcionada en el diplomado, por lo que será importante para el lector se permita dar un tiempo para consultar la bibliografía adjunta con el objeto de que pueda entender algunos de los listados de código que se han propuesto.

Capítulo I

Informe de contenidos sobre el Diplomado en Java Máster

1.1: Administración de Proyectos

1.1.1: Análisis Estructurado de un Sistema de Información.

Actualmente, la Ingeniería del Software forma parte de la vida diaria de los analistas encargados de tomar las decisiones de desarrollo e implementación de los Sistemas Informáticos. Así mismo, los sistemas no sólo se conforman de Bases de Datos o de aplicaciones; es por esto que en su conjunto se integran de lo que hoy en día se conocen como las Tres capas de los Sistemas de información, desde un punto de vista funcional.

En principio, un sistema puede considerarse como un conjunto de componentes, subsistemas o capas que son capaces de interactuar entre sí, aportando sus salidas o resultados correspondientes para convertirse en las entradas de un siguiente subsistema. Bajo este concepto, es posible realizar un análisis en donde las estructuras que debe contener cualquier sistema de información son:

- *Capa de Negocios*: Esta capa no es tangible, ya que en ella se encuentran inmersas las otras dos capas (Persistencia y Aplicación). Esta capa refleja precisamente las *reglas* de la realidad que se pretende modelar y transportar a un sistema de información, todo bajo el concepto denominado *"Reglas de Negocio"*.

- *Capa de Aplicación*: Esta capa se puede considerar tangible en el sentido de componentes, ya que representa el cuerpo del sistema. Particularmente esta capa es muy importante, ya que su exactitud, eficiencia y robustez es la carta de presentación hacia los usuarios que interactúan con el sistema.

- *Capa de Persistencia*: Representa la información con la que el Sistema de Información realiza sus actividades o procesos, y está sustentada normalmente en Bases de Datos.

1.1.2: Componentes de un Sistema de Información.

El Análisis Estructurado proporciona una visión de componentes de los Sistemas Informáticos, a nivel interacción o a nivel procesos, la cual es posible detallar en 5 aspectos fundamentales, los cuales se precisan a continuación:

- *Recursos Humanos*: Son los distintos tipos de usuarios que pueden intervenir tanto directa e indirectamente en el Sistema. Este nivel permite incluir a usuarios normales, técnicos, directivos, especialistas, analistas, consultores, etc.

- *Datos*: Es el componente que se envolverá en el uso, edición, mantenimiento de la información que servirá de entrada y salida al sistema. Esto, junto con la implementación de mecanismos de seguridad, puede permitir lograr un alto grado de integridad en el tratamiento de los datos.

- *Actividades*: Este componente se puede considerar y analizar desde las perspectiva de los distintos tipos de usuarios. Por ejemplo, los usuarios normales (no técnicos, y a menudo finales de un sistema) ven a las actividades como procesos simples de entrada y salida. En contraparte, el analista ve la necesidad dentro de sus límites, de replantear de nuevo los procesos existentes con el objeto de eliminar las redundancias y resaltar los valores de eficiencia, robustez y simplicidad.

- *Conectividad*: Se refiere al intercambio de datos entre las distintas partes físicas que pueden llegar a integrar un sistema informático. Debe realizarse con apoyo de diferentes mecanismos de seguridad, para lograr un alto grado de integridad en la comunicación e intercambio de datos.

- *Tecnología*: Se refiere al hardware y software que interviene en todo el ciclo de vida del Sistema de Información, ya sea desarrollado por el grupo de trabajo o adquirido.

1.1.3: Fases y Modelos del Ciclo de Vida de un Sistema de Información.

Un **proyecto** es un conjunto de actividades planificadas, ejecutadas y supervisadas, que tiene como objeto crear un producto o servicio único. Es importante cumplir con un conjunto de características para que pueda considerarse como tal:

a) Persecución de uno o varios objetivos (dependiendo de la necesidad de la organización).
b) Las actividades deberán ser perfectamente planificadas, ejecutadas y supervisadas.
c) Disponibilidad limitada en recursos.
d) Limitaciones en tiempo: Acotado en términos del principio y fin del mismo y capaz de determinar si los objetivos trazados pueden alcanzarse o no. (fracaso o éxito en el corto, mediano o largo plazo).
e) Deben existir resultados únicos.

De esta manera es posible identificar que las ***fases*** del ciclo de vida de un sistema de información pueden incluir los siguientes aspectos:

Planificación: Es el proceso de establecer metas y elegir los medios para alcanzar las metas establecidas. También se definen e identifican las acciones y actividades prioritarias mediante las alternativas de solución propuestas.

Análisis: Esta fase implica el estudio continuado del problema, centrándose especialmente en la organización, y no en la tecnología. Así mismo, se lleva a cabo el inicio de la Ingeniería del Software, donde se realiza un análisis exhaustivo de las partes del Sistema, identificando subsistemas, entidades, clases, métodos, atributos, relaciones, módulos, etc., y sus respectivas funciones y comunicaciones entre estos.

Diseño: A partir del modelo de la organización obtenida en la fase de análisis se obtiene un modelo del sistema, en donde se comienza a detallar la solución final, y cuyo diseño se acerca dependiendo de la solución tecnológica que se implementará. Los productos a obtener en esta fase, son las especificaciones, y modelos que se envían al grupo de desarrollo (programación), a manera de 'prototipo'.

Implementación: Esta fase implica la construcción y ensamblaje de los distintos componentes que integran la solución global. A esta fase se le conoce como 'producción'.

Soporte: Es la fase de mejoramiento y mantenimiento permanente del sistema y sus componentes, lo que puede llegar a dar lugar a un nuevo inicio del Ciclo de Vida del Sistema de Información.

Para la realización de cualquier tipo de proyecto se requiere un ciclo, en el que se constituyan diferentes etapas, y por las que se pueda tener un control de ellas lo más exitoso posible. Es por ello que se han propuesto ***modelos***, aplicados de alguna forma a las necesidades que se presenten. Los modelos que brevemente se presentan a continuación constituyen algunos de los más importantes:

Modelo "Cascada Pura".

Este es uno de los modelos del que se toman muchas bases para otros modelos. Un proyecto progresa a través de una secuencia ordenada de pasos partiendo del concepto inicial del software hasta la prueba del sistema. El proyecto realiza una revisión al final de cada etapa para determinar si está preparado para pasar a la siguiente etapa. Cuando la revisión determina que el proyecto no está listo para pasar a la siguiente etapa, permanece en la etapa actual hasta que esté preparado.

Modelo "Codificar y corregir".

El modelo 'codificar y corregir' es un modelo poco útil, pero bastante común. El modelo no conlleva ninguna gestión, no se pierde tiempo en la planificación, en la documentación, en el control de calidad, o en cualquier otra actividad que no sea la codificación pura. Para proyectos pequeños de demostración de conceptos, demostraciones de duración corta o prototipos desechables, este modelo puede ser útil.

Modelo "Espiral".

Este es un modelo de riesgos y divide cada uno de los proyectos en subproyectos o mini proyectos para hacer más eficiente su desarrollo. Cada mini proyecto se centra en uno o más riesgos importantes hasta que todos éstos estén controlados. Los riesgos llevan consigo la aplicación de seis pasos u objetivos que se muestran a continuación:

1. **Determinar objetivos, alternativas y límites.**
2. **Identificar y resolver riesgos.**
3. **Evaluar las alternativas.**
4. **Generar las entregas de esta iteración, y comprobar que son correctas.**
5. **Planificar la siguiente iteración.**
6. **Establecer un enfoque para la siguiente iteración (si se decide ejecutarla).**

La idea básica indica que se parte de una escala pequeña en medio de la espiral, se localizan los riesgos, se genera un plan para manejarlos, y a continuación se establece una aproximación a la siguiente iteración.

Modelo "Entrega por Etapas".

En este modelo las consideraciones de requerimientos se van tomando en cuenta al entregar cada una de las etapas y con esto se tiene bien especificado hacia donde se dirige el proyecto. Proporciona una funcionalidad útil en las manos de su cliente antes de entregar el 100% del proyecto al final del mismo. La entrega es por etapas sucesivas a lo largo del proyecto. El principal inconveniente de la entrega por etapas es que no funcionaría sin una planificación adecuada tanto para niveles técnicos como para niveles de gestión.

Modelo "Diseño por herramientas".

Este modelo se basa en la posibilidad de desarrollar la mayor cantidad de herramientas posibles para la solución del proyecto proyectadas no a un uso específico, sino un uso general del mismo modelo. En otras palabras, se incorporan las herramientas disponibles (estructuras de aplicación completas, entornos de programación visuales y de bases de datos, entre otros) y se utilizan las que puedan satisfacer las necesidades, teniendo un factor de ventaja importante: el tiempo.

Selección del ciclo de vida más rápido o eficiente para el proyecto.

Es importante considerar que distintos proyectos tienen necesidades diferentes, incluso si todos necesitan ser desarrollados lo más rápido posible. No existe "un modelo de ciclo de vida de desarrollo rápido", debido a que el modelo más efectivo depende del contexto en el que se utilice.

Un modelo de ciclo de vida apropiado puede orientar el proyecto y ayudar a asegurar que con cada paso se acerque más a la consecución del objetivo. Se puede aumentar la velocidad de desarrollo, mejorar la calidad, el control y el seguimiento del proyecto, minimizar gastos y riesgos, o mejorar las relaciones con los clientes.

1.1.4: ¿Qué es la Administración de Proyectos?

La **Administración de Proyectos** se puede ver como una serie de relaciones entre objetivos múltiples; en donde los administradores tienen que decidir cuales son las metas más importantes, y cuales se pueden llevar con más tranquilidad, con el objetivo de lograr un éxito global a la organización. Pueden visualizarse más fácilmente si el proyecto se ve como un cubo en el que cada eje representa una meta importante del proyecto:

- Costo
- Tiempo
- Alcance
- Calidad (Dimensión 4)

En la figura 1.1 se deja ver una muestra gráfica de estas relaciones y del modelo propuesto, dejando ver que no todas las variables pueden ser abarcadas en su totalidad.

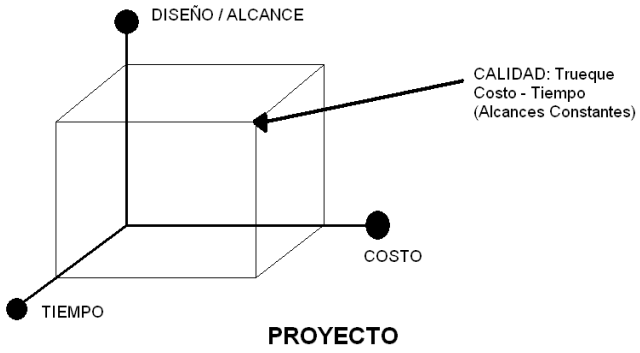


Figura 1.1: Relación de Costos, Tiempos y Alcances en la Administración de Proyectos

Muchos administradores de proyectos experimentados saben que en términos realistas pueden lograr una o quizá dos de estas metas en casi todos los casos.

Dado que la meta de calidad es la menos visible y la más difícil de medir, con frecuencia no se logra en los proyectos. Si el alcance se mantiene constante, a veces las metas de tiempo y costo no se alcanzan.

1.1.5: Elementos fundamentales en el Diseño Estructurado de un Sistema de Información.

Cuando se decide realizar la planeación del diseño general de un proyecto, es necesario considerar una serie de actividades que se deben integrar. Estas actividades sugieren abarcar los siguientes puntos, con el único objetivo de hacer del diseño una estructura de pasos a seguir:

a) **Definición del proyecto**: En toda actividad a realizar se requerirá contar con conocimientos precisos y claros de lo que se va a ejecutar, su finalidad, viabilidad, elementos disponibles, capacidad financiera, etc. Esta etapa es importante aclarar que no forma parte del método.

b) **Lista de Actividades**: Es la relación de actividades físicas o mentales que forman procesos interrelacionados en un proyecto total.

c) **Elaboración de Esquemas y Diagramas de apoyo:**
   - **Matriz de Secuencias:** Ya sea por antecedentes o por secuencias.
   - **Matriz de Tiempos**: Para la medición de cantidades estimadas por los responsables de los procesos: El tiempo medio (M), el tiempo óptimo (o) y el tiempo pésimo (p).
   - **Matriz de Información:** Para la muestra de los eventos, secuencias, interrelaciones y el camino critico de las actividades planteadas.

- **Diagramas de representación de los datos en base al tipo de usuario.** Apoyos en Diagramas organizacionales, Esquemas de Entidad-Relación, Diagramas de Estructuras.
- **Diagramas de Actividades, Procesos y Transacciones.** Elaborados a partir de la perspectiva de los distintos tipos de usuarios que intervienen.
- **Diagramas de Gantt.** Elaborado para visualizar los avances día a día de las actividades contempladas para la realización del proyecto.

En cuestión de la elaboración de los esquemas y diagramas, mientras mejor elaborados y precisos se encuentren, más fácilmente podrán ser interpretados por la gente de desarrollo una vez realizados los análisis correspondientes. Un desarrollo deficiente puede repercutir en factores determinantes, tales como retrasos de tiempo e incoherencia con la información existente proporcionada por el cliente.

1.1.6: Factores de calidad externos en la elaboración de Sistemas de Información.

En un proyecto de Sistemas de Información, se deben de contar con factores de calidad tanto para garantizar la satisfacción del usuario final en el producto entregado en el momento, como para asegurar que dicho sistema cumplirá con su cometido a corto, mediano o largo plazo. Analicemos los factores de calidad externa más importantes:

a) *Exactitud*: Es la habilidad de los productos de software para realizar sus tareas precisas, tal como lo define su especificación. Si un sistema no hace lo que se supone que debe hacer, cualquier otra propiedad que tenga — si es rápido, tiene buena apariencia...— importa muy poco.

b) *Robustez*: Es la habilidad de los sistemas de software para reaccionar apropiadamente a las condiciones anormales. Complementa la exactitud. Exactitud tiene que ver con la conducta del sistema cubierta por su especificación. Robustez caracteriza que sucede fuera de dicha especificación.

c) *Extensibilidad*: Es la facilidad de adaptación del sistema hacia los cambios de especificación. El problema más usual con la extensibilidad es de escala. Para sistemas pequeños, usualmente el cambio no es un problema; pero conforme el sistema crece, se torna en algo mucho más difícil de adaptar, por lo que hay que tenerlo presente siempre.

d) *Reutilización*: Es la habilidad de los elementos de software para servir en la construcción de muchas aplicaciones diferentes. Resolver el problema de reutilización significa, en esencia que la cantidad de software que será necesario desarrollar es menor, de ahí que hay que dedicar más esfuerzo (por el mismo costo total) a mejorar los otros factores, como exactitud y robustez.

e) *Compatibilidad*: Es la facilidad para combinar un elemento de software con otro. Es importante debido a que no se desarrollan productos de software en el vacío, sino que necesitan poder interactuar entre ellos mismos. La clave para la compatibilidad radica en la homogeneización del diseño, y en acuerdos en el uso de estándares y convenios para la comunicación entre equipos y programas.

f) *Eficiencia*: Es la habilidad del software para poner la cantidad mínima de demanda sobre los recursos de hardware como sea posible (tiempo de procesador, espacio ocupado en memorias, ancho de banda usado en dispositivos de comunicación, etc.).

g) *Portabilidad*: Es la facilidad de transportar productos de software a varios ambientes de hardware y software.

h) *Fácil de usar*: Es la simplicidad con la que la gente de varios trasfondos y cualidades pueden aprender a usar productos de software y aplicarlos para resolver problemas. También incluye la facilidad de instalación, operación y monitoreo.

Mientras más factores de calidad existan en un Sistema de Información, podrá ser capaz de ofrecer mayor seguridad y confiabilidad tanto a los usuarios finales, como a la información con la que se esté trabajando. Un buen Sistema de Información repercute positivamente en aspectos tales como en costos de mantenimiento y rendimiento, así como en una sencilla administración del mismo a largo plazo, por mencionar algunas ventajas visibles.

1.2: El Lenguaje de Programación Java

1.2.1: Antecedentes.

El lenguaje de programación Java es un poderoso y completo lenguaje de programación orientado a objetos desarrollado a principios de los años 90 en Estados Unidos por *Sun Soft Inc*. En la actualidad, Java se encuentra inmerso en infinidad de aplicaciones que utilizamos en la vida cotidiana: Algunos de los ejemplos más prácticos se encuentran básicamente en:

- Programas que han sido diseñados para que funcionen en un navegador Web y en servicios Web.
- Desarrollo de aplicaciones para servidores como foros en línea, tiendas, encuestas, procesamiento de formularios HTML, etc.

- Combinaciones de aplicaciones o servicios basados en la tecnología Java para crear servicios o aplicaciones totalmente personalizados.
- Desarrollo de potentes y eficientes aplicaciones para teléfonos móviles, procesadores remotos, productos de consumo de bajo costo y prácticamente cualquier dispositivo digital.

En principio los diseñadores de Java tuvieron una premisa en mente: crear "un lenguaje de programación sencillo, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portátil, de gran rendimiento, multitarea y dinámico".

La primera versión de Java fue escrita en apenas año y medio, teniendo por nombre en primera instancia *'Oak'* y empleándose para uso interno por la gente de Sun. La idea original conllevaba crear un lenguaje orientado a objetos que fuera independiente de la plataforma, y que se pudiera usar en todos los ordenadores. *'Oak'* pasó a llamarse **Java** en 1995 cuando fue lanzado para el uso público.

1.2.2: Comparación de Java con otros lenguajes de programación (C y C++).

Java es simple porque elimina la complejidad de los lenguajes de programación como el C y el C++. Incorpora nuevas características como un recolector automático de elementos no utilizados y elimina aspectos confusos y poco utilizados del C++ como la sobrecarga de operadores. También elimina el manejo difícil y complejo de los apuntadores.

Comparando los productos finales, un programa realizado en Visual C++ de Microsoft apoyado en los recursos de las Microsoft Foundation Classes (MFC) abarca

demasiado espacio en memoria, sin contar con la inclusión de sus respectivas librerías de enlace dinámico (DLL) necesarios para que en un ambiente Windows se puedan ejecutar. Esto provoca que este tipo de programas deban ser completamente ejecutables en la computadora en que residen.

Por otro lado, Java contiene una base de objetos y herramientas que se utilizan con una sintaxis muy similar a la del lenguaje C y C++ y que tienen como principal ventaja permitir utilizar los mismos códigos fuente de programación distribuyéndolos a través de las redes de cómputo y operando de forma tal que son neutrales o independientes a las diferentes arquitecturas de computadoras. Todo esto gracias a la implementación del concepto de '***máquina virtual***' (Java Virtual Machine, o **JVM**), que puede existir en cada plataforma y sistema operativo en los que se permita ejecutar aplicaciones escritas en Java.

1.2.3: Java dentro de la Programación Orientada a Objetos: Conceptos básicos.

Java es un lenguaje de programación Orientado a Objetos. Este paradigma describe una forma de desarrollar software describiendo los problemas mediante el uso de elementos u objetos propios del espacio del problema y no mediante un conjunto de pasos secuenciales que se ejecutan en la computadora, como se había realizado anteriormente con el concepto de Programación Estructurada. Un buen diseño Orientado a Objetos produce componentes reutilizables, extensibles y sostenibles.

Los **API Core**, o simplemente llamadas **API´s** de Java son una colección de estos componentes, previamente creados, denominados *bibliotecas de clases*. El programador no necesita crear todo de nuevo, simplemente utiliza estas bibliotecas estándar que han ido evolucionando a lo largo del tiempo, obteniendo más componentes con una nueva versión o liberación de Java.

Para su desarrollo e implementación, es importante que el programador Java pueda comprender conceptos fundamentales de la Programación Orientada a Objetos, tales como lo son las clases, objetos, métodos, y constructores.

1.2.3.1: Clases.

En el mundo real frecuentemente se encuentran muchos objetos individuales del mismo tipo como: automóviles, computadoras, mesas, bicicletas. Hablando de bicicletas por ejemplo, se puede decir que pueden existir miles de la misma marca y modelo, cada una construida a partir del mismo prototipo o plantilla y con los mismos componentes.

Llevando este concepto a términos orientados a objetos, se puede decir que una bicicleta X es una ***instancia*** de la **clase de objetos** conocida como **bicicletas**. Una c**lase**, por lo tanto, es la plantilla a partir de la cual se crean objetos individuales concretos. La sintaxis de una simple definición o declaración de clase dentro del lenguaje Java es:


```
[modificadores] class  nombre [extends Super]
  [implements interfaces]
 {
    cuerpo
  }
```

Adicionalmente, es posible incluir mucho más información acerca de la clase tal y como se presume en la Tabla 1.1:

public	La clase es públicamente accesible.
private	La clase no es públicamente accesible.
abstract	La clase no puede ser instanciada.
final	La clase no puede ser “subclaseada”
class *nombre*	Nombre de la clase.
extends *Super*	Superclase de la clase
implements i*nterfaces*	Interfaces implementadas por la clase.

Tabla 1.1: Parámetros opcionales en la creación de clases en Java

1.2.3.2: Objetos.

Un objeto es una cosa, generalmente extraída del vocabulario del espacio del problema o del espacio de la solución. Es una representación detallada y particular de algo existente de la realidad. Todo objeto tiene 3 principales características:

- **Identidad** (puede nombrarse o distinguirse de alguna manera de otros objetos).
- **Estado** (generalmente hay algunos datos asociados a él).
- **Comportamiento** (se pueden hacer cosas al objeto, y él puede a su vez puede hacer cosas a otros objetos).

Bajo el contexto de Java, todo es tratado a nivel de objetos. Se puede manipular objetos a través de *handles* (referencias, identificadores). El *handle* o referencia no necesariamente se encontrará conectado a un objeto. Ejemplo: *String s;*

En este caso, sólo fue creada la referencia por lo que es muy importante inicializarlo:

String s = "Curso de Java";

Para conectar un *handle* o referencia a un objeto se usa la palabra llave **new** que es un operador de Java y significa literalmente más o menos lo siguiente: "*Créame un nuevo objeto de este tipo*" o "*Créame una nueva instancia o ejemplar de este tipo*". Ejemplo de esto es el siguiente:

*String s = **new** String("Curso de Java");*

El efecto de crear un objeto con el operador **new** se traduce en la asignación de memoria para ese objeto en la RAM.

1.2.3.3: Métodos.

Los métodos son algo parecido a las funciones del lenguaje C /C++, sin embargo en Java no se permiten colocar métodos fuera de la definición de clase. Con los métodos implementamos el comportamiento de los objetos. Hay varios tipos de métodos:

Métodos de instancia. Son miembros de la clase y son métodos que pueden ser utilizados cuando se crea un objeto de la clase a la que pertenecen, esto es, cuando se crea una instancia de la clase.

Métodos de clase. Son miembros de la clase y sus métodos pueden ser invocados sin necesitar crear una instancia de la clase, y se identifican fácilmente porque cuentan con un modificador.

Métodos constructores. Son métodos que tienen el nombre de la clase y que son invocados al instanciar un objeto o crear un ejemplar de la clase al que pertenecen con el operador **new**.

La sintaxis de una declaración simple de un método es:

```
[modificadores]  tipo-de-retorno  nombre-método (lista-de-parámetros) [ throws claseExcepcion1,.., claseExcepcionN ]
      {
         Cuerpo
}
```

Donde:

modificadores es opcional y puede ser uno más de las siguientes ocho palabras llave:

{static, abstract, final, native, synchronized, public, protected, private }

tipo-de-retorno es cualquier nombre de clase o una de las siguientes nueve palabras reservadas:
{void, boolean, byte, char, short, int, long, float, double }

nombre-método es cualquier identificador válido.
lista-de-parámetro es una secuencia de declaraciones de parámetros.

[*throws claseExcepcion1,.., claseExcepcionN*] es una lista de posibles excepciones que en un momento dado el código del método puede lanzar.

Java soporta la ***sobrecarga*** de métodos: múltiples métodos en la misma clase pueden tener el mismo nombre pero con diferente lista de parámetros, identificaciones y definiciones. Permite a las instancias de su clase tener una interfaz más sencilla para otros objetos y comportarse en forma distinta en base a la entrada de ese método.

Cuando uno llama a un método en un objeto, Java hace coincidir su nombre, número y tipo de argumentos para seleccionar que definición de método ejecutar.

1.2.3.4: Constructores.

Un **método constructor** es una clase especial de método que determina como se inicializa un objeto cuando se crea. A diferencia de los métodos ordinarios, no se puede llamar a un método constructor de forma directa; en su lugar, Java los llama de manera automática. Esto se logra utilizando el atributo *new* para crear una nueva instancia de clase. Cuando esto ocurre, se realizan tres acciones fundamentales:

- Se asigna memoria para el objeto.
- Se inicializan las variables de instancia relacionadas con el objeto.
- Se llaman a los métodos constructores.

Los constructores son muy parecidos a los métodos ordinarios excepto por que siempre tienen el mismo nombre de la clase y tienen un tipo de retorno común. La figura 1.2 muestra la declaración de un método constructor, enfocado a la clase 'Persona' con parámetros como nombre y edad incluidos.

Al igual que los métodos ordinarios, los constructores también pueden tomar varios y diferentes tipos de parámetros, lo que permite crear objetos con las propiedades deseadas, es decir, también son capaces de soportar ***sobrecarga.***

```
class Persona {
   String nombre;
   int edad;

   Persona(String n, int e) {
   nombre=n; edad=e;
   }
}


public static void main(String [] args) {
   Persona p;
   p=new Persona ("Griselda", 34); //Llamada a constructor
}
```


Figura 1.2: Declaración y llamada del método constructor 'Persona'

1.2.4: Elementos avanzados del lenguaje Java.

Hasta este momento, se han incluido solamente los conceptos que es conveniente que el programador en Java conozca en primer lugar acerca del lenguaje. Como se puede apreciar, son conceptos que se tornan fáciles de comprender si se tiene cierta experiencia y práctica con algún otro lenguaje de programación de reciente generación, que por supuesto sea orientado a objetos, como lo es C++.

Sin embargo, Java permite realizar acciones que van más allá de las que un lenguaje orientado a objetos podría llevar a cabo con mayor eficiencia. Tal es el caso del manejo de errores y el multiprocesamiento, los cuales se describen brevemente a continuación.

1.2.4.1: Manejo de errores (excepciones).

Existe una regla de oro en programación: pueden ocurrir errores en un programa. Lo que interesa saber es que pasa después de que ocurre un error de programa.

- ¿El programa imprime un mensaje de error y se para?
- ¿Puede un programa recuperarse y seguir su ejecución después de ocurrido el error?

El lenguaje de programación Java usa las ***excepciones*** para el manejo de errores. Por definición, una ***excepción*** es un evento que modifica el flujo normal de instrucciones durante la ejecución de un programa. Cuando ocurre un error dentro de un método, éste crea un objeto y lo lleva fuera del sistema en tiempo real.

Este objeto contiene información acerca del error, incluyendo su tipo y el estado del programa cuando ocurrió el error. Al manejar este objeto fuera del sistema en tiempo real se le llama ***disparo de excepción***. Después de éste, el sistema en tiempo real intenta encontrar algo que maneje el error.

Para construir un manejador de excepciones hay que definir las instrucciones u operaciones que pueden disparar una excepción e incluirlas dentro de un bloque ***try***. Si ocurre una excepción dentro del bloque ***try***, ésta es manejada por un manejador de excepción asociado. Para asociar un manejador de excepción con un ***try***, se utiliza el bloque ***catch***. La figura 1.3 ejemplifica la declaración de un bloque manejador de errores básico.

```
try {
      . . .
      . . .
} catch (IOException e) { System.err.println("Caught IOException: " + e.getMessage());
} catch ( . . ) { . . . }
```

Figura 1.3: Declaración básica de un bloque de captura de errores

Cada bloque ***catch*** es un manejador de excepción y maneja el tipo de excepción indicado por su argumento. El tipo de argumento, ***ExceptionType***, declara el tipo de excepción que el manejador puede soportar, y debe ser el nombre de una clase heredada de la clase **Throwable.** El bloque ***catch*** contiene una serie de instrucciones las cuales son ejecutadas sólo si el manejador de excepciones es invocado. El sistema en tiempo real invoca al manejador de excepciones cuando la excepción disparada coincide con el tipo de excepción ***ExceptionType*** incluida en el ***try***.

1.2.4.2: Multiprocesos (threads).

Los hilos (threads) son similares a los programas secuenciales: tiene un inicio, una secuencia y un final. En cualquier momento dado durante el desarrollo del hilo, hay un punto simple de ejecución. Sin embargo un hilo no es un programa en sí mismo, en lugar de ello, un hilo corre dentro de un programa. La máquina virtual de Java permite que una aplicación tenga múltiples hilos de ejecución corriendo concurrentemente.

La figura 1.4 ejemplifica la aplicación de múltiples procesos dentro de un programa al momento de realizar su ejecución.

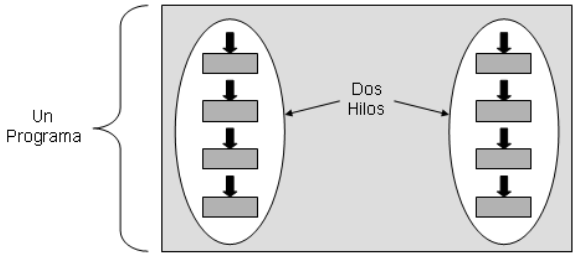


Figura 1.4: Esquema de threads múltiples dentro de un programa

Existen múltiples aplicaciones que pueden programarse utilizando hilos, para la implementación más adecuada es conveniente consultar la documentación de alto nivel API sobre hilos. El soporte básico para hilos en todas las versiones de la plataforma Java se encuentra en la clase **java.lang.Thread.**

1.3: Colecciones y Java Swing

1.3.1: Colecciones.

Una colección (*collection*, a veces llamada contenedor) es un objeto que agrupa múltiples elementos en una sola unidad. Las colecciones se usan para almacenar, recuperar, manipular, y comunicar datos agregados. Típicamente, las colecciones representan elementos de datos que forman un grupo natural, como una mano de póker (una colección de cartas), un archivo de correo (una colección de cartas), o un directorio telefónico (un mapeo de nombres a números telefónicos).

La estructura de colecciones de Java brinda los siguientes beneficios:

- Reduce el esfuerzo de programación.
- Incrementa la velocidad y calidad del programa.
- Permite interoperabilidad entre API´s no relacionadas.
- Reduce el esfuerzo para diseñar, usar y aprender nuevos API´s.
- Promueve la reutilización de software.

El núcleo de interfaces de colección (*core collection interfaces*) encapsula distintos tipos de colecciones. Éstas permiten que las colecciones sean manipuladas independientemente de los detalles de su presentación. El núcleo de interfaces de colección

es la piedra angular de la estructura de colecciones de Java. Como se puede ver en la figura 1.5, el núcleo de interfaces de colección forma una jerarquía.

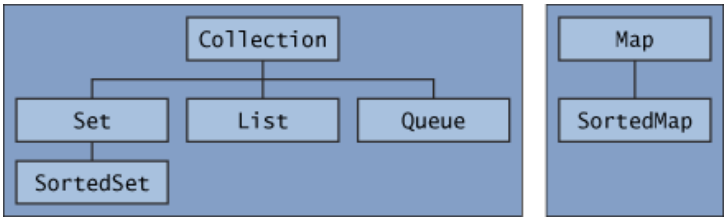


Figura 1.5: Jerarquía de clases en base a la clase Collection

1.3.1.1: La interfaz Collection.

Un objeto 'Collection' representa un grupo de objetos a los que se les denomina elementos. La interfaz 'Collection' se usa para pasar colecciones de objetos donde se desea trabajar con una **máxima generalidad**. La figura 1.6 muestra la definición de la interfaz Collection junto con los métodos más importantes que la componen:


```
public interface Collection<E> extends Iterable<E> {
        // Operaciones básicas
        int size(); boolean isEmpty();
        boolean contains(Object element);
        boolean add(E element);                          //optional
        boolean remove(Object element);                  //optional
        Iterator<E> iterator();

        // Operaciones por volumen
        boolean containsAll(Collection<?> c);
        boolean addAll(Collection<? extends E> c);       //optional
        boolean removeAll(Collection<?> c);              //optional
        boolean retainAll(Collection<?> c);              //optional
        void clear();                                    //optional

        // Operaciones de arreglos
        Object[ ] toArray();
        <T> T[ ] toArray(T[ ] a);
```

Figura 1.6: Definición de la interfaz principal Collection y sus métodos

1.3.1.2: La interfaz Set.

Un conjunto o 'Set' es una colección que **no contiene elementos duplicados**. La interfaz 'Set' contiene *únicamente* métodos heredados de la colección a lo que se le agrega la propiedad de que están prohibidos los elementos duplicados.

1.3.1.3: La interfaz List.

Un conjunto 'List' es una colección ordenada (algunas veces se le llama *secuencia*). Las listas pueden contener elementos duplicados. Además de las operaciones heredadas de la colección, la interfaz List incluye lo siguiente:

- **Acceso posicional:** Manipula elementos basado en su posición numérica en la lista.
- **Búsqueda**: Busca un objeto específico en la lista y regresa su posición numérica.
- **Iteración**: Extiende la semántica del iterador para aprovechar la naturaleza secuencial de la lista.
- **Intervalo de vista**: Ejecuta operaciones de intervalo arbitrarias *en la lista.*

La mayoría de los algoritmos polimórficos en las clases '*Collection'* se aplican específicamente a las listas. Con todos estos algoritmos a disposición, es muy fácil manipular listas. A continuación se muestra un resumen de estos algoritmos.

- **sort** — Ordena una lista usando una mezcla de algoritmos de ordenamiento, que proporcionan un ordenado rápido y estable (no reordena elementos iguales).
- **shuffle** — Permuta aleatoriamente los elementos en una lista.
- **reverse** — Invierte el orden de los elementos en una lista.
- **rotate** — Rota todos los elementos en una lista una distancia especificada.
- **swap** — Intercambia los elementos en las posiciones especificadas en una lista.

- **replaceAll** — Reemplaza todas las ocurrencias de un valor especificado con otro.

- **fill** — Sobrescribe todos los elementos en una lista con el valor especificado.
- **copy** — Copia la lista fuente en la lista destino.
- **binarySearch** — Busca un elemento en una lista ordenada usando el algoritmo de búsqueda binario.

Naturalmente estos algoritmos pueden ser desarrollados con elementos propios del lenguaje (ciclos, secuencias, asignaciones, etc.). Sin embargo, es más sencillo llamar a estos métodos ya desarrollados, incluso para efectos de rendimiento y sencillez.

1.3.1.4: La interfaz Queue (Cola).

'Queue' (cola) es una colección que almacena elementos previos a su procesamiento. Las colas proveen operaciones adicionales de inserción, borrado, e inspección. Cada objeto 'Queue' existe en dos formas: (1) uno que dispara una excepción si la operación falla (***IllegalStateException*** o ***NoSuchElementException***), y (2) la otra regresa un valor especial si la operación falla (ya sea ***null*** o ***false***, dependiendo de la operación). La estructura regular de la interfaz Queue se ilustra en la tabla 1.2:

	Dispara excepción	Regresa un valor especial
Inserta	add (e)	offer (e)
Remueve	remove ( )	poll ( )
Examina	element ( )	peek ( )

Tabla 1.2: Métodos de la interface Queue

1.3.1.5: La interfaz Map.

Esta interfaz recoge una estructura de datos que permite almacenar asociaciones {clave, valor} de forma que dada una clave se puede localizar inmediatamente el valor asociado. A veces se denominan diccionarios o incluso "arrays asociativos". Se satisfacen las siguientes propiedades:

- Las claves, de tipo K, no pueden estar duplicadas.
- Los valores, de tipo V, pueden ser cualesquiera.
- El tamaño del mapa se adapta dinámicamente a lo que haga falta.

El resultado es una especie de *array* de tamaño adaptable que, en vez de indexarse por posición, se indexa por medio de una clave. En la figura 1.11 se visualiza la definición de la interfaz Map junto con los métodos más importantes que la componen:

```
public interface Map<K,V> {

  // Operaciones Básicas
  V put(K key, V value);
  V get(Object key);
  V remove(Object key);
  boolean containsKey(Object key);
  boolean containsValue(Object value);
  int size();
  boolean isEmpty();

// Operaciones por volumen
  void putAll(Map<? extends K, ? extends V> m);
  void clear();

// Vistas de colección
  public Set<K> keySet();
  public Collection<V> values();
  public Set<Map.Entry<K,V>> entrySet();

  // Interfaz para elementos entrySet
  public interface Entry {
    K getKey();
    V getValue();
    V setValue(V value);
  }
}
```


Figura 1.11: Definición de la interfaz Map y sus métodos

1.3.2: Introducción a Java Swing – Interfaz Gráfica de Usuario.

Las interfaces gráficas de usuario (GUI´s) presentan un mecanismo amigable al usuario para poder interactuar con un programa. Al proporcionar distintos programas en los que los componentes de la interfaz de usuario sean consistentes e intuitivos, los usuarios pueden familiarizarse con un programa, incluso antes de utilizarlo. El ejemplo más común de GUI´s se puede encontrar en los navegadores, tales como Internet Explorer.

Las GUI´s son creadas a partir de controles o *widgets* (accesorios de ventana). Un componente de GUI es un objeto el cual interactúa con el usuario mediante el ratón, el teclado u alguna otra forma de entrada. Entre los componentes básicos que se encuentran dentro de los componentes GUI de Swing se encuentran los siguientes:

- Etiquetas (*JLabel*)
- Campos de Texto (*JTextField*)
- Botones (*JButton*)
- Cajas de selección (*JCheckBox*) y Listas (*JList*)

Un panorama más completo de los componentes Swing se puede visualizar en la figura 1.12:

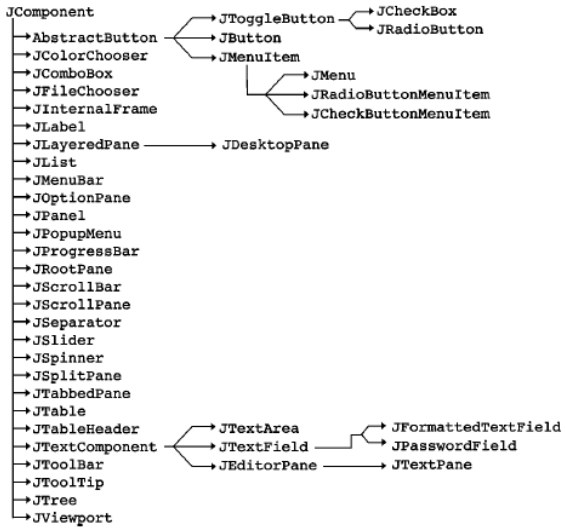


Figura 1.12: Jerarquía de componentes dentro de Java Swing

Las clases que crean los componentes de la GUI de Swing se encuentran en el paquete ***javax.swing***. La mayoría de los componentes de Swing están escritos, se manipulan y muestran completamente en Java. En su conjunto, forman parte de la JFC (Java Foundation Classes), las bibliotecas de Java para el desarrollo de GUI´s en diferentes plataformas.

1.3.2.1: Diferencias con el paquete AWT.

La mayor diferencia entre los componentes AWT y los componentes Swing es que estos últimos están implementados sin nada de código nativo, es decir, esta implementados en código Java. Esto significa que los componentes Swing pueden tener **más funcionalidad y portabilidad** que los componentes AWT, porque no están restringidos al denominador común (las características presentes en cada plataforma). Ejemplos más detallados sobre las diferencias existentes son los siguientes:

- Se puede modificar fácilmente el comportamiento o la apariencia de un componente Swing llamando a métodos o creando una subclase. Por ejemplo, haciendo que los elementos tipo botón puedan ser redondos. Así mismo, se puede especificar el Aspecto y Comportamiento que utilice el GUI del programa. Por el contrario, los componentes AWT siempre tienen el aspecto y comportamiento de la plataforma nativa.

- Las tecnologías asistidas como los lectores de pantallas pueden fácilmente obtener información desde los componentes Swing. Por ejemplo, una herramienta puede fácilmente obtener el texto mostrado en un botón o en una etiqueta.

Como se puede ver, el más sencillo de los componentes Swing es capaz de tener capacidades que van más allá de lo que ofrecen los componentes AWT.

1.3.2.2: Jerarquía de Herencia en Swing.

La jerarquía de herencia contiene clases que declaran atributos y comportamientos comunes para la mayoría de los componentes de Swing. Básicamente depende de 4 clases fundamentales: ***Object, Component, Container y JComponent***.

La clase Object es la superclase de la jerarquía de clases de Java. La clase Component (incluida originalmente en el paquete ***java.awt***) es una subclase de Object:

-Clase Component: Declara los atributos y comportamientos comunes de todas las subclases de Component. Con pocas excepciones, la mayoría de los componentes de la GUI extienden de la clase Component en forma directa o indirecta. Es importante comprender los métodos de la clase Component, ya que las operaciones más comunes para los componentes de la GUI tanto de Swing como de AWT se encuentran en esta clase.

-Clase Container: Administra una colección de componentes relacionados. En aplicaciones con objetos del tipo JFrame y en subprogramas, adjunta componentes al panel de contenido, que es un objeto de la clase Container. Métodos comunes que se originan en esta clase son método **add**, utilizado para adjuntar componentes al panel de contenido, y el método **setLayout** que permite a un programa especificar el administrador de esquemas que ayuda a un Container a posicionar y ajustar el tamaño de sus componentes.

-Clase JComponent: Es la superclase de la mayoría de los componentes de Swing. Esta clase declara los atributos y comportamientos comunes de todas las subclases de JComponent, incluyendo algunas características, tales como una apariencia visual adaptable para personalizar componentes, teclas de método abreviado, herramientas para el manejo de eventos comunes, soporte para tecnologías de ayuda como lectores de pantalla en Braille y para mostrarla en distintos lenguajes, entre otros.

1.3.2.3: Descripción de elementos básicos del paquete Swing.

- **Etiquetas (JLabel).**

Con la clase JLabel, se puede mostrar texto no seleccionable e imágenes. Es una subclase de JComponent. Si se necesita crear un componente que muestre un sencillo texto

o una imagen, reaccionando opcionalmente a la entrada del usuario, se puede hacer utilizando un ejemplar de JLabel.

- **Campos de Texto (JTextField – JPasswordField)**

Un campo de texto es un control básico que permite al usuario teclear una pequeña cantidad de texto y dispara un evento tipo *action* cuando el usuario indique que la entrada de texto se ha completado (normalmente, pulsando Enter). Generalmente se usa la clase JTextField para proporcionar campos de texto. Si se necesita proporcionar un Password Field (un campo de texto editable que no muestra los caracteres tecleados por el usuario) se utiliza la clase JPasswordField.

- **Botones (JButton, JCheckBox, JRadioButton)**

Un botón es un componente que el usuario emplea para desencadenar una acción. Un programa escrito en Java puede utilizar varios tipos de botones, incluyendo botones de comando, casillas de verificación, botones interruptores y botones de opción. Todos estos tipos de botones son subclases de AbstractButton (incluido en el paquete javax.swing), el cual incluye las características comunes para los botones de Swing.

Un botón de comando (JButton) funciona de la siguiente manera: cuando el usuario hace clic en él, genera un evento del tipo ActionEvent, el cual procesará las instrucciones que sean requeridas en la GUI. El texto en la cara de un objeto JButton se llama etiqueta del botón, y normalmente es declarada como parámetro en el constructor de la clase. Una GUI puede tener muchos objetos de tipo JButton, pero cada etiqueta de botón debe generalmente ser única en las partes de la GUI en que se muestre.

Los botones de estado JCheckBox y JRadioButton contienen valores de encendido/apagado o verdadero/falso. Las clases que conforman este tipo de botones son subclases de JToggleButton. Un objeto JRadioButton es distinto a un objeto tipo JCheckBox en cuanto a que generalmente hay varios objetos de tipo JRadioButton que se agrupan, y sólo uno de los objetos JRadioButton en el grupo puede estar seleccionado (verdadero) en un momento dado.

- **JComboBox**

Un cuadro combinado (algunas veces conocido como lista desplegable) proporciona una lista de elementos, de la cual el usuario puede seleccionar sólo uno. Los cuadros combinados se implementan con la clase JComboBox, la cual extiende de JComponent. Los objetos JComboBox generan eventos del tipo ItemEvent, al igual que los objetos JCheckBox y JRadioButton.

1.3.2.4: Manejo de eventos

Cada vez que el usuario teclea un carácter, pulsa un botón del ratón o selecciona un elemento de un menú, ocurre un evento. Cuando ocurre esta interacción, se envía un mensaje al programa. La información de los eventos de la GUI se almacena en un objeto de una clase que extiende a **AWTEvent**.

En la figura 1.13 se muestra una jerarquía que contiene muchas de las clases de eventos del paquete *java.awt.event*, que son utilizadas tanto en los componentes AWT como en los de Swing.

El mecanismo de manejo de eventos consta de 3 partes: El origen del evento, que es el componente específico con el cual interactúa el usuario. El objeto relacionado al evento, que encapsula la información acerca del evento que ocurrió. Finalmente el componente de

escucha del evento, es un objeto que recibe la notificación del origen del evento cuando éste ocurre (la escucha, literalmente) y se ejecuta en respuesta a ese evento.

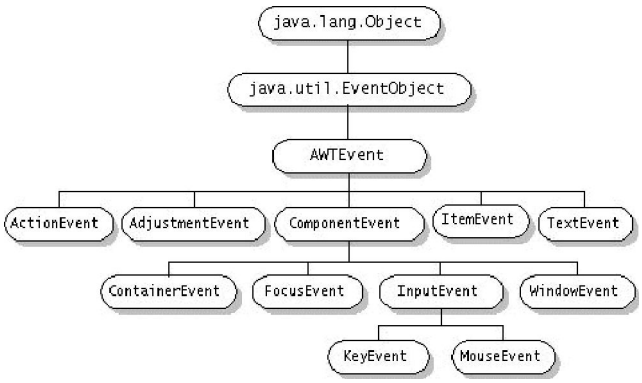


Figura 1.13: Jerarquía de Eventos dentro del paquete *java.awt.event*

Todo componente de la GUI soporta varios tipos de eventos, incluyendo eventos de ratón, eventos de tecla y otros más. Cuando ocurre un evento, éste se despacha solamente a los componentes de escucha de eventos del tipo apropiado. El '*despachamiento*' es el proceso por el cual se llama al método manejador apropiado en cada componente de escucha registrado, para el tipo de evento que ocurrió. Cuando ocurre un evento, el componente de la GUI recibe un ID de evento único, el cual especifica el tipo de evento. Este componente utiliza el ID para decidir a cual tipo de componente de escucha debe despacharse el evento, y para decidir cual método llamar en cada objeto de escucha.

1.4: JDBC – EJB

1.4.1: Introducción a las aplicaciones empresariales en Java.

Las aplicaciones empresariales han surgido con el objeto de romper el alcance que tradicionalmente se tenía en el desarrollo de Sistemas de Información: partiendo de las

rígidas aplicaciones para usuarios Simples (Single User), se ha cambiado el concepto para dar paso al desarrollo de sistemas Multiusuario, con la ventaja de que dichos sistemas sean usados por ilimitada cantidad de usuarios en un mismo tiempo y con diferentes finalidades.

El desarrollo de aplicaciones empresariales es una tarea compleja que requiere en la actualidad el dominio de diversas tecnologías, tales como bases de datos, manejo de transacciones, aplicaciones distribuidas y aplicaciones orientadas al esquema cliente-servidor, entre otras. Dentro del lenguaje Java, para poder complementar cada una de estas tecnologías, se tienen disponibles diferentes API´s de desarrollo, que involucran el desarrollo de tecnologías para abarcar tareas complejas, como lo son:

- Autenticación de usuarios.
- Soporte a usuarios múltiples.
- Persistencia de la información.
- Integridad de datos.
- Comunicaciones a nivel de cliente.
- Comunicación con otras aplicaciones Empresariales, entre otros.

Todas estas API´s en su conjunto, dentro del lenguaje Java son conocidas como J2EE (Java 2 Enterprise Edition), y actualmente se pueden encontrar aplicadas en sistemas donde se procesen grandes cantidades de información, y que requieran difundir servicios y procesar transacciones entre clientes y servidores.

1.4.2: JDBC (Java Database Conectivity).

La API relacionada a JDBC (Java Database Conectivity, Conectividad a Bases de Datos en Java) es una especificación desarrollada por el equipo de SUN Microsystems, que está orientada a describir la forma de cómo realizar conexiones a bases de datos, sin importar el fabricante o la plataforma en donde se encuentren.

JDBC está compuesto fundamentalmente por dos partes:

- **API**, con la documentación Java pertinente para el desarrollo de conexiones a bases de datos.
- **Driver**, en forma de un archivo de librerías (normalmente con extensión **.jar)** con todos los archivos *class* de conexión y manipulación de la base de datos.

Una de sus principales ventajas está en que tiene una composición al 100% basada en Lenguaje Java, lo que facilita se lleve a cabo su implementación. Así mismo, los fabricantes de los SMBD son los que escriben los controladores que se adecuaran a la especificación general JDBC, por lo que es necesario buscar en la documentación pertinente de la base de datos los requisitos para su implementación.

1.4.2.1: Diseño de una conexión a base de datos mediante JDBC.

Para realizar el proceso de conexiones a una base de datos, es necesario cumplir con una serie de pasos ordenados, los cuales se pueden resumir a continuación:

- **Carga del Driver (del SMBD).**

Todo proceso que requiera realizar una conexión a una base de datos requiere de 3 librerías fundamentales: las librerías **java.sql.* y javax.sql** ubicadas dentro de las mismas librerías de desarrollo de la especificación Java, y la librería de desarrollo, o driver que proporciona el fabricante del SMBD con el que se desea establecer la comunicación. Véase el siguiente ejemplo (figura 1.14) en donde se muestra la forma de realizar la declaración de la clase contenida en el driver del fabricante para realizar la conexión a la base de datos tanto en MySQL como en Oracle.

```
import java.sql.*;
public class TestMysql {
  public static void main(String[] argv){
    try{
      Class.forName("com.mysql.jdbc.Driver"); //para una conexión a una base de datos en MySQL
      Class.forName("oracle.jdbc.driver.OracleDriver");  //para una conexión a una base de datos en Oracle
            …
            …
      }catch(Exception e){System.out.println("Problema\n"+e);}
    }
}
```

Figura 1.14: Código para la carga del Driver en MySQL y Oracle en JDBC

- **Definición de la URL de conexión a la Base de Datos.**

Esta parte resulta particularmente especial, ya que es aquí donde se define el host donde se encuentra alojada la base de datos a donde hay que conectarse. Al igual que en la carga del driver del fabricante, la sintaxis cambia dependiendo del manejador de base de datos que se esté empleando. En el siguiente ejemplo (figura 1.15) se ilustra la forma de realizar este procedimiento tanto en MySQL como en Oracle.


```
import java.sql.*;
public class TestMysql {
  public static void main(String[] argv){
            String host;
            String dbName;
    try{
      Class.forName("com.mysql.jdbc.Driver");
      Class.forName("oracle.jdbc.driver.OracleDriver");
            host="jdbc:mysql://127.0.0.1/";
            dbName="jmaster1";
            hostOra="jdbc:oracle:thin:@127.0.0.1:1521:"
            …
            …
            }catch(Exception e){System.out.println("Problema\n"+e);}
    }
}
```

Figura 1.15: Código con la definición del URL de conexión en JDBC

- **Establecimiento de la conexión.**

Es aquí donde se representa la conexión a la base de datos, a partir de 3 parámetros importantes: La definición del host junto con la base de datos a conectar, el nombre de

usuario y la contraseña de acceso. A partir del objeto que resulte se pueden crear los objetos de acceso a la base de datos, ya sean para ser utilizados tanto en MySQL como en Oracle.

- **Creación de un Objeto Statement.**

Una vez creado el objeto de la clase *Connection*, se pueden crear distintos objetos que tendrán acceso a la base de datos, ya sea para realizar consultas o manipulaciones de la información. Estos objetos dependen de la clase *Statement*, tal y como se aprecia en la figura 1.17.

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

Figura 1.17: Creación del objeto Statement en JDBC

- **Ejecución de manipulaciones y consultas.**

Ya que se tiene un objeto de tipo *Statement* que llevará las instrucciones SQL que se desean ejecutar, será necesario guardar el resultado en un objeto dedicado: un objeto de la clase *ResultSet*. En el siguiente ejemplo (figura 1.18) se propone obtener toda la información contenida en una tabla llamada 'cliente' y almacenarla en un objeto de la clase *ResultSet* llamada '*rs*':

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

Figura 1.18: Captura de resultados en el objeto ResultSet

- **Procesamiento de resultados y cierre de conexiones.**

Para poder recuperar los resultados del objeto *ResultSet*, se puede obtener apoyo de métodos propios de esta clase y elementos básicos de lenguaje Java que aplicados al objeto, pueden recuperar la información obtenida directamente de la base de datos. Una vez hecho este proceso, no queda nada más que cerrar todas las conexiones creadas, tanto a la base de datos como las referentes a las instrucciones ejecutadas.

1.4.3: EJB (Enterprise Java Beans) - Desarrollo basado en componentes.

La tecnología *J2EE Enterprise Java Beans* ha marcado una nueva tendencia en la forma de desarrollar y rediseñar los sistemas de información, ya que es posible efectuar el desarrollo de componentes que posteriormente se podrán reutilizar y ensamblar en distintas aplicaciones que se tengan que realizar en cualquier empresa.

Esta tecnología promete un paso importante en el camino de la programación orientada a objetos: antes era posible reutilizar clases, pero con el desarrollo de componentes es posible reutilizar '*n*' mayor nivel de funcionalidades e incluso adaptarlas a cada entorno de trabajo particular sin tocar el código del componente desarrollado.

Para poder desarrollar aplicaciones en EJB, se requiere contar con 2 componentes fundamentales:

1. Contenedor EJB.
2. Enterprise Beans (EJB´s)

Un ejemplo cotidiano de su aplicación sería el siguiente: Se podría desarrollar un 'bean' llamado Cliente que represente un cliente en una base de datos. Posteriormente este mismo 'bean' podría ser utilizado en aplicaciones de comercio electrónico o prácticamente en cualquier programa en el que se necesite representar un cliente. Incluso sería posible que el desarrollador del bean y el ensamblador de la aplicación no fueran la misma persona, o ni siquiera trabajaran en la misma empresa.

1.4.3.1: Servicios proporcionados por el contenedor EJB.

La diferencia fundamental entre los componentes y los objetos clásicos reside en que los componentes *viven* en un contenedor EJB que los envuelve proporcionando una capa de servicios añadidos. Algunos de los servicios más importantes son los siguientes:

- Manejo de transacciones: Apertura y cierre de transacciones asociadas a las llamadas a los métodos del bean.
- Seguridad: Comprobación de permisos de acceso a los métodos del bean.
- Concurrencia: Llamada simultánea a un mismo bean desde múltiples clientes.
- Servicios de red: Comunicación entre el cliente y el bean en máquinas distintas.
- Persistencia: Sincronización entre los datos del bean y tablas de una base de datos.
- Gestión de mensajes: Manejo del Servicio de Mensajería Java (JMS).
- Escalabilidad: Posibilidad de constituir clústeres de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.

En un sistema de información que utiliza Enterprise Java Beans, mientras más servicios puedan ser incorporados, podrá ser más factible el hecho de que cumple con características más verificables sobre integridad y funcionalidad.

1.4.3.2: Modelo de Comunicación RMI y su importancia en el desarrollo de EJB.

RMI (Remote Method Invocation, Invocación de Métodos Remotos) define la forma de comunicación remota entre objetos Java situados en máquinas virtuales distintas. Suponiendo que un objeto cliente quiere hacer una petición a un objeto remoto situado en otra JVM (Máquina Virtual Java, *Java Virtual Machine*), RMI pretende hacer transparente la presencia de la red, de forma que cuando se escriba el código de los objetos clientes y remotos **no se tenga que tratar** con la complicación de gestionar la comunicación física por la red.

Para ello, RMI proporciona al cliente un objeto tipo 'proxy' (llamado *stub*) que recibe la petición del objeto cliente y la transforma en algo que se puede enviar por la red hasta el objeto remoto. En el lado del servidor, un objeto similar (llamado *skeleton*) recibe la comunicación, la desempaqueta y lanza el método correspondiente del objeto remoto al que sirve.

Al igual que la petición, se deben empaquetar los argumentos de la llamada. El programador sólo debe definir el código del método en el objeto remoto.

Los objetos *stub* y *skeleton* los construye el compilador de RMI de forma automática. Para implementar y usar una clase remota con RMI se debe cumplir las siguientes condiciones:

- Se debe definir la clase remota como una interfaz que hereda de la interfaz *java.rmi.Remote*. Todos los métodos de esta interfaz deben declarar la excepción *RemoteException.*
- Se debe definir una clase que implemente la interfaz.
- Se debe llamar al compilador *rmic* para que cree las clases *stub* y *skeleton*
- Un servidor debe crear uno o más objetos remotos y asignarles un nombre a cada uno.
- Algún cliente debe localizar un objeto remoto, referenciando su nombre, obtener el *stub* (que implementa la clase remota) y realizar las llamadas al *stub*.

Es posible que en el servidor no exista un objeto *skeleton* por cada objeto remoto sino que, para hacer la arquitectura más eficiente, se pueda definir un objeto genérico que distribuya las peticiones a los objetos remotos con algún mecanismo de identificación de la petición y de caché de objetos remotos.

1.4.3.3: Roles en el desarrollo de EJB´s.

Cuando se desarrollan aplicaciones en EJB´s es importante comprender que se siguen una serie de roles para garantizar que se está implementando todo lo necesario para su funcionamiento. Estos roles se explican brevemente a continuación:

1.- Enterprise Bean Provider: Es el encargado de implementar la lógica de negocio de la aplicación. Desarrolla las clases e implementa las interfaces Java que necesita el Enterprise Bean, las cuales contienen los métodos de negocio. Construye el descriptor de despliegue. Finalmente todas las clases e interfaces que construyó, las empaqueta junto con el descriptor de despliegue en un archivo JAR, llamado ejb-jar.

2.- Application Assembler: Se encarga de ensamblar la aplicación empresarial, uniendo todos los componentes que puede recibir. También modifica algunos datos de configuración en el descriptor de despliegue ejb-jar.xml, y algunas veces en el web.xml de una aplicación web.

3.- Bean Deployer: Obtiene los Enterprise bean en la forma de un archivo ejb-jar del *application assembler* o del *bean provider* y los despliega en el contenedor EJB. El *bean deployer* es por lo general un experto en el manejo del servidor de aplicaciones que contiene al contenedor EJB.

4.- EJB server provider: Provee el ambiente en el cual el contenedor se ejecuta. Por ejemplo, es posible que se encargue del manejo de procesamiento múltiple y manejo de carga.

5.- EJB Container Provider: Su responsabilidad es implementar el soporte necesario que los Enterprise Beans requieren en tiempo de ejecución, además de proveer servicios tales como gestión de seguridad, de transacción y mantenimiento del pool de los beans.

6.- System Administrator: Será el responsable de la configuración y administración a nivel de red, para que el contenedor EJB siga operando a pesar de que diferentes servicios de red se encuentren ejecutándose dentro y fuera del contexto del contenedor.

1.4.3.4: Tipos de beans.

La tecnología EJB define tres tipos de beans: beans de entidad, beans dirigidos por mensajes y beans de sesión.

Los **beans de entidad** representan un objeto concreto que tiene existencia en alguna base de datos. Una instancia de un bean de entidad representa una fila en una tabla de la base de datos. Se compone de una clase que implementa la interfaz EntityBean y de dos interfaces: *home* y *component*.

Los **beans dirigidos por mensajes** pueden escuchar mensajes de un servicio de mensajes **JMS**. JMS es una API que puede ser usada en la plataforma J2EE para el servicio de mensajería en aplicaciones en empresariales. Un cliente nunca puede llamar de forma directa a un bean, sino que es necesario enviar un mensaje JMS para poder comunicarse con éste. La comunicación entre los componentes puede ser de dos tipos:

a) **Síncrona**: Ambas partes que se están comunicando tienen que estar presentes al mismo tiempo. Por ejemplo, una conversación telefónica, donde las dos personas tienen que estar presentas para entablar una comunicación.

b) **Asíncrona**: En la comunicación asíncrona no es necesario que las partes involucradas en la comunicación estén presentes al mismo tiempo. El caso de enviar un correo a una persona en otro país es un caso de comunicación asíncrona.

Por último, un **bean de sesión** representa un proceso o una acción de negocio. Normalmente, cualquier llamada a un servicio del servidor debería comenzar con una llamada a un bean de sesión. Los beans de sesión se dividen en 2 tipos:

a) **Statefull Session Bean**: Este tipo de bean de sesión es utilizado para mantener un estado conversacional con el cliente, ya que puede guardar determinados datos que un cliente pasó en una de las llamadas a un método. Representan un proceso en la aplicación como la implementación de un carrito de compras.

b) **Stateless Session Bean**: Al contrario de un *statefull session bean,* no puede mantener un estado conversacional con el cliente, de modo que no puede recordar información específica de un cliente. Por lo general los *stateless session bean* se utilizan para realizar cálculos, como calcular el descuento a una compra de un determinado cliente.

Mientras que un bean de entidad representa una cosa o entidad que se puede representar con un nombre, al pensar en un bean de sesión se debería pensar en una acción que se debe ejecutar. Ejemplos de beans de sesión podrían ser un carrito de la compra de una aplicación de negocio electrónico o un sistema verificador de tarjetas de crédito.

1.5: Servlets y Java Server Pages (JSP).

1.5.1: Generalidades acerca de los servlets.

Las aplicaciones CGI (Interfaces Comunes de Compuerta) fueron una de las primeras maneras prácticas de crear contenido dinámico en las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. Este puede estar hecho en cualquier lenguaje que soporte el servidor, aunque por razones de portabilidad se suelen usar lenguajes de script. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional.

CGI ha hecho posible la implementación de funciones nuevas y variadas en las páginas web, de tal manera que esta interfaz rápidamente se volvió un estándar, siendo implementada en todo tipo de servidores web. Los *servlets* son la respuesta de la tecnología basada en Java a la programación de CGI´s. Son programas que se ejecutan en un servidor Web, que fungen como una capa intermedia entre una petición proveniente de un navegador Web u otro cliente HTTP, y las bases de datos o aplicaciones del servidor HTTP. Su finalidad básica se puede describir en los siguientes puntos:

1. *Leer los datos enviados por el usuario*: Estos datos, por lo general son introducidos en un formulario o en una página Web, pero también podrían provenir de algún subprograma Java, como los Applets, o programas clientes HTTP propios.

2. *Buscar información respecto a las peticiones que estén incrustadas en la petición HTTP:* Esta información incluye los detalles respecto a las facultades del navegador, cookies, nombre del equipo, entre otros.
3. *Generar los resultados.*
4. *Dar formato a los resultados en base a un formato definido.*
5. *Establecer los parámetros de respuesta HTTP adecuados:* Se puede ocupar de indicarle al navegador el tipo de documento que será devuelto, establecer las cookies y ocultar los parámetros establecidos.
6. *Devolver el documento al cliente:* Ya sea en un formato binario, de texto HTML o incluso comprimido.

Entre las ventajas que se pueden encontrar acerca de los servlets sobre los CGI tradicionales, se pueden mencionar las siguientes:

- *Eficiencia*: Con los servlets, la Máquina Virtual de Java permanece en ejecución y administra 'n' peticiones mediante un ligero subproceso de Java.
- *Poder*: Los servlets pueden comunicarse directamente con el servidor Web, algo que los CGI estándar no pueden realizar por sí mismos, si no se cuenta con la API especifíca del servidor.
- *Transportabilidad*: Los servlets están escritos en lenguaje Java y siguen una API estándar. De hecho, los servlets pueden utilizarse de manera directa o como complementos en virtualmente cualquier servidor web principal. Ahora son parte de la plataforma Java 2 Enterprise Edition (J2EE) y es la razón por la que la industria está teniendo mayor interés en los servlets.
- *Económicos*: Existen disponibles diversos servidores Web gratuitos o muy económicos que son adecuados para su uso "personal" o para sitios Web de bajo volumen. Esto contrasta con muchas otras de las alternativas de CGI, que requieren una significativa inversión inicial para adquirir un paquete propietario.

1.5.2: La interfaz *Servlet* y el ciclo de vida de un servlet.

Desde el punto de vista arquitectónico, todos los servlets deben implementar a la interfaz *Servlet*. Los paquetes de servlets definen dos clases del tipo *abstract* que implementan a la interfaz *Servlet*: la clase ***GenericServlet*** (incluida en el paquete *javax.servlet*) y la clase ***HttpServlet*** (incluida en el paquete *javax.servlet.http*). Estas clases proporcionan implementaciones predeterminadas de todos los métodos de la clase *Servlet*.

Los métodos de la interfaz *Servlet* son invocados por el contenedor de servlets. Esta interfaz define básicamente 5 métodos que conforman en esencia, el ciclo de vida de un servlet. Estos métodos se describen a continuación:

- **void init (ServletConfig config).** El contenedor de servlets llama a este método una vez durante el ciclo de ejecución de un servlet, para inicializarlo. Se utiliza para establecer valores de inicio de ejecución única.

- **ServletConfig getServletConfig ( ).** Este método devuelve una referencia a un objeto que implementa a la interfaz *ServletConfig*. Proporciona el acceso a la información de configuración del servlet, como sus parámetros de inicialización y el acceso a su entorno (es decir, al contenedor de servlets en el cual se ejecuta este servlet).

- **void service (ServletRequest petición, ServletResponse respuesta).** El contenedor de servlets llama a este método para responder a la petición que un cliente hace a un servlet.

- **doGet (HttpServletRequest request, HttpServletResponse response) - doPost (HttpServletRequest request, HttpServletResponse response).** Una petición ***get*** recupera la información de un servidor (documentos HTML o imágenes comúnmente). Una petición ***post*** es utilizada comúnmente para enviar información, como la autenticación de un usuario para entradas al servidor. Los métodos **doGet**

y **doPost** se llaman mediante el método **service**. Este determina primero el tipo de petición y después llama al método apropiado para manejarla.

- **void destroy ( ).** Este método se llama cuando un servlet es terminado por su contenedor de servlets. Los recursos utilizados por el servlet, tales como una conexión a base de datos, o un archivo abierto se finalizan en este método.

1.5.3: La interfaz HttpServletRequest.

Los objetos que implementa a esta interfaz contienen la información de petición del cliente (incluyendo encabezados e información de la línea principal de petición). Algunos métodos clave para permitir que el servlet procese estas peticiones se pueden ver a continuación:

* **String getParameter (String nombre)** – Obtiene el valor de un parámetro enviado al servlet como parte de una petición *get* o *post*. El argumento representa el nombre del parámetro al cual se hace referencia.

* **Enumeration getParameterNames ( )** – Devuelve una lista enumerada de los nombres de todos los parámetros enviados al servlet. Normalmente es utilizado para efectos de depuración.

*** String [] getParameterValues (String nombre)** – Para un parámetro con múltiples valores, este método devuelve un arreglo de cadenas que contiene los valores para un parámetro de servlet especificado.

* **Cookie [] getCookies ( )** – Devuelve un arreglo de objetos Cookie almacenados en el cliente por el servidor. Estos objetos son utilizados comúnmente para identificar clientes en forma única en el servidor.

* **String getHeader (String nombre), Enumeration getHeaderNames ( )** – El primero devuelve la información contenida en la cabecera dependiendo de la referencia mencionada en el atributo. El segundo obtiene una enumeración de todos los nombres de encabezados recibidos en esta petición en particular.

* **String getRequestURI ( )** – Devuelve la parte del URL que viene después de la mención del host y puerto, pero antes de los datos del formulario.

* **String getMethod ( )** – Devuelve explícitamente el método de la petición principal (GET o POST).

1.5.4: La interfaz HttpServletResponse.

El servidor Web que ejecuta el servlet crea un objeto *HttpServletResponse* que es enviado al método *service* y a su vez, al método *doGet* o *doPost*. Algunos de los métodos más importantes que permiten formular la respuesta al cliente se exponen a continuación:

* **void setHeader (String header, String valor)** – Este método establece el valor de una cabecera definida en el primer parámetro de los atributos.

* **void addCookie (Cookie cookie)** – Se utiliza para agregar un objeto Cookie al encabezado de la respuesta al cliente.

* **PrintWriter getWriter( )** – Obtiene un flujo de salida basado en caracteres para enviar datos de texto al cliente.

* **void sendRedirect (String URL)** – Devuelve un una cabecera del tipo *Location* que contiene el URL del nuevo documento (redireccionamiento).

* **void sendError (int código, String mensaje)** – Envía un código de estado (normalmente el codigo 404 – Not Found) junto con un pequeño mensaje que es automáticamente formateado dentro de un documento HTML y enviado al cliente.

1.5.5: Manejo de Sesiones.

Se dice que HTTP es un protocolo 'sin estado', ya que cada vez que un cliente obtiene una página Web, abre una conexión por separado al servidor Web, el cual no conserva información contextual respecto al cliente. Esta carencia conlleva diversos problemas, por ejemplo, cuando los clientes de una tienda virtual agregan un elemento a su 'carrito de supermercado'. ¿Cómo podría determinar el servidor que elementos están contenidos en el carrito, y si ya habían sido elegidos con anterioridad en caso de insertar nuevos elementos?

Inicialmente, las soluciones típicas a estos problemas contemplaban el manejo de cookies y el manejo de campos ocultos de un formulario. Sin embargo, los servlets otorgan una excelente solución técnica: la *API HttpSession*. El uso de sesiones trae consigo el buscar al objeto *HttpSession* cuando sea necesario, buscar la información asociada con una sesión, almacenar información y descartar las finalizadas o abandonadas.

Para buscar si un objeto *HttpSession* existe, se realiza la ejecución del método **getSession( )**, incluido en la interfaz HttpServletRequest. Si el método **getSession** responde con *null*, significa que el usuario ya no está participando en una sesión, por lo que tendrá la oportunidad de generar una nueva. Para que automáticamente la genere, se le puede pasar al método el valor 'true' como argumento.

Los objetos de sesión cuentan con una estructura de datos incorporada que le permite almacenar cualquier cantidad de claves y sus valores correspondientes. Para poder buscar cierta información asociada con una sesión, se utiliza el método **getAtribute**. Este método funciona asignando como atributo el nombre de la clave la cual se desea obtener su valor, mismo que será devuelto en forma de un objeto genérico (de la clase *Object*).

En contraparte, para asignar un valor específico a una clave en particular se utiliza el método **setAtribute**, que emplea 2 atributos para su funcionamiento: el primero se refiere al nombre de la clave a introducir en la sesión, y el segundo es el valor asignado a dicha clave. Las sesiones se harán inactivas cuando explícitamente se utilice el método invalidate ( ) en la sesión.

1.5.6: Java Server Pages (JSP) – Generalidades.

La tecnología Java Server Pages (JSP) es una extensión a la tecnología de los servlets. Permiten a los programadores de aplicaciones Web mezclar contenidos en lenguaje HTML regular y estático con un contenido dinámicamente generado a partir de los servlets. Cuenta con varias ventajas respecto a las alternativas existentes en el mercado. Algunas de ellas se pueden visualizar a continuación:

- *A comparación de ASP (Active Server Pages):*

  Esta es una tecnología propiedad de Microsoft. Las ventajas de JSP se aprecian en la utilización de Java para escribir la parte dinámica y en la transportabilidad que se puede manejar en cuanto a plataformas y sistemas operativos. Con ASP tendríamos que aprender otros lenguajes, como VBScript, y estar restringidos a implementar en sistemas Windows.

- *A comparación de los servlets:*

  Toda funcionalidad que ofrece un JSP es perfectamente asimilable por un servlet. Sin embargo, es más conveniente escribir y modificar código de tipo HTML que tener que introducir una enorme cantidad de instrucciones Java del tipo '*out.println*' para mostrar su equivalente. Además, con los JSP se pueden separar las tareas de los programadores especialistas: mientras los especialistas en el diseño de páginas Web pueden desarrollar en HTML, los especialistas en servlets pueden encargarse de insertar el contenido dinámico.

Los programadores de un JSP pueden, además de las ventajas citadas, reutilizar Java Beans y crear bibliotecas de marcas personalizadas que encapsulen una funcionalidad compleja y dinámica que permitan, incluso a programadores Web que no estén familiarizados con Java, mejorarlas en cuanto a procesamiento y contenidos dinámicos.

1.5.7: Componentes Clave en el desarrollo de JSP.

1.5.7.1: Directivas.

Una directiva JSP afecta la estructura general del servlet que resulta de la página JSP. Son mensajes para el contenedor JSP que permiten al programador especificar configuraciones de página (tales como páginas de error), incluir contenido de otros recursos y especificar bibliotecas de marcas personalizadas para usarlas en una JSP.

Las directivas son delimitadas por la siguiente sintaxis:

```
<% directive   attribute1="value1"
               attribute2="value2"
               ...
               attributeN="valueN"  %>
```

Las directivas son procesadas en tiempo de traducción, por lo tanto, no producen ningún tipo de salida inmediata, ya que se procesan antes de que las JSP acepten cualquier petición. Existen 3 tipos de directivas, las cuales se mencionan a continuación:

- **Directiva page:** Especifica las configuraciones globales de la JSP en el contenedor de JSP´s. Gracias a los atributos que esta directiva tiene, se pueden controlar entre

otras cosas: la estructura del servlet mediante la importación de clases (atributo **import**), la personalización de la superclase del servlet (atributo **extends**), el establecimiento del encabezado de respuesta, o tipo MIME del documento que será entregado al cliente (atributo **contentType**), la determinación de que la página actual sea una página de error (atributo **isErrorPage**), si la página participa en sesiones (atributo **session**), entre otras.

- **Directiva include:** Esta directiva incluye el contenido de otro recurso una vez, en tiempo de traducción del JSP. Sólo tiene un atributo (*file*) el cual especifica el URL del recurso a incluir. Presenta el inconveniente de que si el archivo incluido cambia, todos los archivos JSP que lo utilizan deben ser actualizados.

- **Directiva taglib:** Esta directiva permite a los programadores definir nuevas marcas en forma de bibliotecas, las cuales pueden utilizarse para encapsular la funcionalidad y simplificar la codificación en un JSP.

1.5.7.2: Elementos para secuencias de comandos.

Estos elementos permiten insertar código en el servlet que será generado a partir del JSP. Existen 3 formas de representación de estas secuencias de comandos:

- **Scriplets**. Son bloques de código que contienen instrucciones Java y que el contenedor coloca en el método **service** del servlet generado. Se encuentran delimitados por los caracteres:

<% code %>;

- **Expresiones**. Contienen una expresión de Java que se evalúa cuando un cliente solicita acceso al JSP que contiene la expresión. El contenedor evalúa la expresión,

la convierte a un objeto de la clase String (cadena) y la inserta en la página como parte de la respuesta al cliente. Su sintaxis es la siguiente:

< %= expresión % >;

- **Declaraciones**: Permiten definir las variables y métodos a utilizar en una JSP. Las variables se convierten en variables de instancia de la clase del servlet que representa a la JSP traducida. Su sintaxis básica es la siguiente:

< %! código % >

No generan ningún resultado, por lo que se utilizan normalmente en conjunto con expresiones JSP y scriplets.

1.5.7.3: Acciones Estándar.

Las acciones estándar proporcionan al programador el acceso a varias de las tareas más comunes que se realizan en un JSP, como incluir el contenido de otros recursos, reenviar peticiones hacia otros recursos e interactuar con elementos de tipo JavaBeans. Son procesados en tiempo de petición por parte de los contenedores de JSP, y contienen la siguiente sintaxis basada en XML:

< prefijo:elemento { atributo=valor} * />

A continuación se muestra un resumen de las acciones estándar más importantes empleadas en el desarrollo de JSP´s:

- **<jsp:include>:** Incluye en forma dinámica otro recurso en el JSP. A diferencia de la directiva *incluye*, esta se ejecuta en tiempo de petición, y no en tiempo de

traducción, por lo que no se necesita volver a compilar el JSP en caso de que el recurso incluido haya cambiado.

- **<jsp:forward>:** Reenvía el procesamiento de la petición hacia otro JSP**,** servlet o página estática**.** Esta acción termina la ejecución del JSP actual.

- **<jsp:plugin>:** Permite agregar componentes complementarios, o subprogramas a la página JSP, en forma de elemento *object* (Internet Explorer) o *embed* (Netscape) de HTML específico para cada navegador.

- **<jsp:param>:** Es utilizado como parte de las acciones *incluye, forward y plugin* para especificar pares de nombres y valores adicionales de información para que sean utilizados por estas acciones.

Acciones para manipulación de JavaBeans.

Los JavaBeans son componentes de software escritos 100% en Java y que trabajan frecuentemente con JSP/Servlets para pasar o proveer información con algún alcance. Dentro de sus características es posible acceder a sus valores (atributos) por medio de métodos especiales (*get* y *set*). El objetivo del uso de JavaBeans en la tecnología JSP es obtener un código más simple y entendible, y por lo tanto más fácil de mantener. JSP provee 3 acciones estándar para manipular JavaBeans:

- **<jsp:useBean>:** Permite a un JSP manipular un objeto (bean) de Java, ya sea creándolo o localizando uno existente para utilizarlo en el JSP. Dentro de la acción es importante ajustar parámetros importantes como lo son el nombre de la clase que derivará el objeto creado, el nombre de la referencia al objeto creado y el alcance que tendrá este objeto dentro de la aplicación.

- **<jsp:setProperty>:** Establece una propiedad en la instancia del JavaBean especificada por la acción *useBean*. Relaciona los parámetros de la petición con las propiedades del bean con el mismo nombre de forma automática.

- **<jsp:getProperty>:** Obtiene una propiedad en la instancia de JavaBean especificada y convierte el resultado en una cadena para enviarla en la respuesta.

1.5.8: JSTL (JSP Standard Tag Library).

La librería JSTL es un componente dentro de la especificación del Java 2 Enterprise Edition (J2EE) y es controlada por Sun MicroSystems. JSTL no es más que un conjunto de librerías de etiquetas simples y estándares que encapsulan la funcionalidad principal que es usada comúnmente para escribir páginas JSP. Las etiquetas JSTL están organizadas en 4 librerías:

- ***http://java.sun.com/jstl/core***: Comprende las funciones script básicas como ciclos (loops), condicionales, y entrada/salida.
- ***http://java.sun.com/jstl/xml***: Comprende el procesamiento y validaciones de documentos XML.
- ***http://java.sun.com/jstl/fmt***: Comprende la internacionalización y formato de valores como de moneda y fechas.
- ***http://java.sun.com/jstl/sql****:* Comprende el acceso a base de datos.

La gran ventaja de JSTL es su simplicidad, por lo que JSTL sólo requiere conocimientos rudimentarios de Java, JSP, y aplicaciones Web. Cualquier desarrollador puede comenzar a usarlo de forma casi inmediata.

Para poder utilizar estas librerías se deben descargar de los siguientes sitios e incorporarlas al proyecto de la aplicación:

Biblioteca jstl.jar: http://jakarta.apache.org/site/downloads/index.html

Biblioteca estándar.jar: http://jakarta.apache.org/site/downloads/downloads_taglibs-standard.cgi

1.6: Struts

1.6.1: Generalidades del Framework Struts.

Un *Framework* es un conjunto de clases e interfaces que cooperan para solucionar un tipo especifico de problema de software. Entre las ventajas de utilizar un framework se pueden encontrar las siguientes:

- **Estructura:** Se tiene una estructura de directorios y archivos generales.
- **Ahorro de trabajo:** Se pueden añadir nuevas funcionalidades a las aplicaciones.
- **Consistencia:** La estructura que provee el Framework es igual a cada parte de tu aplicación.
- **Escribir mejor código:** Como el Framework posee cierto orden, el programador debe seguir ese mismo orden en su aplicación.
- **Utilizar lo último en herramientas técnicas:** Con sólo actualizar el Framework y añadir actualizaciones se pueden ocupar nuevas tecnologías en las aplicaciones.
- **Programadores actualizando el Framework:** Existe gente que constantemente actualiza y revisa el Framework.

Struts es un Framework de código abierto que fue creado para permitir que los desarrolladores pudieran crear aplicaciones Web basándose en la lógica de Servlets y JSP´s. Fue creado por Craig R. McClanahan y donado a la Apache Software Foundation (ASF) en el año 2000.

1.6.2: Modelo Vista Controlador (MVC).

En esencia, Struts es un Servlet que está configurado para responder una serie de peticiones y trabaja en base al Modelo Vista Controlador (MVC). A través del MVC es posible hacer las siguientes divisiones:

- **Modelo**: Se centra con las funcionalidades relacionadas con el modelo de datos, que es el acceso y manipulación de los depósitos informativos que pueden ser una Base de Datos y/o archivos, así como sus reglas de negocio.
- **Vista**: Se toca el aspecto visual (gráfico) que será empleado en la aplicación en cuestión, se crea un formato para interactuar creando la interfaz de usuario.
- **Controlador**: Se encarga de coordinar las acciones que son llevadas entre Vista y Modelo, este responderá a los eventos y usualmente a las acciones del usuario ocasionando cambios en el Modelo y posiblemente en la Vista.

Conforme se incrementan las necesidades de modificar una aplicación, se hace inminente la posibilidad de modificar el código que se tiene, y si no existe una clara visión de cómo está estructurado el código, se puede llegar al punto en que éste se torne impredecible debido a una serie de funcionalidades que pueden surgir en un momento dado. Este esquema permite mantener cierto orden al momento de administrar la aplicación.

1.6.3: Esquema básico del Framework Struts.

1.6.3.1: La clase Action.

Al invocar una acción desde una página Web (un archivo JSP en Java), el servlet accede a su configuración según la acción realizada (contenida en el archivo *struts-config.xml*). Posteriormente redirige el control a una acción en particular (clase específica) que sabe como ejecutar la operación de negocio asociada con la acción solicitada.

Paralelamente se realiza la llamada al controlador para que este disponga a crear y devolver la Vista correspondiente, y si es necesario, acceder al Modelo para mostrar sus datos. Todo este flujo se puede visualizar más fácilmente en la figura 1.20.

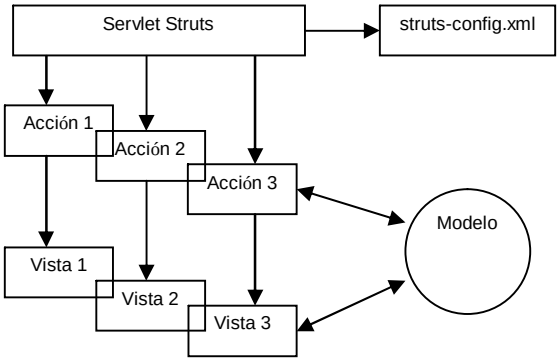


Figura 1.20: Flujo de procesos general en el Framework de Struts.

La clase **Action** de Struts es la clase madre de las acciones que se realizan en la aplicación, porque al recibir un pedido HTTP a través de una interfaz Web (comúnmente generada por un elemento como un botón), Struts lo recibe y lo envía al elemento Action específico correspondiente. Esta clase dispone del método **execute** (), que es invocado por el controlador cuando recibe un petición de un cliente. El controlador sabe que instancia (o subclase) Action debe invocar cuando se realiza una petición, ya que es capaz de inspeccionar la información y utilizar un conjunto de mapeados de acción (**Action Mappings**) encontrados en el archivo de configuración *struts-config.xml.*

En el archivo de configuración *struts-config.xml* no sólo se puede ver a que instancia **Action** se debe invocar ante determinadas acciones. También se pueden dar de alta procesos fundamentales, tales como la declaración de un Bean determinado que será asociado a la aplicación para el manejo de los datos, su ámbito de aplicación y la decisión de realizar validaciones acerca de la información contenida, la URL solicitada por el cliente, así como la declaración de redireccionamiento dependiente a la respuesta que pueda generar la clase **Action** específica.

El análisis del código en el archivo de configuración se muestra gráficamente en la figura 1.21. En este ejemplo se muestra la declaración de una acción con los

parámetros (y su respectivo significado) que normalmente son utilizados en una aplicación de Struts.

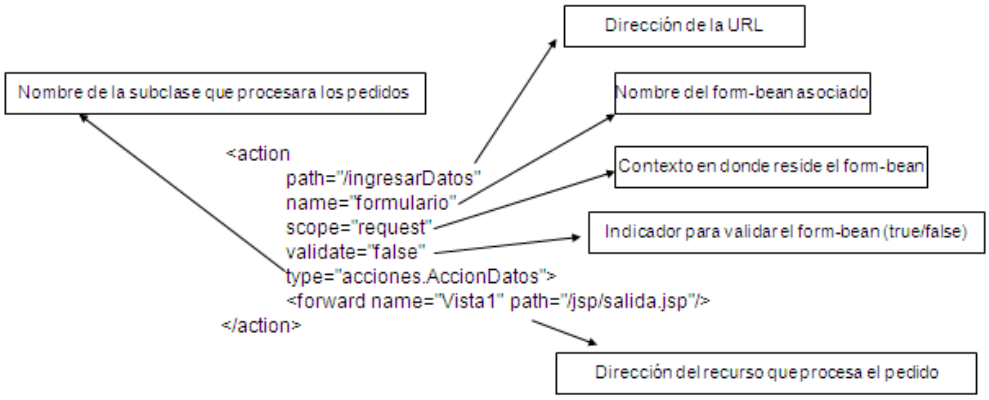


Figura 1.21: Declaración de una acción en el archivo de configuración *struts-config.xml.*

1.6.3.2. Formularios.

Para poder interactuar con una aplicación, se sabe que el usuario debe tener la necesidad de ingresar datos, por lo que se utiliza un formulario para dicho fin. Los objetos **ActionForm** de Struts se utilizan para pasar los datos de entrada del cliente a la lógica del negocio. El Framework será encargado de recopilar automáticamente la entrada de datos de la petición y después los pasa a un elemento **Action** utilizando un *form-bean*, que luego puede pasar a la lógica de negocio.

En los objetos **ActionForm** se ha añadido el método **reset** () para el restablecimiento de las propiedades booleanas de los datos en sus valores por defecto. Entre las ventajas de utilizar un Action Bean se pueden mencionar las siguientes:

- Almacena los datos temporalmente.
- Permite validaciones de los datos.

- Brindan consistencia en los formularios tipo HTML.

- Se pueden compartir para múltiples formularios dentro de la misma aplicación.

Sin embargo, pueden existir desventajas evidentes, como tener muchos **ActionForm** a medida que la aplicación crece, o si se tiene un **ActionForm** para toda la aplicación Web, y a su vez varios programadores, al hacerse un cambio en **ActionForm** afectaría el trabajo de los demás. Es por estas razones que existen variantes denominadas **Action Form** *dinámicas* (o **DynaActionForm**)**.** Un **ActionForm** *dinámica* se implementa por medio de la clase base *org.apache.struts.action.DynaActionForm*. Sus propiedades se configuran totalmente en el archivo ***struts-config.xml***.

1.6.3.3. Validaciones.

Cuando un usuario llena un formulario se envían los datos al servidor para que éste los procese (o valide). Si se percata que uno de los campos está mal, es lógico pensar que el servidor debe devolver el mensaje de error y en algunos casos el formulario en blanco para que el usuario tenga que completar los datos solicitados.

Cuando se están usando **ActionForms** y resulta importante para la aplicación hacer que la información ingresada sea válida, se cuenta con un método especial: **validate** (). Los casos de validación específicos pueden estar complementados dentro de este método, tales como si se exige que algunos datos sean requeridos y el usuario no puede completarlos, otros tienen un rango determinado que no puede excederse, otros deben pertenecer a cierto dominio, entre muchos otros.

Sin embargo, la versatilidad con la que fue creado el Framework Struts permite al usuario invocar validaciones con configuraciones ya dispuestas por default en el Framework, así como la oportunidad de crear nuestras propias reglas de validación.

El método establecido lleva por nombre **Struts Validator**, y los requerimientos básicos para poder utilizarlo en las aplicaciones son los siguientes:

- Para definir validaciones sobre un formulario, se debe proveerle al **Validator** (método validador) un archivo XML con las reglas de validación. Técnicamente se puede usar el archivo *validator-rules.xml* que viene incluido por default, pero ésta es una práctica no recomendable.

- Lo más viable en este caso es crear un archivo XML por cada formulario. Un ejemplo se puede visualizar en la figura 1.24, donde se validan reglas tales como que el campo sea obligatorio y que tenga una longitud máxima a 2 campos definidos por el usuario.

- El nuevo archivo XML debe ser declarado en el archivo de configuración *struts-config.xml*, como si se tratara de un *plugin*, es decir, un componente que Struts que carga al iniciar y destruye al finalizar su ejecución. Un ejemplo del código insertado en el archivo de configuración *struts-config.xml* se puede ver en la figura 1.25.


```
<?xml version="1.0“ encoding ="UTF-8"?>
<!DOCTYPE form- validation PUBLIC
"-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
"http://jakarta.apache.org/commons /dtds / validator _1_1_3. dtd ">
<form - validation>
   <formset>
      <!-- Agregando el formulario-->
      <form name="form1">
         <!--Validando el primer Campo-->
         <field property="name" depends="required">
            <arg position="0"  key="fo.campo1"/>
         </field>
         <!--Validar el segundo Campo-->
         <field property="number" depends="maxlength">
            <arg position="0" key="fo.campo2"/>
            <arg position="1" name="maxlength" key="${var:maxlength}" resource="false"/>
            <var>
               <var – name >maxlength</var-name>
               <var – value >3</var-value>
            </var>
         </field>
      </form>
   </formset>
</form- validation>
```

Figura 1.24: Ejemplo de un archivo validador (*validation-form1.xml*) con su declaración de campos.

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property
      property="pathnames"
      value="/WEB-INF/validaciones/validator-rules.xml,/WEB-INF/validaciones/ validation-form1. xml"/>
</plug-in>
```

Figura 1.25: Declaración del archivo *validation-form1.xml* en el archivo de configuración.

- Contar con mensajes suficientes en un archivo de Propiedades (también llamado **messages-resources,** y llamado por default *AplicationResource.properties*). Este archivo especifica el paquete de recursos que contiene los mensajes localizados para una aplicación. Esto es útil cuando existe la necesidad de *internacionalizar* la aplicación.

- Crear una clase que contenga las variables que se manejan en el archivo de reglas de navegación XML. Esta clase debe heredar de la clase ***ValidatorForm***.

Como se puede ver, con las facilidades mencionadas para realizar validaciones en Struts, el programador puede verificar sin mucho esfuerzo ciertas variables, tales como fechas, longitudes de cadenas, cantidades monetarias, sintaxis en correos electrónicos y tipos de datos básicos en general.

1.7: JSF (Java Server Faces)

1.7.1: Generalidades.

JSF es un *framework* de desarrollo basado en el patrón MVC (Modelo Vista Controlador) que fue diseñado principalmente para construir interfaces de usuario en aplicaciones Web. La construcción de los interfaces de usuario suelen ser la parte más costosa del esfuerzo de desarrollo, sobre todo por la dificultad de mantenimiento y la mezcla frecuente en un mismo archivo JSP de la interfaz de usuario, las reglas de validación y el acceso a la base de datos.

JSF trata de poner cierto énfasis en la parte de vista (la interfaz del usuario) de una forma más similar al estilo de Swing, donde la programación se hace a través de componentes y basada en eventos. Entre otras características importantes, JSF es capaz de ofrecer lo siguiente:

- Un modelo de trabajo basado en componentes UI (User Interface), definidos por medio de **etiquetas** y código elaborado en **XML**.
- Una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC), con lo que se garantiza tener código más limpio, reutilizable y adecuadamente organizado.
- Asociación (de forma modular) entre cada componente gráfico con los datos (beans de respaldo).
- Posibilidad de realizar validaciones tanto en la parte del cliente como en el servidor.
- Control de mensajes y administración de roles.

Al igual que Struts, JSF pretende normalizar y estandarizar el desarrollo de las aplicaciones Web. No puede competir en madurez con Struts, pero si puede ser una opción muy recomendable para nuevos desarrollos, sobre todo si todavía no se cuenta con experiencia suficiente con el funcionamiento de Struts.

1.7.2: Análisis básico de los componentes en una Aplicación JSF.

Las aplicaciones Web constan de 2 partes importantes: La capa de presentación y la lógica del negocio. La presentación (o también llamada vista) es determinada por elementos HTML tales como etiquetas definidas, imágenes, estilos, entre otros. La lógica del negocio que es implementada en código Java determina el comportamiento de la aplicación.

Sin embargo, algunas tecnologías basadas en Web entremezclan código HTML y código de la aplicación. La idea de producir aplicaciones simples escritas en un solo archivo podría no ser complicada ni presentar mayores dificultades. Para aplicaciones importantes y extensas, esta mezcla de códigos puede llevar a generar problemas severos a largo plazo. JSF trata de evitar este tipo de problemas, en base al desarrollo de cada uno de los componentes que lo forman. Una aplicación simple escrita en JSF puede constar de los siguientes elementos básicos:

a) Páginas que determinan la vista de la aplicación. Normalmente se encargan las páginas JSP de realizar este cometido, utilizando *tags* especiales referentes a JSF.
b) Un bean que administre los datos del usuario. Hay que recordar que un Java Bean es una clase que expone propiedades definidas y se tiene acceso a estas en base a métodos *get* y *set* de lectura y escritura.
c) Un archivo de configuración XML que la aplicación necesita para llamar recursos adicionales y establecer reglas de navegación. En el contexto de JSF, este archivo es llamado *faces-config.xml*.
d) Archivos extra que son requeridos por el contenedor para desplegar la aplicación apropiadamente (*web.xml* para el despliegue y archivos de propiedades con declaración de mensajes).

Las aplicaciones JSF más avanzadas y complejas contienen la misma estructura mencionada; sin embargo, contienen clases Java adicionales que permiten hacer la aplicación más robusta, tales como manejadores de eventos, validadores y componentes personalizados.

1.7.3: Ejemplo de ilustración: Autenticación de un usuario.

A continuación se propone un ejemplo sencillo para poder explicar la operación básica que se sigue dentro de las aplicaciones elaboradas en JSF: la autenticación de un usuario a una aplicación en particular.

Para el desarrollo se deberán contemplar la creación de las vistas que el usuario manejará (tanto de entrada, como de salida), desarrolladas en archivos JSP´s; el código que sigue la regla de navegación (flujo), la declaración y desarrollo del Java Bean a manejar y la elaboración de una clase que sugiere la llamada a una base de datos para la verificación de la información ofrecida por el usuario.

1.7.3.1: Páginas JSP de entrada y salida.

En el contexto del lenguaje Java y de las aplicaciones Web, a través de las páginas JSP es posible realizar la interacción entre una aplicación y el usuario. JSF ofrece un conjunto de *tags* (similares a los que se utilizan en Struts) que son analógicos a los elementos más importantes del lenguaje HTML (Tag HTML) y que permiten otras características importantes, tales como la interacción con el usuario en diferentes plataformas, por ejemplo, el despliegue de la aplicación en una pantalla táctil (Tag Core). Para hacer uso de las librerías donde se hace referencia a estos *tags* deben declararse al principio del JSP, y son las siguientes:

```
<% @taglib prefix="f" uri="http://java.sun.com/jsf/core" %>
<% @taglib prefix="h" uri="http://java.sun.com/jsf/html" %>
```

Volviendo al ejemplo propuesto de aplicación, será necesario crear 2 páginas JSP para cubrir el aspecto visual: la primera, mostrará el formulario que deberá llenar el usuario para autenticarse en el sistema. El segundo, solamente contendrá un mensaje de Bienvenida, en caso de que la información sea correcta. En la figura 1.28 se puede ver el código de la página JSP de entrada. A su lado, se puede encontrar el código fuente de la página en donde se muestran los diferentes tags pertenecientes al entorno de JSF.

La explicación del código fuente es la siguiente: En las líneas 2 y 3 se hace la declaración de las librerías que JSF utilizará para poder manejar sus etiquetas. En la línea 15 se hace uso de una de las etiquetas más importantes dentro del contexto de JSF: la instrucción *<f: view>* permite la posibilidad de hacer que la aplicación sea visible ante diferentes tecnologías desde las cuales se pueda tener acceso a la aplicación. Su ámbito de aplicación se encuentra hasta la línea 25.


```
1
2  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
3  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
4
5  <%@page contentType="text/html" pageEncoding="UTF-8"%>
6  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
7     "http://www.w3.org/TR/html4/loose.dtd">
8
9  <html>
10     <head>
11         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12         <title>Un ejemplo de Java Server Faces</title>
13     </head>
14     <body>
15         <f:view>
16         <h2>Acceso al Sistema<br>
17         Registrate!</h2><br><br>
18         <h:form>
19             Login:<h:inputText value="#{BeanUser.name}"/><br>
20             Password:<h:inputSecret value="#{BeanUser.password}"/><br>
21             <h:commandButton value="Ingresar"
22                              action="#{actionLogin.revisaLogin}"/>
23         <br>
24         </h:form>
25         </f:view>
26     </body>
27 </html>
```

Figura 1.28: Aspecto visual de la página ‘entrada.jsp’ y su código fuente

A partir de la línea 18 y hasta la línea 24 aparece la definición del formulario de la aplicación, separado por los tags especiales <*h:form*>. En la línea 19 y 20 se hace referencia a los campos de entrada y a su respectivo mensaje de texto por los cuales es usuario ingresará la información requerida. El primero de los campos es utilizado para insertar en formato de texto el login de usuario, definido por el tag <*h:inputText*> y el segundo es para ingresar una contraseña, definida por el tag <*h:inputSecret*>.

Es importante notar que la propiedad a la cual está asociada cada uno de estos tags (en este caso BeanUser.name y BeanUser.password) debe estar relacionada con un Java Bean y sus propiedades.

Finalmente, en la línea 21 se encuentra la declaración del botón que procesará la petición del usuario. Está determinado por la etiqueta <*h:commandButton*> y a su vez contiene los parámetros del valor asignado (que funcionará como etiqueta) y la acción asociada a este elemento. Este último realizará una llamada explícita a una clase con cierto método en particular (en este caso, ***actionLogin.revisaLogin***), misma que deberá ser incluida en el archivo de validación *faces-config.xml*.

Tanto este procedimiento, como la declaración del alias 'BeanUser' en los campos de texto se detallarán en el capítulo 1.7.3.3: ***Declaración del Java Bean y de la clase auxiliar.***

1.7.3.2: Regla de navegación.

Una regla de navegación en el contexto de JSF indicará el flujo a seguir entre varias páginas JSP. Todo dependerá de un resultado factible representado en forma de cadena (*String*) que se obtenga a partir de la ejecución de una acción en particular. Los resultados posibles que entregará la clase deberán estar contemplados en la declaración del archivo de configuración *faces-config.xml*. Una vista del código referente en el archivo de configuración y su analogía representada en bloques se puede visualizar en la figura 1.29.

La regla de navegación explicada anteriormente se puede interpretar de la siguiente manera: De las líneas 23 a la 34, se realiza la declaración de la misma a través de la cabecera <navigation-rule>. En la línea 24 se indica el origen de donde proviene la petición

del usuario, señalada por la cabecera <from-view-id>, que en este caso es el archivo *entrada.jsp*.

```
22
23      <navigation-rule>
24          <from-view-id>/entrada.jsp</from-view-id>
25          <navigation-case>
26              <from-outcome>error</from-outcome>
27              <to-view-id>/error.jsp</to-view-id>
28          </navigation-case>
29
30          <navigation-case>
31              <from-outcome>exito</from-outcome>
32              <to-view-id>/welcome.jsp</to-view-id>
33          </navigation-case>
34      </navigation-rule>
35
```

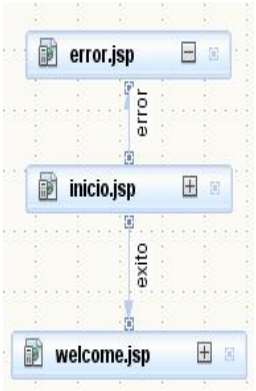


Figura 1.29: Regla de navegación contenida en el archivo *faces-config.xml* y su representación

A partir de la línea 25 se empiezan a declarar los casos de navegación los cuales determinarán el flujo del sistema. El primero de los casos (delimitado entre las líneas 25 y 28) indicará que si el usuario ingreso de forma equivocada sus datos de login y password (determinada por la cadena 'error'), será llevada a una página dedicada que le informará de esta situación (/error.jsp). Si los ha ingresado de forma correcta (líneas 30 a la 33), se le direccionará a una página donde podrá empezar sus labores (/welcome.jsp).

1.7.3.3: Declaración del Java Bean y de la clase auxiliar

Dentro del ámbito de aplicación de JSF, por cada elemento de un formulario, debe existir una propiedad un Java Bean que haga referencia. Para el ejemplo propuesto, la figura 1.30 ejemplifica el código del Java Bean sugerido, creado dentro de un paquete llamado '*beans*'. Nótese que las propiedades coinciden en este caso con las que se declararon en la página JSP de '*entrada.jsp*'.

Una vez creado el Java Bean, será necesario darlo de alta en el archivo de configuración principal de JSF mediante la cabecera <*managed-bean*>. Esta cabecera

incluye 3 parámetros básicos que se deberán complementar para su adecuado funcionamiento: el alias que se asignará para la manipulación del Bean, la clase a la cual se hará referencia y el alcance que tendrá (*request*, *session*, *application* o *none*) dependiendo de la funcionalidad que se le deseé brindar.

```
1
2  package beans;
3
4  public class BeanUser {
5
6     private String name;
7     private String password;
8
9      public BeanUser() {
10     }
11
12     public String getName() {  return name;  }
13
14     public void setName(String name) {  this.name = name;  }
15
16     public String getPassword() {  return password;  }
17
18     public void setPassword(String password) {  this.password = password;  }
19
20 }
```

Figura 1.30: Java Bean con propiedades '*name'* y '*password'* en la clase BeanUser.

En esta cabecera, no sólo se deben dar de alta todos los Java Bean que la aplicación requiera. Por separado, todas las clases Java auxiliares deberán declararse con estas mismas propiedades.

1.7.3.4: Desarrollo de clases auxiliares

El desarrollo de una clase auxiliar no tiene gran relevancia dentro del contexto básico de un JSF. Es una clase Java común que debe cumplir con ciertas características para que sea manipulada adecuadamente por el archivo de configuración *faces-config.xml:*

- Debe retornar como resultado de la función, un objeto de la clase *String*. Este objeto deberá estar mapeado en el archivo faces-config.xml, dentro de las reglas de navegación (<navigation-rule>).

- Debe recuperar la instancia de aplicación del cliente a través de un objeto de la clase ***FacesContext*** y el método *getCurrentInstance()*.
- Debe almacenar la información implícita del usuario en una colección del tipo ***Map***. Se apoyará de los métodos *getExternalContext()* y *getSessionMap()*.
- A partir de aquí, si se necesita traer la información de un Java Bean, se puede crear un objeto que hará referencia a la clase que contiene al Bean, incluyendo el parámetro con el que se le identifica en el archivo de configuración *faces-config.xml*.

El ejemplo de la declaración de esta clase junto con las restricciones mencionadas anteriormente se puede visualizar en la figura 1.32. En esta clase se propone hacer una llamada al método *verificaUser(String, String)*, contenida en una clase llamada *DAOSistema* dentro del paquete *'Class'*, la cual se conectará a la base de datos para realizar el proceso de búsqueda del usuario. Dependiendo del resultado final que se obtenga por parte del método, se dará punto de partida para que el flujo siga su curso retornando ya sea el objeto String "éxito" o "error".


```
1
2  package Actions;
3
4  import Beans.BeanUser;
5  import Class.DAOSistema;
6  import java.util.Map;
7  import javax.faces.context.FacesContext;
8
9  public class ActionLogin {
10
11     public String revisaLogin(){
12         //Regresa la instancia para el cliente en particular...
13         FacesContext context=FacesContext.getCurrentInstance();
14         //Regresa un hash relacionado a un cliente
15         Map sessionMap=context.getExternalContext().getSessionMap();
16         //Se crea un objeto referente al Bean establecido
17         BeanUser bs=(BeanUser)sessionMap.get("BeanUser");
18
19         String name=bs.getName();
20         String password=bs.getPassword();
21         if(DAOSistema.verificaUser(name,password))
22             return "exito";
23         else
24             return "error";
25     }
26 }
```

Figura 1.32: Declaración de una clase auxiliar *ActionLogin* en el paquete *Actions*

Hasta esta parte, se ha abarcado lo más importante desde el punto de vista teórico en cuanto a conocimientos adquiridos en el Diplomado en Java Máster, mas no es lo único que el lenguaje como tal puede ofrecer. Existen muchas áreas de desarrollo que aún no han sido exploradas, y debe ser responsabilidad del programador investigar acerca de estas para emplearlas en las diferentes actividades que involucren su vida diaria. Esto no le quita el valor principal al Diplomado, que es el de perfilar al programador con conocimientos sólidos para quien inicia desde lo más básico del lenguaje.

En el siguiente capítulo, los conocimientos adquiridos tomarán una notable importancia, al ser aplicados dentro del Proyecto de Migración del Sitio de la Red Inalámbrica Universitaria. Es cierto que no todas las ramas vistas anteriormente se utilizan para el desarrollo del proyecto; sin embargo, se recomienda que se tenga cierta práctica en estas para obtener la experiencia necesaria en cuanto a desarrollo de programación.

Capítulo II

Proyecto de migración del sitio de la Red Inalámbrica Universitaria (RIU) en tecnología basada en Java

Capítulo II: Proyecto de migración del sitio de la Red Inalámbrica Universitaria (RIU) en tecnología basada en Java

2.1: Fase de Planificación y Análisis

La Dirección de Telecomunicaciones de la DGSCA desde hace aproximadamente dos años ha llevado a cabo el proyecto de la Red Inalámbrica Universitaria (RIU) con notable éxito, en donde se han logrado acercar a la comunidad universitaria a servicios tales como el Acceso a Internet, siempre y cuando se cuente con un equipo inalámbrico que soporte las especificaciones tecnológicas requeridas y se realice en tiempo y forma el trámite de un login y password de acceso.

Para que un usuario (en su rol de estudiante, académico, personal administrativo o invitado de la UNAM) pueda ser acreedor a una cuenta y contraseña y tener acceso a RIU, se ha dispuesto del sitio https://www.riu.unam.mx, en el cual se puede encontrar tanto información general sobre el servicio, así como la posibilidad de ingresar los datos tanto personales como los del equipo que pretende registrar (sea una laptop, PDA o algún teléfono celular) mediante un formulario único de registro.

Sin embargo, para este propósito, la página en la actualidad presenta inconsistencias que a la larga han entorpecido las actividades del personal encargado de otorgar el servicio (en este caso, la Coordinación del Centro de Atención a Usuarios de la DGSCA), ya que se ofrecen parámetros mínimos de seguridad y el comportamiento en cuanto al flujo que debe seguir la información no llega a ser el óptimo como el que se desearía tener.

Es prioritario que se tenga una solución práctica, sobre todo porque la información personal de los usuarios registrados es muy importante tanto para poder administrarlos de la forma más eficiente, como para evitar en la medida de las posibilidades, que la información sea falsa o errónea y con ello persista la integridad en las bases de datos.

Otra justificación válida sobre la cual hay que considerar la actualización del sistema, se puede percibir en el tiempo promedio de atención a un usuario, ya que el personal del área tarda demasiado en verificar la información dada por los usuarios y tiene que deliberar sobre su veracidad. En caso de presentar errores, tendrá que notificar la necesidad de corregirlos. Todo este proceso de atención puede transcurrir desde los 3-5 minutos como mínimo hasta los 15 minutos únicamente para entregar el comprobante de registro, dependiendo de la cantidad de errores y la saturación de trabajo que se tenga sobre otras diferentes actividades en un momento determinado del día.

La intención es por lo tanto, reducir esta cantidad de tiempo invertido en la activación de cuentas, garantizando con esto un servicio más eficiente hacia los usuarios, así como también utilizar esta ganancia en otras actividades también importantes y sobre todo relacionadas con el servicio, tales como la asesoría técnica personalizada a equipos que presenten problemas de conexión.

Algunos de los problemas que se encuentran en forma habitual en el sistema actual y que será deseable evitar para futuras transacciones son los siguientes:

- No existen algoritmos ni procedimientos de validación importantes en los datos que se solicitan dentro del formulario de inscripción, tales como lo son el CURP, el RFC y las direcciones MAC de los equipos.

- Todos los métodos y código fuente de la aplicación quedan entremezclados con la parte de la vista (código HTML, hojas de estilo y procedimientos JavaScript)

de cada una de las páginas, lo que dificulta realizar procedimientos de mantenimiento.

- El servicio de envío de cuentas por correo electrónico de forma manual es uno de las modalidades que se han tomado en cuenta para beneficio de los universitarios que se encuentran principalmente en la periferia de Ciudad Universitaria (concretamente en las FES). Sin embargo, no existen mecanismos para garantizar que el correo electrónico que sea ingresado sea válido bajo los dominios establecidos para los académicos y estudiantes.

El proyecto de migración del sitio de la Red Inalámbrica Universitaria (RIU) pretende contemplar una solución práctica y expandible a largo plazo que pueda satisfacer la mayoría de los problemas que anteriormente se han descrito. Está dispuesto para realizarse en tecnología Java, donde los conocimientos de dicho lenguaje han sido adquiridos a lo largo del 2do Diplomado en Java Máster, impartido en las instalaciones de la FES Aragón.

2.1.1: Alcances del proyecto.

Dentro del proceso de análisis, es necesario considerar ciertas reglas, o condiciones que se deben de cumplir en la planeación de la lógica del negocio. Algunas de estas reglas ya existían implementadas en el anterior sistema, tales como las que se describen a continuación:

- Debe de existir una distinción clara sobre los tipos de usuario que pueden hacer el registro para obtener su cuenta de RIU. En base a esto, se debe de administrar la serie de cuestionamientos que se harán en la parte del formulario de registro.

- En caso de que el usuario sea un Académico o Investigador de la UNAM, tendrá que verificar previamente si éste cuenta con una contraseña de Acceso a Internet Vía Módem proporcionada por el personal de la DGSCA en forma independiente.

- Si un usuario ya se ha registrado previamente y cuenta con el servicio en forma activa, el sistema debe ser capaz de notificárselo y desplegar un menú con opciones alternas e informativas que son la que probablemente se acerquen a la necesidad real del usuario. Con esto se puede evitar en gran medida la posibilidad de que se registre infinidad de ocasiones y sature la base de datos con información redundante.

Ante esto, es importante reconsiderar algunos procedimientos. Estos pueden incluir una nueva planeación en la lógica de negocio donde se empiecen a contemplar la inclusión de nuevos módulos, cambio en las reglas de navegación, inclusión de parámetros que confirmen la validez en la información y seguridad en las transacciones, por decir algunos. Las novedades que presentará este nuevo sistema en relación con el anterior, se presentan a continuación:

- Se puede poner a disposición de todos los usuarios activos de RIU un módulo donde se pueda comprobar el estado de su cuenta, con el objeto de que pueda enterarse de alguna situación relacionada con ésta: si está activa y no presenta problemas, si próximamente necesitará pasar por el proceso de renovación o si definitivamente el nombre de usuario y contraseña no fueron encontrados en la base de datos.

- Es posible contemplar la inclusión de un filtro de seguridad, que servirá al momento de realizar el registro, con el objeto de que no se pueda ingresar a páginas directamente, a menos que sea la página de bienvenida establecida

independientemente de la cantidad de navegadores web que se estén utilizando en el equipo cliente en ese preciso momento.

El alcance que como objetivo primario se desea cubrir con mayor urgencia en el desarrollo de este proyecto se orienta al replanteamiento de la lógica del negocio para registro de usuarios. Se contemplará que la parte de registro (negociación) se encuentre ubicada en un servidor diferente al que sostiene todo el sitio, ya que el servidor que actualmente sostiene toda la aplicación es compartida con otros servicios, lo que podría repercutir en una lentitud del sistema para realizar transacciones. El aspecto visual (gráfico), no se modificará, tan solo se incorporará al contexto visual establecido (algo que en el sistema actual, no se encuentra).

Otro de los aspectos que no se tiene contemplado modificar con el alcance del nuevo sistema se relaciona con las tablas de la base de datos que maneja el departamento, ya que no sólo soportan la información de los usuarios referentes a la Red Inalámbrica, sino que contienen información de usuarios referente a otros servicios, tales como el Acceso a Internet Vía Modem. Seguirá manejándose la versión de Oracle 10.1.0.2.

Se hará referencia a 2 tablas únicamente: PT6HD_AIVM y PT6HD_WIRELESS_EQUIPO. La primera contiene la información general de los usuarios (Nombre, Nombramiento, Dependencia de Adscripción, CURP, RFC, Correo Electrónico y una serie de banderas que ayudarán a identificar el perfil del usuario). En la segunda se depositará la información de su equipo: Dirección MAC, tipo de equipo, la referencia al usuario en base a su ID y una bandera que identifique a un equipo en particular de otro que probablemente pueda registrar.

La referencia de campos de cada una de las tablas que se manejarán es especificada en las tablas 2.1 y 2.2.

Tabla 2.1: Listado de campos en la tabla PT6HD_AIVM **Nombre de la tabla: PT6HD_AIVM**			
Nombre	**¿Nulo?**	**Tipo**	**Observaciones**
ID	NOT NULL	NUMBER(10)	Calculado por la aplicación
CUENTA		VARCHAR2(15)	
PASSWORD		VARCHAR2(15)	
A_PATERNO		VARCHAR2(50)	
A_MATERNO		VARCHAR2(50)	
NOMBRE		VARCHAR2(50)	
CLAVE_DEP		VARCHAR2(25)	
TICKET_ID		NUMBER(10)	Por default, el valor será nulo
NOMBRAMIENTO		VARCHAR2(200)	
NU_TRABAJADOR		NUMBER(10)	
FECHA_ALTA		DATE	No utilizado
AGREGADO_POR		VARCHAR2(50)	Por default, el valor será nulo
DESCRIPCION		VARCHAR2(2000)	Por default, el valor será nulo
RFC		VARCHAR2(20)	
CURP		VARCHAR2(18)	
CORREO		VARCHAR2(150)	
BANDERA		NUMBER(1)	Valores permitidos: 0,1,2,6,7,8,9
TIPO		NUMBER(1)	Valores permitidos: 1,2,3,4,5,6,null
PLAN		NUMBER(1)	Valores permitidos: 0,1,2,3,9,null
SITUACION		NUMBER(1)	Valores permitidos: 0,null

Tabla 2.2: Listado de campos en la tabla PT6HD_WIRELESS_EQUIPO **Nombre de la tabla: PT6HD_WIRELESS_EQUIPO**			
Nombre	**¿Nulo?**	**Tipo**	**Observaciones**
MAC		VARCHAR2(17)	
MODELO		NUMBER(1)	
ID		NUMBER(10)	Foreign Key (PT6HD_AIVM.ID)
SERIE		NUMBER(7)	Valores: 100, 101, 102

2.2: Fase de Desarrollo

En la realización de esta fase, será necesario considerar las herramientas, equipo de cómputo y referencias técnicas con las que cuentan en la Coordinación en la realización del proyecto. Por parte del Centro de Atención a Usuarios (mismos que desde ahora, los llamaremos usuarios finales), se dispone de los siguientes equipos y software para el desarrollo:

- Equipos Marca Dell con procesador Intel Pentium IV a 3.5 GHZ, 2 GB en memoria RAM, Disco Duro de 200 GB y Tarjeta de Red Ethernet 10/100.

- Manejador de Base de Datos Oracle 10.1.0.2 instaladas en los equipos para desarrollo y pruebas. Esta misma versión se encuentra instalada en los servidores de producción.

Las aplicaciones extra que como mínimo se deberán instalar para poder desarrollar con orientación a tecnología Java son las siguientes:

- Entorno de Desarrollo Integrado (IDE): NetBeans 6.0.1 (descarga gratuita desde http://www.netbeans.org). Incluye por default la última versión de las librerías Java (jdk1.6.0_05, abril del 2008).

- Contenedor Web con soporte para Tecnología Java: Apache Tomcat 6.0.14 (incluido junto con el IDE NetBeans 6.0.1, pero para su implementación en el servidor externo se puede descargar desde http://tomcat.apache.org).

- Driver de conexión para el Manejador de Base de Datos Oracle 10.1.0.2 (el archivo lleva por nombre *ojdbc14.jar*, y se descarga gratuitamente del sitio http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc101040.html previo registro en el sitio).

En base a todas las sugerencias que se han generado en las fases previas (planificación y análisis), y tomando en cuenta los nuevos casos de uso elaborados, es posible dar un panorama abstracto de los componentes que se deberán incluir en el proyecto, los cuales, en la parte de la vista deberán incluir los siguientes elementos:

a) Se creará una página de inicio donde se marcará como punto de partida la negociación con los usuarios, ya sea para registrarse por primera vez (si es que no cuentan con algún antecedente de registro anterior), o conocer el estado de su cuenta si es que ya se encontraban registrados.

b) Debe existir una página donde se pueda evaluar el caso de uso referente a los solicitantes que tengan el requisito de Acceso a Internet Vía Módem por parte de la DGSCA. Se les solicitará la información del nombre de usuario y contraseña respectivos para continuar con el trámite. Aunque esta página ya es funcional en la versión original del sistema, es necesario replicarla con todos los elementos del lenguaje Java en la parte de la vista (mediante cabeceras y etiquetas).

c) Una página donde se despliegue el formulario general de registro. De igual manera ésta ya existe dentro del esquema original, sin embargo, la manera de estructurar el cuestionario deberá ser de lo más sencillo y congruente para el usuario, con el objeto de procurar que no cometa errores al momento de llenarlo. En caso de que exista un problema, las notificaciones o errores desplegadas deberán ser lo más claras posible para que el usuario identifique su problema más fácilmente.

d) La página que despliega el comprobante de registro con todos los datos que el usuario inserto en la base de datos, deberá ser de igual forma duplicada con elementos Java.

e) La página que será de carácter informativo será consultada en el caso de que se llegue a solicitar el estado de la cuenta previamente insertando sus datos de autenticación desde la página de bienvenida, se tendrá la posibilidad de direccionar a sitios de interés mediante ligas, para que el usuario pueda identificar si es que tiene algún problema con su cuenta y procurar darle solución inmediata.

Como se ha mencionado en un principio, toda la lógica de negocio será dispuesta en lenguaje Java, por lo que el código quedará resguardado en clases y métodos estándar, así como en elementos propietarios del Framework, siendo estos últimos el *core* o núcleo de la aplicación. Como pauta para el desarrollo del sistema, se tratará de orientar el desarrollo en base al Modelo-Vista-Controlador, procurando hacer una fuerte distinción entre cada uno de sus componentes.

2.2.1: Definición de la tecnología y creación del proyecto

La solución tecnológica basada en Java será contemplada a realizarse en base al Framework de Struts, en su versión 1.2.9. Sus librerías de desarrollo se encuentran incorporadas junto con el IDE NetBeans 6.0.1. Esta elección se justifica en base a que Struts por naturaleza ofrece una mayor robustez en sus aplicaciones, que lo que en un momento podría ofrecer, por ejemplo, JavaServer Faces (JSF). Ahora bien, si se estuviera limitado a utilizar únicamente páginas JSP y Servlets para el manejo del flujo de navegación y transacciones, el desarrollo podrá ser semejante al que se encuentra en el sistema original, situación que desde las fases de planificación y análisis se ha dispuesto a cambiar por completo.

Normalmente un proyecto basado en JSF es una buena opción cuando existen demasiadas páginas orientadas a formar parte de la vista, ya que gracias a sus librerías y su serie de cabeceras es más fácil crear todos los elementos que se necesiten y verificar su adecuado funcionamiento dentro de su contexto (principalmente, tratándose de elementos tipo HTML). Este proyecto es relativamente pequeño, por los que es preferible que para este fin en particular se haga un esfuerzo por mejorar la parte de la lógica del negocio, que enfatizar sobre la parte de la vista. La parte visual se podrá mejorar posteriormente con elementos tales como hojas de estilo (CSS).

Para iniciar con la creación del proyecto en el IDE, se deben dar los siguientes parámetros una vez que se ejecuta el entorno NetBeans 6.0.1 y se decide realizar un nuevo proyecto de aplicación:

a) El nuevo proyecto deberá seleccionarse de la categoría 'Web' y el tipo de proyecto será ser de Web Application (aplicación Web). Dar clic en el botón *Next* (figura 2.1).

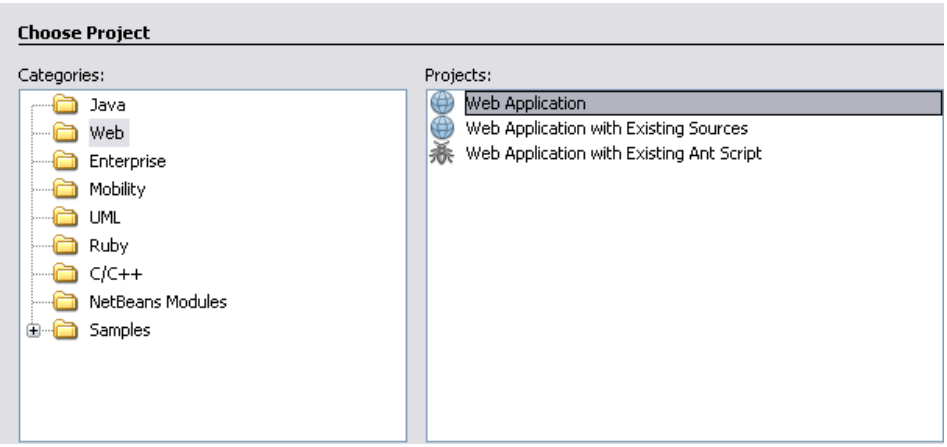


Figura 2.1: Selección de 'Aplicación Web' como tipo de proyecto.

b) Para el paso 2, se ingresará como nombre del proyecto "RIU", el servidor web de despliegue será Apache Tomcat 6.0.14, la versión de Java EE será la versión 5 y el

Path de Contexto donde será desplegada la aplicación será /RIU. Una vez hecho esto hay que dar clic en Next (figura 2.2).

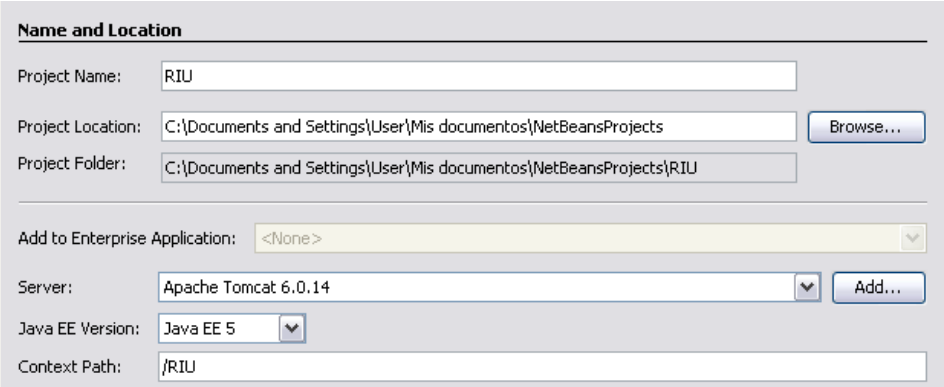


Figura 2.2: Selección del los atributos del proyecto.

c) Finalmente, en la pantalla de la figura 2.3 hay que seleccionar el Framework que se utilizará. Para este proyecto se selecciona Struts 1.2.9 y se da clic en la sección que dice "Add Struts TLD´s". Con esto, se incorporan las librerías referentes al Framework que vienen incluidas en el IDE por default y que contienen la serie de cabeceras estándar. Una vez realizado, se da clic en *Finish*.

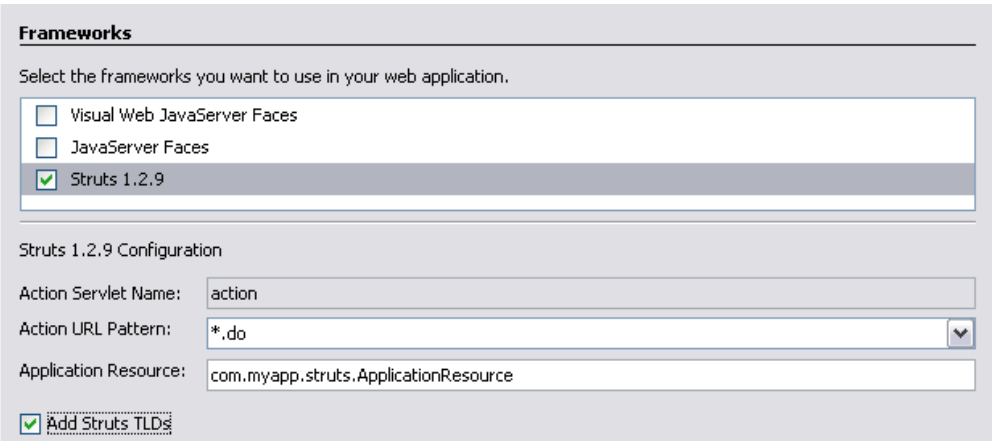


Figura 2.3: Selección del Framework de desarrollo (Struts 1.2.9).

El resultado de estas configuraciones generará el armazón de la aplicación. Se puede observar en la figura 2.4 la distribución de contenidos que han sido creados. Cada uno de estos tiene una utilidad o una función específica, sin embargo, para este desarrollo solo se comentarán las secciones que serán modificadas en el desarrollo del proyecto:

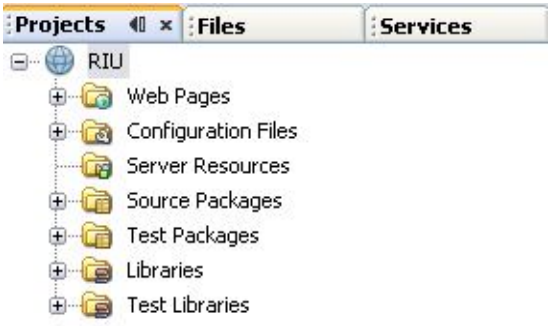


Figura 2.4: Distribución general de contenidos en el proyecto

- Web Pages: En esta sección se crearán todos los elementos visuales de la aplicación (normalmente páginas JSP) y se resguardarán los elementos ajenos a la aplicación (tales como hojas de estilo CSS e imágenes).

- Source Packages: Aquí se podrán incluir todas las clases Java, clases pertenecientes al Framework (por ejemplo, Action Struts) y archivos de recursos de propiedades que soporten la lógica de negocio del sistema.

- Libraries: Aquí se insertan todos los paquetes externos y librerías que se necesitarán de forma adicional. Es en esta parte donde se incorporará el driver de conexión a la base de datos basada en Oracle para su manejo en el proyecto.

En los siguientes puntos, se dispondrá a comentar algunos de los elementos que integran la aplicación, empezando por la parte de la vista, la parte del controlador, las clases Java que se encargarán de la conexión a la base de datos y archivos de configuración.

2.2.2: Desarrollo de capa de vista en la aplicación (páginas JSP)

En este apartado se describirán las páginas JSP que conformarán el proyecto. Sólo se mostrará el archivo JSP que hace referencia al formulario general para ejemplificar el panorama que se sigue dentro del desarrollo de aplicaciones basadas en Struts, pero por la enorme extensión de código que presentan debido a la cantidad de elementos HTML y estilos existentes, sólo será incluida la parte que involucra la creación del formulario como tal (en base a cabeceras propias del Framework Struts).

La distribución de las páginas JSP en la carpeta de contenidos "Web Pages" se muestra en la figura 2.5. En esta carpeta se incluyen tanto las páginas JSP como las hojas de estilo CSS y las imágenes de recursos que se necesitan para mostrar la página adecuadamente.

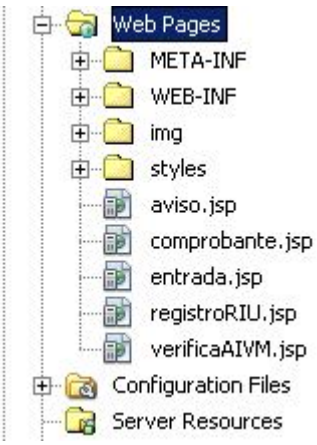


Figura 2.5: Distribución de la carpeta "Web Pages" en el proyecto

- **Acceso al Sistema (*entrada.jsp*)**

Esta es la página que permite el inicio de las transacciones con los usuarios hacia la base de datos. A diferencia del sistema anterior, se puede notar a primera vista que han cambiado considerablemente las reglas de negocio (figura 2.6 A). En el sistema original, se le solicitaba al usuario que proporcionara tres datos importantes: si contaba con un correo electrónico proporcionado por la UNAM, si se le había otorgado una cuenta de Acceso a Internet por parte de la DGSCA y que mencionara su nombramiento básico dentro de la UNAM.

Esta forma de proceder, sin embargo, ha llegado a traer problemas a los usuarios nuevos, ya que la lógica definida tiende a ser muy ambigua. Por esta razón, los usuarios empezaban a llenar de forma incorrecta su formulario de registro. Ahora, con la nueva propuesta, se procura ser lo más breve para que pueda iniciar sesión, garantizando que la información proporcionada sea correcta (figura 2.6 B).

Figura 2.6A: Página principal del sistema. Anterior sistema

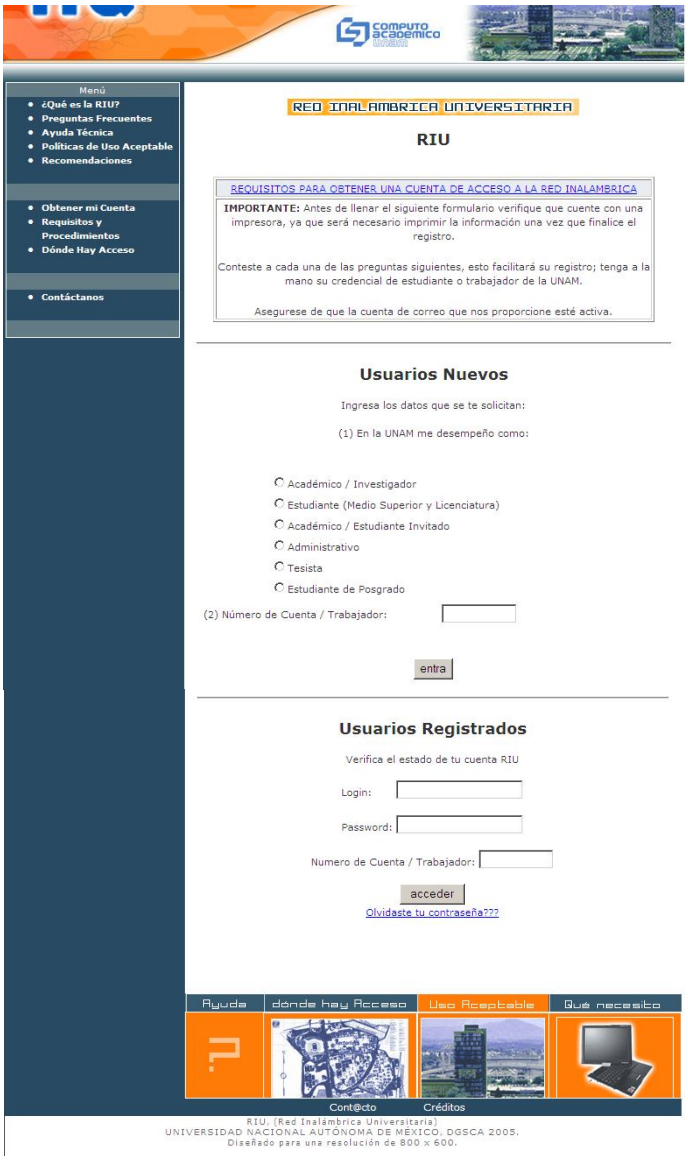


Figura 2.6B: Página principal del sistema. Nueva propuesta

Junto a esta propuesta, se ha dado partida a la creación de otro nuevo módulo, dedicado exclusivamente a los usuarios ya registrados y activos que cuentan con el servicio de RIU. Este módulo encuentra justificación en la posibilidad de evitar en lo más mínimo que un usuario pretenda registrarse en más de una ocasión si es que ya cuenta con el servicio en forma activa, y provocar con esto redundancia en la información de la base de datos.

Si un usuario conoce su *login*, su *password* y su número de cuenta o trabajador, podrá conocer el estado de su cuenta a través de un mensaje informativo. En caso de que algún dato este equivocado, se le podrá informar de las alternativas actuales que existen para que pueda recuperar su cuenta de acceso.

- **Validación de la cuenta de AIVM (*verificaAIVM.jsp*)**

La lógica que presenta esta página es la misma que se ha manejado desde el sistema original (figura 2.7A). Se le solicitará al usuario (sólo en caso de ser Académico o Investigador de la UNAM) que indique si cuenta con el servicio de Acceso a Internet Vía Modem por parte de la DGSCA; si lo confirma, se ingresarán los datos del login y el password. En caso de que sean correctos, tendrá acceso al formulario general del registro. Si no son correctos, o ya estaba registrado en el sistema, se le mostrará un mensaje informándole de la situación (figuras 2.7B y C).

Figura 2.7A: Solicitud de cuenta de AIVM para Académicos e Investigadores. Aspecto General

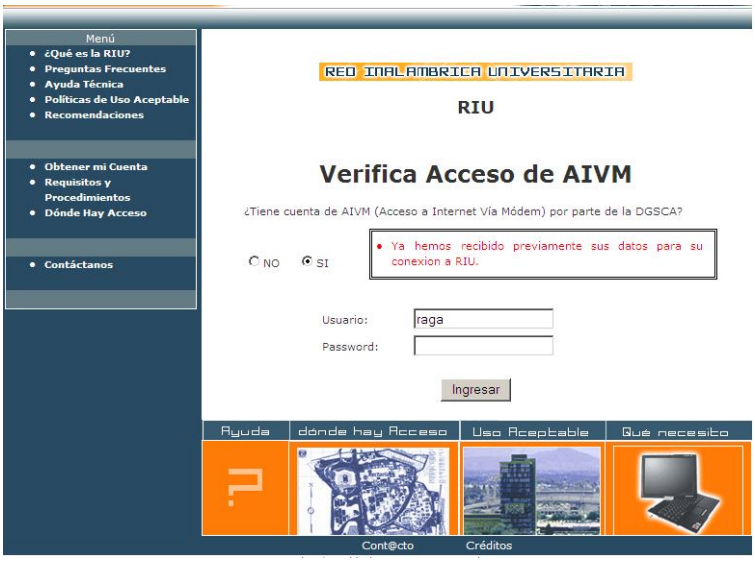


Figura 2.7B: Aspecto después de comprobar que ya ha registrado sus datos

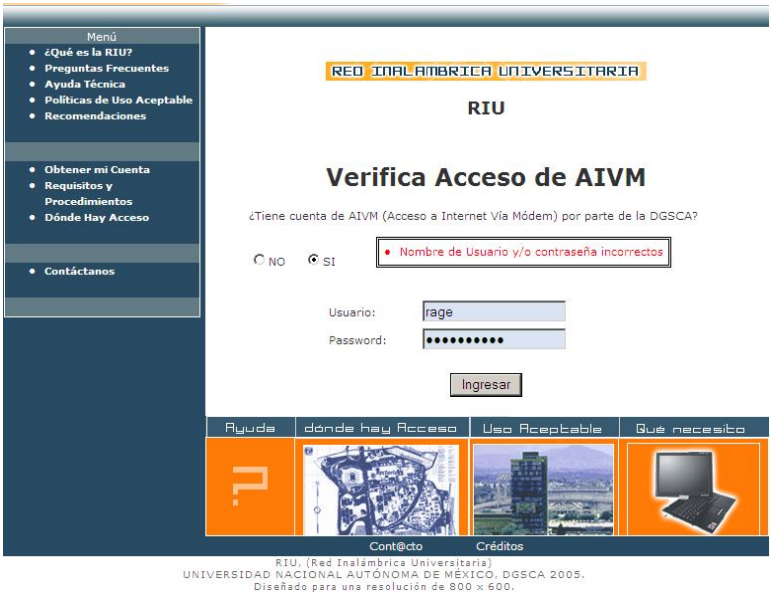


Figura 2.7C: Aspecto después de comprobar que los datos no existen

- **Formulario General de Registro (*registroRIU.jsp*)**

Una vez que se haya comprobado que los datos de la entrada son válidos a partir de las páginas *entrada.jsp* ó *verificaAIVM.jsp*, se procederá a mostrar el formulario general de registro al servicio de RIU. En caso de que un académico se haya autenticado con el servicio de AIVM, anexo al formulario de le mostrarán los datos que se tienen de él sobre dicho servicio. Una vista de esta página se puede encontrar en la figura 2.8. A comparación de la página original, existen datos que aún se siguen solicitando, tales como lo son:

- Nombre(s)
- Apellido Paterno
- Apellido Materno
- Descripción del nombramiento
- RFC
- CURP
- Dependencia de Adscripción
- Registro como máximo de 2 Direcciones MAC
- Correo electrónico (diferenciado sobre si es institucional o particular)

El cambio que se ha hecho en la lógica de negocio se ve reflejado en la solicitud del correo electrónico. En el sistema original, hay que recordar que esta información se solicitaba en 2 partes: primero se confirmaba si se cuenta con un correo electrónico específico y posteriormente se daba lugar a que lo escribiera. Ahora, todo lo relacionado con el correo se encuentra ubicado en una misma posición, y es personalizada en base a la respuesta que haya facilitado sobre el tipo de usuario al que está ligado (obtenido de la página *entrada.jsp*).

Figura 2.8: Formulario General de Registro a la RIU para Académicos

El listado 1 muestra el código fuente de la página JSP *registroRIU.jsp*.

Listado 1: Código fuente de la página JSP *registroRIU.jsp*.

```
1 <%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
2 <%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>
3 <%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
4
5 <%@ page import="Beans.Dependencia,java.util.ArrayList,java.util.Collection"
6       language="java" %>
7 <html:html>
```

Listado 1: Código fuente de la página JSP *registroRIU.jsp (Segunda Parte).*

```
 8   <head>
 9     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
10       <title>Red Inalámbrica Universitaria</title>
11         <link href="styles/styles.css" rel="stylesheet" type="text/css">
12   </head>
13   <body>
14     <html:form action="ActionInsertRIU" method="post">
15
16     <% if (request.getParameter("soy_unam").equals("1")) {%>
17     <html:hidden property="soy_unam" value="1"/>
18     <% }
19     if (request.getParameter("soy_unam").equals("2")) {%>
20     <html:hidden property="soy_unam" value="2"/>
21     <% }
22     if (request.getParameter("soy_unam").equals("3")) {%>
23     <html:hidden property="soy_unam" value="3"/>
24     <% }
25     if (request.getParameter("soy_unam").equals("4")) {%>
26     <html:hidden property="soy_unam" value="4"/>
27     <% }
28     if (request.getParameter("soy_unam").equals("5")) {%>
29     <html:hidden property="soy_unam" value="5"/>
30     <% }
31     if (request.getParameter("soy_unam").equals("6")) {%>
32     <html:hidden property="soy_unam" value="6"/>
33     <% }%>
34
35     <table class="text_general" border="0" width="85%">
36     <tr>
37     <td width="28%" colspan="2">Nombre: *<html:text property="nombre"
38       value="<%=(String)request.getAttribute("nombre")%>"/> </td>
39     <td width="16%" colspan="3"><html:errors property="nombre"/></td>
40     </tr>
41     <tr>
42    <td width="28%" colspan="2">Apellido Paterno: *<html:text property="apaterno"
43    value="<%=(String)request.getAttribute("apaterno")%>"/> </td>
44    <td width="16%"colspan="3"><html:errors property="apaterno"/></td>
45    </tr>
46
47    <tr>
48      <td width="28%" colspan="2">Apellido Materno: *<html:text property="amaterno"
49        value="<%=(String)request.getAttribute("amaterno")%>"/></td>
50       <td width="16%" colspan="3"><html:errors property="amaterno"/></td>
51    </tr>
52    <tr>
53    <td width="28%" colspan="2">Nombramiento: *<html:text property="nombramiento"
54       value="<%=(String)request.getAttribute("nombramiento")%>"/> </td>
55    <td width="16%" colspan="3"><html:errors property="nombramiento"/></td>
56    </tr>
57    <tr>
58      <td width="28%" colspan="2">Dependencia: * </td>
59      <td width="28%"> </td>
60    <td width="16%" colspan="3"><html:errors property="dependencia"/></td>
61    </tr>
62    <tr>
63      <%
64       Dependencia dep = new Dependencia();
65       ArrayList arr = dep.getDependencias();
66       request.setAttribute("objeto", arr);
67    %>
68    <td width="92%" colspan="6" height="23">
69    <html:select property="dependencia">
```

Listado 1: Código fuente de la página JSP *registroRIU.jsp (Tercera Parte).*

```
 70    <html:option value="">Selecciona tu dependencia</html:option>
 71      <logic:iterate id="depend" name="objeto" >
 72        <html:option value="${depend.clave_dep}">
 73          <bean:write name="depend" property="nombre_dep"/>
 74        </html:option>
 75      </logic:iterate>
 76    </html:select>
 77    </td>
 78    </tr>
 79    <tr>
 80      <td width="28%" colspan="2">CURP: *<html:text property="curp" /></td>
 81      <td width="16%" colspan="3"><html:errors property="curp"/></td>
 82    </tr>
 83    <tr>
 84   <td>RFC:<html:text property="rfc" value="<%=(String)request.getAttribute("rfc")%>"/>
 85   <td width="16%" colspan="3"><html:errors property="rfc"/></td>
 86   </tr>
 87   <html:hidden property="nu_cuenta" value="<%=request.getParameter("nu_cuenta")%>"/>
 88   <% if (request.getParameter("tiene_aivm").compareTo("SI") == 0) {%>
 89   <tr>
 90     <td width="28%" colspan="2">
 91     Nombre de Usuario:
 92   </td>
 93   <td><html:text property="login" maxlength="10" disabled="true" size="12"
 94    value="<%=(String)request.getAttribute("login")%>"/></td>
 95   <html:hidden property="login"value="<%=(String)request.getAttribute("login")%>"/>
 96   <html:hidden property="tiene_aivm" value="<%=(String)request.getAttribute
 97       ("tiene_aivm")%>"/>
 98   <td width="16%" colspan="3"><html:errors property="login"/></td> </tr>
 99   <% } else {%>
100   <tr>
101     <td width="28%" colspan="2">Nombre de Usuario:</td>
102     <td width="28%"><html:text property="login" maxlength="10" size="12"/></td>
103     <html:hidden property="tiene_aivm" value=""/>
104     <td width="16%" colspan="3"><html:errors property="login"/></td>
105   </tr>
106   <%}
107    if (request.getParameter("conAIVM").equals("1")) {%>
108    <html:hidden property="conAIVM" value="1"/>
109    <html:hidden property="idUser" value="<%=(String)request.getAttribute("idUser")%>"/>
110   <% } else {%>
111   <html:hidden property="conAIVM" value="0"/>
112   <%}%>
113   <tr>
114     <td width="98%" colspan="7" height="63"><hr></td>
115   </tr>
116   <tr>
117     <td colspan="3"> Datos del equipo a conectar en la red *</td>
118   </tr>
119   <tr>
120   <td><html:radio property="numero_equipo" value="1" onclick="visualizaMAC();">
121       1 </html:radio></td>
122   <td><html:radio property="numero_equipo" value="2" onclick="visualizaMAC2();">
123       2 </html:radio></td>
124   <td><html:errors property="numero_equipo"/></td>
125   </tr>
126   <html:hidden property="resp_numequipo" value="<%=request.getParameter
127       ("numero_equipo")%>"/>
128      <tr>    <td width="28%">Dirección MAC 1 *</td>
129        <td width="18%">
```

Listado 1: Código fuente de la página JSP *registroRIU.jsp (Cuarta Parte).*

```
130          <html:select property="tipo_equipo">
131             <html:option value="1">Laptop</html:option>
132             <html:option value="2">PDA</html:option>
133             <html:option value="3">Otro</html:option>
134          </html:select></td>
135        <td width="7%">MAC:<html:text property="mac1" size="12" maxlength="12"/></td>
136             <td width="16%" colspan="3"><html:errors property="mac1"/></td>
137       </tr>
138       <tr>
139           <td width="28%">Dirección MAC 2:</td>
140           <td width="18%">
141               <html:select property="tipo_equipo2">
142                   <html:option value="1">Laptop</html:option>
143                   <html:option value="2">PDA</html:option>
144                   <html:option value="3">Otro</html:option>
145           </html:select></td>
146         <td width="7%">MAC:<html:text property="mac2" size="12" maxlength="12" /></td>
147             <td width="16%" colspan="3"><html:errors property="mac2"/></td>
148       </tr>
149      <br>
150     <tr>
151      <td width="42%" colspan="3">Correo Electrónico:</td>
152      <td width="7%"> </td>
153      <td width="16%"> </td>
154     </tr>
155      <%   if (request.getParameter("soy_unam").equals("1")) {%>
156      <tr>
157       <td width="67%" colspan="5" height="46"> Tiene usted cuenta de correo en
158         el dominio @servidor.unam.mx o de las FES??? * </td>
159      </tr>
160      <tr>
161       <td width="10%"><html:radio property="tiene_mail" value="SI"
162           onclick="visualizaCorreoSI();">
163           SI</html:radio></td>
164      <td width="31%" colspan="2"><html:text property="email_unam"/>@</td>
165      <td width="27%" colspan="2">
166    <html:select property="tipo_mail"> <!= Cambiar %>
167      <html:option value="servidor.unam.mx">servidor.unam.mx</html:option>
168      <html:option value="apolo.acatlan.unam.mx">apolo.acatlan.unam.mx</html:option>
169      <html:option value="puma2.zaragoza.unam.mx">puma2.zaragoza.unam.mx</html:option>
170      <html:option value="campus.iztacala.unam.mx">campus.iztacala.unam.mx</html:option>
171      <html:option value="fesc.cuautitlan.unam.mx">fesc.cuautitlan.unam.mx</html:option>
172    </html:select></td>
173      <td width="26%"> <html:errors property="email_unam"/></td>
174    </tr>
175     <tr>
176     <td width="5%"> </td>
177   <td width="10%"><html:radio property="tiene_mail" value="NO"
178   onclick="visualizaCorreoNO();"> NO (Hotmail, Yahoo, Gmail, etc) </html:radio></td>
179             <td width="31%" colspan="2"><html:text property="email_alterna"/></td>
180             <td width="27%" colspan="2"><html:errors property="email_alterna"/></td>
181         </tr>
182
183         <% } else {
184         if (request.getParameter("soy_unam").equals("2") ||
185                 request.getParameter("soy_unam").equals("5") ||
186                 request.getParameter("soy_unam").equals("6")) {%>
187         <tr>
188             <td width="67%" colspan="5"> Tienes cuenta de correo vigente en el
189             dominio @escolar.unam.mx *</td>
190         </tr>
```

Listado 1: Código fuente de la página JSP *registroRIU.jsp (Quinta Parte).*

```
191         <tr>
192             <td width="10%"><html:radio property="tiene_mail" value="SI"
193                 onclick="visualizaCorreoSIEST();">
194             SI</html:radio></td>
195             <td width="31%" colspan="2"><html:text property="email_unam" size="9"/>
196             @escolar.unam.mx</td>
197             <td width="26%"><html:errors property="email_unam"/></td>
198         </tr>
199         <tr>
200             <td width="28%" colspan="2"><html:radio property="tiene_mail"
201                 value="NO" onclick="visualizaCorreoNOEST();">
202             NO (Hotmail, Yahoo, Gmail, etc)  </html:radio></td>
203             <td width="18%"><html:text property="email_alterna"/></td>
204             <td width="16%"> <html:errors property="email_alterna"/></td>
205         </tr>
206         <% } else {%>
207         <tr>
208             <td width="28%" colspan="2"> Correo Electrónico: * </td>
209             <td width="18%"><html:text property="email_unam" size="30"/></td>
210             <td width="7%"><html:errors property="email_unam"/></td>
211         </tr>
212         <% }  } %>
213         <tr>
214             <td width="100%" colspan="7"> </td>
215         </tr>
216     </table>
217     <br><br>
218     <html:submit value="Ingresar Datos" property="Estado"/><br><br>
219 </html:form>
220 </body>
221 </html:html>
222
223
```


- **Expedición del comprobante de registro (*comprobante.jsp*)**

Si todos los datos ingresados en el formulario de registro son correctos, el siguiente paso es insertarlos en la base de datos y mostrar una página donde se muestre el comprobante con las instrucciones para que el usuario complete su registro de RIU en las instalaciones de la DGSCA (figura 2.9).

No tiene mayores complicaciones con respecto a la página del sistema original, sólo es necesario migrarla en base a los elementos propios del Framework.

Figura 2.9: Comprobante de Registro al servicio de RIU

- **Página de mensajes informativos (*aviso.jsp*)**

Esta página tiene una funcionalidad dinámica, y funcionará a partir de la página principal de registro (*entrada.jsp*). Es dinámica porque a partir de los errores que se puedan encontrar, se dispondrá a desplegar un mensaje que oriente al usuario sobre las probables acciones que desean realizar. La relación de errores y su posible mensaje de ayuda a desplegar se visualizan en la tabla 2.3:

Tabla 2.3: Casos de error y mensajes de despliegue en la página aviso.*jsp*

Error	Ayuda
El usuario pretende ingresar sus datos nuevamente en el sistema, a sabiendas de que se tienen sus datos en el sistema.	Se le orientará sobre las posibles opciones que puede realizar para evitar que se registre nuevamente.
El usuario ingreso sus datos para conocer el estado de su cuenta (la información es incorrecta).	Se le informará que acciones puede tomar, ya que probablemente ha extraviado su comprobante de registro con su nombre de usuario y contraseña.
El usuario ingreso sus datos para conocer el estado de su cuenta (no presenta problemas).	Se le informa que su cuenta está funcionando correctamente. Si tiene algún problema con el servicio puede ser debido a otros factores.
El usuario ingreso sus datos para conocer el estado de su cuenta (Esta se encuentra próxima a pasar por un periodo de renovación).	Se le brindará orientación sobre las acciones que deberá considerar para realizar el trámite de renovación de su cuenta.

2.2.3: Clases Action (Controlador)

Dejando atrás la parte de la vista en la aplicación, es momento de empezar a preocuparse por la parte de lógica del negocio y las reglas de navegación que se seguirán (véase la figura 2.10). Existen dos archivos que se encargan de empezar a trabajar con la información que se esté generando en ese momento, y a partir de ésta decidir el flujo que seguirá la aplicación: estos archivos son ActionRIU.java y ActionInsertRIU.java:

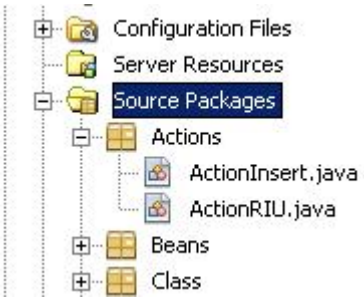


Figura 2.10: Distribución de la carpeta “Source Packages” y su paquete ‘Actions’

- **ActionRIU.java**

Esta clase Action es la más importante del sistema, ya que a partir de ésta se puede direccionar el flujo del sistema a partir de los 2 formularios existentes en la página principal *entrada.jsp*.

Esta clase trabaja diferente a una clase Action común, ya que hereda de la clase *DispatchAction* la posibilidad de realizar múltiples procesos (en este caso, el registro de usuarios nuevos, la comprobación de usuarios existentes, y la verificación de la existencia de cuentas de AIVM), con lo que se evitaría crear clases Action, incrementando con esto la cantidad de archivos a desarrollar en el proyecto.

- **ActionInsert.java**

Esta clase Action tiene como única función recopilar la información ofrecida por el usuario en el formulario de registro (*registroRIU.jsp*), realizar comparaciones como parte del proceso de validación de los datos y enviarla a la clase externa que se encargará de insertarla en la base de datos.

En cuanto a la navegación, esta clase decidirá qué página desplegar en caso de que se hayan insertado los datos en forma correcta (*comprobante.jsp*). Si no es posible insertarlos, se retornará a la página del formulario (*registroRIU.jsp*) para desplegar los errores que se pudieran haber generado.

2.2.4: Clases Java para el manejo de transacciones (DAO)

La clase *accionesRIU.java* es la responsable de realizar la comunicación directa con la base de datos, obtener las consultas solicitadas y realizar tanto las inserciones como las actualizaciones que se puedan llegar a necesitar. Se encuentra almacenada dentro del paquete *Class* dentro del proyecto (véase la figura 2.11).

Cada uno de los métodos existentes es capaz de enviar al controlador, o dicho de otra forma, a la respectiva clase Action (*ActionRIU.java* o *ActionInsert.java*) una señal si pudo realizar su proceso satisfactoriamente o si encontró un error al momento de ejecutarla, para tratarla y continuar con el flujo de navegación en el sistema.

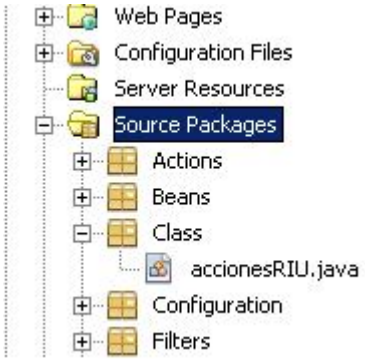


Figura 2.11: Distribución de la carpeta “Source Packages” y su paquete ‘Classs’

Los métodos realizan el proceso de conexión a la base de datos (en Oracle) mediante el complemento de instrucciones basadas en JDBC. Es importante hacer notar que la implementación de esta clase es estándar como cualquier otra clase común de Java y cada uno de los diez métodos declarados debe realizar el proceso de abrir y cerrar la conexión a la base de datos antes y después de ejecutar una instrucción SQL. Con esto se evita que puedan existir problemas al momento de realizar cualquier consulta o movimiento en la base de datos.

2.2.5: Archivo de configuración (struts-config.xml)

El archivo de configuración es uno de los componentes que más trabajo cuesta diseñar en una aplicación Web, ya que es el núcleo central que determina los recursos con los que se contarán y las reglas de navegación que se deberán seguir para que el sistema pueda desempeñarse de la mejor manera. Vale mucho la pena mostrar y explicar todos los cambios en la configuración que se han realizado dentro de este archivo, los cuales se muestran en el listado 2:

Listado 2: Código fuente del archivo de configuración *struts-config.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>
    <form-beans>
        <form-bean name="UserRIU" type="Beans.UserRIU" />

        <form-bean name="dynaEstado" type="org.apache.struts.action.DynaActionForm">
            <form-property name="login"
                           type="java.lang.String"/>
            <form-property name="password"
                           type="java.lang.String"/>
            <form-property name="nu_cuenta2"
                           type="java.lang.String"/>
            <form-property name="nu_cuenta"
                           type="java.lang.String"/>
            <form-property name="soy_unam"
                           type="java.lang.String"/>
            <form-property name="tiene_aivm"
                           type="java.lang.String"/>
            <form-property name="avisos"
                           type="java.lang.String"/>
        </form-bean>

        <form-bean name="dynaAIVM" type="org.apache.struts.action.DynaActionForm">
            <form-property name="user_aivm"
                           type="java.lang.String"/>
            <form-property name="pwd_aivm"
                           type="java.lang.String"/>
            <form-property name="tiene_aivm"
                           type="java.lang.String"/>
            <form-property name="soy_unam"
                           type="java.lang.String"/>
        </form-bean>

    </form-beans>
```

Listado 2: Código fuente del archivo de configuración *struts-config.xml (Segunda Parte)*

```
41      <global-exceptions>
42
43      </global-exceptions>
44
45      <global-forwards>
46          <forward name="welcome"  path="/Welcome.do"/>
47          <forward name="index" path="/entrada.jsp"/>
48      </global-forwards>
49
50      <action-mappings>
51           <action path="/ActionAviso"
52                   input="/entrada.jsp"
53                   type="Actions.ActionRIU"
54                   scope="request"
55                   parameter="estado"
56                   validate="true"
57                   name="dynaEstado">
58          <forward name="success" path="/aviso.jsp"/>
59          <forward name="aivm" path="/verificaAIVM.jsp"/>
60          <forward name="error" path="/entrada.jsp"/>
61          <forward name="registro" path="/registroRIU.jsp"/>
62          <forward name="existe" path="/registroRIU.jsp"/>
63          </action>
64
65          <action path="/ActionRIU"
66                 input="/verificaAIVM.jsp"
67                 type="Actions.ActionRIU"
68                 scope="request"
69                 parameter="ingresar"
70                 validate="true"
71                 name="dynaAIVM">
72              <forward name="registro" path="/registroRIU.jsp"/>
73              <forward name="error" path="/verificaAIVM.jsp"/>
74          </action>
75
76         <action  path="/ActionInsertRIU"
77                  input="/registroRIU.jsp"
78                  name="UserRIU"
79                  scope="request"
80                  validate="true"
81                  type="Actions.ActionInsert">
82          <forward name="success" path="/comprobante.jsp"/>
83          <forward name="error" path="/entrada.jsp"/>
84          </action>
85
86          <action path="/Welcome" forward="/welcomeStruts.jsp"/>
87      </action-mappings>
88
89      <controller processorClass="org.apache.struts.tiles.TilesRequestProcessor"/>
90
91      <message-resources parameter="com/myapp/struts/ApplicationResource"/>
92
93      <plug-in className="org.apache.struts.tiles.TilesPlugin" >
94          <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
95          <set-property property="moduleAware" value="true" />
96      </plug-in>
97
98      <!-- ======================== Validator plugin ============================== -->
99      <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
100     <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-
101         INF/validation.xml,/WEB-INF/validate-formRegister.xml"/>
102     </plug-in>
103     </struts-config>
```

2.2.6: Clase UserRIU.java (ValidatorForm)

Este archivo únicamente funciona como referencia a los datos que se están manejando por parte del archivo de formulario (*registroRIU.java*), y es semejante a un Java Bean tradicional. A diferencia de una clase **ActionForm** normal en donde se hace una invocación a su método ***validate()*** para realizar la validación de los datos, esta clase extenderá a la clase **ValidatorForm**, lo que implicará contar con un archivo en formato XML que apoyará al proceso de validación de los datos.

2.2.7: Archivo de validaciones: (validate-formRegister.xml)

Es sabido que algunos datos requieren ser validados conforme a un patrón específico. Si no llevan como mínimo este requisito, no se pueden considerar como correctos y es responsabilidad del sistema notificar al usuario que ha existido un error en alguno de estos campos. El archivo *validate-formRegister.xml* apoya al archivo *UserRIU.java* para que realice la validación sobre algunos de los datos que en este último archivo se encuentran. La extensión a la clase ValidatorForm en el archivo *UserRIU.java*, implica tener la necesidad de contar con un archivo de apoyo exclusivo para validaciones.

La serie de datos que serán validados por este archivo son: el RFC (Registro Federal de Contribuyentes), la CURP (Clave Única de Registro de Población) y las direcciones MAC con las que cuentan en forma única los dispositivos inalámbricos. Los requerimientos mínimos desarrollados en cada uno de estos datos se comentan en la tabla 2.4**.**

No se tiene contemplado realizar validaciones tan complejas en este archivo, tan sólo se consideran aspectos sencillos, tales como que los campos sean obligatorios, que tengan una mínima o máxima longitud dependiendo del dato a evaluar, o que sigan un

patrón en forma de expresión regular para comprobar que sean válidos. El aspecto de este archivo de validación se muestra en el listado 3.

Nombre del Campo	Patrón	Descripción del patrón
CURP	AAAA######AAAAAA##	Una CURP se compone de 4 letras, 6 números, 5 letras y 2 números
RFC	AAAA######$$$	Un RFC se compone de 4 letras, 6 números y 3 dígitos más que son asignados por Hacienda. Si no lo tiene, basta con los 10 primeros caracteres de su CURP.
Dirección MAC	[0-9A-F]12	Se compone de 12 caracteres, que pueden ser numéricos de 0 al 9 y de la A hasta la F.

Tabla 2.4: Campos especiales en el formulario general de registro

Listado 3: Código fuente del archivo de validación *validate-formRegister.xml*


```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE form-validation PUBLIC
   "-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
          "http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">

<form-validation>
    <formset>
        <form name="UserRIU">
             <field property="nombre" depends="required "></field>
             <field property="apaterno" depends="required"></field>
             <field property="amaterno" depends="required"></field>
             <field property="nombramiento" depends="required"></field>
             <field property="dependencia" depends="required"></field>
             <field property="numero_equipo" depends="required"></field>
             <field property="curp" depends="required,mask">
                 <var>
                     <var-name>mask</var-name>
                     <var-value>^[A-Z]{4}[0-9]{6}[A-Z]{6}[0-9]{2}$</var-value>
                 </var>
             </field>
             <field property="rfc" depends="required,mask">
                 <var>
                     <var-name>mask</var-name>
                     <var-value>^[A-Z]{4}[0-9]{6}$</var-value>
                 </var>
             </field>
```

Listado 3: Código fuente del archivo de validación *validate-formRegister.xml* (Segunda Parte)

```
28               <field property="mac1" depends="required,mask">
29                  <var>
30                      <var-name>mask</var-name>
31                      <var-value>^[A-F0-9]{12}$</var-value>
32                  </var>
33              </field>
34               <field property="mac2" depends="mask">
35                  <var>
36                      <var-name>mask</var-name>
37                      <var-value>^[A-F0-9]{12}$</var-value>
38                  </var>
39              </field>
40              <field property="login" depends="required,minlength,mask">
41                  <arg0 name="minlength" key="${var:minlength}" resource="false"/>
42                  <var>
43                      <var-name>minlength</var-name>
44                      <var-value>4</var-value>
45                  </var>
46                  <var>
47                      <var-name>mask</var-name>
48                      <var-value>^[a-z0-9\\_]</var-value>
49                  </var>
50              </field>
51          </form>
52      </formset>
53 </form-validation>
```

2.2.8: Filtro de Seguridad (Filter1.java)

Se encuentra ubicado dentro del paquete Filters (véase la figura 2.12). La finalidad de contar con este archivo tiene una razón de ser importante. Este archivo se encargará de recibir todas las peticiones del usuario que pretendan dirigirse a páginas o acciones ajenas a la página de bienvenida del sistema (*entrada.jsp*) a través de direcciones URL escritas en el navegador.

Con este filtro, se garantiza que un usuario intervendrá en el flujo de navegación desde el punto de partida establecido, hasta la consecución del mismo en forma satisfactoria (llegando a la página *comprobante.jsp*) o la muestra de algún error provocado por el mismo usuario en la inserción de su información.

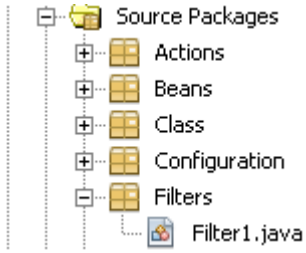


Figura 2.12: Distribución de la carpeta “Source Packages” y su paquete ‘Filters’

Sin este filtro, el sistema estaría susceptible a que pudiera generarse incongruencia en la información, ya que el usuario podría ingresar libremente a secciones que se pueden considerar como ‘privadas’, lo cual provocaría errores inesperados, tales como intentos de conexión a la base de datos fallidos y errores en tiempo real provocados por falta de información que pueda ser requerida por cierto proceso. El diseño del filtro es sumamente sencillo, y es presentado en el listado 4:

Listado 4: Código fuente del filtro *Filter1.java*


```
 1
 2 package Filters;
 3
 4 import java.io.IOException;
 5 import javax.servlet.Filter;
 6 import javax.servlet.FilterChain;
 7 import javax.servlet.FilterConfig;
 8 import javax.servlet.ServletException;
 9 import javax.servlet.ServletRequest;
10 import javax.servlet.ServletResponse;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13
14 public class Filter1 implements Filter{
15
16     public void init(FilterConfig arg0) throws ServletException {  }
17
18     public void doFilter(ServletRequest req, ServletResponse res, FilterChain arg2)
19                 throws IOException, ServletException {
20
21         if(req instanceof HttpServletRequest){
22             String revnuCuenta=((HttpServletRequest)req).getParameter("nu_cuenta");
23             String revnuCuenta2=((HttpServletRequest)req).getParameter("nu_cuenta2");
24             String user_aivm=((HttpServletRequest)req).getParameter("user_aivm");
25
26            if(revnuCuenta!=null || revnuCuenta2!=null || user_aivm!=null) {
27                  arg2.doFilter(req, res);
28            }
```

Listado 4: Código fuente del filtro *Filter1.java (Segunda Parte)*

```
29            else{
30                HttpServletResponse respuesta=(HttpServletResponse)res;
31                respuesta.sendRedirect("./entrada.jsp");
32            }
33         }
34     }
35     public void destroy() { throw new UnsupportedOperationException("Not supported
36                               yet.");
37       }
38     }
```

2.2.9: Archivo de recursos (ApplicationResource.properties)

Los archivos de propiedades tienen una utilidad muy práctica cuando se está trabajando con un Framework, tal y como lo es Struts. En este archivo, se enlistan todos los posibles mensajes informativos, de errores y etiquetas que la aplicación pueda utilizar para mostrar al usuario cuando esté ejecutando la aplicación. En este proyecto, se contempla utilizar el archivo que se genera por default (ubicado en el paquete *com.myapp.struts*) y se actualizará con todos los mensajes que explicarán al usuario lo que ocurre con cada una de sus peticiones.

2.2.10: Aspecto general del sistema

Un panorama gráfico que engloba todas las observaciones hechas se pueden encontrar en la figura 2.13, donde se visualiza el flujo de navegación que se sigue a través de las diferentes pantallas gráficas propuestas. Depende del resultado generado por cada uno de los procesos internos para determinar a qué página se direccionará en caso de realizar una acción exitosa, o una acción que genere un error.

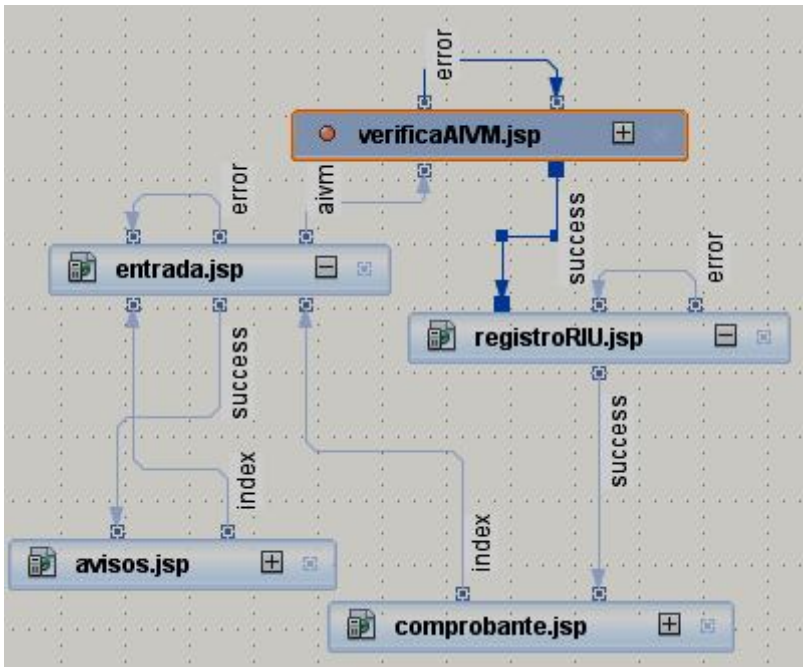


Figura 2.13: Flujo de navegación sugerido para el desarrollo del proyecto

CONCLUSIONES

CONCLUSIONES

Los Sistemas de Información actuales no sólo reflejan su eficiencia en base a la tecnología seleccionada, estilo de programación o lógica de negocio definida para su desarrollo. Si no se cumple con suficiente tiempo para dedicarlo a las fases de plantificación y análisis, por más que la tecnología utilizada llegue a ser tan poderosa y que el producto final sea un servicio tan común (como lo es el formulario de registro a usuarios en el sitio de la Red Inalámbrica Universitaria), no se podrán recopilar los resultados esperados, aparte de que existe una alta probabilidad de ser susceptible a pérdidas tanto en tiempo como en costos de desarrollo.

Sin embargo, en el caso de que la fase de planeación pueda considerarse satisfactoria, la prioridad puede recaer en la selección de la tecnología para su desarrollo. A lo largo del presente trabajo, se han expuesto las ventajas de contar con el lenguaje Java como base para la implementación del sistema, así como de la inclusión de un marco de desarrollo para facilitar y homologar los procesos de codificación.

Dentro del marco de trabajo, Struts resulta ser una buena herramienta para el desarrollo de sistemas Web basados en Java, indistintamente de la magnitud y objetivos planificados. Utiliza una metodología que permite progresar en forma modular, lo que permite al momento tener una clara distinción de cada uno de los componentes desarrollados, lo que en el largo plazo puede dar paso sin mayor dificultad a un crecimiento más completo de la aplicación y adaptarla para que pueda ser más robusta que el diseño original.

De acuerdo a lo anterior, se puede afirmar que el diseño y la implementación que han sido aplicados al proyecto de desarrollo, han dado la pauta para que permita al personal del Centro de Atención a Usuarios de la DGSCA, en un futuro no muy lejano, poder actualizar su sistema con el objeto de integrar más servicios de los que en esta ocasión se han propuesto, redefinir sus necesidades particulares ante los cambios, y con esto acercarse más a la realización de uno de los objetivos más importantes que se han descrito en el área de la informática en los últimos años: la automatización de procesos.

Aunque Struts es un buen marco de trabajo, no hay que limitarse a ser un experto en este Framework. Tal y como lo se mencionó puntualmente, existen muchas más alternativas que permiten un desarrollo igual de cómodo para el programador, y satisfactorio para el usuario final. JavaServer Faces (JSF) es una buena opción, sobre todo para los que lo se consideran expertos en la materia de programación.

En el manejo de la capa de persistencia, sería bueno considerar otras opciones ajenas al desarrollo en JDBC igual de populares en el mercado, tales como Ibatis y Hibernate, mismo que en área profesional han tenido en los últimos años una enorme aceptación.

Hay que recordar que los conocimientos para la realización del presente proyecto han sido adquiridos en el Diplomado de Java Máster, impartido en las instalaciones de la FES Aragón. A juicio personal, este diplomado ha sido benéfico para todos los participantes, ya que los contenidos se adaptan a la mayoría de las necesidades que un profesionista en el campo del desarrollo de sistemas informáticos en materia del lenguaje Java debe poseer.

Sin embargo, aterrizando en un plano más realista, no es posible hacer que los conocimientos adquiridos encuentren su límite en este curso. Un verdadero desarrollador de sistemas tiene que tener la capacidad de ser autodidacta, valor importante que el mercado laboral contempla hoy en día.

Para el desarrollo de este proyecto, se han consultado de manera importante una serie de fuentes y referencias ajenas a la información base del diplomado. Estas referencias se pueden encontrar en la bibliografía anexa al presente trabajo.

BIBLIOGRAFÍA

- DEITEL, HARVEY M.
  Como programar en Java. Quinta Edición
  México, 2004
  Ed. Pearson Education
  p.p. 1332

- GONZALEZ BUSTAMANTE, OSCAR ALEJANDRO
  Java Básico
  México, 2001
  Centro de Investigación en Computación, Instituto Politécnico Nacional
  p.p. 471

- GEARY, DAVID
  Core JavaServer Faces, Second Edition
  2007
  Ed. Prentice Hall
  p.p. 752

- HALL, MARTIN
  Servlets y Java Server Pages. Guía Práctica
  México, 2001
  Ed. Pearson Education.
  p.p. 608

- HOLMES, JAMES
  Struts: The Complete Reference. Second Edition
  2007
  Ed. Mc. Graw Hill / Osborne
  p.p. 550

- TURNER, JAMES
  MySQL and JSP Web Applications: Data-Driven Programming Using Tomcat and MySQL
  2002
  Ed. Sams Publishing
  p.p. 400