UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

"SISTEMA DE AUTOMATIZACIÓN DEL PROCESO DE PREINSCRIPCIÓN Y GESTIÓN DE CURSOS VÍA WEB"

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

CHRISTIAN BORUNDA PACHECO

ASESOR
BIOL. LIZBETH HERAS LARA

México D.F., Mayo 2008.

Agradecimientos

- *Dios, por haberme brindado la fortaleza, tiempo y sobre todo salud para concluir con este sueño.*

- *Hilda Josefina Pacheco Aguilar, Luis Borunda Trevizo y Mónica Borunda Pacheco por su amor, apoyo, educación y compañía a lo largo de mi vida.*

- *Rosa Ángela Padrón Serna, a ti cariño por tu amor, apoyo y compañía en esta etapa de nuestras vidas.*

- *Horacio, Ana Luisa, Jesús y Jorge, por su invaluable amistad y los alegres momentos que hemos vivido juntos en tantos años.*

- *Lidia y Ricardo Arteaga, por su amistad y gran apoyo invaluable para la terminación de este trabajo.*

- *Ricardo, Saúl, Sergio y Noemí, por su amistad y grandes momentos que hemos pasado tanto en el cielo como en la tierra.*

- *Adrián, Hanoi, Domingo, Miguel y Arturo, por su divertida amistad y los momentos agradables durante mi estancia en la universidad.*

- *Biol. Lizbeth Heras Lara, por su dirección, asesoría y ayuda en este trabajo.*

- *M. en C. Marcelo Pérez Medel, Ing. Blanca Estela Cruz Luévano, M. en C. Jesús Hernández Cabrera y Biol. José Luis Villarreal Benítez, por la valiosa revisión que hicieron a este trabajo.*

- *Y sobre todo a la Universidad Nacional Autónoma de México, Facultad de Estudios Superiores Aragón, por haberme brindado la oportunidad de ser un orgulloso egresado de la máxima casa de estudios.*

A todos ustedes, gracias totales.

Í N D I C E

INTRODUCCIÓN

Los lenguajes de programación, los paradigmas y las metodologías de desarrollo no son conceptos antiguos. Sin embargo, a menudo se tiene la sensación de que los métodos y las técnicas que aplicábamos ayer, hoy son anticuadas. Estos cambios tan rápido hacen que la programación siga siendo interesante, ya que siempre hay algo nuevo en el horizonte. La mayoría de los cambios son increméntales y simplemente es necesario integrarlos en nuestra estrategia de programación. Sin embargo, hay elementos hacen que volvamos a plantear conceptos fundamentales y que volvamos a analizar qué significa ser programador. Java es uno de estos e internet es otro.

Actualmente, el incremento en el uso de Internet ha provocado que se traten una serie de problemas que habían sido ignorados durante las décadas pasadas: seguridad y portabilidad. Ningún lenguaje ya existente podía tratar estos problemas adecuadamente. Durante años, todos los esfuerzos se han centrado en desarrollar software con grandes ventajas que se pueden ejecutar en una PC mono-usuario, y no en el entorno distribuido de Internet. Sin embargo, con la aparición del mundo Web, los problemas de portabilidad y seguridad no han podido ser ignorados. Java ha resuelto estos dos problemas de una forma elegante.

Como ya se sabe, Java está basado en el lenguaje C++. Y por supuesto, el lenguaje C++ está basado en el lenguaje C. Los lenguajes C, C++ y ahora Java, son los lenguajes de los programadores profesionales, es decir, los lenguajes en los que se implementan la mayoría de los trabajos de programación actuales. Pero lo que es más importante, si se divide el mundo de los lenguajes de programación en dos partes, C, C++ y Java estarían en un lado y el resto de los lenguajes en el otro. Actualmente, estos tres lenguajes representan la corriente principal de la programación en el mundo laboral. El conocimiento de estos lenguajes no es sólo algo opcional, sino que llegar a ser algo necesario, por lo que para ser un buen programador es necesario tener fluidez en los lenguajes C, C++ y Java.

La tecnología Java ha evolucionado de ser un lenguaje de programación diseñado para crear sistemas empotrados independientes de la plataforma hasta convertirse en una tecnología robusta para servidores, independiente del proveedor y de la plataforma, la cual permite a la comunidad obtener el potencial completo de las aplicaciones centradas en web.

Java comenzó con el lanzamiento del Kit de Desarrollo de Java (JDK). Desde el principio fue obvio que Java iba por la vía rápida para convertirse en una solución a muchos problemas de los sistemas empresariales. El JDK fue extendiéndose con interfaces y bibliotecas según el mundo empresarial exigía —y recibía— interfaces de programación de aplicaciones (API) que resolvían problemas del mundo real.

Con el lanzamiento de Java 2 Standard Edition (J2SE) se integraron por completo muchas extensiones a las Interfaces de Programación de Aplicaciones (API) del JDK. J2SE contiene todas las API necesarias para construir aplicaciones Java robustas de misión crítica. Sin embargo, el mundo corporativo pensaba que J2SE carecía de la fuerza requerida para desarrollar aplicaciones empresariales.

Nuevamente el mundo empresarial motivó a Sun Microsystems a revisar la tecnología Java para resolver las necesidades de la empresa. Sun inició entonces el Programa comunitario de Java *(Java Community Program* o JCP), el cual reunió a los usuarios corporativos, proveedores y técnicos para desarrollar una forma estándar de las API empresariales de Java. El resultado fue la plataforma Java 2, Enterprise Edition, conocida comúnmente como J2EE.

Tradicionalmente los sistemas corporativos se diseñaban utilizando el modelo cliente-servidor, en el cual los sistemas clientes solicitaban procesamiento de los sistemas servidores. Sin embargo, los sistemas empresariales estaban a su vez evolucionando. Un nuevo modelo conocido como servicios web fue reemplazando en forma gradual al modelo cliente-servidor en las corporaciones.

En el modelo de servicios web, los programadores de aplicaciones ensamblan aplicaciones a partir de un conjunto de componentes de procesamiento conocidos como servicios web. Cada servicio web es independiente de los demás servicios web y de las aplicaciones. Las aplicaciones cliente se comunican con una aplicación servidora en la capa central, la cual a su vez interactúa con los servicios web necesarios que también se encuentran en el extremo del servidor.

Al momento de que las empresas adoptaron los servicios web, el JCP se dio cuenta de que era necesario que J2EE sufriera otro ciclo evolutivo. Con la presentación de J2EE 1.4 la comunidad Java ha reunido la tecnología J2EE con la tecnología de servicios web.

Un caso práctico de que la competencia empresarial es cada vez más cerrada, es el caso de la empresa de capacitación “Edutecsa”, la cual se dedica a impartir cursos de informática donde enseñan a manejar herramientas de última generación para poder realizar sistemas a la medida. Para poder sobrevivir esta empresa en la competencia empresarial, no solo es necesario otorgar los cursos de tecnología de punta y con la mayor calidad posible, es necesario tener una herramienta que sea capaz de hacer llegar la información de los servicios que proporciona la empresa a el mayor número de clientes posibles, con la certeza de

que esta herramienta sea amigable, confiable, robusta, eficiente, que tenga alta disponibilidad y que sea altamente accesible para que esté al alcance de todos.

Por lo anterior se busca como objetivo, lograr un buen análisis, diseño, desarrollo e implementación de aplicaciones empresariales utilizando la metodología de desarrollo adecuada, patrones de diseño y el lenguaje de programación Java, que nos ayudarán a cumplir los objetivos de tiempo, costo, dinero y calidad que actualmente nos exige la competencia empresarial, para que las empresas como el caso de “Edutecsa” (nuestro usuario), se sienta cómodo, que el software le apoyó en la solución de sus problemas y que no se rezague en la competencia del mundo empresarial.

El presente trabajo se basa en el diplomado “Java Master”, impartido en la Universidad Nacional Autónoma de México, FES Aragón y se enfoca al caso práctico de la empresa “Edutecsa”, en donde se mostrará todo lo que conlleva la implementación de un sistema. Esto es, empezando con la metodología a utilizar para el desarrollo de este (donde se explica la metodología de análisis y diseño orientado a objetos), seguido de cómo se puede integrar el lenguaje de modelado unificado (UML) en el análisis del sistema y una breve descripción de los patrones de diseño. Posteriormente en el desarrollo del sistema se muestra lo que es el lenguaje de programación Java, desde los conceptos básicos del paradigma orientado a objetos hasta lo que es la versión empresarial de dicho lenguaje. Todo esto en conjunto será lo utilizado para poder llevar a cabo el caso práctico de la empresa de capacitación “Edutecsa”.

En conclusión el principal objetivo de este trabajo es analizar, diseñar y desarrollar un sistema para el centro de capacitación, que además de que sea confiable, robusto, amigable y que tenga alta disponibilidad, sea un sistema que le ayude al usuario a cubrir todos sus requerimientos para que pueda conseguir su objetivo de una manera más fácil, esto es, que sea funcional. Todo esto, sin perder de vista que este trabajo está basado en el diplomado impartido en la FES Aragón, y que por lo tanto muestra en gran parte el contenido dicho diplomado, que a grandes rasgos mostró el lenguaje de modelado unificado UML y el lenguaje de programación Java en su edición estándar y en la empresarial.

El trabajo realizado se desarrolla en el presente documento de la siguiente manera:

En el capitulo 1, se expone y describe la metodología de análisis y diseño orientado a objetos (OOA&D), seguido de cómo se puede complementar el lenguaje de modelado unificado (UML) y algunos patrones de diseño para facilitar el desarrollo del sistema.

En el capitulo 2, se expone lo que es el lenguaje de programación Java edición estándar, desde los conceptos básicos de lo que es la programación orientada a objetos, pasando por las características de Java, sus palabras claves, tipos de datos y arreglos, hasta sentencias de control de flujo y algunos paquetes.

En el capitulo 3, se describe el lenguaje de programación Java en su edición empresarial, se exponen los componentes básicos que componen a la edición empresarial de este lenguaje, tales como, “JDBC”, “servlets”, “beans” y “JSP”, entre otros.

En el capitulo 4, se expone el caso práctico, que es el análisis, diseño, desarrollo e implementación del sistema de automatización del proceso de preinscripción de cursos vía web, desarrollado para el centro de capacitación Edutecsa, también se presentan las conclusiones obtenidas con el desarrollo de este trabajo.

CAPÍTULO I

Análisis y diseño orientado a objetos con UML

1.1 Introducción al análisis y diseño orientado a objetos

La metodología de análisis y diseño orientado a objetos (OOA&D) es un proceso que contiene una notación soportada para la definición de software, basada en el concepto de "objeto" que combina la estructura y comportamiento dentro de una simple entidad. Este proceso no es necesariamente lineal, esto es, puede ser repetitivo. La notación de esta metodología es una definición visual común que permite a la gente compartir los conocimientos acerca del sistema.

La metodología de análisis y diseño orientado a objetos (OOA&D) consiste en tres partes:

- Un proceso o actividades paso por paso que son usadas para modelar o construir el software.
- Una notación o una representación, generalmente gráfica, de los subsistemas que componen al sistema y describe como interactúan entre ellos.
- Un conjunto de reglas que describen cómo puede trabajar el sistema.

Esta metodología se centra en la fase de análisis y en la fase de diseño de un sistema.

La fase de análisis se enfoca en:

- Establecer una clara visión de la forma en que trabaja la empresa (reglas del negocio).
- Resaltar las tareas del sistema que deben mejorar.
- Desarrollar un vocabulario común para el problema de negocio.
- Destacar las mejores soluciones para el problema de negocio.

La fase de diseño se encarga de:

- Resolver el problema de negocio.
- Definir el como se va a resolver.
- Introducir los elementos necesarios que harán que el sistema funcione.
- Definir una estrategia de implementación para el sistema.

El análisis y diseño orientado a objetos como su nombre lo indica, está dirigida a la programación orientada a objetos. Para esto habrá que definir los conceptos de objetos y clases.

Objeto: los objetos son los bloques construidos de la programación orientada a objetos (POO[1]), son elementos únicos, singulares, sencillos o complejos, pueden ser reales o imaginarios y tienen atributos y operaciones.

Clase: las clases son las definiciones o modelos de los objetos, todos los objetos son instanciados o creados a partir de clases.

Esta metodología propone dividir el proyecto en una serie de mini proyectos, utilizando el lenguaje de modelado unificado (UML[2]) como su notación gráfica y el proceso de desarrollo de software en forma iterativa e incremental.

Los proyectos pueden ser desarrollados utilizando una metodología iterativa e incremental, es decir, repitiendo los pasos del ciclo de vida cuantas veces se necesite (iterativa) y que cada iteración de los pasos vaya creciendo hasta cubrir cada paso del proyecto. Por lo que en cada iteración de las fases de un desarrollo se debería considerar lo siguiente:

- Seleccionar y analizar los casos de uso relevante.
- Crear y diseñar usando la arquitectura escogida (Unix, Linux, Windows, etc.).
- Implementar el diseño en componentes.
- Verificar que los componentes satisfagan los casos de uso.

1.1.1 Fases del ciclo de vida iterativo e incremental:

Inicio.- consiste en echar a andar el proyecto, establecer los requerimientos del sistema, definir las reglas de negocio, identificar los riesgos y definir el alcance.

Elaboración.- esta fase se enfoca en el análisis y diseño del sistema, establecer la línea arquitectónica en la que se va a desarrollar, monitorear los riesgos críticos y crear un plan de trabajo para llegar a las metas del proyecto.

Construcción.- básicamente la fase de construcción va a consistir en la programación del sistema.

Transición.- en la fase de transición se va a introducir la versión beta del sistema, se van a realizar las pruebas de este, se va a implementar y se le va a dar mantenimiento.

[1] OOP (Object-oriented programming) un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento. Su uso se popularizó a principios de la década de 1990.

[2] UML: por sus siglas en inglés *Unified Modeling Language* es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, está respaldado por el OMG (Object Management Group).

Cada fase de esta metodología contiene el siguiente flujo:

- Requerimiento
- Análisis
- Diseño
- Desarrollo
- Pruebas
- Implementación

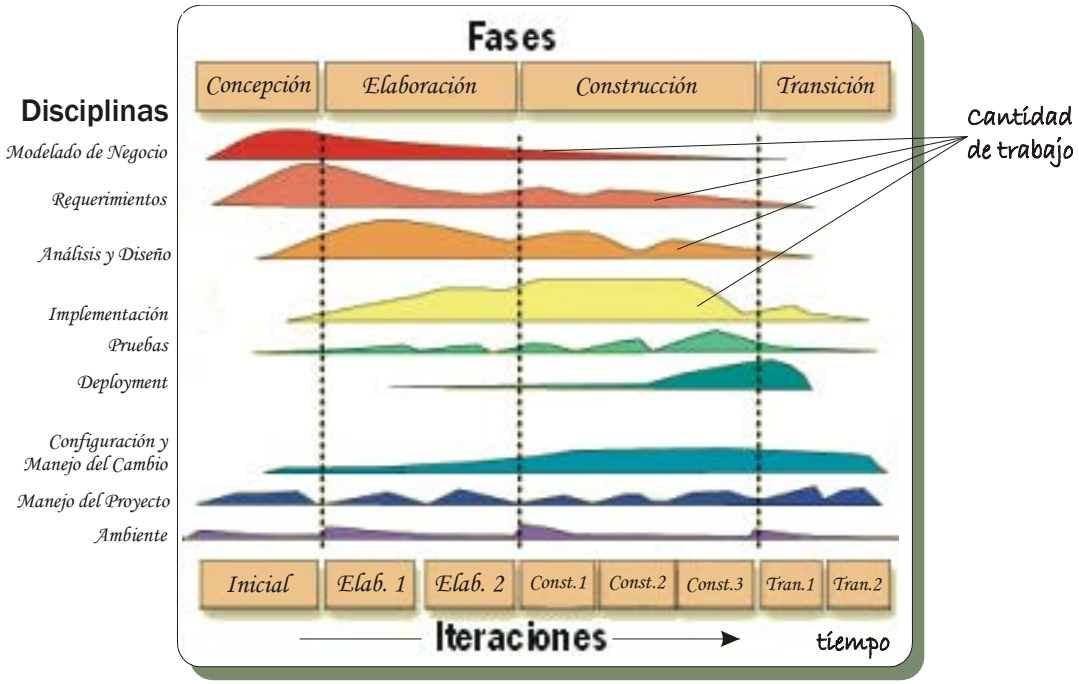


Figura 1.1.1. Fases de ciclo de vida interactivo e incremental.

Los beneficios de utilizar una metodología con un ciclo de vida iterativo e incremental son reducción de costos, mejor mantenimiento del plan de trabajo del proyecto, mejor administración del tiempo para el equipo de desarrollo y cubrir de forma activa las necesidades del cliente y/o cambios que se requieran.

1.1.2 Características de la programación orientada a objetos:

Abstracción: se refiere al concepto de ignorar los detalles para concentrar las características esenciales de los objetos.

Encapsulación: se refiere a esconder o proteger los atributos de un objeto de otros, permitiendo solamente su acceso a ellos mediante acciones u operaciones.

Asociación: se refiere a la manera en que se comunican los objetos entre si.

Agregación: se refiere al proceso que define a un objeto en términos de sus componentes, es un tipo de asociación. La agregación se puede definir por la frase “tiene un”.

Composición: es otro tipo de asociación y toma lugar cuando se tiene un objeto dentro de otro. La composición se puede distinguir por la frase “contiene un”.

Herencia: se refiere al mecanismo que define una nueva clase de una clase ya existente (la descendencia de clases padres a hijas). Se puede distinguir con la frase “es un” o “es del tipo de”.

Cohesión y acoplamiento: la cohesión es la forma en que un grupo de clases contribuyen a un propósito en común. El acoplamiento se refiere a la relación de las clases (dependencia).

Polimorfismo: se refiere a las diferentes formas en que los métodos van a ser aplicados a objetos de diferentes clases para llegar a un mismo resultado.

1.2 Requerimientos y análisis inicial

El primer paso a realizar en el proceso de análisis y diseño orientado a objetos es obtener la información referente al sistema a desarrollar, esta información se puede obtener de diversas fuentes como la especificación de los requerimientos iniciales del cliente, expertos en el tema, clientes o usuarios, administradores, mercadotecnia y hasta de proyectos anteriores.

Por otro lado también se tiene que tomar en cuenta el evitar suposiciones que tradicionalmente hacemos, como que los requerimientos nunca cambian, que los usuarios saben más que los desarrolladores, que uno puede construir la solución correcta en la primera vez o que los proyectos no van a evolucionar. Para evitar estas suposiciones, podemos identificar lo más claro posible los requerimientos del usuario, asegurar que el modelo que se realice pueda adaptarse a una posible evolución de los requerimientos y asegurar que el modelo que se realice pueda corregirse si hay deficiencias en la comprensión del proyecto.

Generalmente cuando se empieza un proyecto se puede realizar un documento del problema o requerimientos del proyecto (Problem Statement). Este documento debe especificar claramente lo siguiente:

- Toda la información que es pertinente para el análisis y diseño.
- Posibles riesgos, tales como la información que arrojan sistemas externos, o hasta el hardware.
- Usuarios que van a utilizar el sistema.

- Las entradas y salidas de la red donde se va a implementar el sistema.

Dicho documento se compone de lo siguiente:

Definición del problema.- la definición del problema puede ser gráfica o textual, el cual describe que áreas y problemas cubrirán el nuevo sistema.

Clases y objetos candidatos.- se refiere a identificar las clases y los objetos candidatos a clases de la definición del problema. Una manera de identificar estos, es subrayando los nombres o sujetos de la definición del problema.

Diccionario de datos.- es un documento que describe el vocabulario a usar en un proyecto específico. Este es importante en grandes proyectos, donde los equipos de trabajo pueden jugar diferentes roles de manera conjunta, y así descartar dudas sobre el proyecto. Los elementos más comunes de un diccionario de datos son nombres de productos o procesos, definiciones, sinónimos, tipo de datos y tamaño, valores válidos, mapeo de tablas de base de datos, entre otras.

Otra parte fundamental en el análisis inicial es la declaración de riesgos. Un riesgo es un elemento de un proyecto de software que amenaza con la terminación exitosa del mismo. Las principales áreas de riesgos en un proyecto son:

- Riesgos de requerimientos
- Riesgos tecnológicos
- Riesgos de conocimientos
- Riesgos de recursos
- Riesgos políticos

1.3 Introducción al Lenguaje de Modelado Unificado (UML)

El Lenguaje de Modelado Unificado (UML) es un lenguaje gráfico para especificar, construir visualizar y documentar los componentes de un sistema de software, este lenguaje usa un lenguaje de notación gráfica para representar a los componentes que son parte de un OOA&D.

Se debe recalcar que el Lenguaje de Modelado Unificado, no es un proceso, ni define un proceso, si no que es un lenguaje que se utiliza como herramienta en la metodología de OOA&D como apoyo para definir y analizar procesos claramente.

Los diagramas del UML se dividen básicamente en 2 grupos:

- **Modelo estático**.- los diagramas del modelo estático construyen y documentan los aspectos estáticos de un sistema y refleja la estructura estática del sistema, también crean una representación de los elementos principales del sistema a analizar. Este modelo contiene los siguientes tipos de diagramas:
  - Diagramas de caso de uso
  - Diagramas de clase
  - Diagramas de objetos
  - Diagramas de componentes
  - Diagramas de despliegues

- **Modelo dinámico**.- los diagramas del modelo dinámico ayudan a visualizar el comportamiento del sistema, estos son:
  - Diagramas de secuencia
  - Diagramas de colaboración
  - Diagramas de transición de estado
  - Diagramas de actividad

- **Otros elementos**.- otros elementos importantes en UML son:
  - Notación de paquetes
  - Mecanismos de extensión de UML
    - Comentarios (notas) en UML
    - Estereotipos
    - Valores
    - Restricciones

1.4 Análisis de modelos estáticos

El modelado de un sistema implica identificar los componentes que son importantes desde un cierto punto de vista. Estas cosas forman el vocabulario del sistema que se está modelando.

La fase de análisis de modelos estáticos, identifica los objetos requeridos en tiempo de ejecución para asegurar la funcionalidad del sistema. La fase de análisis de un proyecto sigue los requerimientos levantados en el ciclo de vida de este y utiliza los casos de uso, posteriormente le sigue la fase de diseño del sistema, y se enfoca a lo que debe hacer el sistema, esto es, a la funcionalidad de este, por lo tanto debe identificar los objetos que son candidatos a ser clases, esto se puede hacer realizando las siguientes preguntas:

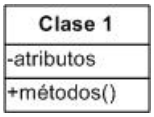
- ¿Cómo son los objetos que se relacionan con otros?
- ¿Cuál es la responsabilidad o función de cada objeto?
- ¿Cómo funcionan los objetos para relacionarse con otros?

1.4.1 Clases del sistema

Una vez identificados los objetos que son candidatos a clases de la definición del problema, podemos sacar las clases del sistema obteniéndolas de la lista de los objetos candidatos, los cuales representen las funcionalidades principales del sistema y su interacción con otros objetos.

Ya obtenidas las clases del sistema, las podemos representar con el lenguaje de modelado unificado (UML) con dos tipos de diagramas del modelo de objetos, el diagrama de clases y el diagrama de objetos, el primero muestra las clases que tu debes escribir para crear un sistema y deben mostrar todas las posibles relaciones entre las clases, el segundo representa los objetos actuales que se encuentran en el sistema y describe las relaciones explicitas en casos específicos.

Ambos diagramas toman el mismo formato básico y la misma sintaxis y pueden ser trabajados simultáneamente, aunque usualmente el diagrama de clases es desarrollado antes del diagrama de objetos. Sus notaciones en UML son las siguientes:



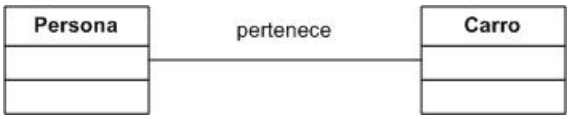
La clase es representada por un rectángulo en posición horizontal con tres divisiones, la primera división contiene el nombre de la clase escrita con la primera letra en mayúscula, la segunda por lo general contiene los atributos de la clase y la tercera contiene la lista de métodos de la clase.

La representación de un objeto en la notación UML es muy parecida a la de las clases, es un rectángulo en posición horizontal, pero sin divisiones, aunque también es válido ponerle atributos al objeto. El nombre del objeto se escribe en minúscula y subrayado, ya que un objeto es una instancia de una clase.

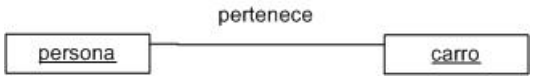
1.4.2 Asociaciones y ligas

Los diagramas de clases muestran clases que son usadas dentro de un sistema para construir objetos que corren el sistema. Pero el diagrama debe mostrar más que las clases por sí mismas, este debe mostrar la manera en que las clases se relacionan entre si. Estas relaciones también se pueden representar en el diagrama de objetos.

Para representar la relación entre dos clases en un diagrama de clases, se hace dibujando una línea que va de una clase a otra, esta línea se llama "asociación" y puede ser vertical u horizontal. El siguiente diagrama es un ejemplo:

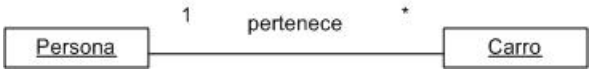


Por otro lado cuando se representa la relación entre dos objetos en un diagrama de objetos, se realiza de la misma manera que en el diagrama de clases, mediante un vínculo o línea de un objeto a otro, a este vínculo o línea se le llama "liga".



1.4.3 Asociaciones y multiciplidad

Los diagramas de objetos muestran las relaciones exactas entre objetos, aunado a esto, también se puede representar la cantidad de relaciones que puede tener un objeto a otro o a muchos objetos. El número exacto de relaciones entre objetos es muy importante, ya que representa el número de instancias de un objeto en el sistema, y por lo tanto determina la manera en que la información va a fluir en el sistema. El siguiente diagrama es un ejemplo de una asociación múltiple.



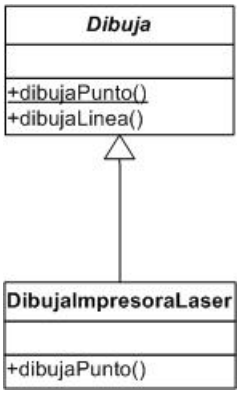
Cuando se tiene una relación múltiple de muchos a muchos, se dice que es una relación compleja y esta se tiene que resolver con una tabla de rompimiento o de asociación como se muestra a continuación:

1.4.4 Herencia

La herencia es una de las cualidades del paradigma orientado a objetos, esta muestra como los atributos y la funcionalidad pueden ser compartidas entre clases que tengan similar naturaleza. Esta puede ser simple o múltiple herencia y se representa de la siguiente forma:

1.4.5 Clases abstractas

Las clases abstractas son clases que contienen funcionalidad incompleta y no pueden ser instanciadas en el sistema. Cuando una clase extiende de una clase abstracta, hereda todos sus métodos, incluyendo a los abstractos, y estos deben ser sobrescritos en la clase que hereda de la abstracta. En notación de UML se representa simplemente con el nombre de la clase en letra “cursiva”, ejemplo:



1.4.6 Diagramas

Cuando se modela algo, se crea una simplificación de la realidad para comprender mejor el sistema que se está desarrollando. Con UML se construyen modelos a partir de bloques de construcción básicos, tales como clases, interfaces, colaboraciones, componentes, nodos, dependencias, generalizaciones y asociaciones.

Los diagramas son los medios para ver estos bloques de construcción. Un diagrama es una presentación gráfica de un conjunto de elementos, que la mayoría de las veces se dibuja como un grafo conexo de nodos (elementos) y arcos (relaciones). Los diagramas se utilizan para visualizar un sistema desde diferentes perspectivas. Como ningún sistema puede ser comprendido completamente desde una única perspectiva, UML define varios diagramas que permiten centrarse en diferentes aspectos del sistema independiente.

Los buenos diagramas hacen comprensible y accesible el sistema. La elección del conjunto adecuado de diagramas para modelar un sistema obliga a plantearse las cuestiones apropiadas sobre el sistema y ayuda a clarificar las implicaciones de las decisiones.

1.4.7 Diagramas de caso de uso

Ningún sistema se encuentra aislado. Cualquier sistema interesante interactúa con actores humanos o mecánicos que lo utilizan con algún objetivo y que esperan que el sistema funcione de forma predecible. Un caso de uso especifica el comportamiento de un sistema o de una parte del mismo, y es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecutan un sistema para producir un resultado observable de valor para un actor.

Los casos de uso se emplean para capturar el comportamiento deseado del sistema en desarrollo, sin tener que especificar como se implementa ese comportamiento. Estos proporcionan un medio para que los desarrolladores, los usuarios finales del sistema y los expertos del dominio lleguen a una compresión común del sistema mientras evoluciona a lo largo del desarrollo. Conforme se desarrolla el sistema, los casos de uso son realizados por colaboraciones, cuyos elementos cooperan para llevar a cabo cada caso de uso.

Los diagramas de caso de uso muestran quién o qué usa el sistema y sus características, esto es, muestra la relación de quienes o que componentes estan relacionados con los procesos del sistema, estos procesos son los casos de uso y pueden describirse en forma general, y a su vez estos se pueden detallar en otros casos de uso hasta el nivel de detalle que se quiera. Los elementos básicos de un diagrama de caso de uso son:

- **Actor.**- identifica quién o qué usa el caso de uso, es representado con un “monito”.
- **Caso de Uso**.- es la característica requerida o proceso de un sistema.
- **<<extend>>**.- estereotipo que muestra como va a extender el caso de uso.
- **Generalización**.- es representada con una flecha en blanco.
- **<<include>>**.- estereotipo que muestra como se incorpora un comportamiento a un caso de uso.

El siguiente diagrama es un ejemplo de un diagrama de caso de uso:

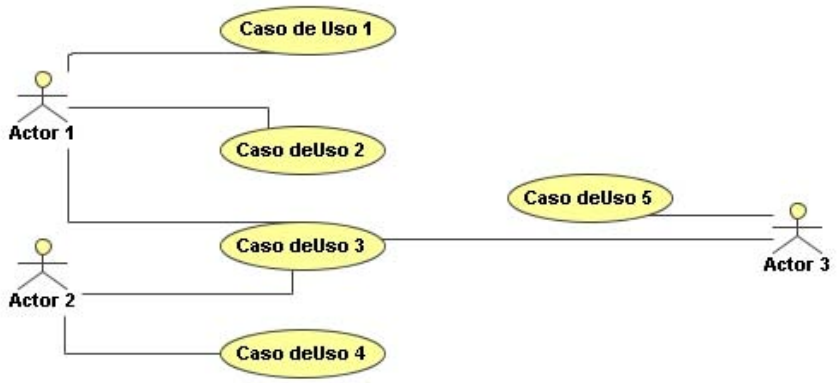


Figura 1.4.1 Diagrama de caso de uso.

1.4.8 Diagramas de clases

Estos diagramas son un conjunto de objetos que comparten atributos y características en común. Estos diagramas estructurales muestran conjuntos de clases de interfaces, colaboraciones entre objetos y sus relaciones. Los diagramas de clases se componen de una o más secciones de rectángulos, las cuales representan:

- Tipo (nombre de la clase, es obligatorio)
- Atributos (opcional)
- Métodos (opcional)

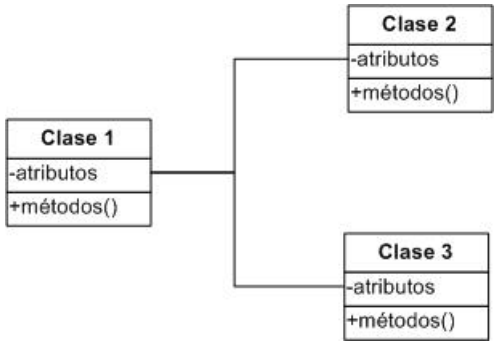


Figura 1.4.2 Diagrama de clases.

1.4.9 Diagramas de objetos

Representan a objetos en específico que existen como parte del espacio real de un proyecto o sistema. Su estructura muestra a un conjunto de objetos y sus relaciones. El siguiente diagrama es un ejemplo de un diagrama de objetos.

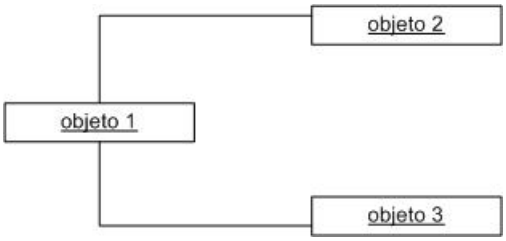


Figura 1.4.3 Diagrama de objetos.

1.4.10 Diagramas de componentes

Muestran las relaciones entre los componentes de software. El siguiente diagrama lo ejemplifica:

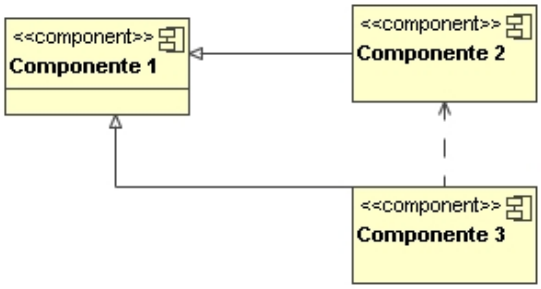


Figura 1.4.4 Diagrama de componentes.

1.4.11 Diagramas de despliegue

Los diagramas de despliegue o entrega muestran los dispositivos físicos que pueden interactuar en la entrega de las aplicaciones de software. El siguiente diagrama es un ejemplo.

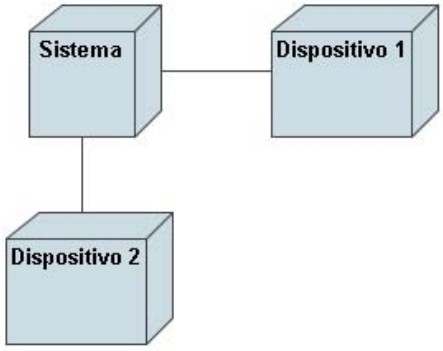


Figura 1.4.5 Diagrama de despliegue.

1.5 Análisis de modelos dinámicos

En cualquier sistema interesante, los objetos interactúan entre sí intercambiando mensajes. Una interacción es un comportamiento que incluye un conjunto de mensajes intercambiados por un conjunto de objetos dentro de un contexto para lograr un propósito.

Las interacciones se utilizan para modelar los aspectos dinámicos de las colaboraciones que representan sociedades de objetos que juegan roles específicos, y colaboran entre sí para llevar a cabo un comportamiento mayor que la suma de los comportamientos de sus elementos. Las interacciones bien estructuradas son como los algoritmos bien estructurados: eficientes, sencillos, adaptables y comprensibles.

En UML, los aspectos dinámicos de un sistema se modelan mediante interacciones. Al igual que un diagrama de objetos, una interacción tiene una naturaleza estática, ya que establece el escenario para un comportamiento del sistema introduciendo todos los objetos que colaboran para realizar alguna acción. Pero a diferencia de los diagramas de objetos, las interacciones incluyen los mensajes enviados entre objetos. La mayoría de las veces, un mensaje implica la invocación de una operación o el envío de una señal; un mensaje también puede incluir la creación o la destrucción de objetos.

Interacción.- es un comportamiento que comprende un conjunto de mensajes intercambiados entre un conjunto de objetos dentro de un contexto para lograr un propósito.

Mensaje.- es la especificación de una comunicación entre objetos que transmite información, con la expectativa de que se desencadenará una actividad.

Enlace.- es una conexión semántica entre objetos. En general, un enlace es una instancia de una asociación, y esto hace que siempre que haya un enlace entre dos objetos, un objeto puede enviar un mensaje al otro.

La mayoría de las veces, las interacciones se utilizan con el propósito de modelar el flujo de control que caracteriza el comportamiento de un sistema, incluyendo casos de uso, patrones, mecanismos, o el comportamiento de una clase o una operación individual. Mientras que las clases, las interfaces, los componentes, los nodos y sus relaciones modelan los aspectos estáticos del sistema, las interacciones modelan los aspectos dinámicos.

Cuando se modela una interacción, lo que se hace esencialmente es construir una representación gráfica de las acciones que tienen lugar entre un conjunto de objetos. Algunas técnicas tales como las tarjetas CRC[3] son particularmente útiles para ayudar a descubrir y pensar sobre tales interacciones. Para modelar un flujo de control:

- Hay que establecer el contexto de la interacción, si es el sistema global, una clase o una operación individual.

[3] Las tarjetas CRC son una metodología para el diseño de software orientado por objetos creada por Kent Beck y Ward Cunningham.

- Hay que establecer el escenario para la interacción, identificando qué objetos juegan un rol; se deben establecer sus propiedades iniciales, incluyendo los valores de sus atributos, estado y rol.
- Si el modelo destaca la organización estructural de esos objetos, hay que identificar los enlaces que los conectan y que sean relevantes para los trayectos de comunicación que tienen lugar en la interacción.
- Hay que especificar los mensajes que pasan de un objeto a otro mediante un orden temporal.
- Para expresar los detalles necesarios de la interacción, hay que adornar cada objeto con su estado y rol, siempre que sea preciso.

1.5.1 Diagramas de secuencia

Estos diagramas capturan los mensajes que ocurren entre diferentes objetos dentro de un periodo de tiempo. Un diagrama de secuencia es como un tipo de diagrama de interacción. Estos enfatizan el orden cronológico de los mensajes.

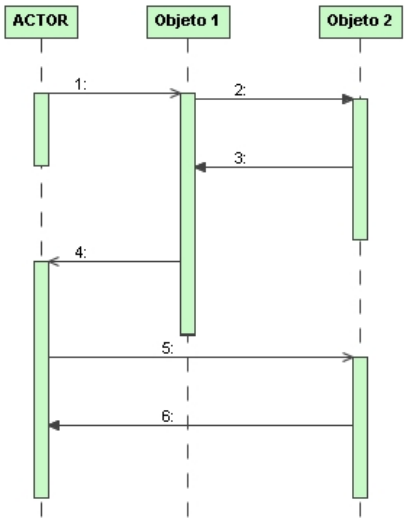


Figura 1.5.1 Diagrama de secuencia.

1.5.2 Diagramas de colaboración

Muestran la colaboración entre objetos en el sistema utilizando mensajes. Los diagramas de colaboración enfatizan la organización estructural de los objetos que envían y reciben mensajes.

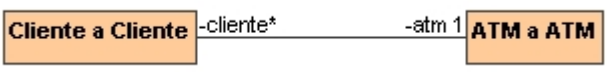


Figura 1.5.2 Diagrama de colaboración.

1.5.3 Diagramas de transición de estados

Los diagramas de transición de estados, o simplemente llamados de estados, muestran el estado de un dispositivo o sistema que enfatiza el comportamiento ordenado de eventos de un objeto. El siguiente diagrama es una muestra.

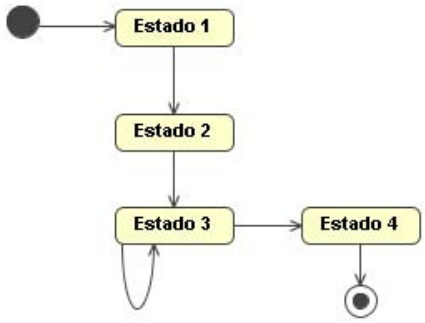


Figura 1.5.3 Diagrama de transición de estados.

1.5.4 Diagramas de actividades

Los diagramas de actividades describen el flujo de una actividad del sistema a otra, el siguiente diagrama es un ejemplo de un diagrama de actividades.

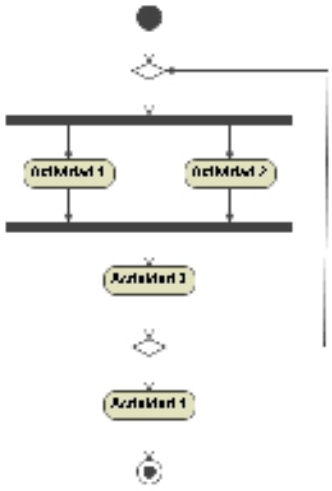


Figura 1.5.4 Diagrama de actividades.

1.6 Patrones

Todos los sistemas bien estructurados están llenos de patrones. Un patrón proporciona una solución común a un problema común en un contexto dado. Un mecanismo es un patrón de diseño que se aplica a una sociedad de clases; un "framework" es típicamente un patrón arquitectónico que proporciona una plantilla extensible para aplicaciones dentro de un dominio.

Los patrones se utilizan para especificar los mecanismos que configuran la arquitectura del sistema. Un patrón se puede hacer identificando de manera clara todos los elementos variables que puede ajustar un usuario para aplicarlo en un contexto particular.

Cada vez que se mira por encima del código fuente, se detectan ciertos mecanismos comunes que configuran la forma en que se organizan las clases y otras abstracciones. Por ejemplo, en un sistema dirigido por eventos, una forma común de organizar los manejadores de eventos es utilizar un patrón de diseño *Cadena de responsabilidad.* Si se mira por encima de estos mecanismos, aparecen estructuras comunes que configuran la arquitectura completa del sistema. Por ejemplo en los sistemas de información, la utilización de una arquitectura de tres capas es una forma común de lograr una separación de intereses clara entre la interfaz de usuario del sistema, la información persistente y los objetos y reglas de negocio.

Patrón.- es una solución común a un problema común en un contexto dado.
Mecanismo.- es un patrón de diseño que se aplica a una sociedad de clases.
Framework.- es un patrón arquitectónico que proporciona una plantilla extensible para aplicaciones dentro de un dominio.

1.6.1 Ventajas de los patrones de diseño

Conforme ha aumentado el poder de las computadoras, también han aumentado las expectativas de los usuarios. Esto ha ido a la par con las nuevas posibilidades que ofrecen las redes e Internet y la introducción de la informática a nuevas áreas de la actividad humana. Todo ello ha hecho que crezca la complejidad del software.

Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de manera frecuente en el desarrollo. Por tanto está basado en la recopilación del conocimiento de los expertos en desarrollo de software.

No deben verse los Patrones de Diseño como una teoría o una corriente. No es conveniente tratar de tomar partido por una u otra alternativa. Los patrones de diseño provienen de la experiencia real, probada y funcional. Son, de alguna forma, la historia del desarrollo de sistemas y en ese sentido nos ayudan a no cometer los mismos errores.

1.6.2 Características de los patrones de diseño

- **Son soluciones concretas**. Proponen soluciones a problemas concretos, no son teorías genéricas.
- **Son soluciones técnicas**. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.
- **Se utilizan en situaciones frecuentes**. Ya que se basan en la experiencia acumulada al resolver problemas reiterativos.
- **Favorecen la reutilización de código**. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar.
- **El uso de un patrón no se refleja en el código**. Al aplicar un patrón, el código resultante no tiene por que delatar el patrón o patrones que lo inspiró. No obstante últimamente hay múltiples esfuerzos enfocados a la construcción de herramientas de desarrollo basados en los patrones y frecuentemente se incluye en los nombres de las clases el nombre del patrón en que se basan, facilitando así la comunicación entre desarrolladores.
- **Es difícil reutilizar la implementación de un patrón**. Al aplicar un patrón, aparecen clases concretas que solucionan un problema concreto y que no será aplicable a otros problemas que requieran el mismo patrón. Sin embargo se pueden crear implementaciones genéricas que pueden servir de base para simplificar la construcción de implementaciones específicas, a estás soluciones genéricas se les conoce como “frameworks”.

1.6.3 Clasificación

Los Patrones de Diseño se clasifican generalmente de acuerdo a su propósito:

- **Patrones de creación**: Tratan la creación de instancias.
- **Patrones estructurales**: Tratan la relación entre clases, la combinación de clases y la formación de estructuras de mayor complejidad.
- **Patrones de comportamiento**: Tratan la interacción y cooperación entre clases.

1.6.4 Patrones de creación

Los patrones de creación abstraen la forma en que se crean los objetos, de forma que permite tratar las clases a construir de forma genérica, apartando la decisión de qué clases crear o como crearlas.

Según a donde quede ubicada dicha decisión se habla de Patrones de Clase (utiliza la herencia para determinar la creación de las instancias, es decir en los constructores de las clases) o Patrones de Objeto (es en métodos de los objetos creados donde se modifica la clase).

Nombre del patrón	Sinopsis	Referencia
Abstract Factory	Proporciona un contrato para la creación de familias de objetos relacionados o dependencias sin tener su clase concreta.	http://www.fluffycat.com/java/JavaNotes-GoFAbstractFactory.html http://patterndigest.com/patterns/AbstractFactory.html
Builder	Simplifica la creación de objetos complejos definiendo una clase cuyo propósito es construir instancias de otra clase. Aunque puede haber más de una clase en el producto, este constructor genera un producto principal por que siempre debe de existir un producto principal	http://www.fluffycat.com/java/JavaNotes-GoFBuilder.html http://patterndigest.com/patterns/Builder.html
Factory Method	Permite definir un método estándar para crear un objeto, además del constructor propio de la clase, si bien la decisión del objeto a crear se delega a las subclases.	http://www.fluffycat.com/java/JavaNotes-GoFFactoryMethod.html http://patterndigest.com/patterns/FactoryMethod.html
Singleton	Permite tener una sola instancia de una clase, a la vez que permite que todas las clases tengan acceso a esta instancia.	http://www.fluffycat.com/java/JavaNotes-GoFSingleton.html

Tabla 1.6.1 Patrones de creación.

1.6.5 Patrones estructurales

Estos patrones describen formas efectivas de partir y combinar los elementos de una aplicación. Las formas en que los patrones estructurales afectan a las aplicaciones varían enormemente; por ejemplo, el patrón “Adapter” permite que dos sistemas incompatibles se comuniquen, mientras que el patrón “Facade” permite presentar una interfaz simplificada a un usuario sin eliminar todas las opciones disponibles del sistema.

Nombre del patrón	Sinopsis	Referencia
Adapter (wraper)	Sirve como intermediario entre dos clases convirtiendo las interfaces de una clase para que pueda ser utilizada por otra.	http://www.fluffycat.com/java/JavaNotes-GoFAdapter.html http://patterndigest.com/patterns/ClassAdapter.html
Composite	Desarrolla una forma flexible de crear jerarquías en una estructura de árbol de una complejidad arbitraria, permitiendo a la vez que todos los elementos de la estructura funcionen con una interfaz uniforme.	http://www.fluffycat.com/java/JavaNotes-GoFComposite.html http://patterndigest.com/patterns/Composite.html
Proxy	Proporciona un representante de otro objeto, por distintas razones como puede ser el acceso, la velocidad o la seguridad.	http://www.fluffycat.com/java/JavaNotes-GoFProxy.html http://patterndigest.com/patterns/Proxy.html
Facade	Proporciona una interfaz simplificada para un grupo de subsistemas o un sistema complejo.	http://patterndigest.com/patterns/Facade.html http://www.fluffycat.com/java/JavaNotes-GoFAbstractFactory.html

Tabla 1.6.2 Patrones de estructurales.

1.6.6 Patrones de comportamiento

Los patrones de comportamiento están relacionados con el flujo de control en un sistema. Ciertas formas de organizar un sistema pueden derivar en grandes beneficios para la eficiencia y mantenimiento del sistema. Los patrones de mantenimiento destilan la esencia de estas prácticas, cuya utilidad ha sido probada en heurísticas conocidas, fácilmente comprensibles y aplicables.

Nombre del patrón	Sinopsis	Referencia
Chain of responsability	Establece una cadena en un sistema para que un mensaje pueda ser manejado en el nivel en que se recibe en primer lugar o ser redirigido a un objeto que pueda manejarlo.	http://www.fluffycat.com/java/JavaNotes-GoFChainOfResponsibility.html http://patterndigest.com/patterns/ChainOfResponsibili.html
Observer	Proporciona a los componentes una forma flexible de enviar mensajes de difusión a los receptores interesados.	http://www.fluffycat.com/java/JavaNotes-GoFObserver.html http://patterndigest.com/patterns/Observer.html
Strategy	Define un grupo de clases que representan un conjunto de posibles comportamientos. Estos comportamientos pueden ser fácilmente intercambiados en una aplicación. Modificando la funcionalidad en cualquier instante.	http://www.fluffycat.com/java/JavaNotes-GoFStrategy.html http://patterndigest.com/patterns/Strategy.html

Tabla 1.6.3 Patrones de comportamiento.

1.7 Patrones J2EE

1.7.1 Patrones de la capa de presentación

Los patrones de esta capa permiten manejar la interacción con el sistema, así como la captura y presentación de información desde y hacia el usuario.

Nombre del patrón	Sinopsis	Referencia
Front Controler	Proporciona control centralizado para la gestión y manejo de la petición (http request).	http://www.corej2eepatterns.com/Patterns2ndEd/FrontController.htm http://www.programacion.com/java/tutorial/patrones/4/
View Helper	Encapsula lógica que no está relacionada al formato de la presentación en componentes Helper.	http://www.corej2eepatterns.com/Patterns2ndEd/ViewHelper.htm http://www.programacion.com/java/tutorial/patrones/5/

MVC (Model, Viewer Controller)	Este patrón consiste de tres clases principales: Model, Viewer y Controller. El model representa el modelo de negocio o datos. View es el despliegue del modelo o la interfaz gráfica. Controller coordina la vistas (views) y hace las llamadas respectivas al modelo para las consultas y guardado de la información.	http://www.bill.boyall.btinternet.co.uk/html/reference/development/java/UnderstandingJSPModel2architecture.htm http://www.ftponline.com/javapro/2002_05/magazine/columns/weblication/default.aspx http://www.ftponline.com/javapro/2002_06/online/servlets_06_11_02/default.aspx

Tabla 1.6.4 Patrones de capa de presentación.

1.7.2 Patrones de la capa de lógica de negocio

Estos patrones manejan el corazón de las reglas de negocio, proporcionando las interfaces para los componentes de servicio de negocio subyacentes, típicamente estos componentes son implementados con EJB's[4], también facilitan la interacción entre las distintas clases y servicios que integran esta capa. Típicamente permite atender las preocupaciones de persistencia, gestión transaccional y asignación de recursos.

Nombre del patrón	Sinopsis	Referencia
Business Delegate	Desacopla las capas de presentación y servicios, ocultando la complejidad técnica de localización de dichos servicios.	http://www.corej2eepatterns.com/Patterns2ndEd/BusinessDelegate.htm http://www.programacion.com/java/tutorial/patrones2/1/
Value Object (transfer object)	Facilita el intercambio de datos entre capas, reduciendo el diálogo y tráfico de red.	http://www.corej2eepatterns.com/Patterns2ndEd/TransferObject.htm http://www.programacion.com/java/tutorial/patrones2/4/
Session facade	Oculta la complejidad de los objetos y centraliza el manejo de work-flow.	http://www.corej2eepatterns.com/Patterns2ndEd/SessionFacade.htm http://www.programacion.

[4] Un Enterprise JavaBea (EJB) es un componente de la arquitectura J2EE cuya función principal es proporcionar la lógica de negocio.

		com/java/tutorial/patrones2/3/
Composite Entity	Representa una mejor práctica para el diseño de entity beans de baja granularidad por medio del agrupamiento de objetos con dependencia padre-hijo en un solo entity bean.	http://www.corej2eepatterns.com/Patterns2ndEd/CompositeEntity.htm http://www.programacion.com/java/tutorial/patrones2/7/
Value Object Asambler	Ensambla un Value Object compuesto de múltiples fuentes de datos.	http://www.corej2eepatterns.com/Patterns2ndEd/CompositeEntity.htm
Value List Handler	Maneja la ejecución de consultas que regresan varios registros, permitiendo la ejecución del query, paginación y cache del resultado.	http://www.corej2eepatterns.com/Patterns2ndEd/ValueListHandler.htm http://www.programacion.com/java/tutorial/patrones2/6/
Service Locator	Encapsula la complejidad de la localización y creación de los servicios de negocio. Utiliza el patrón factory para la creación de los servicios.	http://www.corej2eepatterns.com/Patterns2ndEd/ValueListHandler.htm http://www.programacion.com/java/tutorial/patrones2/2/
Business Object	Permite una aproximación orientada a objetos al modelo de negocio, centraliza el comportamiento y promueve el rehusó, evita duplicación de funciones y mejora el mantenimiento.	http://www.corej2eepatterns.com/Patterns2ndEd/BusinessObject.htm
Application Service	Centraliza la lógica de negocio, permitiendo descomponer la lógica compleja en capas de servicios. También evita que los objetos de negocio contengan demasiado código para la lógica de negocio que sea compleja.	http://www.corej2eepatterns.com/Patterns2ndEd/ApplicationService.htm

Tabla 1.6.5 Patrones de capa de lógica de negocio.

1.7.3 Patrones de la capa de persistencia

Es responsable de la comunicación con recursos externos y sistemas tales como bases de datos y aplicaciones legacy[5], esta capa usa JDBC, conectores J2EE o algún producto de middleware[6] específico.

Nombre del patrón	Sinopsis	Referencia
Data Access Object	Abstrae las fuentes de datos, proporciona acceso transparente a los datos.	http://www.corej2eepatterns.com/Patterns2ndEd/DataAccessObject.htm http://www.programacion.com/java/tutorial/patrones2/8/

Tabla 1.6.6 Patrones de capa de persistencia.

[5] Es una aplicación basada en tecnologías y hardware viejos, tal como mainframe, la cual continúa brindando servicios esenciales para una organización.

[6] Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red).

CAPÍTULO II

Lenguaje de programación Java Edición Estándar (J2SE)

2.1 El origen de Java

2.1.1 Antecedentes

Java está relacionado con C++, que es un descendiente del lenguaje C. La mayor parte del carácter de Java está heredado de estos dos lenguajes. De C, Java deriva su sintaxis. La mayoría de las características orientadas a objetos están basadas en C++. De hecho, varias de las características de Java vienen de (o son respuesta a) sus predecesores. Por otra parte, la creación de Java estaba profundamente arraigada en el proceso de refinamiento y adaptación que se ha estado produciendo en los lenguajes de programación durante las últimas tres décadas.

A finales de los 80 y principios de los 90, el control lo tenía la programación orientada a objetos y C++. De hecho, durante algún tiempo parecía que los programadores habían encontrado finalmente el lenguaje perfecto. Debido a que C++ había mezclado la alta eficiencia y los elementos de estilo de C junto con el paradigma de la orientación a objetos, era un lenguaje que se podía utilizar para crear un amplio tipo de programas. Sin embargo, al igual que ocurrió en el pasado, surgieron fuerzas que pretendían, una vez más, la evolución hacia el nuevo lenguaje de programación. En pocos años, tanto la *World Wide Web* como Internet llegaron a la masa crítica. Este hecho provocó otra revolución en el mundo de la programación.

2.1.2 La creación de Java

Java fue concebido por James Gosling, Patrick Naughton, Chris Warth, Ed Frank y Mike Sheridan en Sun Microsystems Inc. en 1991. El desarrollo de la primera versión duró 18 meses. Este lenguaje se llamó inicialmente “Oak”, pero se le puso el nombre de “Java” en 1995. Entre la implementación inicial de Oak en el otoño de 1992 y el anuncio público de Java en 1995 mucha gente ha colaborado en el diseño y evolución del lenguaje. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin y Tim Lindholm han sido colaboradores en la madurez del prototipo original.

La motivación inicial de Java era un lenguaje que fuese independiente de la plataforma (es decir, con arquitectura neutral) que se pudiese utilizar para crear software para diversos dispositivos electrónicos, como hornos de microondas y controles remotos. El problema de C y C++ (y otros lenguajes) es que están diseñados para ser compilados para un destino específico. Aunque es posible compilar un programa C++ para cualquier tipo de CPU, para hacer esto es necesario tener el compilador de C++ correspondiente a ese tipo de CPU. El problema es que los compiladores son caros y se tarda en desarrollarlos. Por esta razón era necesaria una solución más sencilla y más eficiente en cuestión de

costos. En un intento por encontrar esta solución, Gosling y otros colaboradores comenzaron a trabajar en busca de un lenguaje portable y que fuese independiente de la plataforma, que se pudiese utilizar para producir código que se ejecutara en una serie de CPU's en diferentes entornos. Este esfuerzo condujo finalmente a la creación de Java.

Mientras se elaboraban los detalles de Java, apareció un segundo y más importante factor que jugaría un papel crucial en el futuro de Java, esta es la *World Wide Web* (la red mundial). Si el mundo Web no hubiese tomado forma a la vez que se estaba implementando Java, Java podría haber sido un lenguaje útil pero oscuro que se habría dedicado a la programación de dispositivos electrónicos para el consumo. Sin embargo, con la aparición de la World Wide Web, Java ha sido impulsado frente al diseño de los lenguajes de programación, ya que la red también exigía programas portables.

En 1993, a los miembros del equipo que estaba diseñando Java les pareció obvio que los problemas de portabilidad que habían encontrado al crear el código para los controladores eran similares a los que surgían cuando se intentaba crear código para Internet. De hecho, el mismo problema que intentaba resolver Java a pequeña escala estaba en Internet a gran escala. Este hecho hizo que Java cambiara su orientación y se dedicara a la programación en Internet en lugar de ser utilizado para programar dispositivos electrónicos de consumo. Por lo tanto, aunque la chispa inicial la proporcionó el deseo que había que tener un lenguaje de programación que fuese independiente de la plataforma, el éxito a gran escala de Java se ha conseguido gracias a Internet.

Debido a similitudes que existen entre Java y C++, es fácil pensar en Java como la "versión para Internet de C++". Sin embargo, pensar eso sería un gran error. Java tiene importantes diferencias teóricas y prácticas. Aunque es cierto que Java se basa en C++, no es una versión mejorada de C++. Por ejemplo, no es compatible de ninguna forma con C++. Por otro lado, Java no ha sido diseñado para sustituir a C++. Java ha sido diseñado para resolver una serie de problemas. C++ fue diseñado para resolver otros problemas diferentes. Ambos lenguajes coexistirán durante los próximos años.

2.1.3 Internet y Java

Internet le ha ayudado a Java a situarse como líder de los lenguajes de programación y por su lado, Java ha tenido un profundo efecto sobre Internet. La razón de esto es bastante simple: Java amplía el universo de objetos que pueden moverse libremente por el ciberespacio. En una red hay dos amplias categorías de objetos que se pueden transmitir entre un servidor y un cliente: información pasiva y programas activos y dinámicos. Por ejemplo, cuando uno lee un correo electrónico está viendo datos pasivos. Incluso cuando se transfiere un programa, el código del programa es información pasiva hasta que se ejecuta. Sin embargo, un segundo tipo de objetos que se pueden transmitir por la red: programas

dinámicos autoejecutables. Estos programas son agentes activos que se ejecutan sobre el cliente, a pesar de que haya sido el servidor el que los iniciara. Por ejemplo, el servidor podría proporcionar un programa para presentar correctamente los datos que está enviando.

Aunque este tipo de programas es bastante conveniente, estos programas también presentan serios problemas en cuestiones de seguridad y portabilidad. Antes de Java, el ciberespacio estaba cerrado a la mitad de las entidades que ahora viven allí. Como veremos, Java controla esas cuestiones y abre la puerta a un nuevo y excitante tipo de programas: los *applets*[1].

2.2 El lenguaje Java

2.2.1 Características de Java

Como ocurre en todos los lenguajes de programación, los elementos de Java no existen aisladamente, trabajan entre sí para formar el lenguaje como un todo. Sin embargo, esta interrelación puede hacer difícil un aspecto de Java sin involucrar a los otros. Frecuentemente, la explicación de un aspecto implica el conocimiento previo de otro. Por esto, es importante mencionar las características principales de Java.

Simple

Java fue diseñado para que los programadores profesionales lo pudiesen aprender de forma fácil y lo utilizaran de forma efectiva. Si ya se comprende los conceptos básicos de la programación orientada a objetos, aprender Java resultará aún más fácil debido a que Java hereda la sintaxis de C/C++ y muchas de las características orientadas a objetos de C++.

Orientado a objetos

Aunque tiene influencias de sus predecesores, java no fue diseñado para tener un código fuente compatible con otros lenguajes. Esto le proporcionó al equipo de Java la libertad para diseñar un lenguaje partiendo desde cero. Un resultado de esto fue una aproximación limpia, útil y pragmática a los objetos.

Robusto

El entorno multiplataforma de la red le exige bastante a un programa, ya que tiene que ejecutarse en una gran variedad de sistemas. Por esta razón, la capacidad de crear programas robustos tuvo prioridad en el diseño de Java. Por esto, Java le restringe en unas cuantas áreas para obligarle a encontrar pronto los

[1] Un applet es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web.

errores que se producen en el desarrollo de un programa. A la vez, Java le libera de tener que preocuparnos por muchas de las principales causas de error en la programación. Como Java es un lenguaje fuertemente tipado, le permite comprobar el código en tiempo de compilación. Sin embargo, también comprueba el código en tiempo de ejecución.

Multihilo

Java proporciona la programación Multihilo, que permite escribir programas que hacen varias cosas a la vez. El intérprete de java dispone de una elegante aunque sofisticada solución para sincronizar múltiples procesos que hace posible construir fácilmente sistemas interactivos.

Arquitectura neutral

Las actualizaciones del sistema operativo, de los procesadores y los cambios en los recursos básicos del sistema pueden provocar que un programa deje de funcionar correctamente. Los diseñadores de Java tomaron varias decisiones difíciles en el lenguaje Java y en el intérprete para intentar cambiar esta situación. Su principal objetivo era “escribir una vez, ejecutar en cualquier sitio, en cualquier momento y para siempre”. Y en gran parte este objetivo se ha logrado.

Interpretado y de alto rendimiento

Java permite la creación de programas que se pueden ejecutar en varias plataformas compilando en una representación intermedia llamada código binario (J*ava bytecode*). Este código se puede interpretar en cualquier sistema que tenga un intérprete de Java (Máquina Virtual de Java).

Distribuido

Java fue diseñado para el entorno distribuido de Internet, por lo que trabaja con el protocolo TCP/IP. De hecho, el acceso a un recurso utilizando un URL, no es muy diferente de acceder a un archivo. La versión original de Java (*Oak*) incluía facilidades para que los objetos pudiesen ejecutar procedimientos de forma remota. Java dispone de estas interfaces en un paquete de *invocación de método remoto* (*RMI, Remote Method Invocation*). Esta característica le aporta un nuevo nivel de abstracción en la programación cliente/servidor.

Dinámico

Los programas de Java se transportan con cierta cantidad de información que se utiliza para verificar y resolver los accesos a los objetos en tiempo de ejecución. Esto permite enlazar dinámicamente el código de una forma segura y conveniente, lo que es crucial para la solidez del entorno de los *applets*, donde

pequeños fragmentos de código binario pueden ser actualizados dinámicamente en un sistema que está ejecutándose.

2.2.2 Los tres principios de la programación orientada a objetos

Los lenguajes de programación orientada a objetos proporcionan mecanismos que ayudan a implementar el modelo orientado a objetos. Estos mecanismos se llaman encapsulado, herencia y polimorfismo.

Encapsulado

Es el mecanismo que permite juntar el código y los datos que manipula, ambos, alejados de posibles interferencias o usos indebidos. El encapsulado es como un envoltorio protector que evita que otro código que está fuera pueda acceder arbitrariamente al código o a los datos. El acceso al código y a los datos se realiza de forma controlada a través de una interfaz bien definida. El poder del código encapsulado es que todo el mundo conoce como acceder a él y pueden utilizarlo independientemente de los detalles de implementación (y sin temor a efectos laterales inesperados).

En Java la base del encapsulado es la clase. Una clase define la estructura y el comportamiento que serían compartidos por un conjunto de objetos. Cada objeto de una clase dada contiene la estructura y el comportamiento definidos por la clase, como si fuera grabado por un molde con la forma de la clase. Por esta razón, a los objetos se les llama a veces “instancias de una clase”.

Herencia

La herencia es el proceso mediante el cual un objeto adquiere las propiedades de otro. Esto es importante ya que así se consigue la clasificación jerárquica. Si no se utilizaran las jerarquías, cada objeto debería definir todas sus características explícitamente. Sin embargo, utilizando la herencia, un objeto sólo necesita definir aquellas cualidades que lo hacen único dentro de su clase. Este objeto puede heredar los atributos generales de su padre. Por lo tanto, la herencia es el mecanismo que le permite a un objeto ser una instancia específica de un caso más general.

Polimorfismo

El polimorfismo es una característica que le permite a una interfaz ser utilizada por una clase general de acciones. La acción específica se determina a partir de la naturaleza exacta de la situación.

De forma general, el concepto de polimorfismo se puede expresar con la frase “una interfaz, varios métodos”. Esto significa que es posible diseñar una interfaz genérica para un grupo de actividades relacionadas. Esto reduce la

complejidad permitiendo que la misma interfaz se utilice para especificar una clase general de acción.

2.2.3 Estructura de un programa

Una vez vistos los conceptos básicos de la orientación de objetos de Java, podemos empezar a crear un primer programa real de Java. A continuación se muestra la estructura básica de un programa de Java.

```
1  /*
2    Este es un programa de Java.
3  */
4  package mx.unam.ico;
5
6  import  java.util.*;
7
8  class Ejemplo{
9    // El programa comienza con una llamada a main().
10   public static void main(String args[]){
11        System.out.println("Este es un programa sencillo de
Java.");
12   }
13 }
```

En la mayoría de los lenguajes, el nombre de los archivos que contienen el código fuente de un programa es arbitrario. Sin embargo, este no es el caso de java. En java, el nombre que damos a los archivos fuentes es muy importante, ya que en java, un archivo fuente es llamado oficialmente unidad de compilación. Es un archivo fuente que contiene una o más definiciones de clase. El compilador de Java espera que estos archivos se almacenen con la extensión de archivo ".java".

La extensión del archivo tiene cuatro caracteres de longitud. Como se puede suponer, el sistema operativo debe ser capaz de aceptar nombres de archivos largos. Esto significa que DOS y Windows 3.1 no pueden soportar Java (por lo menos en este momento). Sin embargo, en Windows 95, NT, 2000, XP y Vista funciona correctamente.

En Java, todo el código tiene que estar dentro de una clase. Por convenio, el nombre de esa clase tiene que coincidir con el nombre del archivo que contiene el programa. También se debe asegurar que coincidan las mayúsculas, ya que Java es sensible al contexto. Así pues, el nombre del archivo fuente expuesto arriba, debe ser "Ejemplo.java".

Para compilar el programa "Ejemplo", ejecutamos el compilador "javac", dando el nombre del archivo fuente en la línea de comandos, como se muestra a continuación:

```
C:\mx\unam\ico>javac Ejemplo.java
```

El compilador “javac” crea un archivo llamado “Ejemplo.class” que contiene el código binario (*bytecode*) compilado del programa. Como ya se había comentado, el código binario es la representación inmediata del programa, que contiene las instrucciones que el intérprete de Java ejecutará.

Para ejecutar el programa, necesitamos usar el intérprete de Java (JVM), llamado “java”. Esto se hace pasándole el nombre de la clase “Ejemplo” como argumento en la línea de comandos, como se muestra a continuación:

```
C:\mx\unam\ico>java Ejemplo
```

Cuando se ejecuta el programa, se obtiene el siguiente resultado:

```
Este es un programa sencillo de Java.
```

Para entender un poco la sintaxis de Java, analizaremos cada parte del código del programa “Ejemplo”. De la línea 1 a la 3 es un comentario. Como muchos otros lenguajes de programación, Java permite introducir notas en el archivo fuente del programa. El contenido de un comentario es ignorado por el compilador. Este tipo de comentario esta delimitado por “/*” y “*/”, todo lo que esté dentro es un comentario, sin importar cuantas líneas sean. Existe otro tipo de comentario, es el de la línea 9, este comentario está precedido por “//” y solo aplica a una línea.

La línea 4 indica a qué paquete pertenece la clase, esto es, la ruta relativa donde se encuentra la clase y la línea 6 indica las clases o clase que se van a importar de otros paquetes para que sean utilizadas en el código, estas clases deben estar declaradas en la variable de ambiente llamada “CLASSPATH”.

La línea 8 utiliza la palabra reservada **class** para declarar que se va a definir una nueva clase. “Ejemplo” es un identificador que es el nombre de la clase. La definición de la clase completa, incluyendo todos sus miembros, estará entre la llave de apertura “{“ y la correspondiente llave de cierre “}”.

En la línea 10 comienza el método **main**(). Como el comentario precedente sugiere, esta es la línea en la que el programa comenzará a ejecutarse. Todas las aplicaciones Java comienzan su ejecución llamando a **main**() (igual que en C/C++). Para entender esta línea vemos que la palabra clave **public** es un especificador de acceso que permite que el programador controle la visibilidad de las clases miembros. En este caso, **main**() debe ser declarado como **public** ya que es llamado por código que está fuera de su clase cuando el programa comienza. La palabra clave **static** permite que **main**() sea llamado sin tener que referirse a una instancia particular de la clase. Esto es necesario hacerlo así ya que **main**() es llamado por el intérprete de Java antes de que se cree cualquier objeto. La palabra clave **void**, simplemente le indica al compilador que **main**() no

devuelve ningún valor. Por último, en el método **main**() hay un único parámetro, **String args[]** declara un parámetro **args**, que es una matriz de instancias de la clase **String**, esto es, si al momento de ejecutar el programa o la clase **Ejemplo** en la línea de comandos, le siguen algunos textos, estos van a ser los argumentos que la clase **Ejemplo** va a tomar como parámetros de entrada, como se muestra a continuación:

```
C:\mx\unam\ico>java Ejemplo argumento1 argumento2
```

Por último la línea 11 manda una instrucción al sistema a través del método **println**() para que se visualice en consola la cadena “Esto es un programa sencillo de Java.” La sentencia **println**() termina con punto y coma (;). Todas las sentencias en Java terminan con punto y coma. La razón por la que otras líneas del programa no terminaban en punto y coma es que ellas no son, técnicamente, sentencias.

2.2.4 Palabras claves de Java

Actualmente hay definidas 49 palabras clave reservadas en el lenguaje Java. Estas palabras clave, combinadas con la sintaxis de operadores y separadores constituyen la definición del lenguaje Java. Estas palabras clave no se pueden utilizar como nombres de variable, clases o métodos.

Las palabras clave **const** y **goto** son reservadas pero no se utilizan. En las primeras versiones de Java aparecían algunas otras palabras clave reservadas para posibles usos futuros. Sin embargo, la versión actual (versión 1.4) de Java sólo define las palabras claves que se muestran a continuación:

```
abstract   boolean    break      byte         case       catch
char       class      const      continue     default    do
double     else       extends    final        finally    float
for        goto       if         implements   import     instanceof
int        interface  long       native       new        package
private    protected  public     return       short      static
strictfp   super      switch     syncronized  this       throw
throws     transient  try        void         volatile   while
assert
```

2.3 Tipos de datos, variables y matrices

Es importante afirmar que java es un lenguaje fuertemente tipado. De hecho, parte de la seguridad y robustez de Java viene de este hecho. Esto es, que en primer lugar, cada variable tiene un tipo, cada expresión tiene un tipo y cada tipo está definido estrictamente. En segundo lugar, en todas las asignaciones, bien sean explicitas o a través de paso de parámetros en llamadas a método, se comprueba la compatibilidad de los tipos. No hay ninguna conversión automática entre tipos en la que haya un conflicto, como ocurre en otros lenguajes. El compilador de Java comprueba todas las expresiones y parámetros para asegurar que los tipos son compatibles. Si los tipos no coinciden, se generan errores que deben ser corregidos para que el compilador termine de compilar la clase.

2.3.1 Tipos de datos simples o primitivos

Java define ocho tipos de datos primitivos: **byte**, **short**, **int**, **long**, **char**, **float**, **double** y **boolean**. Estos tipos se pueden distribuir en cuatro grupos:

- **Enteros**.- En este grupo están los tipos **byte**, **short**, **int** y **long**, que son para números con signo de valor completo.
- **Números en coma flotante**.- Este grupo incluye los tipos **float** y **double**, que representan números con precisión fraccionaría.
- **Caracteres**.- Corresponde al tipo **char**, que representa símbolos de un conjunto de caracteres como letras y números.
- **Bolean**.- Es el tipo **boolean**, que es un tipo especial para valores lógicos.

byte

Es el tipo entero más pequeño con 8 bits con signo que tiene un rango desde -128 a 127. Las variables de tipo byte son especialmente útiles cuando estamos trabajando con un flujo de datos recibiendo desde una red o un archivo.

short

Es un tipo de 16 bits con signo. Su rango corresponde desde -32,768 a 32,767. Probablemente es el tipo de Java menos utilizado, puesto que tiene su byte más significativo primero (formato *big-endian*).

int

Es el tipo entero más utilizado. Es de 32 bits con signo. El rango de este tipo comprende desde -2,147,483,648 a 2,147,483,647. Además de otros usos las variables del tipo **int** se utilizan normalmente para realizar el control de bucles y para indexar matrices.

long

Es un tipo de 64 bits con signo y se utiliza en aquellas ocasiones en las que el tipo int no es lo suficientemente grande como para almacenar el valor deseado. Comprende desde -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.

float

Es un tipo en coma flotante el cual especifica valores con precisión simple que utilizan 32 bits de almacenamiento. La precisión simple es más rápida en algunos procesadores y necesita la mitad de espacio que la precisión doble, pero es imprecisa cuando los valores son muy grandes o muy pequeños. Las variables de este tipo son útiles cuando necesitamos valores fraccionarios que no requieren gran precisión.

double

Este tipo utiliza 64 bits para almacenar un valor. Realmente la precisión doble es más rápida que la simple en los procesadores que han sido optimizados para cálculos matemáticos a alta velocidad. Todas las funciones matemáticas trascendentes, como **sin**(), **cos**(), y **sqrt**() devuelven valores dobles. Cuando necesite mantener la precisión tras muchos cálculos iterativos, o está manipulando números de gran valor, **double** es la mejor opción.

char

En C/C++, char es un tipo entero con 8 bits. Java, utiliza Unicode para representar caracteres. Unicode define un conjunto de caracteres completamente internacional que puede representar todos los caracteres encontrados en todos los idiomas y lenguas humanas. En Java, char es un tipo de 16 bits. El rango de un carácter es de 0 a 65,536. No hay caracteres negativos. El conjunto de caracteres estándar, conocido como ASCII, tiene un rango de 0 a 127, y el conjunto de caracteres extendido de 8 bits, ISO-Latin-I tiene un rango de 0 a 255.

boolean

Este es un tipo simple para valores lógicos. Solo puede tomar uno entre dos posibles valores, **true** (verdadero) o **false** (falso). Este es el tipo que devuelven todos los operadores de comparación, como **a<b**.

2.3.2 Literales

Todos los valores que son asignados a variables se denominan literales, estos pueden ser de tipo primitivo como **byte**, **short**, **int**, **long**, **float**, **double**, **char** o **boolean**, o de tipo objeto como el **String**, que es un objeto utilizado para las cadenas.

2.3.2.1 Literales de tipo entero

Los enteros son el tipo más utilizado habitualmente en los programas. Cualquier valor numérico entero es un literal entero, como 1, 2, 46 y 937. Todos son valores decimales, lo que significa que describen un número en base 10.

Los literales enteros crean un valor **int** por naturaleza. Cuando se asigna un valor literal a una variable **byte** o **short**, no se genera ningún tipo de error si el valor literal está dentro del rango del tipo destino. Por otro lado, siempre podemos asignar un literal entero a una variable **long**. Sin embargo, para especificar un literal **long**, es necesario decirle de forma explicita al compilador que el valor literal es de tipo **long**, esto se hace añadiéndole una L o l (mayúscula o minúscula) al final de la literal, ejemplo:

```
int a = 54393;
byte b = 12;
short c = 432;
long d = 297862L;
```

2.3.2.2 Literales en coma flotante

Los números en coma flotante representan valores decimales con una componente fraccional. Los literales en con flotante en java son de doble precisión por defecto (tipo **double**). Para especificar un literal de tipo float se tiene que añadir una F o f a la constante. También se puede especificar explícitamente que un literal es de tipo double añadiendo D o d, aunque esto es redundante, ejemplo:

```
float a = 3.1416F;
double b = 836293.3928;
double = 836293.3928D;
```

2.3.2.3 Literales booleanos

Los literales booleanos son simples. Un valor booleano sólo puede tener dos valores lógicos, **true** (verdadero y **false** (falso). Los valores **true** y **false** no se convierten en ninguna representación numérica, ejemplo:

```
boolean a = true;
boolean b= false;
```

2.3.2.4 Literales tipo carácter

Los caracteres en Java son índices en el conjunto de caracteres Unicode. Son valores de 16 bits que se pueden convertir a enteros y manipularse con los operadores de enteros, como la suma y la resta. Un carácter de tipo literal se representa dentro de un par de comillas simples (´´). Todos los caracteres visibles ASCII se pueden introducir directamente dentro de las comillas, como 'a', 'z' y '@'. Para los caracteres que no se puedan introducir directamente, hay varias secuencias de escape que permiten introducir el carácter deseado, como '\'' para introducir una comilla simple, como '\n' para el carácter de línea nueva o '\u2342' para un carácter Unicode, ejemplo:

```
char a = 'e';
char b = '\t';
char c = '\u0012';
```

2.3.2.5 Literales tipo cadena

Los literales tipo cadena en Java se parecen mucho a los de cualquier otro lenguaje ya que son texto arbitrario entre un par de comillas dobles (""), solo que en Java las literales de tipo cadena se asignan a un tipo **String**, que al contrario de los tipos vistos arriba, no es primitivo o simple, es un objeto, y se puede concatenar con el signo (+). Ejemplo:

```
String a = "palabra";
String b = "Hola mundo " + "fantástico";
```

2.3.3 Matrices

Una matriz es un grupo de variables del mismo tipo a las que se hace referencia con el mismo nombre. Se pueden crear matrices de cualquier tipo y pueden tener una o más dimensiones. A un elemento en específico de una matriz se accede por su índice. Las matrices ofrecen un medio de agrupar información relacionada.

2.3.3.1 Matrices unidimensionales

Una matriz unidimensional es, básicamente, una lista de variables del mismo tipo. Para crear una matriz, primero se tiene que crear una variable matriz

del tipo deseado. La forma general de una declaración de matriz unidimensional es:

```
tipo nombre-matriz[];

int diasDelAnio[];
String mesesDelAnio[];
```

Y se pueden instancíar así:

```
Nombre-matriz = new tipo[tamaño];

diasDelAnio = new int[365];
mesesDelAnio = new String[12];
mesesDelAnio = {enero, febrero, marzo, abril, mayo, junio, Julio,
agosto, septiembre, octubre, noviembre, diciembre};
```

2.3.3.2 Matrices multidimensionales

En Java, las matrices multidimensionales realmente son matrices de matrices. Éstas se parecen y funcionan como las matrices multidimensionales habituales. Para declarar una variable del tipo matriz multidimensional, es necesario especificar cada índice adicional utilizando otra pareja de corchete ([]). Esta es una vista conceptual de una matriz bidimensional de 4 x 5:

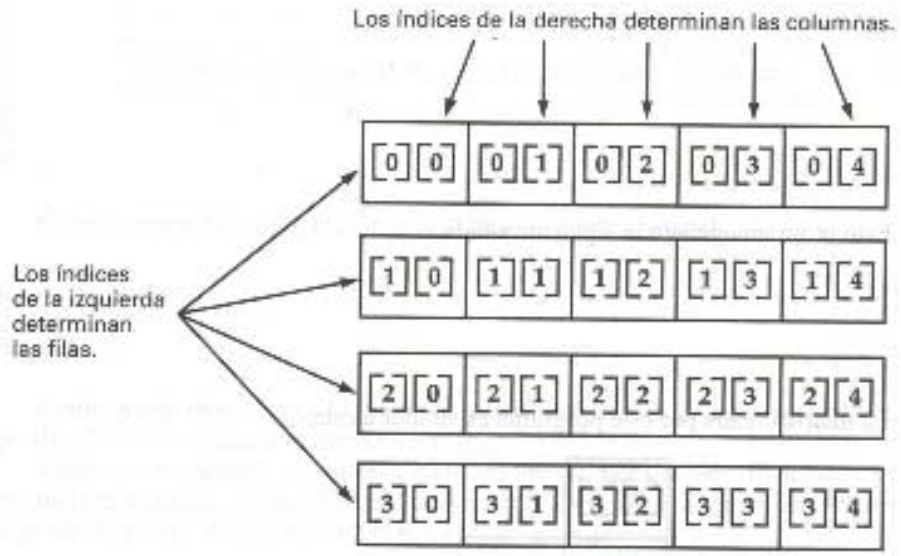


Figura 2.3.3.1. Matrices Multidimensionales.

La manera de declarar e instanciar una matriz multidimensional es la siguiente:

```
int dosDim[][] = new int[4][5];
```

2.4 Control de acceso y operadores

2.4.1 Control de acceso

Cuando se crea una nueva clase, se puede especificar el nivel de acceso que se quiere para las variables de instancia y los métodos definidos en la clase por medio de los siguientes modificadores de acceso:

- **public:** Cualquier clase puede acceder a las propiedades y métodos públicos.
- **protected:** Sólo las clases heredadas y aquellas situadas en el mismo paquete pueden acceder a las propiedades y métodos protegidos.
- **private:** Las variables y métodos privados sólo pueden ser accedidos desde dentro de la clase.
- **fiendly:** (valor por defecto). Si no se especifica ningún modificador de acceso, las variables y métodos se declaran *friendly* (amigas), lo que en la práctica significa lo mismo que *protected*.
- **private protected:** Sólo las clases heredadas pueden acceder a las propiedades y métodos protegidos.

La siguiente tabla le muestra los niveles de acceso permitidos por cada especificador.

Especificador	Clase	Subclase	Paquete	Mundo
private	X			
protected	X	X*	X	
public	X	X	X	X
package	X		X	

Tabla 2.4.1.1 Niveles de acceso.

La primera columna indica si la propia clase tiene acceso al miembro definido por el especificador de acceso. La segunda columna indica si las subclases de la clase (sin importar dentro de que paquete se encuentren estas) tienen acceso a los miembros. La tercera columna indica si las clases del mismo paquete que la clase (sin importar su parentesco) tienen acceso a los miembros. La cuarta columna indica si todas las clases tienen acceso a los miembros.

2.4.2 Operadores

Los operadores realizan algunas funciones en uno o dos operandos. Los operadores que requieren un operador se llaman operadores unarios. Por ejemplo, ++ es un operador unario que incrementa el valor su operando en uno.

Los operadores que requieren dos operandos se llaman operadores binarios. El operador = es un operador binario que asigna un valor del operando derecho al operando izquierdo.

Los operadores se pueden dividir en las siguientes categorías: aritméticos; relacionales y condicionales; lógicos y de desplazamiento y; de asignación.

2.4.3 Operadores aritméticos

Este cuadro muestra las operaciones aritméticas binarias.

operador	uso	descripción
+	op1 + op2	Suma op1 y op2
-	op1 - op2	Resta op2 de op1
*	op1 * op2	Multiplica op1 y op2
/	op1 / op2	Divide op1 por op2
%	op1 % op2	Obtiene el resto de dividir op1 por op2

Nota: El lenguaje Java extiende la definición del operador “+” para incluir la concatenación de cadenas.

Los operadores “+” y “-“ tienen versiones unarias que seleccionan el signo del operando.

operador	uso	descripción
+	+ op	Indica un valor positivo
-	- op	Niega el operando

Además, existen dos operadores de atajos aritméticos.

operador	uso	descripción
++	op ++	Incrementa op en 1; evalúa el valor antes de incrementar
++	++ op	Incrementa op en 1; evalúa el valor después de incrementar
--	op --	Decrementa op en 1; evalúa el valor antes de decrementar
--	-- op	Decrementa op en 1; evalúa el valor después de decrementar

2.4.4 Operadores relaciónales y condicionales

Los valores relacionales comparan dos valores y determinan la relación entre ellos.

operador	uso	devuelve true si
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
!=	op1 != op2	op1 y op2 son distintos

Frecuentemente los operadores relacionales se utilizan con otro juego de operadores, los operadores condicionales, para construir expresiones de decisión más complejas.

Aquí hay tres operadores condicionales.

operador	uso	devuelve true si
&&	op1 && op2	op1 y op2 son verdaderos
||	op1 || op2	uno de los dos es verdadero
!	! op	op es falso

El operador “&” se puede utilizar como un sinónimo de “&&” si ambos operadores son booleanos. Similarmente, “|” es un sinónimo de “||” si ambos operandos son booleanos.

2.4.5 Operadores de desplazamiento

Los operadores de desplazamiento permiten realizar una manipulación de los bits de los datos. Esta tabla muestra los operadores lógicos y de desplazamiento disponibles.

operador	uso	descripción
>>	op1 >> op2	desplaza a la derecha op2 bits de op1
<<	op1 << op2	desplaza a la izquierda op2 bits de op1
>>>	op1 >>> op2	desplaza a la derecha op2 bits de op1(sin signo)
&	op1 & op2	bitwise and
|	op1 | op2	bitwise or
^	op1 ^ op2	bitwise xor
~	~ op	bitwise complemento

Los tres operadores de desplazamiento simplemente desplazan los bits del operando de la izquierda el número de posiciones indicadas por el operador de la derecha. Los desplazamientos ocurren en la dirección indicada por el propio operador.

2.4.6 Operadores de asignación

Se puede utilizar el operador de asignación =, para asignar un valor a otro. Además del operador de asignación básico, Java proporciona varios operadores de asignación que permiten realizar operaciones aritméticas, lógicas o de bits y una operación de asignación al mismo tiempo (como una sobrecarga de operadores C++). Por ejemplo: i = i + 2, se puede ordenar esta sentencia utilizando el operador +=; i += 2; son equivalentes.

Este cuadro lista los operadores de asignación y sus equivalentes.

operador	uso	equivale a
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2

%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
|=	op1 |= op2	op1 = op1 | op2
^=	op1 ^= op2	op1 = op1 ^ op2

2.5 Sentencias de control de flujo

Las sentencias de control de flujo determinan el orden en que se ejecutarán las otras sentencias dentro del programa.

sentencias	**palabras clave**
toma de decisiones	`if-else, switch-case`
Bucles	`for, while, do-while`
Excepciones	`try-catch-finally, throw`
Miscelaneas	`break, continue, label:, return`

2.5.1 La sentencia if - else

La sentencia **if-else** proporciona a los programas la posibilidad de ejecutar selectivamente otras sentencias basándose en algún criterio.

La sentencia **if**: se ejecuta si alguna condición es verdadera. Generalmente, la forma sencilla de **if** se puede escribir así.

```
if (expresión){
sentencia }

Si la expresión es falsa, puedes utilizar la sentencia else.
. . .
if (respuesta == OK) {
. . .
// Código para la acción OK
. . .
} else {
. . .
// Código para la acción Cancel
. . .
}
```

Existe otra forma de la sentencia **else**, **else if** que ejecuta una sentencia basada en otra expresión. Una sentencia **if** puede tener cualquier número de sentencias de acompañamiento **else if**.

2.5.2 La sentencia switch

La sentencia **switch** se utiliza para realizar sentencias condicionalmente basadas en alguna expresión. La sentencia **switch** evalúa su expresión, y ejecuta la sentencia case apropiada.

```
int valor;
. . .
switch (valor) {
case 1:
       . . .;
break;
case n:
       . . .;
break;}
```

Cada sentencia **case** debe ser única y el valor proporcionado a cada sentencia case debe ser del mismo tipo que el tipo de dato devuelto por la expresión proporcionada a la sentencia **switch**. La sentencia **break** hace que el control salga de la sentencia **switch** y continúe con la siguiente línea; es necesaria porque las sentencias case se siguen ejecutando hacia abajo. Esto es, sin un **break** explícito, el flujo de control seguiría secuencialmente a través de las sentencias case siguientes. Sin embargo, habrá escenario en los que querrás que el control proceda secuencialmente a través de las sentencias case.

Finalmente, puede utilizar la sentencia **default** al final de la sentencia **switch** para manejar los valores que no se han manejado explícitamente por una de las sentencias case.

```
int valor;
. . .
switch (valor) {
case 1:
       . . .;
break;
case n:
       . . .;
break;
default:
       . . .;
break;
}
```

La sentencia while -*ejecución cero o más veces*-

Generalmente hablando, una sentencia **while** realiza una acción mientras se cumpla una cierta condición. La sintaxis general de la sentencia **while** es.

```
while (expresión){
sentencia }
```

Esto es, mientras la expresión sea verdadera, ejecutará la sentencia.

2.5.3 La sentencia for

Puedes utilizar este bucle cuando conozcas los límites del bucle (su instrucción de inicialización, su criterio de terminación y su instrucción de incremento).

```
. . .
int i;
for (i = 0; i < valorX; i++) {
. . .
// hace algo en el elemento i
. . .
}
```

La forma general del bucle for puede expresarse asi.

```
for (inicialización; terminación; incremento){
     sentencias }
```

2.5.4 La sentencia do-while -*ejecución al menos una vez*-

Es similar al bucle **while**, excepto en que la expresión se avalúa al final del bucle.

```
do {
sentencias
} while (Expresión Booleana);
```

La sentencia **do-while** se usa muy poco en la construcción de bucles pero tiene sus usos.

2.6 CLASSPATH

Es una lista de directorios que indican al sistema donde ha instalado varias clases e interfaces compiladas Java. Cuando busque una clase, el intérprete Java busca un directorio en su CLASSPATH cuyo nombre coincida con el nombre del paquete del que la clase es miembro. Los ficheros .class para todas las clases e interfaces definidas en un paquete deben estar en ese directorio de paquete.

Los nombres de paquetes pueden contener varios componentes (separados por puntos). De hecho, los nombres de los paquetes de Java tienen varios componentes: java.util, java.lang, etc...

Cada componente del nombre del paquete representa un directorio en el sistema de ficheros. Así, los ficheros .class de java.util están en un directorio llamado *util* en otro directorio llamado java en algún lugar del CLASSPATH.

Para ejecutar una aplicación Java, se especifica el nombre de la aplicación que se desea ejecutar en el intérprete. Para ejecutar un *applet*, se especifica el nombre del *applet* en una etiqueta <APPLET> dentro de un fichero HTML. El navegador que ejecute el applet pasa el nombre del *applet* al intérprete Java. En cualquier caso, la aplicación o el *applet* que se está ejecutando podría estar en cualquier lugar del sistema o de la red. Igualmente, la aplicación o el *applet* pueden utilizar otras clases y objetos que están en la misma o diferente localización.

Como las clases pueden estar en cualquier lugar, se debe indicar al intérprete Java donde puede encontrarlas. Se puede hacer esto con la variable de entorno CLASSPATH que comprende una lista de directorios que contienen clases Java compiladas. La construcción de CLASSPATH depende de cada sistema.

Cuando el intérprete obtiene un nombre de clase, desde la línea de comandos, desde un navegador o desde una aplicación o un applet, el intérprete busca en todos los directorios de CLASSPATH hasta que encuentra la clase que está buscando.

2.7 Algunos paquetes de Java

Existen muchos paquetes en la librería estándar de Java, paquetes que, además, han ido variando según se sucedían las versiones del mismo. Así, pues, vamos a destacar algunos paquetes presentes en todos los JDK:

java.lang

Este paquete incluye las clases imprescindibles para que el lenguaje Java funcione como tal, es decir, clases como *Object*, *Thread*, *Exception*, *System*,

Integer, *Float*, *Math*, *Package*, *String*, etc., que implementan la base del lenguaje. No es necesario importar nada desde ese paquete porque se carga por defecto.

java.util

Este paquete es el segundo en importancia, ya que incluye muchas clases útiles como pueda ser Date (fecha) y, sobre todo, diversas clases que permiten el almacenamiento dinámico de información, como *Vector*, *LinkedList*, *HashMap*, etc.

java.io

Este paquete contiene las clases necesarias para realizar las operaciones de entrada/salida, ya sea a pantalla o a ficheros, clases heredadas de *FileInputStream* y *FileOutputStream*.

java.awt

Este paquete, cuyo nombre corresponde a las siglas de *Abstract Windowing Toolkit*, contiene las clases necesarias para crear interfaces de usuario, por medio de menús, botones, áreas de texto, cajas de confirmación, etc..

java.applet

Contiene la archifamosa clase *Applet*, que nos permite crear applets para verlos en nuestro navegador.

java.net

Paquete que permite la programación de aplicaciones que accedan a bajo nivel a redes TCP/IP.

2.8 El recolector de basura

El entorno de ejecución tiene un recolector de basura que periódicamente libera la memoria ocupada por los objetos que no se van a necesitar más.

El recolector de basura es un barredor de marcas que escanea dinámicamente la memoria de Java buscando objetos, marcando aquellos que han sido referenciados. Después de investigar todos los posibles *paths* de los objetos, los que no están marcados (esto es, no han sido referenciados) se les conoce como basura y son eliminados.

El colector de basura funciona en un *thread* (hilo) de baja prioridad y funciona tanto síncrona como asíncronamente dependiendo de la situación y del sistema en el que se esté ejecutando el entorno Java.

El recolector de basura se ejecuta síncronamente cuando el sistema funciona fuera de memoria o en respuesta a una petición de un programa Java. Un programa Java le puede pedir al recolector de basura que se ejecute en cualquier momento mediante una llamada a System.gc().

Nota: Pedir que se ejecute el recolector de basura no garantiza que los objetos sean recolectados.

En sistemas que permiten que el entorno de ejecución Java note cuando un *thread* a empezado a interrumpir a otro *thread* (como Windows 95/NT), el recolector de basura de Java funciona asíncromamente cuando el sistema está ocupado. Tan pronto como otro *thread* se vuelva activo, se pedirá al recolector de basura que obtenga un estado consistente y termine.

2.8.1 Finalización

Antes de que un objeto sea recolectado, el recolector de basura le da una oportunidad para limpiarse él mismo mediante la llamada al método finalize() del propio objeto. Este proceso es conocido como finalización.

Durante la finalización de un objeto se podrían liberar los recursos del sistema como son los ficheros, etc. y liberar referencias en otros objetos para hacerse elegible por la recolección de basura.

El método finalize() es un miembro de la clase java.lang.Object. Una clase debe sobrescribir el método finalize() para realizar cualquier finalización necesaria para los objetos de ese tipo.

2.9 Constructores

Todas las clases Java tienen métodos especiales llamados Constructores que se utilizan para inicializar un objeto nuevo de ese tipo. Los constructores tienen el mismo nombre que la clase --el nombre del constructor de la clase *Rectangle* es Rectangle(), el nombre del constructor de la clase *Thread* es Thread(), etc...

Java soporta la sobrecarga de los nombres de métodos, por lo que una clase puede tener cualquier número de constructores, todos los cuales tienen el mismo nombre. Al igual que otros métodos sobrecargados, los constructores se diferencian unos de otros en el número y tipo de sus argumentos.

Consideremos la clase *Rectangle* del paquete java.awt que proporciona varios constructores diferentes, todos llamados Rectangle(), pero cada uno con

número o tipo diferentes de argumentos a partir de los cuales se puede crear un nuevo objeto *Rectangle*. Aquí tiene las firmas de los constructores de la clase java.awt.Rectangle.

```
public Rectangle()
public Rectangle(int width, int height)
public Rectangle(int x, int y, int width, int height)
public Rectangle(Dimension size)
public Rectangle(Point location)
public Rectangle(Point location, Dimension size)
```

El primer constructor de *Rectangle* inicializa un nuevo *Rectangle* con algunos valores por defecto razonables, el segundo constructor inicializa el nuevo *Rectangle* con la altura y anchura especificadas, el tercer constructor inicializa el nuevo *Rectangle* en la posición especificada y con la altura y anchura especificadas, etc...

Típicamente, un constructor utiliza sus argumentos para inicializar el estado del nuevo objeto. Entonces, cuando se crea un objeto, se debe elegir el constructor cuyos argumentos reflejen mejor cómo se quiere inicializar el objeto.

Basándose en el número y tipos de los argumentos que se pasan al constructor, el compilador determina cual de ellos utilizar, Así el compilador sabe que cuando se escribe.

```
new Rectangle(0, 0, 100, 200);
```

El compilador utilizará el constructor que requiere cuatro argumentos enteros, y cuando se escribe.

```
new Rectangle(miObjetoPoint, miObjetoDimension);
```

Utilizará el constructor que requiere como argumentos un objeto *Point* y un objeto *Dimension*.

Cuando se escriban clases, no se tiene porque proporcionar constructores. El constructor por defecto, el constructor que no necesita argumentos, lo proporciona automáticamente el sistema para todas las clases. Sin embargo, frecuentemente se querrá o necesitará proporcionar constructores para las clases.

CAPÍTULO III

Lenguaje de programación Java Edición Empresarial (J2EE)

3.1 Java Edición Empresarial (J2EE)

La tecnología Java ha evolucionado de ser un lenguaje de programación diseñado para crear sistemas empotrados independientes de la máquina, hasta convertirse en una tecnología para servidores robusta, independiente del proveedor y de la máquina, la cual permite a la comunidad corporativa obtener el potencial completo de las aplicaciones centradas en Web.

Java comenzó con el lanzamiento del kit de desarrollo de Java (JDK), el cuál fue extendiéndose con interfaces y bibliotecas según el mundo corporativo exigía y recibía interfaces de programación de aplicaciones (API[1]) que resolvían problemas del mundo real. Con el lanzamiento de *Java 2 Standard Edition* (J2SE) se integraron por completo muchas extensiones a las API de JDK, necesarias para construir aplicaciones Java robustas de misión crítica; sin embargo carecía de la fuerza para desarrollar aplicaciones empresariales.

Para asegurar el crecimiento de Internet era necesario proporcionar una forma en que las aplicaciones Web pudieran interactuar con otros servicios como, por ejemplo las bases de datos, y generar páginas en forma dinámica. La tecnología de la Interfaz común de pasarela (*Common Gateway Interface* o CGI[2]) fue una solución que adoptaron muchas empresas. La tecnología CGI consistía de un programa que el navegador podía llamar cada vez que se seleccionara el enlace adecuado o se enviara un formulario Web.

Además de llamar al programa CGI, el navegador podía enviar al programa datos, ya sea introducidos por el usuario en un formulario o codificados directamente en el enlace. El programa CGI utilizaba dichos datos para interactuar con notros componentes de la infraestructura de la empresa, por ejemplo extraer datos de una base de datos. La información se incorporaba entonces a una página Web generada dinámicamente por el programa CGI y se enviaba al navegador para su despliegue.

La tecnología de CGI resolvió el problema de conectar a los clientes con la infraestructura de la empresa. Sin embargo, surgió un nuevo conjunto de problemas al ir migrando las empresas hacia las aplicaciones centradas en Web. La tecnología de CGI era muy pesada en cuanto a recursos y no era suficientemente escalable para hacer frente al aumento dramático en el número de

[1] Un API (del inglés *Application Programming*) es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

[2] Interfaz de entrada común (en inglés *Common Gateway Interface*, abreviado **CGI**) es una importante tecnología de la World Wide Web que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web.

clientes que necesitaban obtener acceso a los recursos de la empresa a través de programas CGI.

El equipo de desarrollo de Java ideó una solución a los problemas de la tecnología CGI. La solución era escalable y requería menos recursos que la tecnología de CGI, a la vez que era capaz de conectar con la infraestructura de la empresa y generar páginas Web dinámicas. Su solución fue el servlet.

Un servlet consiste de clases Java, datos y métodos, los cuales puede llamar un navegador en forma similar a como llama un programa CGI. Aunque los *servlets* mejoraron la base sentada por la tecnología de CGI, sufrían de una seria desventaja. Un servlet exige que los programadores sean conocedores del lenguaje de programación Java. Los programadores Web de esa época conocían bien HTML y lenguajes sencillos como Java Script, pero no estaban cómodos con lenguajes de programación completos como Java.

Esto creó un problema para el equipo de desarrollo de Java. Los *servlets* tenían que ser más fáciles de programar antes de ser aceptados ampliamente por los programadores Web. Su solución fue crear una nueva tecnología conocida como *Java Server Pages* (JSP). Los programas JSP se pueden escribir con pocos o ningunos conocimientos de Java, debido a que la mayor parte del código de una JSP consiste de HTML, con fragmentos de código Java entremezclados.

3.1.1 ¿Por qué J2EE?

Con la aparición de las aplicaciones Web que dependían cada vez más de tecnologías de servidor como es el *middleware*, los departamentos de informática necesitaban alguna forma sustentable de desarrollar aplicaciones y el *middleware* relacionado que fueran portables y escalables.

Era necesario diseñar estas aplicaciones de forma que pudieran atender a miles de usuarios en forma simultánea, 24 horas al día, siete días a la semana, sin ningún tiempo de inactividad. Uno de los desafíos para construir una aplicación compleja de este tipo es poder diseñar y probarla. J2EE simplifica la creación de las aplicaciones empresariales, ya que la funcionalidad se encapsula en los componentes de J2EE. Esto permite a los diseñadores y programadores dividir la aplicación de acuerdo con sus funciones, las cuales se distribuyen entre los componentes del servidor que se construyen con J2EE.

J2EE es una tecnología versátil, ya que los componentes de las aplicaciones que se construyen con J2EE se comunican entre sí detrás de las cámaras mediante métodos estándar de comunicación como son HTTP, SSL, XML, RMI e IIOP.

Todos los programas J2EE están escritos en Java, lo cual permite a las empresas utilizar los programadores Java con los que ya cuenta para crear

programas que funcionan en cada una de las capas de la infraestructura multicapa. Ya no es necesario que las empresas busquen programadores para escribir programas que se conecten con componentes específicos de cada proveedor.

Los *Enterprise JavaBeans*, los *servlets* y las *Java Server Pages* son los componentes clave de J2EE. Además de ello, J2EE incluye siete servicios más, los cuales se describen de forma breve.

3.1.1.1 Compatibilidad con CORBA[3]

Sun Microsystems incluyó en J2EE dos tecnologías de CORBA que permiten a los programas Java comunicarse con cualquier sistema empresarial compatible con tecnología CORBA e interactuar con sistemas heredados. Estas tecnologías son JavaIDL y RMI-IIOP. JavaIDL se utiliza para conectar programas Java con sistemas CORBA. RMI-IIOP es la unión de la API de Invocación Remota de Métodos (RMI) de Java con el Internet Inter.-ORB Protocol (IIOP) que utiliza CORBA, la cual permite conectar programas Java con sistemas heredados.

3.1.1.2 JavaMail

El equipo de desarrollo de Java necesitaba alguna forma eficiente para que los clientes y los sities de comercio electrónico pudieran intercambiar información como, por ejemplo, las confirmaciones de pedidos. La solución es la API JavaMail, la cual permite a los programadores Java comunicarse mediante el envío y recepción de mensajes de correo electrónico.

3.1.1.3 Servicio de mensajes de Java (JMS)

La API del Servicio de mensajes de java (*Java Message Service* o JMS) se utiliza para incluir en los programas Java un enlace de transmisión entre los componentes. Este enlace permite la transmisión y recepción en forma asíncrona de mensajes tolerantes a fallos.

3.1.1.4 Interfaz de nombres y directorios para Java (JNDI)

Los objetos se pueden ubicar en distintos puntos en servidores conectados a la infraestructura de la empresa. El equipo de desarrollo de Java necesitaba alguna forma para permitir a los programas Java localizar estos objetos en forma sencilla. Su solución fue el crear estándares para la nomenclatura junto con la API

[3] CORBA (*Common Object Request Broker Architecture*) arquitectura común de intermediarios en peticiones a objetos, es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

Interfaz de nombres y directorios para Java (*Java Naming and Directory Interface* o JNDI), de forma que los programadores pudieran buscar objetos desde dentro de sus programas Java.

3.1.1.5 API Java para transacciones (JTA)

Una transacción puede involucrar a varios componentes, y el equipo de desarrollo de Java necesitaba alguna forma en que los componentes pudieran gestionar sus propias transacciones. El equipo creó la API Java para transacciones (*Java Transaction API* o JTA) para permitir a los programadores incluir código para gestionar las transacciones dentro de los componentes.

3.1.1.6 JDBC

El equipo de desarrollo de java creó la API JDBC, la cual permite a un programa conectarse e interactuar con prácticamente cualquier Base de Datos comercial (y la mayoría de los no comerciales).

3.1.1.7 Descriptores de despliegue XML

Muchas empresas y algunas industrias han adoptado XML, como una forma de almacenar, manipular e intercambiar información textual contenida en documentos. El equipo de desarrollo de Java incluyó en J2EE un conjunto de descriptores que permiten a los programadores crear herramientas y componentes que interactúan con documentos XML. Los descriptores de despliegue XML definen el entorno y la funcionalidad de los componentes al desplegarlos dentro del contenedor J2EE. El descriptor indica al contenedor cómo y dónde desplegar los componentes.

3.2 Servlets

3.2.1 Inducción a los servlets.

Los *servlets* son una respuesta de la tecnología Java a los CGI's (*Common Gateway Interface*, *Interfaz Común de la Puerta de Entrada*). Los servlets son programas que corren en un Servidor Web, actuando como una capa intermedia entre una solicitud viniendo de un navegador Web u otro cliente HTTP y bases de datos o aplicaciones en el servidor Web. Sus principales funciones son:

- **Leer cualquier dato enviado por un usuario.** Estos datos son usualmente introducidos mediante una página Web, pero también podrían venir mediante un *applet* de Java o un cliente personalizado HTTP.

- **Leer cualquier otra información que venga de la cabecera de una solicitud.** Esta información incluye detalles acerca de las características del navegador, *cookies*, el nombre de *host* del cliente y muchas más.
- **Generar resultados.** Este proceso puede requerir comunicación con una base de datos, ejecutando llamadas RMI o CORBA, invocando una aplicación o resolviendo la respuesta directamente.
- **Dar formato al resultado dentro de un documento.** En la mayoría de los casos, esto implica colocar la información dentro de una página HTML.
- **Establecer los parámetros apropiados de la respuesta HTTP.** Esto significa decirle al navegador que tipo de documento esta empezando a regresar (ej.- HTML), estableciendo *cookies* y atrapando parámetros, y otras tareas.
- **Enviar el documento de regreso al cliente.** Este documento puede ser enviado al cliente en formato de texto (HTML), formato binario (imagen GIF), o tal vez en formato de compresión como un *gzip*.

Muchas solicitudes de cliente pueden ser atendidas regresando documentos preconstruidos, y estas solicitudes serían manejadas por el servidor sin tener que invocar *servlets*. En muchos casos, una respuesta estática no es suficiente, y una página necesita ser generada por cada solicitud. Hay un sin número de razones por la que las páginas Web necesitan ser construidas “al momento” como estas:

- **Las páginas Web están basadas en datos enviados por el usuario.** Por ejemplo, los resultados de una página de búsqueda, páginas de confirmación de órdenes de tiendas son específicas para las solicitudes particulares de los usuarios.
- **Las páginas Web son derivadas de datos que cambian frecuentemente.** Por ejemplo, un reporte del tiempo o una página de noticias podrían construir la página dinámicamente.
- **Las páginas Web usan información de las bases de datos de empresas u otras fuentes de servicio.** Por ejemplo, un sitio de comercio electrónico podría usar un servlet para construir una página Web que liste los precios actuales y disponibilidad de cada producto que vende.

En conclusión, los *servlets* no están restringidos solo a servidores Web o de Aplicativos que manejan solicitudes HTTP, si no que pueden ser usados con otro tipo de servidores. Por ejemplo, los *servlets* pueden ser puestos en servidores de mail o FTP para extender su funcionalidad.

3.2.2 Ventajas de los servlets sobre CGI’S

Los *servlets* son más eficientes, fáciles de usar, más poderosos, más portables, seguros y económicos que los tradicionales CGI’s y otras tecnologías alternativas parecidas a los CGI’s.

Eficiente

Con los CGI's tradicionales, un nuevo proceso es lanzado por cada solicitud de HTTP. Si los programas CGI por si mismos son relativamente cortos, el evento de lanzar los procesos puede abarcar la mayor parte del tiempo de ejecución. Con los *servlets*, la Maquina Virtual de Java permanece corriendo y manejando cada solicitud usando un hilo de Java (Java *Thread*) ligero, no un proceso del sistema operativo pesado. Similarmente en los CGI's, si existen N solicitudes simultaneas solicitando al mismo CGI, el código del CGI es cargado dentro de la memoria N veces. Con los *servlets*, habría N hilos pero solamente una sola copia de la clase del servlet. Finalmente, cuando un CGI termina de manejar una solicitud, el programa termina. Esto hace más difícil el procesamiento del cache, en cambio los *servlets* permanecen en memoria una vez que mandaron una respuesta, así que no se tiene que cargar datos y clases otra vez.

Conveniente

Los *servlets* tienen una amplia infraestructura para interpretar automáticamente y decodificar HTML de datos, leer y establecer cabeceras HTTP, manejar *cookies*, manejar sesiones y otras utilerías.

Poderoso

Los *servlets* soportan grandes capacidades que les son difíciles o imposibles soportar a los CGI's. Los *servlets* pueden comunicarse directamente con los servidores Web, lo que un CGI común no puede, al menos no sin usar un específico API. Esta comunicación con el servidor Web hace más fácil el traslado de URL's relativas dentro de nombres concretos de *path*. Los *servlets* pueden compartir datos, haciendo más fácil la implementación de la conexión a la base de datos. También los *servlets* pueden mantener la información de solicitud a solicitud, simplificando así técnicas de manejo de sesiones.

Portable

Los *servlets* están escritos en el lenguaje de programación Java y sigue un estándar API. Por consecuente, los *servlets* pueden ser ejecutados sobre *I-Planet, Apache*, IIS (*Microsoft Internet Infromation Server*), *IBM WebSphere* o *StarNine WebStar* sin cambios. Por lo que los *servlets* se vuelven portables por naturaleza ya que cada servidor tiene su maquina virtual de Java quien interpreta al *servlet*.

Seguro

Una de las principales fuentes de vulnerabilidad de los CGI's proviene del hecho de que ellos usualmente son ejecutados por *shells* del sistema operativo de propósito general. Así que los programadores de CGI's tienen que ser muy cuidadosos en filtrar caracteres como comillas simples o dobles que son tratados especialmente por los *shells*. Una segunda fuente de problemas es el hecho de

que algunos CGI's son procesados por lenguajes que no revisan automáticamente arreglos o cadenas. Por ejemplo, en C y C++ es perfectamente válido alojar un arreglo de 100 elementos y después escribir en el elemento 99, el cual realmente es una parte aleatoria de la memoria del programa. Algunos programadores quienes olvidan revisar esto, ellos mismos abren su sistema a deliberados o accidentales ataques de sobreflujo del buffer. Los *servlets* no sufren de estos problemas. Siempre si un servlet ejecuta una llamada remota para invocar un programa en el sistema operativo, no utiliza un *shell* para hacer eso. Y por supuesto, el chequeo de arreglos y otras características de protección de memoria son un parte central y controlada por el lenguaje Java.

Económico.

Existen muchos servidores Web gratis o económicos disponibles que son adecuados para sitios Web personales de bajo volumen. Como quiera que sea, con la excepción del Apache, que es gratis, la mayoría de los servidores Web comerciales son relativamente caros y los *servlets* se pueden adecuar a cualquier servidor Web, desde los que son gratuitos hasta los más caros. En contraste con muchas alternativas de los CGI's, las cuales requieren una inversión inicial al comprar los paquetes propietarios.

3.2.3 Estructura básica de los servlets.

Un navegador genera una solicitud GET cuando el usuario escribe una URL sobre la línea de dirección, entra en una liga de una página Web, o envía una forma HTML que no especifica un METHOD (GET o POST). Los *servlets* pueden fácilmente manejar solicitudes POST, las cuales son generadas cuando alguien envía una forma HTML especificando METHOD="POST".

Para ser un servlet, una clase debería extender de *HttpServlet* y sobrecargar el método *doGet* o *doPost*, dependiendo del dato que es enviado por el GET o POST. Si se requiere manejar los dos métodos, simplemente se sobrecargan el *doGet* y el *doPost*, o viceversa. Ejemplo:

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http.* ;

public class HolaMundo extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    // Usar "request" para leer cabecers HTTP entrantes
    // (ej.- cookies) y datos de formas HTML (ej.- datos
```

```
    // enviados por el usuario).

    // Usar "response" para especificar el código de estatus
    // y cabeceras de respuestas HTTP (ej.- tipo de contenido
    // y cookies).

    PrintWriter out = response.getWriter();
    out.println("Hola Mundo");
    // Usar "out" para enviar el contenido del navegador.
  }
}
```

Ambos métodos toman dos argumentos: un *HttpServletRequest* y un *HttpServletResponse*. El *HttpServletRequest* tiene métodos por los cuales tu puedes encontrar información entrante como puede ser datos de la forma, solicitudes de cabecera HTTP y el *hostname* de los clientes. El *HttpServletResponse* te permite especificar la información saliente, tales como códigos de estatus HTTP (200, 404, etc.), cabeceras de respuestas (*Content-Type*, *Set-Cookie*, etc), y lo más importante, te permite obtener un *PrintWriter* usado para enviar el contenido del documento al cliente.

Hablando estrictamente, *HttpServlet* no es un punto de comienzo de los *servlets*, ya que los *servlets* podrían, en principio, extender de mail, FTP, u otros tipos de servidores. Para estos ambientes los *servlets* extenderían de una clase personalizada derivada de *GenericServlet*, la clase padre de *HttpServlet*. En resumen, los *servlets* son usados casi exclusivamente para servidores que se comunican vía HTTP, como servidores Web y de Aplicaciones.

3.2.4 Ciclo de vida del servlet.

Cuando un servlet es creado por primera vez, su método *init* es invocado, y es ahí donde se pone el código de inicialización un sola vez. Después de esto, cada solicitud de usuario resulta en un hilo que llama al método *service* de la instancia creada anteriormente. Normalmente las múltiples solicitudes concurrentes resultan en múltiples hilos llamando a *service* simultáneamente, aunque el servlet puede implementar una interfase especial que estipula que solo se permite que corra un solo hilo a la vez. Después el método *service* llama a los métodos *doGet*, *doPost* o algún otro, dependiendo de la solicitud HTTP recibida. Finalmente, cuando el servidor decide descargar el servlet, se llama al método del servlet *destroy*.

3.2.4.1 El método init.

El método *init* es llamado cuando el servlet es creado por primera vez y no es vuelto a llamar en cada solicitud del usuario. Así que, esto es usado solo una vez en las inicializaciones, tal como en el método *init* de los *applets*. El servlet

puede ser creado cuando un usuario invoca por primera vez a una URL correspondiente al servlet o cuando el servidor es levantado por primera vez.

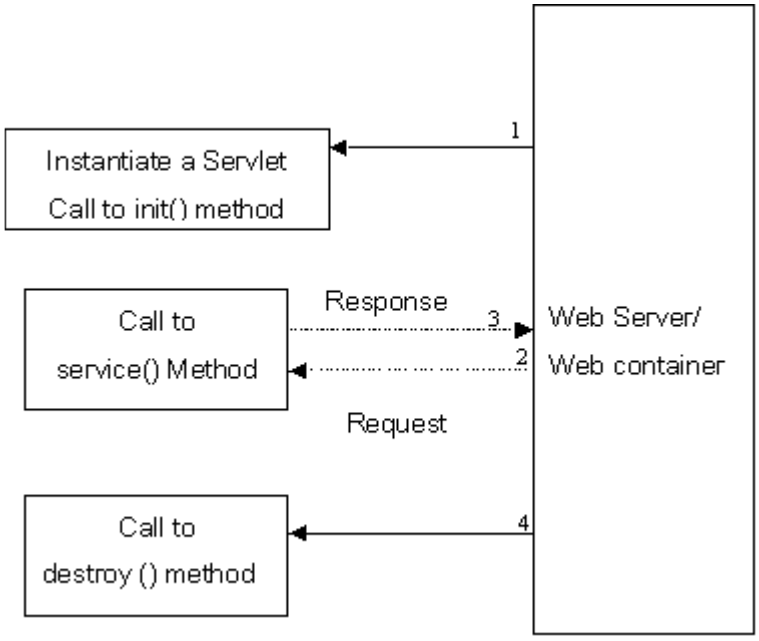


Figura 3.2.3.1 Ciclo de vida del Servlet.

3.3 Beans

En el campo del software se puede crear una interfaz de usuario en un programa Java sobre la base de componentes: Paneles, botones, etiquetas, caja de listas, barras de desplazamiento, diálogos, menús, etc. Si se ha utilizado Delphi o Visual Basic, ya estamos familiarizados con la idea de componente, aunque el lenguaje de programación sea diferente. Existen componentes que van desde los más simples como un botón hasta otros mucho más complejos como un calendario, una hoja de cálculo, etc.

Muchos componentes son visibles cuando se corre la aplicación, pero no tienen por qué serlo, solamente tienen que ser visibles en el momento de diseño, para que puedan ser manipulados por el Entorno de Desarrollo de Aplicaciones (IDE[4]).

Podemos crear una aplicación en un IDE seleccionando los componentes visibles e invisibles en una paleta de herramientas y situarlas sobre un panel o una ventana. Con el ratón unimos los sucesos (*events*) que genera un objeto (fuente), con los objetos (*listeners*) interesados en responder a las acciones sobre dicho

4

CAPÍTULO IV

Análisis, desarrollo e implementación

4.1 Propuesta para sistema de automatización del proceso de preinscripción y gestión de cursos vía web

4.1.1 Antecedentes

Hoy en día, la mayoría de las empresas (grandes, medianas o pequeñas) necesitan tener acceso a la información las 24 horas del día, el uso de la tecnología, apoya el hecho que se puedan conjugar una gama de sistemas para hacer realidad la disposición de la información desde cualquier lugar que el usuario lo requiera.

De las distintas plataformas que hoy en día se presentan solo se requieren aquellas que no sean complicadas y que su manejo sea más ágil, para poder obtener al final un servicio eficiente.

Las grandes cantidades de información que se tienen que almacenar y/o procesar hacen de este un gran reto para aquellas personas que distribuyen la información en una base de datos ya que son estas las que tienen la responsabilidad de dar un pequeño empujón para que su consulta sea segura y ágil.

EDUTECSA es una empresa dedicada a la capacitación de personal y particulares en las diversas tecnologías desarrolladas por Sun Microsystems. EDUTECSA siempre preocupado ante la gran ola de nuevas tecnologías, tomó el compromiso de poder aportar los conocimientos que conllevan estas, tales como Java, Sun One, Solaris, entre otras, a las personas o empresas que quieran ir siempre un paso adelante. Para poder llevar a cabo esta tarea es necesario apoyarse en las nuevas tecnologías y en el auge que está teniendo Internet. Actualmente, el proceso de preinscripción y gestión de cursos, se lleva a cabo de forma manual por el área administrativa de la empresa, de ahí surge la necesidad de automatizar dicho proceso para su mayor eficiencia.

4.1.2 Descripción del problema

EDUTECSA necesita un sistema de automatización del proceso de preinscripción y gestión de cursos que opere tanto de forma local como en Internet, y que sea de alta disponibilidad para que el cliente pueda estar informado las 24 horas, los 365 días del año.

Se requiere que el sistema abarque las dos partes principales del proceso, estas son la gestión centralizada de los alumnos y cursos que se imparten en la empresa, y la automatización de la preinscripción de alumnos que no dependa del horario e instalaciones de la empresa.

Para la parte de la gestión de alumnos y cursos, se necesita que los alumnos se puedan registrar en una base de datos para poder llevar el control de sus evaluaciones, dichas evaluaciones las podrá aplicar el instructor de forma manual o mediante el sistema, según convenga.

El sistema debe permitir almacenar toda la información referente a los perfiles de usuarios que van a utilizar el mismo, estos perfiles son administrador, instructor, alumno y empresa.

El administrador debe tener acceso a todos los catálogos con los que va a poder dar de alta, baja o hacer cambios a la información. Estos catálogos deben ser de usuarios, cursos, agendas, aulas, etc. Como complemento a esto, el administrador debe tener las funcionalidades básicas para poder realizar la gestión del sistema, tales como la inscripción de alumnos y parametrización de los contenidos del sitio, entre otros.

El instructor puede aplicar los exámenes mediante el sistema y podrá guardar y consultar las calificaciones de sus alumnos correspondientes.

El alumno tendrá derecho de consultar su expediente, así como la consulta y la cancelación de los cursos a tomar, entre otras.

La empresa solo podrá conocer los expedientes de sus empleados que tomaron algún curso.

Todos los perfiles tendrán derecho a poder modificar su contraseña.

Para la parte automatización de preinscripción de alumnos, se necesita que el sistema tenga disponibilidad de 7 días x 24 horas y que esté abierto a todo el público en general, aquí se podrá mostrar información sobre la empresa, por ejemplo:

- ¿Quienes somos?
- Historia
- Visión
- Misión

A su vez deberá mostrar al público toda la información necesaria sobre los cursos y servicios que se proporcionan, por ejemplo:

- Calendario de cursos
- Planes de carrera
- Lista de cursos con temarios
- Información sobre los instructores
- Disponibilidad en aulas
- Precios de curso y planes de carrera

Cuando una persona está interesada en tomar un plan de carrera, tiene la curiosidad de saber que conocimiento tiene sobre el tema, por eso, se necesita un examen de conocimientos en línea para que el futuro alumno pueda darse cuenta de a que nivel de la carrera puede entrar.

Con estos puntos se cubre lo que es la información al público en general y la prestación de algunos servicios para orientar a los futuros alumnos, por lo tanto también es necesario que esos futuros alumnos puedan apartar su lugar con un tiempo límite y posteriormente formalizar su inscripción en las oficinas. Para esto se necesita que el público se pueda preinscribir en el curso que deseé, o en el plan de carrera que haya seleccionado, todo esto tomando en cuenta la capacidad de las aulas y la disponibilidad de los horarios.

4.1.3 Necesidades del Cliente

Los requerimientos que deberá solucionar el nuevo sistema de automatización del proceso de preinscripción y gestión de cursos vía web son los siguientes:

- Las reglas de negocio y políticas en las que se basa el actual proceso de preinscripción y gestión de cursos, las cuales deberán documentarse antes de diseñar ó construir.
- Considerar la generación de un prototipo que permita garantizar la funcionalidad requerida por el usuario.
- Alta disponibilidad en el proceso de preinscripción y que no dependa del horario de oficina.

4.1.4 Solución propuesta

Este proyecto se desarrollará bajo una arquitectura de tres capas (capa de cliente, capa de negocios y capa de datos) bajo un patrón de diseño modelo-vista-controlador, cuyas características funcionales permiten un mejor performance, escalabilidad y mantenimiento.

Para los formatos y páginas web en la capa del cliente se utilizará tecnología JSP con validaciones en *JavaScript*. En el lado de la capa de negocios se hará uso de la tecnología de s*ervlets*. En la capa de datos se hará uso de *beans*, los cuales se comunicarán con la BD por medio de JDBC para SQL u Oracle mediante la tecnología de *Connection Pool*, con un rango máximo de 200 conexiones simultáneas a base de datos.

Se utilizará el manejo de perfiles de usuario con contraseña para validar la seguridad.

Dentro del sistema, se han podido identificar 10 módulos a saber con algunos procesos que a groso modo resumen la lógica de la solución propuesta para el funcionamiento del sistema.

Estos procesos se definen en diagramas de flujo que se encuentran en el siguiente apartado.

4.1.5 Requerimientos Identificados para la implantación de la solución

Este apartado considera al sistema de automatización del proceso de preinscripción y gestión de cursos vía web en las fases de *inception* y *elaboration* de la metodología OOA&D, que se apoya en los diagramas y esquemas de UML. Por lo que las herramientas y tecnologías requeridas son las siguientes:

- Licencia para servidor web aplicativo WebLogic 8.0.
- Lenguaje Java (Tecnologías Servlets, JSP y posiblemente EJB).
- Tecnología web de Multi Tier (capa cliente, capa negocio y capa de base de datos).
- Páginas HTML y tecnología JSP para ser cargadas en el lado del cliente.
- Licencia para uso de SSL.
- DBMS Oracle, así como sus licencias.
- Licencia de uso para JDeveloper como herramienta de desarrollo.
- Notación UML y metodología OOA&D.
- Red local con protocolo TCP-IP.

4.1.6 Beneficios esperados

- Aplicación robusta, estable, confiable y multiplataforma.
- Obtención de datos en forma automática.
- Sistema multiusuario.
- Mantenimiento sencillo.
- Escalabilidad abierta.
- Seguridad de la información.
- 1 Año de garantía contra cualquier falla del sistema y soporte profesional respaldado por sus socios tecnológicos.
- Participación de recursos especializados en las herramientas de desarrollo.
- Posibilidad de servicio de mantenimiento indefinido.

4.2 Análisis

El sistema de automatización del proceso de preinscripción y gestion de cursos vía web consiste en un sistema que tiene como objetivo la automatización de cursos impartidos en el centro de capacitación, esto trae consigo la inscripción de alumnos vía Internet, la calendarización de los cursos, el control de instructores, evaluaciones de los alumnos, expedientes de estos, foros de discusión y la presentación de temarios y contenidos del curso en línea.

El sistema será desarrollado con tecnología Java, utilizando, JSP´s, *servlets*, *beans* y Oracle como base de datos. Los servlets son utilizados para control del flujo del sistema, el manejo de la sesión, así como para la interacción de los *beans* y de las JSP's, los *beans* son utilizados para las reglas de negocio y el manejo de datos para la interfaz gráfica del usuario y las JSP´s son la interfaz gráfica para el usuario que utilizan a los *beans* para mostrar la información, también se utiliza Java Script para la validación de la captura de los datos. El sistema es multiplataforma por naturaleza, pero se pretende la instalación en una plataforma UNIX y como *webserver* Apache. El siguiente diagrama muestra el caso de uso general del sistema:

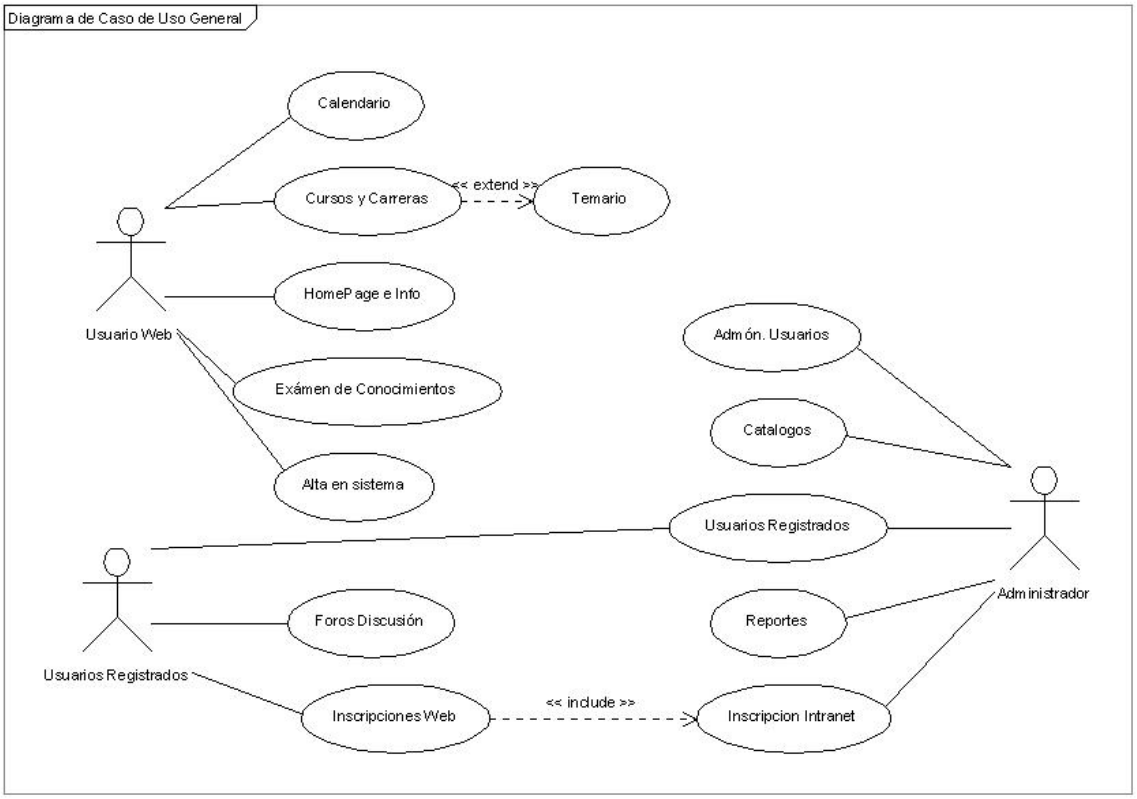


Figura 4.2.1 Caso de Uso General.

4.2.1 Funcionalidad del sistema

El sistema de automatización de Enseñanza-Aprendizaje requiere tener las siguientes funcionalidades:

4.2.1.1 Calendarización de cursos

El sistema requiere tener la funcionalidad de permitir al usuario con rol de usuario web, consultar el calendario de los cursos que se imparten en el centro de capacitación. Los cursos tienen como datos: nombre, duración, precio, moneda, contenido, temario y área a la que pertenece; y en la calendarización se van a capturar los cursos con sus fechas de inicio y fin, horarios, aulas, disponibilidad de lugares e instructores entre otras cosas. Cabe mencionar que para el contenido de los cursos se utilizara una liga a los archivos en HTML que tienen el contenido de cada curso, y los temarios de los cursos se realizarán en un archivo plano, el cual es leído por el sistema y será actualizado en la base de datos.

4.2.1.2 Administración de usuarios

El control de usuarios se llevará a cabo por el catálogo de usuarios (alumnos, administradores, instructores y empresas) y un catálogo para la captura del curriculo del instructor. Cada tipo de usuario tendrá diferentes privilegios, por lo tanto, podrán entrar a diferentes pantallas dependiendo de su usuario, esto es, una persona puede tener 1 o más usuarios pero de distinto tipo (p.e. un administrador puede tomar un curso y puede ser también alumno). Es importante hacer hincapié de que solo los alumnos que estén dados de alta en el sistema, tendrán derecho a pre-inscribirse y para ver la calendarización y cursos que se imparten no se solicita usuario ni *password*.

4.2.1.3 Pre-Inscripción de alumnos

Los alumnos podrán inscribirse por ellos mismos vía Internet o bien por el administrador en el centro de capacitación. Para la inscripción vía Internet se le solicitará al alumno sus datos personales así como un usuario y un *password*; una vez ya dado de alta el alumno en el sistema, este puede pre-inscribirse a los cursos que estén disponibles en ese momento, y si es la primera vez que se pre-inscribe a un curso, se le realizará una evaluación del área al que pertenece el curso, para así dependiendo de la calificación sugerirle otros cursos o en su defecto que se inscriba al que seleccionó. Para la inscripción por el administrador, este le pedirá al alumno sus datos y el curso en el que desea inscribirse. Cabe

mencionar que hasta que pague el alumno el administrador le cambiará el status de pre-inscrito a inscrito, y así tendrá derecho a otros servicios.

4.2.1.4 Evaluaciones y expedientes

Todos los cursos tendrán dos evaluaciones, una al inicio del curso y otra al final de este, como ya se mencionó también habrá una evaluación por área, que se realizará al momento en que el alumno se pre-inscriba vía Internet. Las calificaciones se guardarán en la base de datos para su posterior consulta por el alumno (expediente).

4.2.1.5 Foros de discusión

Con los foros de discusión se pretende dar un seguimiento aún después de que el alumno haya tomado un curso, esto es, se creará automáticamente un foro de discusión por cada curso calendarizado, y tendrá de vida 30 días naturales después de haber comenzado el curso. Con esto se busca que el alumno resuelva sus dudas y/o comentarios aun después de haber terminado el curso, y el instructor podrá entrar al foro para resolver las dudas.

4.2.2 Casos de uso

4.2.2.1 Caso de uso “Módulo administrador”

Descripción:
El sistema requiere tener la funcionalidad de permitir al usuario con rol de Administrador, realizar preinscripción de alumnos, consulta de los mismos, consulta de preinscripciones a cursos, administración de los catálogos del sistema y su cambio de contraseña.

Complejidad:
Baja.

Diagrama de caso de uso:

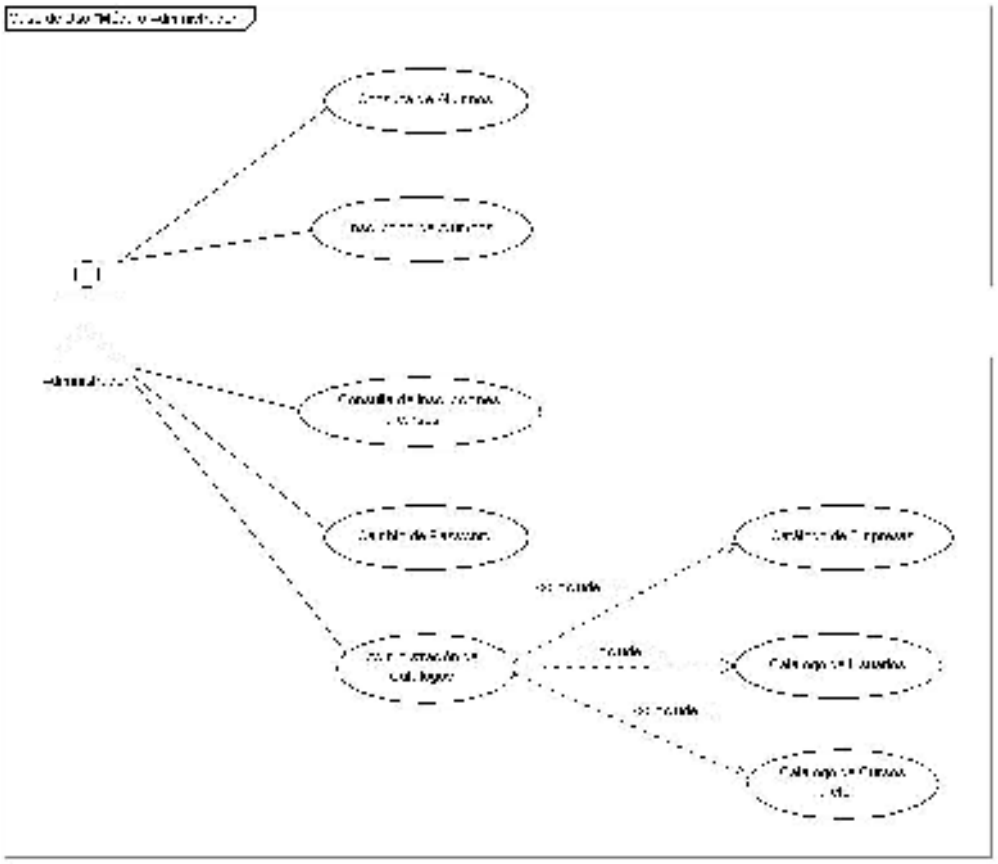


Figura 4.2.2.1 Caso de uso “Modulo administrativo”.

Diagrama de secuencia (inscripción de alumnos):

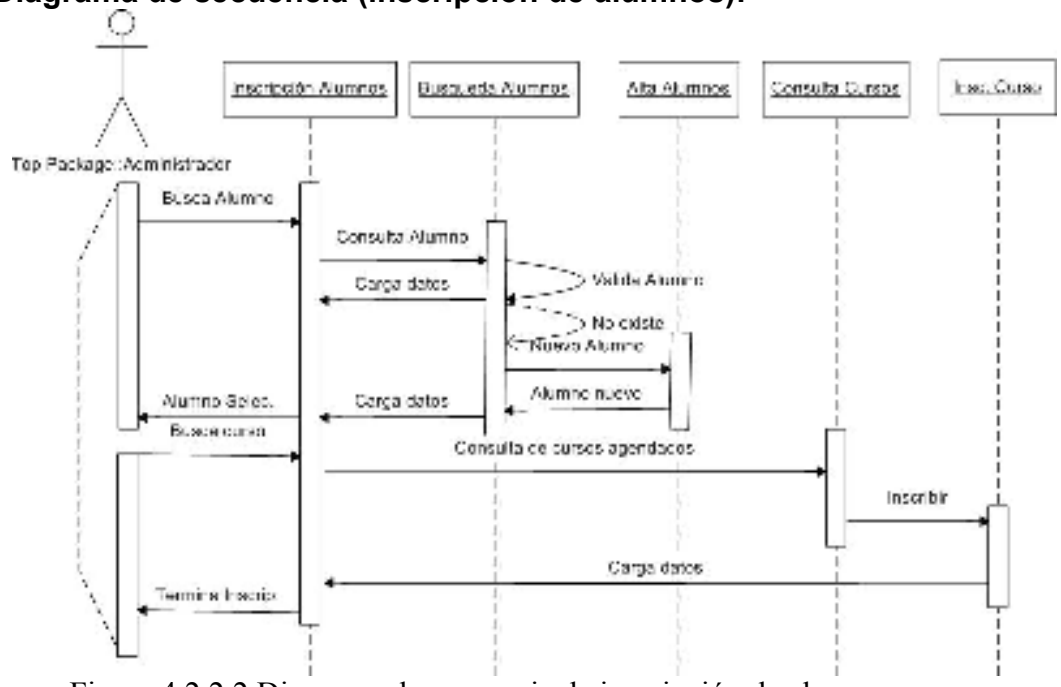


Figura 4.2.2.2 Diagrama de secuencia de inscripción de alumnos.

Flujo principal de eventos:

1. El administrador realiza el alta del alumno a través del catálogo de usuarios.
2. El administrador podrá realizar la preinscripción o inscripción de un alumno a algún curso agendado.
3. El administrador podrá realizar la consulta de alumnos inscritos a los cursos que se imparten en el centro de capacitación.
4. El administrador podrá cambiar su contraseña.
5. El administrador podrá realizar cualquier cambio de datos a través de la administración de catálogos.

Flujo alterno de eventos:

2.a El administrador podrá preinscribir al alumno.
2.b El administrador podrá inscribir al alumno.
2.c El administrador podrá dar de baja al alumno.

3.a El administrador podrá consultar a los alumnos inscritos.
3.b El administrador podrá quitar a los alumnos de los cursos inscritos.

Precondiciones:

1. El usuario debe estar firmado en el sistema.
2. El usuario debe ser administrador.

4.2.2.2 Caso de uso “Módulo alumno”

Descripción:

El sistema requiere tener la funcionalidad de permitir al usuario con rol de Alumno, realizar un examen inicial, un examen final, tener acceso a los foros de discusión, consultar su expediente, consultar sus inscripciones a cursos y el cambio de su contraseña.

Complejidad:
Baja.

Diagrama de caso de uso:

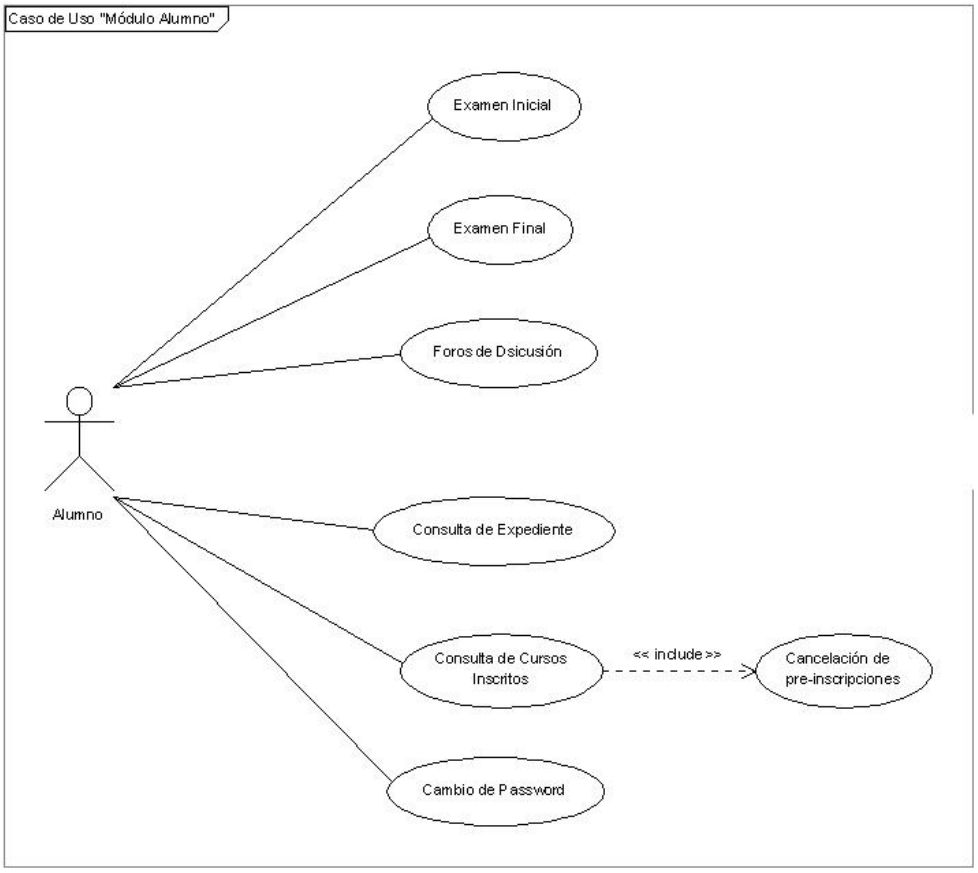


Figura 4.2.2.3 Caso de uso "Módulo alumno".

Diagrama de secuencia (Consulta de cursos inscritos):

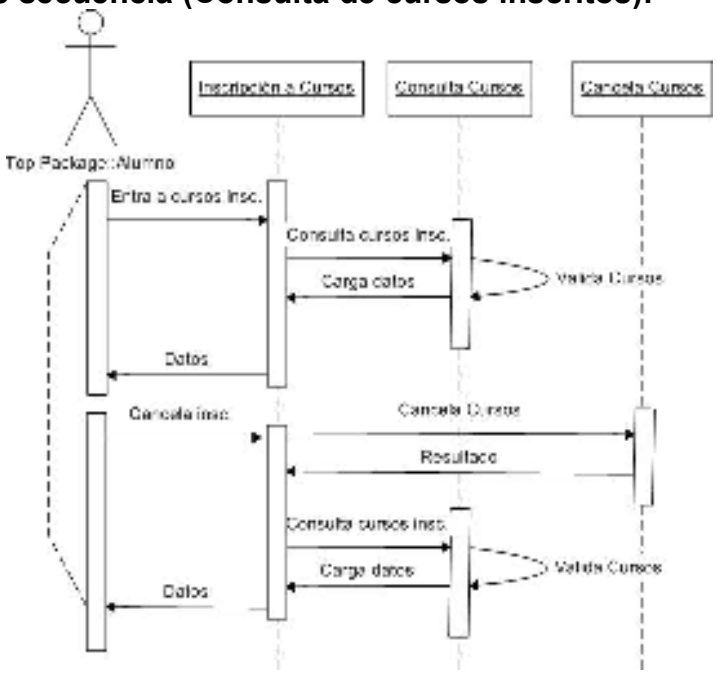


Figura 4.2.2.4 Diagrama de secuencia "Consulta de cursos inscritos".

Flujo principal de eventos:

1. El alumno realiza el examen inicial del curso al que está inscrito en un curso.
2. El alumno realiza el examen final del curso al que está inscrito en un curso.
3. El alumno podrá realizar la consulta de las calificaciones de las evaluaciones iniciales y finales que ha realizado este.
4. El alumno podrá entrar a los foros de discusión general y de los cursos en que esté inscrito.
5. El alumno podrá realizar la consulta de los cursos a los que se encuentra inscrito.
6. El alumno podrá cambiar su contraseña.

Flujo alterno de eventos:

5.a El alumno podrá realizar consultar sus inscripciones a cursos.
5.b El alumno podrá realizar cualquier cancelación de algún curso al que haya estado inscrito.

Precondiciones:

1. El usuario debe estar firmado en el sistema.
2. El usuario debe de ser alumno.

4.2.2.3 Caso de uso “Módulo usuario web”

Descripción:

El sistema requiere tener la funcionalidad de permitir al usuario con rol de Usuario Web (el usuario que entra por Internet), realizar las consultas de cursos y planes de carreras impartidos en el centro de capacitación, realizar exámenes de colocación, consulta de noticias, pre-inscripción a cursos o planes de carrera, así como también poder acceder al sistema en caso de que ya esté dado de alta en el sistema.

Complejidad:
Media.

Diagrama de caso de uso:

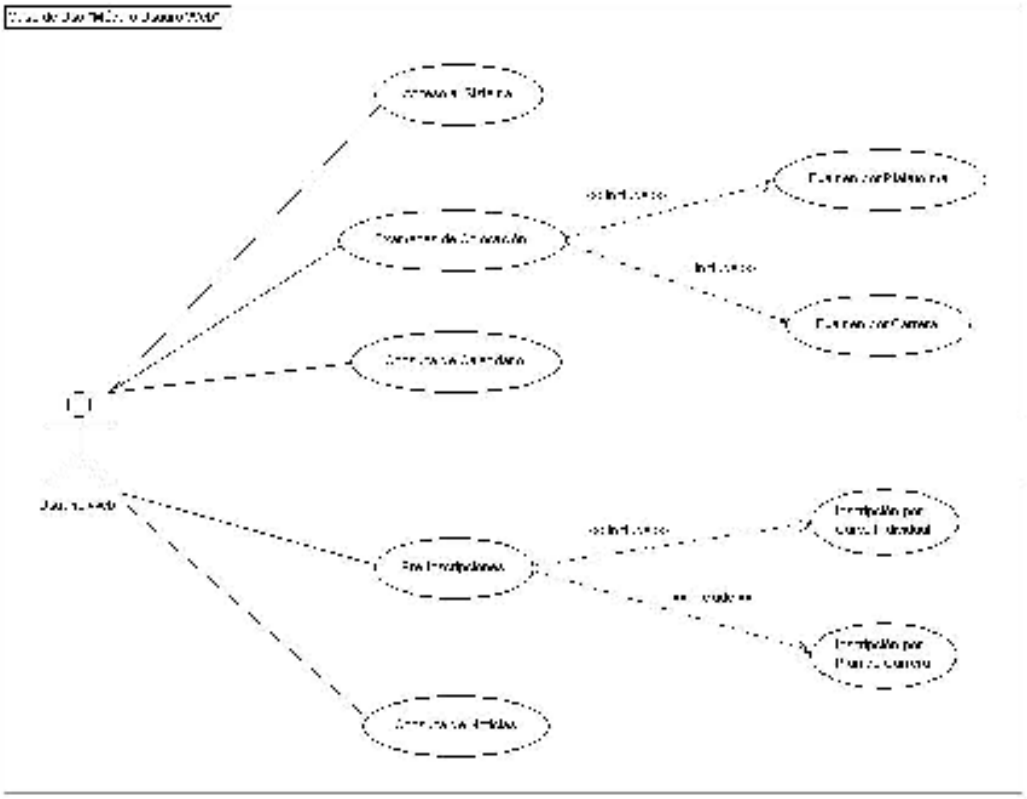


Figura 4.2.2.5 Caso de uso “Módulo usuario red”.

Diagrama de secuencia (Preinscripción cursos web):

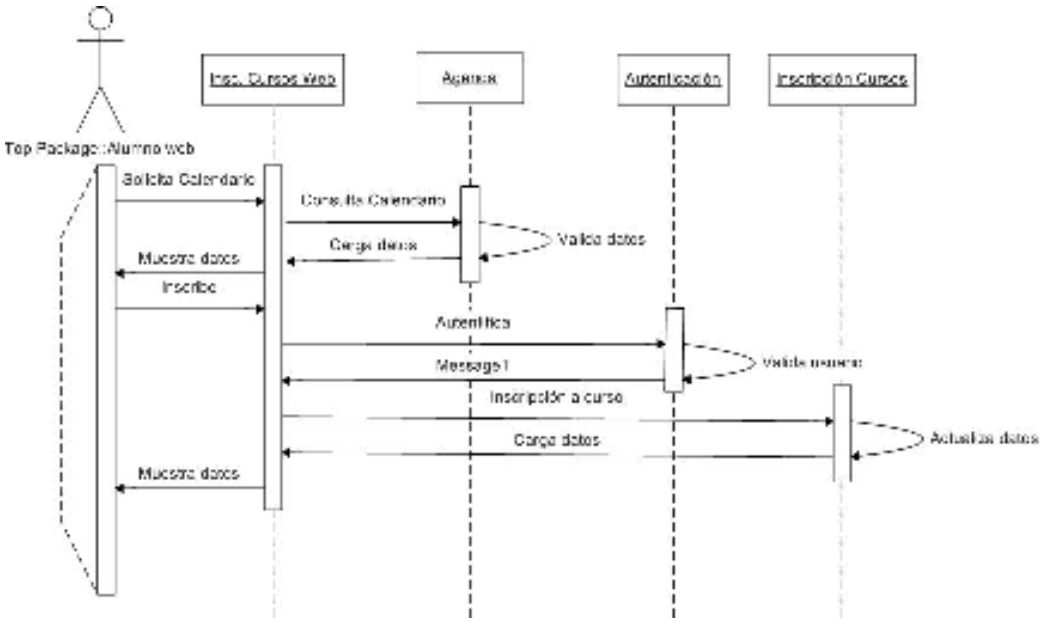


Figura 4.2.2.6 Diagrama de secuencia “Preinscripción cursos web”.

Flujo principal de eventos:

1. El alumno entra al sitio web del centro de capacitación vía Internet.
2. El alumno entra al módulo de examen de colocación:
   2.1 El alumno puede realizar el examen por plataforma.
   2.2 El alumno puede realizar el examen por carrera.
3. El alumno podrá realizar la consulta de los cursos que se imparten en el centro de capacitación, que se encuentren calendarizados hasta el momento.
4. El alumno podrá preinscribirse al centro de capacitación de 2 formas:
   4.1 Podrá inscribirse a un solo curso en particular.
   4.2 Podrá tomar preinscribirse a una serie de cursos (plan de carrera).
5. El alumno podrá realizar la consulta de las noticias que publica el centro de capacitación.

Flujo alterno de eventos:

4.1.a El alumno podrá revisar el currículo del instructor que impartirá el curso.
4.1.b El alumno podrá ver el temario del curso a tomar.

4.2.a El alumno podrá revisar el currículo del instructor que impartirá el curso.
4.2.b El alumno podrá ver el temario del curso a tomar.

Precondiciones:
1. No aplica.

4.2.2.4 Caso de Uso "Módulo instructor"

Descripción:

El sistema requiere tener la funcionalidad de permitir al usuario con rol de Instructor, tener acceso a los foros de discusión, consultar y modificar las calificaciones de sus alumnos y el cambio de su contraseña.

Complejidad:
Media.

Diagrama de caso de uso:

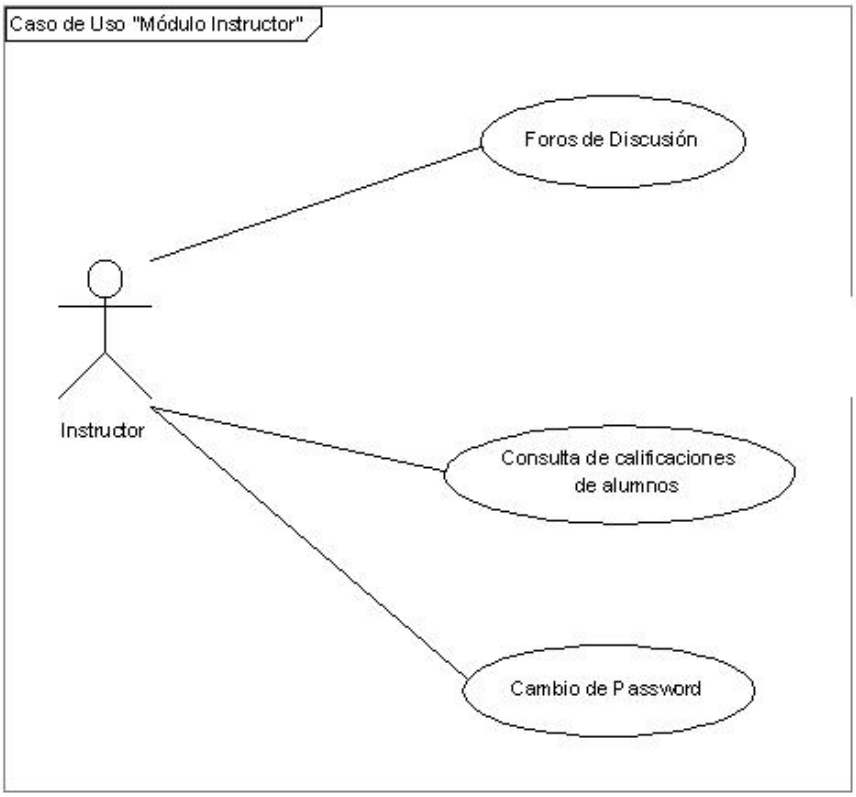


Figura 4.2.2.7 Caso de uso “Modulo instructor”.

Diagrama de secuencia (Consulta calificaciones):

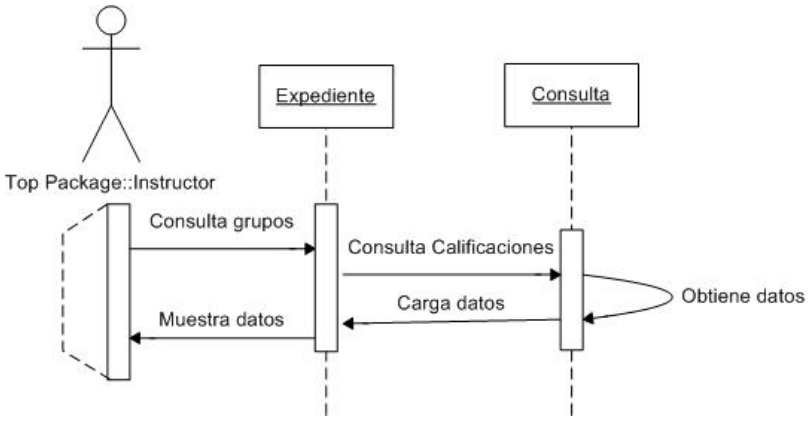


Figura 4.2.2.8 Diagrama de secuencia “Consulta Calificaciones”.

Flujo principal de eventos:

1. El instructor podrá entrar al módulo de foros de discusión.
2. El alumno podrá entra al módulo de consulta de calificaciones:
   2.1 El instructor puede consultar las calificaciones de sus alumnos.
   2.2 El instructor puede modificar las calificaciones de sus alumnos.
3. El alumno podrá cambiar su contraseña.

Flujo alterno de eventos:

No aplica.

Precondiciones:

1. El usuario debe estar firmado en el sistema.
2. El usuario debe de ser instructor.

4.2.2.5 Caso de uso "Módulo empresa"

Descripción:

El sistema requiere tener la funcionalidad de permitir al usuario con rol de empresa, tener acceso a la consulta de las calificaciones de sus empleados (alumnos) y el cambio de su contraseña.

Complejidad:
Media.

Diagrama de caso de uso:

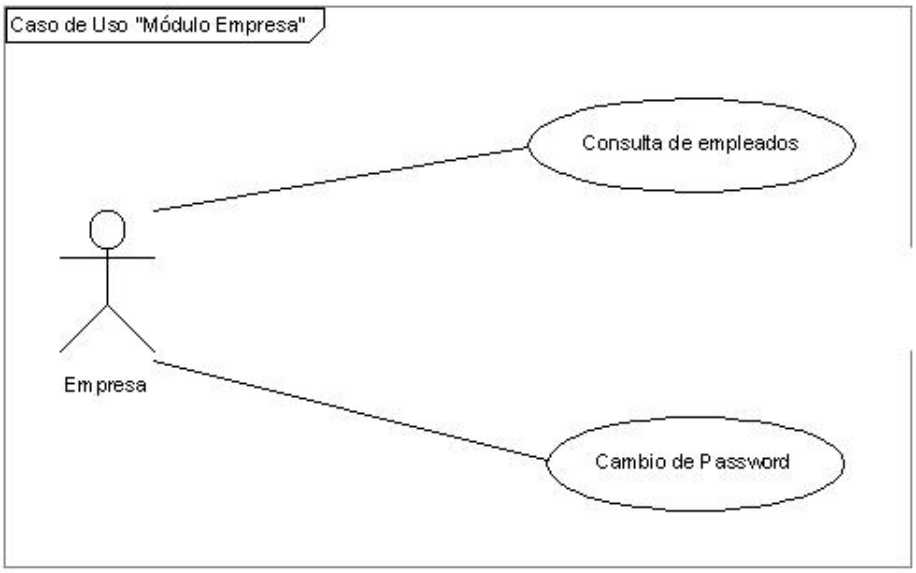


Figura 4.2.2.9 Caso de uso "Módulo empresa".

Flujo principal de eventos:

1. La empresa podrá entrar al módulo de foros consulta de empleados.

Flujo alterno de eventos:

2. El alumno podrá cambiar su contraseña.

Precondiciones:

1. El usuario debe estar firmado en el sistema.
2. El usuario debe ser empresa.

4.3 Diseño

4.3.1 Modelo de base de datos relacional

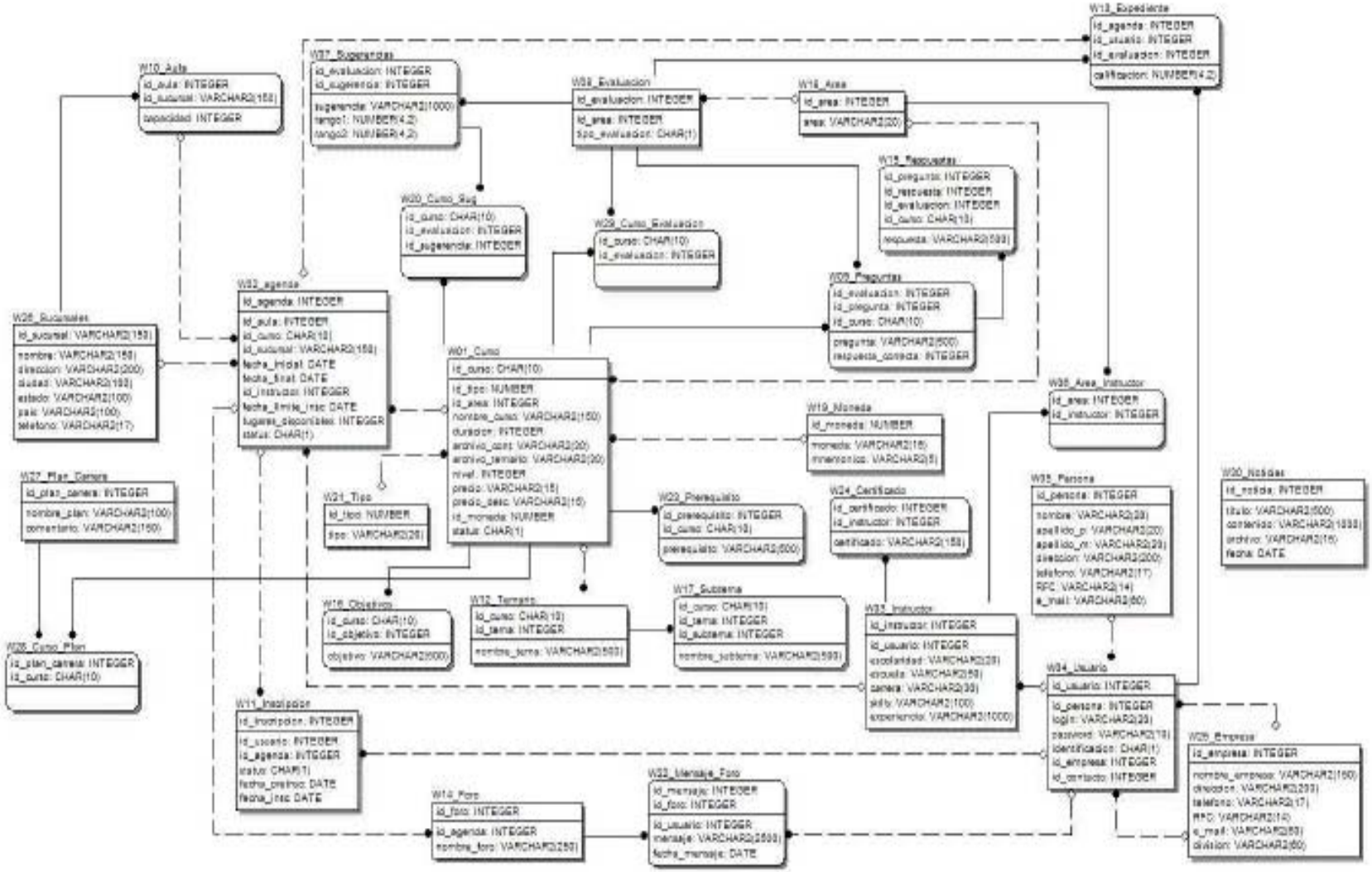


Figura 4.3.1.1 Modelo de base de datos relacional.

4.3.2 Parte publicada en internet:

Pantalla principal del sistema de automatización del proceso de preinscripción y gestión de cursos vía web que será mostrada al usuario web, es decir, es la parte que será publicada en Internet y que cualquier persona podrá tener acceso.

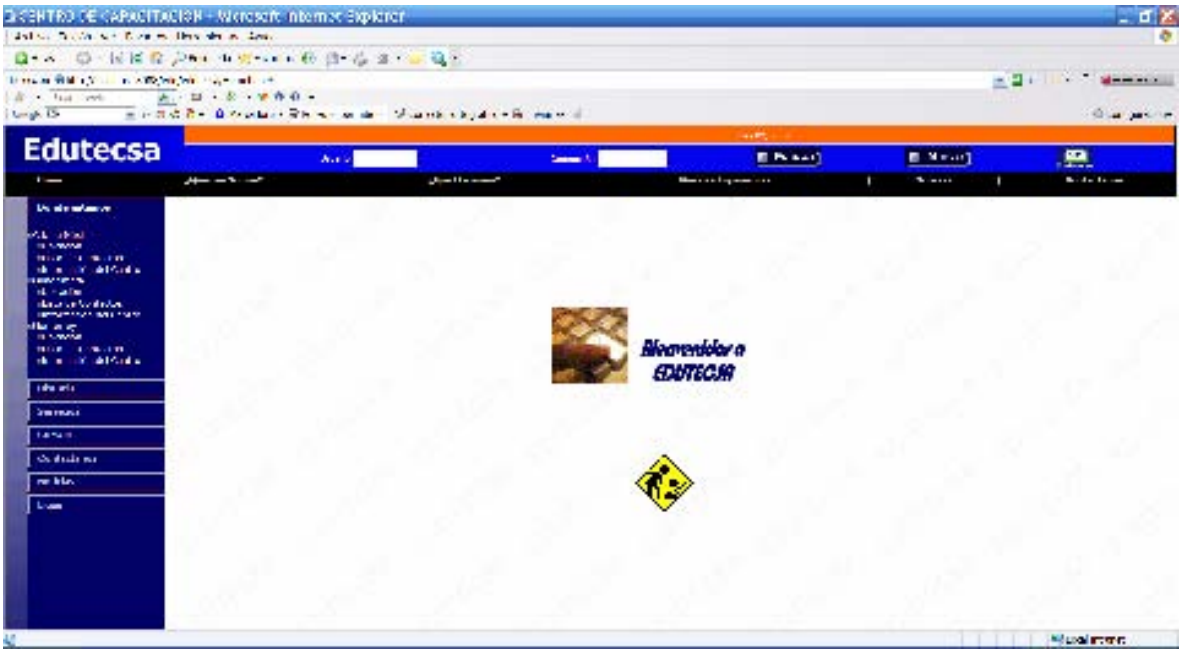


Figura 4.3.1.2. Pantalla principal.

Pantalla donde un usuario web puede darse de alta en el sistema, para así poder inscribirse a cursos o planes de carrera, y obtener los beneficios del sistema.

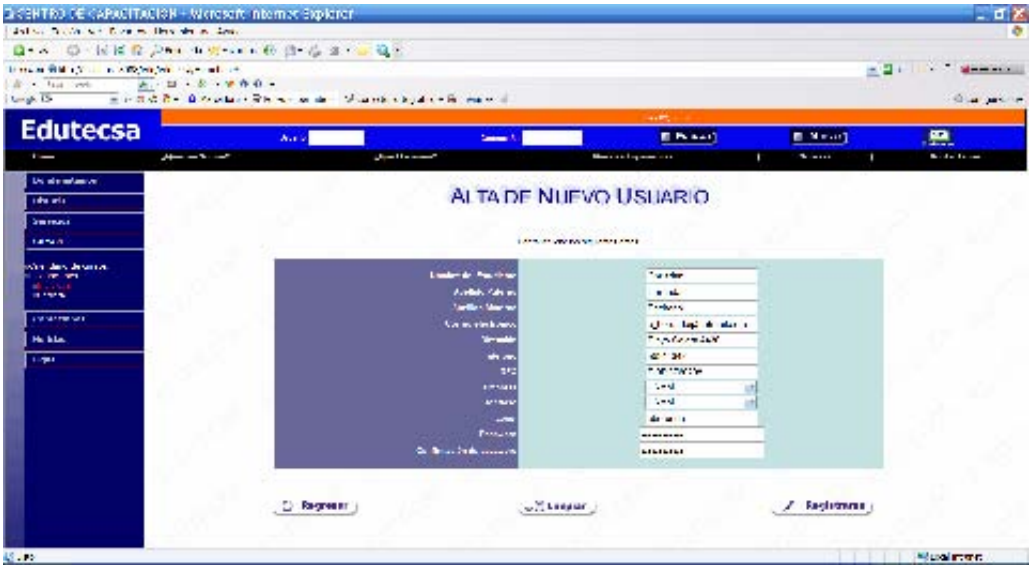


Figura 4.3.1.3. Alta de usuario.

Módulo de cursos por plataforma donde el usuario Web podrá consultar sin necesidad de que esté dado de alta en el sistema. En este módulo se muestra toda la información relacionada a los cursos por Plataforma, tales como nombre del curso, duración, objetivos, temario y el calendario con los días en los cuales se impartirá dicho curso, mostrando también el instructor que lo impartirá y teniendo un vinculo para que el interesado pueda ver el currículo del instructor.

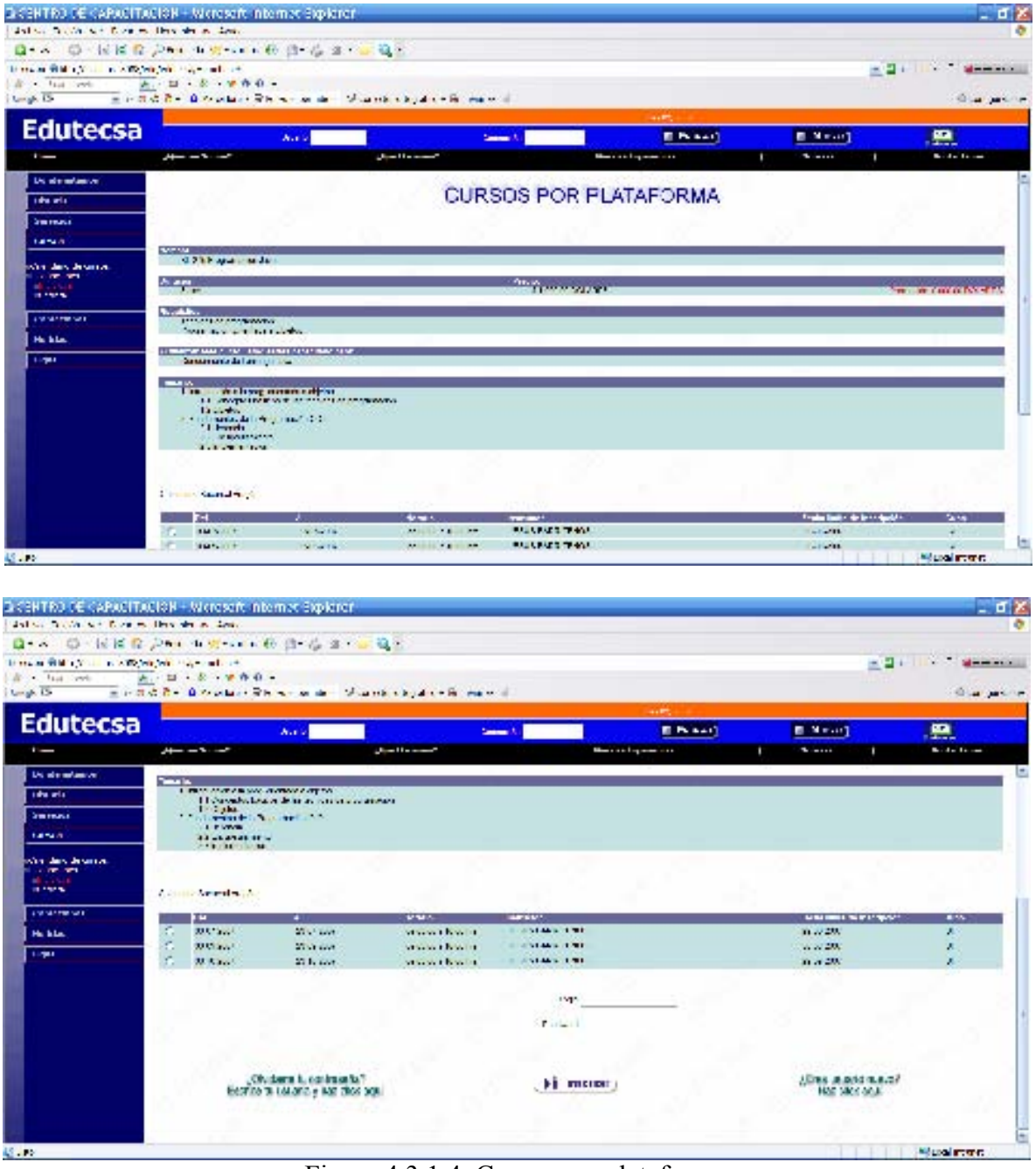


Figura 4.3.1.4. Cursos por plataforma.

Pantalla que muestra el currículo del instructor. Para llegar a esta, habrá que dar clic en la liga con el nombre del instructor que aparece en el calendario.

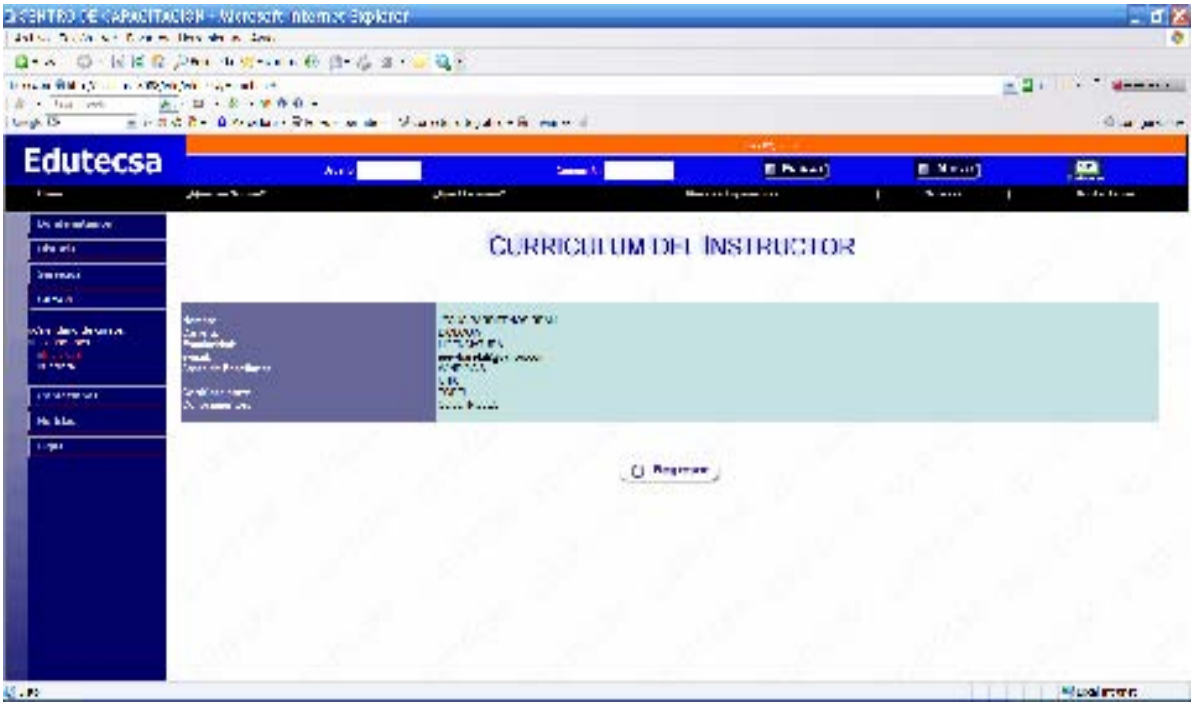


Figura 4.3.1.5. Curriculo del instructor.

4.3.3 Parte del BackOffice:

Pantalla principal del BackOffice, que se mostrará en la Intranet del centro de capacitación, aquí podrán acceder al sistema los usuarios que tienen como perfiles el de administrador, alumno, instructor y el de empresa.

Figura 4.3.1.6. Pantalla principal del BackOffice.

Pantalla del menú del administrador donde tiene acceso a la consulta de alumnos, inscripciones, consulta de inscripciones y catálogos.

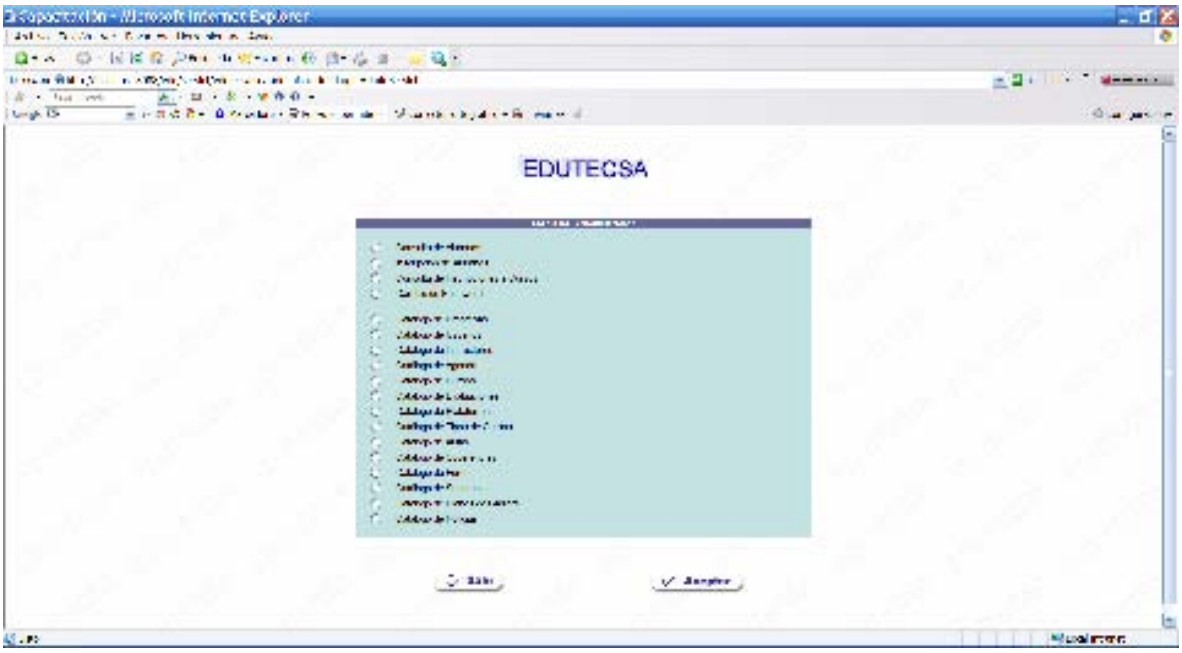


Figura 4.3.1.7. Menú del administrador.

Pantalla de consulta de alumnos que tiene acceso el administrador, en la cual también puede preinscribir a los alumnos en los cursos que hayan seleccionado.

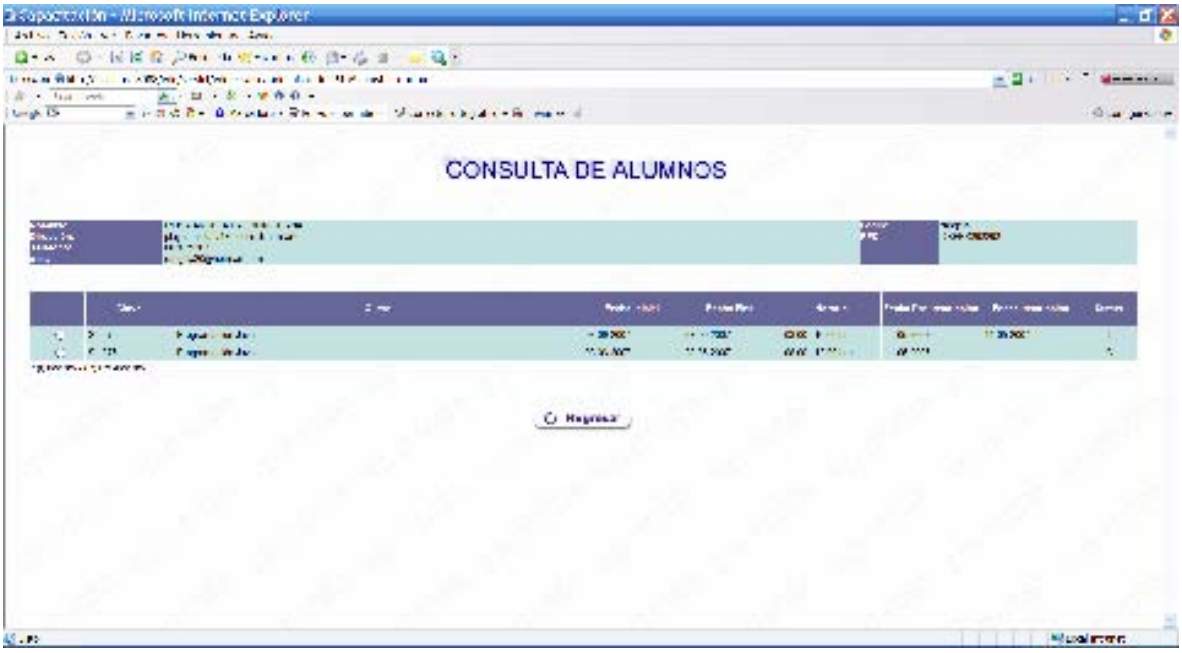


Figura 4.3.1.8. Consulta de alumnos.

Pantallas de catálogo de usuarios y catálogo de agendas que utiliza el usuario con perfil de administrador, para poder manejar los datos del sistema.

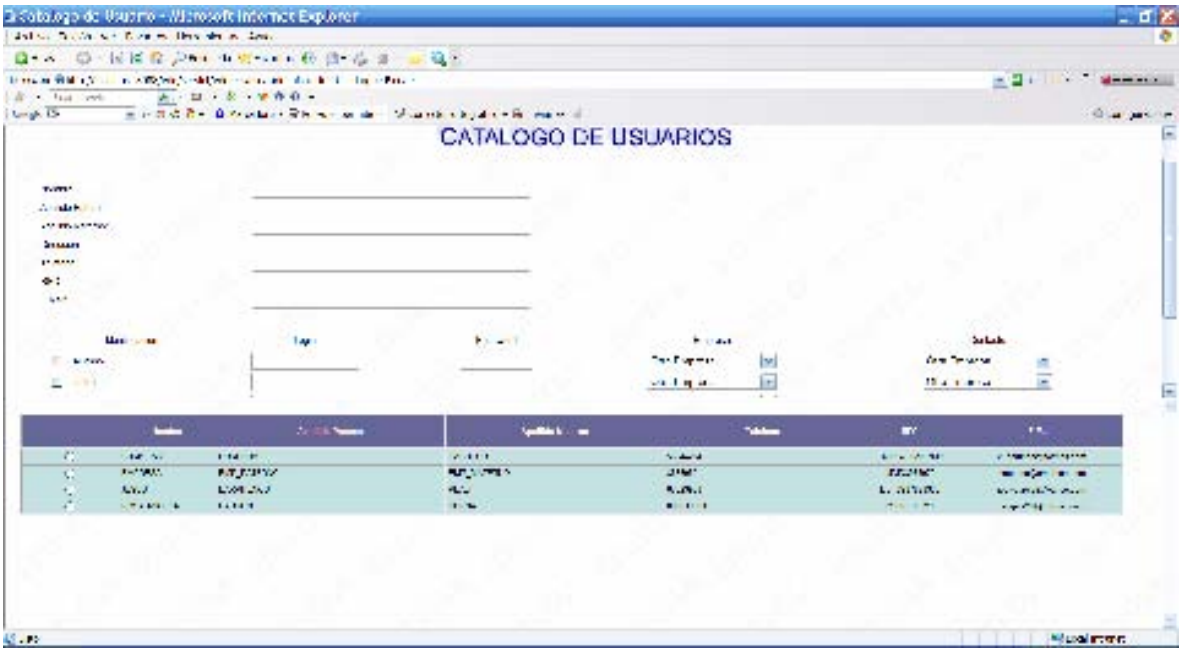


Figura 4.3.1.9. Catálogo de usuarios.

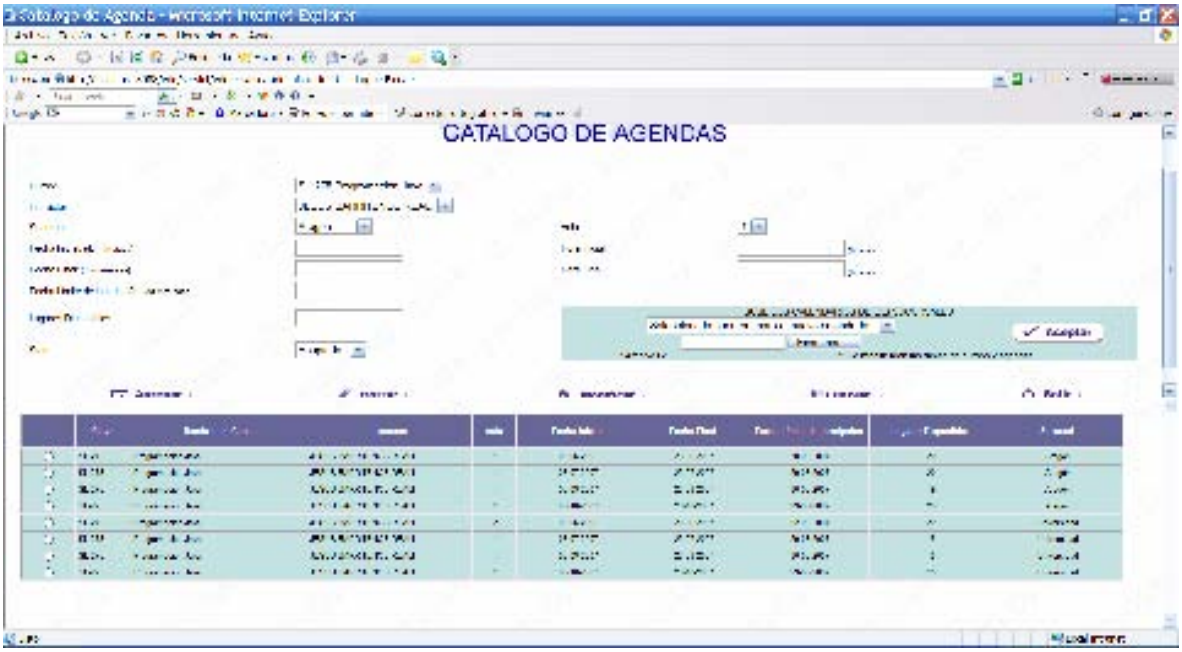


Figura 4.3.1.10. Catálogo de agendas.

Pantalla de menú del usuario con perfil de alumno, en el cual se presentan opciones como examen inicial y final, foros de discusión, expediente del alumno, consulta y cancelación de cursos y cambios de *password*.

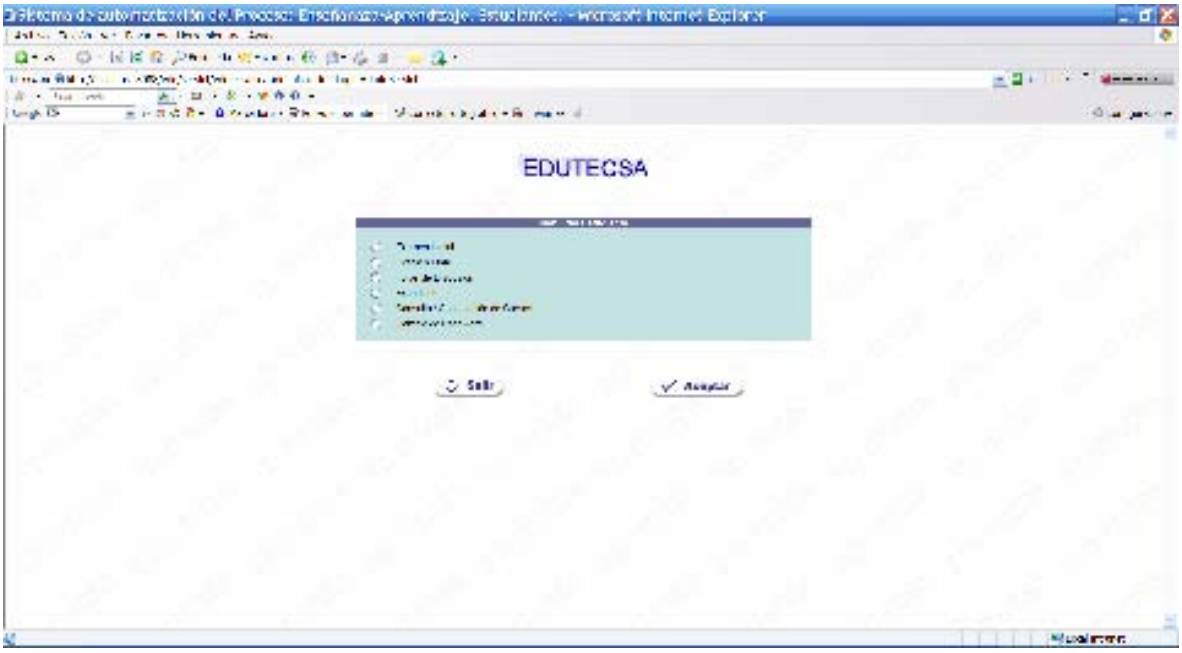


Figura 4.3.1.11. Menú del usuario (alumno).

Pantalla de foros de discusión, los cuales son de 2 tipos, públicos o privados, los públicos pueden ser accedidos desde Internet, y los privados solo desde la parte del BackOffice pueden ser accedidos por administradores, alumnos e instructores.

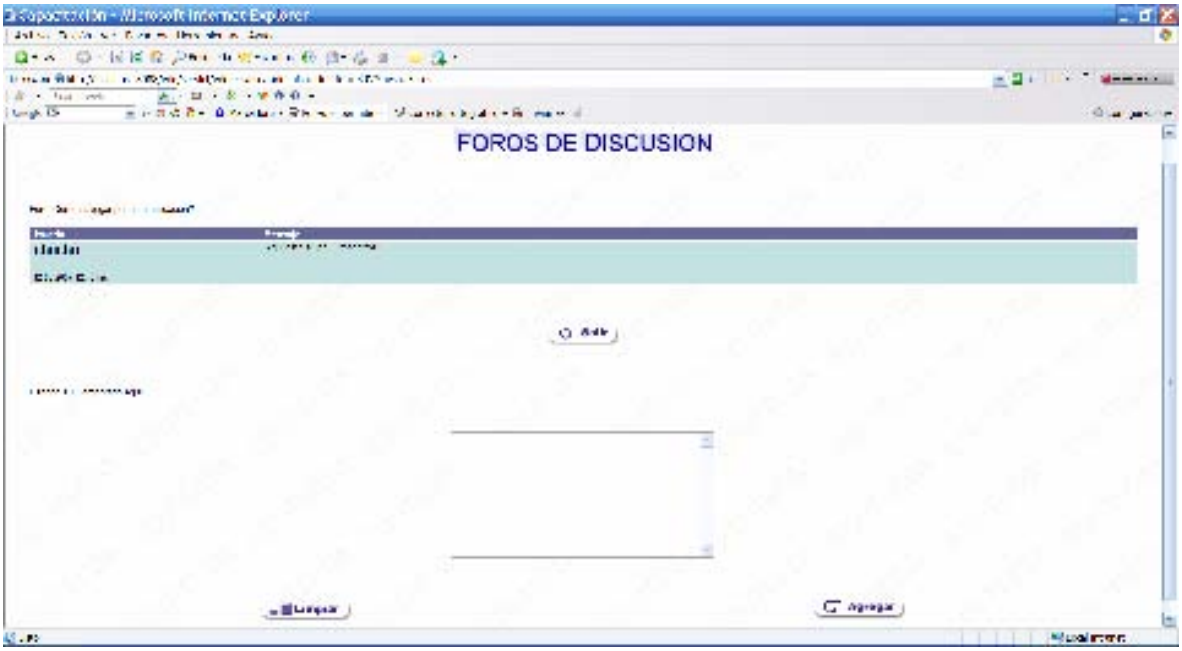


Figura 4.3.1.12. Foros de discusión.

4.3.4 Diagrama de clases

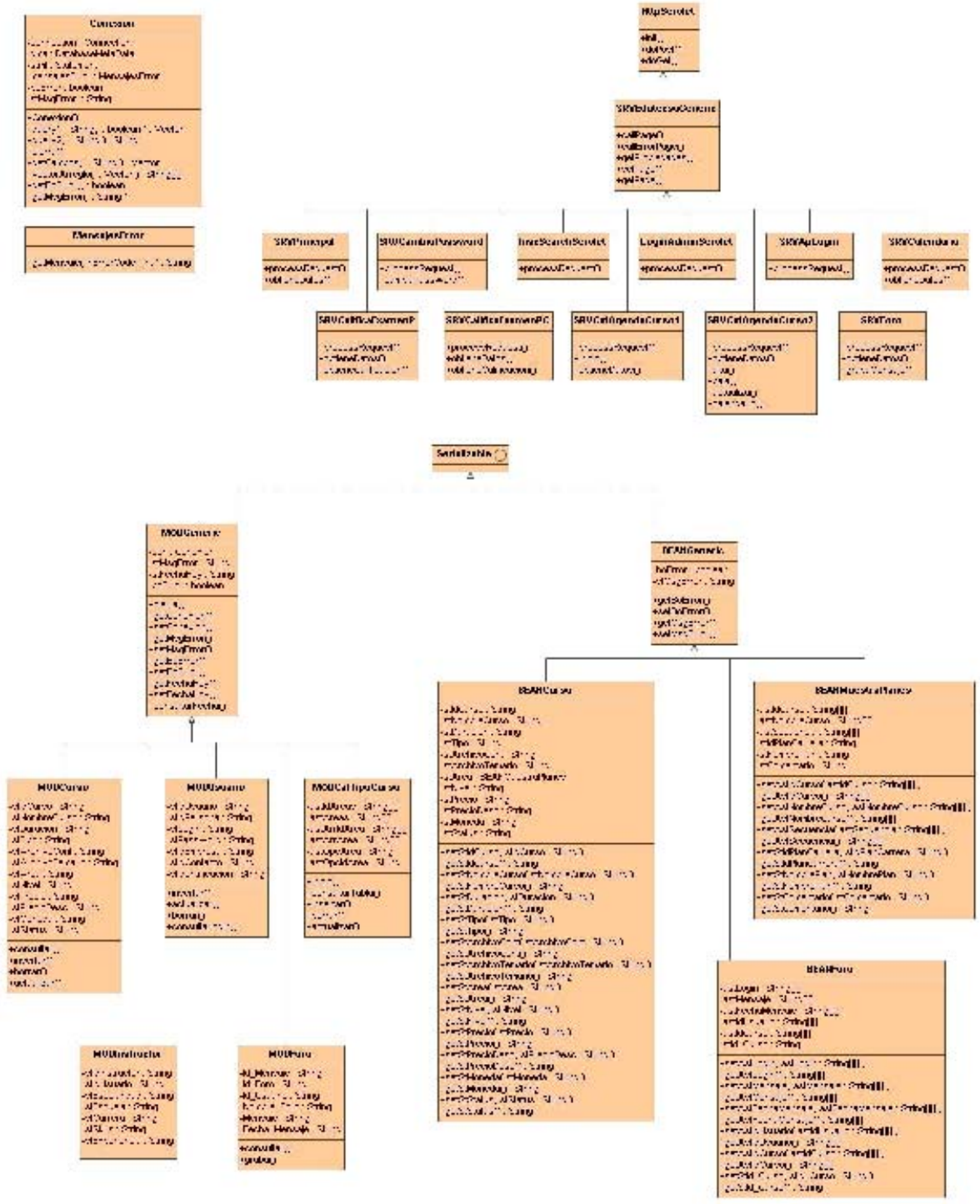


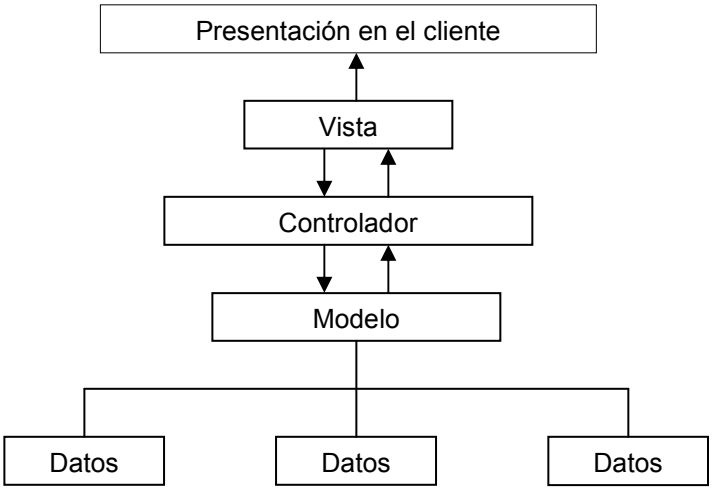
Figura 4.3.2.1 Diagrama de clases

4.4 Desarrollo

El desarrollo de una aplicación empresarial es complejo, ya que la aplicación debe ser capaz de dar servicio a muchos clientes de diversos tipos en forma simultánea a través de una infraestructura distribuida. Además la aplicación debe ser escalable de forma que pueda proporcionar un rendimiento aceptable sin importar el aumento en el número de clientes que la utilizan.

El sistema de automatización del proceso de preinscripción y gestión de cursos vía web fue desarrollado con el lenguaje de programación Java y se utilizaron tecnologías como JSP's, Servlets, HTML, JavaScript, CSS[1] (hojas de estilo), JDBC con SQL y utilizando como base de datos Oracle.

Este sistema fue desarrollado utilizando el patrón de diseño Modelo-Vista-Controlador (MVC[2]), este patrón consiste en dividir la aplicación en 3 capas, la capa de Modelo es la encargada de realizar la regla de negocio, generalmente esta es la que obtiene la información de la base de datos, está capa está compuesta por elementos de tipo *Java Beans*, la capa de Vista es la encargada de presentar la información al cliente, es representada por JSP's y la capa de Controlador que utiliza clases de tipo *Servlets* es la que lleva la lógica de negocio como se muestra en la siguiente figura:



La estrategia MVC divide la aplicación en una clase modelo, una clase vista y una clase controlador, la cual coordina las actividades entre las otras dos clases.

[1] *Cascading Style Sheets* (CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML).
[2] Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

A continuación se muestra la clase Conexión.java, la cual es la encargada de crear una conexión a la BD, ejecutar sentencias de SQL de los tipos “select”, “insert”, “update” y “delete” que son las indispensables para que tenga un correcto funcionamiento la aplicación. Esta clase utiliza el API de JDBC (*Java DataBase Connectivity*) creando una conexión con el servidor de BD y cerrándola por cada transacción.

```
package edutecsa;

import java.sql.*;
import java.util.*;
import edutecsa.*;
/*
Clase que realiza conexion con la
base de datos con el driver de Oracle
*/

public class Conexion
{
      private Connection connection;
      private DatabaseMetaData dma;
      private Statement stmt;
      private MensajesError mensajesError = null;
      private boolean boError = false;
      private String stMsgError = "";

      public Conexion()
      {
            try
            {
            ResourceBundle                  recursos                  =
PropertyResourceBundle.getBundle("edutecsa.srv.propiedades");

            DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());
            connection                                                =
DriverManager.getConnection(recursos.getString("direccionDB")+":"+
            recursos.getString("puertoDB")+":"+recursos.getString("i
      nstanciaDB"),
            recursos.getString("loginDB"),
      recursos.getString("passwordDB"));

            connection.setAutoCommit(true);
            stmt = connection.createStatement();

            mensajesError = new MensajesError();
            }
            catch(Exception iae)
            {
                  System.out.println("Excepcion ocurrida al efectuar
la conexion a la base de datos.");
```

```
                iae.printStackTrace();
            }
      }


/**
Metodo que ejecuta un query y devuelve el resultado en un vector.
El vector contiene arreglos del tamaño de las columnas solicitadas
en el query.
Si hay error, el vector contiene un arreglo con el string ERRORBD
en la primera posicion (astDatos[0]) y el mensaje de error en la
segunda (astDatos[1])
Esto solo sirve para Select
*/
      public Vector query(String sentencia, boolean condicion)
      {
            boError = false;

            ResultSet rs = null;
            Vector vtResultado = new Vector();
            String astDatos[] = null;
            String strResultado="";

            try
            {
                  rs = stmt.executeQuery(sentencia);
                  ResultSetMetaData rsmd = rs.getMetaData();
                  int numCols = rsmd.getColumnCount();
                  while (rs.next())
                  {
                        //Crea un arreglo del tamaño de las columnas
                        astDatos = new String[numCols];
                        for (int i=1; i<=numCols; i++)
                        {//Obtiene   el   resultado   de   la   columna
especificada,lo asigna al arreglo
                              strResultado = rs.getString(i);
                              if(strResultado == null)
                                    strResultado = "";
                              astDatos[i-1] = strResultado;
                              System.out.println("DATO   "+astDatos[i-
1]);
                        }

                        //Agrega el arreglo al vector
                        vtResultado.addElement(astDatos);
                  }
                  rs.close();
            }
            catch(SQLException sqle)
            {//Si hubo un error al ejecutar el query pone la bandera
en true y el error en el string.
                  boError = true;
```

```
                stMsgError                                              =
mensajesError.getMensaje(sqle.getErrorCode());
                stMsgError = sqle.getMessage();
                if( rs != null )
                     try{rs.close();}catch(SQLException sql){}
           }
           return vtResultado;
     }

/*
Metodo que ejecuta un query
Este metodo funciona para Update, Insert y Delete
Si la operacion se realizo con exito el metodo no regresa nada
Si la operacion fallo el metodo regresa el mensaje de error
*/
     public String query2(String sentencia)
     {
           boError = false;
           ResultSet rs = null;
           String strResultado = "";
           System.out.println("Query2: " + sentencia);

           try
           {
                rs = stmt.executeQuery(sentencia);
                rs.close();
           }
           catch(SQLException sqle)
           {//Si hubo un error al ejecutar el query pone la bandera
en true y el error en el string.
                System.out.println("Query2: " + sentencia);
                System.out.println("Error: " + sqle);
                boError = true;
                stMsgError                                              =
mensajesError.getMensaje(sqle.getErrorCode());
                stMsgError = sqle.getMessage();
                if( rs != null )
                     try{rs.close();}catch(SQLException sql){}
           }

           return strResultado;
     }

/*
Metodo que cierra la conexion con la base de datos
*/
     public void cierra(){
           try{
                connection.close();
           }catch(Exception e){e.printStackTrace();}
     }
```

```
	public Vector getCampos(String resultadoQuery){
		Vector datos= new Vector();
		try
		{
			StringTokenizer token = new
StringTokenizer(resultadoQuery,"|");
			while(token.hasMoreTokens())
			{
				Vector interno = new Vector();
				String str = new String();
				str = token.nextToken("|");
				StringTokenizer token2 = new
StringTokenizer(str,"*");
				while(token2.hasMoreTokens())
				{

	interno.addElement(token2.nextToken("*"));
				}
				datos.addElement(interno);
			}
		}
		catch(Exception e)
		{
		}
		return datos;
	}//getColumnas

	public String[][] vectorArreglo(Vector vt) {
		String arrDatos[][] = null;

		if (vt != null)
		{

			if(vt.size() > 0)

			{
				arrDatos = new String [vt.size()][];
				for(int i = 0; i < vt.size(); i++)
				{
					arrDatos[i] = (String[])vt.elementAt(i);
				}
			}
		}
		return arrDatos;
	}
	public boolean getBoError(){
		return boError;
	}
	public String getMsgError(){
		return stMsgError;
	}
}
```

La clase SRVEdutecsaGeneric.java es la clase de la cual van a heredar todos los *servlets* de la aplicación, los cuales van a llevar la lógica de negocio del sistema, estas clases van a formar la capa de Controlador en el patrón de MVC.

```
package edutecsa.srv;

import java.util.*;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import edutecsa.bean.BEANGeneric;

public class SRVEdutecsaGeneric extends HttpServlet
{
     public     void     callPage(HttpServletRequest     request,
HttpServletResponse response, String stPage) throws Exception
     {
          ResourceBundle recursos = getPropiedades();
          String stContexto = recursos.getString("contexto");

          if(stPage.startsWith(stContexto))
          {
               stPage    =    stPage.substring(stContexto.length(),
stPage.length());
          }

          RequestDispatcher              dispatcher              =
getServletContext().getRequestDispatcher(stPage);
          dispatcher.forward(request, response);
     }

     public    void    callErrorPage(HttpServletRequest    request,
HttpServletResponse response,
                               String stPage, Exception exception)
throws Exception
     {
          ResourceBundle recursos = getPropiedades();
          BEANGeneric bEANGeneric = new BEANGeneric();
          String stMensage = exception.toString();
          stMensage =
          stMensage.startsWith("java.lang.Exception:")?stMensage.s
          ubstring(stMensage.indexOf(":")+1,
          stMensage.length()):stMensage;
          bEANGeneric.setMsgError(stMensage);
          request.setAttribute("bEANGeneric", bEANGeneric);
          request.setAttribute("stPageError", stPage);

          callPage(request,                                   response,
recursos.getString("direccionError"));
     }
```

```
     public ResourceBundle getPropiedades(){
          ResourceBundle                recursos                =
PropertyResourceBundle.getBundle("edutecsa.srv.propiedades");

          return recursos;
     }

     public void setPage(HttpSession session, String stPage){
          Stack                    stackPaginas                 =
(Stack)session.getValue("stackPaginas");
          stackPaginas.push(stPage);
          session.putValue("stackPaginas", stackPaginas);
     }

     public String getPage(HttpSession session){
          Stack                    stackPaginas                 =
(Stack)session.getValue("stackPaginas");
          return (String)stackPaginas.pop();
     }

}
```

La clase SRVPrincipal.java es una de los *servlets* del sistema que hereda de la clase SRVEdutecsaGeneric.java e implementa una funcionalidad específica.

```
package edutecsa.srv.inscripcion;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import edutecsa.srv.*;
import edutecsa.*;
import edutecsa.mod.inscripcion.*;
import edutecsa.bean.inscripcion.*;

public class SRVPrincipal extends SRVEdutecsaGeneric  {

     public      void      doGet(HttpServletRequest      request,
HttpServletResponse response){
          processRequest(request, response);
     }

     public      void      doPost(HttpServletRequest      request,
HttpServletResponse response){
          processRequest(request, response);
     }

     public   void   processRequest(HttpServletRequest   request,
HttpServletResponse response){
```

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

               //Hace que la session dure una hora sin actividad
por parte del usuario
               session.setMaxInactiveInterval(3600);

               ResourceBundle recursos = getPropiedades();
               String              stDireccionIMG              =
recursos.getString("direccionIMG");
               String              stDireccionSRV              =
recursos.getString("direccionSRV");
               String              stDireccionJSP              =
recursos.getString("direccionJSP");

               session.putValue("stDireccionIMG",
stDireccionIMG);
               session.putValue("stDireccionJSP",
stDireccionJSP);
               session.putValue("stDireccionSRV",
stDireccionSRV);

                    //response.setHeader("Pragma","no-cache");
               //response.setHeader("Cache-Control", "no-cache");
                    //response.setDateHeader("Expires",0);
               session.removeValue("idCurso");

                    obtieneDatos(request, response, session);
                    callPage(request,                   response,
recursos.getString("direccionJSP")+
"/jsp/inscripcion/JSPPrincipal.jsp
}catch(Exception e){
               try
               {
                    System.out.println("Error   en   SRVPrincipal:
"+e);
               }catch(Exception ee){}
    }

   }//processRequest

  public     void     obtieneDatos(HttpServletRequest     request,
HttpServletResponse response,HttpSession session){
         try{
           MODPrincipal mODPrincipal = new MODPrincipal();
           Conexion con = new Conexion();
           mODPrincipal.setConexion(con);
           mODPrincipal.consultar();
           if(con.getBoError())//Si existe un error en la base de
datos, pone la pagina de error.
           {
```

```
                System.out.println("Error en SRVPrincipal");
                con.cierra();
                return;
           }
           BEANPrincipal bEANPrincipal= new BEANPrincipal();

     bEANPrincipal.setBoError(mODPrincipal.getBoError());
     bEANPrincipal.setMsgError(mODPrincipal.getMsgError());
     bEANPrincipal.setIdAreas(mODPrincipal.getIdAreas());
     bEANPrincipal.setAreas(mODPrincipal.getAreas());
     bEANPrincipal.setIdSucursal(mODPrincipal.getIdSucursal());
     bEANPrincipal.setSucursal(mODPrincipal.getSucursal());
     con.cierra();

     request.setAttribute("bEANPrincipal",bEANPrincipal);
        }catch(Exception e)
             {
           try
                {
                     System.out.println("Error   en   SRVPrincipal:
"+e);
                     }catch(Exception ee){}
               }
  }//obtieneDatos

}
```

La clase MODGeneric.java es la clase genérica de la cual van a heredar las clases que tengan la función de obtener la información de la BD (reglas de negocio) y la cual se encuentra en la capa de “Modelo”.

```
package edutecsa.mod;

import java.util.Vector;
import java.io.*;
import edutecsa.*;

public class MODGeneric implements Serializable
{
     transient Conexion con = null;
     String stMsgError = null;
     String stFechaHoy = "";
     boolean boError = false;

     public MODGeneric(){  }

     public void cierra(){
          con.cierra();
     }

     public Conexion getConexion(){
```

```
            return con;
      }

      public void setConexion(Conexion con){
            this.con = con;
      }

      public String getMsgError(){
            return stMsgError;
      }

      public void setMsgError(String msg){
            stMsgError = msg;
      }

      public boolean getBoError(){
            return boError;
      }

      public void setBoError(boolean boErr){
            boError = boErr;
      }

      public String getFechaHoy(){
            return stFechaHoy;
      }

      public void setFechaHoy(String stFechaHoy){
            this.stFechaHoy = stFechaHoy;
      }

      public boolean consultarFecha(){
            String stQuery = "select to_char(sysdate, 'dd-mm-YYYY')
from dual";
            Vector vt = con.query(stQuery, true);
            if(con.getBoError())
                  return true;
            String astDatos[][] = con.vectorArreglo(vt);
            stFechaHoy = astDatos[0][0];
            return false;
      }

}
```

La clase MODCurso.java es una de las clases del sistema que hereda de la clase MODGeneric.java e implementa una funcionalidad específica.

```
package edutecsa.mod;

import java.util.Vector;
```

```
public class MODCurso extends MODGeneric
{
	private String stIdCurso		= "";
	private String stNombreCurso	= "";
	private String stDuracion		= "";
	private String stTipo			= "";
	private String stArchivoCont	= "";
	private String stArchivoTemario	= "";
	private String stArea			= "";
	private String stNivel			= "";
	private String stPrecio			= "";
	private String stPrecioDesc		= "";
	private String stMoneda			= "";
	private String stStatus			= "A";

	public MODCurso(){
	}

	public String getIdCurso(){
		return stIdCurso;
	}

	public void setIdCurso(String stIdCurso){
		this.stIdCurso = stIdCurso;
	}

	public String getNombreCurso(){
		return stNombreCurso;
	}

	public void setNombreCurso(String stNombreCurso){
		this.stNombreCurso = stNombreCurso;
	}

	public String getDuracion(){
		return stDuracion;
	}

	public void setDuracion(String stDuracion){
		this.stDuracion = stDuracion;
	}

	public String getTipo(){
		return stTipo;
	}

	public void setTipo(String stTipo){
		this.stTipo = stTipo;
	}

	public String getArchivoCont(){
		return stArchivoCont;
```

```
}

public void setArchivoCont(String stArchivoCont){
     this.stArchivoCont = stArchivoCont;
}

public String getArchivoTemario(){
     return stArchivoTemario;
}

public void setArchivoTemario(String stArchivoTemario){
     this.stArchivoTemario = stArchivoTemario;
}

public String getArea(){
     return stArea;
}

public void setArea(String stArea){
     this.stArea = stArea;
}

public String getNivel(){
     return stNivel;
}

public void setNivel(String stNivel){
     this.stNivel = stNivel;
}

public String getPrecio(){
     return stPrecio;
}

public void setPrecio(String stPrecio){
     this.stPrecio = stPrecio;
}

public String getPrecioDesc(){
     return stPrecioDesc;
}

public void setPrecioDesc(String stPrecioDesc){
     this.stPrecioDesc = stPrecioDesc;
}

public String getMoneda(){
     return stMoneda;
}

public void setMoneda(String stMoneda){
     this.stMoneda = stMoneda;
```

```
	}

	public String getStatus(){
		return stStatus;
	}

	public void setStatus(String stStatus){
		this.stStatus = stStatus;
	}


	public boolean consultar(){
		String astDatos[][];
		String stQuery = "select id_curso, nombre_curso,
duracion, id_tipo, archivo_cont, " +
		"area, nivel, precio, moneda, status, archivo_temario,
precio_desc from W01_Curso C,
		W18_Area A, W19_Moneda M " +
		"where id_curso = '" + getIdCurso() + "' " +
		"and C.id_area = A.id_area " +
		"and C.id_moneda = M.id_moneda";

		Vector vt = con.query(stQuery, true);
		if(con.getBoError())
			return true;

		astDatos = con.vectorArreglo(vt);
		if(vt != null && vt.size() > 0)
			{
				if(astDatos != null && astDatos.length > 0)
				{
					setIdCurso(astDatos[0][0]);
					setNombreCurso(astDatos[0][1]);
					setDuracion(astDatos[0][2]);
					setTipo(astDatos[0][3]);
					setArchivoCont(astDatos[0][4]);
					setArea(astDatos[0][5]);
					setNivel(astDatos[0][6]);
					setPrecio(astDatos[0][7]);
					setMoneda(astDatos[0][8]);
					setStatus(astDatos[0][9]);
					setArchivoTemario(astDatos[0][10]);
					setPrecioDesc(astDatos[0][11]);
				}
			}

		return false;
	}

	public boolean insertar(){
		String stQuery = "";
```

```
          stQuery = "insert into W01_Curso (id_curso,
nombre_curso, duracion, id_tipo,
          archivo_cont, id_area, nivel, precio, id_moneda, status,
archivo_temario, precio_desc)"+
          "values ('" + getIdCurso().toUpperCase().trim() + "', '"
+ getNombreCurso() + "', " +
          getDuracion() + ", '" +  getTipo()  +  "',  '"  +
     getArchivoCont().toUpperCase().trim() +
          "','"+getArea()+",  "  +  "'"+getNivel()+"',  '"  +
     getPrecio() + "', " + getMoneda() + ", '"+
          getStatus()       +       "',       '"       +
     getArchivoTemario().toUpperCase().trim() + "', '" +
          getPrecioDesc() + "')";

          stQuery = getConexion().query2(stQuery);
          if(getConexion().getBoError())
               return true;

          return getBoError();

     }

     public boolean borrar(){
          String stQuery = "";

          stQuery = "delete W23_Prerequisito where id_curso = '" +
getIdCurso() + "'";

          stQuery = getConexion().query2(stQuery);
          if(getConexion().getBoError())
               return true;

          stQuery = "delete W17_Subtema where id_curso = '" +
getIdCurso() + "'";

          stQuery = getConexion().query2(stQuery);
          if(getConexion().getBoError())
               return true;

          stQuery = "delete W12_Temario where id_curso = '" +
getIdCurso() + "'";

          stQuery = getConexion().query2(stQuery);
          if(getConexion().getBoError())
               return true;

          stQuery = "delete W16_Objetivos where id_curso = '" +
getIdCurso() + "'";

          stQuery = getConexion().query2(stQuery);
          if(getConexion().getBoError())
               return true;
```

```
            stQuery = "delete W01_Curso where id_curso = '" +
getIdCurso() + "'";

            stQuery = getConexion().query2(stQuery);
            if(getConexion().getBoError())
                  return true;

            return getBoError();

      }

      public boolean actualizar(){
            String stQuery = "";

            stQuery = "update W01_Curso set id_area = " + getArea()
+ ", " +"nombre_curso = '" +
            getNombreCurso() + "', " +"duracion = " + getDuracion()
+ ", " +
            "id_tipo = '" + getTipo() + "', " +
            "precio_desc = '" + getPrecioDesc() + "', " +
            "precio = '" + getPrecio() + "', " +
            "id_moneda = " + getMoneda() + ", " +
            "status = '" + getStatus() + "', " +
            "archivo_temario           =           '"           +
getArchivoTemario().toUpperCase().trim() + "' " +
            "where id_curso = '" + getIdCurso().toUpperCase().trim()
+ "'";
            stQuery = getConexion().query2(stQuery);
            if(getConexion().getBoError())
                  return true;

            return getBoError();

      }

}
```

La clase BEANGeneric.java es la clase genérica de los componentes de negocio de la aplicación, de la cual van a heredar las clases de componentes específicos y la cual se encuentran en la capa de “Modelo”.

```
package edutecsa.bean;

import java.util.*;
import java.io.*;

public class BEANGeneric implements Serializable
{
      private boolean boError = false;
      private String stMsgError = "";
```

```
	public BEANGeneric(){
	}

	public boolean getBoError(){
		return boError;
	}

	public void setBoError(boolean boError){
		this.boError = boError;
	}

	public String getMsgError(){
		return stMsgError;
	}

	public void setMsgError(String stMsgError) {
		this.stMsgError = stMsgError;
	}

}
```

La clase BEANCurso.java es una de las clases del sistema que hereda de la clase BEANGeneric.java e implementa una funcionalidad específica.

```
package edutecsa.bean;

import java.util.*;

public class BEANCurso extends BEANGeneric
{

	private String stIdCurso         = "";
	private String stNombreCurso     = "";
	private String stDuracion        = "";
	private String stTipo            = "";
	private String stArchivoCont     = "";
	private String stArea            = "";
	private String stNivel           = "";
	private String stPrecio          = "";
	private String stPrecioDesc      = "";
	private String stMoneda          = "";
	private String stStatus          = "";
	private String stArchivoTemario  = "";

	public BEANCurso(){
	}

	public String getIdCurso(){
		return stIdCurso;
	}
```

```
public void setIdCurso(String stIdCurso){
      this.stIdCurso = stIdCurso;
}

public String getNombreCurso(){
      return stNombreCurso;
}

public void setNombreCurso(String stNombreCurso){
      this.stNombreCurso = stNombreCurso;
}

public String getDuracion(){
      return stDuracion;
}

public void setDuracion(String stDuracion){
      this.stDuracion = stDuracion;
}

public String getTipo(){
      return stTipo;
}

public void setTipo(String stTipo){
      this.stTipo = stTipo;
}

public String getArchivoCont(){
      return stArchivoCont;
}

public void setArchivoCont(String stArchivoCont){
      this.stArchivoCont = stArchivoCont;
}

public String getArea(){
      return stArea;
}

public void setArea(String stArea){
      this.stArea = stArea;
}

public String getNivel(){
      return stNivel;
}

public void setNivel(String stNivel){
      this.stNivel = stNivel;
}
```

```
	public String getPrecio(){
		return stPrecio;
	}

	public void setPrecio(String stPrecio){
		this.stPrecio = stPrecio;
	}

	public String getPrecioDesc(){
		return stPrecioDesc;
	}

	public void setPrecioDesc(String stPrecioDesc){
		this.stPrecioDesc = stPrecioDesc;
	}

	public String getMoneda(){
		return stMoneda;
	}

	public void setMoneda(String stMoneda){
		this.stMoneda = stMoneda;
	}

	public String getStatus(){
		return stStatus;
	}

	public void setStatus(String stStatus){
		this.stStatus = stStatus;
	}

	public String getArchivoTemario(){
		return stArchivoTemario;
	}

	public void setArchivoTemario(String stArchivoTemario){
		this.stArchivoTemario = stArchivoTemario;
	}

}
```

Las clases mostradas anteriormente, forman parte de las capas de modelo y controlador, del patrón de diseño MVC. Para la capa de Vista, se utilizaron JSP's (*Java Server Pages*) en esta aplicación, las cuales son controladas por los *servlets* y mostrando la información a través de los *beans*, integrando así el patrón MVC. El siguiente código es un ejemplo de una JSP utilizada en el desarrollo del sistema.

```
<html>
<head>
<title>Capacitaci&oacute;n</title>

<style type="text/css">
<!--
-->
</style>
<%@                      page                        language="java"
import="edutecsa.bean.calendario.BEANCalendario" %>
<jsp:useBean          id="bEANCalendario"          scope="request"
class="edutecsa.bean.calendario.BEANCalendario" />
<link                                            rel="stylesheet"
href='<%=(String)session.getValue("stDireccionJSP")+"/jsp/styleedu
tecsa.css"%>' type="text/css">
<body
background='<%=(String)session.getValue("stDireccionIMG")+"/Fondo.
gif"%>' bgcolor='#FFFFFF' text="#663399">
<center>
   <table width="97%" border="0" cellspacing="0" cellpadding="0">
      <tr>
        <td> </td>
      </tr>
      <tr>
        <td>
          <div                                  align="center"><img
src='<%=(String)session.getValue("stDireccionIMG")%>/TituloCalenda
rio.gif' border="0">
            </div>
        </td>
      </tr>
    </table>
  </center>
<form name="forma" action="">
<input type="hidden" name="opcion" value="">
<center>
    <table width="97%" border="0" cellspacing="0" cellpadding="0">
          <% if(bEANCalendario.getPlataformas() != null) { %>
          <%                    for(int                     k=0;
k<bEANCalendario.getArrIdSucursal().length; k++) { %>
    <tr>
      <td> </td>
    </tr>
    <tr>
      <td> </td>
    </tr>
    <tr class="titulo1">
      <td
class="titulo1"> <%=bEANCalendario.getArrNombreSuc(k)%></td>
    </tr>
    <tr>
      <td> </td>
```

```
    </tr>
           <% int b = 0; %>
           <%                        for(int                        i=0;
i<bEANCalendario.getPlataformas().length; i++) { %>
           <%
if(bEANCalendario.getArrIdSucursal(k).equals(bEANCalendario.getIdS
ucursal(i))) { %>
           <% if(i==0) { %>
      <tr class="calendarioArea">
        <td
class="calendarioArea"> <%=bEANCalendario.getPlataformas(0)%>
</td>
      </tr>
      <tr>
        <td>
          <div align="center">
            <table    width="100%"    border="0"    cellspacing="1"
cellpadding="1">
              <tr class="calendarioCurso">
                <td width="4%" class="calendarioCurso"> </td>
                <td                                        width="11%"
class="calendarioCurso"> <%=bEANCalendario.getIdCursos(0)%></
td>
                <td                                        width="54%"
class="calendarioCurso"> <%=bEANCalendario.getNombreCursos(0)
%></td>
                <td width="13%" class="calendarioCurso"
                align="center"> <%=bEANCalendario.getDuracion
           (0)%> días</td>
                <td width="18%">
                <table   width="100%"   border="0"   cellspacing="0"
cellpadding="0">
           <%   for(int   j=0;   j<bEANCalendario.getFechas().length;
j++) { %>
                 <%
if(bEANCalendario.getIdCursos(j).equals(bEANCalendario.getIdCursos
(0)) &&

     bEANCalendario.getArrIdSucursal(k).equals(bEANCalendario.getI
dSucursal(j))){%>
               <tr class="calendarioCurso">
                          <td
class="calendarioCurso"> <%=bEANCalendario.getFechas(j)%></td
>
                    </tr>
           <% } } %>
                  </table>
                </td>
              </tr>
            </table>
          </div>
        </td>
```

```
      </tr>
     <% b = 1; } else { %>
     <%
if(!bEANCalendario.getPlataformas(i).equals(bEANCalendario.getPlat
aformas(i-1)) || b==0) { %>
         <tr class="calendarioArea">
           <td
class="calendarioArea"> <%=bEANCalendario.getPlataformas(i)%>
</td>
         </tr>
     <% } %>
     <%
if(!bEANCalendario.getIdCursos(i).equals(bEANCalendario.getIdCurso
s(i-1))) { %>
      <tr>
        <td>
          <div align="center">
            <table    width="100%"    border="0"    cellspacing="1"
cellpadding="1">
              <tr class="calendarioCurso">
                <td width="4%" class="calendarioCurso"> </td>
                <td                                          width="11%"
class="calendarioCurso"> <%=bEANCalendario.getIdCursos(i)%></
td>
                <td                                          width="54%"
class="calendarioCurso"> <%=bEANCalendario.getNombreCursos(i)
%></td>
                <td width="13%" class="calendarioCurso"

     align="center"> <%=bEANCalendario.getDuracion(i)%>
días</td>
                <td width="18%">
                  <table  width="100%"  border="0"  cellspacing="0"
cellpadding="0">
          <%  for(int  j=0;  j<bEANCalendario.getFechas().length;
j++) { %>
                 <%
if(bEANCalendario.getIdCursos(j).equals(bEANCalendario.getIdCursos
(i)) &&

          bEANCalendario.getArrIdSucursal(k).equals(bEANCalendario
          .getIdSucursal(j))) { %>
          <tr class="calendarioCurso">
                     <td
class="calendarioCurso"> <%=bEANCalendario.getFechas(j)%></td
>
                   </tr>
                                                          <% } } %>
                  </table>
                </td>
              </tr>
            </table>
```

```
          </div>
        </td>
       </tr>
      <% } %>
      <% b = 1; } %>
      <% } } %>
      <% } } %>
    </table>
    <table width="97%" border="0" cellspacing="0" cellpadding="0"
height="45">
      <tr>
        <td height="46"> </td>
      </tr>
    </table>
    </center>
</form>
</body>
</html>
```

4.5 Pruebas

Matriz de pruebas

# CASO DE PRUEBA	COMPONENTE (módulo, programa, etc.)	ESCENARIO	DATOS DE ENTRADA	RESULTADO ESPERADO	EJECUTOR	FECHA	APROBADO (SI O NO)	OBSERVACIONES (En caso de que la prueba falle, describir el defecto, y ponerle un número consecutivo)
1	Login del sistema (parte del backoffice)	Autenticación en el sistema de la parte de Intranet. Aplica para cualquier perfil de usuario.	Usuario y contraseña	Datos correctos.- Dejar acceder al menú principal, según el perfil. Datos incorrectos – Mandar mensaje de error, especificando el tipo de error.	CBP	02-08-2007	SI	
2	Menú del administrador	Probar el menú del administrador que solo se ejecuté una opción correcta.	Selección de opciones.	Ver las opciones correctas y mostrar las páginas correctas	CBP	02-08-2007	SI	
3	Consulta de alumnos	Buscar a un alumno ya sea por su usuario o por su nombre para mostrar sus datos e inscripciones a cursos.	Usuario del alumno y nombre parcial o completo del alumno.	Mostrar el o los alumnos que salieron de la búsqueda y posteriormente mostrar los datos completos.	CBP	02-08-2007	SI	
4	Inscripción de alumnos	Inscripción a cursos de un alumno existente en el sistema.	Usuario del alumno y nombre parcial o completo del alumno.	Mostrar el o los alumnos que salieron de la búsqueda y posteriormente seleccionar el curso a para preinscribir o inscribir.	CBP	02-08-2007	SI	

4.6 Implementación

Para garantizar las bondades de una aplicación Web construida con tecnología Java, como son la robustez, confiabilidad, estabilidad y eficiencia del sistema de automatización del proceso de preinscripción y gestión de cursos vía Web se requieren los siguientes requerimientos de hardware y software.

Software:

- JDK ver 1.4.2 o superior
- WebLogic 8.0, como servidor aplicativo
- Windows Server 2000 o superior, como sistema operativo
- Oracle 8i o superior, como base de datos

Hardware:
Para el servidor aplicativo:

- Procesador Xeon a 2 GHz o superior
- GB de memoria RAM
- Disco duro SCSI de 40 GB

Para la base de datos:

- Procesador Xeon a 3.2 GHz o superior
- GB de memoria RAM
- Disco duro SCSI de 80 GB

De manera complementaria se tiene que configurar la aplicación con los datos de los recursos, mediante el archivo de configuración **edutecsa/srv/propiedades.properties** el cual contiene lo siguiente:

```
############################################################Archi
vo Para Configuracion Del Sistema de Capacitacion
############################################################

## direccion ip en la cual se van a correr los Servlets
## p.e. 132.248.200.13
direccionIp=localhost
direccionSRV=/edu/servlet
direccionJSP=/edu/edutecsa
direccionIMG=/edu/edutecsa/images
direccionIMGNOT=/edu/edutecsa/images/noticias/
direccionError=/edu/servlet/edutecsa.srv.SRVError
direccionSRVContenido=/edu/contenidos/
direccionGrabaImagen=c:/Tomcat5.0/webapps/edu/edutecsa/images/noti
cias/
contexto=/edu

## Base de Datos
```

```
puertoDB=1521
direccionDB=jdbc:oracle:thin:@205.231.118.143
instanciaDB=UNAM3
loginDB=scott
passwordDB=tiger

## Servidor de Correo Electronico
mailHost=edutecsa.com.mx
dirEmail=info@edutecsa.com.mx
```

CONCLUSIÓN

Este trabajo presenta el panorama general de lo que es el ciclo de vida de un desarrollo de sistemas, utilizando como metodología el análisis y diseño orientado a objetos, Java como lenguaje de programación y como caso práctico el desarrollo del sistema de automatización del proceso de preinscripción y gestión de cursos vía web.

En este se muestra que el desarrollo de un sistema conlleva toda una planeación regida mediante una metodología para que así se puedan cumplir los tres objetivos principales en una empresa: tiempo, costo y calidad. Gracias a la metodología utilizada en este trabajo (análisis y diseño orientado a objetos), un desarrollo de un sistema empresarial puede salir en los tiempos estimados en el análisis, con un costo bastante aproximado al estimado y con una calidad que gracias a está se pueden prever casi la totalidad de los escenarios y así minimizar riesgos para el proyecto.

Por lo anterior se concluye que se llevó a cabo con éxito el desarrollo de la aplicación basandose en el análisis y diseño orientado a objetos, llevando a cabo el levantamiento de requerimientos con el usuario en todo el ciclo de vida del proyecto, para identificar sus necesidades y así poder realizar un análisis y diseño que cumplieran con las expectativas del usuario. Para llegar a este objetivo se utilizó como herramienta de análisis el lenguaje de modelado unificado, ya que sin este se huebiera complicado todo el proceso de análisis, también se utilizó el patrón de diseño MVC, el cual facilitó en gran medida la fase de desarrollo ya que la aplicación utiliza una arquitectura web multicapa.

Respecto al desarrollo, se procuró que fuera lo mas independiente posible, con una alta cohesión y un bajo acoplamiento, logrando con ello, la fácil adaptación e integración de nuevos componentes para que en un futuro la aplicación sea lo más flexible posible para crecerla sin tantos obstáculos. También se documentó en gran medida, tanto en documentos del proyecto como en comentarios del código para hacer comprensible su funcionalidad y así facilitar el desarrollo de futuros programadores.

Por último cabe señalar que se concluyó con el desarrollo de la primera fase del sistema y que el usuario quedó satisfecho con la funcionalidad de esta, reconociendo la necesidad de nuevas funcionalidades que integren componentes ya desarrollados (como el pago con tarjeta de crédito), para que se puedan cubrir las demás necesidades del centro de capacitación.

BIBLIOGRAFÍA

MANUALES DEL DIPLOMADO

- Sun Microsystems, Inc. OO-226, *Object-Oriented Application Análisis and Design for Java Technology (UML)*. Revision B; 2000.
- Sun Microsystems, Inc. SL110, *Fundamentals of the Java Programming Language*. Revision C; 2002.
- Sun Microsystems, Inc. SL-275, *Java Programming Language*. Revision DB; 2001.
- Sun Microsystems, Inc. SL-285, *Java Programming Language Workshop*. Revision B.2; 2000.
- Sun Microsystems, Inc. SL-314, Web Component Development with Servlet and JSP Technologies. Revision C; 2002.
- Sun Microsystems, Inc. SL-351, *Business Component Development with Enterprise JavaBeans Technology*. Revision B; 2002.

LIBROS

- Patrick Naughton, Herbert Schildt. *Java, Manual de Referencia*. 1ª Ed. Osborne/McGraw-Hill; 1997.
- Jim Keogh. J2EE, Manual de Referencia. 1ª Ed. McGraw-Hill/Interamericana de España; 2003.
- Maruyama, Tamura, Uramoto. Creación de sitios Web con XML y Java. 1ª Ed. Prentice may; 2000.
- Booch G., Rumbaugh J., Jacobson I. El Proceso Unificado de Desarrollo de Software. Addison-Wesley, Madrid, 2000.

DOCUMENTACIÓN ELECTRÓNICA (WEB)

- Ingeniería de software
  - http://es.wikipedia.org/wiki/Desarrollo_de_software
  - http://www.monografias.com/trabajos5/inso/inso.shtml
  - http://www.cs.ualberta.ca/~pfiguero/soo/metod/
  - http://www.chuidiang.com/ood/metodologia/metodologia.php

- Patrones de diseño
  - http://www.corej2eepatterns.com/Patterns2ndEd
  - http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o
  - http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

- UML
  - http://www.scribd.com/doc/395783/RUP-etapa-diseno
  - http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado
  - http://oness.sourceforge.net/docbook/oness.html#metodologia

- Documentación de programación en Java
  - http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java
  - http://www.unav.es/cti/manuales/Java/indice.html
  - http://www.javahispano.org/
  - http://www.programacion.net/java/
  - http://133.9.108.158/~nbaloian/waseda2008/JDBC/JDBC.ppt#6
  - http://www.ldc.usb.ve/~ruckhaus/materias/ci6872/clase10.pdf
  - http://www.dcc.uchile.cl/~lmateu/CC60H/Trabajos/jfernand/

REVISTAS

- Santos, Dávila. “Programación Java EE desde cero [III] Modelo Vista Controlador”. En: Sólo programadores, [No.143], Distribución en México DIMSA-C/Mariano Escobedo, 218 Col. Anáhuac. 11320 México, D.F. Revistas profesionales S.L. Agustín. Asociación Española de Editoriales de Publicaciones Periódicas. P. 30-35

- Moren, Alejandro; Romay, Pilar. “Metodologías de desarrollo [I]”. En: Sólo programadores, [No.144], Distribución en México DIMSA-C/Mariano Escobedo, 218 Col. Anáhuac. 11320 México, D.F. Revistas profesionales S.L. Agustín. Asociación Española de Editoriales de Publicaciones Periódicas. P. 46-51