14

# UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

A R A G O N

"ESTUDIO DEL LENGUAJE JAVA COMO TECNOLOGIA DE CODIGO DISTRIBUIBLE Y SU INTERACCION CON UN MANEJADOR DE BASE DE DATOS A TRAVES DE UNA APLICACION SOBRE INTERNET"



**T E S I S  P R O F E S I O N A L**

QUE PARA OBTENER EL TITULO DE:

**INGENIERO EN COMPUTACION**

P R E S E N T A :

**HORTENSIA ERICKA CERON VARGAS**

ASESOR: M. EN C. LEOBARDO HERNANDEZ AUDELO

SAN JUAN DE ARAGON, EDO. DE MEX. 2001

# "ESTUDIO DEL LENGUAJE JAVA COMO TECNOLOGÍA DE CÓDIGO DISTRIBUIBLE Y SU INTERACCIÓN CON UN MANEJADOR DE BASE DE DATOS A TRAVÉS DE UNA APLICACIÓN SOBRE INTERNET"

**Autor:** **Hortensia Ericka Cerón Vargas.**

**Asesor:** **M. en C. Leobardo Hernández Audelo.**

# INDICE


*Prólogo* ..... *4*
*Acerca de este trabajo.* ..... *6*
*Agradecimientos* ..... *7*
*Capitulo 1. Lenguaje de programación Java* ..... *8*

1.1 **Historia de *Java*** ..... 9

1.2 **Características específicas de *Java*** ..... 11
1.2.1 Simplicidad ..... 11
1.2.2 Orientado a objetos ..... 11
1.2.3 Compilado e Interpretado ..... 12
1.2.4 Multiplataforma ..... 13
1.2.5 Seguro ..... 13
*1.2.5.1 Modelos de seguridad en Java* ..... *14*

1.3 **Multibilo, (*Multithread*)** ..... 17
1.3.1 Ciclo de vida de un thread ..... 18
1.3.2 Planificador (Scheduling) ..... 19

1.4 **Características de *Java* como lenguaje** ..... 19
1.4.1 Comentarios ..... 19
1.4.2 Palabras reservadas ..... 20
1.4.3 Tipos de datos en Java ..... 20
1.4.4 Control de acceso a los datos ..... 21
1.4.5 Interfaces ..... 21
1.4.6 Métodos ..... 22
*1.4.6.1. Sobreescritura de métodos* ..... *23*
*1.4.6.2. Sobrecarga de métodos. ("Overloading Methods")* ..... *24*
1.4.7 Constructores y destructores ..... 24

1.5 **Estructuras de control** ..... 25
1.5.1 Condicional if – else ..... 25
1.5.2 Condicional switch ..... 26
1.5.3 Bucle while ..... 26
1.5.4 Bucle do/while ..... 26
1.5.5 Bucle for ..... 27

1.6 **Excepciones** ..... 27
1.6.1 ¿Qué es una excepción y cómo manejarla? ..... 27
1.6.2 Ventajas del uso del manejo de excepciones ..... 28
1.6.3 Lanzamiento de excepciones ..... 28
1.6.4 Captura de excepciones ..... 28

1.7 **Aplicaciones Standalone y *Applets*** ..... 31
1.7.1 Aplicaciones Standalone ..... 31
1.7.2 Applets ..... 31
1.7.3 Ciclo de vida de un applet ..... 32
*Capitulo 2. Software y Sistema* ..... *33*

2.1 **Herramientas de desarrollo *Java* (*JDK*)** ..... 33

2.2 **Características del *JDK* 1.1** ..... 34

2.3 **Instalación del *JDK*** ..... 36
2.3.1 Instalacón del JDK1.1.5 ..... 36
2.3.2 Herramientas del JDK ..... 38

**2.4 Paquetes.** ..... 39
2.4.1 El paquete java.net ..... 39
*2.4.1.1. Clase Socket* ..... *40*
*2.4.1.2. Clase Server Socket.* ..... *41*
2.4.2 El paquete java.awt ..... 41
*2.4.2.1. Componentes.* ..... *42*
*2.4.2.2. Contenedores.* ..... *43*
2.4.3 El paquete java.io ..... 44
2.4.4 El paquete java.util ..... 45
2.4.5 El paquete java.lang ..... 46
***Capitulo 3. Arquitectura cliente-servidor y bases de datos.*** ..... ***48***
**3.1 Definición de aplicación cliente-servidor** ..... 49
**3.2 Conceptos generales de bases de datos** ..... 49
3.2.1 Modelos de bases de datos ..... 51
*3.2.1.1. Modelo relacional* ..... *51*
*3.2.1.2. Modelo jerárquico* ..... *52*
*3.2.1.3. Modelo de red* ..... *53*
3.2.2 Estructura de la base de datos ..... 53
*3.2.2.1. Tablas* ..... *54*
*3.2.2.2. Llaves primarias* ..... *54*
*3.2.2.3. Llaves foráneas* ..... *54*
*3.2.2.4. Índices* ..... *55*
*3.2.2.5. Tipos de columnas* ..... *55*
*3.2.2.6. Procedimientos almacenados* ..... *55*
*3.2.2.7. Disparadores.* ..... *56*
*3.2.2.8. Restricciones* ..... *56*
*3.2.2.9. Catálogo del sistema.* ..... *56*
3.2.3 Comandos SQL – DDL ..... 56
*3.2.3.1. CREATE TABLE* ..... *56*
*3.2.3.2. ALTER TABLE* ..... *57*
*3.2.3.3. DROP TABLE* ..... *57*
*3.2.3.4. CREATE INDEX* ..... *57*
*3.2.3.5. DROP INDEX* ..... *58*
3.2.4 Comandos SQL – DML ..... 58
*3.2.4.1. INSERT* ..... *58*
*3.2.4.2. SELECT* ..... *58*
*3.2.4.3. UPDATE* ..... *59*
*3.2.4.4. DELETE* ..... *59*
**3.3 Manejador de base de datos *Oracle*.** ..... 60
**3.4 El paquete *java.sql*** ..... 60
**3.5 Herramientas de *Java* para el manejo de bases de datos ( *JDBC*)** ..... 61
**3.6 La interfaz *Driver*** ..... 61
***3.7* *Driver* Manager** ..... 62
**3.8 Como hacer una conexión a la base de datos** ..... 63
**3.9 Consultas a la base de datos** ..... 64
**3.10 Tipos de controladores *JDBC* (*driver JDBC*)** ..... 64
3.10.1 Tipo 1. Puente entre JDBC y ODBC (JDBC – ODBC bridge driver) ..... 64
3.10.2 Tipo 2. Acceso en Java las bibliotecas nativas. (Native-API/partly-Java driver) ..... 64
3.10.3 Tipo 3. Controlador todo Java. (Net-protocol/all-Java driver) ..... 64
3.10.4 Tipo 4 Protocolo nativo ..... 65

*3.11 Driver* JBDC de *Oracle* ..... 66
***Capitulo 4. Desarrollo de la aplicación*** ..... ***67***

**4.1 Presentación de la aplicación** ..... 67

**4.2 Metodología *RAD*.** ..... 68
4.2.1 Conceptualización ..... 69
4.2.2 Prototipo (o versión beta) ..... 70
4.2.3 Versión Final ..... 70

***4.3 UML*** ..... 71

**4.4 Diagramas estructurales** ..... 72
4.4.1 Diagramas de clases ..... 72
*4.3.1.1. Administración / Alumnos* ..... *73*
*4.3.1.2. Administración / Profesores* ..... *74*
*4.3.1.3. Administración / Materias* ..... *75*
*4.3.1.4. Administración / Planteles* ..... *76*
*4.3.1.5. Administración / Grupos* ..... *77*
*4.3.1.6. Administración / Carreras* ..... *78*
*4.3.1.7. Alumno* ..... *79*
*4.3.1.8. Profesor* ..... *79*
*4.3.1.9. Estructura de la base de datos* ..... *81*
4.4.2 Diagramas de componentes ..... 82
*4.3.2.1. Paquetes* ..... *82*
*4.3.2.2. Paquete principal* ..... *83*
*4.3.2.3. Paquete administrador* ..... *83*
*4.3.2.4. Paquete cliente* ..... *85*
*4.3.2.5. Paquete utilerías* ..... *86*
*4.3.2.6. Paquete alumno* ..... *86*
*4.3.2.7. Paquete profesor* ..... *87*
*4.3.2.8. Paquete fija* ..... *89*
4.4.3 Diagramas de despliegue ..... 90
*4.4.3.1 Estructura cliente-servidor* ..... *90*

**4.5 Diagramas de comportamiento** ..... 91
4.5.1 Diagramas de casos de uso ..... 91
*4.4.1.1. Estructura plantel* ..... *92*
*4.4.1.2. Estructura Facultad* ..... *93*
*4.4.1.3. Estructura Carrera* ..... *94*
*4.4.1.4. Estructura Materia* ..... *95*
*4.4.1.5. Estructura Grupo* ..... *96*
*4.4.1.6. Estructura Asignatura* ..... *97*
*4.4.1.7. Estructura Alumno* ..... *98*
*4.4.1.8. Estructura Profesor* ..... *99*
*4.4.1.9. Proceso Examen* ..... *100*
4.5.2 Diagramas de colaboración ..... 101
***Capitulo 5. Análisis de resultados*** ..... ***102***

**5.1 Presentación del Sistema de Evaluación** ..... 102
5.1.1 Módulo del Administrador ..... 104
5.1.2 Módulo del Profesor ..... 125
5.1.3 Módulo del Alumno ..... 133

**5.2 Pruebas del Sistema** ..... 135

**5.3 Conclusiones de las pruebas del Sistema** ..... 136
***Capitulo 6. Conclusiones*** ..... ***138***
***Referencias*** ..... ***140***

# Prólogo

Hoy en día las necesidades humanas y la tecnología ya no son las mismas que hace diez años, éstas han cambiado y por consiguiente los sistemas de información también; ahora la información ha tomado gran importancia y los desarrolladores de sistemas se han visto en la necesidad de desarrollar sistemas que permitan hacer un buen manejo de la información, además de utilizar bases de datos para el almacenamiento de la misma. Debido a estos cambios, los especialistas en ingeniería de software han creado nuevos lenguajes de programación que facilitan el trabajo que enfrentan los programadores, uno de estos nuevos lenguajes de programación es *Java*, él cual ahora, permite desarrollar de manera sencilla aplicaciones para la red de redes *Internet*, gracias a que es un lenguaje multiplataforma e ideal para aplicaciones de este tipo; éste lenguaje permite realizar la programación del sistema una sola vez y ejecutarlo en diferentes plataformas, debido a que es un lenguaje compilado e interpretado, el cual al compilarse genera *código de bytes* que a su vez es interpretado por la máquina virtual del *browser*, también conocido como *navegador*, de la máquina cliente.

En una época como la actual, donde escuchamos frecuentemente frases como *"la era de la información"* y *"la ubicuidad de los sistemas"*, se necesita el apoyo de sistemas interactivos útiles porque ahorran tiempo a los usuarios, ya que no deben estar presente físicamente ni de manera concurrente en un solo lugar para desarrollar ciertas funciones inherentes a su actividad. Es por está razón por la cual se decidió conjuntar en el presente trabajo de tesis el estudio del lenguaje *Java*, su interacción con una base de datos y además que fuera mediante una aplicación sobre *Internet*.

El desarrollo de está aplicación tiene la finalidad de conocer de forma real algunos de los problemas más comunes con los que se enfrentan los desarrolladores durante la realización de sistemas. La aplicación denominada *"Sistema de Evaluación"*, se analiza y define para un sistema que sirve de apoyo en las actividades de profesores y alumnos en la elaboración de exámenes, esto con el fin de demostrar la utilidad de la aplicación en un esquema cliente servidor sobre *Internet*. Esta aplicación es solo un prototipo para comprender mejor el estudio del lenguaje *Java* y su interacción con la base de datos de forma real.

El sistema realizado en el presente trabajo, no está conceptualizado para sustituir al profesor, sino más bien, para apoyarlo en sus actividades académicas al proporcionarle la facilidad de introducir sus exámenes con sus respectivas respuestas y evaluar a sus alumnos de un grupo específico.

Para lograr la realización de este sistema se hace un análisis, durante el cual hubo la necesidad de delimitar el alcance de la aplicación debido a que los requerimientos que en un principio se contemplaban iban mas allá del propósito de esta tesis pero esto implicaba un mayor número de recursos tanto materiales como humanos y económicos, lo cual se reflejaría consecuentemente en el consumo de una mayor cantidad de tiempo.

Es importante mencionar que la aplicación realizada en este trabajo de tesis es solo una parte de lo que pudiera ser un sistema más grande. La elaboración del sistema se basó en las necesidades básicas de los requerimientos de un *Sistema de Evaluación* capaz de administrar su información, con el fin único de hacer notorio el estudio de el lenguaje *Java* y su interacción con una base de datos. El sistema en general hace el manejo de las entidades mínimas necesarias para un *Sistema de Evaluación* como los son el manejo de la información de alumnos, profesores, materias, exámenes, planteles, carreras, etc., para la elaboración de un examen que pueda ser generado por un profesor de forma remota y que los alumnos puedan contestarlo y saber su calificación instantáneamente también remotamente.

Sin embargo, se deja pié para un posterior uso del código fuente y que se amplíe la funcionalidad de la aplicación, con lo cual se deja de manifiesto la reutilización de código ya que se pueden extender las clases utilizadas, como una característica propia de *Java*. También para que se pueda elaborar un sistema que ofrezca además un valor agregado como pudieran ser desde trabajos administrativos de impresión de tiras de materias, impresión y consulta de lista de grupos, inscripción de alumnos hasta la explotación de información estadística de tendencias de aprovechamiento, educación a distancia o quizás pueda servir como punto de partida para la realización de un sistema inteligente aprovechando así los paradigmas de la programación orientada a objetos.

Es importante mencionar que el presente trabajo no pretende ser un estudio formal del lenguaje *Java*, de las bases de datos, de *Internet*, *UML*, arquitectura cliente-servidor, etc, pues aunque estas tecnologías son muy prometedoras su potencial no se aprovechará plenamente si no se posee la habilidad ni los conocimientos necesarios para integrarlas en la realización de una aplicación real.

# Acerca de este trabajo.

Este trabajo de tesis realiza un estudio del lenguaje *Java* y su interacción con una base de datos a través de la realización de una aplicación sobre *Internet*; además muestra de forma general los conceptos y problemas durante la realización de dicha aplicación.

El capítulo I, <<*Lenguaje de programación Java*>> trata un poco de historia acerca del origen del lenguaje, características básicas, estructuras básicas de control, manejo de excepciones, aplicaciones *standalone* y *applets*.

En el capítulo II, <<*Software y Sistema*>> se habla acerca del *JDK ("Java Developer Kit")*, el cuál es un conjunto herramientas para el desarrollo de aplicaciones, sus características y el proceso de instalación. Se estudian también los diferentes paquetes propios del lenguaje utilizados en el desarrollo de la aplicación.

El capítulo III, <<*Arquitectura cliente-servidor y bases de datos*>> trata acerca de la arquitectura cliente-servidor, algunos conceptos básicos de base de datos relacionales, el paquete *java.sql* y de los drivers utilizados para realizar la conexión entre el lenguaje de programación y la base de datos.

El capítulo IV, <<*Desarrollo de la aplicación*>> se ocupa del desarrollo de la aplicación y de la metodología empleada para su análisis. Se detalla la manera en que se hace un buen análisis de requerimientos empleando una metodología de gran aceptación en el ámbito de la programación por su efectividad, conocida como metodología *RAD*[1]. Además se hace uso del lenguaje *UML*[2] propio para el modelado, desarrollo y documentación de sistemas orientados a objetos.

El capítulo V, <<*Análisis de resultados*>> detalla la aplicación realizada en este trabajo. En él se muestra ordenadamente una impresión de las pantallas tal cómo se haría una navegación por la aplicación y se da una breve explicación de lo que ocurre en cada una de ellas. El código que corresponde a cada uno de los programas de la aplicación se encuentra en un medio magnético anexo a este trabajo.

El capítulo VI, <<*Conclusiones*>> contiene las conclusiones obtenidas del presente trabajo.

Finalmente las *referencias*, contiene la bibliografía de los libros y direcciones utilizadas para la realización del presente trabajo.

[1] *Rapid Application Development*
[2] *Unified Modeling Language*

# Agradecimientos

Está tesis está dedicada a:

A mis padres Hortensia y Joaquín por sus enseñanzas y cariño a lo largo de mi vida, las cuales me han permitido llegar hasta donde estoy.

A mis hermanas Rocío y Josefina, por su constante aliento y apoyo

A mi amadísimo esposo Andres por su apoyo, comprensión y tiempo gastado durante la realización de este trabajo de tesis.

A Helios por su apoyo para la realización de este trabajo.

A mi asesor de tesis Leobardo por sus enseñanzas y paciencia para las revisiones.

*Lo esencial para ser feliz en ésta vida*
*es tener algo que hacer,*
*alguien a quién amar*
*y algo que esperar.*

***-- Jeremy Taylor.***

# Capitulo 1. Lenguaje de programación *Java*.

Este capítulo trata acerca del origen del lenguaje *Java* y sus principales características, las cuales hacen de él un lenguaje ideal para el desarrollo de aplicaciones sobre la red de redes *Internet*.

*Java* es un lenguaje fácil de aprender debido a que es simple, orientado a objetos y se encuentra basado en el lenguaje de programación C++, por lo que si el programador ha utilizado ya lenguaje C++ o cualquier otro lenguaje orientado a objetos *Java* le parecerá muy familiar.

*Java* además de ser orientado a objetos, es un lenguaje compilado e interpretado que nos permite desarrollar aplicaciones multiplataformas. Esto quiere decir que el código sólo se programa una vez y se ejecuta en diferentes plataformas, llámese *WindowsNT*, *UNIX*, etc. basta con tener un *navegador* o *browser* (*Netscape*, *Explorer, etc.*) en la máquina cliente, que es en donde se va a ejecutar la aplicación. Esta característica es de gran utilidad para los desarrolladores de sistemas quienes pueden reutilizar la aplicación sin necesidad de modificarla para cada plataforma o simplemente volver a utilizar fracciones de código en otras aplicaciones ahorrando tiempo y dinero.

Otra de las características útiles del lenguaje *Java* a tratar dentro de este capítulo es la concurrencia la cual se explica en el tema de los hilos de ejecución [3]. Además las características propias del lenguaje tales como los tipos de datos, comentarios, palabras reservadas, métodos, interfaces, constructores, destructores, etc. Los diferentes tipos de acceso a los métodos y variables, los cuales pueden ser declarados como *públicos*, *privados*, *protegidos o amigables*[4]. Aquí se explica cuando un método o una variable puede ser accesada sólo desde el programa donde ha sido declarado y cuándo desde otros programas, es decir el alcance "scope" de los métodos y las variables.

También se menciona la forma en la cual *Java* hace el manejo de errores, evitando que el programa se vea afectado en tiempo de ejecución cuando ocurren situaciones anómalas.

Finalmente se habla de las dos modalidades para desarrollar aplicaciones en *Java* las cuales son: *Aplicaciones Standalone y Applets*. Las primeras son totalmente independientes de un *navegador* y las segundas se integran dentro de una página *HTML ("HyperText Markup Language")* para ser visualizadas a través de el.

*3 Multihilo también es conocido como multithread.*
*4 Comúnmente conocidos como friendly*

## 1.1 Historia de *Java.*

El lenguaje de programación *Java* remota su origen hacía el año de 1991, cuando un grupo de investigadores que trabajaban para *"Sun Microsystems"*[5] realizaba un proyecto de desarrollo de software para productos electrónicos de consumo [2 (pp. 52)], llamado Proyecto Verde *("Green Project")*, en el cual se requería que los productos fueran sencillos y fiables. Para ello se necesitaba una forma de crear código que fuera independiente de la plataforma y que permitiera ejecutar el software en cualquier *CPU* (Unidad Central de Procesamiento). Al principio pensaron en utilizar C++. Sin embargo no tenía las características adecuadas al igual que ninguno de los lenguajes de esa época, por lo cual decidieron crear un lenguaje nuevo llamado *Oak*[6], el cual estaba basado en C++. Este lenguaje eliminaba algunas deficiencias de C++, tales como la herencia múltiple, la conversión automática de tipos, el uso de apuntadores, la administración de la memoria, además de incluir un recolector de basura, ser orientado a objetos, incluir características *multihilo*, soporte nativo de red, seguridad en red, adaptación de los caracteres al formato *UNICODE*, que a diferencia del *ASCII* permite la representación de gran cantidad de lenguas y alfabetos, un API *("Aplication Program Interface")* estructurada en una jerarquía de clases estándar y herramientas para generar documentación automáticamente.

El lenguaje *Oak* corrió su primer programa en Agosto de 1991 [3 (pp.6)] y fue utilizado en la creación del dispositivo electrónico llamado *Estrella 7* (start 7)[7]. Sin embargo no tuvo gran auge debido a los altos costos que representaba la manufactura de los chips.

Poco tiempo después y porque ya existía un lenguaje con el nombre de *Oak* fue renombrado con el nombre de *Java*[8], en honor a todas las tazas de café que se habían tomado discutiendo el proyecto. En ese entonces Bill Joy, uno de los cofundadores de *Sun*, tuvo la gran idea de liberar la Tecnología *Java* para que estuviera disponible gratuitamente para uso no comercial a través de *Internet* y se convirtiera en un estándar. Sin embargo *Sun* no solo quería que fuera un estándar; también deseaba que el producto fuera negocio y decidió vender licencias comerciales.

En septiembre de 1994, Naughton y Jonathan Payne al ver que la *Web* tenía grandes posibilidades decidieron utilizar *Java* como base para crear un *navegador* y fue aquí donde surgió el *WebRunner*, que más tarde se mejoraría y llamaría *HotJava*, el cual a principios de 1995 junto con *Java*, su documentación y el código fuente ya se encontraban disponibles en la red en su versión *alfa*, la cual era parte del *JDK* *("Java Developer's Kit")*. Es importante mencionar que inicialmente *Java* se presentó para *SPARC Solaris* y poco tiempo después para *Windows NT*[9], *Windows 95* y *LINUX*.

---

*5 Empresa que proporciona sistemas, software, almacenamiento, soporte y servicios. Además de la tecnología Java.*
*6 Este nombre fue tomado de un roble que había frente al despacho de Sheridan, uno de los integrantes del "Proyecto Verde".*
*7 ¿?*
*8 En el argot norteamericano significa café.*
*9 Sistema Operativo de Microsoft.*

En el otoño de este mismo año se publicó la versión *Beta 1* de *Java*, y *Netscape Incorporation* anunció que incorporaría soporte para *Java* en su *navegador Netscape 2.0*. Esta versión despertó una gran inquietud entre las empresas, por lo cual se vio la necesidad de que *Java* fuera portable para los principales sistemas operativos y las diferentes arquitecturas.

En Diciembre de 1995 se dio a conocer la versión *Beta 2* de *Java* y tanto *Sun* como *Netscape* anunciaron la próxima aparición de *JavaScript*[10].

A principios de este mismo mes *Microsoft* e *IBM*[11] se vieron interesados en obtener una licencia para hacer uso de la tecnología, lo que marcó un gran éxito para *Java*. Adicionalmente, *Microsoft* acordó dar a *Sun* la implementación de la *"Interfaz para Programación de Aplicaciones"* (*API*) para *Windows* y soporte a *Java* en el *Internet Explorer* versión 3.0. Aunque para 1996 Sun Microsystems se vio obligada a pleitear con *Microsoft* por incumplimiento del contrato, al incluir en sus productos tecnología *Java* sólo para la plataforma Win32, a lo que Microsoft reaccionó y dijo que no daría más soporte a *Java*.

El 23 de enero de 1996 se publicó la versión oficial *Java 1.0* y *JavaScript*. Y durante los años siguientes está versión de *Java* fue evolucionando, pasando así por las versiones *JDK 1.1* hasta la versión actual que es la *JDK 2.1*. mejorando en cada versión al lenguaje.

Hoy en día y debido a la acelerada expansión de *Internet*, *Java* es sin duda el lenguaje más prometedor que existe, por lo que todos los líderes de sistemas operativos y plataformas de *navegador*es han construido una Máquina Virtual dentro de sus sistemas. Un ejemplo es *Netscape* que da soporte tanto a *Java* como a *JavaScript* desde su versión *3.0*. Otro es *Oracle*[12] que ha implementado la Máquina Virtual *Java* en sus herramientas para desarrollo de sistemas y aplicaciones desarrolladas por él mismo.

*10 Lenguaje de Scripts que solo se utiliza para darle funcionalidad a las páginas HTML.*
*11 IBM es una empresa que se dedica a la fabricación de equipo de computo.*
*12 Empresa lider en el desarrollo de bases de datos y herramientas para la elaboración de sistemas, así como presentación de soluciones integrales para el manejo de datos.*

## 1.2 Características específicas de *Java*.

Durante los últimos años *Internet* ha tenido un gran auge gracias al crecimiento de las redes, lo que ha hecho necesario desarrollar aplicaciones en línea a través de la red de redes, *Internet*.

Actualmente la tecnología *Java* es ideal para desarrollar este tipo de aplicaciones porque cuenta con algunas características, muy propias de este lenguaje, las cuales se explican a continuación [1].

### *1.2.1 Simplicidad.*

*Java* es un lenguaje simple [4], porque prácticamente sin saber mucho de programación , se aprende ha hacer *Applet*s y aplicaciones fácilmente, permitiendo mejorar nuestra programación en un tiempo reducido.

Su simplicidad se debe principalmente a que es un lenguaje basado en C++. Por lo tanto si el programador ya conoce C++ será más fácil aprender *Java*, ya que su sintaxis es muy parecida.

Además *Java* tiene la capacidad de manejar la memoria automáticamente; para ello se auxilia del recolector de basura *("garbage collection")*, el cual se encarga de liberar los segmentos de memoria no utilizada. Esto es de gran utilidad porque el programador ya no tiene que preocuparse explícitamente de ello, en su vez, *Java* lo hace automáticamente por él. Además obliga a que todos los objetos se pasen por referencia, evitando de esta manera que se cometan errores por la mala utilización de la aritmética de apuntadores.

El lenguaje *Java* también elimina la herencia múltiple, la cual es difícil de utilizar; pero permite realizar funciones semejantes a través de las interfaces, las cuales se verán a lo largo de este capítulo.

Otra característica que lo hace simple es que la conversión de datos sólo se puede hacer de manera explícita, asegurando así que no haya perdida de información.

### *1.2.2 Orientado a objetos.*

Otra de las características importantes del lenguaje *Java* es que es un lenguaje orientado a objetos [2], el cual enfoca el diseño del sistema en los objetos y las interfaces hacia ellos. Un objeto es una identidad capaz de tener un estado definido por los *datos* y una conducta representada por *operaciones y métodos* para afectar su estado. Estos pueden representar cosas reales como una casa, un árbol, etc. o representar conceptos abstractos tales como el tiempo o números por mencionar algunos. Cada objeto dentro de un programa tiene sus propios atributos y comportamiento. La función de los objetos es interactuar entre ellos para ejecutar el programa. *Java* también involucra el concepto de clases para identificar al conjunto de objetos con características similares, con el fin de especificar atributos y comportamientos comunes para objetos del mismo tipo. Por ejemplo, la clase

humano se compone de 2 objetos: hombre y mujer, cada uno con sus propios atributos, que pueden ser nombre, edad, color de piel, etcétera, solo por mencionar algunos.

El lenguaje *Java* soporta las tres características propias del paradigma de la orientación a objetos: Encapsulación, herencia y polimorfismo [2]. Las cuales se explican en la tabla T.1.1.:

| PROPIEDAD | DESCRIPCION |
|---|---|
| Encapsulamiento | Es también conocido como el ocultamiento de la información. Se utiliza con el fin de ocultar los detalles internos de una clase para el resto del código. Esto nos permite realizar modificaciones en la implementación de un objeto sin afectar las aplicaciones que la utilizan |
| Herencia | Es la propiedad que nos permite crear una nueva clase en términos de una clase existente. La clase derivada hereda los datos y las funciones miembro de la clase base, y puede redefinir algunas funciones miembro o definir otras nuevas, para modificar la funcionalidad que ha recibido de la clase base. Mediante esta propiedad se puede volver a utilizar el código existente. |
| Polimorfismo | Es el mecanismo mediante el cual diferentes objetos pueden responder al mismo mensaje de diferente forma. Por ejemplo si enviamos la orden de movimiento a diferentes objetos como un pez, un águila y un perro. Cada uno responderá a la orden de movimiento de diferente forma. El pez se mueve nadando, el águila volando y el perro en la suya propia. [5] |

Tabla T.1.1. Características de la programación orientada a objetos.

## *1.2.3 Compilado e Interpretado.*

*Java* no es un lenguaje únicamente compilado como tradicionalmente se estilaba, sino un lenguaje compilado e interpretado [10]. Los programas (código fuente *Java*) se compilan para generar los códigos de bytes, *(bytecodes)*; estos códigos son instrucciones y datos relacionados a una computadora hipotética llamada **Máquina Virtual *Java ("Java Virtual Machine")*** para poder ejecutar la aplicación; estas instrucciones parecen código máquina, pero no son específicas para ningún procesador. Esto significa que las diferencias que existen entre las plataformas de las diferentes computadoras como la longitud de instrucciones puede ser reconocida y acomodada localmente sin necesidad de realizar diferentes versiones del programa ni recompilar. El intérprete de *Java* en cada sistema operativo realiza una revisión de cada objeto con el fin de asegurar su integridad. La Fig. 1.1. muestra como el código fuente *Java* guardado en un archivo con extensión *.java*, pasa por el compilador al ejecutar la instrucción "*javac* `archivo.java`"[13] generando el **código de bytes** (archivo con extensión *.class*) siempre y cuando el programa no genere ningún error de compilación. Este código de bytes es interpretado por la Máquina Virtual *Java*. la cual está incluida dentro del *navegador* de *Internet*.

*13 javac es el comando para compilar en Java y archivo.java . es el nombre del archivo con extensión .java*

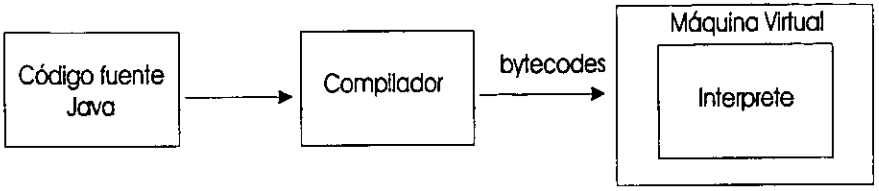



Fig. 1.1.Proceso de compilación *Java*.

## *1.2.4 Multiplataforma.*

La independencia de plataforma es una de las principales características del lenguaje *Java*. Esto se refiere a que un programa realizado en *Java* es capaz de ejecutarse fácilmente en un sistema operativo o en otro, sin necesidad de re-escribirlo, modificarlo o recompilarlo.

Esta característica es posible gracias que es un lenguaje compilado e interpretado como se mencionó en el punto anterior, capaz de generar códigos de bytes (*"bytecodes"*) independientes de cada plataforma y arquitectura, para posteriormente ejecutar la aplicación desde cualquier plataforma siempre y cuando se encuentre disponible él interprete de *Java*.

Por lo tanto, se puede decir que lo único dependiente del sistema operativo de cada computadora es la Máquina Virtual y las librerías fundamentales, que permiten acceder directamente al hardware de la máquina.

El contar con un lenguaje multiplataforma permite ahorrar una gran cantidad de trabajo y dinero a las empresas que se dedican a desarrollar aplicaciones y requieren trabajar en diferentes plataformas. Al igual que proporciona una gran ventaja sobre otros lenguajes y probablemente algún día dé la oportunidad a los usuarios de tener aplicaciones para cualquier sistema operativo y cualquier tipo de equipo de cómputo sin tener la necesidad de adquirir cada que haya una aplicación nueva un equipo más potente y un sistema operativo específico, para esa aplicación.

## *1.2.5 Seguro.*

La seguridad en el lenguaje *Java* representa un factor muy importante, debido a que la mayoría de las aplicaciones realizadas son enfocadas a *Internet* y se incorporan dinámicamente o ejecutan código desde alguna máquina remota, por lo que si no se toma en cuenta se podría enviar desde cualquier página *Web* un *applet* que se ejecute en la computadora local sin conocimiento del usuario haciendo posible cualquier tipo de ataque, por ejemplo podrían cifrar el contenido del disco duro o quizá introducir algún virus o simplemente permitir a cualquier intruso invadir el sistema. *Java* elimina estos riesgos, al no permitir acceso a zonas de memoria ni del sistema. Además de implementar un sistema de autenticación por **clave pública** para evitar que los intrusos de la red invadan al sistema y permitir a los *applets* utilizar libremente los recursos del mismo.

Este método de autentificación garantiza la seguridad desde el momento en el que el código de bytes, *bytecode* es interpretado, ya que antes de realizar cualquier instancia de cualquier objeto, el "*Cargador de Clases*"[14] se encarga de verificar la ***firma digital (firma de applets)*** y proporcionar los privilegios adecuados, evitando así la creación y almacenamiento de objetos en la memoria sin la validación correspondiente a los privilegios de acceso.

De este modo *Java* garantiza la seguridad ya que sin la firma digital no se permite abrir archivos de la máquina local, ni ejecutar aplicaciones nativas de alguna otra plataforma, ni utilizar la computadora local como puente para realizar operaciones con otra. Por lo cual podemos decir que es un lenguaje seguro. Sin embargo tiene un pequeño inconveniente para los programadores. Este se refiere a que el *JDK* cuenta con un *desemsamblador* de *ByteCode* (*javap*), mediante el cual cualquier persona puede obtener el programa desmontado, mostrando el algoritmo del programador. Ante este problema los programadores incluyen llamadas a métodos nativos en algún otro lenguaje que no sea *descompilable*. Lo que hace que la aplicación pierda *portabilidad*.

### 1.2.5.1 Modelos de seguridad en *Java*.

El modelo de seguridad en *Java* ha ido cambiando en cada versión. En la versión *JDK 1.0* el modelo de seguridad que maneja *Java* se denomina *Caja de Arena* [31], mediante el cual el código que no fuera confiable, por ejemplo el código remoto se ejecuta sólo dentro de la Caja de Arena teniendo acceso sólo a los recursos que se encuentran dentro de esa caja sin acceder a los recursos del sistema. Además de contar con la clase *Security Manager*, encargada de determinar que accesos y a que recursos se permiten o no se permiten como se muestra en la figura 1.2.

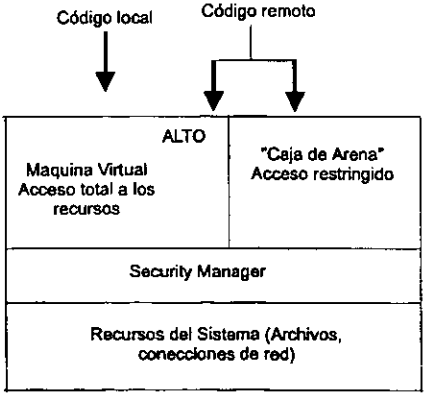



Fig. 1.2. Modelo de seguridad *Java* de la versión *JDK* 1.0.

*14 Véase el siguiente punto que habla acerca del modelo de seguridad de Java.*

La clase "*Security Manager*" proporciona los métodos mostrados en la tabla T.1.2:

| METODO | DESCRIPCION |
|---|---|
| CheckAccept(String, int) | Se llama antes de aceptar una conexión con un socket de la máquina especificada en el puerto especificado |
| CheckConnect(String, int) | Se llama antes de abrir una conexión con un socket a la máquina especificada en el puerto especificado. |
| CheckConnect(String, int, Object) | Se llama antes de abrir una conexión con un socket a la máquina especificada en el puerto especificado para el contexto de seguridad actual. |
| CheckListen(int) | Se llama para determinar si el proceso en curso tiene permiso para quedarse a la escucha de conexiones en el puerto especificado. |
| CheckSetFactory() | Determina si el proceso en curso tiene permiso para establecer la factoría de manejadores de socket o streams de URL |

T.1.2. Métodos de la clase *Security Manager*.

En la versión *JDK* 1.1. se maneja la firma de *applets* [31] (en el siguiente punto se explica como firmar un *applet*), en donde el *applet* firmado se considera como si fuera código local con acceso completo a los recursos del sistema. Mientras que aquellos que no estén firmados sólo tienen acceso a los recursos dentro de la caja de arena, como muestra la figura 1.3.

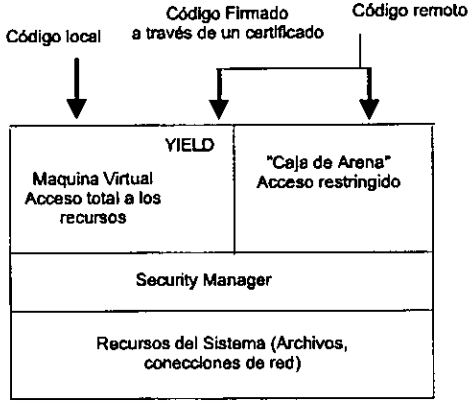



Fig. 1.3. Modelo de seguridad *Java* de la versión *JDK* 1.1.

En la versión *JDK* 1.2. se maneja todo el código remoto y local con políticas de seguridad para definir los permisos disponibles para el código [31]. El entorno de ejecución de *Java* organiza el código en dominios individuales como se ve en la figura 1.4., cada uno de ellos tiene un grupo de clases con el mismo conjunto de permisos. Los dominios pueden ser configurados por el usuario o por el administrador de tal forma que se haga el acceso restringido o con acceso completo a los recursos del sistema.

El *JDK* 1.2. también cuenta con una herramienta llamada *keytool* para el manejo de claves y certificados utilizados en la firma digital de las aplicaciones y *applets*.

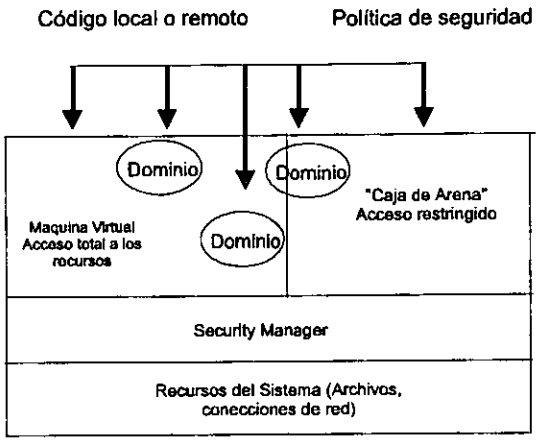



Fig.1.4. Modelo de seguridad *Java* de la versión *JDK* 1.2.

**Firma de *Applets*.**

Si se requiere que un *applet* haga uso de los recursos del sistema de la máquina local (en donde se ejecuta el *applet*) es necesario implementar un mecanismo que les permita saber de dónde y de quién es el *applet* al que se le va a dar acceso al sistema. La firma del *applet* proporciona este mecanismo de seguridad.

Para firmar un *applet* se utilizan una serie de herramientas y pasos que varían de acuerdo al tipo de *navegador* con que se cuente [11]. Por ejemplo para firmar un *applet* en *Internet Explorer* es necesario tener el *SDK ("Microsoft Software Development Kit")* y para el *Netscape Communicator* la herramienta *SignTool 1.1.* por mencionar alguna.

Por lo cual si el *applet* fue firmado para *Netscape Communicator* no se visualizará en *Internet Explorer* ni viceversa. Esto nos hace perder la portabilidad de los *applets*, ya que condiciona su visualización al tipo de *navegador* y firma que este tenga. Sin embargo en estos casos es necesario evaluar la importancia entre la seguridad y la *portabilidad.* Aunque como se acaba de mencionar los pasos para la firma del *applet* varían. Existe una serie de pasos comunes a realizar en el firmado de los *applets*:

1) **Crear el certificado.-** Obtener una pareja de claves, pública y privada extendida por alguna autoridad de Certificación. Por ejemplo en *Internet Explorer Microsoft* necesita un certificado adecuado, mientras que *Netscape* utiliza otro certificado diferente. La clave privada se encarga de firmar el código que se escriba, mientras la clave pública es utilizada por las personas que consultan el *applet* para comprobar la legitimidad del mismo.

2) **Empaquetar los archivos con las clases.**- Una vez que el código de *Java* ha sido compilado sin errores y creado una serie de archivos con extensión "*.class*". Los archivos de clase se deben compactar y comprimir en un solo archivo para ser firmado.

3) **Firma del archivo.**- El archivo que contiene todos los archivos de clase deberá ser firmado con la herramienta adecuada según el *navegador*. Para que finalmente el *applet* pueda ser incrustado dentro de la página *HTML* para su visualización.

## 1.3 Multihilo, (*Multithread*).

*Java* es un lenguaje con la característica multihilo[15] también conocida como *multithread* [12] que permite que un programa tenga más de un hilo de ejecución. En otras palabras se refiere a que dos o más tareas se ejecutan "aparentemente" al mismo tiempo dentro de un mismo programa sin necesidad de que estén entrelazadas unas con otras, y se dice aparentemente porque por lo regular solo se cuenta con un CPU, el cual, es compartido por los procesos.

Es importante mencionar que dentro del tema de los threads existen programas de un solo hilo llamados programas de flujo único ("*single thread*") y programas de flujo múltiple de varios hilos de ejecución ("*multithread*"). Los primeros son aquellos que sólo utilizan un solo flujo de control ("*thread*") para controlar su ejecución. Por ejemplo, en el código siguiente el programa solo imprime el mensaje y termina al ser llamada la función *main()*, durante un único thread:

```
public class Hola{
static public void main( String args[]){
        System.out.println("Este programa solo imprime hola");
}
}
```

Mientras que en los programas de flujo multiple ("*multithread*"), existen varios hilos de ejecución para que cada uno de ellos realice una tarea específica. Por ejemplo en un *applet* en el cual existen imágenes, un *thread* puede estar controlando el movimiento de la imagen, mientas que otro *thread* puede mostrar el texto. Esto permite presentar una mejor respuesta en tiempo real.

Los procesos pueden compartir información y tiempo de ejecución debido a la sincronización.

*15 Multihilo es diferente de multitarea. Multitarea quiere decir que dos programas se ejecutan aparentemente a la vez bajo el control del Sistema Operativo. Y multihilo quiere decir que dos o más tareas se ejecutan aparentemente a la vez dentro del mismo programa. [10]*

## *1.3.1 Ciclo de vida de un thread.*

En la Fig.1.5. [10] se ve como un *thread* cuando es creado (*new thread()* ) puede tomar dos caminos, uno seguir detenido sin ejecutarse mediante el método *stop*() o ejecutarse e inicializarse mediante el método *start*(). Si se decide por *inicializarse*, el método *start*() se encarga de crear los recursos necesarios del sistema para su ejecución e incorporarlo a la lista de procesos disponibles para la ejecución del sistema. Y posteriormente realizar una llamada al método *run*(), la parte ejecutable del diagrama , en la cual las instrucciones encontradas dentro del método *run()*, son ejecutadas.

Si alguien invoca al método *suspend()*, *sleep()* o *wait()*[16], el *thread* se detendrá y estará en estado de *"stop"*. Y permanecerá así hasta que pase el lapso de tiempo especificado dentro del método en el caso del *sleep()*, o cuando se invoque al método *resume()*, en el caso de haberlo detenido con *suspend()* o si el *thread* está en espera del cumplimiento de una condición debe llamarse al método *notify()* o *notifyAll()* cada que la variable encargada de la condición varíe.

El *thread* termina cuando concluye de forma habitual su método *run()* o cuando se invoca al método *stop()* dentro del código. Por ejemplo, cuando una página web se minimiza o cuando el *browser* cambia a otra página el *applet* llama al método *stop()* para detener todos sus *threads*.

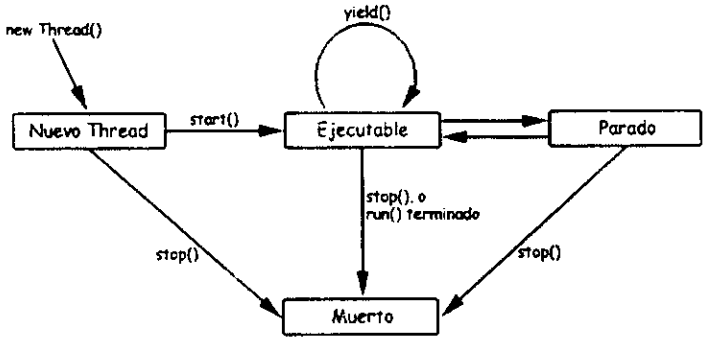



Fig. 1.5. Ciclo de vida de un *thread*.

*16 Mientras espera a que se cumpla determinada condición*

### *1.3.2 Planificador (*Scheduling*).*

El p*lanificador* es una lista de procesos encargados de monitorear los *threads* que se ejecutan dentro de un programa. Además de decidir que *threads* deben ejecutarse y cuales deben estar preparados para su ejecución. Para esta decisión el *planificador* verifica la prioridad de los *threads* (que va de 1 a $10^{17}$) y se auxilia del método *yield()* Fig 1.5., el cual obliga al planificador a comenzar la ejecución de algún otro *thread* que este en espera.

## 1.4 Características de *Java* como lenguaje.

### *1.4.1 Comentarios.*

El lenguaje *Java* maneja 3 tipos de delimitadores para comentarios [7 (pp-186)]. El primer tipo de delimitador es representado por /* ... */, y es útil cuando se requiere un bloque de código como comentario. El segundo delimitador es // y se utiliza para indicar que el resto de la línea es comentario. Este tipo de comentario es muy útil para colocar comentarios junto a las líneas de código.

Finalmente el último delimitador que utiliza *Java*, indica al compilador que lo que se encuentre entre los delimitadores /** y */ se utiliza para generar la documentación *HTML* con ayuda de la herramienta *javadoc*. Los tres tipos de delimitadores son mostrados en la tabla T.1.3.

| Inicio de comentario | Final de comentario | Descripción |
|---|---|---|
| /* | */ | • Lo que se encuentra entre los delimitadores, el compilador lo toma como comentario. |
| // | | • Todo lo que sigue al delimitador, siempre y cuando este en la misma línea es comentario. |
| /** | */ | • Lo que se encuentra entre los delimitadores es comentario. Estos delimitadores permiten utilizar la herramienta *javadoc* para generar documentación. |

Tabla T.1.3. Delimitadores para comentarios

*17 Los threads de prioridad más alta serán primeramente atendidos por el procesador antes de los de prioridad más baja MIN_PRIORITY = 1 a MAX_PRIORITY = 10 como se define en la clase Thread.*

## *1.4.2 Palabras reservadas.*

Dentro del lenguaje *Java*, al igual que en otros lenguajes, existe una gran cantidad de palabras reservadas [1 (pp.35)] que no se pueden utilizar como el nombre de una clase, variable o método, a este conjunto de palabras se les conoce como "palabras reservadas". A continuación se muestran algunas de esas palabras en la tabla T.1.4:

| PALABRAS RESERVADAS. | | | |
|---|---|---|---|
| *abstract* | *double* | *int* | *Static* |
| *boolean* | *else* | *Interface* | *Super* |
| *break* | *extends* | *Long* | *Switch* |
| *byte* | *final* | *Native* | *Synchronized* |
| *byvalue* | *finally* | *New* | *This* |
| *case* | *float* | *Null* | *Throw* |
| *cast* | *for* | *Package* | *Throws* |
| *catch* | *future* | *operator* | *Transient* |
| *char* | *generic* | *Outer* | *Try* |
| *class* | *goto* | *Private* | *Var* |
| *const* | *if* | *Protected* | *void* |
| *continue* | *implements* | *Public* | *volatile* |
| *default* | *import inner* | *rest return* | *while* |
| *do* | *instance of* | *Short* | |

Tabla T.1.4.Palabras reservadas.

## *1.4.3 Tipos de datos en Java.*

El lenguaje *Java* tiene ocho tipos de datos primitivos: *byte*, *short*, *int*, *long*, *float*, *double*. *char*, *boolean*. La tabla T.1.5 muestra el tipo de dato y su longitud en bits.

| Tipo de dato | Medida en bits |
|---|---|
| *Byte* | 8 bits (*Integer*) |
| *Short* | 16 bits (*Integer*) |
| *Int* | 32 bits (*Integer*) |
| *Long* | 64 bits (*Integer*) |
| *Float* | 32 bits (*float point*) |
| *Double* | 64 bits (*float point*) |
| *Char* | Carácter *Unicode* de 16 bits |
| *Boolean* | *True / false* |

Tabla T.1.5. Tipo de datos en *Java*.

El tipo *integer* tiene cuatro subtipos: *byte*, *short*, *int*, *long*. Mostrados en la tabla T.1.6.

| Tipo de dato. | Rango |
|---|---|
| *byte* | 127 a −128 ($2^7-1$ a $-2^7$) |
| *short* | 32 76 a −32 768 ($2^{15}-1$ a $-2^{15}$) |
| *Int* | 2 147 483 647 a −2 147 483 648 ($2^{31}-1$ a $-2^{31}$) |
| *Long* | 9223372036 8547755807L a −92233720368547758081. ($2^{63}-1$ a $-2^{63}$) |

Tabla T.1.6. Subtipos de datos del tipo *integer*.

### *1.4.4 Control de acceso a los datos.*

En el lenguaje *Java* cuando se crea una clase es permitido especificar el nivel de acceso a las variables y métodos definidos en esa clase. Indicando así que partes del código pueden ser accesibles y de que forma. Este nivel de acceso se indica con las palabras reservadas *public*, *protected*, *private* y *friendly* al principio de la declaración.

***public*** .- Permite acceder a cualquier clase desde cualquier lugar a todas las variables y métodos de instancia públicos. A continuación se muestra un ejemplo de cómo declarar una clase pública:

```
public void Miclase() { }
```

***protected*** .- Permite el acceso únicamente a las subclases de la clase. Todas las clases de un paquete pueden ver los métodos protegidos de ese paquete. Si no se requiere utilizar de esta forma es necesario declararlos como *private protected*, para que solo se pueda acceder a las variables y métodos protegidos de las clases derivadas. El siguiente ejemplo muestra como se declara una clase protegida:

```
protected void Miclase() { }
```

***private*** .- Cuando se declara una variable o un método privado, estos sólo pueden ser accedidos desde y dentro de la clase en la que son declarados. Por ejemplo la variable *Nombre_Alumno*, está declarada como una cadena con acceso privado. Por lo tanto solo puede ser accedida desde y dentro de la clase en la que se encuentra declarada..

```
private String Nombre_Alumno;
```

***friendly*** .- Este tipo de acceso es el que *Java* deja por defecto cuando no se especifica ninguno de los anteriores. Es igual que *protected*. Son accesibles por todos los objetos dentro del mismo paquete. Un ejemplo de un método *friendly* es mostrado a continuación:

```
void MiMétodo() { }
```

### *1.4.5 Interfaces.*

Una interfaz se define como un conjunto de métodos abstractos y clases constantes que se implementan en otro lugar [9]. Los métodos de una clase son *public*, *static* y *final*. La interfaz proporciona un mecanismo de encapsulamiento de los protocolos de los métodos sin forzar al programador a utilizar la herencia. A continuación se muestra la forma como declarar una interfaz :

```
Modificadores_de_Interfaz   Nombre de la interfaz   ClausulaExtends {
cuerpo de la Interfaz
}
```

```
Y el ejemplo de la declaración de una interfaz

public EditaTexto {
  public void reemplazaTexto(String s);
  public void mayusculas( );
  public void minuscula( );
}  Fin de la Interfaz
```

Si el usuario requiere hacer uso de una interfaz lo que debe hacer es utilizar la palabra reservada *implements* [8 (pp.-83)] en la declaración de la clase, la cual se encarga de proporcionar el código necesario para la implementación de los métodos definidos en la interfaz, sin necesidad de que la clase interfaz tenga conocimiento de la implementación que hagan otras clases que la utilicen y no importando la localización que tenga en la jerarquía de clases. A continuación se muestra como realizar la implementación de la interfaz declarada arriba:

```
class Modifica implements EditaTexto{
.
..
}  Fin de la clase Modifica
```

## *1.4.6 Métodos*

Los métodos tienen código para manipular el estado de un objeto. Estos pueden ser *públicos*, *protegidos* o *privados* y deben de tener un nombre único, si existe más de un método con el mismo nombre, el método es sobrecargado (*overloader*.).

La definición de un método debe incluir la palabra reservada de control de acceso (*public, protected, private*), lugar del método en la jerarquía (*abstract, final, native, synchronized*), tipo de dato que regresa (datos *primitivos* o *void* [18]), nombre del método, parámetros y excepciones lanzadas por el método.

Existen métodos estáticos, los cuales pueden acceder datos estáticos, crear instancias métodos y acceder instancias de datos.

Un ejemplo de método es el que se utiliza en el programa *CapturaExam* dentro del paquete de *profesor*[19], por mencionar alguno. Este método se encarga de colocar los componentes en la pantalla. A continuación se muestra un fragmento del código.

```
** Este metodo se encarga de establecer los componentes en la pantalla del profesor para introducir las preguntas y
respuestas del examen *
  public void Componentes(){
  fv = new FijarValores().
  gbl = new GridBagLayout().
  gbc = new GridBagConstraints().
  setLayout(gbl);

  l1 = new Label("Introduzca la Pregunta: " +contador + " de " + total);
  l1.setFont(new Font("TimesRoman",Font.ITALIC,14));
  gbc = fv.FijarValores(gbc,0,1,1,1,1,1, new Insets(5,5,5,5));
  add(l1);
```

[18] *void indica que el método no devuelve ningún valor*

[19] *El programa completo de CapturaExam se muestra en un disco anexo al presente trabajo [6]*

```
gbl.setConstraints(l1,gbc);
...
/* Botones */
gbc = fv.FijarValores(gbc,2,10,1,1,1,1, new Insets(5,5,5,5));
b1 = new Button("Aceptar");
b1.addActionListener(new Escucha());
add(b1);
gbl.setConstraints(b1,gbc);
} // Fin de Componentes
```

### 1.4.6.1. Sobreescritura de métodos.

La *sobreescritura* de métodos se utiliza cuando un método heredado es redefinido por una subclase, esto pasa porque cuando se realiza la llamada de algún método, el lenguaje *Java* realiza una búsqueda primero en los métodos asociados del objeto que hizo la llamada y si no lo encuentra entonces sube a la superclase a realizar la búsqueda y así sucesivamente hasta encontrarlo, por lo que cuando se redefine el método se está ocultando el método de la clase madre.

Si lo que se desea es llamar al método original, es necesario utilizar la palabra reservada *super* seguida de un punto y del nombre del método, así como también de los argumentos que recibe éste método.

Ejemplo:

```
super.Mi_Metodo(arg1, arg2);
```

Los constructores no pueden sobreescribirse porque estos tienen el mismo nombre de la clase que se crea. Sólo se pueden inicializar las variables de la nueva clase. Por ejemplo, en el programa *CapturaExam.java* , el constructor está definido de la siguiente forma:

```
public CapturaExam(Frame f, String titulo,int y, int cuenta, String grupo, String clave, String cadena3, String cadena4, String
cadena5, String cadena6, String cadena7,String descripcion){
  super(f,titulo,true);
  total= y;
  contador = cuenta;
  this.grupo = grupo;
  this.clave = clave;
  this.cadena3 = cadena3;
  this.cadena4 = cadena4;
  this.cadena5 = cadena5;
  this.cadena6 = cadena6;
  this.cadena7 = cadena7;
  this.descripcion = descripcion;
  Componentes();
}  Fin del Constructor
```

Nótese que la llamada al constructor de la superclase se realiza mediante

```
super(f,titulo,true);
```

### 1.4.6.2. Sobrecarga de métodos. ("Overloading Methods").

Se llama sobrecarga de métodos *"overloading"* cuando se tienen dos o más métodos con el mismo nombre pero con diferente tipo de parámetros como argumento. Su utilidad radica en que permite proporcionar diferentes implementaciones del mismo método dependiendo del tipo de los parámetros proporcionados. Es importante mencionar que la característica de *overloading* no es única de los métodos sino también de los constructores, de los cuales se habla en el siguiente punto.

## *1.4.7 Constructores y destructores.*

**Constructores.**

El constructor es un método especial propio de cada clase que permite crear una instancia, un nuevo objeto de esa clase. El constructor se encarga de reservar la memoria para el nuevo objeto y de inicializar sus variables.

Para declarar un constructor primero se debe definir el tipo de acceso *(public, private, protected, etc)* y después el nombre con el que se identificará. El nombre del constructor debe ser igual al nombre de la clase y puede tener cero o más parámetros como argumentos.

Ejemplo:

```
...
public class CalifAlumno extends Frame{
private Label l1;
private List l;
private Button b1,b2;
private String s,clave_grupo,login,passw,cadena2,cadena3,cadena4,aux,cad,auxpla;
...

/** El Constructor recibe como parametros la clave del grupo, el login
   y password del profesor del grupo *

public CalifAlumno(String titulo, String clave_grupo, String login, String passw) {
  super(titulo);
  this.s = s;
  this.clave_grupo=clave_grupo;
  this.login=login;
  this.passw=passw;
 System.out.println("clave_grupo " +clave_grupo);
} // Fin del Constructor
...
} // Fin de la clase CalifAlumno
```

La clase *CalifAlumno* tiene un constructor público *CalifAlumno* que recibe como parámetros la cadena *titulo*, una *clave_grupo*, un *login* y un *password*.

**Destructores.**

En *Java* existe el ***"Garbage Collector"*** (Recolector de basura) mediante el cual se destruyen los objetos que no se utilizan dentro de un programa una vez que se haya terminado la definición de un objeto sin necesidad de hacer una llamada explícita.

Si se requiere que el objeto haga algo antes de ser recogido por el *"Garbage Collector"* se debe sobreescribir el siguiente método:

```
protected void finalize() throws Throwable {
}
```

Generalmente los objetos que asignan recursos externos deben proporcionar un método *finalize()* que los limpie. Por ejemplo, cuando se trabaja con una clase que abre archivos para realizar su trabajo debe tener algún método *close*() para cerrar el archivo. A veces se puede no invocar *close()* aunque ya se haya terminado de utilizar el objeto y para garantizar que no se quede ningún archivo abierto utilizamos el método *finalize*() que invoque al método *close( )* de la siguiente manera [8 -p.48]:

```
public class ProcessFile{
private Stream file;
public ProcessFile(String path) {
  file = new Stream(path);
} // Fin de ProcessFile
...
public void close(){
if (file !=null) {
file.close();
fille=null;
} // Fin del if
} // Fin del close

protected void finalize() throws Throwable {
super.finalize();
close();
} Fin de finalize
} Fin de la clase ProcessFile
```

También podemos hacer una llamada explícitamente al *"Garbage Collector"* a través del método *gc()* de la clase *Runtime*:

```
System.gc();
```

# 1.5 Estructuras de control.

## *1.5.1 Condicional if – else*

La sentencia *if* se utiliza para elegir entre rutas alternas. Su sintaxis es la siguiente:

```
if(expresión-booleana)
sentencia 1
else
sentencia2
```

La sentencia 1 se ejecuta cuando la expresión booleana es verdadera y la sentencia 2 cuando es falsa.

### *1.5.2 Condicional switch.*

Se utiliza para evaluar una expresión y transferir el control del programa, dependiendo del valor de la expresión. Su sintaxis es la siguiente:

```
switch (expresion) {
case valor1:
sentencias1;
break;
case valor2:
sentencias2;
break;
default:
sentencias_por_default;
} // Fin del switch
```

El *break*, se utiliza para indicarle al programa que las sentencias correspondientes a un valor sean ejecutadas hasta llegar al *break*, y después salte al final del *switch*. Si la expresión no corresponde a ninguno de los valores proporcionados por el *case* entonces se ejecuta el bloque de sentencias por defecto y sale del bloque *switch*. Este bloque es opcional.[20]

### *1.5.3 Bucle while.*

El *while* se utiliza para la repetición condicional, en la cual el bucle se repite mientras la expresión booleana sea verdadera. Este bucle puede no ser ejecutado ni una sola vez, si la condición booleana no es verdadera. Su sintaxis es la siguiente:

```
while(expresión booleana) {
        sentencia
} // Fin del while
```

### *1.5.4 Bucle do/while.*

Se utiliza para la repetición condicional, un número no determinado de veces. A diferencia del *while* el *do/while* se ejecuta al menos una vez y se repetirá mientras la expresión booleana sea verdadera. Su sintaxis es la siguiente:

```
do{
        sentencia;
} // Fin del do
while(expresión booleana)
```

[20] *Un ejemplo del uso del switch se muestra en el código del programa Protocolo.java*

### *1.5.5 Bucle for.*

Se usa para la repetición de un bloque de sentencias por un número predeterminado de veces. Su sintaxis es la siguiente[21]:

```
for(expresión inicial;expresión booleana; incremento) ){
sentencia1;
sentencia2;
..
} // Fin del for
```

## 1.6 Excepciones.

*Java* proporciona un sistema para el tratamiento de excepciones y errores en tiempo de ejecución, con lo cual se pueden detectar situaciones anómalas y responder a ellas con un mínimo impacto en el programa principal; esto se hace mediante la adición del código necesario para su tratamiento. Las excepciones son objetos de la clase *java.lang.RuntimeException* o de sus subclases [8 (pp.- 139)].

### *1.6.1 ¿Qué es una excepción y cómo manejarla?.*

Se considera como una excepción a cualquier evento que ocurra durante la ejecución de un programa que ocasiona un mal funcionamiento al flujo normal de un programa.

Las causas que pueden provocar una excepción son; entre otras: Intento de lectura de un archivo inexistente, una división por cero, intento de acceso a un elemento dentro de un arreglo fuera de su dimensión, etcétera. Una vez que un error de estos ocurre dentro de un método *Java*, el método genera un objeto de excepción y lo maneja fuera de sistema en tiempo de ejecución. Dicho objeto contiene información que indica el tipo de excepción de que se trata, así como el estado en que se encontraba el programa en el momento en que sucedió. El sistema en tiempo de ejecución es el responsable de encontrar el código de programa apropiado para manejar el error. En términos de *Java* se dice que el sistema en tiempo de ejecución lanza una excepción (*"throwing an exception"*)

Inmediatamente después de que una excepción es lanzada por un método, el sistema en tiempo de ejecución interrumpe su acción normal y busca la manera de tratarla. Las maneras de tratar la excepción son un conjunto de métodos almacenados en la pila de llamadas del método que la genera. El sistema hace una búsqueda hacia atrás hasta encontrar el método que contenga el manejador apropiado. Un manejador de excepción es

[21] *Se puede ver un ejemplo del for dentro de los programas desarrollados en el presente trabajo de tesis.*

considerado apropiado cuando el tipo que éste puede manejar corresponde al tipo de error a manejar. El manejador escogido se dice que atrapa la excepción (*"catch the exception"*).

Si el sistema en tiempo de ejecución no encuentra un manejador que coincida con la excepción lanzada, este termina y consecuentemente también el programa.

## *1.6.2 Ventajas del uso del manejo de excepciones.*

Los programas *Java* tienen las siguientes ventajas mediante el uso de excepciones:

- Separar el manejo de errores del código regular
- Propagar los errores sobre la pila de llamadas
- Agrupar los tipos de errores y diferenciarlos

## *1.6.3 Lanzamiento de excepciones.*

Las excepciones se lanzan mediante la sentencia *throw*. Su sintaxis es la siguiente:

*throw Expresión;*

En donde "Expresión" debe evaluar un objeto que es una instancia de una subclase de la clase *java.lang.Exception*. La clase *Exception* está definida dentro de la API de *Java*. Si una excepción es lanzada y esta no llega a ser capturada, el hilo de ejecución actual terminará y se visualizará un mensaje de error en la pantalla.

## *1.6.4 Captura de excepciones.*

Como se indicó anteriormente, *Java* debe atrapar o especificar una excepción que sea lanzada.

Un método puede atrapar una excepción proporcionando un manejador para tal tipo de excepción, o bien , si el método decide no atrapar la excepción deberá de indicarse que este método la puede lanzar. Debido a que una excepción que puede ser lanzada por un método es parte de su interfaz pública, quienes hagan llamado al método deben conocer su existencia; por lo tanto las excepciones se especifican en la firma del método para que de esta forma se puedan manejar apropiadamente.

Los componentes de un manejador de excepciones son los bloques de sentencias *try*, *catch* y *finally* .

- **El bloque *try***

Es el bloque que marca el inicio de un manejador de excepciones y está identificado mediante la sentencia *try* que es quien controla los demás bloques de sentencias que este abarca y define el alcance de cualquiera de los manejadores asociados a él.

- **El bloque *catch***

El bloque *catch* puede capturar una excepción si su argumento puede asignarse legalmente al objeto lanzado por una instrucción *throw*. La sentencia *try*, probará secuencialmente cada una de estas cláusulas y selecciona á la primera que pueda capturar la excepción.

- **El bloque *finally***

Este bloque marca el fin del bloque *try* y proporciona un mecanismo que permite al método ordenarse a sí mismo, independientemente de lo que suceda en el bloque *try*.

Ejemplo:

```
public void Ejemplo() {
    try {
        /*
         * codigo ....
         */
    } catch (parámetro) {
        /*
         * codigo ....
         */
    } catch (Parámetro) {
        /*
         * codigo ....
         *
    } finally {
        /*
         * codigo ....
         */
      }
      }
}
```

A continuación la figura 1.6. muestra algunas de las excepciones más utilizadas:

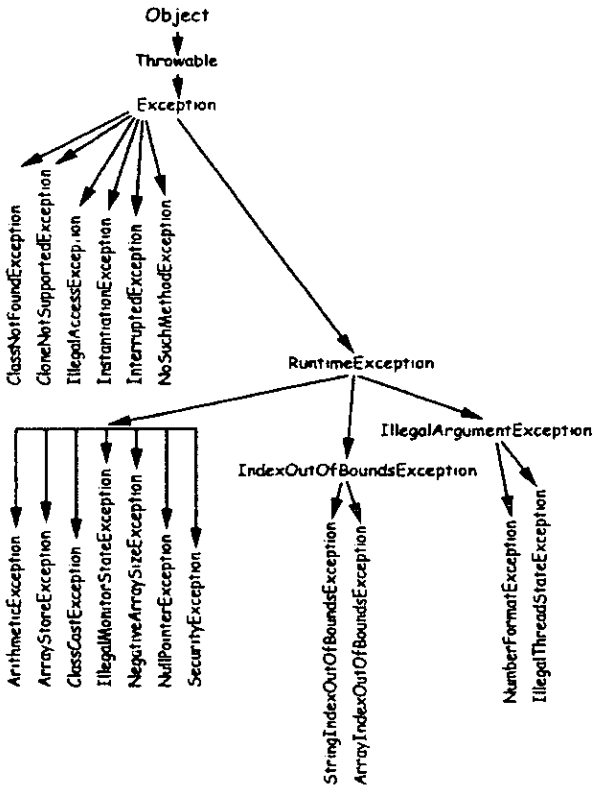



Fig 1.6. Excepciones.

# 1.7 Aplicaciones Standalone y *Applets*.

El lenguaje *Java* maneja dos tipos de aplicaciones las *standalone* y los *applets*.

## *1.7.1 Aplicaciones Standalone.*

Las aplicaciones *standalone* a diferencia de los *applets* no necesitan estar dentro de una página *Web*, éstas pueden correr desde el *appletviewer*, basta con teclear lo siguiente desde el prompt:

```
% java OrganizaPeticiones
```

En donde *OrganizaPeticiones* se refiere al programa *OrganizaPeticiones.java* compilado y % es el promt.

## *1.7.2 Applets.*

Un *applet* es un programa de *Java* que se integra dentro de un página *Web*, por lo tanto para su visualización es necesario un *navegador* que soporte *Java* o desde la herramienta *appletviewer* proporcionada por el *JDK*. Esto se debe a que el *navegador* y el *appletviewer* contienen dentro la Maquina *"Virtual de Java"*, encargada de interpretar el código de bytes, que se integra dentro del código *HTML*, a través de la página *Web*.

Las páginas *Web* hacen referencia a los *applet* mediante la etiqueta <*APPLET*> de la siguiente manera:

```
<HTML>
<HEAD>
<TITLE>Programa en Java para el modulo del administrador</TITLE>
</HEAD>
<BODY BACKGROUND="./imagenes/T013.gif">
<H1><CENTER> Administrador </BLINK></CENTER></H1>

<APPLET CODE = "AppletAdmon.class" WIDTH = 400 HEIGHT = 250 >
</APPLET>

</BODY>
</HTML>
```

En donde *AppletAdmon.class* son los *bytecodes* generados al compilar el programa *AppletAdmon.java*.

Los *applets* se desarrollan creando subclases de la clase *Applet* del paquete *java.applet*.

### *1.7.3 Ciclo de vida de un applet.*

El ciclo de vida de un *applet* está compuesto por cuatro etapas [1], cada una de ellas tiene asociado un método.

El primer método es el de "Inicialización" también conocido como "*start*". En él se crea una instancia del *applet*, se inicializa y comienza la parte de ejecución.

El siguiente método es el de "*run*" . Donde se ejecuta el *applet*.

Otro de los métodos es "*stop*". El cual ocurre cuando se sale de la página o cuando la aplicación es iconizada (minimizada)

Y finalmente el método de liberación. El cual ocurre cuando se sale del *browser* y se borra la memoria caché para hacer la descarga y nueva carga efectiva.

# Capitulo 2. Software y Sistema.

En el presente capítulo se habla acerca del software (*JDK1.1.6.*) utilizado para el desarrollo del ***Sistema de Evaluación*** elaborado en el presente trabajo de tesis con el propósito de comprender mejor la realización de este sistema y los programas mostrados en el apéndice anexo, en un disco, al presente trabajo.

Se comenzará con una breve definición del Conjunto de *Herramientas Java para Desarrollo* también llamado "*Java Developer Kit" (JDK)* y se explicará cómo realizar la instalación del mismo sobre el sistema operativo *Solaris 2.6*. Después, se hablará acerca de las herramientas que proporciona el *JDK* al programador, de los paquetes, su estructura y algunos de sus métodos.

Es importante mencionar que se explicarán sólo aquellos paquetes que se hayan usado en la parte de programación de la aplicación realizada en este trabajo, tales como el paquete *java.applet* , *java.awt*, *java.io*, *java.lang*, *java.net*, *java.sql* y *java.uti*, con el propósito de conocer la función de cada uno de ellos. Y para su ejemplificación se mostrarán fragmentos de código del *Sistema de Evaluación* [22].

El paquete *java.applet* proporciona las clases para la implementación de *applets*.

El paquete *java.awt* se encarga de las clases para el *GUI ("Graphic User Interface")*, ventanas, *scrollbars*, etc..

El paquete *java.io* es utilizado para todo tipo de entrada y salida.

El paquete *java.lang* contiene un conjunto de clases para el lenguaje.

El paquete *java.net* proporciona un conjunto de clases para programar en entornos de red a través de los protocolos de comunicaciones que utiliza *Internet*.

El paquete *java.sql* contiene el *JDBC* ("*Java Database Conectivity*"), que se explica a detalle en el siguiente capítulo.

El paquete *java.util* contiene una serie de clases e interfaces útiles para diversas cosas de programación .

## 2.1 Herramientas de desarrollo *Java* (*JDK*).

"*Java Developer Kit" (JDK)* es la herramienta de desarrollo *Java* que permite escribir *applets* y aplicaciones. Incluye un conjunto de bibliotecas y clases que proporcionan tipos de datos básicos, capacidad de sistema de entrada y salida y otras funciones útiles. También clases para soportar aplicaciones de red y protocolos comunes de *Internet*.

[22] *Aplicación realizada en el presente trabajo de tesis.*

Estas bibliotecas están escritas en *Java*, son portables a través de las diferentes plataformas y comprenden métodos y variables.

El *JDK* también incluye un conjunto de herramientas tales como, el compilador de *java*, su intérprete y el *appletviewer*[23], entre otros.

La plataforma 1.1. ofrece internacionalización, firma de *applets*, archivos en formato *JAR*, *AWT*, *JavaBeans*, paquete matemático, métodos de invocación remota *RMI*, reflexión, conectividad a bases de datos (*JDBC*[24]) y red, nueva interfaz nativa *java*, serialización de objetos e *inner clases*.

## 2.2 Características del *JDK* 1.1

**Internacionalización**.- Permite al programador incluir el despliegue de caracteres de código *UNICODE*, y realizar programas con localizador sensitivo a fechas y tiempo según la zona; contiene además un conjunto de convertidor de caracteres, formato de parámetros y soporte para encontrar caracteres, palabras ó limites de sentencias.

**Firma de *applets***.- La versión del *JDK 1.1.* incluye *APIs* para firmas digitales e interfaces abstractas para el manejo de llaves, certificados y control de acceso. Además incluye una herramienta para la firma de archivos *JAR* ("*Java Archive*"), los cuales pueden contener clases y otros datos, tales como imágenes y sonidos. El *appletviewer* permite a cualquier *applet* en formato *JAR* firmarse utilizando esta herramienta para probar su identidad y ejecutarse con todos los derechos como una aplicación local.

***AWT* ("*Abstract Windows Toolkit*")**.- Incluye *APIs* para imprimir, para el manejo de barras de desplazamiento (*scrolling*), *popups* menú , *clipboard (copy/paste)*, cursores por componente, modelos de eventos, mayor flexibilidad en fuentes para dar soporte a la internacionalización, entre otros.

***JavaBeans***.- Un *Java Beans* es un componente de software reutilizable que puede ser manipulado por una herramienta de ensamblado de *Beans*, tal como un editor de páginas *HTML*, un editor de interfaces gráficas *Java* ó el *Sunsoft Java Studio* por mencionar algunos. Un componente de software debe ser comercializable; es decir, puede venderse por separado de cualquier otro programa porque cumple una función bien definida, puede ser utilizado en una aplicación compleja hecha de componentes de orígenes diversos, no constituye nunca una aplicación completa. Y pueden ser de diferentes tamaños que van desde un componente pequeño (un objeto en *Java*) hasta un componente grande (una hoja de cálculo).

**Archivos en formato *JAR***.- Un archivo en formato *JAR* es independiente de la plataforma y contiene muchos archivos dentro de uno solo. Por ejemplo un *applet* y sus

[23] *Visualizador de applets.*
[24] *"Java DataBase Conectivity"*

archivos *.class* y algunos otros componentes necesarios tales como imágenes y sonidos pueden estar en un archivo con formato *JAR* para ser bajados a través de un *browser*. Es importante mencionar que el formato *JAR* soporta la compresión de archivos por lo que reduce el tamaño del archivo.

***Paquete matemático.***- El paquete matemático proporciona dos nuevas clases: *BigInteger* y *BigDecimal*. La primera permite definir enteros de longitud variable y la segunda permite gestionar los números reales con una buena precisión, lo cual resulta muy útil para los cálculos financieros con grandes cantidades.

**Métodos de invocación remota *(RMI)*.-** *RMI* (*"Remote Method Invocation"*), permite al programador realizar aplicaciones distribuidas, en las cuales los métodos de los objetos remotos *Java* pueden ser invocados desde otra máquina virtual *Java*, posiblemente en diferente *host*.

**Serialización de objetos.-** La serialización le permite al lenguaje *Java* almacenar los objetos en archivos que se comuniquen a través de la red y que se usen en aplicaciones distribuidas.

**Reflexión.-** Permite la identificación del campo, método, objeto o constructor que se va a utilizar dependiendo de la clase que realice la llamada en tiempo de ejecución.

**Conectividad a bases de datos *(JDBC)*-** El *JDBC* (*"Java DataBase Conectivity"*) es una interfaz estándar para el acceso a bases de datos. Esta interfaz se ve a detalle en el siguiente capítulo.

**Conectividad a red.-** La conectividad a red se realiza mediante el paquete *java.net* .

***Inner clases-*** Proporciona una sintaxis simple para la creación de un adaptador de clases. Un adaptador de clases es una clase que implementa una interfaz (o clase) requerida por un *API*, y delega el flujo de control cuando se cierra el programa que la invocó.

**Interfaz nativa *Java*.-** Es una interfaz estándar de programación utilizada para la escritura de métodos nativos en *Java*

## 2.3 Instalación del *JDK*.

El *JDK* utilizado para la realización de la aplicación es el *JDK 1.1.6*. Esta versión es de mantenimiento [14] y no contiene nuevas características en comparación con las versiones anteriores.

En el caso del presente trabajo esta versión ya venía instalada con el sistema operativo *Solaris 2.6*[25]. Lo único que se tuvo que realizar fue la declaración de él en la variable de ambiente *CLASSPATH* y en el *PATH* dentro del archivo *.cshrc*[26].

Si el *JDK* no hubiera estado instalado en el sistema operativo, esta herramienta se puede conseguir libremente en la siguiente dirección: *http://www.java.sun.com* , de acuerdo al tipo de plataforma que se tenga, al igual que la documentación asociada a la versión. A continuación se muestra la forma de realizar la instalación del *JDK1.1.5* bajo sistema operativo *UNIX* (*Solaris*):

Lo único que se tiene que hacer es bajar los archivos binarios *JDK115Solaris2sparc.bin* (tamaño 12694373) y realizar lo siguiente [13]:

### *2.3.1 Instalacón del JDK1.1.5*

```
[erickacv]/usr/local # ls
total 33248
-rw-r--r--  1 root   other   4297073 Feb 23 20:54 JDK115-doc.tar.Z
-rw-r--r--  1 root   other   12694373 Feb 23 20:23 JDK115-Solaris2-sparc.bin
```

Dar los permisos necesarios al archivo binario.

```
[erickacv]/usr/local #   chmod a+x JDK115-Solaris2-sparc.bin
[erickacv]/usr/local #   ls
total 33248
drwxr-xr-x  4 root   other     1024 Feb 11 15:26 bin
drwxr-xr-x  3 root   other      512 Jan 27 18:00 etc
drwxr-xr-x  2 root   other      512 Jan 27 12:52 include
drwxr-xr-x  2 root   other     1024 Jan 29 17:36 info
-rw-r--r--  1 root   other  4297073 Feb 23 20:54 JDK115-doc.tar.Z
-rwxr-xr-x  1 root   other  12694373 Feb 23 20:23 JDK115-Solaris2-sparc.bin
drwxr-xr-x  5 root   other      512 Feb 11 19:01 lib
drwxr-xr-x  3 root   other      512 Jan 27 12:52 man
drwxr-xr-x  3 root   other      512 Jan 27 12:52 sparc-sun-Solaris
drwxr-xr-x  3 root   other      512 Jan 29 17:32 sparc-sun-Solaris2.5.1
```

Ejecutar el archivo binario:

```
[erickacv]/usr/local #   ./JDK115-Solaris2-sparc.bin

                Java(tm) Development Kit
```

[25] *Sistema operativo para máquinas SUN.*
[26] *Archivo de configuración en el sistema operativo UNIX para el cshell.*

```
                    Version 1.1.5

                    Binary Code License

This binary code license ("License") contains rights and restrictions
associated with use of the accompanying software and documentation
("Software"). Read the License carefully before installing the Software.
By installing the Software you agree to the terms and conditions of this
License.

1. Limited License Grant. Sun grants to you ("Licensee") a non-exclusive,
non-transferable limited license to use the Software without fee for
evaluation of the Software and for development of Java(tm) compatible
applets and applications. Licensee may make one archival copy of the Software
...
..
 ..
held to be in
violation of applicable law, void, or unenforceable in any jurisdiction,
then such provisions are herewith waived to the extent necessary for the
License to be otherwise enforceable in such jurisdiction.  However, if
in Sun's opinion deletion of any provisions of the License by operation
of this paragraph unreasonably compromises the rights or increase the
liabilities of Sun or its licensors, Sun reserves the right to terminate
the License and refund the fee paid by Licensee, if any, as Licensee's
sole and exclusive remedy.

Do you agree to the above license terms? [yes or no]
Yes

Unpacking...
Checksumming...
0
0
Extracting...
Archive:  ./install.sfx.23417
  creating: JDK1.1.5/
  inflating: JDK1.1.5/CHANGES
  inflating: JDK1.1.5/COPYRIGHT
 ..
...
...
...

inflating: JDK1.1.5/lib/psfont.properties.ja
  inflating: JDK1.1.5/lib/appletviewer.properties
  inflating: JDK1.1.5/lib/rmic.properties
  inflating: JDK1.1.5/lib/serialver.properties
  inflating: JDK1.1.5/lib/classes.zip
  inflating: JDK1.1.5/src.zip
Done

[erickacv]/usr/local #    ls
total 33250
drwxr-xr-x  4 root    other     1024 Feb 11 15:26 bin
drwxr-xr-x  3 root    other      512 Jan 27 18:00 etc
drwxr-xr-x  2 root    other      512 Jan 27 12:52 include
drwxr-xr-x  2 root    other     1024 Jan 29 17:36 info
drwxr-xr-x  6 root    other      512 Feb 24 17:58 JDK1.1.5
-rw-r--r--  1 root    other  4297073 Feb 23 20:54 JDK115-doc.tar.Z
-rwxr-xr-x  1 root    other 12694373 Feb 23 20:23 JDK115-Solaris2-sparc.bin
drwxr-xr-x  5 root    other      512 Feb 11 19:01 lib
drwxr-xr-x  3 root    other      512 Jan 27 12:52 man
drwxr-xr-x  3 root    other      512 Jan 27 12:52 sparc-sun-Solaris
drwxr-xr-x  3 root    other      512 Jan 29 17:32 sparc-sun-Solaris2.5.1
```

Ir al subdirectorio donde se hizo la instalación:

```
[erickacv]/usr/local #      cd JDK1.1.5
[erickacv]/usr/local/JDK1.1.5 #      ls
total 3416
-r--r--r--  1 root    other     94168 Nov 24 19:41 CHANGES
-r--r--r--  1 root    other       910 Nov 24 19:41 COPYRIGHT
-r--r--r--  1 root    other      5994 Nov 24 19:41 LICENSE
-r--r--r--  1 root    other     34548 Nov 24 19:41 README
drwxr-xr-x  3 root    other       512 Feb 24 17:57 bin
drwxr-xr-x 25 root    other       512 Feb 24 17:57 demo
drwxr-xr-x  5 root    other      1024 Feb 24 17:58 include
-r--r--r--  1 root    other      2384 Nov 24 19:41 index.HTML
drwxr-xr-x  4 root    other      1024 Feb 24 17:58 lib
-rw-r--r--  1 root    other   1594253 Nov 24 19:41 src.zip
```

Declarar las variables de ambiente en el archivo de configuración llamado .cshrc[27]

```
[erickacv]/#    vi .cshrc
...
...
setenv CLASSPATH :/ usr/local/JDK1.1.5/
set path=($path /usr/local/JDK1.1.5/bin)
Cargar el .cshrc:

[erickacv]/# source ~/.cshrc


[erickacv]/usr/local/JDK1.1.5% which java
/usr/local/JDK1.1.5/bin/java
```

## 2.3.2 *Herramientas del JDK.*

Como ya se mencionó al inicio del capítulo, *java* proporciona además de las bibliotecas un conjunto de herramientas muy útiles para el desarrollo de *applets* y aplicaciones. A continuación se verán cuáles son las herramientas que se instalan junto con el *JDK* [15]:

***javac***.- Es el compilador de *java*. Este se encarga de pasar el código fuente a código de bytes (*bytecode*)

**Interprete (*java*)**.- Como su nombre lo dice es el encargado de interpretar el código de bytes que genera el compilador y permitir de esta manera ejecutar el programa.

***Appletviewer***.- Esta herramienta permite visualizar *applets* sin necesidad de *browser Java* compatible.

***Java Debugger (jdb)***.- Muy útil para depurar programas.

**Desensamblador *java* (*javap*)**.- Se utiliza para reproducir el código fuente a partir de código compilado.

**Generador de documentación (*javadoc*)**.- Produce una jerarquía de clases y un índice. Crea un conjunto de páginas *HTML* describiendo las clases públicas y protegidas, interfaces, constructores, métodos y campos, gracias a los comentarios que se hayan puesto

[27] *Si se tratara de la instalación en un ambiente Windows la declaración de variables se hace en el autoexec.bat de la siguiente manera:*
*PATH C:\orawin95\bin;"C:\JDK1.1.5\BIN"*
*SET CLASSPATH=.;[bin]\..\classes;[bin]..lib\classes.zip*

a lo largo de los programas dentro de los delimitadores de comentario especiales para la documentación[28].

***Generador de archivos de cabecera C***- Esta herramienta se utiliza para generar archivos de cabecera en lenguaje C y archivos fuente a partir de un archivo *.class* [29]

***Java Archive Tool (jar)***.- Combina diferentes archivos dentro de un archivo *jar*.

**Firma digital (*javakey*)**.- Esta herramienta se encarga del manejo de entradas incluyendo llaves y certificados, así como la veracidad asociada a ellos.

**Convertidor Nativo a *ASCII* (*native2ascii*)**- Utilizado para convertir código nativo en *ascii*.

## 2.4 Paquetes.

Un paquete es un conjunto de clases e interfaces. Los nombres de los paquetes son palabras separadas por puntos y se almacenan en directorios que coinciden con esos nombres. Por ejemplo, el paquete *java.applet* tiene su conjunto de clases bajo el directorio *java/applet*

Para hacer referencia a ellos dentro de un programa se debe importar el paquete que se requiera, por ejemplo para importar el paquete *java.applet* es necesario escribir lo siguiente en la primera línea del código[30]:

*import java.applet.Applet;*
ó
*import java.applet.*;*

En la primera línea le decimos al programa que utilice únicamente la *clase Applet* del paquete *java.applet*. Y en la segunda estamos importando todas las clases del paquete *java.applet.*.

A continuación se explican los paquetes utilizados en la programación de la aplicación desarrollada en el presente trabajo.

### *2.4.1 El paquete java.*net

Este paquete tiene un conjunto de clases útiles para programar en entornos de red a través de los protocolos de comunicaciones que utiliza *Internet* ("*Internet Protocol*" IP; "*Transport Control Protoco*" TCP y el "*User Datagram Protocol*" UDP) [3].

Este paquete permite realizar la comunicación cliente/servidor a través de *sockets* (se definen en el siguiente punto). Está compuesto por las clases mostradas en la tabla

[28] *La documentación generada mediante el javadoc se encuentra en un disco anexo al presente trabajo.*
[29] *Contiene código bytecode*
[30] *Para ver más ejemplos, por favor consulte el código de los programas en el disco anexo a este trabajo*

T.2.1, pero en este trabajo solo se explicarán aquellas que se utilizaron en el desarrollo de la aplicación[31]:

| CLASES DEL PAQUETE *java.net* |
|---|
| *ContentHandler* |
| *DatagramPacket* |
| *DatagramSocket* |
| *InetAddress* |
| *ServerSocket* |
| *Socket* |
| *SocketImpl* |
| *URL* |
| *URLConnection* |
| *URLEncoder* |
| *URLStreamHandler* |

T.2.1.Clases del paquete *java.net*

### 2.4.1.1. Clase *Socket.*

Una aplicación servidor usualmente escucha un puerto específico esperando una petición de conexión de un cliente. Cuando llega una petición de conexión, el cliente y el servidor establecen una conexión dedicada sobre la que pueden comunicarse. Durante el proceso de conexión, el cliente es asignado a un número de puerto, asignando un *socket*[32] a la conexión. El cliente se comunica con el servidor escribiendo sobre el *socket* y obtiene información del servidor cuando lee de él. En el desarrollo de la aplicación sobre *Internet*, se utilizaron *sockets* para establecer la comunicación entre el servidor y el cliente mediante el puerto 1101. También se implementó un programa para atender múltiples peticiones a través del mismo puerto[33] y evitar la necesidad de asignar un puerto a cada petición.

Todo esto es posible gracias a la **clase *Socket*** [7(pp.294)] que tiene los métodos mostrados en la tabla T.2.2.

| Método | Descripción |
|---|---|
| *close()* | Cierra el *socket* |
| *GetInetAddres()* | Obtienen la dirección en donde el *socket* está conectado. |
| *GetInputStream()* | Obtiene el *InputStream* para el *socket* |
| *GetLocalPort()* | Devuelve el puerto en el que esta conectado el *socket* |
| *GetOutputStream()* | Devuelve el *OutPutStream* del *socket* |
| *GetPort()* | Obtiene el puerto remoto en el cual el *socket* está conectado |
| *SetSocketImplFactory* | Cambia a una implementación de *socket* personalizada. |
| *ToString* | Convierte un *socket* a una cadena |

Tabla T.2.2. Métodos de la clase *Socket*

[31] *Para mayor detalle consultar la documentación del JDK 1.1.1.*

[32] *Los sockets son los puntos finales de una interfaz de comunicación sobre un protocolo de red.*

[33] *El programa que implementa el servidor que atiende múltiples peticiones es el llamado "ServidorMultipeticiones.java" y se muestra en el disco anexo a este trabajo.*

### 2.4.1.2. Clase Server *Socket.*

Esta clase implementa un *socket* del servidor *TCP*. En el *Sistema de Evaluación* desarrollado en el presente trabajo se creó un objeto de tipo *ServerSocket* especificando el puerto por el cual escucha el servidor del *socket* las peticiones de la siguiente manera:

*ServerSocket servidorSocket = new ServerSocket(1101);*

Donde 1101, es el número de puerto[34].

La clase *Server Socket* tiene entre sus métodos uno que se llama *accept()*, el cual sirve para que el servidor escuche y espere mientras se establece la conexión de entrada.

Ejemplo:

*new ServidorMultipeticiones(servidorSocket.accept()).start();*

También tiene otros métodos mostrados en tabla T.3.2.

| Método | Descripcion |
|---|---|
| *Close()* | Cierra el *server socket*. |
| *GetInetAddres()* | Obtiene la dirección en la que el *socket* está conectado. |
| *GetLocalPort()* | Obtiene el puerto en el cual el *socket* está escuchando. |
| *GetSocketFactory(SocketImplFactory)* | Permite personalizar la implementación predeterminada del *ServeSocket*. |
| *ToString()* | Devuelve una cadena con la dirección y número de puerto del *socket*. |

Tabla T.3.2. Métodos de la clase *Server Socket.*

## *2.4.2 El paquete java.*awt

El paquete AWT[35] contienen las bibliotecas necesarias para el desarrollo de Interfaces de Usuario Gráficas. Incluye las clases *Button*, *Checkbox*, *Choice*, *Component*, *Graphics*, *Menu*, *Panel*, *TextArea* y *TextField*. Su estructura básica se basa en Componentes y Contenedores.

Los Componentes ayudan al usuario a interactuar con la aplicación y a proporcionar información desde el programa. En el *AWT* los Componentes son instancias de la clase *Component* y siempre se encuentran dentro de un Contendor.

Los Contenedores son instancias de la clase *Container* y se utilizan para organizar y contener a los Componentes.

[34] *El programa se encuentra en la sección de apéndices y se llama "OrganizaPeticiones.java".*
[35] *Acrónimo de "Abstract Window Toolkit" para Java*

### 2.4.2.1. Componentes.

Los componentes son elementos gráficos básicos que constituyen una interfaz gráfica, pertenecen a la clase *Component*, la cual representa todo aquello que tiene una posición, un tamaño, puede ser pintado en pantalla y recibir algún evento. Algunos de estos son mostrados en la tabla T.3.3:

| Componente | Descripcion |
|---|---|
| *Button* | Define un botón de pulsación con un texto |
| *Canvas* | *Lienzo*. Utilizado para trazar imagines |
| *Checkbox* | Botones de comprobación, se utilizan frecuentemente como botones de estado. Proporcionan información del tipo Sí o No (*true o false*), dependiendo si se ha seleccionado o no. |
| *Choice* | Definir una lista de opciones (*combo-box*) |
| *Container* | Es un componente que puede contener a otros componentes, para hacerlo posible, se utilizan sus subclases:<br>*Panel*.- Contenedor genérico que se puede mostrar dentro de un *applet* ó ventana<br>*Window*.- Ofrece una clase superior común para las ventanas de las aplicaciones (objetos *frame*) y para ventanas *Dialog* |
| *Label* | Etiquetas. Se utilizan para colocar texto estático en la pantalla. |
| *List* | Listas. Sirven para la manipulación de muchos elementos. |
| *Scrollbar* | Barras de desplazamiento. Se utilizan para trabajar con rangos de valores lineales ó áreas. |
| *TextComponent* | *Text Componeni* es la clase superior de todas las clases basadas en texto. Proporciona un conjunto de métodos que emplean sus subclases :<br>*TextField* .- Son campos de texto<br>*TextArea*.- Son área de texto que permiten al usuario incorporar texto multilinea . |
| *Dimenston* | La clase *Dimension* se utiliza para representar la altura y anchura de un objeto bidimensional. |
| *Even* | La clase *Even*, captura las características fundamentales de los eventos generados por el usuario, tales como los relacionados con los movimientos del ratón y del teclado. |
| *FileDialog* | Está clase se emplea para construir cuadros de diálogo que soportan la selección de archivos para operaciones de entrada y salida. Proporciona métodos que se emplean para acceder al directorio y nombre de los archivos seleccionados por el usuario. |
| *Flow Layout* | Se utiliza para situar los componentes de un objeto *Container* en disposición de izquierda a derecha y de arriba abajo. |
| *Font* | La clase *Font* implementa un conjunto de fuentes independientes del sistema que controla la visualización de los textos. Estas fuentes son: *Courier Dialog. DialogInput. Helvetica. TimesRoman. y ZapfDingbats*. Esta clase también ofrece métodos para consultar los parámetros de una fuente. |
| *FontMetrics* | Esta clase se emplea para acceder a los parámetros de visualización de un objeto *Font*, tales como tamaño, estilo, nombre de la fuente. |
| *Graphics* | Está clase soporta opciones de dibujo y texto dentro de una ventana. |
| *GridLayout* | Se utiliza para situar los componentes de un objeto *Conteiner* en una cuadrícula. Todos los componentes tienen el mismo tamaño. |
| *GridBagLayout* | Es igual que el *GridLayout*, sólo que aquí los componentes pueden ser de diferente tamaño. |
| *Image* | Define clases e interfaces para generación, almacenamiento y procesamiento de una imagen. |
| *Insets* | Especifica la cantidad de espacio alrededor del borde interior de un contenedor, donde no se situara ningún componente. |
| *MediaTracker* | Ofrece un conjunto de métodos para el manejo de imágenes. |
| *MenuComponent* | Está clase es utilizada para generar menús, utiliza a :<br>*MenuBar* .- Para crear una barra de menús<br>*MenuItem*.- Se utiliza para implementar elementos que pueden seleccionarse desde un menú desplegable que pueda asociarse a una barra de menús o a otro menú. |
| *Point* | Se utiliza para representar coordenadas bidimensionales del tipo "X" y "Y" |
| *Polygon* | Se utiliza para representar un polígono como una lista de coordenadas "X" y "Y" que identifican los vertices de un polígono. |
| *Rectangle* | Se utiliza para representar un rectángulo mediante las coordenadas "X" y "Y" |
| *Toolkit* | La clase *toolkit* proporciona un enlace entre la implementación independiente de la plataforma *Java* y sus caracteristicas especificas de la plataforma |

Tabla T.3.3. Lista de *Componentes*.

### 2.4.2.2. Contenedores.

Los componentes no se encuentran aislados, siempre se encuentran agrupados dentro de los contenedores, los cuales contienen y organizan a los componentes. Los contenedores son en sí mismo componentes y como tales pueden ser contenidos por otros contenedores. Estos son instancias de la clase *Container*.

A continuación se muestra la jerarquía de clases [18] e interfaces que proporciona *AWT* para la creación de interfaces de usuario (cuadro C.2.1.).

**Clases:**

- BorderLayout
- CardLayout
- CheckboxGroup
- Color
- Component
  - Button
  - Canvas
  - Checkbox
  - Choice
  - Conteiner
    - Panel
    - Window
      - Dialog
      - Frame
  - Label
  - List
  - Scrollbar
  - TextComponent
    - TextArea
    - TextField
  - Dimension
  - Event
  - FileDialog
  - FlowLayout
  - Font
  - FontMetrics
  - Graphics
  - GridLayout
  - GridBagConstraints
  - GridBagLayout
  - Image
  - Insets
  - MediaTracker
  - MenuComponent
    - MenuBar
    - MenuItem
      - CheckboxMenuItem
      - Menu
  - Point
  - Polygon
  - Rectangle
  - Toolkit

**Interfaces:**

- LayoutManager
- MenuContainer

C.2.1. Jerarquía de clases e interfaces de *AWT* para la creación de interfaces de usuario.

## 2.4.3 *El paquete java.*io

Contiene las clases relativas a entradas y salidas, las cuales se encuentran organizadas a través de *streams*. Un *stream* es una secuencia objetos.

En el paquete *java.io* todas las entrada se realizan a través de las subclases de la clase *InputStream* y las salidas a través de las subclases de la clase *OutpuStream*. Estas subclases contienen un conjunto de clases para el manejo de flujo de bytes.

Este paquete contiene también un conjunto de clases para el manejo del flujo de caracteres derivadas de la clase *Reader* y *Writer*, además de algunas clases para controlar el acceso a los archivos tales como la clase *File, FilenameFilter, RandomAccessFile* y las clases de *ObjectOutpuStream* y *ObjectInputStream* utilizadas en la serialización .

La jerarquía de clases [18] definida en el paquete *java.io* se muestra a continuación (cuadro C.2.2.[36]) las clases mostradas en negritas están en *java.io* y las otras en *java.lang*:

- Object
  - interface **DataInput**
  - interface **DataOutput**
  - **InputStream**
    - **FileInputStream**
    - **PipedInputStream**
    - **ByteArrayInputStream**
    - **StringBufferInputStream**
    - **SequenceInputStream**
    - **FilterInputStrean**
      - **BufferedInputStream**
      - **DataInputStream**
      - **LineNumberInputStream**
      - **PushBackInputStream**
  - **StreamTokenizer**
  - **OutputStream**
    - **FileOutputStream**
    - **PipedOutputStream**
    - **ByteArrayOutputStream**
    - **FilterOutputStream**
      - **BufferedOutputStream**
      - **DataOutputStream**
      - **PrintStream**
  - **RandomAccessFile**
  - **File**
  - interface **FileNameFilter**
  - **FileDescriptor**
  - Throwable
    - Exception
      - **IOException**
        - **EOFException**
        - **FileNotFoundException**
        - **InterruptedIOException**
        - **UTFDataFormatException**

C.2.2. Jerarquía de clases definida en el paquete *java.io*

[36] *Las viñetas utilizadas en el cuadro no significan nada sólo representan la jerarquía de clases. Derivando todas de la clase Object*

### 2.4.4 *El paquete java.*util

Este paquete es una miscelánea de clases e interfaces útiles para diversas cosas en programación. Se incluyen, entre otras, las clase de manejo de fechas (*Date, Calendar, TimeZone*), clases de estructuras de datos (*HashTable, Vector, Enumeration*), manejo de internacionalización (*Locale y ResourceBundle*), manejo de eventos (*EventObject y EvenListener*), diccionario (*Dictionary*), números pseudoaleatorios (*Random*), y clases de utilidad diversa tales como *StringTokenizer* utilizada para dividir una cadena en palabras; *BitSet*, utilizada para dividir una matriz de bits, la clase *Observer y Observable* para notificar a otros objetos, llamados observadores cuándo un objeto observado ha cambiado.

A continuación se muestra la jerarquía de clases [18] del paquete *java.util* (C.2.3.[37]). Las clases en negritas pertenecen al paquete *java.util* y las otras al paquete *java.lang*.

- Object
  - interface **Enumeration**
  - **BitSet**
  - **Date**
  - **Dictionary**
    - **Hashtable**
      - **Properties**
  - **Observable**
  - interface **Observer**
  - **Random**
  - **StringTokenizer**
  - **Vector**
    - **Stack**
  - Throwable
    - Exception
      - RunTimeException
        - **EmptyStackException**
        - **NoSuchElementException**

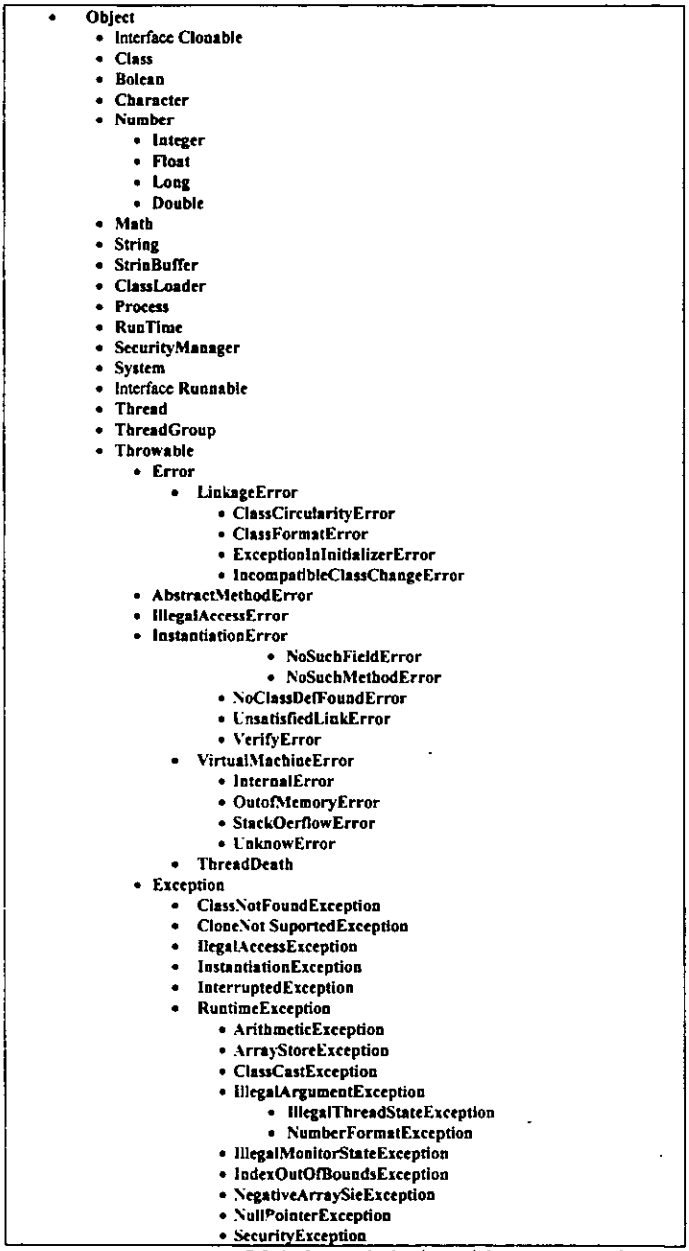
C.2.3. Jerarquía de clases del paquete *java.util*

[37] *Las viñetas sólo indican la jerarquía de clases derivando de la clase Object*

### 2.4.5 *El paquete java.*lang

Este paquete incluye las clases fundamentales para el diseño del lenguaje *Java* propiamente dicho: *Object, Thread, Exception, System, Integer, Float, Math, String*, etc. La clase más importante de este paquete es la clase *Object*, la cual es la raíz de la jerarquía de todas las clases del lenguaje *Java*. Dentro de este paquete también se encuentra la clase *Math*, la cual proporciona las funciones matemáticas de seno, coseno, raiz cuadrada, por mencionar algunas. La clase *String* y *StringBuffer* encargadas de proporcionar operaciones utilizadas en cadena de caracteres. La clase *ClassLoader, Process, Runtime, SecurityManager* y *System* para proporcionar operaciones del sistema que manejan la carga de clases dinámica, creación de procesos externos, políticas de seguridad, fecha y hora del sistema. La clase *throwable* utilizada en el manejo de errores y excepciones. A continuación se muestra la jerarquía de clases [18] del paquete *java.lang* (C.2.4. [38]):

[38] *Las viñetas sólo indican la jerarquía de clases derivando de la clase Object*




C.2.4. Jerarquía de clases del paquete *java.lang*

# Capitulo 3. Arquitectura cliente-servidor y bases de datos.

El presente capítulo trata acerca de la arquitectura cliente-servidor, la cual fue utilizada en la aplicación desarrollada en el presente trabajo de tesis, en donde se tiene, por un lado, la conexión a una máquina servidor con la aplicación y la base de datos y, por el otro lado, una o mas máquinas clientes que solicitan la aplicación al servidor a través de un *navegador* de *Internet*.

También se habla acerca de los conceptos básicos de bases de datos, debido a que la tesis hace un estudio de la interacción del lenguaje *Java* con un manejador de base de datos. El manejador de base de datos es *Oracle*[39]. También se tratan las sentencias básicas para manipular la información contenida dentro de la base de datos.

Este capítulo incluye entre sus temas al *JDBC ("Java Data Base Conectivity")* herramienta que utiliza el *JDK ("Java Developer Kit")* para realizar comunicación con la base de datos y describe la forma de efectuar una conexión a través de un programa desarrollado en código *Java*. Además de explicar la forma en la que se clasifican los manejadores mejor conocidos como "*drivers*", los cuales son los controladores que permiten el acceso a diferentes bases de datos.

La importancia de este capítulo radica en su utilidad para el desarrollo de la aplicación debido a que permite comprender mejor cómo y porqué utilizar una base de datos, además de explicar cómo se realiza la comunicación hacia la base de datos a través de la aplicación que muestra su código en los apéndices del presente trabajo[40].

[39] *Manejador de Base de datos.*
[40] *Los apéndices se encuentran en un disco anexo al presente trabajo.*

## 3.1 Definición de aplicación cliente-servidor

Una aplicación cliente-servidor es aquella en la que cual se tiene una computadora desde la cual se accede a la red (computadora local), en la que existe un programa cliente y una computadora a la que se accede (computadora remota), la cual tiene un programa servidor instalado y un protocolo de comunicación para red, en este caso *TCP/IP*[41] [30].

El cliente es el encargado de la presentación y de gestionar la comunicación con el servidor y proporcionar las herramientas necesarias para trabajar con el mismo. Este es el que hace diferentes peticiones al servidor, y se encarga de transmitir la información requerida por el usuario.

Un ejemplo de arquitectura cliente-servidor es la que utiliza *Internet*. De un lado se tiene al cliente con su *navegador* de *Web*, por ejemplo *Netscape*. Y del otro lado al servidor con el *Servidor de Web*. El cliente hace la solicitud al servidor de algún recurso *HTML* (*"HyperText Markup Language")* y este la satisface al proporcionar la información solicitada por el cliente (como usualmente se diría, muestra la página).

Otro ejemplo puede ser la aplicación desarrollada en el presente trabajo, en el cual se tiene del lado del cliente un *browser* mediante el cual se realizan las diferentes peticiones al servidor. Este último es el encargado de responder a todas estas solicitudes mediante consultas y actualizaciones a la base de datos, así como de validaciones a través de los diferentes programas.

## 3.2 Conceptos generales de bases de datos

La estructura de la base de datos define los diferentes tipos de objetos para el almacenamiento y consistencia de la información y el manejo de la misma se realiza a través de un lenguaje de consultas. El lenguaje adoptado entre los proveedores *DBMS*[42] para la consulta y mantenimiento de la información de la base de datos se le conoce como *SQL* (*"Structured Query Language"*) o lenguaje estructurado de consultas. *SQL* define las instrucciones básicas para la manipulación de datos (*DML*[43]) y para la definición de datos (*DDL*[44]).

Una base de datos es un conjunto de datos utilizados por los sistemas de programación para cumplir con un propósito específico. Está compuesta de archivos de datos, *metadatos*, índices, *metadatos de la aplicación*, al igual que un diccionario de datos, en el cual se describe la estructura de los mismos. A continuación se describirá brevemente cada uno de sus componentes:

---

[41] *Protocolo de Control de Transferencia /Protocolo de Internet*
[42] *"Data Base Manager System", es un conjunto de programas de aplicación usados para acceder, actualizar y manipular los datos.*
[43] *"Data Manipulation Language"*
[44] *"DataDefinition Language"*

a) **Datos**.- Los datos son representados en forma de relaciones que pueden verse como tablas y representan la información.

b) ***Metadatos***.- Los *metadatos* se almacenan en las tablas del sistema. Estos son un conjunto de información que describen el contenido, calidad, condiciones y otras características de la estructura de datos.

c) **Indices**.- Sirven para ordenar y hacer más rápido el acceso a los datos.

d) ***Metadatos* de la aplicación**.- Estos se encargan de guardar la estructura y los formatos de las formas, los reportes, consultas y otros componentes de la aplicación.. Estos datos se procesan mediante herramientas del *DBMS*.

Es importante aclarar que no todos los *DBMS* guardan los componentes de la aplicación ni todos almacenan la estructura de estos componentes como *metadatos* de la aplicación en la base de datos[17].

El objetivo principal de las bases de datos es mantener la consistencia e integridad de los datos, conservar la información y presentarla a través de reportes cada vez que sea necesario.

Su uso es muy importante debido a que permite compartir información evitando redundancia e inconsistencia de la información.

## *3.2.1 Modelos de bases de datos*

Un modelo de datos es la combinación de una colección de objetos de información, reglas de identidad y operadores válidos para obtener información. Existen diferentes modelos, tales como, el modelo relacional, modelo jerárquico y modelo de red.

### 3.2.1.1. Modelo relacional

En este tipo de modelo los datos se representan en forma de tablas, a las cuales se les denomina relaciones, y éstas a su vez están formadas por columnas, también llamadas atributos y por renglones denominados *"tuplas"*.

Al número de *tuplas* también se conoce como "cardinalidad" y al número de atributos se le llama *grado*.

Atributos o grado

| PLA_CLAVE | FAC_CLAVE | CAR_CLAVE | CAR_NOMBRE | CAR_DURACION | CAR_PLAN |
|---|---|---|---|---|---|
| 1 | 1 | 1 | Ingeniería | 10 | Semestral |
| 2 | 2 | 2 | Contaduría | 4 | Anual |

Tuplas o cardinalidad

Fig. 3.1.Tabla de grado 6.

En la figura 3.1, se muestra la tabla CARRERA de la base de datos de la aplicación[45] realizada durante este trabajo de tesis. En ella podemos observar cada una de las columnas que son PLA_CLAVE, FAC_CLAVE, CAR_CLAVE, CAR_NOMBRE, CAR_DURACION y CAR_PLAN y los renglones que son cada uno de los datos que se introducen a la tabla. También se muestra gráficamente cuales son las *tuplas* y cuales los atributos y podemos decir que el grado de la tabla es 6 debido a que tiene 6 columnas.

Es importante mencionar que la tarea del modelo relacional es mantener la estructura, integridad y manipulación de los datos.

[45] *La estructura completa de la base de datos del Sistema de Evaluación se muestra en el siguiente capítulo.*

### 3.2.2.1. Tablas

Los datos generalmente se almacenan de tal forma que se pueda clasificar su información; así una clasificación puede ser la de "alumnos", en donde para cada uno de los alumnos se deberá mantener información como nombre, dirección, clave, etc.

Esta clasificación y almacenamiento se realiza dentro de un objeto denominado tabla y a los atributos de ella se les denomina columnas. Una tabla es el objeto de almacenamiento principal dentro de una base de datos, así por ejemplo, se puede definir la tabla *ALUMNOS*, cuyas columnas son *ALU_CLAVE*, *ALU_NOMBRE*, *ALU_CALLEYNUMERO*, etc, dentro de la cual estarán contenidos los datos relativos a cada uno de los alumnos.

Las tablas pueden ser de dos tipos, dependientes o independientes, cuya diferencia reside en que las tablas independientes contienen datos y su existencia no se ve influenciada por ninguna otra información, mientras que en las tablas dependientes (tablas hijas) la información almacenada depende de la existencia previa de la tabla de la cual depende (tablas padres) [21(pp.14)].

### 3.2.2.2. Llaves primarias

Una llave es un identificador único, el cual es una combinación de atributos ó relaciones, ó ambos, que sirven para distinguir ocurrencias[46] de cualquier entidad [28].

Una llave primaria (*"Primary key"*) es un objeto que permite hacer único a un registro dentro de una tabla. Las llaves primarias pueden ser simples o compuestas.

Una llave primaria simple se encuentra dentro de una tabla independiente y se refiere a una sola columna o grupo de columnas (aunque generalmente se opta por ser solo una columna por simplicidad), en donde el valor de esa columna hace referencia a un registro único dentro de la tabla, por ejemplo la clave de un alumno.

Una llave compuesta se encuentra dentro de una tabla dependiente y se refiere a una o mas columnas de la tabla hija y una o mas columnas de la tabla padre.

### 3.2.2.3. Llaves foráneas

Una llave foránea (*"Foreign Key"*) es un objeto que indica que a un registro de una tabla se le asocia un registro único dentro de otra tabla. La manera de hacerse es indicando que una columna es el correspondiente de otra columna dentro de la otra tabla (generalmente su llave primaria). Esta es la forma de enlazar datos entre diferentes tablas

[46] *Registro*

para evitar duplicidad en la información, por ejemplo, si a una lista de un grupo hay que asociarle a cada uno de los alumnos inscritos en él, no será necesario escribir todos los datos del alumno dentro de la tabla *GRUPO*, sino que en lugar de ello solo se guardan las claves de los alumnos y se dice que la columna *ALU_CLAVE* de la tabla *GRUPO* es una llave foránea, asociada a la columna del mismo nombre en la tabla *ALUMNO*.

### 3.2.2.4. Índices

Un índice es un objeto que se utiliza para hacer el manejo de la información más eficiente. Para esto se define una o mas columnas dentro de una tabla cuya información se requiera manipular con rapidez y lo que se tiene es una pseudo tabla conteniendo únicamente a las columnas indexadas y se almacenan de forma ordenada y una columna que hace referencia al registro original de la tabla, así cuando por ejemplo, se consulta una tabla por una columna indexada, esta consulta se hace sobre la pseudo tabla índice, buscando mediante algoritmos de búsqueda mas eficientes dentro de un conjunto de datos ordenados y de menor cardinalidad. Esto trae consigo que las consultas y actualizaciones se efectúen con mayor rapidez pero cuando se trata de tablas con alto número de transacciones genera una doble tarea, ya que por un lado se actualiza la tabla original y por otro se hace el mantenimiento sobre la pseudo tabla indexada, por ello se debe tener cuidado en escoger los índices apropiados en el diseño de la base de datos. En general las llaves primarias tienen un índice implícito.

### 3.2.2.5. Tipos de columnas

Los tipos de datos (*"Data Type"*) indican el tipo de información que puede almacenar una columna. Los tipos de datos mas comúnmente utilizados son:

- *NUMBER* Tipos de datos numéricos
- *DATE* Tipos de datos con formato de fecha y hora
- *BOOLEAN* Tipos de datos binarios
- *INTEGER* Tipos de datos numéricos enteros
- *CHAR* Tipos de datos alfanuméricos de longitud fija
- *VARCHAR* Tipos de datos alfanuméricos de longitud variable

### 3.2.2.6. Procedimientos almacenados

Los procedimientos almacenados (*"Store Procedure"*) son código escrito en el lenguaje de la base de datos que se puede ejecutar dentro de una sentencia *SQL* y son muy similares a las funciones o procedimientos de un lenguaje de programación, los cuales aceptan parámetros y pueden devolver algún valor o simplemente manipular la información.

### 3.2.2.7. Disparadores.

Los disparadores (*"Triggers"*) son objetos que contienen código de procedimientos almacenados que se ejecuta cuando sucede un evento dentro de la base de datos cuando un registro se inserta (*INSERT*), se modifica (*UPDATE*) o se elimina (*DELETE*) dentro de una tabla. Los disparadores se pueden definir del tipo antes (*BEFORE*) o después (*AFTER*), en los cuales el código asociado al disparador se ejecutará antes o después del evento especificado.

### 3.2.2.8. Restricciones.

Las restricciones (*"Constraints"*) son objetos que se definen de una manera muy simple para efectuar una operación sencilla dentro de la base de datos cuando un registro se inserta, se modifica o se elimina. Estas verificaciones son muy particulares en cada *DBMS*. Un ejemplo de ellas son las llaves foráneas, los índices únicos, las columnas que no permiten valores nulos, verificaciones de rangos, etc.

### 3.2.2.9. Catálogo del sistema.

El catálogo (*"System tables"*) del sistema son todas las tablas que almacenan la información propia de la base de datos y sus objetos. Sirve para mantener la integridad de la base de datos y para guardar su definición como nombres de las tablas, tipos de columnas, espacio libre, nombre de los archivos en donde se encuentra la información, usuarios, tipos de índices. Indica también a quién pertenecen cada uno de los objetos, etc. El manejo del catálogo del sistema se realiza de forma automática por el manejador de base de datos.

## *3.2.3 Comandos SQL – DDL*

El lenguaje de definición de datos (*DDL*[47]) está definido para crear, alterar o eliminar objetos dentro de la estructura de la base de datos [28].

### 3.2.3.1. CREATE TABLE

La cláusula *CREATE TABLE* sirve para definir nuevas tablas para el almacenamiento de la información. Su sintaxis es la siguiente:

```
CREATE TABLE nombre_tabla
  (nombre_columna tipo_dato, ...,
  nombre_columna tipo_dato)
```

[47] *Data Definition Language*

Por ejemplo para la tabla *ALUMNOS* se empleó la siguiente sintaxis en *ORACLE* para su creación.

```
CREATE TABLE ALUMNOS
  (ALU_CLAVE            NUMBER(5)
   ALU_NOMBRE           VARCHAR2(30)
   ALU_APELLIDO_P       VARCHAR2(25)
   ALU_APELLIDO_M       VARCHAR2(25)
   ALU_LOGIN            VARCHAR2(9)
   ALU_PASSWORD         VARCHAR2(8)
   ALU_CALLEYNUMERO     VARCHAR2(50)
   ALU_COLONIA          VARCHAR2(30)
   ALU_TELEFONO         NUMBER(7)
   ALU_EMAIL            VARCHAR2(60)
   ESC_CLAVE            NUMBER(2)
   DEL_CLAVE            NUMBER(2)
   ENT_CLAVE            NUMBER(2));
```

### 3.2.3.2. ALTER TABLE

La cláusula *ALTER TABLE* proporciona un mecanismo para modificar las tablas una vez creadas [28]. Es posible añadir columnas, restricciones o modificar un tipo de dato pero no se pueden borrar columnas ni modificar tipos de datos si las columnas contienen información en algunos casos, algunas de sus sintaxis mas usuales son:

```
ALTER TABLE nombre_tabla ADD (nombre_columna tipo_dato, ... , nombre_columna tipo_dato);
ALTER TABLE nombre_tabla MODIFY (nombre_columna tipo_dato, ... , nombre_columna tipo_dato;
ALTER TABLE nombre_tabla ADD CONSTRAINT (nombre_restriccion tipo_resticcion, ..., nombre_restriccion
tipo_resticcion;
```

Por ejemplo, para definir la llave primaria de la tabla *ALUMNOS* se utilizó el siguiente comando:

```
ALTER TABLE ALUMNOS ADD PRIMARY KEY (ALU_CLAVE);
```

### 3.2.3.3. DROP TABLE

La cláusula *DROP TABLE* indica al *DBMS* que elimine una tabla [28]. Su sintaxis es la siguiente:

```
DROP TABLE nombre_tabla;
```

### 3.2.3.4. CREATE INDEX

La cláusula *CREATE INDEX* crea un índice sobre una tabla para una o más columnas [28]. Su sintaxis es:

```
CREATE [UNIQUE] INDEX nombre_indice ON nombre_tabla (nombre_olumna, ... , nombre_columna);
```

La cláusula *UNIQUE* es opcional e indica que las columnas indexadas deben ser únicas. Para un índice en la columna *ALU_CLAVE* de la tabla *ALUMNOS* se utilizó la siguiente instrucción:

```
CREATE UNIQUE INDEX PK_ALUMNOS ON ALUMNOS (ALU_CLAVE);
```

### 3.2.3.5. DROP INDEX

Cuando un índice ya no es necesario o simplemente se desea eliminarlo se utiliza la cláusula *DROP INDEX* [28]. Su sintaxis es:

```
DROP INDEX nombre_indice;
```

## *3.2.4 Comandos SQL – DML*

El lenguaje de manipulación de datos (*DML*[48]) se utiliza para consultar, insertar, actualizar o borrar información en las tablas de la base de datos.

### 3.2.4.1. INSERT

Para ingresar información nueva dentro de la base de datos se utiliza la sentencia *INSERT* [28] de la siguiente forma.

```
INSERT INTO nombre_tabla
  (nombre_columna, ... , nombre_columna)
VALUES
  (dato, ... ,dato);
```

Por ejemplo si se quiere añadir la información de un alumno en la tabla *ALUMNOS* se puede utilizar la siguiente instrucción.

```
INSERT INTO ALUMNOS
  (ALU_CLAVE, ALU_NOMBRE, ALU_APELLIDO_P, ALU_APELLIDO_M, ALU_LOGIN)
VALUES  (1, 'ERICKA', 'CERÓN', 'VARGAS', 'erickacv');
```

### 3.2.4.2. SELECT

La cláusula *SELECT* se utiliza para consultar la información contenida dentro de una tabla [28], la sintaxis empleada es:

```
SELECT [alias_tabla.nombre_columna alias_columna, ..., alias_tabla.nombre_columna alias_columna | *]
FROM nombre_tabla alias_tabla, ..., nombre_tabla alias_tabla
```

[48] *Data Manipulation Language*

```
WHERE [condicion | join]
GROUP BY alias_tabla.nombre_columna, ..., alias_tabla.nombre_columna
HAVING condicion
ORDER BY alias_tabla.nombre_columna, ..., alias_tabla.nombre_columna;
```

Por ejemplo se quiere seleccionar sólo el registro de la tabla *ALUMNOS* que se insertó en la sección anterior.

```
SELECT *
FROM ALUMNOS
WHERE ALU_CLAVE = 1;
```

El asterisco (*) indica que se seleccionen todas las columnas. Nótese que debido a que *ALU_CLAVE* es la llave primaria y es única basta con indicar en la condición *WHERE* que seleccione todas aquellas que sean igual a "1" y el resultado será:

```
ALU_CLAVE ALU_NOMBRE ALU_APELLIDO_P ALU_APELLIDO_M ALU_LOGIN
--------- ---------- -------------- -------------- ------------
1         'ERICKA'   'CERÓN'        'VARGAS'       'ericka'

1 row selected
```

### 3.2.4.3. UPDATE

Una vez que la información se encuentra almacenada dentro de una tabla, es necesario regularmente que ésta sufra modificaciones las cuales se efectúan mediante la cláusula *UPDATE* [28]. Su sintaxis es:

```
UPDATE nombre_tabla
SET nombre_columna = dato
WHERE condicion;
```

Por ejemplo si el login del alumno se requiere cambiar, entonces se puede utilizar la siguiente instrucción:

```
UPDATE ALUMNOS
SET ALU_LOGIN = 'ericka1'
WHERE ALU_CLAVE = 1;
```

Aquí nuevamente se hace uso de la llave primaria para identificar el registro correspondiente. Es importante notar que si se omite la cláusula *WHERE* se hará una actualización sobre todos los registros de la tabla.

### 3.2.4.4. DELETE

La cláusula *DELETE* hace un borrado de la información que se indique dentro de una tabla [28], su sintaxis es:

```
DELETE [FROM] nombre_tabla
WHERE condicion;
```

Tomemos como ejemplo borrar el registro que hemos estado usando, la sintaxis sería entonces:

```
DELETE FROM ALUMNOS
WHERE ALU_CLAVE = 1;
```

Nuevamente es importante notar que si se omite la cláusula *WHERE* se borrarán todos los registros de la tabla *ALUMNOS*.

## 3.3 Manejador de base de datos *Oracle.*

El *DBMS* (*"Data Base Management System"*), utilizado en la elaboración del *Sistema de Evaluación* [49] es el *Oracle 8.0.1*, el cual ofrece una base de datos robusta y hace un manejo de datos conservando su integridad y su estructura.

*Oracle 8.0.1* es una de las mejores bases de datos que existen actualmente en el mercado proporcionando una amplia gama de productos para conectar la base de datos y las aplicaciones de red. Entre estos productos se encuentran los protocolos de adaptación de red a través de *Net 8* y los *Drivers JDBC* para interactuar con aplicaciones elaboradas en lenguaje *Java*. [29]

## *3.4* El paquete *java.sql*

El paquete *java.sql* contiene las interfaces (*API's*) necesarias para acceder y procesar datos utilizando el lenguaje de programación *Java*, gracias a este paquete es posible realizar las siguientes tareas:

- Establecer una conexión a la base de datos, mediante las clases *DriverManager*, *DriverPropertyInfo* y de las interfaces *Driver y Coneccion.*
- Enviar sentencias *SQL*, mediante la interfaz *Statement, PreparedStatement* ó *CallableStatement.*
- Procesar y actualizar el resultado de un query, mediante la interfaz *ResultSet.*
- Proporcionar información acerca de las columnas de un objeto *ResultSet*, a través de las interfaces *DatabaseMetaData* y *ResultSetMetaData.*
- Lanzar excepciones, tales como *SQLException*, lanzada por la mayoría de los métodos cuando no pueden acceder a los datos. También existen algunas otras excepciones tales como: *SQLWarning*, que indica precaución, *DataTruncation*, lanzada para indicar que los datos de la tabla podrían estar truncados, *BatchUpdateException*, para indicar que no todos los comandos ejecutados en batch se realizaron exitosamente.
- Proporcionar seguridad, mediante la interfaz *SQLPermission.*

[49] *Aplicación desarrollada en el presente trabajo.*

## 3.5 Herramientas de *Java* para el manejo de bases de datos ( *JDBC*)

El lenguaje *Java* es un excelente candidato para el desarrollo de aplicaciones de bases de datos ya que es un lenguaje robusto, fácil de comprender, seguro y descargable automáticamente por la red. Este lenguaje tiene una herramienta para el manejo de bases de datos que permite implementar fácilmente medios de acceso para las aplicaciones *Java*, llamada *JDBC* [20].

El *JDBC* es un conjunto de clases e interfaces escritas en *Java* que permiten a otros programas en *Java* establecer una conexión a una base de datos, ejecutar sentencias *SQL* y procesar los resultados. El *JDBC* se incluye con el *JDK ("Kit de Desarrollo de Java")* a partir de la **versión 1.1**, dentro del paquete *java.sql*.

Para poder utilizar el *JDBC* es necesario contar con controladores mejor conocidos como *drivers* que permitan el acceso a las diferentes bases de datos. Estos son objetos que definen cómo se ejecutan las instrucciones para una base de datos en particular, es por ello que cada base de datos ocupa un controlador diferente.

A continuación se muestra en la tabla T3.3. una lista de clases e interfaces más importantes que *JDBC* [21] ofrece. Estas clases se encuentran en el paquete ***java.sql***

| Clase/Interface | Descripción |
|---|---|
| *Driver* | Permite conectarse a una base de datos. |
| *DriverManager* | Permite gestionar todos los *drivers* instalados en el sistema y crear nuevas conexiones. |
| *DriverPropertyInfo* | Proporciona diversa información acerca de un *driver*. |
| *Connection* | Representa una conexión con una base de datos. |
| *DatabaseMetadata* | Utilizada para obtener información acerca de una base de datos, ya sean tablas, índices, tipos de datos, etc. |
| *Statement* | Permite ejecutar sentencias *SQL* sin parámetros. |
| *PreparedStatement* | Permite ejecutar un procedimiento precompilado, ya que una sentencia *SQL* es precompilada y guardada en un objeto *PrepareStatement*. |
| *CallableStatement* | Permite ejecutar sentencias *SQL* con parámetros de entrada y salida, típicamente procedimientos almacenados. |
| *ResultSet* | Contiene las filas o registros obtenidos al ejecutar un *SELECT*. |
| *ResultSetMetadata* | Permite obtener información sobre un *ResultSet*. |

Tabla T3.3. Clases e interfaces más importantes del *JDBC*.

## 3.6 La interfaz *Driver*

La interfaz *Driver* proporciona un conjunto de métodos para obtener información acerca del controlador, y proporciona el método *connect*() para realizar la conexión a la base de datos.

Algunos de sus métodos son:

***AcceptsURL***.- Indica si el controlador puede o no conectarse al *URL* especificado mediante un valor *booleano*.

***Connect*** .- Realiza la conexión a la base de datos y devuelve un objeto *Connection*.

***GetPropertyInfo***.- Proporciona una lista de las propiedades requeridas por el controlador para crear una conexión a la base de datos.

## 3.7 *Driver* Manager

Esta interfaz es muy útil para administrar objetos, ya que permite conocer las propiedades del *JDBC.drivers* para especificar diferentes controladores *JDBC* cuando se tienen distintas aplicaciones [21 (pp.51)].

Algunos de los métodos del objeto *DriverManager* son:

***DeregisterDriver***().- Mediante este método se pueden eliminar los controladores de la lista de drivers.

***GetConnection***().- Realiza la conexión con la base de datos y devueive un objeto *Connection* .

***GetDriver***().- Localiza un controlador que conectará al *URL* especificado.

***GetsDrivers***().- Puede obtener todos los controladores registrados actualmente.

***GetLoginTimeout***().- Proporciona el límite de tiempo que el controlador esperará para conectarse a una base de datos.

***GetLogStream***().- Permite obtener el flujo de registro por el que se enviará la información y los errores generados por el *driver*. Por ejemplo, el flujo de registro puede ser *System.out*.

***Println***().- Permite mostrar información al usuario relacionada con la base de datos.

***RegisterDriver***().- Permite registrar un controlador específico.

***SetLoginTimeout***().- Especifica el tiempo que esperará el controlador para obtener la conexión a la base de datos.

***SetLogStream***().- Indica hacia dónde se enviarán los errores e información generada por los controladores.

## 3.8 Como hacer una conexión a la base de datos

Para comunicarse una base de datos mediante el *JDBC* primero se debe registrar el controlador a utilizar (en el presente trabajo se utilizó el *JDBC thin driver de Oracle*), de la siguiente manera:

```
DriverManager.registerDriver(new oracle.JDBC.driver.OracleDriver());
```

Después se realiza la conexión a la base de datos:

```
Connection.c=DriverManager.getConnection("JDBC:oracle:thin:@132.248.173.15:1521:evalua","evalua","tesis");
```

La clase *DriverManager* gestiona los controladores registrados en el sistema al llamar a *getConnection*, recorre la lista de controladores cargados hasta encontrar uno que sea capaz de gestionar la petición especificada por el *URL*, que en este caso es :

```
"JDBC:oracle:thin:@132.248.173.15:1521:evalua"
```

El parámetro *evalua* corresponde al nombre de usuario y el de *tesis* al *password* de la base de datos. A continuación se muestra un fragmento de programa *Protocolo.java*, en donde se realiza la conexión a la base de datos [50], en el primer bloque *try* se puede observar el registro del controlador mientras que en el segundo bloque *try* se muestra cómo se abre una conexión a la base de datos y la captura de la excepción *SQLException*, activada por el *DriverManager.getConnection()*.

```
...
try{
   DriverManager.registerDriver(new oracle.JDBC.driver.OracleDriver());
 } // Fin del try
 catch(Exception egral){
   System.out.println("Error al cargar el driver.");
 } // Fin del catch

 try {
   c = DriverManager.getConnection ("JDBC:oracle:thin:@132.248.173.15:1521:evalua","evalua","tesis");
 } // Fin del try
 catch(SQLException sqle){
   System.out.println("Error : " +sqle);
 } // Fin del catch
...
```

Una vez realizada la conexión a la base de datos, se pueden realizar las consultas y modificaciones a los datos.

Finalmente cuando se termina de utilizar la conexión a la base de datos es importante cerrarla mediante la sentencia *close*. Por ejemplo, para cerrar una conexión *c* :

```
c.close();
```

[50] *El código completo puede ser consultado del disco anexo en la sección de apéndices. En la parte del servidor*

## 3.9 Consultas a la base de datos

Las consultas de la base de datos de la aplicación realizada en este trabajo de tesis, se realizan dentro de un programa llamado *Protocolo.java*[51].

Para realizar consultas a la base de datos se debe obtener primero un objeto *Connection* y pedir un objeto *Statement* que tiene asociado.

## 3.10 Tipos de controladores *JDBC* (*driver JDBC*)

Los controladores del *JDBC* se clasifican en cuatro categorías . A continuación se describe cada una de ellas [5]:

### *3.10.1 Tipo 1. Puente entre JDBC y ODBC (JDBC – ODBC bridge driver)*

Este tipo de controlador es ideal para utilizar en aplicaciones de tres capas, ya que procesa las llamadas *JDBC* a través del *ODBC*[52]. En la mayoría de los casos es necesario cargar código en la máquina cliente.

### *3.10.2 Tipo 2. Acceso en Java las bibliotecas nativas.* (Native-API/partly-*Java* driver)

Este tipo de controlador necesita que el código binario sea cargado en la máquina cliente y lo que hace es convertir las llamadas del *JDBC* en llamadas nativas de la base de datos para comunicarse directamente con la base de datos del servidor.

### *3.10.3 Tipo 3. Controlador todo Java.* (Net-protocol/all-*Java* driver)

Este controlador hace uso de un protocolo propietario, definido por el proveedor del controlador entre un cliente *Java* y la base de datos. El controlador es el encargado de la traducción de llamadas del *JDBC* a las llamadas del *DBMS* independientemente del protocolo de red, el cual es traducido al protocolo *DBMS* por el servidor. El protocolo utilizado depende del vendedor.

[51] *Su código se encuentra en un disco anexo a este trabajo.*
[52] *"Open Data Base Conectivity", estándar ó interface de programación utilizada para acceder a una base de datos.*

### *3.10.4 Tipo 4 Protocolo nativo*

Convierte las llamadas del *JDBC* a llamadas del protocolo de red utilizado por el *DBMS*, permitiendo realizar llamadas directas desde la máquina cliente hacia el servidor del *DBMS*. Existe una gran variedad de controladores para el *JDBC*. En el desarrollo de la aplicación realizada en el presente trabajo, se empleó el controlador de *Oracle* llamado *JDBC Thin Driver*. A continuación se proporciona una lista (T.3.4.) de algunos de los distribuidores y su controlador correspondiente [7]:

| Distribuidor | Controlador (driver) |
|---|---|
| *Agave Software Design* | *JDBC Net Server* |
| *Asgard Software* | *OpenV A for Java* |
| *Borland* | *InterClient* |
| *Connect Software* | *Conenct* |
| *DataRamp* | *Client for Java* |
| *IBM* | *DB2 Client Support* |
| *Imaginary* | *MSQL-JDBC Driver* |
| *InterSoft* | *Essentia-JDBC* |
| *Intersolv* | *DataDirect* |
| *JavaSoft* | *JDBC-ODBC Bridge* |
| *OpenLink* | *JDBC Drivers* |
| *SAS* | *SHARE*NET* |
| *SCO* | *SQL-Retriever* |
| *TormCloud Development* | *WebDBC 3.0 Enterprise* |
| *Symantec* | *DbAnywhere* |
| *Visigenic* | *VisiChannel for Java* |
| *WebLogic* | *JDBCKona, JDBCKonaT3* |
| *Oracle* | *JDBC Thin driver*<br>*ODBC Driver* |

T.3.4. Distribuidores y controlador correspondiente.

## 3.11 *Driver* JBDC de *Oracle*

Como ya se mencionó anteriormente *JDBC* es un conjunto de clases e interfaces escritas en *Java* que permiten enviar sentencias *SQL* a un sistema manejador de base de datos.

*Oracle* proporciona dos categorías de controladores *JDBC* [29]:

***JDBC Thin driver***

El *JDBC Thin* es un controlador de tipo 4 que utiliza *sockets* para realizar una conexión directa con la base de datos *Oracle*. Este controlador es independiente de la plataforma y posee su propia implementación de *TCP/IP*. Esta hecho 100% en *Java* y es ideal para el desarrollo de *applets*. Es importante mencionar que este es el controlador utilizado en la aplicación desarrollada a lo largo de esta tesis y se encuentra en el CD de instalación del *Oracle* Versión 8; además no necesita instalarse nada del lado del cliente, basta con instalarlo del lado del servidor y levantar el *TNS Listener*[53] para escuchar sobre los *sockets* de *TCP/IP*.

***JDBC* OCI (*Oracle Call Interface*)**

Es un controlador de tipo 2 [16] que utiliza métodos nativos en *Java* para hacer llamada a una serie de librerías en C contenidas en la librería del *OCI*. El uso de métodos nativos hace que este tipo de controlador sea específico de la plataforma. *Oracle* proporciona el controlador *OCI* [29] para el sistema operativo *Solaris* y sistema operativo *Windows*. La versión de *Windows* trabaja tanto en *Windows 95/98* como *Windows NT*.

El que sea específico para una plataforma hace que no sea un controlador adecuado cuando se hace uso de *applets*, ya que estos se descargan de la red para ser ejecutados en una plataforma desconocida y además este tipo de controlador requiere instalación de software del lado del cliente (*SQL*Net, V 2.3.* o superior). Sin embargo este controlador es ideal para trabajar con aplicaciones en *Java*.

[53] *Proceso que escucha las conexiones de entrada y servicios enviándolos o redireccionandolos al servidor correspondiente.*

# Capitulo 4. Desarrollo de la aplicación.

Debido a que el tiempo de desarrollo de sistemas se considera como un punto importante, algunos estudios han arrojado estadísticas que muestran que tres cuartas partes de los desarrollos han superado las expectativas en los tiempos estimados; esto claramente se ve reflejado de primera mano en el costo del proyecto, pero también significa una frustración para los desarrolladores quienes dedican su esfuerzo con horas extraordinarias de trabajo y en muchas ocasiones parte de este trabajo se ve desechado debido a un mal análisis [23]. En este capítulo se presenta formalmente la aplicación realizada en este trabajo de tesis y la manera de cómo hacer un buen análisis para el desarrollo de la misma. La metodología rápida para desarrollo de aplicaciones o metodología *RAD*[54] y del lenguaje *UML*[55] para el modelado sistemas son dos de las herramientas más usuales que se usan para controlar de forma eficiente los proyectos para desarrollo de sistemas informáticos.

## 4.1 Presentación de la aplicación

Durante la realización del presente trabajo de tesis se desarrollo un aplicación denominada "*Sistema de Evaluación*", la cual conjunta la programación en *Java* y su interacción con una base de datos *Oracle*. Este sistema está diseñado para trabajar sobre *Internet*, ya que hoy en día las aplicaciones necesitan ser interactivas y compartidas a través de la red.

Esta aplicación se analiza y define para una sistema que sirve de apoyo en las actividades de profesores y alumnos en la elaboración de exámenes y en el manejo de la información de los mismos. Es importante mencionar que solo es un prototipo con la finalidad de comprender mejor el estudio del lenguaje *Java* y su interacción con la base de datos; así como también dar a conocer de forma real algunos problemas a los cuales se enfrentan los desarrolladores de sistemas y comprender mejor las ventajas y desventajas del lenguaje *Java*.

El sistema realizado en el presente trabajo, no está conceptualizado para sustituir al profesor, sino más bien, para apoyarlo en sus actividades académicas al proporcionarle la facilidad de introducir sus exámenes con sus respectivas respuestas y evaluar a sus alumnos de un grupo específico. El sistema está formado por cuatro módulos: Administrador, profesor, alumno y ayuda en línea, los cuales se ven a detalle en el capítulo V <<Análisis de resultados>>

[54] "*Rapid Aplication Development*"
[55] "*Unified Modeling Language*"

## 4.2 Metodología *RAD*.

En todo desarrollo de una aplicación de software, los analistas se enfrentan frecuentemente a problemas que en general son comunes a la mayoría de aplicaciones. El desarrollo de la aplicación sin un buen análisis lleva de forma muy regular a repetir muchas líneas de código en el mejor de los casos y en el peor a repetir enteramente el sistema. Debido a estas circunstancias los analistas enfrentan una importante tarea previa al desarrollo de la aplicación para evitar el desperdicio de trabajo. A la forma de controlar el análisis y desarrollo de la aplicación se le conoce como metodología. Una metodología es un conjunto de métodos con los cuales se aborda un problema de forma lógica y sistemática

Las metodologías para el desarrollo e implantación de aplicaciones tienen como esenciales los siguientes puntos:

- Análisis
- Desarrollo
- Prueba de sistemas
- Correcciones
- Implantación

Bajo ciertas variantes han surgido diferentes metodologías que adicionan, eliminan, invierten o combinan en diferentes ciclos los pasos anteriores. La metodología adoptada en este desarrollo es una variante conocida como *Metodología Rápida para Desarrollo de Aplicaciones* o metodología *RAD* [23]. En la Metodología *RAD* se hacen dos o más ciclos dependiendo de la complejidad del sistema hasta tener una versión operante.

En la figura 4.1 se puede observar que se tiene un primer ciclo con los pasos de Análisis, Desarrollo, Pruebas y Correcciones. Se pueden adicionar ciclos dependiendo de la complejidad del sistema y de su nivel de abstracción, lo cual se hace agregando un nuevo ciclo en paralelo al anterior, inmediatamente después de la fase de Pruebas en dónde el nuevo nivel de abstracción se hace más detallado. Se continúan agregando tantos ciclos como se requiera evitando que sean demasiados como para perderse en las abstracciones y que sean tan pocos como para perder el significado de cada uno de los ciclos. En general para un sistema de tamaño mediano, con tres ciclos es más que suficiente [26].

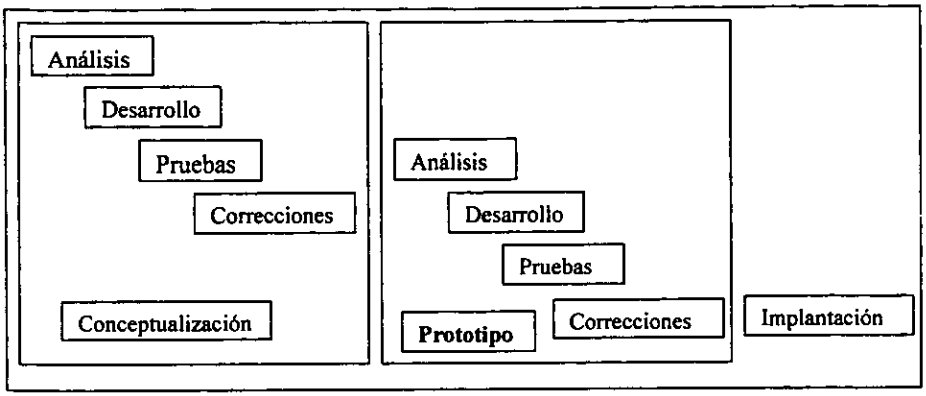



Figura 4.1. Desarrollo *RAD* de dos ciclos.

A los tres ciclos que se utilizan en este desarrollo lo llamaremos según el nivel de abstracción con los siguientes nombres:

- Conceptualización
- Prototipo (o versión beta) y
- Versión Final

## *4.2.1 Conceptualización*

La conceptualización del sistema es el primer ciclo de la metodología, en la cual se plasman los requerimientos del sistema sin que con esto se especifique la forma en que se efectúe. Para este ciclo, primero se identifican los requerimientos del sistema utilizando los diagramas de casos de uso del lenguaje *UML*, en dónde en cada caso de uso se pone de manifiesto de forma muy general los requerimientos del sistema en el primer análisis. El desarrollo es la documentación de los casos de uso. Y las pruebas y correcciones se efectúan haciendo una extensión de estos diagramas en los correspondientes diagramas de colaboración y de esta forma identificar si estas abstracciones son correctas.

Hasta aquí no ha intervenido para nada la programación del sistema y por consiguiente cualquier cambio en la conceptualización de los requerimientos únicamente se hace en los diagramas de casos de uso y de colaboración.

También se incluye una parte de los diagramas de componentes para modelar la arquitectura cliente – servidor.

### 4.2.2 *Prototipo (o versión beta)*

En el segundo ciclo se modela el comportamiento de la aplicación, la cual comienza a tomar sentido en términos de la tecnología empleada como el lenguaje *Java* y el manejador de base de datos *Oracle*.

En este ciclo, en la parte del análisis; se pone de manifiesto la estructura de datos y de los programas que los manejan en su forma más simple. Para lo anterior se emplean los diagramas de despliegue, de clases, de secuencia y de actividades

Como tal, el aplicativo se comienza a desarrollar generando a partir del análisis la interfaz del usuario, las estructuras de la base de datos y los programas principales. El empleo del modelado de *UML* se hace muy importante en este ciclo ya que una buena definición en el modelo representa un buen desarrollo y por consiguiente pocos cambios en las definiciones de los requisitos.

Una vez programados los requerimientos que trabajan sobre las estructuras de datos se tiene una versión beta de la aplicación. Una versión beta es aquella en la cual se puede poner a consideración de los usuarios para que comprueben que los requerimientos se han plasmado tal y como se hizo en el diseño y también para que verifiquen la funcionalidad de la aplicación y la consistencia de los datos.

Cuando se han detectado los elementos faltantes se procede a corregirlos pero no inmediatamente, sino que se espera hasta estar en el ciclo siguiente para que las correcciones no interfieran con el desarrollo de la aplicación final o que se omitan en el análisis del tercer ciclo.

### 4.2.3 *Versión Final*

La versión final de la aplicación se comienza a analizar después de la implementación de la versión beta y considera los requerimientos adicionales y las validaciones finales de la aplicación. Esto quiere decir que, si hay (normalmente siempre hay) nuevos requerimientos se procede a modelarlos en este último ciclo. También se extienden los diagramas adicionando restricciones, ligas entre los componentes, etc.

Una vez analizado, se comienza con el desarrollo y se procede a modificar los programas e interfaces según los nuevos requerimientos y validaciones.

Se hacen nuevamente pruebas más orientadas considerando las validaciones y casos particulares. Si por alguna razón existen correcciones, se procede a desarrollarlas y se implantan para la versión final.

Los diagramas obtenidos durante el análisis de los diferentes ciclos, sirven como documentación técnica del sistema. En el siguiente apartado se muestra cómo se elaboraron dichos diagramas en *UML*.

## 4.3 *UML*

Debido a que el desarrollo de software se hace cada vez más costoso y complicado se han creado lenguajes que permiten hacer más sencilla esta tarea y por tanto menos costoso. De entre las tecnologías de software destacan los lenguajes orientados a objetos como el caso del lenguaje *Java* que se estudia en esta tesis, por tanto es conveniente entonces emplear una metodología de análisis y de modelado acorde a esto. *UML*[56] o *Lenguaje de Modelado Unificado* es un lenguaje que ha surgido, como su nombre lo dice, de la unificación de otros métodos (*Método de Booch, OMT*[57] [27] y *OOSE*[58]) que se han propuesto para el modelado de objetos. Cabe aclarar que *UML* [22] no sólo sirve para modelado de software orientado a objetos. En general se hace uso de *UML* para obtener la abstracción de los requerimientos, construir y documentar el sistema de forma precisa y completa.

Los diagramas aquí presentados son el resultado final del análisis realizado durante cada uno de los ciclos del proyecto. Los diagramas no se presentan como han evolucionado, sino más bien como quedaron al final del desarrollo de la presente tesis.

Tampoco se dan muchos detalles de lo que es *UML* ni de cómo se implementaron los diagramas. Si se quiere ahondar en los detalles de *UML* refiérase a las notas de la bibliografía propuesta al final de la tesis [22, 24, 25 y 27].

[56] *"Unified Modeling Language"*
[57] *"Object Modeling Technique"*
[58] *"Object Oriented Software Engeneering"*

## 4.4 Diagramas estructurales

Los diagramas estructurales definen los aspectos estáticos de aquello que se va a modelar; por decirlo de alguna forma, son el esqueleto del sistema. Los diagramas estructurales utilizados en este trabajo son:

- Diagramas de clases
- Diagramas de componentes
- Diagramas de despliegue

### *4.4.1 Diagramas de clases*

Las clases son los elementos más importantes para el modelado de objetos. Una clase es una descripción de objetos que comparten los mismos atributos, operaciones y responsabilidades. Los diagramas de clases se emplean para el modelado de las clases escritas en el lenguaje *JAVA* y de la estructura de la base de datos.

Los diagramas de clase elaborados para el desarrollo de la aplicación realizada en este trabajo de tesis, se subdividen para la comprensión de su análisis en:

- Administración Alumnos
- Administración Profesores
- Administración Materias
- Administración Grupos
- Administración Planteles
- Administración Carreras

En donde en cada uno de ellos se muestra la forma en que se ligan las diferentes clases y su flujo en el sistema.

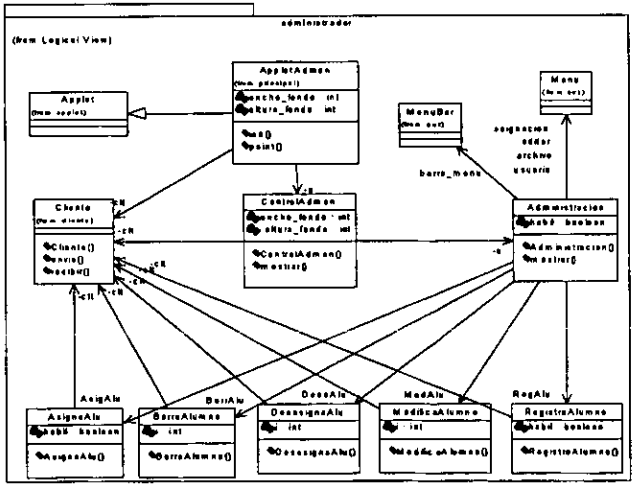
### 4.3.1.1. Administración / Alumnos

El diagrama D.4.1. ilustra la conceptualización de los elementos y métodos de los objetos utilizados por la entidad alumno y la forma en que las diferentes clases se relacionan. Cada cuadro dentro del diagrama representa a una de las clases con su respectivo nombre, sus variables públicas y sus métodos públicos. Por ejemplo la clase *AppletAdmon* tiene como variables *ancho_fondo* y *altura_fondo* de tipo entero (*int*) y sus métodos son *init()* y *paint()*

Observese que se tiene como clase principal la clase *AppletAdmon*, la cual es una subclase de la clase *Applet*, que permite realizar las invocaciones desde el *navegador*. La clase *AppletAdmon* se relaciona con la clase *Cliente*, encargada de realizar las peticiones al servidor, mediante la creación de una instancia de ella (creación de un objeto *ctl*).

A su vez las clases *AsignaAlu, BorraAlumno, DesasignaAlu, ModificaAlumno* y *RegAlu* se relacionan con la clase *Cliente* para realizar las operaciones de asignación de alumnos a un grupo, eliminación del alumno dentro del sistema, eliminar la asignación de alumnos de un grupo en particular, modificación de los datos correspondientes al alumno y registro de alumnos al sistema respectivamente.

Existe también una clase llamada *Administración* la cual se relaciona con las clases *MenuBar* y *Menu* para mostrar en pantalla el menú correspondiente al alumno; esta clase utiliza a la clase *Cliente* y a todas las clases mostradas en la parte inferior del diagrama.

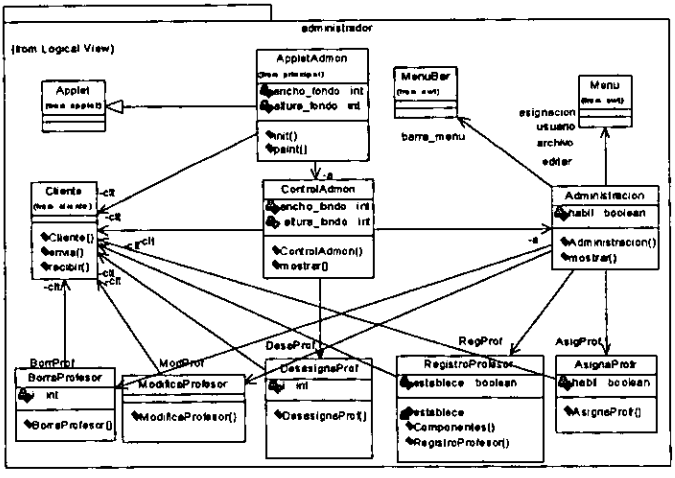



D.4.1.Diagrama de clases Administración/ Alumnos.

### 4.3.1.2. Administración / Profesores

El diagrama D.4.2. muestra los elementos, métodos y relación de clases utilizadas por el profesor. Al igual que el diagrama anterior la clase *AppletAdmon* (clase principal) se relaciona con la clase *Applet* y *Cliente*. Además de la clase *Administración* que se relaciona con la clase *MenuBar* y *Menu*.

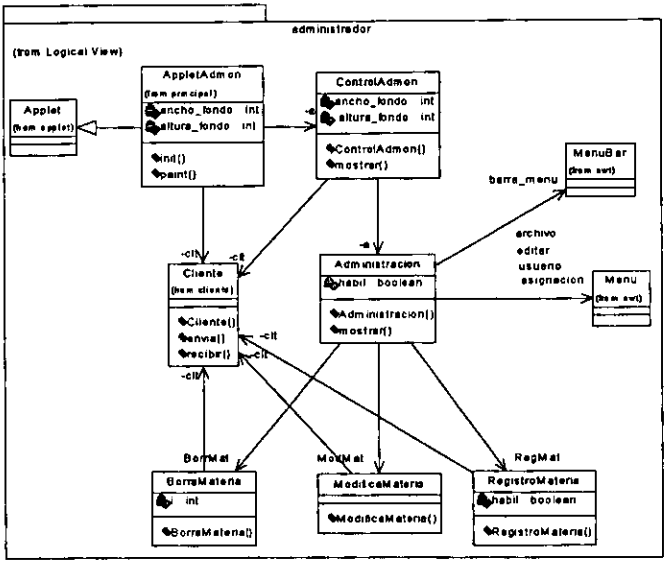
Nótese que las clases *BorrProf* (encargada de borrar a un profesor dentro del sistema), la clase *ModProf* (modifica los datos de un profesor especificado), la clase *DesasignaProf* (su función es eliminar la asignación de un profesor con un grupo en particular), la clase *RegProf* (ésta se encarga de registrar a un profesor dentro del sistema) y la clase *AsigProfr* (utilizada para asignar un profesor a un grupo); se relacionan con la clase *Cliente* y con la clase *Administración*.




D.4.2.Diagrama de clases Administración/ Profesores.

### 4.3.1.3. Administración / Materias

El diagrama D.4.3. es igual a los anteriores a excepción de que en este diagrama se puede observar que las clases *BorrMat, ModMat* y *RegMat* encargadas de borrar, modificar y registrar materias dentro de la aplicación realizada en el presente trabajo se relacionan con la clase *Cliente* y la clase *Administración.* Cada una de estas clases muestra sus variables publicas y sus métodos públicos.

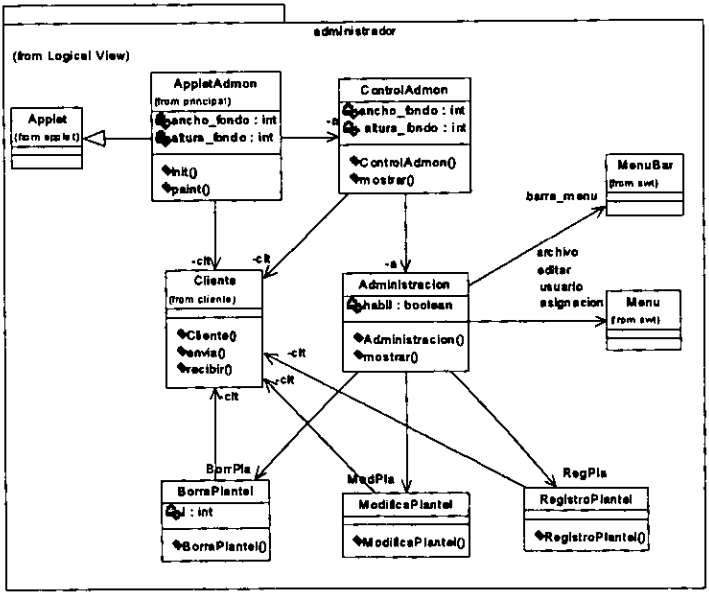



D.4.3.Diagrama de clases Administración/ Materias.

### 4.3.1.4. Administración / Planteles

El diagrama D.4.4. muestra los métodos variables y relación de clases utilizadas por los planteles.

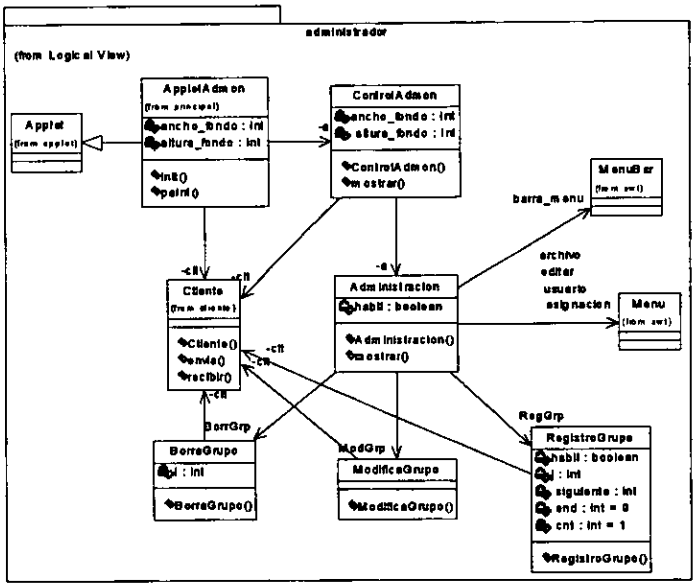
Este diagrama no tiene gran diferencia en su estructura inicial, al igual que los anteriores tiene la misma clase principal. Su diferencia radica en que ahora las clases que se relacionan con la clase *Cliente* y la clase *Administración* son: *BorrPla, ModPla* y *RegPla,* utilizadas para borrar, modificar y registrar planteles dentro de la aplicación.




D.4.3. Diagrama de clases Administración/ Planteles.

### 4.3.1.5. Administración / Grupos

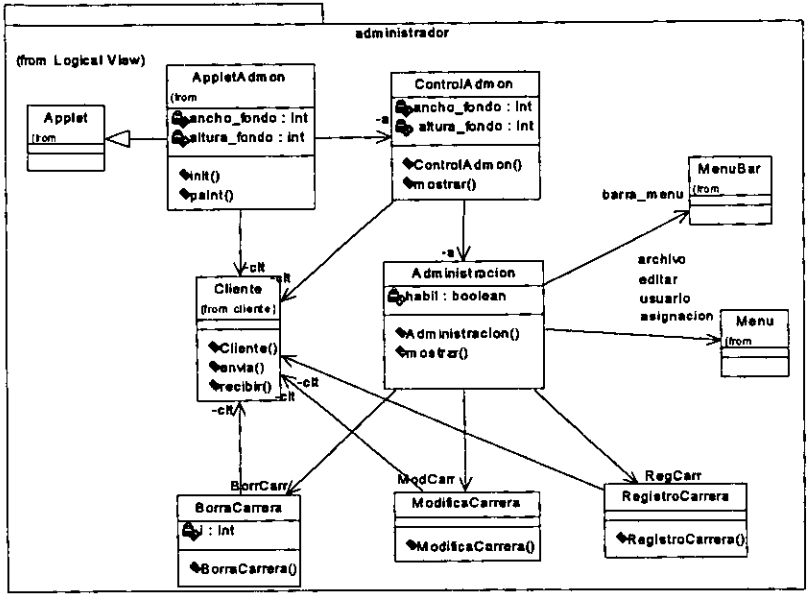
El diagrama D.4.5. muestra el diagrama de clases para los grupos; en el se puede observar como las clases *BorraGrupo, ModificaGrp* y *RegistroGrupo* (utilizadas para eliminar, modificar y registrar grupos a la aplicación desarrollada en este trabajo), se relacionan con la clase *Cliente* y con la clase *Administración*.




D.4.5. Diagrama de clases Administración/ Grupos.

### 4.3.1.6. Administración / Carreras

El diagrama D.4.6. muestra el diagrama de clases para las carreras, al igual que en los anteriores la parte inferior del diagrama es la que varia. En ella se observa la relación de las clases *BorrCarr, ModCarr* y *RegCarr* (utilizadas para borrar, modificar y registrar una carrera dentro de la aplicación) con la clase *Cliente* y *Administración*.



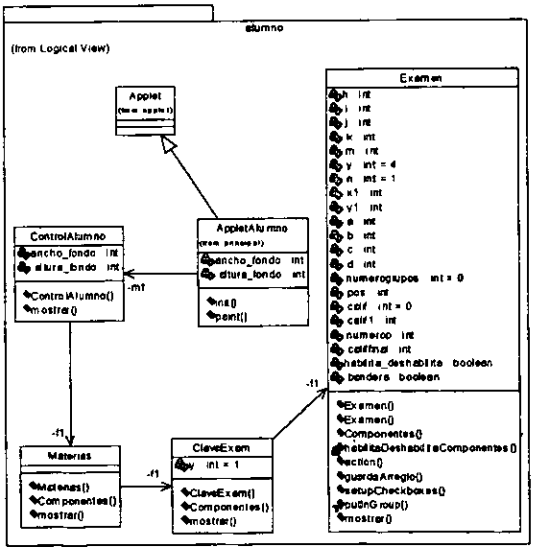

D.4.6. Diagrama de clases Administración/ Carreras.

### 4.3.1.7. Alumno

El diagrama D.4.7. muestra las variables *públicas*, métodos *públicos*, nombres y relaciones de clase utilizadas por el alumno.

Es importante observar que en este diagrama la clase principal es *ControlAlumno*, encargada de mostrar la validación de la identidad del alumno que se conecta a la aplicación. Esta clase es una subclase de *AppletAlumno*, que a su vez hereda de la clase *Applet*.

La clase *ControlAlumno* se relaciona con la clase *Materias*, para mostrar al alumno una lista de grupos y materias a las que pertenece. La clase *Materias* se relaciona con la clase *ClaveExam* (para la validación de la clave del examen) y ésta a su vez con la clase *Examen*, utilizada para mostrar el examen en la pantalla.




D.4.7. Diagrama de clases Alumnos.



### 4.3.1.8. Profesor

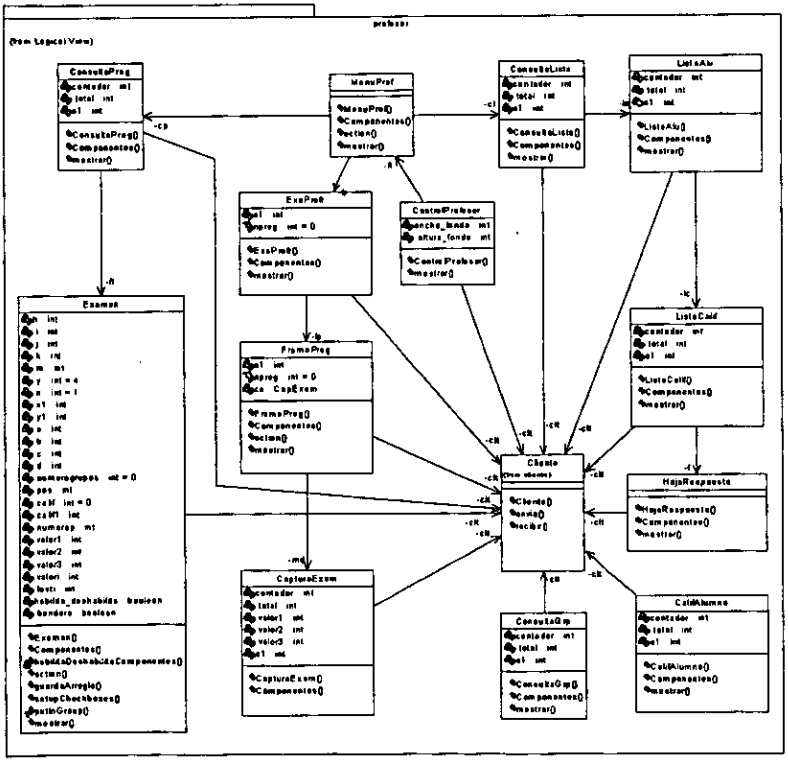
El diagrama D.4.8. muestra el diagrama de clases para el profesor. La clase principal en este caso es la de *ControlProfesor*, utilizada para la validación de la clave del profesor. Esta clase se relaciona con *MenuProf*, encargada de mostrar su menú al profesor.

Para que esta a su vez se relacione con *ConsultaPreg*, encargada de mostrar las preguntas de los exámenes ya aplicados (de ahí la relación *ConsutaPreg - Examen*).

La clase *MenuProf* se relaciona con *ExaProfr* (Donde el profesor introduce el examen) y esta a su vez con *FramePreg*, la cual se relaciona con *CapturaExam*.

La clase *MenuProf* también se relaciona con *ConsultaLista*, y está última con *ListaAlu*, para mostrar una lista de alumnos y calificaciones (mediante la relación *ListaAlu – ListaCalif, ListaCalif - HojaRespuesta*).

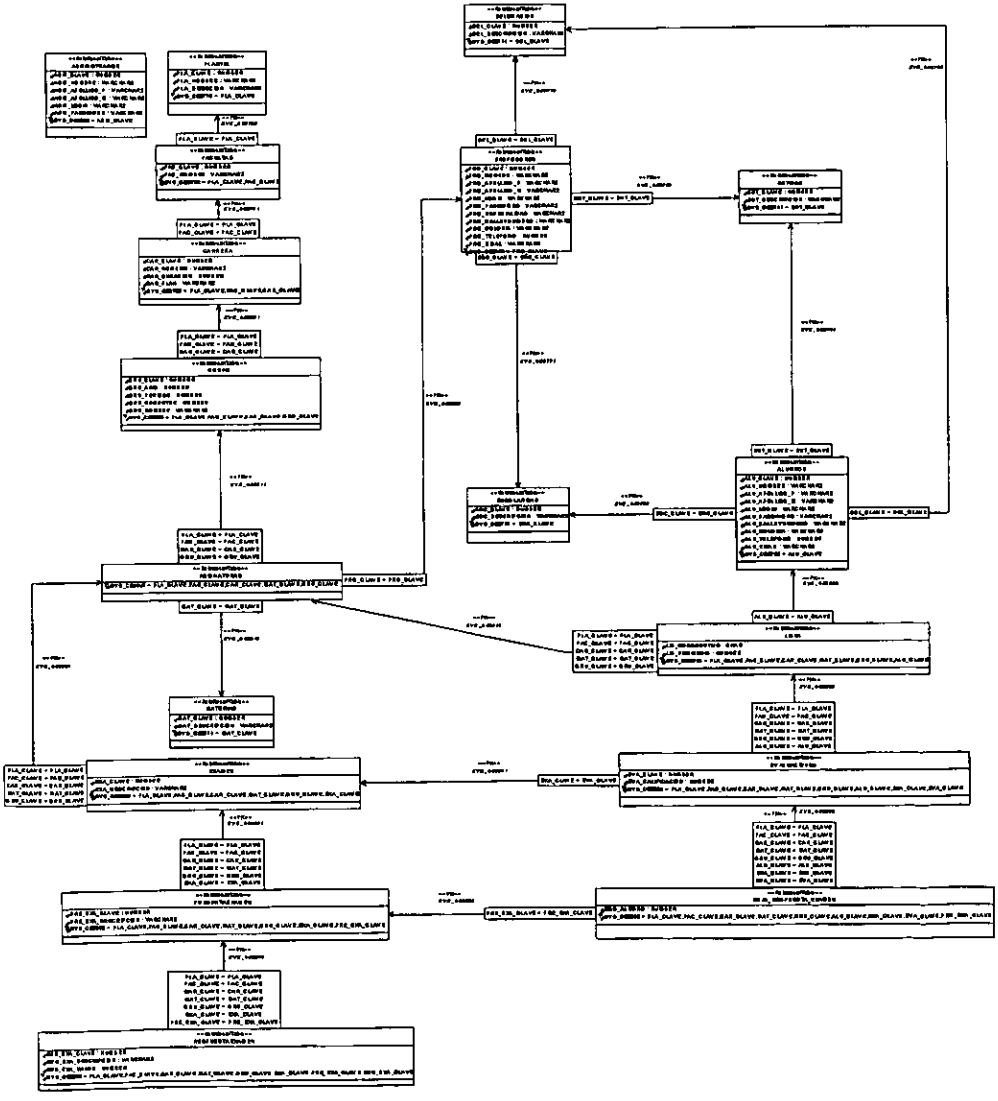
La clase *Cliente* encargada de enviar las peticiones al servidor se relaciona con las todas clases, menos la de *MenuProf*.



D.4.8. Diagrama de clases Profesor.

### 4.3.1.9. Estructura de la base de datos

El diagrama D.4.9.corresponde al diagrama de clases elaborado para la base de datos. Nótese que cada cuadro representa a cada una de las tablas con su nombre y sus respectivas llaves primarias y cada flecha indica la llave foránea que establece la relación entre las tablas.



D.4.9. Diagrama de clases de la base de datos.

## 4.4.2 *Diagramas de componentes*

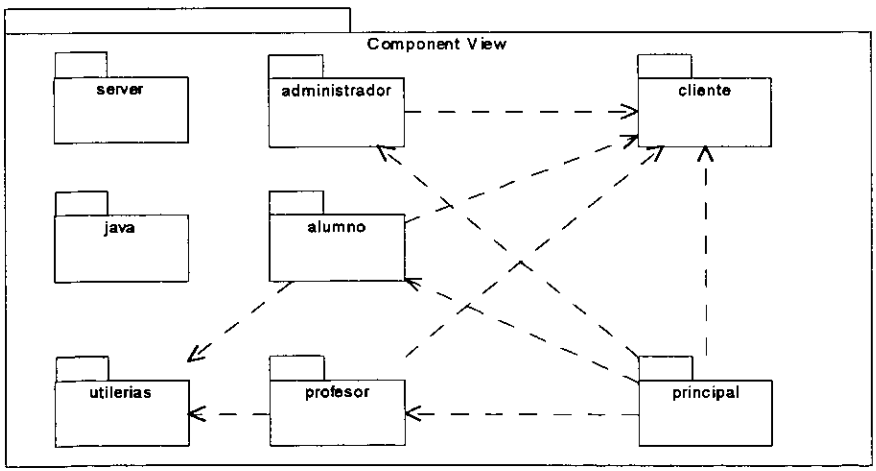
En el diagrama de componentes se modela la implementación estática del sistema. Se emplean los diagramas de componentes para modelar la gestión de los archivos de código fuente de la aplicación escrita en *Java*. Los paquetes representan los módulos lógicos de la aplicación en dónde cada paquete es uno de los subdirectorios físicos en que se organizan los archivos **java*; los componentes a su vez son la representación lógica de las clases y representan cada uno un archivo **java*. Las relaciones muestran las dependencias entre los paquetes y componentes, esto es cuando se hace referencia a un *extend* o a un *import* de una clase dentro de otro archivo **java*.

### 4.3.2.1. Paquetes

El diagrama D.4.10., muestra la relación que guardan entre sí todos los paquetes desarrollados en la aplicación del presente trabajo de tesis, que de aquí en adelante se hará referencia a ella como el *Sistema de Evaluación* .

En los diagramas siguientes se omite las relaciones que se hacen a los paquetes de la clase *java.** debido a que siendo el paquete de la *API* de *java*, presenta ligas hacia todos los demás paquetes y componentes y esto distorsiona la comprensión de los diagramas.

Cada fólder dentro del diagrama representa a cada uno de los paquetes.

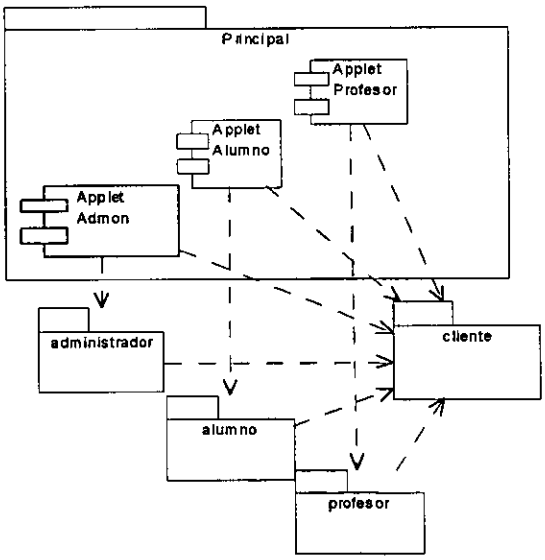



D.4.10. Paquetes de la aplicación desarrollada en este trabajo.

Los subsecuentes diagramas muestran a detalle los componentes contenidos por cada uno de los paquetes.

### 4.3.2.2. Paquete principal

El paquete *principal,* está compuesto por los programas *AppletProfesor.java, AppletAlumno.java* y *AppletAdmon.java.* En el diagrama D.4.11, se puede observar como estos programas se relacionan directamente con el paquete *Cliente* y con el paquete *Profesor, Alumno* y *Administrador* respectivamente, debido a que los programas dentro del paquete principal actúan como interfaz al ser invocados por el código *HTML,* y el comportamiento de cada uno de los módulos se representa en paquetes independientes. Esto se realizó de está manera pensando que en un futuro, alguno de los módulos necesitara cambios, y que estos se puedan realizar de manera independiente de los otros paquetes.




D.4.11. Componentes del paquete *Principal.*

### 4.3.2.3. Paquete administrador

El paquete del *administrador,* contiene todos aquellos programas útiles a las actividades del administrador de la aplicación desarrollada en este trabajo. Estos programas

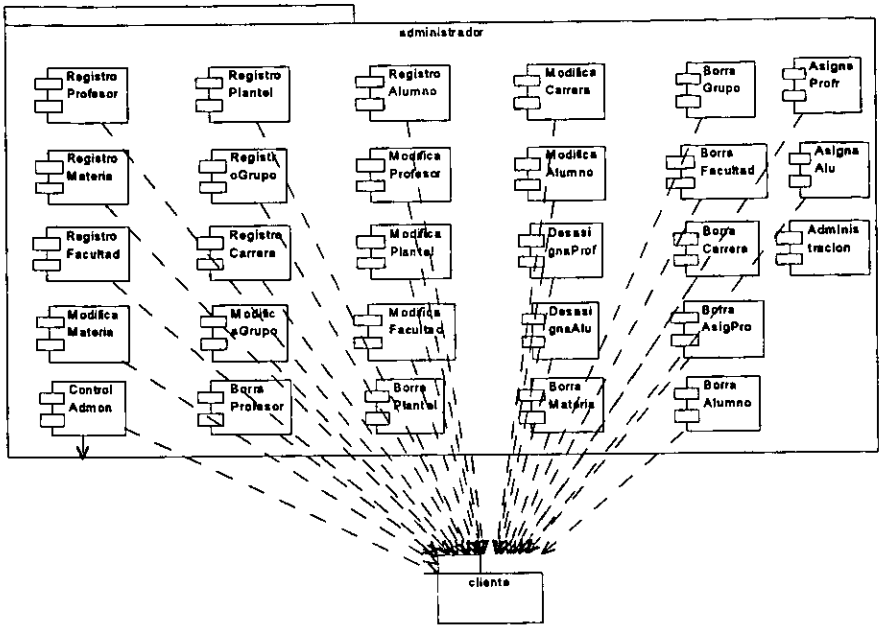
incluyen desde realizar altas, bajas y cambios de profesores, alumnos, planteles, carreras, facultades, materias, asignación de alumnos, profesores a grupos , etc.[59]

En el diagrama D.4.12 se puede observar como todos los programas del paquete administrador se relacionan con el paquete *Cliente*, el cual permite a cada uno realizar las operaciones correspondientes hacia la base de datos.

Este paquete se conforma por los siguientes programas:

*RegistroProfesor.java* .- Permite dar de alta a un profesor dentro de la aplicación.
*RegistroPlantel.java* .- Permite dar de alta un plantel dentro de la aplicación.
*RegistroMateria.java* .- Permite ingresar una materia a la aplicación.
*RegistroGrupo.java* .- Permite dar de alta un grupo en la aplicación.
*RegistroFacultad.java* .- Permite dar de alta una facultad dentro de la aplicación.
*RegistroCarrera.java* .- Permite dar de alta una carrera en la aplicación.
*RegistroAlumno.java* .- Permite dar de alta a un alumno en la aplicación.
*ModificaProfesor.java* .- Permite modificar los datos del profesor.
*ModificaPlantel.java*.-Permite modificar los datos referentes a un plantel específico.
*ModificaMateria.java*.- Permite modificar los datos de una materia.
*ModificaGrupo.java* .- Permite la modificación de los datos de un grupo .
*ModificaFacultad.java*.- Permite la modificación de los datos de una facultad.
*ModificaCarrera.java* .- Permite modificar los datos referentes a una materia.
*ModificaAlumno.java* .- Permite la modificación de los datos de un alumno,
*DesasignaProf.java*.- Permite la asignación de un profesor a un grupo específico.
*DesasignaAlu.java* .- Elimina la asignación de un alumno con cierto grupo.
*ControlAdmon.java*.- Valida la identidad del administrador con la aplicación.
*BorraProfesor.java*.- Elimina a un profesor específico de la aplicación.
*BorraPlantel.java*.- Elimina los datos de un plantel determinado.
*BorraMateria.java*.- Elimina una materia de la aplicación.
*BorraGrupo.java*.- Elimina los datos de un grupo.
*BorraFacultad.java*.- Elimina los datos de un facultad.
*BorraCarrera.java*.- Elimina los datos de una carrera.
*BorraAsigPro.java*.- Elimina la asignación de un profesor con un grupo.
*BorraAlumno.java*.- Elimina los datos de un alumno en particular.
*AsignaProfr.java*.- Asigna un profesor con un grupo.
*AsignaAlu.java*.- Asigna un alumno a un grupo especifico.
*Administracion.java*.- Muestra el menú del administrador, para que elija la operación a realizar.

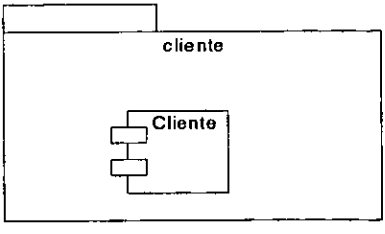
[59] *Si se requiere conocer mejor las actividades del administrador , consulte el capitulo I*




D.4.12. Componentes del paquete *administrador*.

### 4.3.2.4. Paquete cliente

El paquete cliente, es el encargado de atender las peticiones que realizan los programas de los paquetes *administrador, alumno* y *profesor*, se encarga de establecer la comunicación con el servidor (donde se encuentra la base de datos) con el fin de enviar y recibir datos del mismo. Este paquete sólo esta compuesto por un programa llamado *Cliente.java*[60].(D.4.13.)



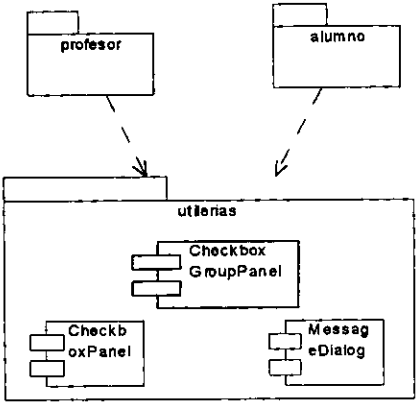

D.4.13. Componentes del paquete *cliente*.

[60] *El código del programa* Cliente *de incluye en el disco anexo a este trabajo.*

### 4.3.2.5. Paquete utilerías

El paquete utilerías (D.4.14.) contiene los componentes, de la interfaz gráfica de la aplicación, necesarios para mostrar mensajes, selección de opciones y grupos de selección de datos. Incluye los programas:

*CheckboxGroupPanel.java* .- Permite realizar el manejo de grupos de selección.
*CheckboxPanel.java* .- Permite realizar el manejo de selección de opciones.
*MessageDialog.java* .- Permite la creación de cajas de diálogo dinámicamente, para interactuar con el usuario de la aplicación.



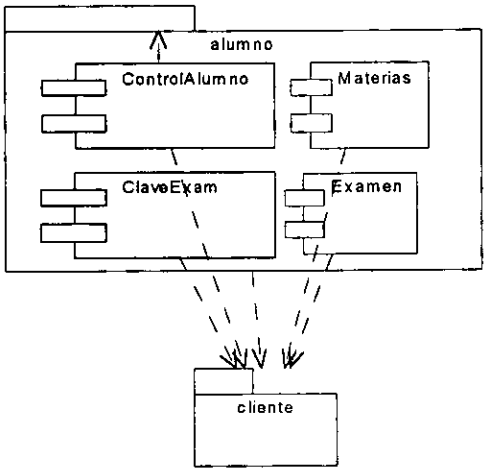

D.4.14. Componentes del paquete *utilerias*.

### 4.3.2.6. Paquete alumno

En este paquete (D.4.15.) se encuentran las clases y programas utilizados por el alumno para validarse en el sistema, visualizar una lista de materias con sus respectivos grupos en las que se encuentra inscrito, elegir en cual se realizará el examen y finalmente visualizar el examen a realizar. Los programas que forman parte de este paquete son:

*Materias.java* .- Muestra al alumno una lista de materias y grupos en los que se encuentra inscrito.
*Examen.java* .- Muestra el examen a realizar.
*ControlAlumno.java* .- Permite al alumno acceder a la aplicación, mediante este programa se realiza la verificación de la clave proporcionada por el alumno.
*ClaveExam.java* .- Permite al alumno ingresar la clave del examen a realizar.

En el diagrama D.4.15. también se puede observar que todos los programas hacen uso del paquete *Cliente*.




D.4.15. Componentes del paquete *alumno*.

### 4.3.2.7. Paquete profesor

En el diagrama D.4.16, se observan los programas que contiene el paquete *profesor*, para realizar las actividades propias del profesor; tales como el control de acceso al sistema, la muestra del menú del profesor, consulta de grupos, consulta de listas, consulta de preguntas, calificación de alumnos, lista de calificaciones, captura, modificación y consulta de exámenes. Los programas que forman parte de este paquete son:

*CalifAlumno.java* - Muestra la calificación de un alumno específico.

*CapturaExam.java*- Permite al profesor realizar la captura de preguntas y respuestas con sus valores respectivos.

*ConsultaGrp.java*- Muestra una lista de los exámenes correspondientes a un grupo específico.

*ConsultaLista.java*.- Muestra una lista con los grupos correspondientes al profesor que realiza la consulta, con el fin de permitir la elección de un grupo y posteriormente mostrar la lista de los alumnos que se encuentran inscritos en dicho grupo.

*ConsultaPreg.java*.- Muestra una lista con los exámenes correspondientes a un grupo especifico, para que el profesor seleccione el examen en el que desea consultar los reactivos

*ControlProfesor.java*.- Muestra la pantalla de control de acceso para el profesor compara login y password proporcionado por el profesor con el login y password que se encuentra en la base de datos.

*Examen.java*.-Permite la visualización del examen en pantalla.

*ExaProf.java.*- Muestra al profesor la lista de grupos, para la selección de uno, con la finalidad de introducir un examen a la aplicación.

*FramePreg.java.*-Permite introducir a la aplicación el número de preguntas correspondientes a un examen.

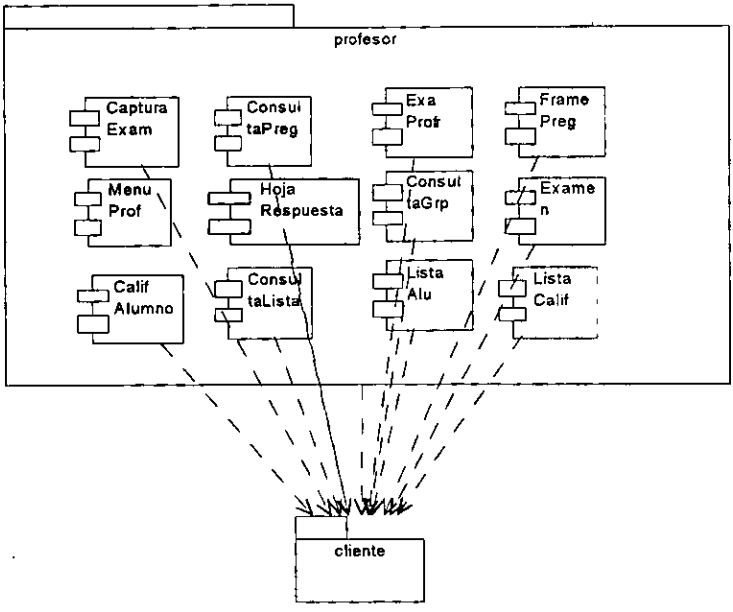
*HojaRespuesta.java.*-Muestra en pantalla las respuestas de un alumno. Por sí este requiere alguna aclaración respecto a su examen..

*ListaAlu.java.*- Muestra la lista de alumnos de un grupo específico.

*ListaCalif.java.*- Muestra la lista de alumnos de un grupo específico y sus calificaciones.

*MenuProf.java.*- Despliega el menú, en el cual el profesor elige la operación a realizar.

En este paquete al igual que en el anterior todos los programas utilizan el paquete *Cliente* para realizar sus operaciones con la base de datos.




D.4.16. Componentes del paquete *profesor*.

### 4.3.2.8. Paquete fija

El paquete *fija* al igual que el paquete *utilerias* forma parte de la interfaz gráfica de la aplicación. Este paquete, es utilizado para mostrar en pantalla cajas de diálogo, reacomodar componentes tales como botones, campos de texto, etiquetas, etc, y asignar claves para los registros de las tablas de la base de datos, esto es con el fin de conservar la integridad. Los programas que forman parte de este paquete son:

*ValorRegistro.java.*- Asignación de claves únicas a los registros automáticamente, dependiendo del tipo de tabla que se le indique.

*FijarValores.java.*- Establece los valores para la colocación de componentes dentro de un contenedor con la finalidad de visualizarlos correctamente en pantalla.

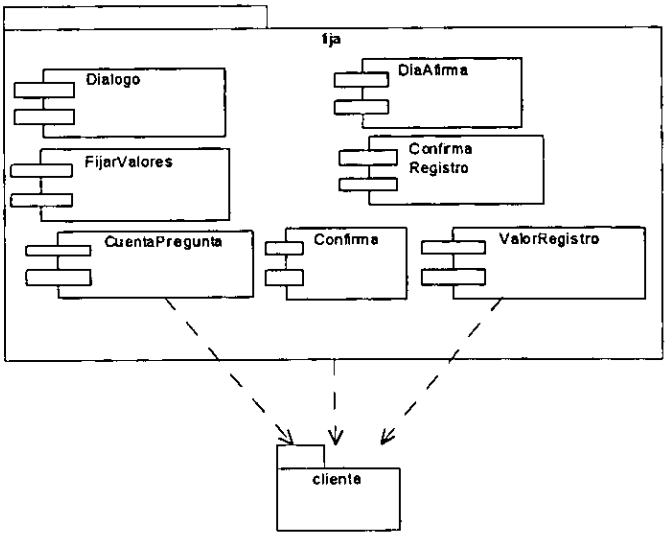
*Dialogo.java.*- Creación de objetos tipo caja de diálogo, para interactuar con el usuario de la aplicación.

*DiaAfirma.java.*- Creación de objetos tipo caja de diálogo.

*CuentaPregunta.java.* -Cuantifica el número de preguntas de un examen.

*ConfirmaRegistro.java* .- Verificar la existencia de un registro.

*Confirma.java.*- Realiza una caja de diálogo para la confirmación de los registros a borrar.




D.4.17. Componentes del paquete *fija*.

## *4.4.3 Diagramas de despliegue*

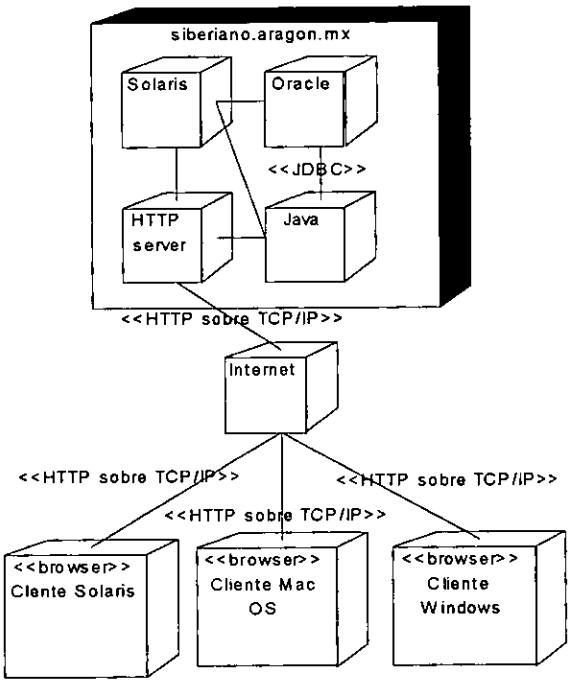
El diagrama de despliegue muestra el modelado estático del despliegue del sistema en función de sus elementos de arquitectura. El diagrama de despliegue muestra la disposición de la arquitectura cliente - servidor.

### 4.4.3.1 Estructura cliente-servidor

En el diagrama D.4.18. se puede observar el diagrama de despliegue para la arquitectura cliente–servidor, de la aplicación realizada en este trabajo

La parte inferior del diagrama representa a una máquina cliente con sistema operativo *Windows*, *MacOS* o *Solaris*, los cuáles tienen un *browser* con su *máquina virtual Java*[61]. La máquina cliente realiza peticiones que viajan a través de la red (*Internet*), para llegar al otro extremo, donde se encuentra una máquina servidor, encargada de atender dichas peticiones; el servidor identificado por la dirección *siberiano.aragon.unam.mx*, contiene el sistema operativo *Solaris*, la base de datos *Oracle*, los *drivers* necesarios para interactuar con la base de datos, el servidor de *Web* (*"http server"*), el lenguaje de programación *Java* y los programas de la aplicación.

[61] *Encargada de interpretar el código de bytes, como se explico en el capitulo I.*




D.4.18. Estructura cliente - servidor

# 4.5 Diagramas de comportamiento

Los diagramas de comportamiento se utilizan para modelar la parte dinámica de la aplicación y conforman las bases para el desarrollo de la misma. Los diagramas para modelado del comportamiento que se emplean este análisis son:

- Diagramas de casos de uso
- Diagramas de colaboración

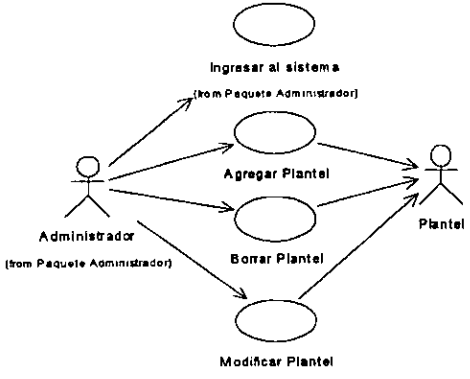
## *4.5.1 Diagramas de casos de uso*

Los casos de uso proporcionan un mecanismo para el modelado de los requisitos del sistema, los actores y los comportamientos comunes. Los casos de uso representan cada uno de los requerimientos del sistema y los actores representan un rol de actividades plenamente identificado.

Los siguientes diagramas son la representación de los requerimientos del sistema en función del desarrollo de la navegación del mismo, es decir, representan el flujo de la interfaz de la aplicación de acuerdo a su funcionalidad operativa.

### 4.4.1.1. Estructura plantel

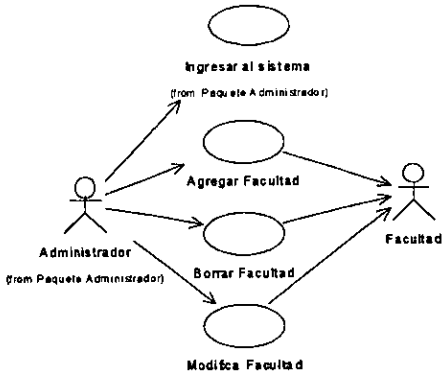
En el diagrama D.4.19 se modelan el comportamiento de los actores, *Administrador* y *Plantel*, en el se puede ver como el *Administrador* interactúa valiéndose de los casos de uso con respecto del actor *Plantel*. Los casos de uso aquí representados muestran la forma en que el actor administrador puede agregar al actor plantel, así como eliminarlo o modificarlo. Dentro del contexto de *UML* la representación de un actor se hace mediante el icono que parece un muñeco. Hay que tener cuidado de no confundir la representación de un actor con la de una persona, es decir que el actor representa una entidad con roles claramente establecidos, los cuales no forzosamente deben ser una persona. Los casos de uso se representan mediante la figura de una elipse y se coloca debajo de ella la descripción de la función que representan.




D.4.19. Caso de uso de la *Estructura Plantel.*

### 4.4.1.2. Estructura Facultad

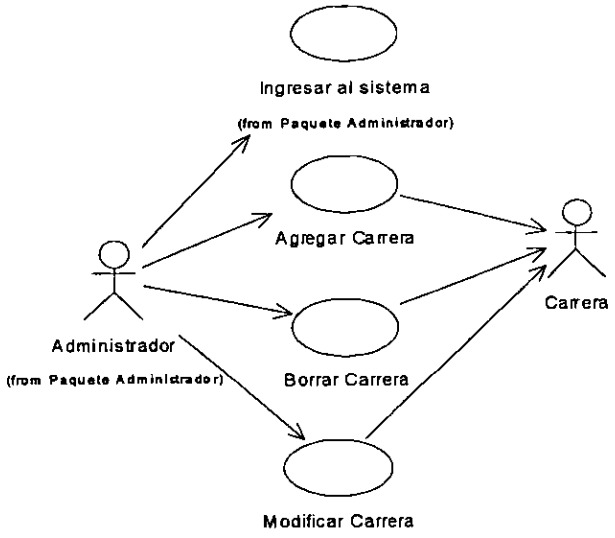
En el diagrama D.4.20. se modela la *Estructura Facultad* la cual tiene un actor del mismo nombre y los casos de uso que permiten su creación, modificación y eliminación. Estas tareas deben de ser realizadas por el actor etiquetado como *Administrador*.




D.4.20. Caso de uso de la *Estructura Facultad.*

### 4.4.1.3. Estructura Carrera.

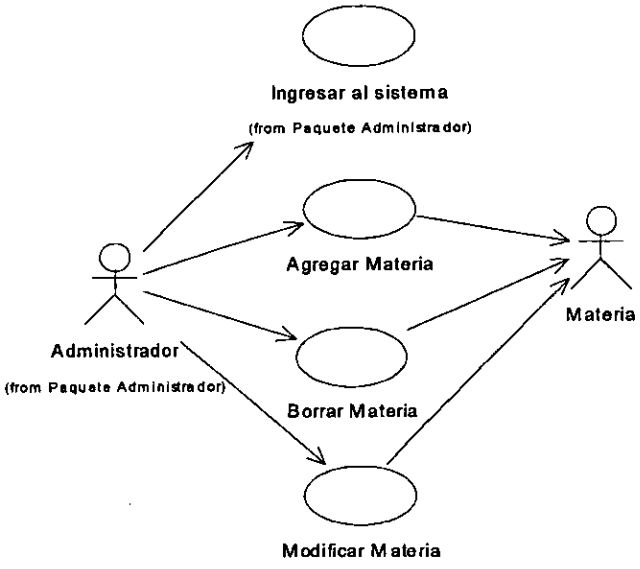
En el diagrama D.4.21., al igual que en las anteriores representaciones tenemos el actor *Administrador*, el cual utiliza el caso de uso para ingresar al sistema, además de los correspondientes para el manejo del actor *Carrera*.




D.4.21. Caso de uso de la *Estructura Carrera*.

### 4.4.1.4. Estructura Materia.

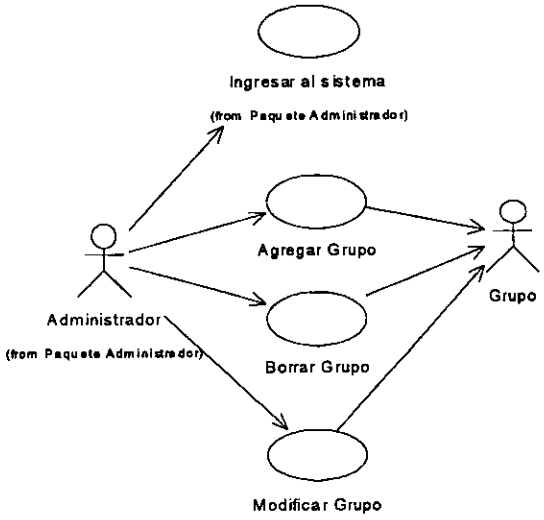
El diagrama D.4.22. tiene la misma conceptualización utilizada hasta ahora en el sentido de los casos de uso básicos y cuya diferencia es el actor resultante que en este caso es el actor *Materia*.




D.4.22. Caso de uso de la *Estructura Materia*.

### 4.4.1.5. Estructura Grupo.

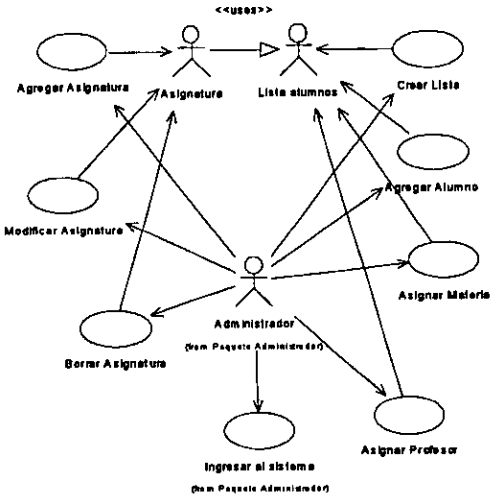
El diagrama D.4.23. muestra los casos de uso referentes a los actores *Grupo* y *Administrador* de la *Estructura Grupo*.




D.4.23. Caso de uso de la estructura grupo.

### 4.4.1.6. Estructura Asignatura.

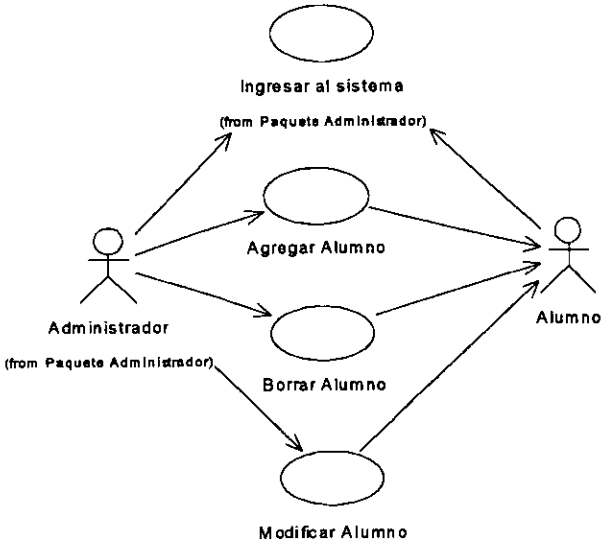
En el diagrama D.4.24. se tiene un modelo un poco más complejo en el que hay tres actores que se relacionan para poder generar la *Estructura Asignatura*. En él se observa que el actor *Administrador* emplea los casos de uso para manejo de *Asignatura*. El actor *Asignatura* usa a su vez al actor *Lista alumnos*. El actor *Lista alumnos* es creado también por el *Administrador* con el caso de uso *Crear lista*. Además, el *Administrador* puede agregar o quitar alumnos, asignar materias y asignar un profesor. Todo esto en conjunto forma una *Estructura Asignatura*, la cual está formada por un conjunto de alumnos, una materia y un profesor.




D.4.24. Caso de uso de la estructura asignatura.

### 4.4.1.7. Estructura Alumno.

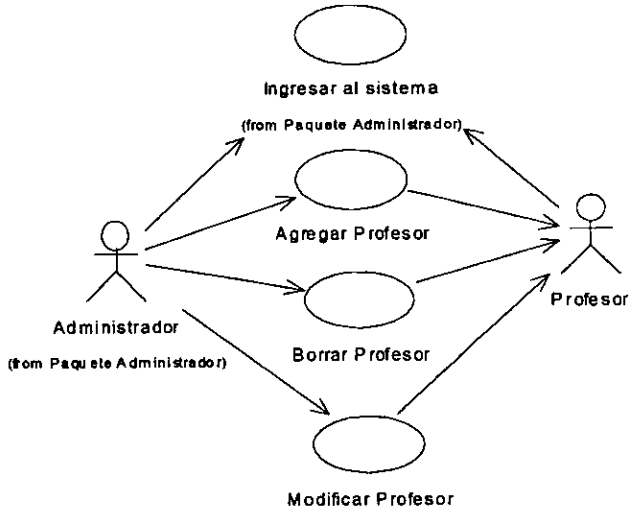
El diagrama D.4.25. al igual que los diagramas anteriores, ayuda a identificar los requerimientos de la aplicación para poder hacer el manejo de la información referente a los alumnos. En este diagrama se observa una diferencia con respecto de los anteriores y es que el actor *Alumno* además de ser creado, modificado o borrado por el *Administrador* también utiliza el caso de uso *Ingresar al sistema* ya que este actor después de ser creado puede utilizar la aplicación.




D.4.25. Caso de uso de la estructura alumno.

### 4.4.1.8. Estructura Profesor.

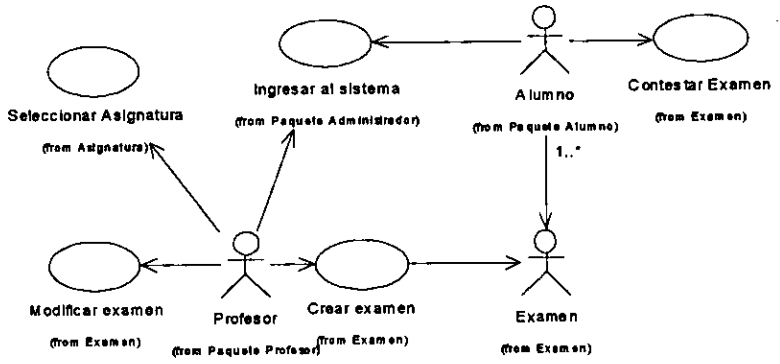
La *Estructura Profesor* modelada en el diagrama D.4.26. representa las necesidades de la aplicación referente a profesores en este diagrama también se observa que el actor *Profesor* además de ser creado, puede acceder a la aplicación.




D.4.26. Caso de uso de la estructura profesor.

### 4.4.1.9. Proceso Examen.

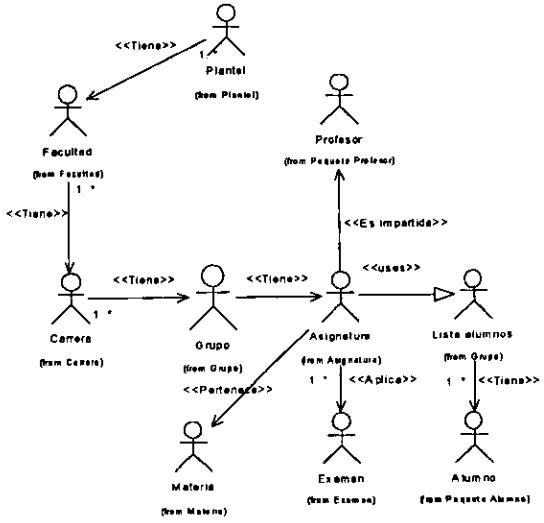
El diagrama D.4.27. plasma los requerimientos del sistema en la elaboración de un examen por parte del actor *Profesor* y una vez creado éste, la resolución del examen por parte del *Alumno*.




D.4.27. Caso de uso del proceso examen.

## *4.5.2 Diagramas de colaboración.*

Los diagramas de colaboración modelan la organización y la manera en que participan los objetos como una interacción dinámica entre ellos. También muestra la relación que existe entre los actores de acuerdo a su estructura jerárquica organizacional, como se muestra en diagrama D.4.28.




D.4.28. *Estructura Jerárquica Organizacional.*

# Capitulo 5. Análisis de resultados

A lo largo de éste capítulo, se muestra el *Sistema de Evaluación* desarrollado en el presente trabajo de tesis con el propósito de conocerlo, explicar las pruebas realizadas y comprender mejor los resultados obtenidos de ellas.

## 5.1 Presentación del Sistema de Evaluación

El *Sistema de Evaluación* es una aplicación desarrollada a lo largo de este trabajo de tesis cuyo objetivo es el de proporcionar una herramienta de trabajo tanto a profesores como alumnos, ya que permite al profesor colocar exámenes sobre *Internet* con el fin de que el alumno realice sus evaluaciones remotamente sin necesidad de tener al profesor presente físicamente. Es importante mencionar que no se trata en ningún momento de sustituir la labor del profesor como evaluador, sino de brindarle, una herramienta de apoyo para la realización de sus actividades.

Este sistema está formado por cuatro módulos:

- Administrador
- Profesor
- Alumno
- Ayuda en línea

El modulo del administrador es el encargado de realizar las operaciones de altas, bajas y cambios en cuanto a las escuelas, grupos, materias y profesores.

El módulo del profesor permite a éste introducir preguntas para exámenes de diferentes materias y asignar un valor para la calificación. Es importante mencionar que el sistema sólo acepta preguntas con respuestas del tipo opción múltiple.

El módulo del alumno, permite al alumno realizar los exámenes colocados por el profesor. Y conocer su calificación inmediatamente al término de su examen.

El módulo de ayuda está en línea, esto quiere decir que se encuentra directamente dentro de la *página HTML*, su propósito es mostrarle a los usuarios el funcionamiento del sistema y los pasos necesarios para realizar una operación, ya sea una alta, baja, cambio, responder exámenes, dar de alta preguntas, etc.

A continuación se muestran las pantallas del *Sistema de Evaluación*. La figura 5.1. es la página principal del *Sistema de Evaluación* .

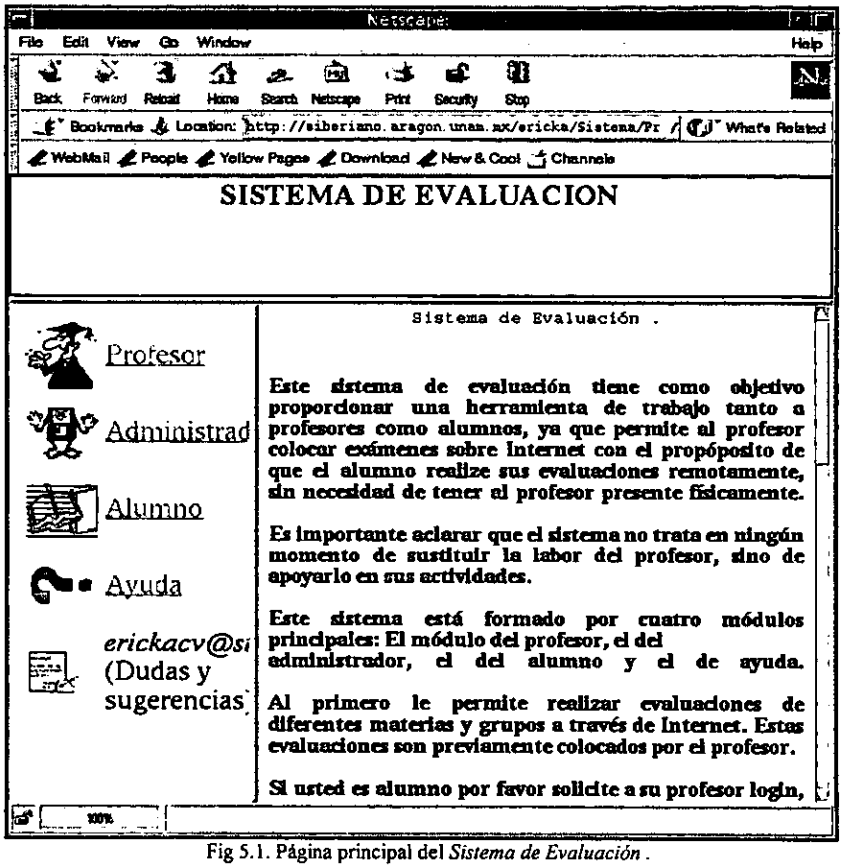



Fig 5.1. Página principal del *Sistema de Evaluación* .

En ésta figura se puede observar una liga en la página *HTML*, la cual lleva a cada módulo del sistema y una breve introducción del objetivo principal del mismo.

## *5.1.1 Módulo del Administrador.*

Para acceder al módulo del administrador es necesario proporcionar un login[62] y un password[63] mediante una pantalla como la que se muestra en la figura 5.2.

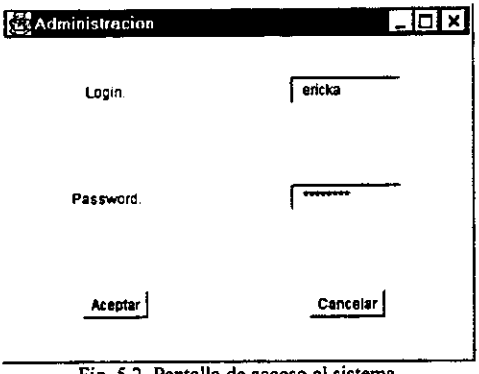



Fig. 5.2. Pantalla de acceso al sistema.

Si cualquiera de estos datos no es correcto, aparecerá una caja de diálogo, mostrando que el password no es autorizado, cómo se muestra en la figura 5.3.

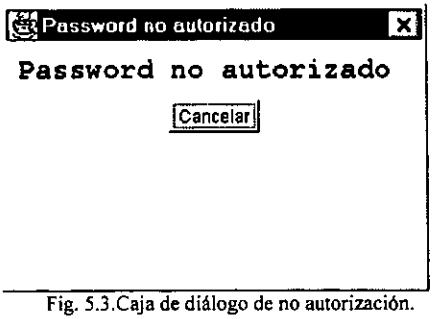



Fig. 5.3.Caja de diálogo de no autorización.

[62] *Clave personal de acceso a la aplicación*
[63] *Clave secreta de acceso, complementaria a la clave personal.*

Si no son proporcionados cualquiera de estos datos, una caja de diálogo solicitará nuevamente se introduzca el login y el password (Fig. 5.4).

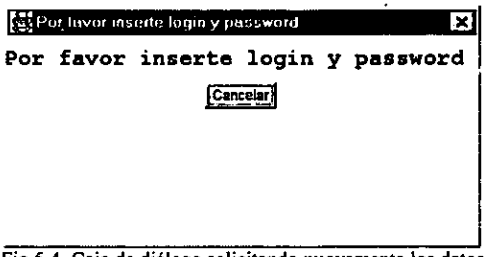



Fig.5.4. Caja de diálogo solicitando nuevamente los datos.

Una vez que el administrador ha proporcionado login y password correctamente, aparecerá en la pantalla el menú que corresponde a la administración del *Sistema de Evaluación*, el cual está formado de tres opciones: *Salir*, *Editar* y *Asignar* (Fig.5.5).

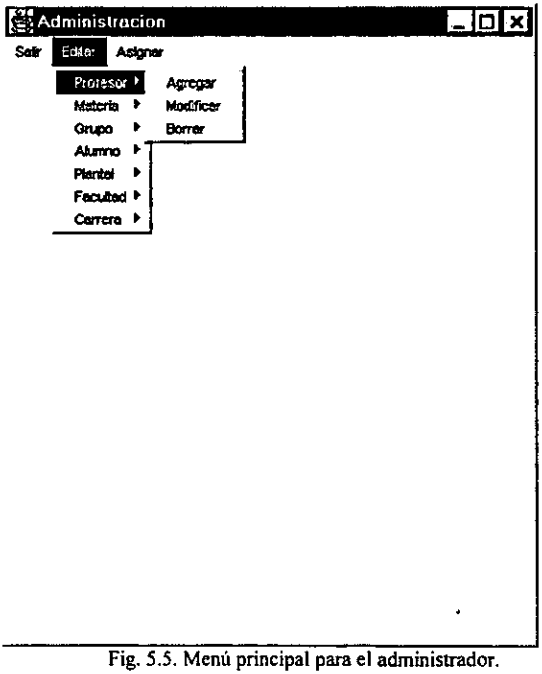



Fig. 5.5. Menú principal para el administrador.

Al presionar en la opción de *"Editar"* se despliega un submenú de opciones como: profesor, materia, grupo, alumno, plantel, facultad y carrera y cada una de ellas a su vez tiene la opción de *"Agregar"*, *"Modificar"* ó *"Borrar"*.

Si se presiona la opción *"Asignar"*, el administrador puede asignar un profesor a un grupo y materia determinados, o bien, asignar un alumno a una materia específica. También puede eliminar esta asignación de profesores y de alumnos, si así lo requiere.

Para dar de alta un plantel, el *Sistema de Evaluación* asignará una clave única para cada plantel. Y el usuario proporcionará el nombre del plantel ó facultad y su dirección. A continuación se muestra la pantalla que realiza el alta del planteles (Fig. 5.6.).

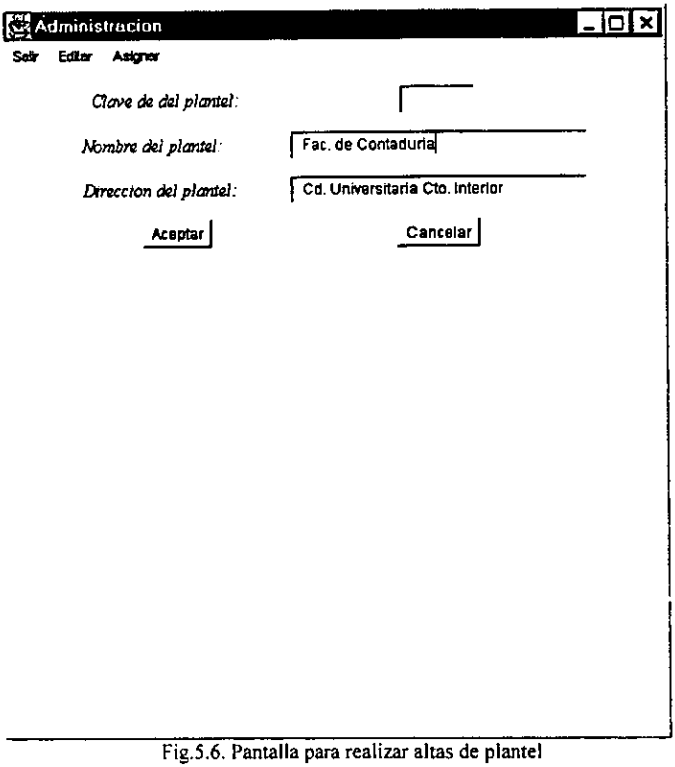



Fig.5.6. Pantalla para realizar altas de plantel

Antes de borrar el plantel definitivamente del sistema aparece una caja de diálogo pidiendo la confirmación para borrar el plantel (Fig. 5.9.)

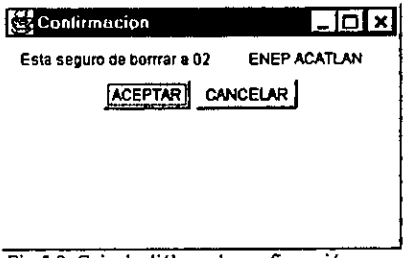



Fig 5.9. Caja de diálogo de confirmación.

Para la modificación de los datos de un plantel (Fig.5.10.) se debe seleccionar el plantel que se requiere modificar y después presionar el botón *"Aceptar"*, esto es con el fin de que los datos del plantel se muestren y se puedan cambiar, para finalmente presionar el botón *"Actualizar"*.

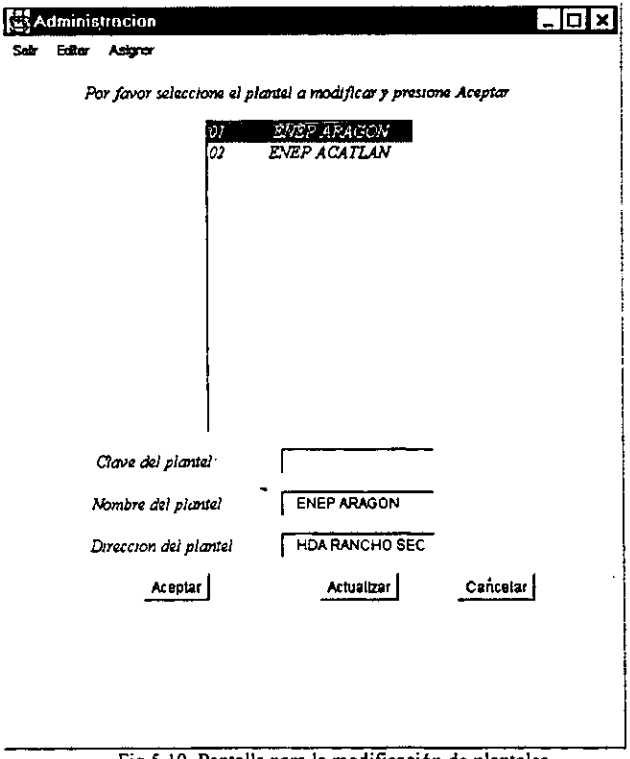



Fig.5.10. Pantalla para la modificación de planteles.

Una vez realizada la modificación sobre el plantel elegido, aparece una caja de diálogo indicando el éxito de la operación (similar a las anteriores).

La forma en la que se realizan las operaciones de alta, baja y modificación para una facultad y carrera es similar a la del plantel

Otra de las operaciones realizadas por el administrador es realizar la alta de una materia dentro del catálogo del *Sistema de Evaluación* , para ello es necesario proporcionar únicamente el nombre de la materia, ya que el sistema le asignará una clave automáticamente. (Fig. 5.11)

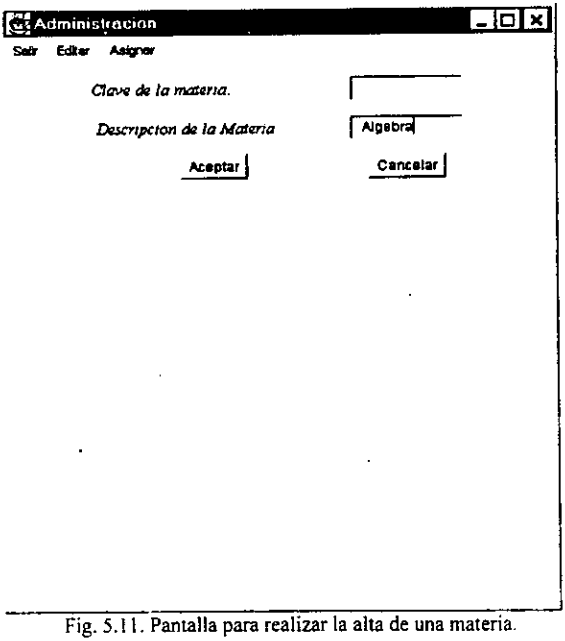



Fig. 5.11. Pantalla para realizar la alta de una materia.

Una vez dada de alta la materia, aparecerá el mensaje de *"Materia dada de alta"* en una caja de diálogo como se muestra en la Fig. 5.12.

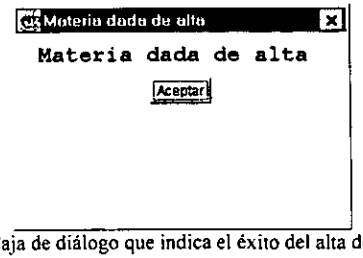



Fig. 5.12. Caja de diálogo que indica el éxito del alta de materia.

Para dar de baja una materia dentro del sistema se utiliza la siguiente pantalla (Fig. 5.13.). En ella se elige la materia que se desea borrar y se presiona el botón de *"Aceptar"*.

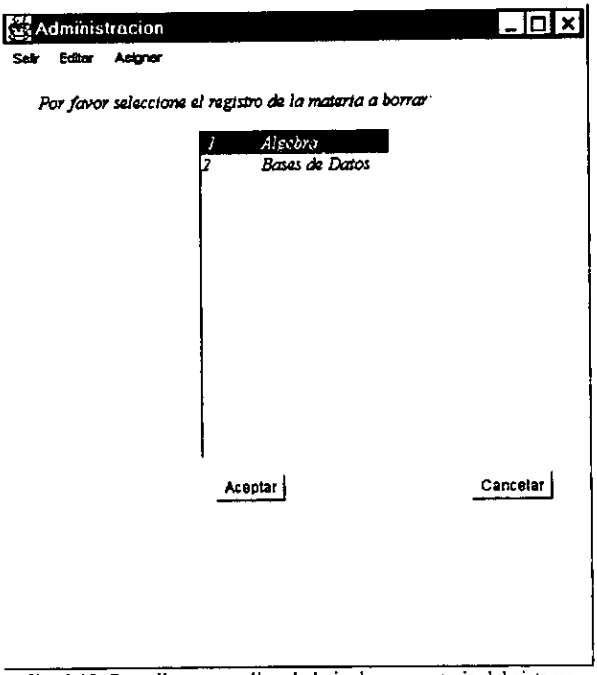



Fig. 5.13. Pantalla para realizar la baja de una materia del sistema.

Antes de borrar la materia aparece un mensaje pidiendo la confirmación[64] para borrarla definitivamente del sistema.

[64] *Similar a la que aparece cuando se borra un grupo.*

Para realizar la modificación de los datos de alguna materia se utiliza la siguiente pantalla (Fig.5.14.), en ella, se elige la materia que se quiere modificar para que los datos correspondientes a ésta aparezcan en la parte inferior de la pantalla y se modifiquen:

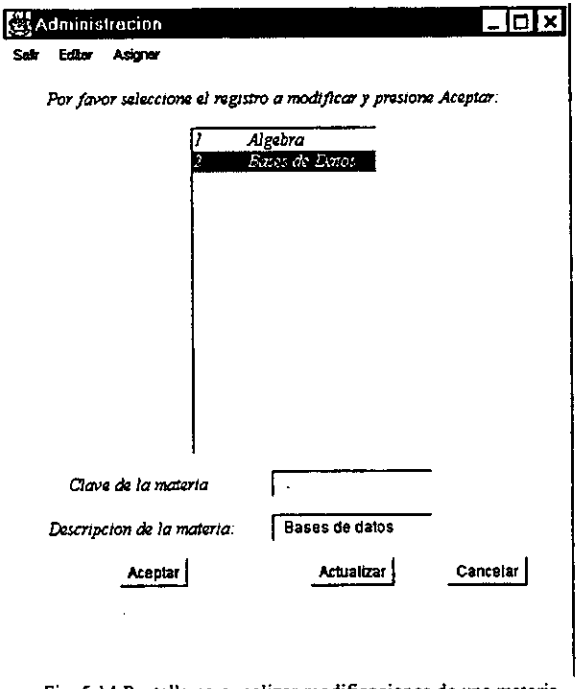



Fig. 5.14.Pantalla para realizar modificaciones de una materia.

Una vez modificada la materia aparece una caja de diálogo indicando que la operación se realizó exitosamente.

Si el administrador requiere realizar la alta de un grupo se presenta la siguiente pantalla (Fig.5.15.):

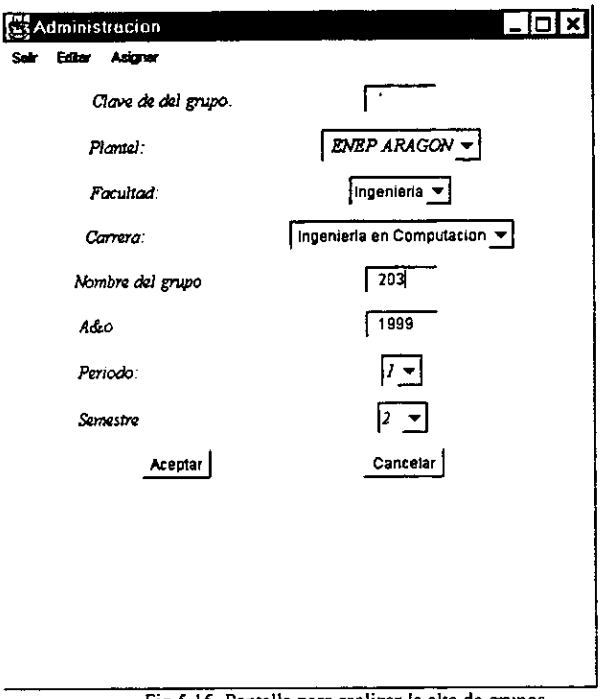



Fig.5.15. Pantalla para realizar la alta de grupos.

Es importante mencionar que el *Sistema de Evaluación* asigna automáticamente una clave de grupo. Esta clave es un número secuencial y único para cada grupo.

El usuario debe proporcionar a que plantel, facultad, y carrera pertenece el grupo, así como también el nombre del grupo, año, periodo y semestre al que será asignado.

Una vez dado de alta en el sistema aparecerá una caja de diálogo indicando que la operación se realizó exitosamente:

Para dar de baja un grupo se utiliza la siguiente pantalla Fig.5.16

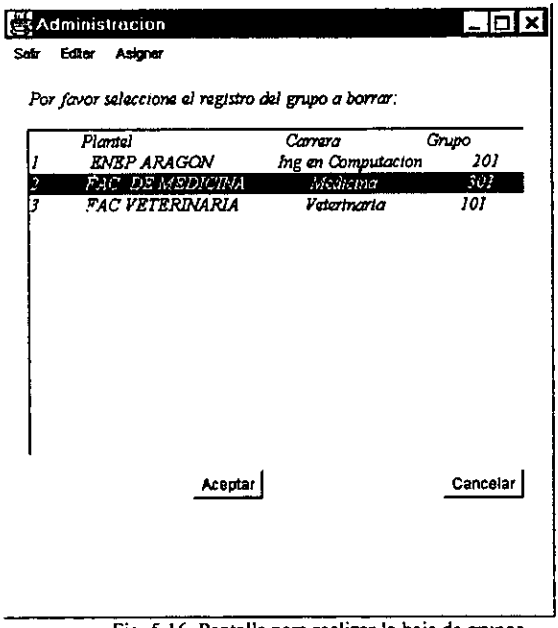



Fig. 5.16. Pantalla para realizar la baja de grupos.

En ésta pantalla se debe elegir el registro a borrar. Antes de borrar al grupo del sistema se pide confirmación al administrador como se muestra en la Fig. 5.17.

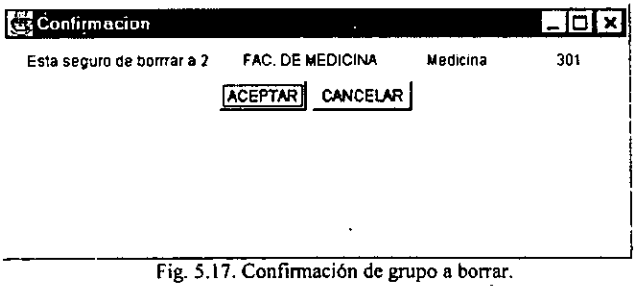



Fig. 5.17. Confirmación de grupo a borrar.

Si se requiere realizar la modificación de los datos de un grupo se muestra la siguiente pantalla (Fig. 5.18.) .

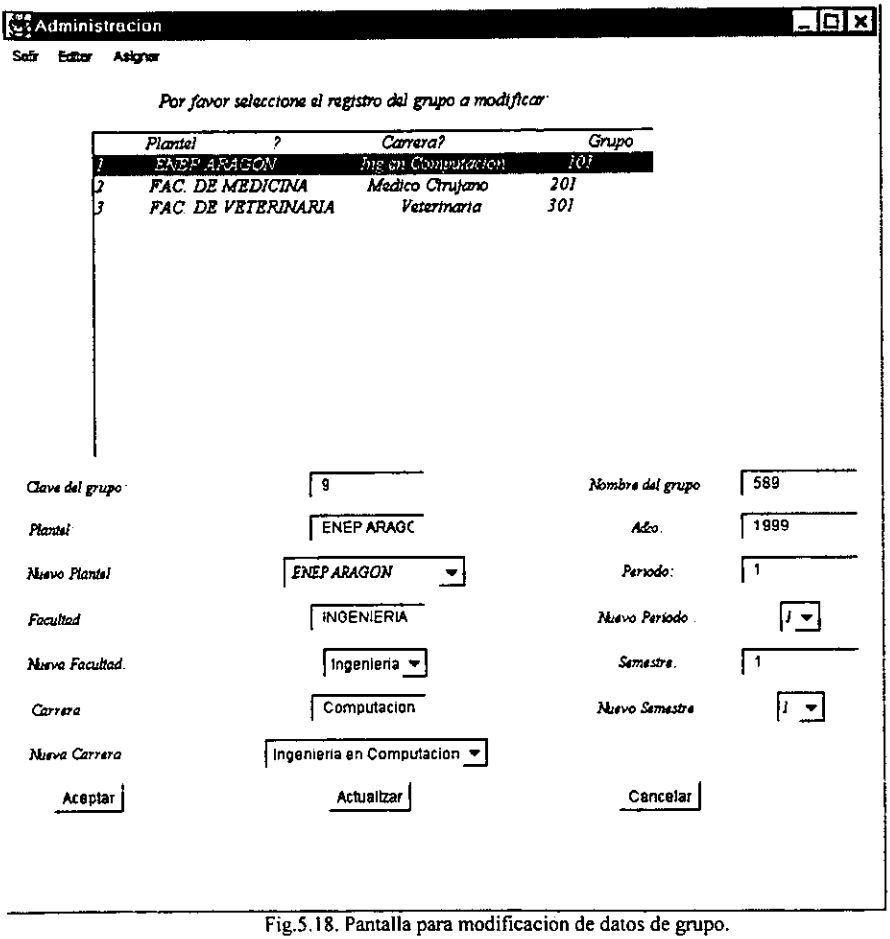



Fig.5.18. Pantalla para modificacion de datos de grupo.

En ella se debe elegir el grupo a modificar para mostrar sus datos y de esta manera realizar las modificaciones correspondientes.

Una vez modificado el grupo aparece una caja de diálogo con el mensaje *de "Grupo Modificado"*.

Para realizar la alta de un alumno al *Sistema de Evaluación* se utiliza la siguiente pantalla (Fig. 5.19.). En ella se deben anotar los datos del alumno, tales como nombre apellidos, dirección, teléfono, número de cuenta, nivel de escolaridad y un password con el que éste tendrá acceso al Sistema.

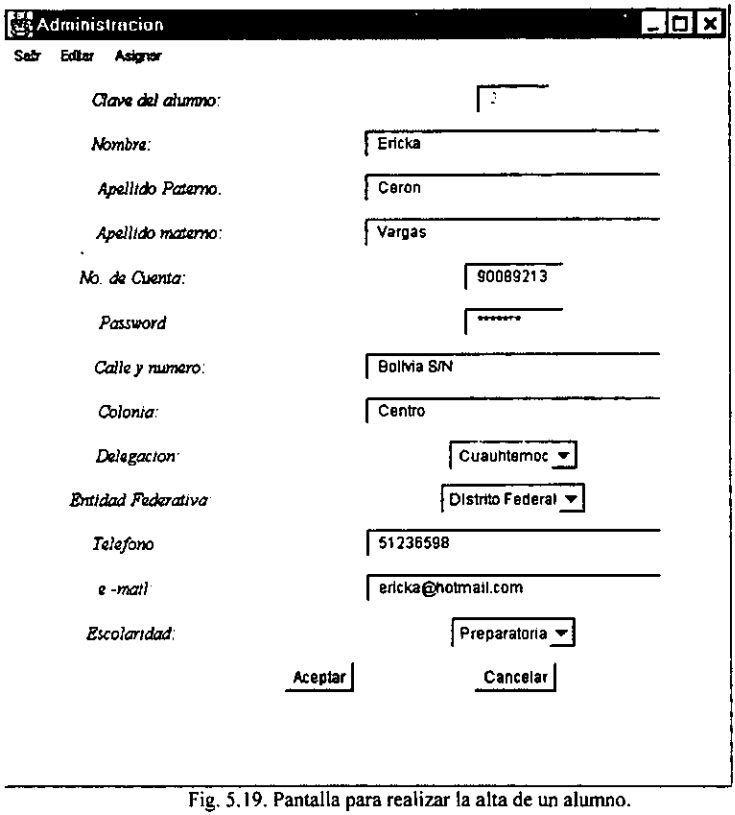



Fig. 5.19. Pantalla para realizar la alta de un alumno.

Una vez que se ha presionado el botón de *"Aceptar"*, se manda un mensaje avisando que el alumno ha sido dado de alta en el sistema.

Para dar de baja a un alumno se utiliza la siguiente pantalla (Fig. 5.20.), sólo se debe seleccionar el registro a borrar. Y presionar el botón "Aceptar"

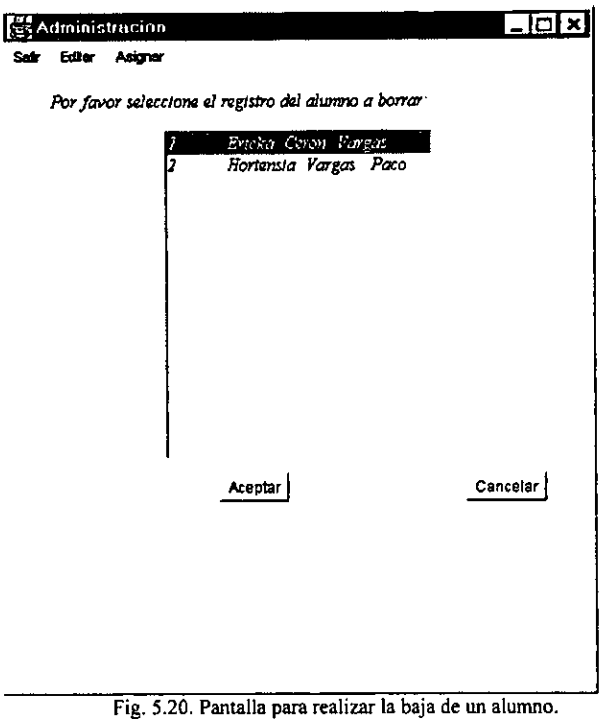



Fig. 5.20. Pantalla para realizar la baja de un alumno.

Antes de realizar la baja del alumno del *Sistema de Evaluación* se manda un mensaje pidiendo la confirmación.

Para la modificación de los datos de un alumno (Fig. 5.21), se selecciona el registro a modificar y se presiona el botón *"Aceptar"*, posteriormente se muestran los datos correspondientes al registro del alumno seleccionado y se modifican los datos necesarios .

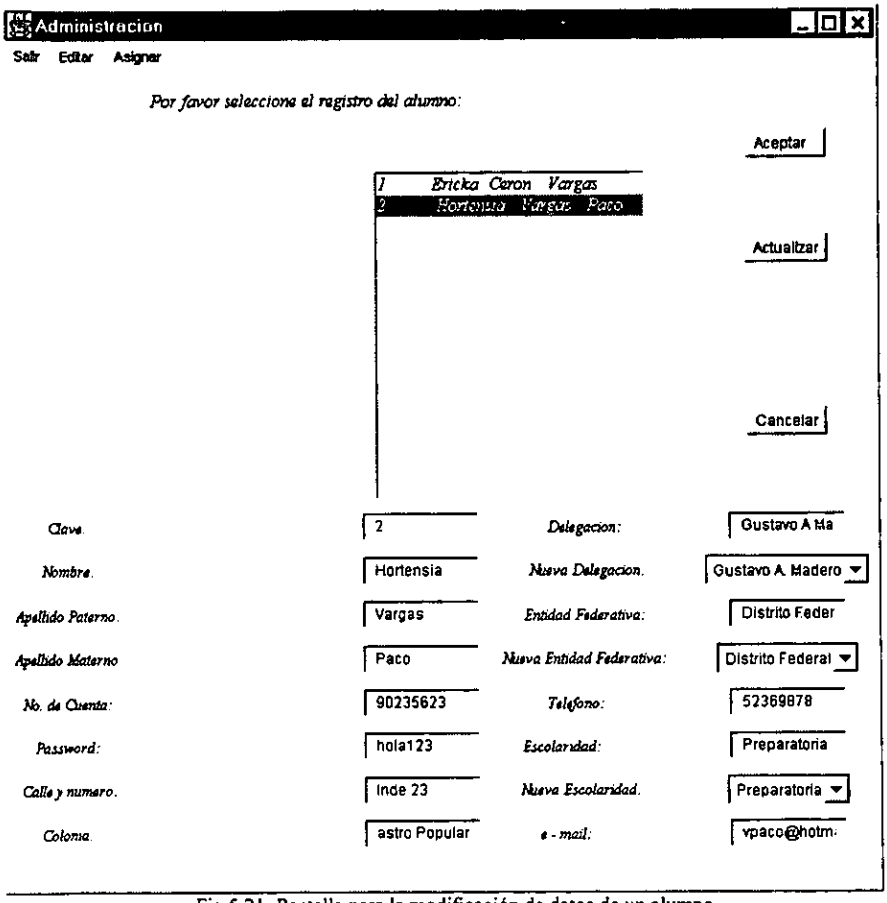



Fig.5.21. Pantalla para la modificación de datos de un alumno.

Una vez realizada la modificación se muestra una caja de diálogo, para indicar que ha sido modificado en alguno de sus datos.

Para la realización de la alta de un profesor se muestra la siguiente pantalla (Fig.5.22.)

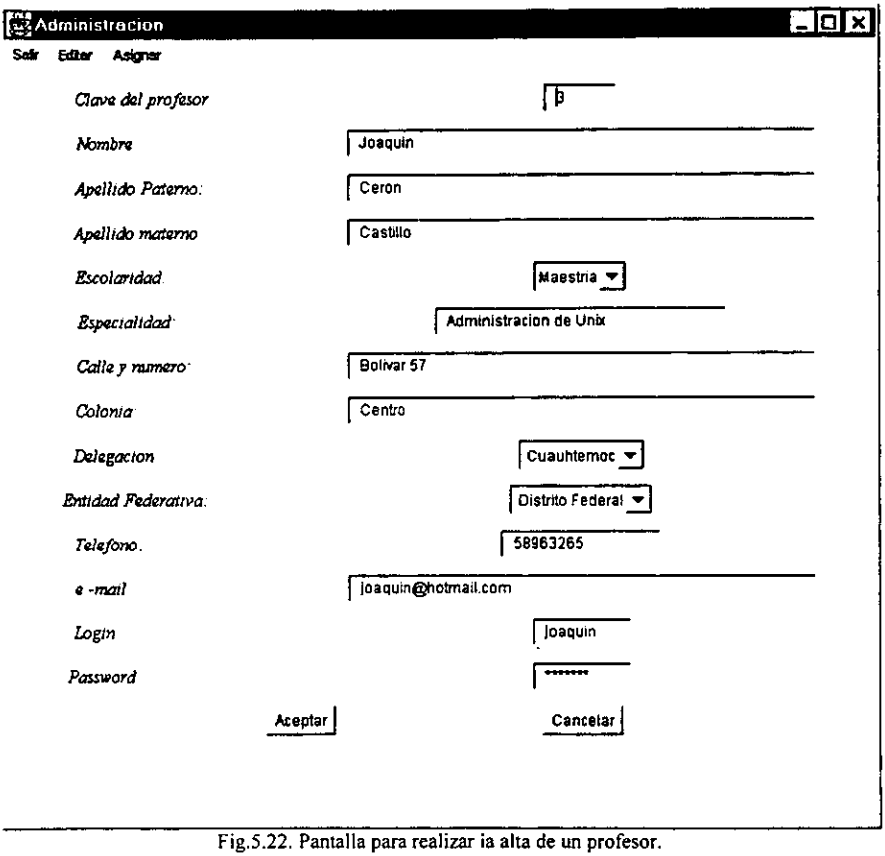



Fig.5.22. Pantalla para realizar ia alta de un profesor.

Una vez dado de alta en el sistema, se manda un mensaje al administrador de *"Profesor dado de alta"*.

Para realizar la baja de un profesor se utiliza la siguiente pantalla (Fig. 5.23.). El administrador solo debe seleccionar el registro a borrar y presionar el botón de *"Aceptar"*.

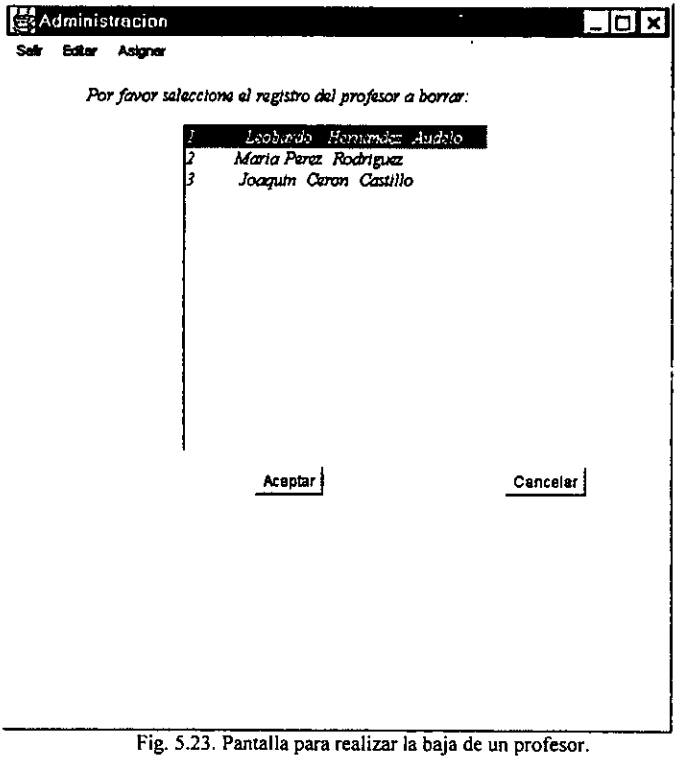



Fig. 5.23. Pantalla para realizar la baja de un profesor.

Antes de borrar el registro seleccionado, el sistema pide confirmación mediante una caja de diálogo.

Para realizar la modificación de alguno de los datos del profesor, se debe seleccionar el registro a modificar y presionar el botón de *"Aceptar"*, con el fin de que los datos correspondientes a dicho registro se muestren en la pantalla para ser modificados y finalmente se debe de presionar el botón *"Actualizar"*. (Fig. 5.24.)

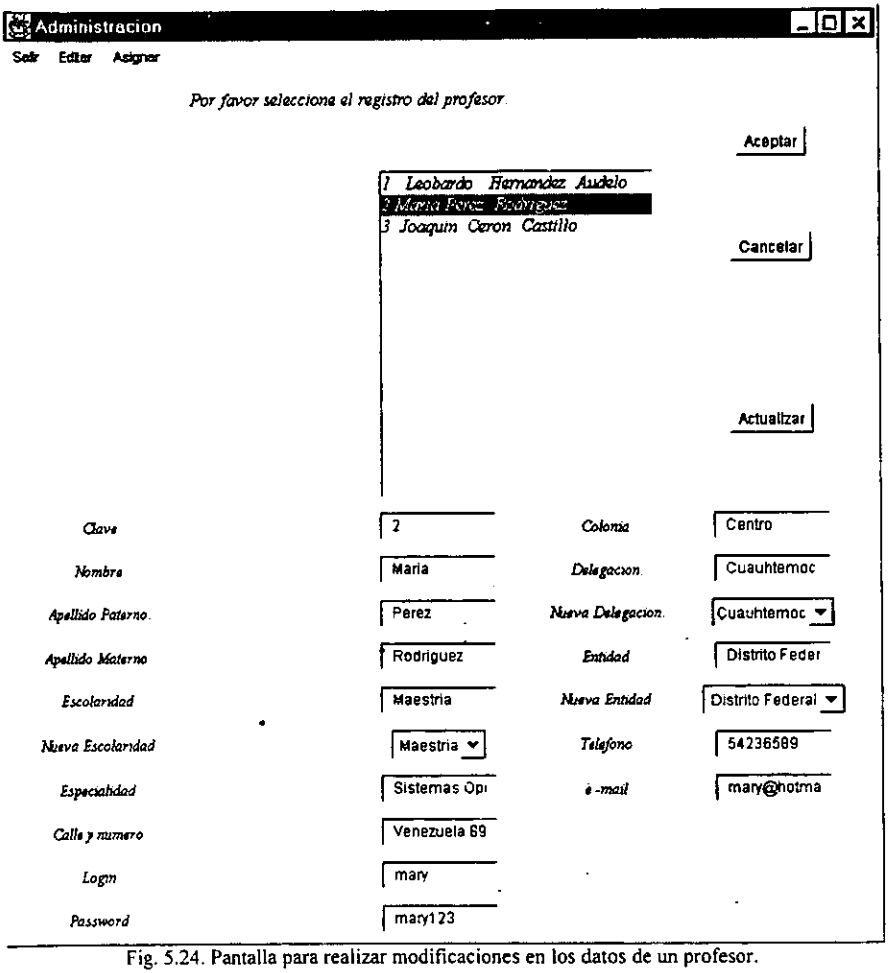



Fig. 5.24. Pantalla para realizar modificaciones en los datos de un profesor.

Para asignar un alumno a un grupo se realiza mediante la opción de *"Asignación de Alumnos"* (Fig.5.25) en la opción *"Asignar"* del menú principal

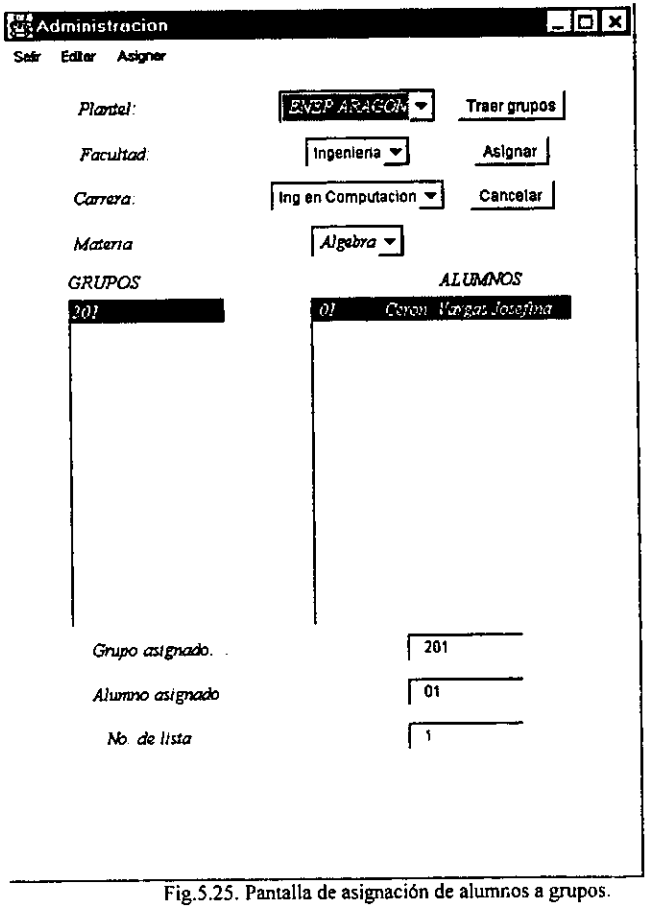



Fig.5.25. Pantalla de asignación de alumnos a grupos.

Para eliminar la asignación se utiliza la opción de *"Des-Asignación de Alumnos"* (Fig. 5.26)

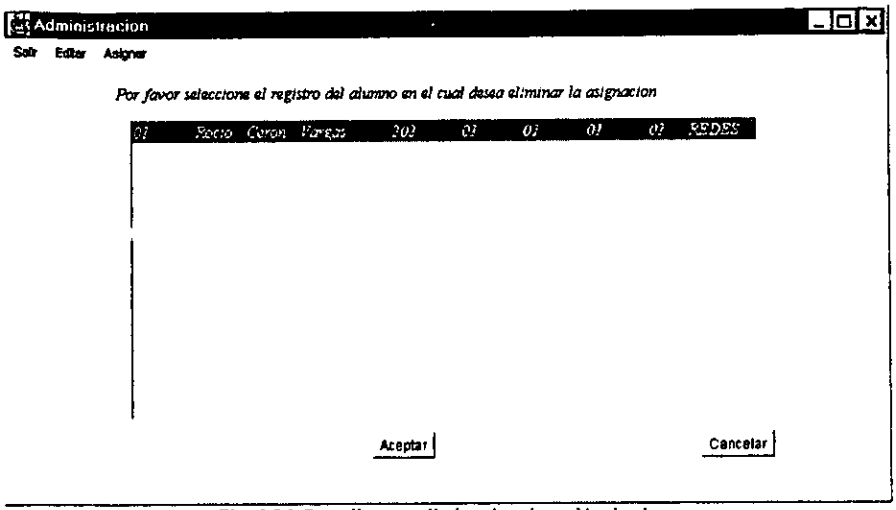



Fig. 5.26. Pantalla para eliminar la asignación de alumnos a grupos.

Si se requiere asignar un profesor a un grupo se utiliza la opción de *"Asignación de profesores"* y aparece la siguiente pantalla (Fig. 5.27). En ella se selecciona el plantel, la facultad, carrera, materia y grupo correspondiente.

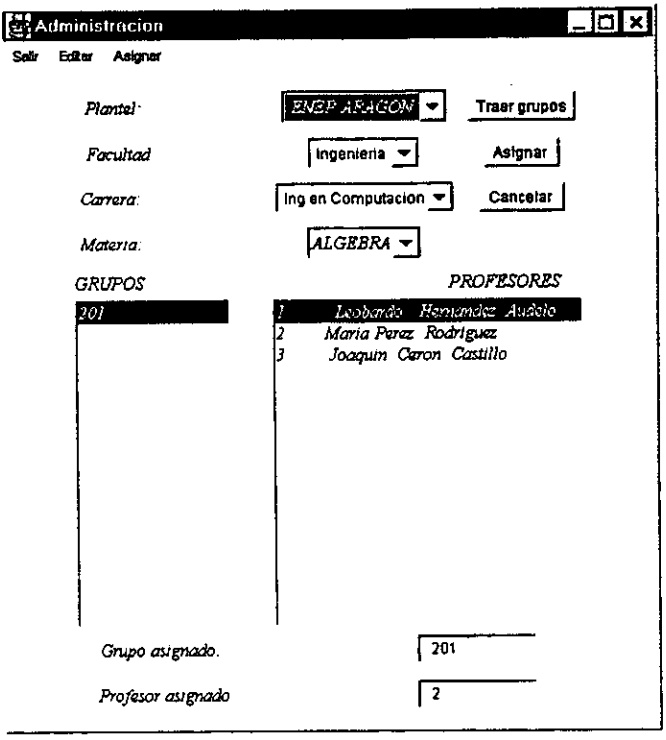



Fig.5.27. Pantalla de asignación de profesores a grupos.

Para eliminar la asignación de un profesor se utiliza la opción *"Des-Asigna Profesor"*, mediante la siguiente pantalla (Fig. 5.28.)

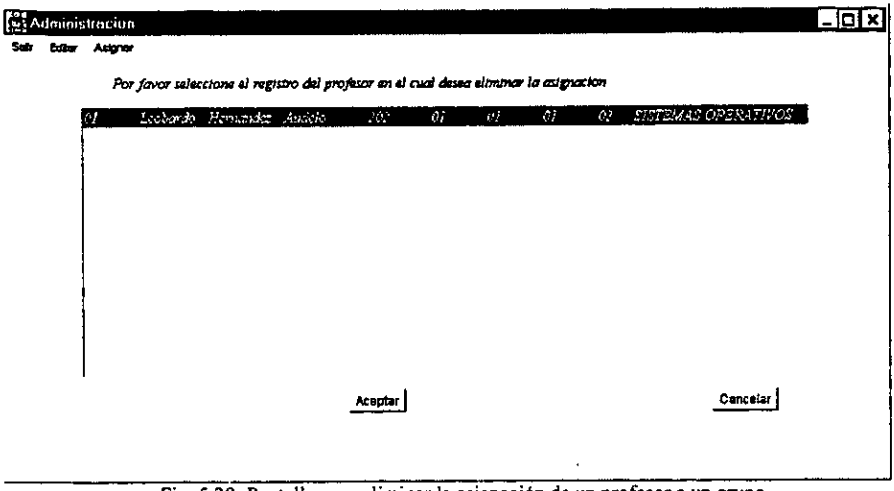



Fig. 5.28. Pantalla para eliminar la asignación de un profesor a un grupo.

Antes de eliminar la asignación pide confirmación mediante la siguiente caja de diálogo (Fig.5.29)

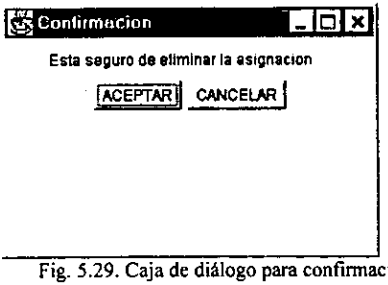



Fig. 5.29. Caja de diálogo para confirmación.

## 5.1.2 *Módulo del Profesor.*

Para acceder al módulo del profesor es necesario proporcionar login y password[65] mediante una pantalla como la que se muestra en la figura 5.30.

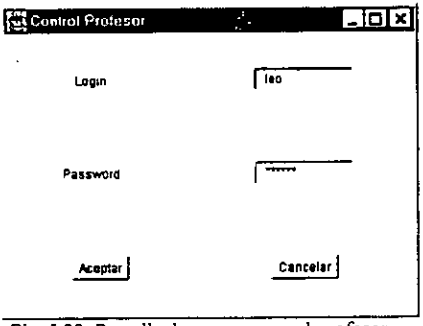



Fig. 5.30. Pantalla de acceso para el profesor.

Si los datos proporcionados no son correctos aparece una caja de diálogo[66] indicando que los datos son incorrectos o si no se proporcionaron se le solicitan nuevamente al profesor.

Si los datos son correctos, aparece en la pantalla el *Menú Principal* del profesor como se muestra en la figura 5.31.

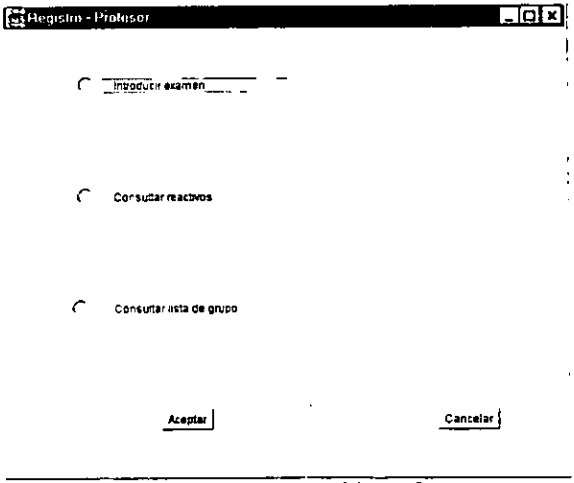



Fig. 5.31. *Menú principal* del profesor.

[65] *Estos son proporcionados por el administrador del Sistema de Evaluación.*
[66] *Como en el módulo del administrador*

Si se elige la opción de introducir examen aparece la siguiente pantalla (Fig. 5.32), en la cual se debe elegir el grupo correspondiente al examen.

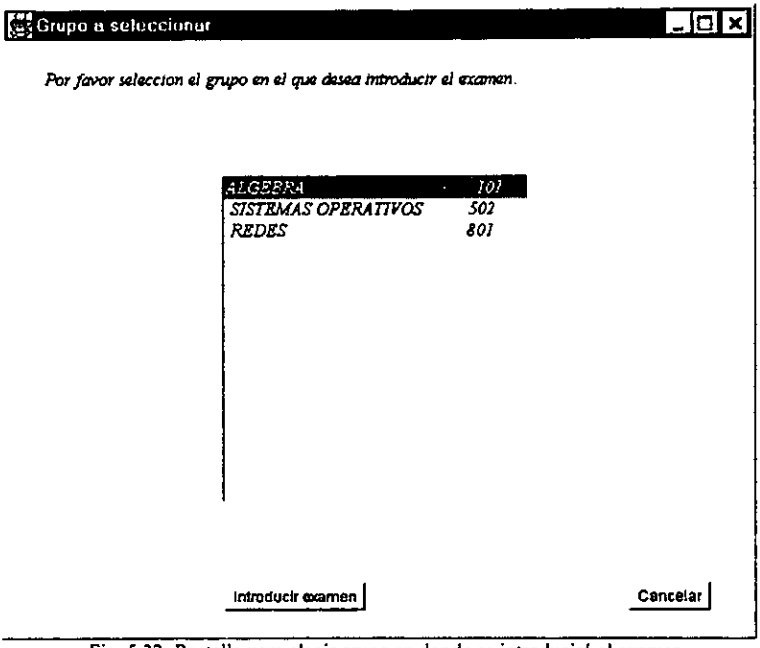



Fig. 5.32. Pantalla para elegir grupo en donde se introducirá el examen.

Una vez seleccionado el grupo se muestra la siguiente pantalla (Fig. 5.33) para introducir el número de preguntas y la descripción del examen

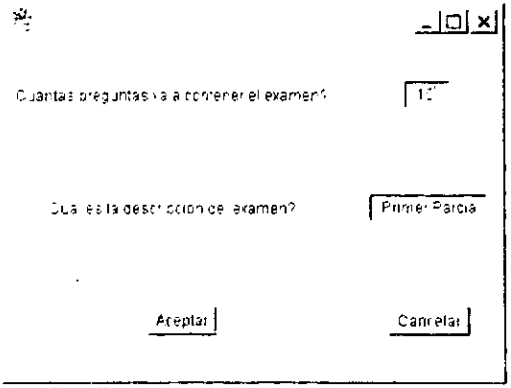



Fig. 5.33. Pantalla para introducir número de preguntas y descripción.

Una vez definidas, se presiona el botón *"Aceptar"* y se procede con la captura de cada una de las preguntas con sus respectivas respuestas y valores de las mismas, mediante la siguiente pantalla Fig. 5.34.

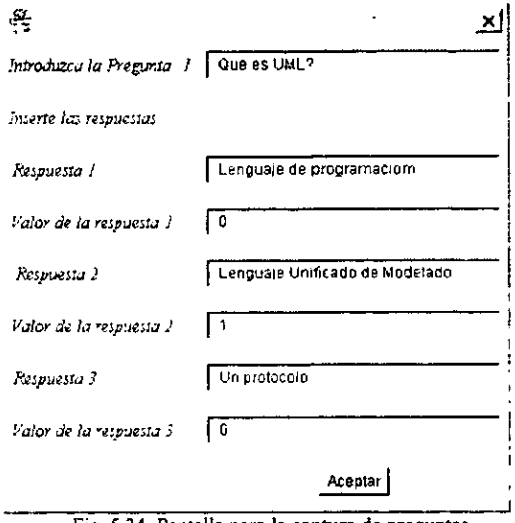



Fig. 5.34. Pantalla para la captura de preguntas.

Una vez introducido el examen aparece un caja de diálogo como la Fig. 5.35.

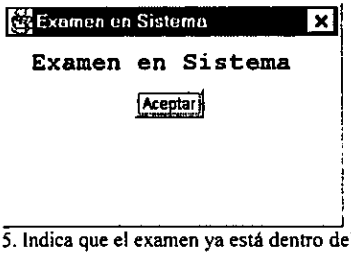



Fig. 5.35. Indica que el examen ya está dentro del sistema.

Si en el menú principal del profesor se elige la opción de *"Consultar Reactivos"* se muestra la siguiente pantalla (Fig. 5.36.), la cual presenta las opciones de *"Traer Examen"*, *"Eliminar Examen"* y *"Cancelar"*, así como, una lista de todos los exámenes que correspondan al profesor cuya clave de acceso corresponda al profesor conectado en el sistema.

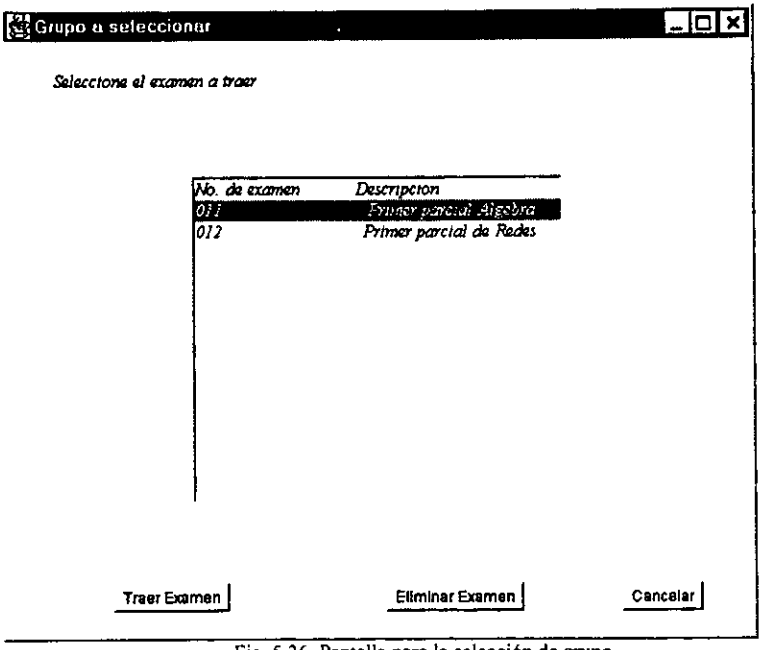



Fig. 5.36. Pantalla para la selección de grupo.

Si elige la opción de *"Eliminar Examen"*, el examen seleccionado es borrado de la base de datos junto con las preguntas asociadas.

Si elige la opción de *"Traer examen"* se muestra la siguiente pantalla (Fig. 5.37), en la cual el profesor proporciona la clave de la pregunta y presiona el botón *"Traer Pregunta"* para que la pregunta, junto con sus respuestas y sus valores se muestren y se puedan modificar.

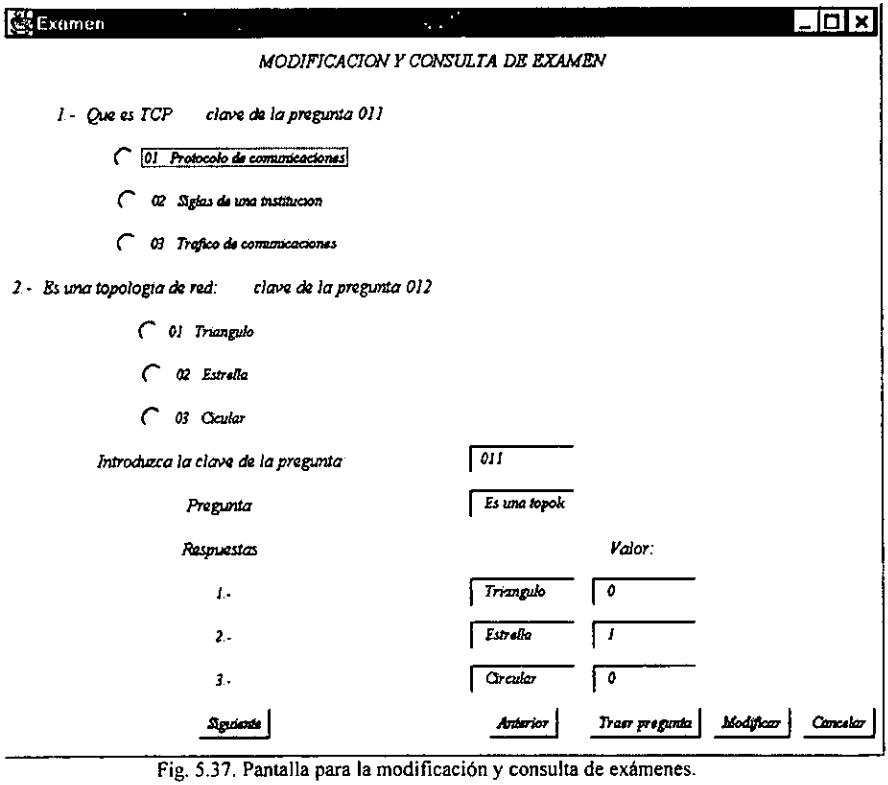



Fig. 5.37. Pantalla para la modificación y consulta de exámenes.

Si se elige la opción *"Consulta Lista de Grupo"* del menú principal del profesor aparece la siguiente pantalla. (Fig. 5.38.). En ésta pantalla se debe presionar el botón de *"Traer Grupos"*, para que muestre todos los grupos correspondientes de acuerdo al plantel, facultad, carrera y materia que haya proporcionado el profesor.

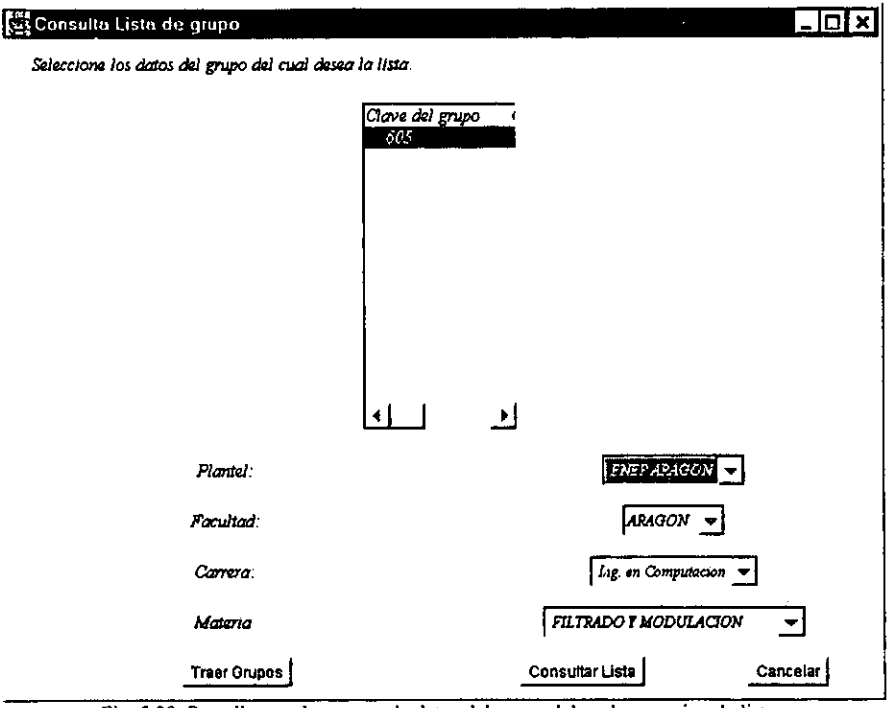



Fig. 5.38. Pantalla para la captura de datos del grupo del cual se requiere la lista.

Una vez mostrados los grupos, se elige uno de ellos para consultar la lista y aparece la siguiente pantalla (Fig. 5.39.)

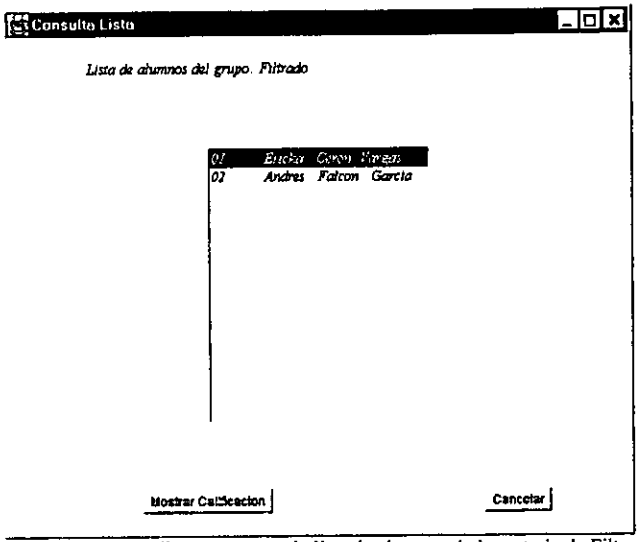



Fig. 5.39. Pantalla que muestra la lista de alumnos de la materia de Filtrado.

En esta pantalla es posible mostrar la calificación del alumno presionando el botón *"Mostrar Calificación"* para que aparezca la siguiente pantalla. (Fig. 5.40)

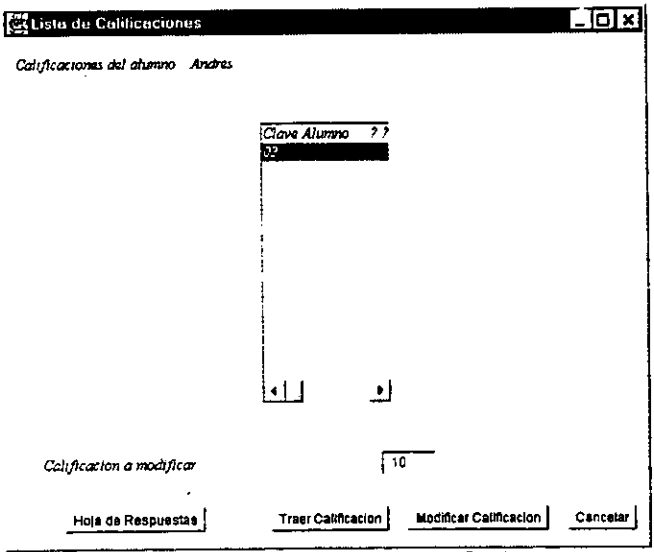



Fig. 5.40.Pantalla para mostrar la calificación.

En la Fig.5.40. se muestran todas las calificaciones correspondientes al alumno seleccionado en la Fig., 5.39. en esta pantalla es posible modificar la calificación del alumno y mostrar también su hoja de respuestas, mediante el uso del botón *"Hoja de Respuestas"*, el cual al ser presionado muestra la siguiente pantalla. (Fig. 5.41)

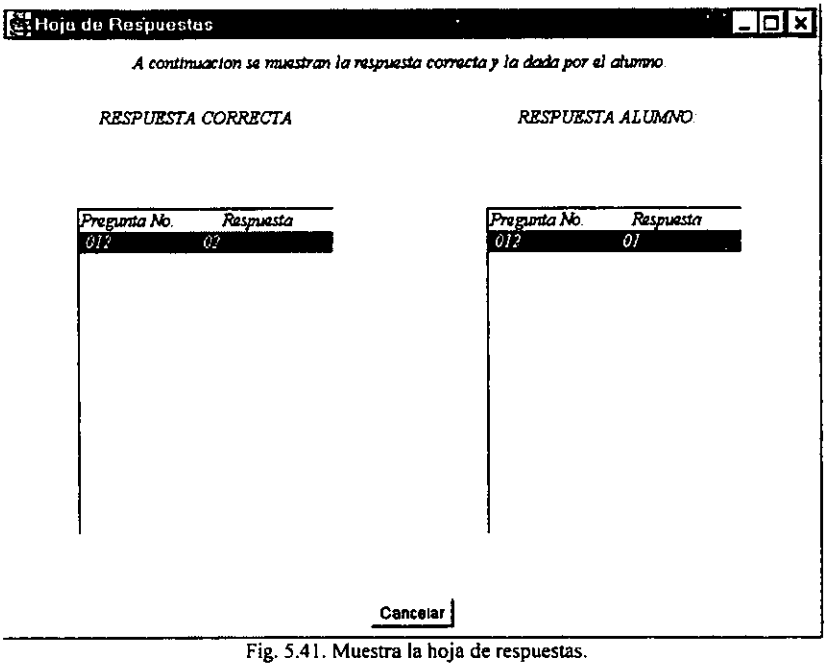



Fig. 5.41. Muestra la hoja de respuestas.

Esta pantalla es muy útil cuando se requieren hacer aclaraciones con el alumno, ya que el sistema muestra de un lado la respuesta correcta y del otro la respuesta dada por el alumno.

## 5.1.3 *Módulo del Alumno.*

Para acceder al módulo del alumno es necesario proporcionar número de cuenta y password mediante una pantalla como la que se muestra en la figura 5.42.

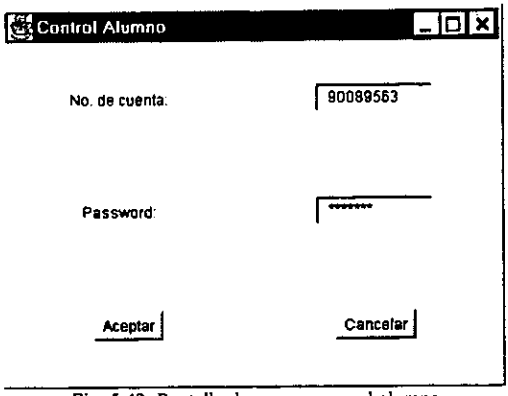



Fig. 5.42. Pantalla de acceso para el alumno.

Una vez que se han proporcionado los datos correctos, se muestra una pantalla (Fig.5.43.) en la cual están todos los grupos de las distintas materias en las que se encuentra inscrito. En ésta pantalla debe elegir un grupo en el cual va a realizar el examen y presionar *"Aceptar"*.

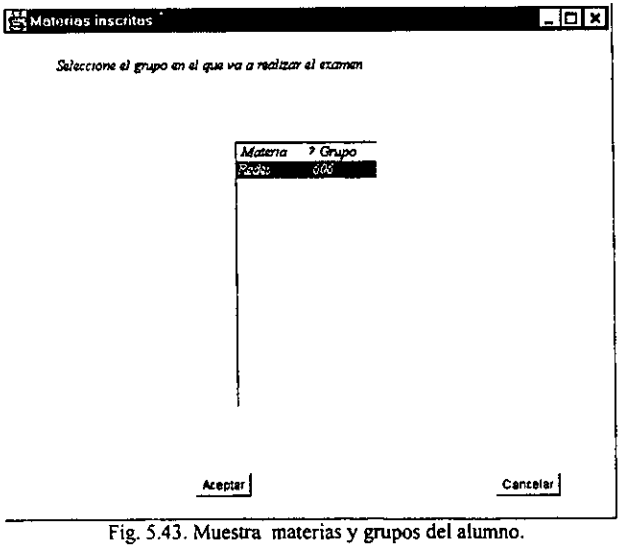



Fig. 5.43. Muestra materias y grupos del alumno.

Después el sistema le preguntara la *clave del examen*[67] mediante la siguiente pantalla (Fig. 5.44.)

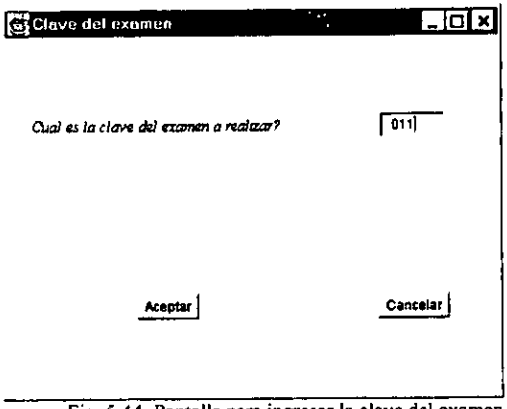



Fig. 5.44. Pantalla para ingresar la clave del examen.

Si la clave proporcionada es correcta, entonces se muestra el examen a contestar. En éste examen (Fig. 5.45) el sistema muestra en grupos de tres las preguntas y sus correspondientes opciones de respuesta. El alumno puede navegar hacia delante o hacia atrás para ver todas las ternas de preguntas.

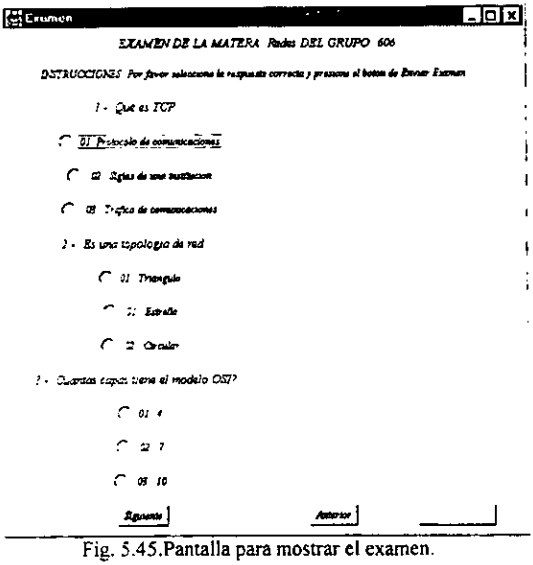



Fig. 5.45.Pantalla para mostrar el examen.

[67] *Esta clave es previamente proporcionada por el profesor .*

## 5.2 Pruebas del Sistema

Una vez terminada la programación del *Sistema de Evaluación* fue necesario definir un plan de pruebas con el propósito de conocer el desempeño de éste frente a los usuarios. Además de conocer diferentes opiniones sobre su implementación, uso y funcionalidad.

Es necesario mencionar que el trabajo de programación se realizó en un servidor *SUN* con sistema operativo *Solaris 2.6* y en *Windows 95*.

Las pruebas se hicieron en computadoras con *Windows 95 y 98,* con navegadores *Explorer* y *Netscape*. Se eligió esta plataforma para trabajar con el *Sistema de Evaluación* debido a que es la más comercial para los equipos cliente.

**Plan de pruebas del sistema.**

El plan de prueba se divide en tres etapas, una para el módulo del administrador, otra enfocada al módulo del profesor y una última dirigida al módulo del alumno.

**Administrador.**

En cuanto al módulo del administrador se solicitó ayuda a un compañero familiarizado con los sistemas de información y la administración de los mismos. El plan de pruebas consistió en :

- Se le proporcionaron por escrito una lista de planteles, materias, grupos, profesores y alumnos para dar de alta, baja y modificar en el sistema.
- Se le mostró la página principal del *Sistema de Evaluación* y la ayuda en línea del módulo del administrador.
- Se le proporcionó un login y password para el acceso al *Sistema de Evaluación* .
- Finalmente se procedió a realizar las altas, bajas y modificaciones de los planteles, materias, grupos, profesores y alumnos proporcionados en el primer punto.
- Se le pidió su opinión acerca del *Sistema de Evaluación* .

**Profesor.**

En cuanto al plan de pruebas del profesor se solicitó la ayuda de un profesor de nivel medio superior, para que realizara la introducción de un examen de 10 preguntas generales a través de *Internet*. El plan de pruebas consistió en:

- Explicarle al profesor el propósito y funcionamiento del sistema y mostrarle la ayuda en línea.
- Proporcionarle un login, password, grupo y materia para introducir el examen.
- Finalmente el profesor procedió a la introducción del examen.
- Se le pidió su opinión acerca del *Sistema de Evaluación* .

**Alumno.**

El plan de pruebas consistió en conseguir un grupo de 5 personas dispuestas a realizar un examen a través de *Internet*. Es preciso mencionar que el lugar de pruebas fue un *Café Internet*, solicitando a estudiantes de diferentes carreras, todos de nivel superior, realizaran un examen de conocimientos generales de 10 preguntas.

El plan de pruebas consistió de los siguientes puntos:

- Primeramente se les explicó brevemente el propósito y funcionamiento del sistema. Se les proporcionó a cada uno un número de cuenta y un password para el acceso, además de un grupo y una clave de examen.
- Después se hizo un preexamen de prueba para asegurar que cada uno de ellos había comprendido las instrucciones. Durante éste preexamen de pruebas la opción de pedir ayuda estaba permitida.
- Una vez terminado el preexamen de prueba. Se procedió a realizar el examen. Durante la realización de este examen los alumnos no tuvieron ningún tipo de ayuda, excepto la ayuda proporcionada en línea.
- Finalmente cada uno de los estudiantes dio su opinión acerca del *Sistema de Evaluación*.

## 5.3 Conclusiones de las pruebas del Sistema.

El resultado de las pruebas del sistema fue favorable para este debido a que todas la opiniones fueron constructivas.

La persona que realizó las actividades del administrador, dijo que el sistema estaba estructurado organizadamente y permitía realizar las funciones de administración fácilmente.

En cuanto al profesor, éste mostró un gran interés en la forma en la que trabaja el sistema y manifestó su opinión argumentando que la tecnología actualmente está creciendo

a pasos agigantados y esto del *Internet* también. Mencionó que es una buena forma de aprovechar el tiempo al realizar exámenes fuera de clase y durante las clases ordinarias explicar temas. Además le pareció muy útil que el sistema calculara la calificación del alumno y así él tuviera la opción de modificar la calificación de éste, ya que algunas veces considera que los alumnos ganan 1 ó 2 puntos más si realizan las tareas y trabajos a lo largo del curso.

Por otro lado los estudiantes opinaron que es una buena forma de realizar exámenes y a ninguno se le dificultó el funcionamiento del sistema. Además les pareció de gran ayuda que el sistema proporcionara la calificación inmediatamente y no tuvieran que esperar a la siguiente clase para conocerla.

# Capitulo 6. Conclusiones.

Durante el desarrollo de esta aplicación, se presentaron algunos problemas que deben tomarse en cuenta durante el análisis de cualquier sistema, ya que esto afectará directamente el desarrollo y posteriormente la implementación. Estos problemas son de orden tecnológico y económico principalmente como se detalla a continuación.

Dentro de los problemas del ámbito económico, hubo la necesidad de delimitar el alcance de la aplicación debido a que los requerimientos que en un principio se contemplaban para el *Sistema de Evaluación* iban mas allá del propósito de esta tesis y esto implicaba el consumo de un mayor número de recursos humanos y materiales.

Dentro de los problemas de orden tecnológico se tiene por un lado, el vertiginoso avance de las tecnologías, que en este caso se vio claramente reflejado en el constante cambio de versiones del lenguaje *Java* y en la visualización de los *applets* dentro de los diferentes *navegadores*. De forma particular éste problema se solucionó utilizando *plugins*[68], los cuales permiten visualizar dentro de cualquier *navegador* un *applet* de la misma forma en la que se observaría en el *appletviewer*[69] sin importar la plataforma. Para hacer uso de los *plugins* fue necesario instalar un convertidor de páginas *HTML* llamado "*RunHTMLConverter*", que es el encargado de incluir el *plugin* dentro de la *navegador* y visualizar perfectamente los *applets* del sistema.

Este problema de instalación de software adicional al *JDK* ("*Java Developer Kit*") debe tomarse en cuenta para el desarrollo de sistemas y para la evaluación de los pros y contras en la selección del lenguaje, de acuerdo a la aplicación que se quiera realizar, ya que ésta adición elimina una de las principales características de *Java*, la multiplataforma, debido a que los *plugins* dependen enteramente del sistema operativo de la máquina en que se deban de instalar. Sin embargo, esto se puede solucionar proporcionándole al usuario una lista de *plugins* para diferentes sistemas operativos y así él elija el que se adecue al suyo propio. De ésta forma se garantiza la funcionalidad del sistema sin modificar nada en cuanto a la programación; también se puede solucionar comprando de algún vendedor externo librerías escritas cien por ciento en *Java* ó pagar por el uso de aplicaciones distribuidas. Otro ejemplo similar fue la instalación del *JDBC Thin Driver* de *Oracle*, el cual contiene las clases escritas en *Java* puro, que permiten realizar la conexión ligándolas con los programas de la aplicación en *Java* y con el manejador de base de datos.

De lo anterior, se deriva ahora otro problema con el cual se enfrentó durante el desarrollo de esta tesis. Si bien existen soluciones alternas, éstas implican un costo adicional, que por lo general es alto. Por otro lado, aunque el kit de desarrollo de *Java* es de distribución gratuita, el manejador de la base de datos *Oracle*, los drivers para la conexión, la infraestructura necesaria y los certificados para garantizar la seguridad de los *applets* no lo son.

[68] *Un plugin es un programa externo, que se instala en el cliente y se invoca por el navegador.*
[69] *El appletviewer es el visualizador nativo de Java para* applets.

En conclusión podemos decir, que de forma personal el estudio del lenguaje *Java* me ha sido de gran utilidad en el ámbito laboral ya que las tendencias de las aplicaciones del tipo *eBusiness*[70], *eComerce*[71], *DotCom*[72], *B2B*[73], *CRM*[74], *etc.* que son términos acuñados para los diferentes tipos de aplicaciones de negocios sobre *Internet* se están desarrollando en su mayoría en *Java.* También de manera pretenciosa se espera que éste trabajo sirva como material de apoyo y consulta para personas con inquietudes en el desarrollo de sistemas en lenguaje *Java* que interactúen con un manejador de base de datos.

[70] *Aplicaciones de negocios sobre Internet.*
[71] *Aplicaciones de comercio electrónico sobre Internet.*
[72] *Empresas con aplicaciones varias sobre Internet (.com).*
[73] *Aplicaciones de negocios a negocios (*"Busines to Busines"*).*
[74] *Aplicaciones de manejo de relaciones con el cliente (*"Customer Relationship Manager"*)*

# Referencias.


[1] Cohn M., Morgan B., Morrison M., Nygard M., Joshi D., TrinkoT.; *Java Developers's Reference*, Sams Net, USA, 1996

[2] Waite M., Lafore R.; *Object Oriented Programming in Java*, The Waite Group Inc., USA, 1997

[3] Hobbs A; *Teach Yourself Java in 21 Days*, Sams Net, USA, 1996

[4] http: //whatis.com/*java.HTML*

[5] Olse A.; *Java Programming for Oracle Developers*, USA, 1998

[6] Mark C., Steven W., Griffith, Anthony F.; *La guía más completa para programadores de Java y Visual J −+ 10001 Tips para programar con Java*, Mc. Graw Hill, USA, 1997.

[7] Jaworski J.; *Java Guía de desarrollo*, Prentice Hall, España, 1997

[8] Arnold K., Gosling J.; *El lenguaje de Programación Java*. Addison Wesley, España, 1997

[9] http://www.dcen.uson.mx/*Java*Tut/Cap1/interf.*HTML*

[10] http://carpanta.dc.fi.udc.es/~mosky/docs/members.es.tripod.de/froufe/introduccion/indice1.*HTML*

[11] http://www.iec.csic.es/criptonomicon/*java*/*java.HTML*

[12] http://www.moon.act.uji.es/~ancacam/pfc/tutor/*java*/index.*HTML*

[13] http://*java*.sun.com/products/*JDK*/1.1/installation-*Solaris*2.*HTML*#install

[14] http://www.cab.u-szeged.hu/WWW/*java*/*JDK*/docs/relnotes/features.*HTML*

[15] http://*java*.sun.com/docs/books/jls/*HTML*/index.*HTML*

[16] http://tns.sdsu.edu/~diebel/*JDBC*/*JDBC*oci2.htm

[17] http://witss.gdl.iteso.mx/gunter/componen.*HTML*

[18] http://www.fie.us.es

[19] http://usuarios.bitmailer.com/jblazquez/cec/historia.*HTML*#arquitect

[20] http://www.*java*sun.com/products/*JDBC*/*JDBC*.drives.*HTML*

[21] Hobbs A.;*Aprendiendo programación para bases de datos con JDBC en 21 días*, Prentice Hall, México, 1998

[22] Booch G., Rumbaugh J., Jacobson I.; *El Lenguaje Unificado de Modelado*, Addison Wesley, España, 1999

[23] McConnel S; *Desarrollo y Gestión de Proyectos Informáticos*, McGraw Hill, España, 1997

[24] Murria P.; *Modelado de Objetos con UML*, Eyrolles, España, 1997

[25] Larman C.; *UML y Patrones*, Prentice Hall, México, 1999

[26] Whitten J.; Bentley L., Barlow V.; *Análisis y Diseño de Sistemas de Información*, MacGraw Hill, Colombia, 1997

[27] Rumbaugh J., Blaha M., Premerlani W., Hedí F., Lorensen W.; *Modelado y Diseño Orientado a Objetos*, Addison Wesley, España, 1997

[28] *Oracle* Corporation; *Introducción a Oracle: SQL and PL/SQL Using Procedure Builder, Oracle* Education México, 1996

[29] http://oracle.com

[30] Nicolas C., Avare C., Najman F.; *Java Cliente – Servidor*, Eyrolles, España, 1998

[29] http://aries.dif_im.es