Implementación

CAPÍTULO 4

CAPÍTULO 4 – Implementación

En este capítulo se especifican los detalles del desarrollo, las herramientas utilizadas y los diagramas de paquetes. Todas las clases fueron programadas específicamente para este proyecto excepto la clase de LuceneIndexer tomada de las notas del curso de Administración de la información de Carlos Proal [Proal, 2005]

4.1 Componentes utilizadas durante el desarrollo.

Para que la aplicación funcione adecuadamente es necesario contar con los siguientes componentes:

- Hibernate 3.0.5. Se utilizó para la comunicación entre el lenguaje de programación Java y la base de datos en este caso MySQL Server 4.1.14. Disponible para su descarga en www.hibernate.org. Posteriormente se generaron las clases y archivos de mapeo de los objetos a las tablas, se explicarán más a detalle posteriormente.

- Apache Tomcat 5. De acuerdo al contexto de nuestra aplicación es necesario un contendor de servlets y JSPs, estos últimos son especificaciones de Sun Microsystems para entorno web. Los *servlets* son programas desarrollados en Java. Para el caso de los JSPs, Tomcat cuenta con un compilador *Jasper* que se encarga de compilarlos para convertirlos en servlets. Cabe destacar que Apache es el servidor web y Tomcat es el contenedor. Esta escrito en Java por lo que funciona en cualquier sistema operativo que disponga de una maquina virtual. Está disponible para su descarga en www.apache.org. Para la configuración necesaria para la interacción con *Hibernate* véase capítulo 1.2.3

- MySQL Server 4.1.14 (apple-darwin 8.2.0), este componente ofrece un servidor de bases de datos SQL. Es el encargado de almacenar los datos persistentes de la aplicación. Disponible para su descarga desde www.mysql.com. Para el funcionamiento de este servidor es necesario obtener el conector de Java a MySQL. Disponible en la misma URL mencionada anteriormente, veáse capítulo 1.2.3

- Apache Lucene 1.4.3: Tal como se mencionó anteriormente en el capítulo 2, es una herramienta de RI. Se encarga de dos funciones, la primera es crear un documento integrado propiamente del código fuente del programa, descripción y nombre del usuario que guardó el programa. Cabe aclarar que Lucene se encarga de actualizar convenientemente sus archivos de índices. Y la segunda función es realizar consultas sobre los documentos generados. Por ser un API de Java es necesario incluirlo en el ext de nuestro compilador de Java. Disponible para su descarga en: http://lucene.apache.org

- Apache FileUpload 1.1. Este API es el encargado de efectuar la tarea de subir los archivos al servidor. Este API es implementado en la clase GuardarProgramaController en las siguientes líneas de código:

```
// Se crea los objetos capaces de parsear la petición
FileItemFactory factory = new DiskFileItemFactory();
ServletFileUpload upload = new ServletFileUpload(factory);

// Número máximo de bytes que se permitirán en el archivo
upload.setSizeMax(yourMaxRequestSize);

// Se recibe la lista de los archivos a subir
List /* FileItem */ items = upload.parseRequest(request);

// Se crea el archivo físico en el servidor
File uploadedFile = new File(...);
```

```
// Se escribe el archivo al disco
    item.write(uploadedFile);
```

Este API utiliza a su ves utiliza otra herramienta auxiliar “commons-io-1.1.jar” estas dos se encuentran disponibles para su descarga en http://jakarta.apache.org/commons/fileupload/

Para colorear el texto se buscaron varias alternativas para lograr este objetivo, primero se pensaba implementar a mano el *parser* necesario para obtener el texto de manera que se quería, se observó la existencia de herramientas gratuitas que ofrecían esta utilidad. Después de consultar algunas de estas herramientas se optó por utilizar la siguiente por su facilidad de implementación:

- Java2Html 5 de Markus Gebhard. Es un API de Java disponible para su descarga en http://www.java2html.de/

```
/* Se crea una instancia de PrintWriter para escribir el
archivo*/
PrintWriter pr = new PrintWriter(htmlFile);

// se asignan las opciones de conversión del archivo
JavaSourceConversionOptions opciones =
JavaSourceConversionOptions.getDefault();
opciones.setShowLineNumbers(true);//mostrar número de línea
opciones.setShowFileName(true); //mostrar nombre

/* para convertirlo se manda un String con el código Java
y regresa un String en formato HTML */
text = Java2Html.convertToHtml(text, opciones);
```

```
// se imprime en el archivo y se cierra.
pr.println(text);
pr.close();
```

- Java jdk1.5 release3. Componente indispensable para la ejecución y desarrollo de aplicaciones Java. Contiene las herramientas de desarrollo como por ejemplo el compilador (*javac*) y el depurador o *debugger* (*jbd*). Además de ser utilizado en el desarrollo se utiliza para compilar los programas a través de un proceso *bash* que es llamado en tiempo de ejecución.

```
// se crea el string con el comando a realizar
String compilar = "javac " + allRuta;

// se se crea un runtime para poder ejecutar el proceso
Runtime rt = Runtime.getRuntime();
Process proc = rt.exec(compilar);
```

- Macromedia Dreamweaver MX 2004. Se utilizó para la creación de las formas y los archivos JSPs del sistema. Se incluyen unas animaciones realizadas en Macromedia Flash MX 2004. Estos no son software gratuito, pero se puede obtener una versión de prueba. Disponible en http://www.macromedia.com/

- Borland JBuilder 2005. Es un sofisticado y poderosa entorno de desarrollo (IDE), ayuda a la generación automática de código, como de diagramas de clase, permite ver el código con mayor legibilidad y estructura. Disponible en: http://www.borland.com/us/products/jbuilder/index.html

4.2 Procesamiento de Archivo de Java

El procesamiento del archivo de Java es el que tiene una interacción con la mayoría de las herramientas como se observa en la figura 4.1. Se obtiene el archivo del disco local del usuario a través del FileUpload y lo guarda en el sistema de archivos del disco duro del servidor. Al tener el archivo lo trata de compilar a través de un proceso bash, llamado por la aplicación, en caso de tener algún error de compilación borra el archivo y notifica al usuario de lo que sucedio, en el caso contrario a este archivo se le aplica un *parser* con la herramienta Java2Html para generar el código coloreado; al obtener el código coloreado en un archivo en formato html, se agrega el programa de Java al índice de Lucene para futuras consultas y finalmente se almacena la información relacionada con el programa en la base de datos del sistema a través de la interacción con Hibernate.

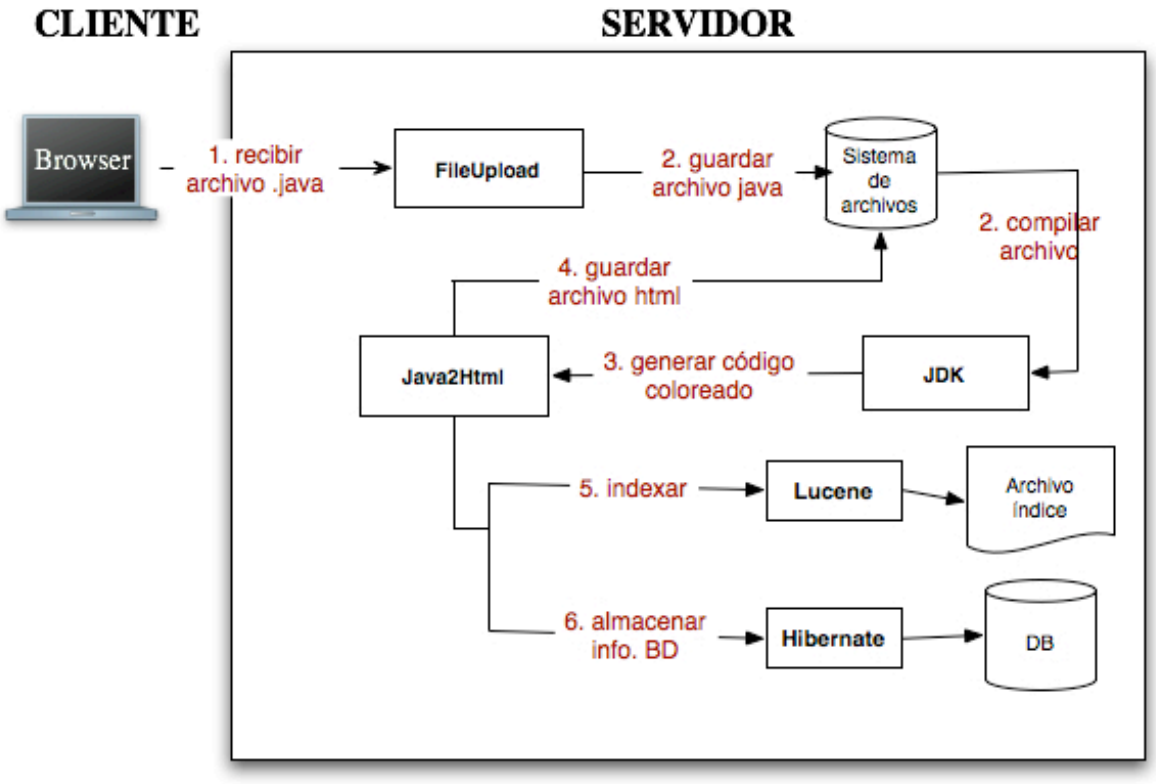


Figura 4.1 Procesamiento de Archivo de Java

4.3 Diagrama de paquetes

Los paquetes ofrecen un mecanismo de organización MVC como se había mencionado. Continuación se explica más a detalle cada paquete:

4.3.1 Paquete bean

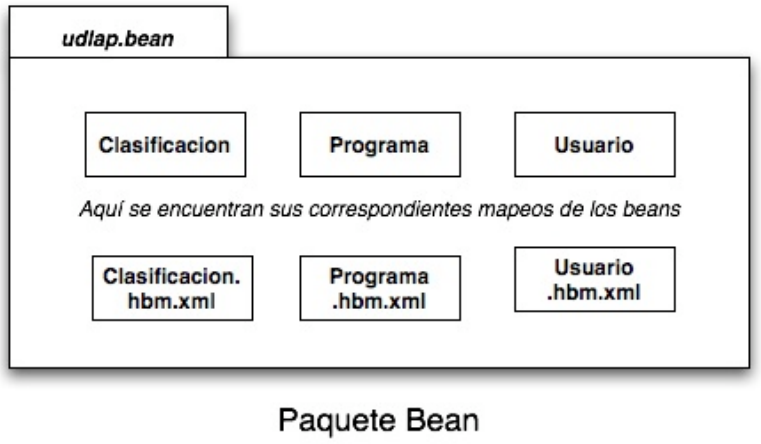
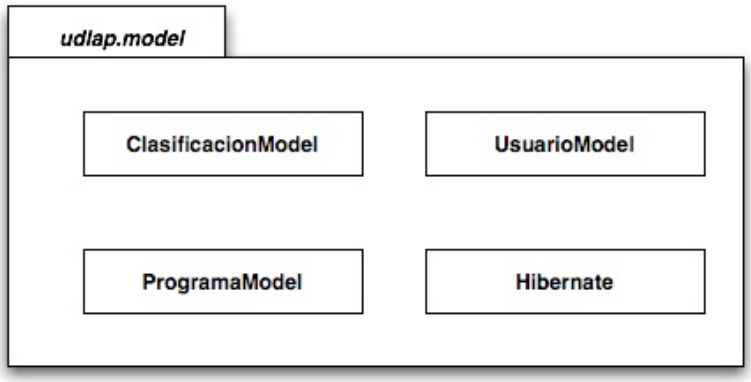


Figura 4.2 Paquete Bean

Este paquete contiene los *beans* Clasificación, Programa y Usuario, así como los archivos .hbm.xml necesarios para el mapeo a la base de datos. Los archivos que se utilizan para el mapeo se encuentran en el Apéndice B.

4.3.2 Paquete model

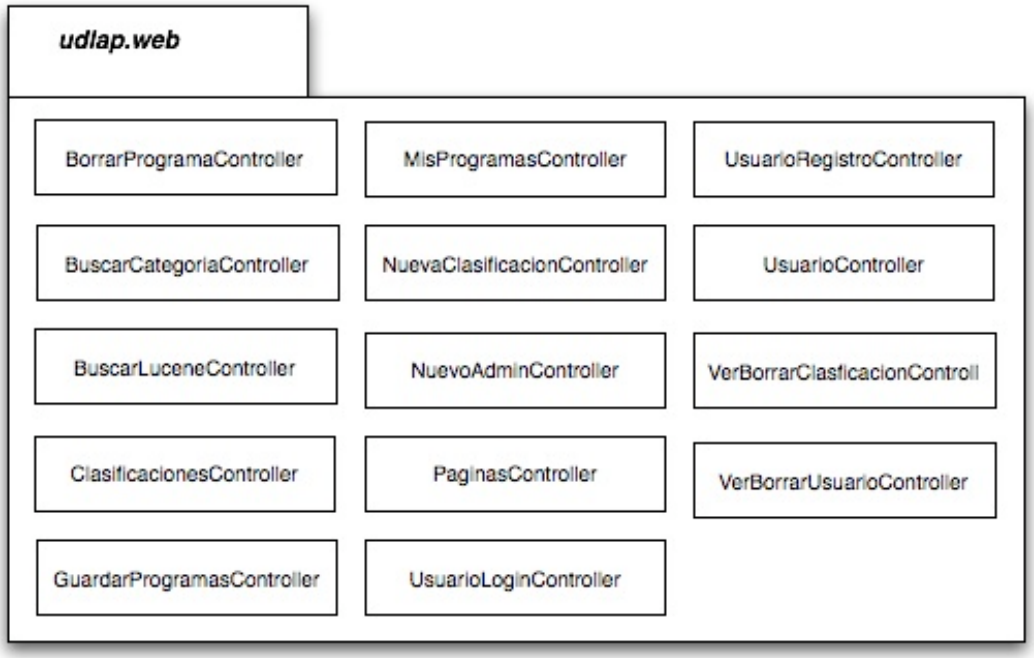
En estas clases se encuentra la lógica aplicativa del sistema. ClasificacionModel, UsuarioModel y ProgramaModel contienen los métodos que directamente interactúan con *Hibernate* (Agregar, Borrar, Buscar). La clase Hibernate contiene la configuración y se encarga de la conexión con el ORM *Hibernate*.



Paquete Model

Figura 4.3 Paquete Model

4.3.3 Paquete web



Paquete Web

Figura 4.4 Paquete Web

En este paquete se encuentran los *servlets* encargados de la comunicación con el modelo y la vista (JSPs).

BorrarProgramaController. Se encarga de borrar un programa de la base de datos. Este *servlets* es llamado a través de la forma que se encuentra en programas.jsp o programasAdmin.jsp. Utiliza una instancia de *LuceneIndexer* para borrar del índice al programa que quiere ser borrado.

BuscarCategoríaController. Se encarga de recuperar los programas existentes, en determinada clasificación. Este *servlet* es llamado a través de buscar.jsp y los datos recuperados son mostrados en resultados.jsp

BuscarLuceneController: Se encarga de las consultas de texto. Comunica a *LuceneIndexer* del paquete *utils* con el *servlet* para obtener los resultados de la búsqueda. Este *servlet* es llamado en buscar.jsp , la respuesta es desplegada en resultados.jsp

ClasificacionController: Se encarga de recuperar las categorías existentes, su clasificación y sus programas. Este *servlet* es llamado a través del menú principal "Clasificaciones", los datos recuperados son mostrados en clasificaciones.jsp

GuardarProgramaController: Este es el *servlet* que ejecuta más tareas: sube el archivo al servidor, lo compila, si no contienen errores de compilación, crea la referencia de este en la base de datos, lo agrega al índice de archivos para futuras consultas por texto, y finalmente genera el archivo html con el código coloreado. Este *servlet* es llamado y respondido en el mismo jsp: programas.jsp . Como se pudo observar hace uso de *LuceneIndexer* del paquete *utils.*

MisProgramasController: Este *servlet* muestra los programas, en el caso de estudiantes solamente muestra programas del mismo; y para el caso del administrador recupera todos los programas del repositorio. Este *servlet* es llamado en el menú principal, para el caso de estudiantes en "Mis Programas", para el caso de Administradores en el menú de "Programas". Los datos recuperados son mostrados en: programas.jsp y programasAdmin.jps

respectivamente. Hace uso de la clase *Page* del paquete *utils* para mostrar los resultados paginados de 10 en 10.

NuevaClasifiacionController: Este *servlet* se encarga de crear una nueva clasificación . Es llamado por el administrador cuando agrega una clasificación desde clasificaciones.jsp la respuesta es enviada al mismo jsp.

NuevoAdminController: Este *servlet* se encarga del registro de un nuevo administrador. Es llamado por el administrador cuando agrega un usuario nuevo desde usuarios.jps, la respuesta es enviada al mismo jsp.

PaginasController: Este *servlet* muestra los resultados paginados en listas de 10 en 10 de usuarios, programas y resultados. Es llamado cada vez que se hace alguna acción sobre los jsps de usuarios.jsp, resultados.jsp y usuarios.jsp

UsuarioLoginController: Este *servlet* es llamado desde el index.html. La respuesta de la acción es mostrada en logExitoso.jsp en caso de haber tenido una autentificación exitosa por el contrario si hubo algún error la respuesta es enviada a error..jsp

UsuarioRegistroController: Este *servlet* se encarga de hacer el registro correspondiente con el usuario. Es llamado en la pagina registro.html y la respuesta a esta acción es mostrada en logExitoso.jsp en caso de haber tenido un registro exitoso o error.jsp en caso de hacer encontrado algún error en este

UsuarioController: Este *servlet* muestra a los usuarios existentes en el repositorio, este es disponible solo para usuarios administrador. Este *servlet* es llamado en el menú principal "Usuarios". Los datos recuperados son mostrados en: usuarios.jsp. Hace uso de la clase PaginasController y *Page* del paquete *utils* para mostrar los resultados paginados de 10 en 10.

VerBorrarClasificacionController: Este *servlet* permite ver los programas de determinada clasificación, independientemente de poder borrar esta clasificación. Este *servlet* es llamado desde clasificaciones.jsp y la respuesta es enviada a programasAdmin.jsp

VerBorrarUsuarioController: Este *servlet* tiene dos funciones: la primera es permitir ver los programas de algún usuario determinado. La segunda función es permitir borrar el usuario con todos sus respectivos programas. Este *servlet* es llamado desde usuarios.jsp y la respuesta es enviada a programasAdmin.jsp

4.3.4 Paquete utils

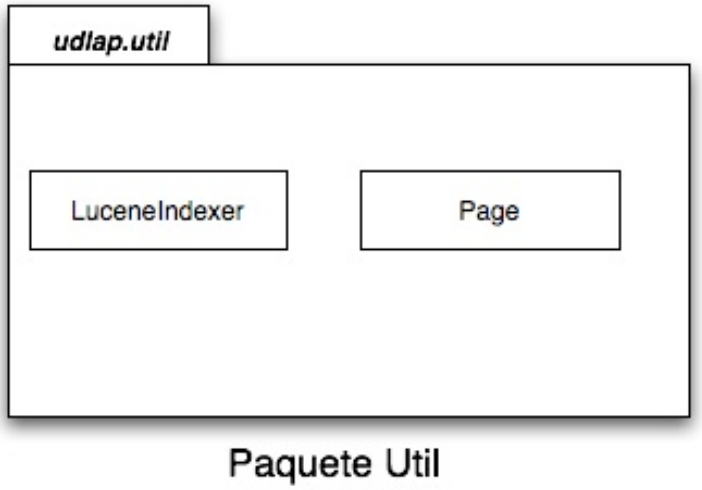


Figura 4.5 Paquete Util.

Luceneindexer: Implementado originalmente por Carlos Proal [Proal, 2005]. Se tomo esta clase y se adapto para que funcionara con objetos de tipo Programa. La clase recibe objetos programa, los convierte a documentos para ser agregados al índice, y hace operaciones de recuperación.

Page: Esta clase es el objeto pagina, la cual agrupa objetos (usuarios o programas) de 10 en 10 para su mejor visualización en lugar mostrar una lista de objetos muy larga.

4.4 Interacción Java, Hibernate y MySQL.

La interacción de Java con MySQL, se lleva a cabo totalmente a través de *Hibernate*, ahora se mostrará como está la base de datos realmente:

Tabla Clasificación:

```
mysql> desc clasificacion;
+-----------------+-------------+------+-----+---------+----------------+
| Field           | Type        | Null | Key | Default | Extra          |
+-----------------+-------------+------+-----+---------+----------------+
| CLASIFICACION_ID | int(11)     |      | PRI | NULL    | auto_increment |
| NOMBRE          | varchar(25)  |      |     |         |                |
| PALABRAS        | varchar(180) | YES  |     | NULL    |                |
| PADRE_ID        | int(11)     | YES  | MUL | NULL    |                |
+-----------------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Figura 4.6 Tabla Clasificación

Tabla Programa:

```
mysql> desc programa;
+--------------+-------------+------+-----+---------------------+----------------+
| Field        | Type        | Null | Key | Default             | Extra          |
+--------------+-------------+------+-----+---------------------+----------------+
| PROGRAMA_ID  | int(11)     |      | PRI | NULL                | auto_increment |
| NOMBRE       | varchar(50)  |      |     |                     |                |
| DESCRIPCION  | text        |      |     |                     |                |
| FECHA        | datetime    |      |     | 0000-00-00 00:00:00 |                |
| clasificacion | int(11)     |      | MUL | 0                   |                |
| RUTA         | varchar(200) |      |     |                     |                |
| AUTOR        | varchar(50)  |      |     |                     |                |
| usr          | int(11)     |      | MUL | 0                   |                |
+--------------+-------------+------+-----+---------------------+----------------+
8 rows in set (0.04 sec)
```

Figura 4.6 Tabla Programa

Tabla Usuario:

```
mysql> desc usuario;
+------------+-------------+------+-----+---------------------+----------------+
| Field      | Type        | Null | Key | Default             | Extra          |
+------------+-------------+------+-----+---------------------+----------------+
| USUARIO_ID | int(11)     |      | PRI | NULL                | auto_increment |
| USRNAME    | varchar(25) |      | UNI |                     |                |
| NOMBRE     | varchar(50) |      |     |                     |                |
| APELLIDO   | varchar(60) |      |     |                     |                |
| PASSWORD   | varchar(30) |      |     |                     |                |
| TIPOUSR    | char(1)     |      |     |                     |                |
| FECHA      | datetime    |      |     | 0000-00-00 00:00:00 |                |
+------------+-------------+------+-----+---------------------+----------------+
7 rows in set (0.00 sec)
```

Figura 4.8 Tabla usuario.

Ahora bien después de tener las dos perspectivas, se pueden percibir diferencias entre los objetos y la base de datos. (Para diferenciar el objeto Java de la tabla en la base de datos, ahora nos referiremos a todo lo referente a la base de datos con nombres en letras mayúsculas).

Clasificación tiene 6 atributos mientras que CLASIFICACIÓN tiene solo 4, se puede observar en la Figura 4.8 que los 2 atributos que faltan están relacionados en el primer caso hijas se asocia con otras tuplas directamente por el atributo PADRE_ID, para el caso de programas está definido en PROGRAMA en la columna CLASIFICACIÓN. Se puede observar que la recuperación de objetos implicaría una serie de operaciones SQL un tanto laboriosas para poder recuperar el objeto completo. Si ahora en lugar de buscar se quisiera borrar todo lo relacionado con alguna clasificación padre, sería aun más complicado. Sin embargo al manejar la persistencia con *Hibernate* estás operaciones se vuelven sencillas. Como ya se vio en el capítulo 2.

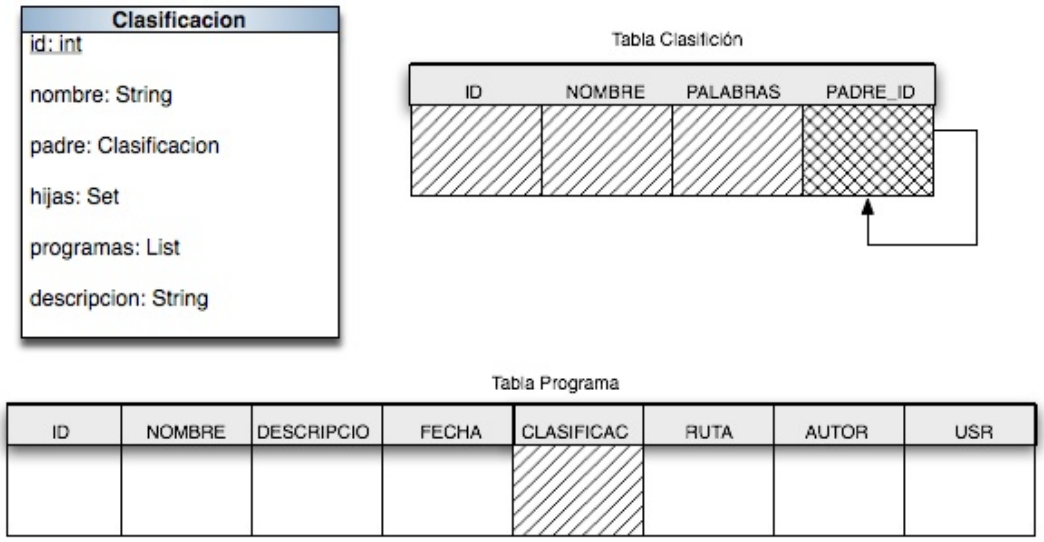


Figura 4.9 Relación Objeto – Tabla Clasificación

Para el caso de Programa se observa (Figura 4.9) que la tabla tiene igual numero columnas que la clase de Java tiene de atributos. Esto se debe a que las relaciones con los objetos Clasificación y Programa es de uno a muchos.

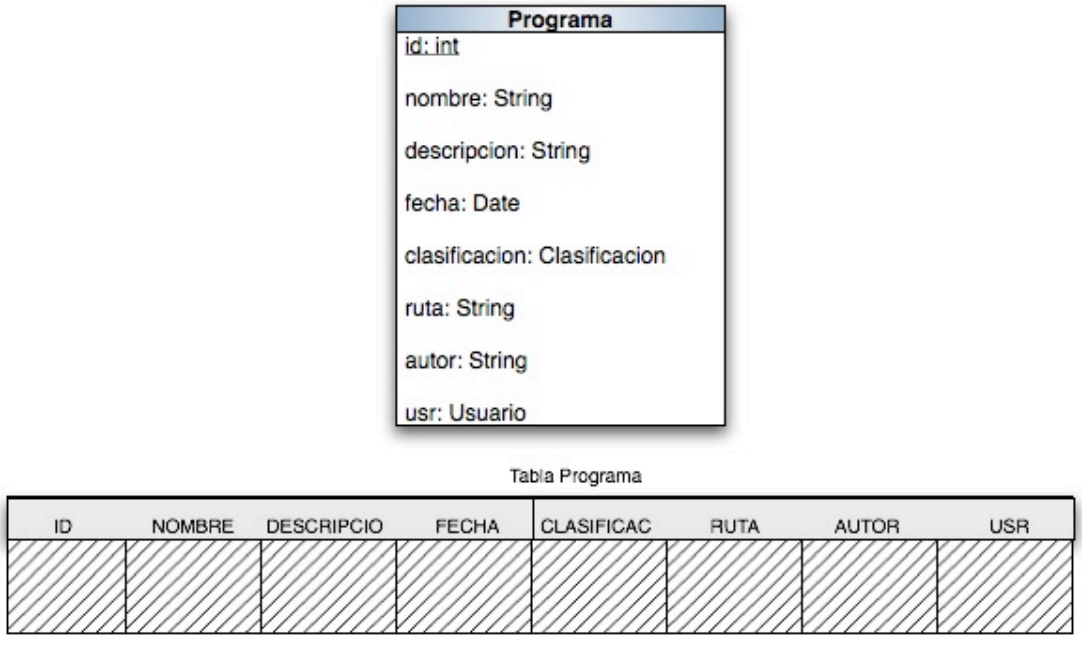


Figura 4.10 Relación Objeto – Tabla Programa

Para el caso de usuario (Véase Figura 4.10) se observa la situación similar a Clasificación la clase tiene más atributos que la tabla, debido a que se está asociando a muchos programas, el atributo programas se ve relacionado con el id de USUARIO

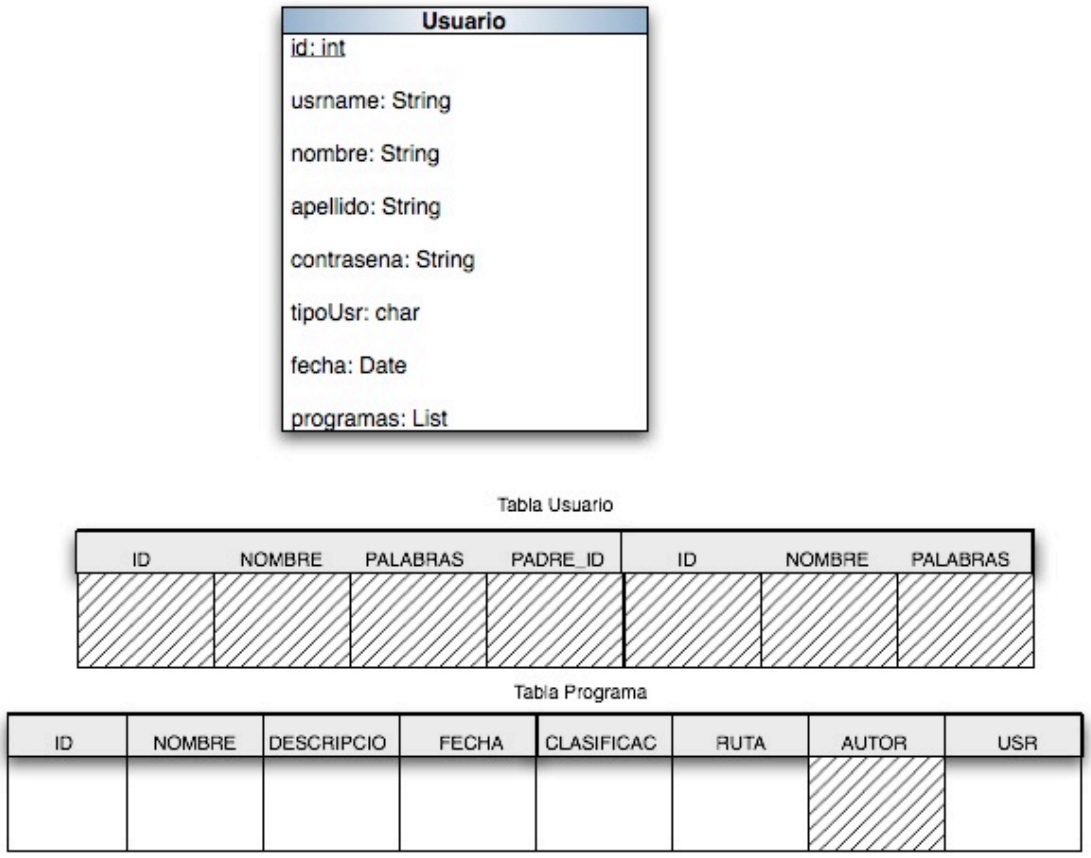


Figura 4.11 Relación Objeto – Tabla Usuario

4.5 Diagramas de clase

Los diagramas de clase nos muestran la representación de las clases en el sistema, incluyendo las definiciones de los atributos y las operaciones. Para ver los diagramas del sistema véase Apéndice C.