14-6-2017

Manual técnico

Micronapster

Integrantes:

- Díaz Medina Jesús Kaimorts
- Galindo Reyes Agustín
- Vargas Romero Erick Efraín

Profesor: Tecla Parra Roberto
Asignatura: Programación Orientada a Objetos
Tema: Proyecto final MicroNapster
Grupo: 2CM4
Fecha: 14/06/2017

Introducción:

A lo largo del curso de programación orientada a objetos se ha obtenido el conocimiento suficiente para poder elaborar una aplicación, para este caso se ha desarrollado una micro versión del legendario Napster, el cual hizo utilización de la tecnología P2P (peer to peer) la cual es una red de computadoras en las que algunos o todos los aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Para la creación de esta versión micro de Napster es necesario tener diversos conocimientos como lo es el uso de sockets cliente y sockets servidor, el manejo de hilos, manejo de eventos y como es evidente tener en cuenta el paradigma de la programación orientada a objetos. Siendo así y una vez tomado en cuenta lo anterior, debemos retomar lo que hacía a Napster tan popular, eso era el compartir música de una manera simple entre diferentes personas que hacían uso de la aplicación, pero no solo eso, como ya se mencionó se hacía uso de la tecnología P2P, además de que el servidor de esta antigua compañía no almacenaba las canciones que los usuarios querían compartir, sino que solo almacenaban las direcciones IP de cada usuario conectado para posteriormente hacer el envío de esta dirección a otro usuario que deseara una canción que estuviese alojada en la computadora de la dirección IP que el servidor envió a ese usuario, de tal manera que una vez hecho lo anterior es necesario que la computadora que tiene la canción que se desea trabaje como un microservidor para poder enviar la canción solicitada haciendo una conversión a bits y posteriormente enviar la canción en pequeños paquetes de bits que recibirá el usuario que solicitó la canción y como resultado su obtención.

Desarrollo

Para la elaboración de este proyecto se tomaron en cuenta diferentes aspectos, primeramente, la creación de dos clases que en el caso de esta aplicación son de real importancia ya que hacen más sencillo el manejo de los datos del programa, y son dos clases la primera llamada Song y la segunda llamada User.

La primera clase es utilizada para crear objetos, esta clase tiene diferentes variables de instancia las cuales son utilizadas para la manipulación del contenido de los objetos creados, además de que su nivel de acceso es privado, además de lo anterior. También se encuentran diferentes métodos que no son más que simples getters y setters y como es de esperase, tenemos un costructor.

```
public class Song implements Serializable {

    private String title, artist, genere, album, user, name;
    private int songSize;

    public Song(String title, String artist, String genere, String album,
    String user, int songSize, String name) {...}

    public String getTitle() {...}

    public String getArtist() {...}
```

```
    public String getGenere() {...}

    public String getAlbum() {...}

    public String getUser() {...}

    public int getSongSize() {...}

    public String getName() {...}

    public void setTitle(String string) {...}

    public void setName(String string) {...}

    public void setArtist(String string) {...}

    public void setGenere(String string) {...}

    public void setAlbum(String string) {...}

    public void setUser(String string) {...}

    public void setSongSize(int Int) {...}
}
```

De manera similar temenos a nuestra clase User, la cual también tiene diversos métodos y variables, pero en caso de los métodos solamente son getters y al igual que la clase Song tiene su respectivo constructor.

```
public class User implements Serializable {

    private String name, user, password, ipAddress;

    public User(String name, String user, String password, String
    ipAddress) {...}

    public String getName() {...}

    public String getPassword() {...}

    public String getIpAddress() {...}

    public String getUser() {...}

    public void setIpAddress(String string){...}
}
```

Una vez dicho lo anterior, es necesario argumentar que en el caso del resto del funcionamiento la parte "importante" es dividida en dos, la primera la parte del cliente y la segunda la del servidor.

En el caso del servidor tenemos diferentes clases, pero todas comparten el estar en ejecución de manera infinita (a menos que algo aborte el programa).

Primeramente, tenemos la clase addClientServer. Esta clase como ya se ha mencionado todo el funcionamiento de la clase esta implementado dentro del método main. Primeramente, dentro de nuestro método main tenemos nuestras variables, la primera es un objeto de tipo ServerSocket, la segunda es un objeto de tipo Socket (cliente) posteriormente tenemos dos objetos más el primero un ObjectInputStream y el segundo un ObjectOutputStream, finalmente tenemos una variable de tipo entero la cual establece un puerto.

Posteriormente tenemos un bloque try, y dentro de este primeramente se crea un objeto tipo ServerSocket, el cual en su constructor recibe una variable de tipo entero la cual es el puerto al cual se conectará, para esta clase el puerto usado es el 5501.

Una vez hecho lo anterior nos encontramos con un ciclo while, el cual tiene como argumento true, esto para hacer un ciclo infinito o bien darle ese comportamiento. Dentro de este ciclo infinito o indeterminado encontramos en la primera línea de código la obtención de un socket el cual nos establece la conexión entre nuestro servidor y un cliente. Lo siguiente es obtener los flujos de entrada y salida de datos del cliente esto se hace usando nuestros objetos ObjectInputStream y ObjectOutputStream, el primero para los flujos de entrada y el segundo para los flujos de salida, al crear los objetos ya antes mencionados el argumento de los constructores es el nombre de nuestro objeto Socket y usando el operador punto getInputStream y getOutputStream respectivamente, con esto se han obtenido los flujos de entrada y salida de nuestro cliente, lo siguiente es crear un objeto tipo User previamente mencionado el cual se obtiene leyendo el flujo de entrada, que debe de ser un objeto tipo User, para posteriormente obtener una dirección IP (La que esta usando el usuario) y colocarla en nuestro objeto User, para después pasar este objeto User a un método alojado en una clase llamada ServerDB en su método addUser, los cuales serán descritos más adelante. Para finalizar la descripción del bloque try se cierra el socket cliente.

```
public class addClientServer implements Serializable{

    public static void main (String args[]){
        ServerSocket serverSocket;
        Socket socket;
        ObjectInputStream input;
        ObjectOutputStream output;
        int port = 5501;
        try{
            serverSocket = new ServerSocket(port);
            while(true) {
               socket = serverSocket.accept();
               input = new ObjectInputStream(socket.getInputStream());
               output = new ObjectOutputStream(socket.getOutputStream());
               User newUser = (User)input.readObject();
newUser.setIpAddress(((socket.getLocalAddress().toString()).replace((char
)47, ' ')).replaceAll("\\s+", ""));
                new ServerDB().addUser(newUser);
                socket.close();
            }
        } catch (IOException e) {...}
```

```
            catch (Exception e) {...}
    }
}
```

En el caso del resto de clases utilizadas para nuestro servidor, son prácticamente idénticas con excepción de un par de líneas de código, en el caso de nuestra clase addSongServer en vez de leer un usuario se lee un ArrayList<Song> tal y como se muestra a continuación.

```
public class addSongServer {
    public static void main(String argas[]){
        ...
        new ServerDB().addSong((ArrayList<Song>)input.readObject());
        ...
    }

}
```

Nuevamente en el caso de la clase deleteSongs repetimos lo mismo pero dentro de esta clase se crea un objeto tipo new ServerDB y usamos el método sessionClose la cual recibe como argumento un String la cual se obtiene del flujo de entrada.

```
public class deleteSongs implements Serializable {
    public static void main(String args[]){
        ...
        serverDB.sessionClose((String)input.readObject());
        ...
    }
}
```

La siguiente clase llamada getIPAddress es muy similar a la deleteSongs solo que aquí se obtendrá una cadena desde una base de datos y posteriormente se enviará como respuesta al usuario que hizo la petición

```
public class getIPAddress implements Serializable{

    public static void main(String args[]){
      ...
      ServerDB db = new ServerDB();
      output.writeObject(db.getIPAddress((String)input.readObject()));
      ...
    }
}
```

Otra clase que pertenece al servidor es getSongsAvailable, que al igual a getIPAddress el servidor envía una respuesta, solo que en este caso la respuesta es un ArrayList<Song>.

```
public class getSongsAvailable implements Serializable{
    public static void main(String args[]){
      ...
      output.writeObject(new
      ServerDB().getSongs((Request)input.readObject()));
      ...
    }
}
```

Otra clase más del servidor es getUserInfo la cual usamos para validar si hay un usuario ya registrado o no, esta al igual que las dos anteriores nos retorna un objeto, en este caso es tipo boolean, además de que al utilizarse esta clase para verificar el logeo de los usuarios se actualiza la dirección IP

```
public class getUserInfo implements Serializable {
    public static void main(String args[]) {
        ...
        serverDB = new ServerDB();
        User user = (User)input.readObject();
        state = serverDB.verifyUser(user);
        serverDB.UpdateIPAddress(user,
       (((socket.getLocalAddress().toString()).replace((char)47, '
       ')).replaceAll("\\s+", "")));
        output.writeObject(state);
       ...
   }
}
```

Otra clase que es considerada de alta importancia es la llamada microServer, solo que esta clase está de parte del cliente, como ya se mencionó previamente, cada cliente al mismo tiempo es un micro servidor ya que si alguien desea una canción el usuario que la posee debe de enviarla convirtiéndola a bits, de tal manera que por ese breve periodo de tiempo se comportara como un servidor.

Esta clase es similar a las de nuestro servidor "principal" pero hay algunos cambios en su implementación, para que se pueda hacer el envio de los paquetes de bits al usuario que desea la canción. Como es posible observar pimeramente se obtiene un objeto Song con los flujos de entrada, posteriormente se crea un objeto tipo File que recibe como argumento una ruta, la cual es almacenada en cada computadora usando preferencias. Una vez mencionado lo anterior se obtiene la longitud del archivo. Acontinuación se establece el tamaño de los paquetes de bytes que se enviaran en este caso de 16 * 1024. Continuamos con la creación de un InputStream el cual crea un Objeto tipo FileInputStream y como argumento recibe nuestro objeto File. Posteriormente tenemos un objeto tipo OutputStream el cual obtiene los flujos de salida del socket cliente. En la siguiente línea se declara una variable tipo entero, el cual se usará de contador y en la línea de abajo tenemos un ciclo while, el cual evalua al contador, el cual es igualado a lo que lea del objeto InputStram, leyendo sus bytes, si lo obtenido es mayor a cero se hace el envio de un paquete de bytes al cliente. Finalmente, se cierra el InputStream y con esto se ha hecho el envío de bytes al cliente.

```
public class microServer implements Serializable {

    public static void main(String args[]) {
        ...
        song = (Song)input.readObject();
        File file = new File(new
        preferencesClient().getPath(song.getUser()) + "/" +
        song.getName());
        long len = file.length();
        byte[] bytes = new byte[16 * 1024];
        InputStream in = new FileInputStream(file);
```

```
        OutputStream out = socket.getOutputStream();
        int count;
        while ((count = in.read(bytes)) > 0) {
          out.write(bytes, 0, count);
        }
        input.close();
        ...
    }
}
```

Para finalizar, se mencionará la clase downloadSong, esta implementa la interfaz Runnable la cual contiene al método abstracto run, además esta clase está del lado del cliente que quiere una canción, dentro de esta clase se tienen dos métodos, el primero es downloadSong (El constructor de esta clase) el cual recibe como argumente un objeto tipo Song, de igual manera dentro de este método se crea un hilo, al ejecutarse se obtiene una dirección IP la cual nos envía el servidor, se cierra el socket cliente y ahora se invoca al método downloadingSong, el cual recibe como parámetro un String el cual es el nuevo host al cual se conectará el cliente para descargar la canción, dentro de este método similarmente a la clase descrita anteriormente se crean diversos objetos, el primero es un FileOutputStream cuyo argumento es el nombre que tendrá la canción descargada, o bien podría ser su ruta, posteriormente se establece el tamaño de los paquetes de bytes que recibirá, y al igual que la clase anterior usando un contador, se leen los bytes que tenga nuestro objeto InputStream previamente creado y el ciclo que se encarga de esto se detiene cuando los bytes leídos no sean mayores a cero

```
public class downloadSong implements Runnable, Serializable {

    public downloadSong(Song song) {
        this.song = song;
        try {
            thread = new Thread(this);
            socket = new Socket(host, port);
            output = new ObjectOutputStream(socket.getOutputStream());
            input = new ObjectInputStream(socket.getInputStream());
            thread.start();
        } catch (IOException e) {...}
        catch (Exception e) {...}
    }

    public void downloadingSong(String newHost) {
        try {
            socketDownload = new Socket(newHost, otherPort);
            output = new
            ObjectOutputStream(socketDownload.getOutputStream());
            output.writeObject(song);
            InputStream in = socketDownload.getInputStream();

            FileOutputStream out = new FileOutputStream(song.getName() +
            ".mp3");
            byte[] bytes = new byte[16 * 1024];
            int count;
            while ((count = in.read(bytes)) > 0) {
```

```
                out.write(bytes, 0, count);
            }
            out.close();
            output.close();
        } catch (IOException e) {...}
          catch (Exception e) {...}
    }

    @Override
    public void run() {
        try {
            output.writeObject(song.getUser());
            ipaddress = (String) input.readObject();
            socket.close();
            downloadingSong(ipaddress);
        } catch (IOException e) {...}
          catch (Exception e) {...}
    }
}
```

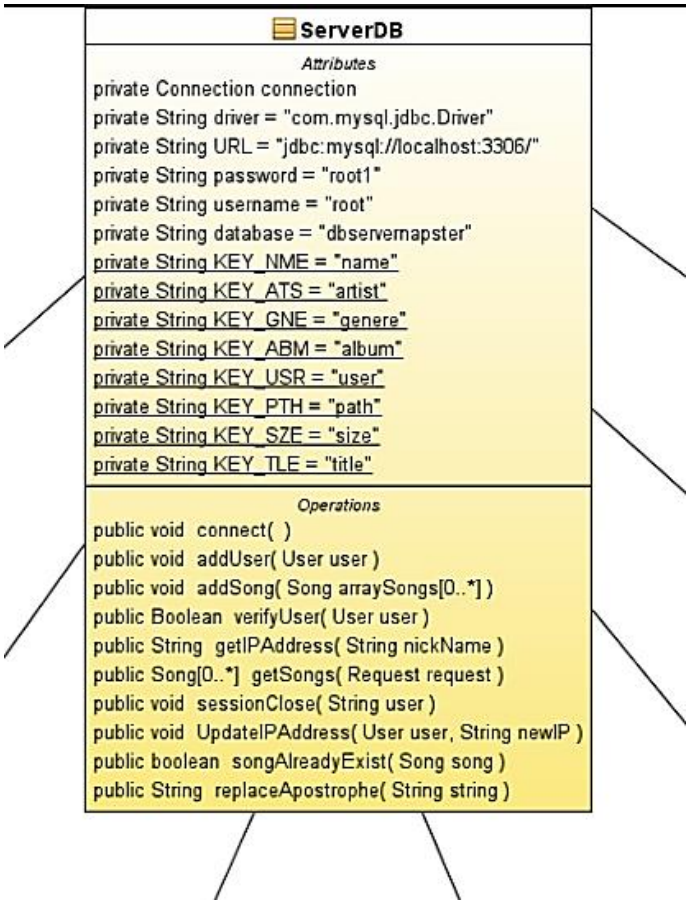
Bases de datos

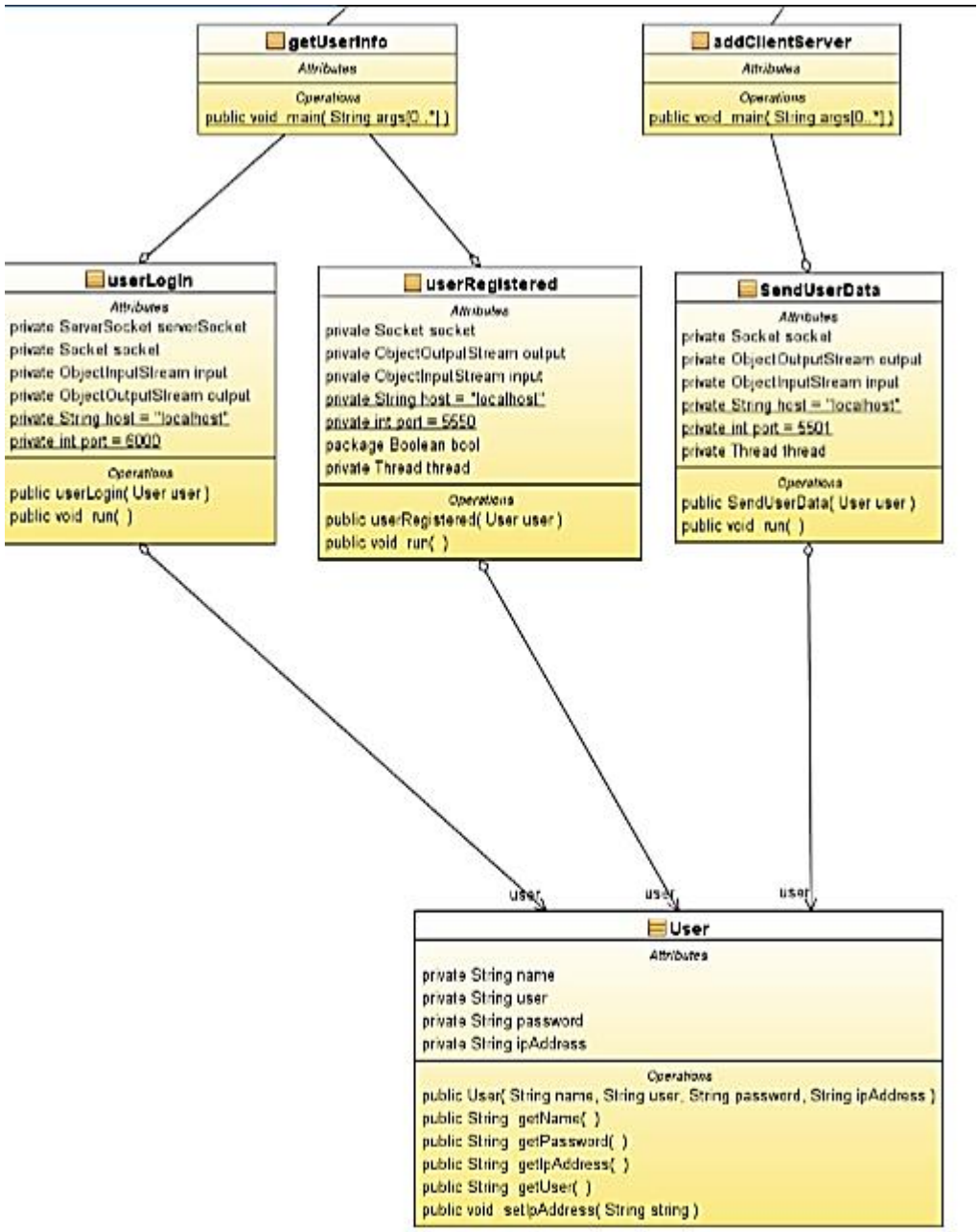
En el apartado de bases de datos se ha requerido crear una base de datos llamada dbservernapster que contiene dos tablas una llamada tblsongsnapster y otra llamada tblusers, la primera como es evidente es para almacenar los datos de las canciones y la segunda para almacenar los datos de nuestros clientes, tal y como se muestra en las siguientes imágenes

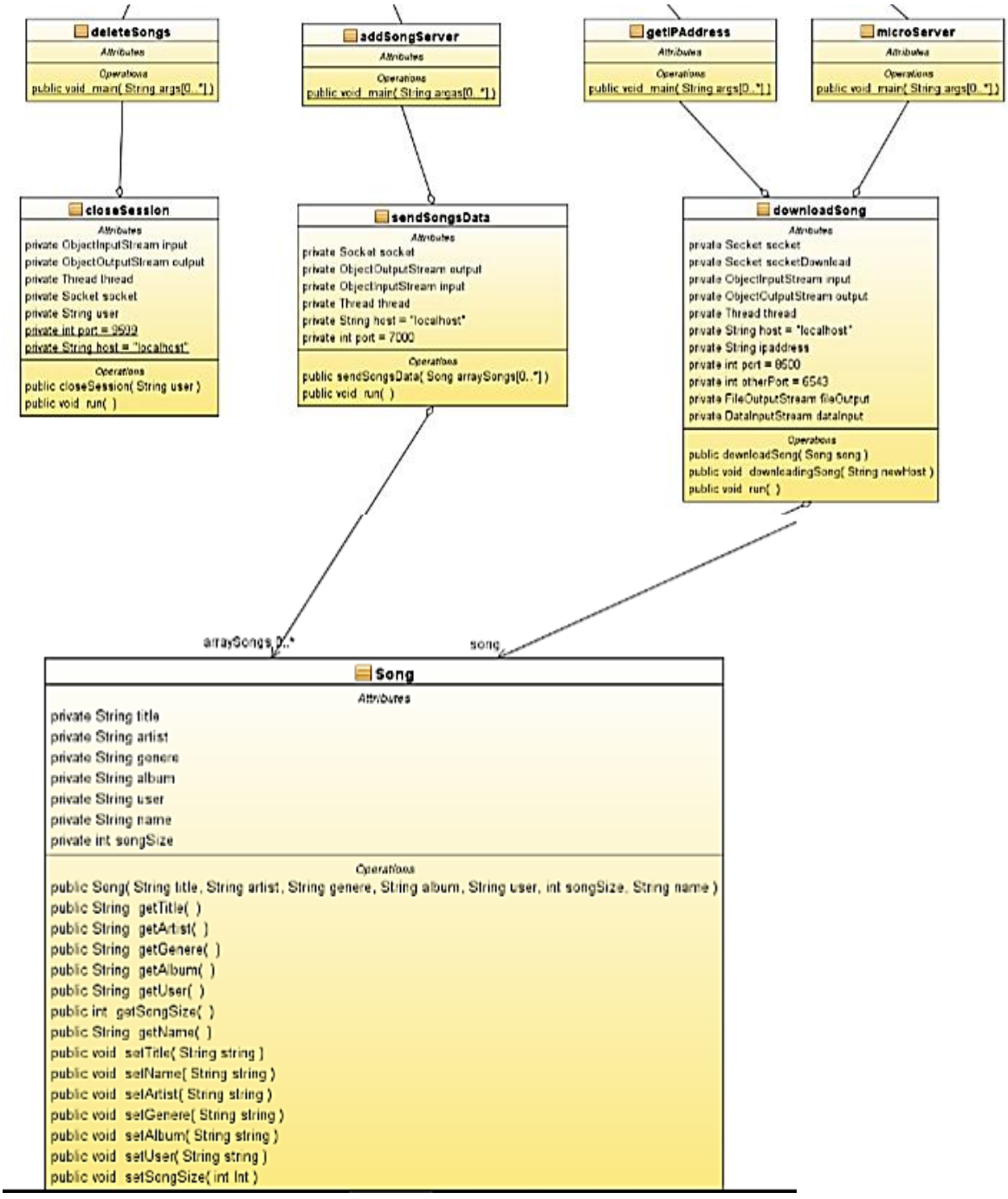
```
mysql> desc tblsongsnapster;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| name   | varchar(255)| YES  |     | NULL    |                |
| artist | varchar(255)| YES  |     | NULL    |                |
| genere | varchar(15) | YES  |     | NULL    |                |
| album  | varchar(15) | YES  |     | NULL    |                |
| user   | varchar(15) | YES  |     | NULL    |                |
| size   | int(255)    | YES  |     | NULL    |                |
| title  | varchar(200)| YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
```

```
mysql> desc tblusers;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| name      | varchar(10) | YES  |     | NULL    |                |
| password  | varchar(10) | YES  |     | NULL    |                |
| ipaddress | varchar(20) | YES  |     | NULL    |                |
| nick      | varchar(50) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Diagrama de clase




getUserInfo
Attributes
Operations
public void main(String args[0..*])
addClientServer
Attributes
Operations
public void main(String args[0..*])
userLogin
Attributes
private ServerSocket serverSocket
private Socket socket
private ObjectInputStream input
private ObjectOutputStream output
private String host = "localhost"
private int port = 6000
Operations
public userLogin(User user)
public void run( )
userRegistered
Attributes
private Socket socket
private ObjectOutputStream output
private ObjectInputStream input
private String host = "localhost"
private int port = 5550
package Boolean bool
private Thread thread
Operations
public userRegistered(User user)
public void run( )
SendUserData
Attributes
private Socket socket
private ObjectOutputStream output
private ObjectInputStream input
private String host = "localhost"
private int port = 5501
private Thread thread
Operations
public SendUserData(User user)
public void run( )
user
user
user
User
Attributes
private String name
private String user
private String password
private String ipAddress
Operations
public User(String name, String user, String password, String ipAddress)
public String getName( )
public String getPassword( )
public String getIpAddress( )
public String getUser( )
public void setIpAddress(String string)


deleteSongs
Attributes
Operations
public void main(String args[0..*])
addSongServer
Attributes
Operations
public void main(String args[0..*])
getIPAddress
Attributes
Operations
public void main(String args[0..*])
microServer
Attributes
Operations
public void main(String args[0..*])
closeSession
Attributes
private ObjectInputStream input
private ObjectOutputStream output
private Thread thread
private Socket socket
private String user
private int port = 9999
private String host = "localhost"
Operations
public closeSession(String user)
public void run( )
sendSongsData
Attributes
private Socket socket
private ObjectOutputStream output
private ObjectInputStream input
private Thread thread
private String host = "localhost"
private int port = 7000
Operations
public sendSongsData(Song arraySongs[0..*])
public void run( )
downloadSong
Attributes
private Socket socket
private Socket socketDownload
private ObjectInputStream input
private ObjectOutputStream output
private Thread thread
private String host = "localhost"
private String ipaddress
private int port = 9500
private int otherPort = 6543
private FileOutputStream fileOutput
private DataInputStream dataInput
Operations
public downloadSong(Song song)
public void downloadingSong(String newHost)
public void run( )
arraySongs[0..*]
song
Song
Attributes
private String title
private String artist
private String genere
private String album
private String user
private String name
private int songSize
Operations
public Song(String title, String artist, String genere, String album, String user, int songSize, String name)
public String getTitle( )
public String getArtist( )
public String getGenere( )
public String getAlbum( )
public String getUser( )
public int getSongSize( )
public String getName( )
public void setTitle(String string)
public void setName(String string)
public void setArtist(String string)
public void setGenere(String string)
public void setAlbum(String string)
public void setUser(String string)
public void setSongSize(int Int)

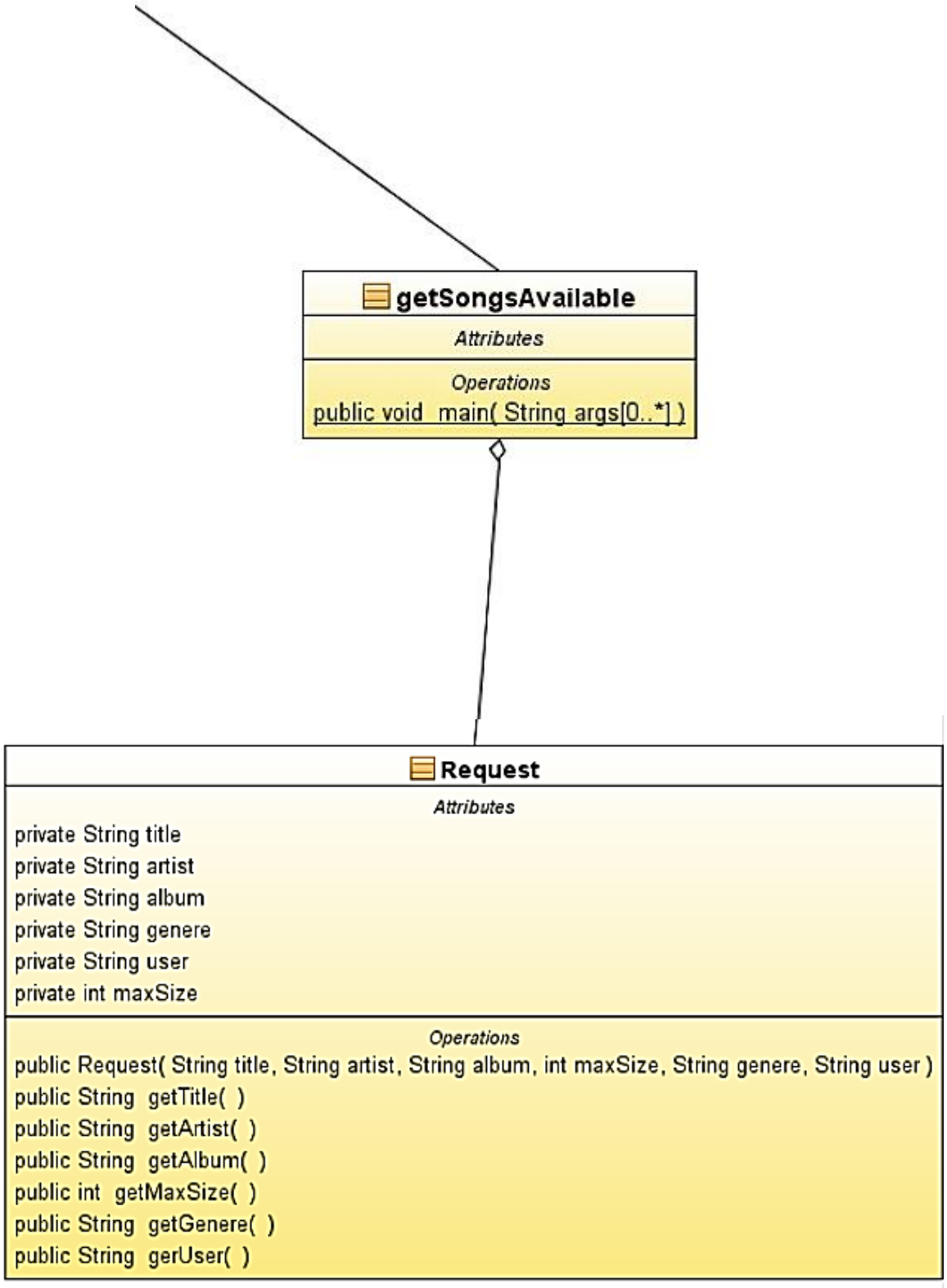

getSongsAvailable
Attributes
Operations
public void main(String args[0..*])
Request
Attributes
private String title
private String artist
private String album
private String genere
private String user
private int maxSize
Operations
public Request(String title, String artist, String album, int maxSize, String genere, String user)
public String getTitle( )
public String getArtist( )
public String getAlbum( )
public int getMaxSize( )
public String getGenere( )
public String gerUser( )

Diagrama entidad - relación