Middle East Technical University | Department of Computer Engineering

CENG 443

Object Oriented Programming

Spring '2014-2015

Programming Assignment 3
Remote Music Interface needs you

By Yamaç Kurtuluş

Due date: 7 June 2015, Sunday, 23:55

1 Objectives

In this assignment you are going to simulate a song tagging service using Java features such as JDBC MySQL interface, Remote Method Invocation and serialization.

Keywords: *Java, JDBC, RMI, MySQL, Serialization*

Managing and organizing audio files is a huge problem for archiving enthusiasts out there. Therefore, some services like Gracenote[1] provide automatical audio tagging using sophisticated audio analysis methods. Another leading music organization called RMIvice (Remote Music Interface Service) and hired you to implement a similar service. Its specifications are given below.

2 Specifications

We are going to keep things simple and use only MD5 hashing. The audio files are not even audio files per se, just a random sequence of bytes with variable length between 1024 - 10240 elements (1 - 10 KBs) in binary files (extension: .audio) that signify audio data. Your task is to implement server and client classes and simulate a simple audio tagging service. You are given some sample files, the database structure and the skeleton code along with this text.

2.1 Classes

- **Song**: This class will consist of a byte Array representing the audio data, and one SongInfo object. Your task is to convert the .audio files to serializable Song objects by reading the binary files and getting info from the server.
- **SongInfo**: This class has 5 data fields and an MD5 hash[2]. It stores information about the song as well as being used for data exchange between server and clients. The data fields are: `name`, `artist`, `album`, `year` and `genre`. The MD5 hash is obtained from the data part of the Song objects and used for identifying the song in the server.

[1] `http://www.gracenote.com/`

[2] You can use Java's MD5 class. For those of you that are interested more deeply: `https://www.ietf.org/rfc/rfc1321.txt`

- **Client**: Clients communicate with the server. They are able to:

  read the .audio files and get the relevant information from the database. This information can be empty or partially full. After getting the info, it should combine the info and data in a Song object, lastly, you should serialize the object with .ser extension.

  Additionally, they can update the information of serialized files and store the updated song information to the server.

  You need to use serialisation API both for reading and updating the objects and to store and get information to the server, you need to use RMI

- **Server**: Server class is reponsible for managing the database, fetching info from it and reply to clients. It checks the hash sent via a SongInfo object, and sends back a full one when a client wants to get song data. Updating is done similarly, using the hash of the sent SongInfo and updating the table accordingly.

- interface **ISongInfoService**: This class defines the RMI methods to be implemented. It has two methods with the following signatures: `SongInfo get(SongInfo s)` and `void store(SongInfo s)`

2.2 Requirements

- The clients will communicate to the server via RMI.

- Server will have a MySQL database with one table that has the following structure:

name : varchar(45)	artist : varchar(45)	album : (varchar(45)	year : int	genre : varchar(45)	hash : binary(16)
Nightfall	Blind Guardian	Nightfall in Middle Earth	1998	Power Metal	cb4c88b1771dc85b1eaa39ab61fca9fa
Ever Dream	Nightwish	Century Child	2002	Symphonic Metal	5d126c2c4e4c74eb02ebab2734d2bc3a
Holy Diver	R. J. Dio	Holy Diver	1983	Classical Rock	5cc1847e0788eb9aaf7cab43460a22b5

  To access this database, you will use a JDBC driver. The schema is provided to you. The schema name is "peertopeer". And some of the fields migt be null, which is intended.

- You can use the following commands in the client:

  `get <filename>`: create a Song object by reading the .audio file `<filename>.audio`. Then gets the SongInfo using RMI by comparing the MD5 hash of the data and saves the final object at hand to `<filename>.ser`

  `update <filename>`: Update the SongInfo of the `<filename>.ser` and send it to server to replace the info. Again, you can use a desktop or command based interface or process additional arguments.

  `read <filename> | *`: Output the information of the serialized file `<filename>`.ser. If "*" is given as argument, then output the information of all the serialized song objects (i.e. ser files).

- To keep things simple and standardized, while testing, we are going to use lowercase letters only for commands, string fields etc.

- Some example files and the SQL structure will be given to you, along with a skeleton code. They may be good points to start.

- A simple UML class diagram is given in 1.

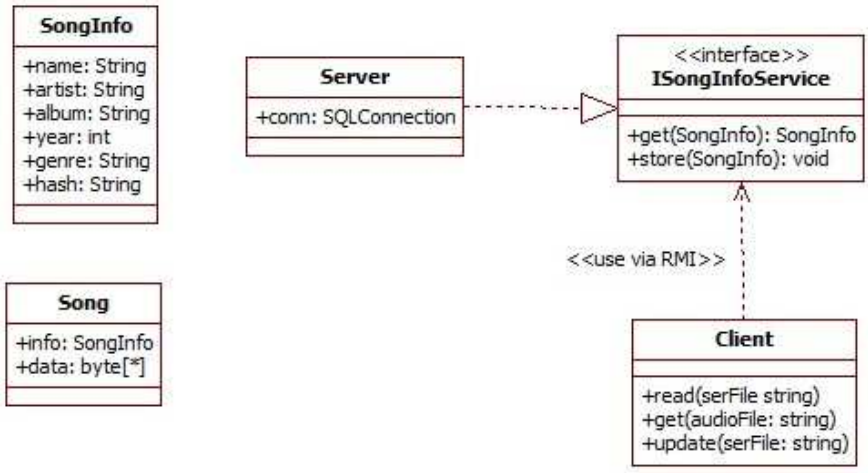

Figure 1: A simplified class diagram of the system

3 Regulations

- **Programming Language:** Java
- There are a number of design decisions and possible opportunities for visual illustrations and creative extensions for this homework. There will be bonuses awarded for all types of extra effort.
- **Late Submission:** No late submission is allowed, therefore, try to have at least a working baseline system by the deadline.
- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.

4 Submission

Submission will be done via COW. Create a tar.gz file named `e1234567_hw3.tar.gz` that contains only your source code files. Please do not use packages.
The following command sequence is expected to run your program on a Linux system:

```
$ tar -xf e1234567_hw3.tar.gz
$ javac *.java
$ java server
$ java client
```

Good luck and have fun.