

Individual Project 1– Simple Database

15 Points

CS152 – Dr. Shaffer

Due: February 2, 2008

1 Goals

We wish to make some changes to the FirstDatabase project discussed in class so that it represents a more robust file-based database system. We will deal with the following problems (described in subsequent sections):

- user interface doesn't support "remove DVD" operation,
- concurrent access to database file should not be allowed,
- preventing concurrent access to file shouldn't prevent concurrent use of program.

The following sections describe the changes that you must make to the FirstDatabase project.

2 Remove DVD

Add a remove DVD action, modeled after the existing actions. The `JTable` includes a method to find the currently selected rows. Read the javadocs for details. Make sure that the view of the database updates properly after you remove the DVD. See the `addDVD` method to find out how that's done.

3 Locking

Most operating systems support the concept of a "file lock." In Java that facility is accessed through the `java.nio.channels.FileChannel` class' `lock()` and `tryLock()` methods. Familiarize yourself with this class and those methods in particular. Note: as indicated in the documentation you obtain a `FileChannel` from a `FileInputStream` or `FileOutputStream` using the `getChannel()` method.

Modify the `DVDDatabase` class so that loading the database locks the database file and saving unlocks it. The `load` method will now also need to handle the case where it fails to obtain the lock (because some other process has the file locked) by informing the user that the file is "in use by someone else." (You can use `javax.swing.JOptionPane.showMessageDialog(...)` for this purpose). Ensure that the UI does not permit the user to change the database unless they have a lock on it (so once they press save, they'll have to re-load the data before they can edit it).

To be a good citizen, it is a good idea to make sure that you release any locks you are holding when your program exits (most operating systems will take care of this for you but let's be careful about it). Use `JFrame.addWindowListener(...)` to handle the window close event by unlocking (if needed) the database file.

Check that your program is working correctly by running it multiple times and verifying that only one of those running instances actually opens the database successfully. Also be sure to check that the lock is properly released when that instance saves or exits.

4 Concurrent access

OK, now you have implemented a simple locking scheme and it prevents loss of data by preventing multiple users from entering data at the same time. This is a substantial improvement from the original version but now you have a piece of software that is rather inconvenient to use in a shared environment. We need to take this one step further by only locking the database for short periods of time. We achieve this by changing the structure of the database and the way we save and load it. This is going to be a substantial change to the existing code. The `DVDListView` class should not change at all (or maybe very minimally) during this phase, though.

I will refer to a running instance of our program as a “client” of the database. We wish to support multiple clients without loss of data except in the rare case where two clients make conflicting modifications to the database. Among the many ways to accomplish this is the lock-and-modify concept. Instead of updating the data file each time a change is made we, instead, keep track of what changes have been made during the current session and, upon save, lock and load the most recent copy of the database, apply our changes and then save. This means that the database file will be locked for only the short time it takes to load it, apply the changes and save it.

Make changes to the `DVDDatabase` class to track changes to the database and apply them to the most recently saved version using the mechanism described above. Keep in mind that in order to do this you will have to modify `DVD` so that it informs the database whenever one of its setters is invoked. You should create classes to represent all of the different kinds of changes that can be made to the database:

- add DVD
- remove DVD
- change DVD title
- change DVD number of tracks

You will have to substantially modify the save operation so that it loads the list of DVD. Once the data is saved the client’s view of the data should be updated to reflect the most recently saved version (that is, it will show other client’s changes to the data as well).

Hint: In order to replay the changes, each DVD needs to be assigned a unique id. For new DVD’s, this id should be assigned during the save process. All of the changes (except add DVD) need to remember the **id** of the DVD that was changed, not the actual DVD object. This should be clear from our object identity discussions in class.

Note: It is possible for conflicting changes to occur to the database. For example, in one session a client may delete a DVD while in another session the client might modify that DVD. Your save method should detect many of these cases and inform the user of the conflict. The few cases that it can’t easily detect will be discussed in class.

5 Checking your design

Before you turn in your solution, make sure that your design is clean. All database-related concepts should be handled by the `DVDDatabase` class. Your `DVDListView` should only be concerned with user-interface aspects of the system and informing the database about any changes resulting from those actions. Finally `DVD` should relatively unchanged from its original version except for the addition of the id field and notifying the `DVDDatabase` whenever one of its setters gets called.

6 To hand in

e-mail a jar containing all of your source code to me at cdshaffer@acm.org.