605.201 Mini-Project 3:

Please do the following to complete this assignment.

Purpose:

The purpose of this project is to provide non-trivial practice in the use of Java object-oriented GUI programming features to implement an object-oriented GUI design and have a bit of fun doing it.

Resources Needed:

You will need a computer system with Java 8 or greater SE edition run-time and Java Development Kit (JDK). You may optionally use a Java IDE for example NetBeans, Eclipse, etc. However application builders are not allowed.

Submitted Files:

Design and Analysis:

This is an informal essay-style single-spaced word-processed document. The file formats accepted are .odt, .doc, and .docx. The length of the document should be between 1 and 1.5 pages. The following subjects should be discussed in this order:

1. General program design. How is the program organized? What major data structures were used? How did you divide the functionality among your classes? How are commands processed? Etc.
2. What alternative approaches were considered and why were they rejected?
3. What did you learn from doing this project and what would you do differently?

Source files:

This application must use JavaFX for the GUI component. Each public class *must* be contained in a separate Java source file. Only one source file will have a main() method and this source will be named **SongDatabase.java**. Other source/class names are up to you following the guidelines specified so far in the course.

The format of the Java source must meet the general Java coding style guidelines discussed so far during the course. Pay special attention to naming guidelines, use of appropriate variable names and types, variable scope (public, private, protected, etc.), indentation, and comments. Classes and methods should be commented with JavaDoc-style comments (see below). Please use course office hours or contact the instructor directly if there are any coding style questions.

JavaDocs:

Sources should be commented using JavaDoc-style comments for classes and methods. Each class should have a short comment on what it represents and use the @author annotation. Methods should have a short (usually 1 short sentence) description of what the results are of calling it. Parameters and returns should be documented with the @param and @return annotations respectively with a short comment on each.

JavaDocs must be generated against every project Java source file in one JavaDocs folder (i.e. one index.html file). They should be generated with a **-private** option (to document all protection-level classes) and a **–**d [dir] option to place the resulting files in a **javadocs** directory/folder at the same level as your source files. See the JavaDocs demonstration for more details.

Submit file:

The submit file is to be a Zip file containing your design and analysis document, your Java sources, and your javadocs directory/folder. Any appropriate file name for this Zip file is acceptable.

If you know how to create a standard Java JAR file, this is also acceptable for your source code. However, make sure you include the source code in your JAR file.

Collaboration:

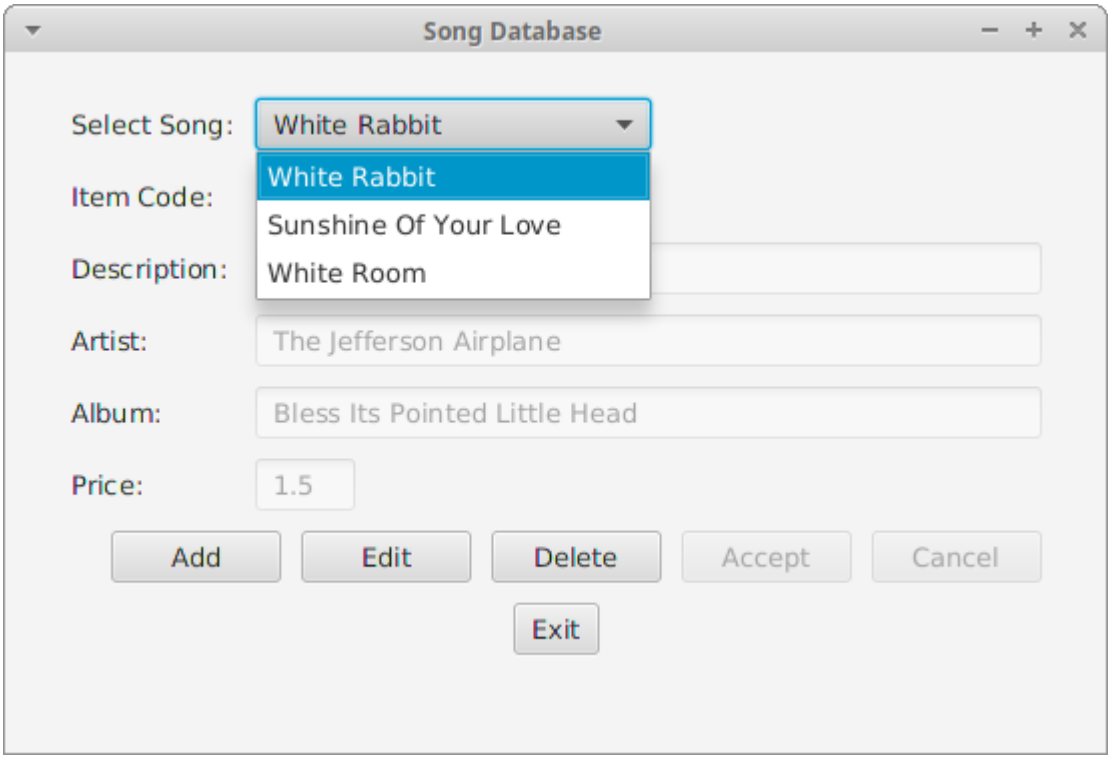
It is encouraged to discuss technical or small design parts of this project with your fellow students. However the resulting design and implementation must be your own. For example, it is acceptable to discuss different ways of maintaining the system state but not detailed design or implementation information on processing the purchase command. When in doubt, ask during office hours or contact your instructor.

Program Specification:

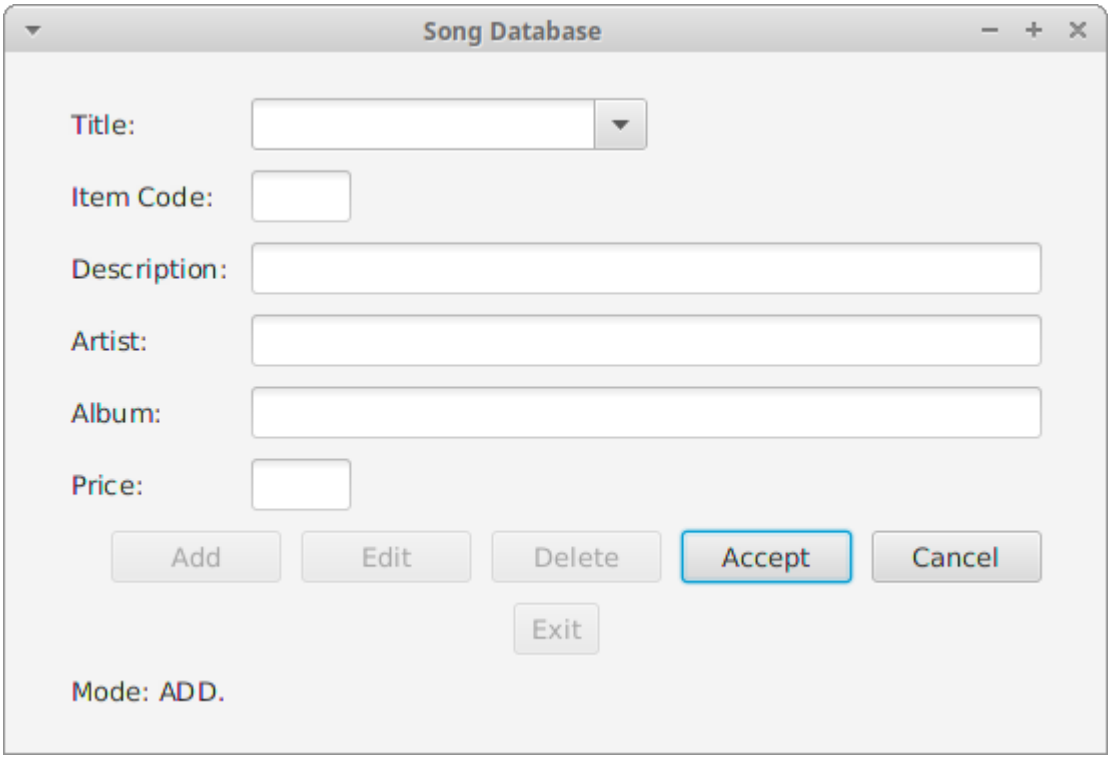
This project involves implementing a Java program that builds and manages a database of songs using a GUI-based user interface implemented with JavaFX.

The application will consist of a single frame that allows a user to add, edit, or delete songs from the database. Here's a rough prototype for what the window might look like. Yours doesn't have to look exactly like this…the prototype is provided just to help you envision the ultimate application.

Selecting a song



Adding a song



The application shall work as follows:

1. Upon start-up, the application will read the database file. The path of the database file will be supplied by a run-time parameter on the terminal window. If the database file does not exist, the user will be told the database does not exist and prompted in the terminal window asking if they want to create a new one. If the user answers positive, the application will continue with an initially empty song database. If the user answers negatively, the application will exit. An example of an appropriate start-up command is as follows:

   java SongDB mySongDB.txt

2. The original state of the GUI window shall display the combo box with the first song in the database selected or blank if using an empty song database. The Title (optionally, see below), Item Code, Description, Artist, Album, and Price fields for this item shall be displayed with the fields disabled (non-editable) or be blank if using an empty song database. For a non-empty song database, the Add, Edit, Delete, and Exit buttons shall be enabled, and the Accept and Cancel buttons shall be disabled. For an empty song database, only the Add and Exit buttons shall be enabled with the rest disabled.

   Note in the example images, the title of the current song is contained in the combo box. Also when adding a song, the song title is entered in the combo box. You can choose to do it this way or you can add an extra Title text box field when adding a song (see below).

3. To add a song, the user clicks on the Add button. This will cause the application to clear and enable the Item Code, Description, Artist, Album, and Price fields so the user can enter information for the new song. *If the combo box is not being used to enter a new song title, a new text field should be supplied to enter the song title*. The Edit and Delete buttons are also disabled, and the Accept and Cancel buttons are enabled. When the user presses the Accept button, the new song will be added to the database and combo box. If the user presses the Cancel button, the entry transaction is canceled and the frame reverts to its original state.

4. To edit an existing song, the user shall select the song from the combo box. The information for the selected song shall then be displayed in the disabled fields. The user then presses the Edit button, which enables the Description, Artist, and Price fields. The user may not change the song title or Item Code. The Add, Edit, and Delete buttons are disabled, and the Accept and Cancel buttons are enabled. If the user presses the Accept button, the changes are saved and the combo box is updated. If the user presses the Cancel button, the edit transaction is canceled and the frame reverts to its original state.

5. To delete an existing song, the user shall select the song from the combo box. The information for the song shall then be displayed in the disabled fields. If the user presses the Delete button, the song is deleted from the database and the combo box.

6. When the user presses the Exit button the application shall terminate. The current state of the database shall be saved in a file using the pathname supplied at program start-up (see above).

7. Data fields shall be validated to ensure they are not blank with appropriate error messages displayed in the GUI interface (either in a otherwise hidden text field or as a separate dialog box).

If the song is a single, the text “None” will be used for the album name. The price field shall be validated to ensure only numeric data is entered.

Other Activates:

1. Create a compressed zipped folder containing your Design and Analysis document, your Java source code files, and your javadocs folder.

2. Submit your compressed zipped folder as directed by your instructor.

Assignment Rubric:

Part	70%	80%	90%	100%	% of Grade
Design and Analysis Document	All but one subject addressed with relevant, information. Few minor typographical issues. Document is close to assigned length	All assigned subjects address with mostly relevant information. Nicely formatted document. Document is close to assigned length	All assigned subjects address with accurate and relevant. Nicely formatted document. Document is within assigned length	All assigned subjects address with accurate, relevant, and insightful information. Very nicely formatted. Document is within assigned length	15%
Functionality Note: Compilation errors and warnings are part of this evaluation.	Majority of required function parts work as indicted in the assignment text. One major or 3 minor defects. All major functionality at least partially working (example change provided but not correct). Design document does not fully reflect functionality.	Most required function parts work as indicted in the assignment text above and submitted documentation. One major or 3 minor defects. All major functionality at least partially working ((example change provided but not correct).	Nearly all required function parts work as indicted in the assignment text above and submitted documentation. One to two minor defects.	All required function parts work as indicted in the assignment text above and submitted documentation.	60%
Code	Majority of the code conforms to coding standards as explained and demonstrated so far in the course	Most of the code conforms to coding standards as explained and demonstrated so far in the course	Almost all code conforms to coding standards as explained and demonstrated so far in the course (ex.	All code conforms to coding standards as explained and demonstrated so far in the course	25%

	(ex. method design, naming, formatting, etc.). Five to six minor coding standard violations. Some useful comments. Some JavaDocs commenting. Code compiles with multiple warnings or fails to compile with difficult to diagnose error.	(ex. method design, naming, formatting, etc.). Three to four minor coding standard violations. Mostly useful comments. Public class JavaDocs complete. Code compiles with one to two warnings.	method design, naming, formatting, etc.). One to two minor coding standard violations. Appropriate level of useful comments. Public class JavaDocs complete. Code compiles. Code compiles with no errors or warnings.	(ex. method design, naming, formatting, etc.). Appropriate level of useful comments. Complete JavaDocs as specified. Code compiles with no errors or warnings.	