**CSC 335 Analysis and Design Artifacts for Jukebox**

*Each team complete this form, put it in your project in a folder named* **doc** *and push to Github. This will be part of your Iteration 1 grade*

**1) Team Members**: \_\_Abdullah\_\_Asaad\_\_\_\_ \_\_\_\_\_Gavin Daniel\_\_\_\_\_\_\_\_

**2) Candidate Objects**

List the most important objects, or an inheritance hierarchy name, and the responsibility of each.

|  |  |
| --- | --- |
| **Candidate Object** | **Responsibility in 1 or 2 sentences** |
| Song | Holds relevant information about song (e.g. title, length, file path, times played) |
| User | Holds account information such as ID, password, current playlist |
| SongQueue | Each user has a song queue. Object contains Song objects and operates in FIFO order, validates songs before adding to ensure maximum limits have not been exceeded. |
| Jukebox | Holds a list of users. Handles login/logout functionality. Displays a GUI used to control model. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

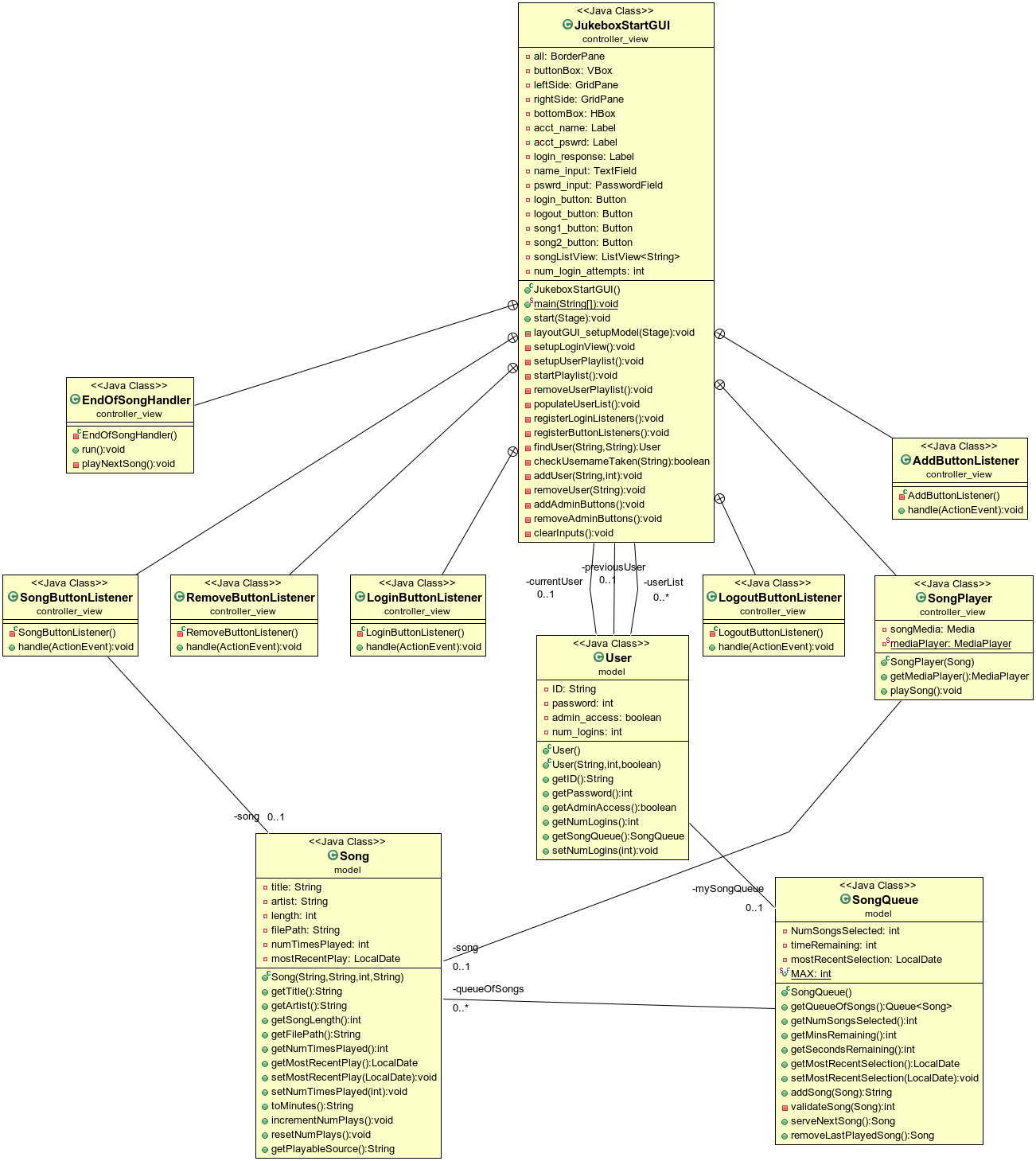
**3) Sequence Diagram:** Write a UML Sequence Diagram should show the most important scenario you can think of. Your sequence diagram should show most of your candidate objects you listed above and how they communicate with each other.

**Recommended:** *Use the Sequence Diagram editor found at*

<https://www.websequencediagrams.com/>

*Export and Image and copy and paste it here*

**4) Class Diagram:** Write a UML Class Diagram that shows all of your candidate objects from above. Show any relationships between them the classes such as inheritance or interface implementation. Draw general associations such as dependency or aggregation. Label some to help explain things. Add any multiplicity adornments that seem appropriate. Use notes to explain things if you feel it will help. Each UML class must show the class name. For full credit, each class must have an average of at least one attribute per class. There must be an average of about 1.5 methods per class.



**5) Estimate and Assign Tasks** For each Iteration 1 task, estimate its difficulty using the numbers 1, 2, 3, 5, or 8. These are points that represent the relative complexity of the task. Mark 8 for the most difficult and/or time consuming and 1 for what appears to be the easiest. Indicate which person (s) will complete the task before iteration 1 due date.

|  |  |  |
| --- | --- | --- |
| **Points** | **Who will**  **complete this?** | **Task** |
| 2 | Abdullah &  Gavin | Your GitHub repo has a completed copy of this analysis and design document in a folder named **doc** that represents a serious effort to analyze and design Jukebox |
| 3 | Gavin | Users can log in and log out |
| 3 | Abdullah | Songs can be ***selected*** up to a maximum of 3 times per calendar date. Use the time the song is added to the song queue, not when the song ends |
| 3 | Abdullah | Any valid user can select a maximum of 3 songs per calendar date. Use the time the song is added to the queue, not when the song ends. Reset time occurs at midnight, so users could have three new plays tomorrow.  A song that has played 3 times today, could be played 3 times tomorrow. |
| 3 | Gavin | The administrator can add and remove accounts |
| 2 | Abdullah | Songs can be played in FIFO order, like a Jukebox, the first one plays on order. |
| 8 | Abdullah &  Gavin | Complete a functional spike to determine the interactions are actually working. JukeboxStartGUI.java in package controller is an event-driven program with a graphical user interface to affirm the functionality all Iteration 1 tasks have been completed and are working correctly.  We will use this to test your code for the first 100 points of Jukebox. |