**C Sc 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*

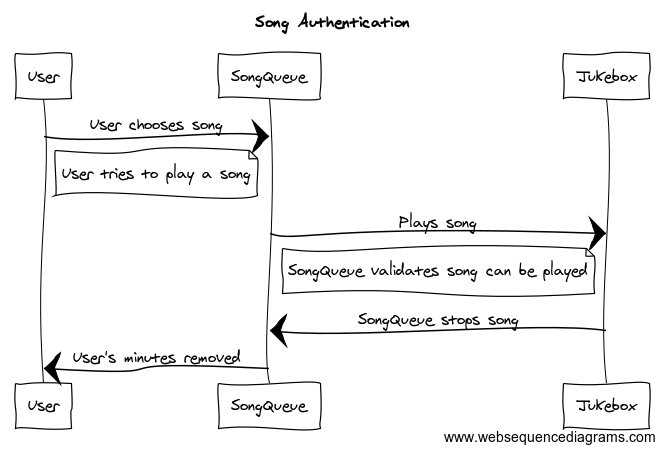
**1) Team Members**: \_\_\_\_\_Andrew Marrufo\_\_\_\_\_\_ \_\_\_\_\_\_\_Derian Acuna\_\_\_\_\_\_

**2) Candidate Objects**

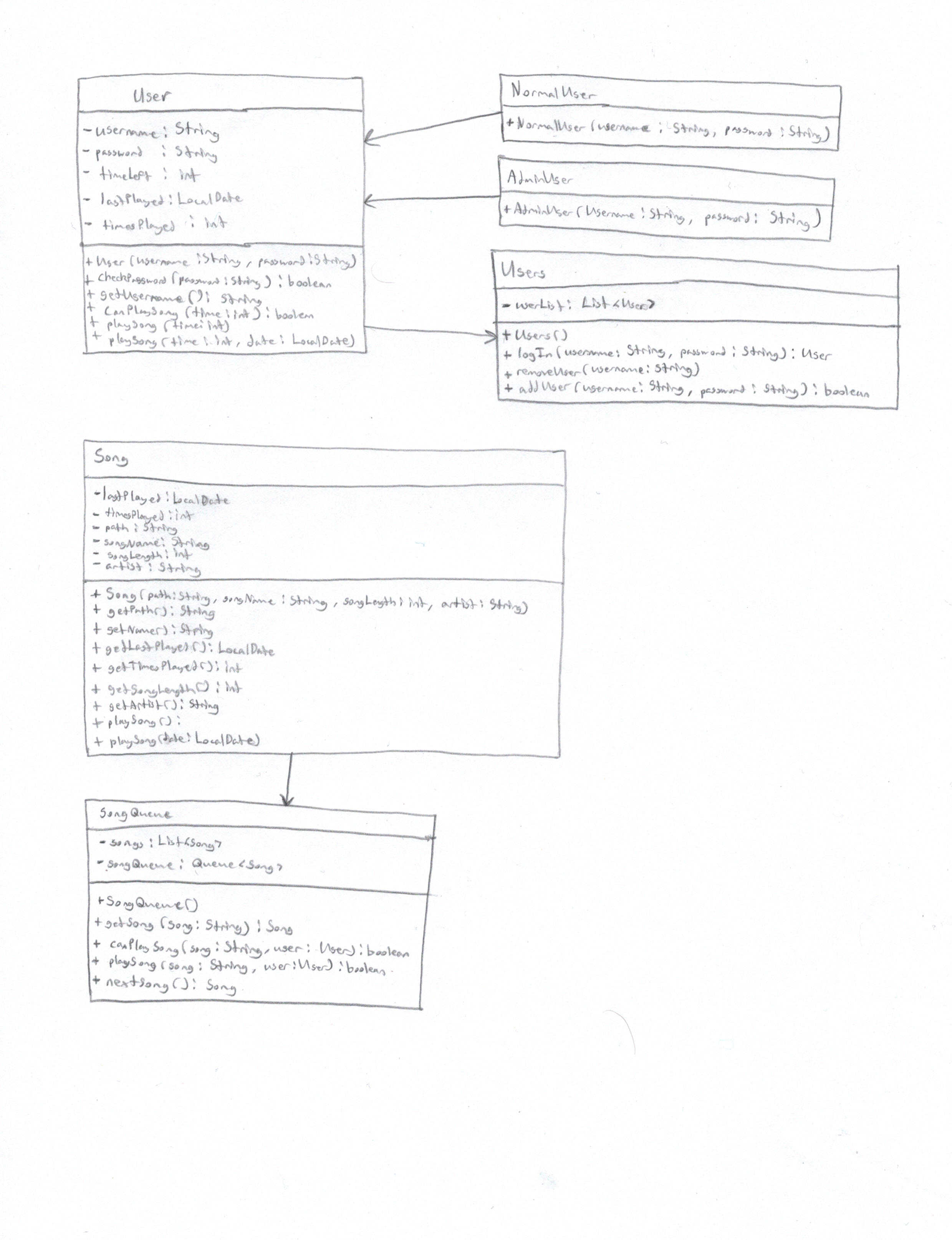
List the most important objects, or an inheritance hierarchy name, and the responsibility of each.

|  |  |
| --- | --- |
| **Candidate Object** | **Responsibility in 1 or 2 sentences** |
| User | An abstract class for which to organize all types of users. |
| NormalUser | An object for standard users without administrative permissions. |
| AdminUser | An object for elevated users with administrative permissions. |
| Song | An object to keep the information of the song that the user has selected. |
| SongQueue | An object to organize songs into a queue, and lets the user know if a certain song can be played. |
| Users | An object that allows for the organization of users and to tell if a user can be added/removed from the list. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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.



**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** |
| 1 | Andrew | 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 |
| 2 | Derian | Users can log in and log out |
| 2 | Derian | 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 |
| 5 | Derian | 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 | Derian | The administrator can add and remove accounts |
| 2 | Derian | Songs can be played in FIFO order, like a Jukebox, the first one plays on order. |
| 8 | Andrew | 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. |

**Iteration 2**

**Estimate and Assign Tasks** For each Iteration 2 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 2 due date.

|  |  |  |
| --- | --- | --- |
| **Points** | **Who will  complete this?** | **Task** |
| 3 | Andrew | TableView of selectable songs (displays the number of times the song played today with its title, artist, and play time. Should be sortable by column) |
| 3 | Andrew | ListView of song queue (selected songs are played in FIFO order, when a song finishes, it is removed from the ListView of songs to play) |
| 2 | Derian | When a song is selected, the system shows an appropriate message if the song cannot be selected. There are several reasons why a song cannot be selected. |
| 2 | Andrew | When a song is successfully selected  - the song is added to the end of the ListView of songs to play  - the song selector updates times played today  - the user status updates |
| 5 | Derian | A way to persist user data/song data, and current song queue.  - the song that was playing begins playing again, even if it is tomorrow.  - the queue of songs is shown as it was when the program terminated. It behaves the same to show new selections and songs finish playing  - the song selector shows the same number of times each has been played |