Student ID_____**Student Name**_____

Advanced Software Development DE

Midterm Exam April 6 2019

PRIVATE AND CONFIDENTIAL

1. **Allotted exam duration is 2 hours.**
2. **Closed book/notes.**
3. **No personal items including electronic devices (cell phones, computers, calculators, PDAs).**
4. **Cell phones must be turned in to your proctor before beginning exam.**
5. **No additional papers are allowed. Sufficient blank paper is included in the exam packet.**
6. **Exams are copyrighted and may not be copied or transferred.**
7. **Restroom and other personal breaks are not permitted.**
8. **Total exam including questions and scratch paper must be returned to the proctor.**

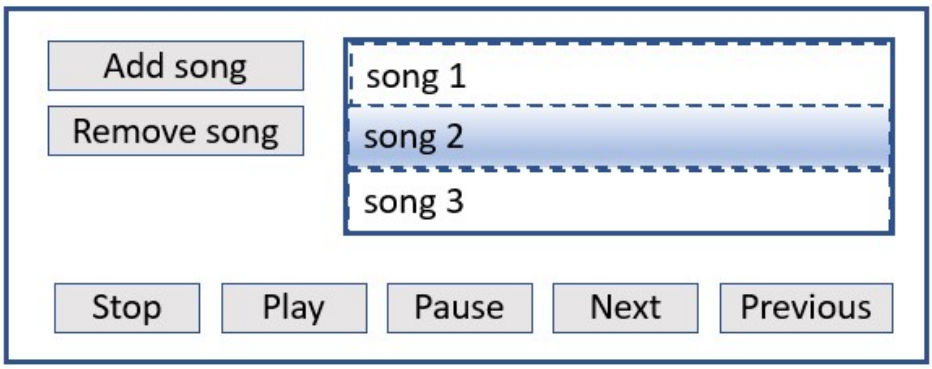
6 blank pages are provided for writing the solutions and/or scratch paper. All 6 pages must be handed in with the exam

BE VERY CAREFUL WITH THE GIVEN 2 HOURS AND USE YOUR TIME WISELY. THE ALLOTED TIME IS GIVEN FOR EVERY QUESTION.

Write your name and student id at the top of this page.

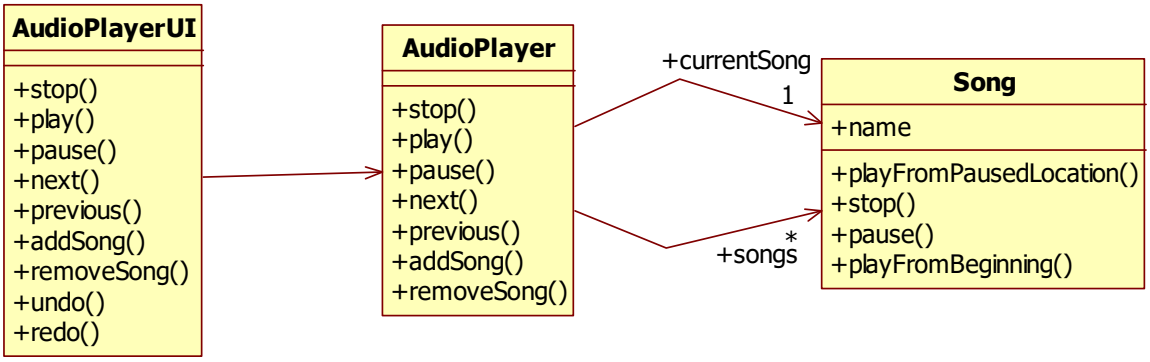
Question 1 [45 points] {50 minutes}

Suppose you need to design and implement an audio player that has the following user interface:



You can add and remove songs to a list, and the buttons **Stop, Play** and **Pause** operate at the selected song. You can change the selected song with the **Next** and **Previous** buttons.

Your first design looks like this:



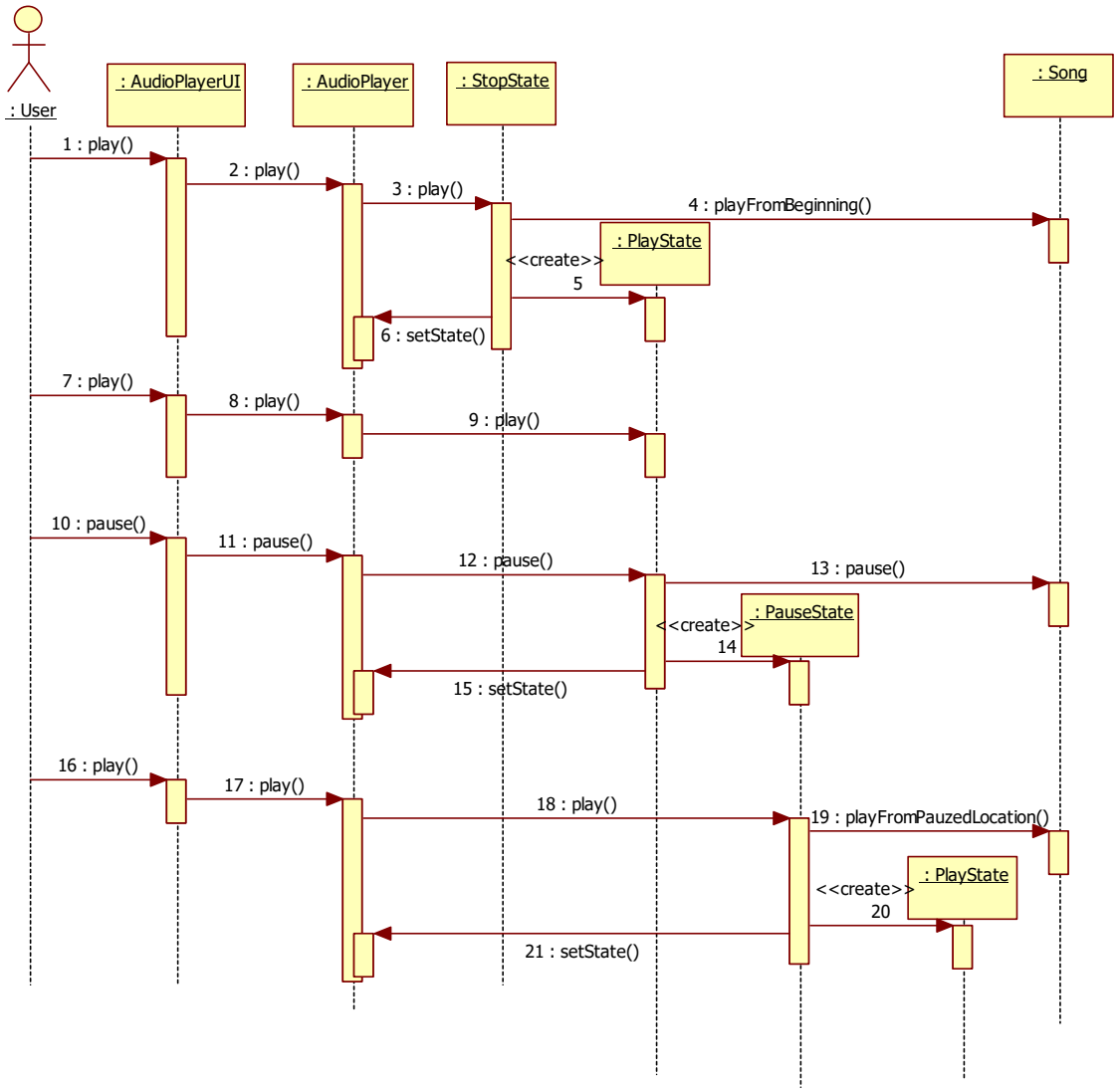
The problem with this design is that the AudioPlayer class contains a lot of conditional logic. For example if we click the Play button, it depends on the current state of the audio player what will happen. The play() method looks like this:

```
public void play() {
  if (currentState.equals("stop")) {
        currentSong.playFromBeginning();
        currentState = "play";
  } else if (currentState.equals("pauze")) {
        currentSong.playFromPauzedLocation();
        currentState = "play";
  }
}
```

We learned that if we use the State pattern, we can get rid of this complex conditional logic.

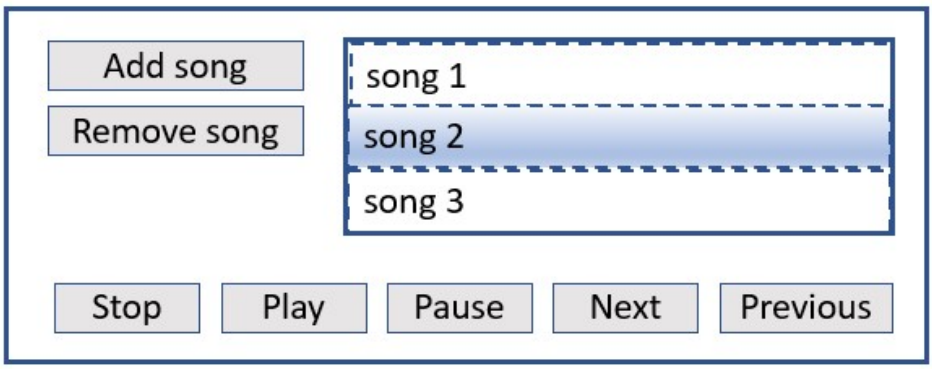
Draw the sequence diagram that shows clearly how your design works if you use the state pattern in this application. The initial state of the audio player is stop, so assume the audio player is already in the stop state when the sequence diagram starts. Show the sequence diagram of the following user actions:

1. The user presses the play button
2. The user presses the play button again
3. The user presses the pause button
4. The user presses the play button



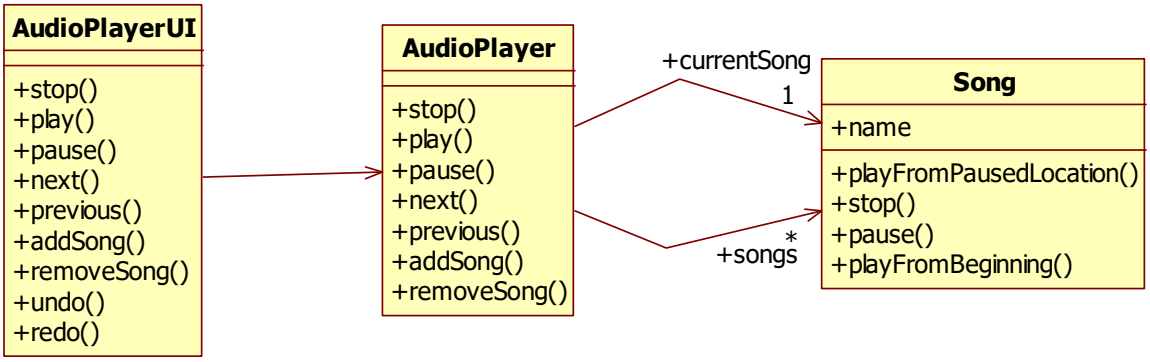
Question 2 [40 points] {50 minutes}

Suppose you need to design and implement an audio player that has the following user interface:

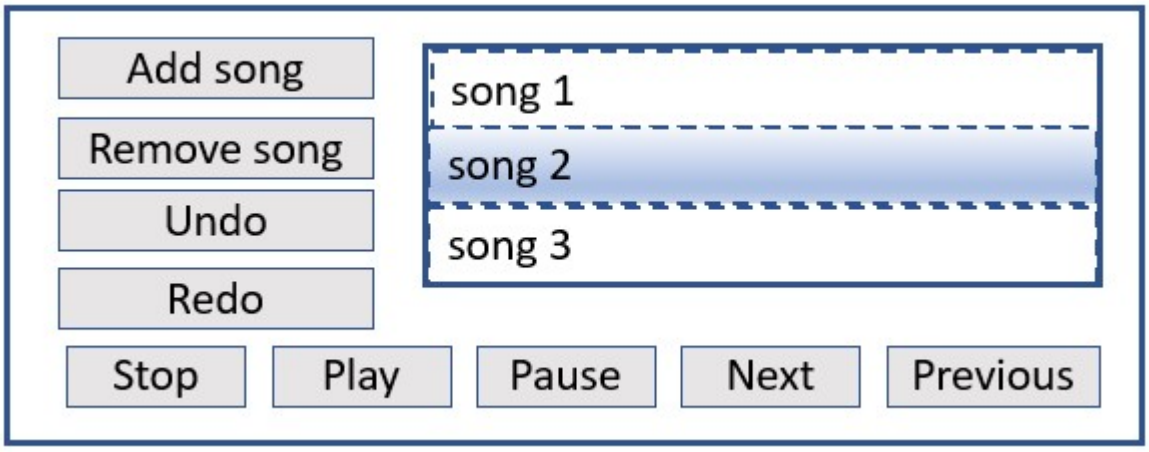


You can add and remove songs to a list, and the buttons **Stop, Play** and **Pause** operate at the selected song. You can change the selected song with the **Next** and **Previous** buttons.

Your first design looks like this:



But now you get a new requirement that the audio player also need undo/redo support.

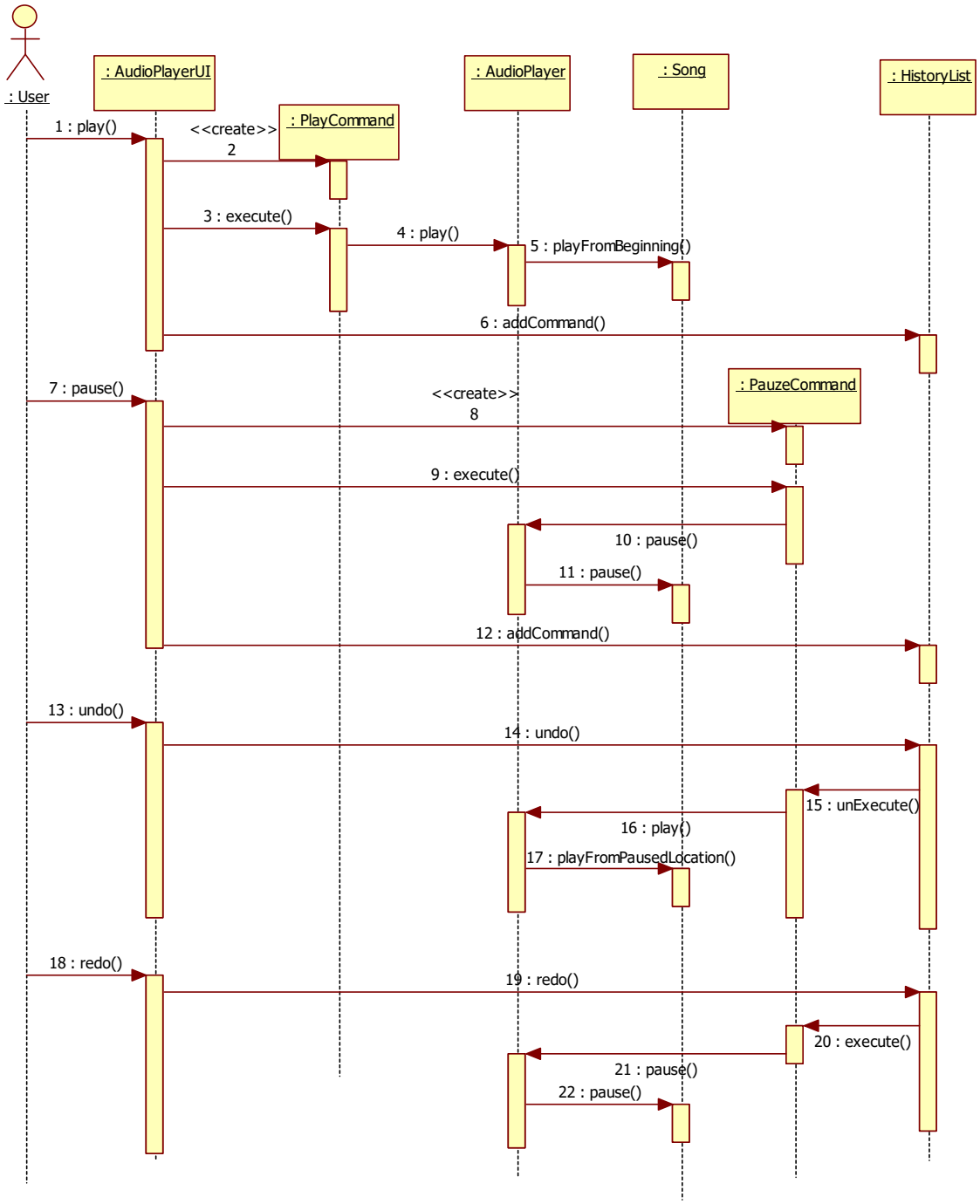


In question 1 we saw already that the state pattern can be used to get rid of the complex conditional logic in the AudioPlayer class. For this question you can assume that we don't care about this conditional logic. For this question, do **NOT apply the state pattern**.

For this question we only need to implement the **undo/redo** functionality.

Draw the sequence diagram that shows clearly how your new design works. Suppose the last action that is done is clicking the Stop button. So your sequence diagram starts when the audio player is in the Stop state. Show the sequence diagram of the following user actions:

1. The customer clicks the Play button
2. The customer clicks the Pause button
3. The customer clicks the Undo button
4. The customer clicks the Redo button


: User
: AudioPlayerUI
: PlayCommand
: AudioPlayer
: Song
: HistoryList
: PauzeCommand
1 : play()
<<create>>
2
3 : execute()
4 : play()
5 : playFromBeginning()
6 : addCommand()
7 : pause()
<<create>>
8
9 : execute()
10 : pause()
11 : pause()
12 : addCommand()
13 : undo()
14 : undo()
15 : unExecute()
16 : play()
17 : playFromPausedLocation()
18 : redo()
19 : redo()
20 : execute()
21 : pause()
22 : pause()

Question 3 [10 points] {10 minutes}

For the observer pattern we learned that there are 2 options we can use when we apply the observer pattern.

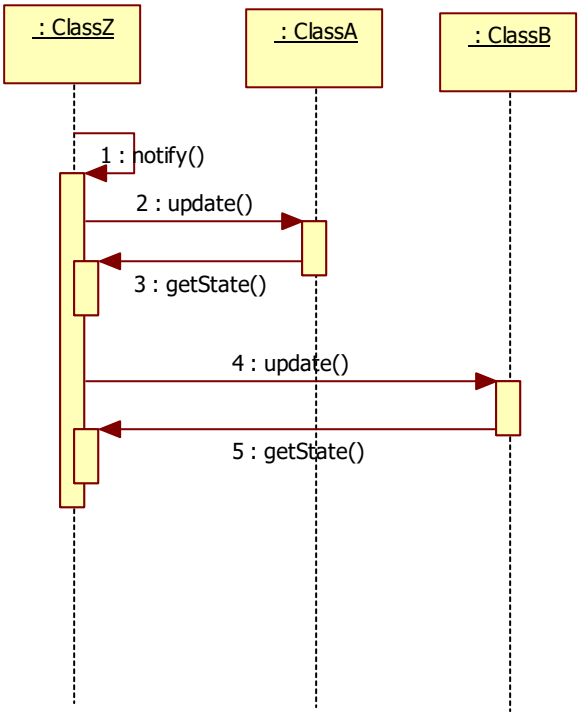
1. The pull model
2. The push model

Suppose classA and classB are observers of classZ.

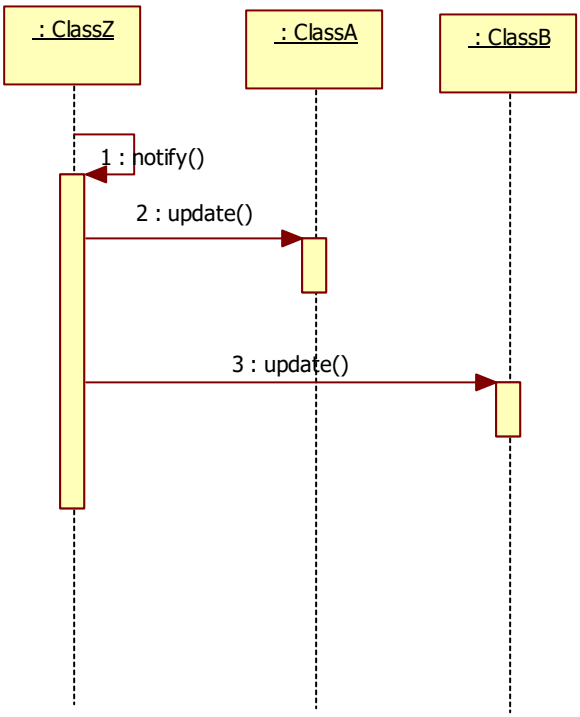
a. Draw the sequence diagram that shows how the observer pattern works when we use the pull model.
b. Draw the sequence diagram that shows how the observer pattern works when we use the push model.

The sequence diagrams should make it very clear what the difference is between these 2 models.

Pull:



Push:



Question 4 [5 points] {10 minutes}

Describe how we can relate the **Mediator** pattern to one or more principles of SCI. Your answer should be about half a page, but should not exceed one page (handwritten). The number of points you get for this questions depends how well you explain the relationship between the **Mediator** pattern and one or more principles of SCI.