Object Oriented Analysis and Design Project Part 4

Team: Dea Allen, Sesha Sailendra Chetlur, Pravin Venkatesh Venkataraman
Title: Media Player
Project Summary: Our goal was to design and build a Media Player that could be extensively customized to specific user's needs. We made extension to the core part of the application possible through inclusion of modules or plug-ins.

1. What features were implemented?

   Features that were implemented:

   a. Play media
      i. Play a selected media regardless of the format through a click of the button that reads "Play".
   b. Stop media
      i. Stop a playing media through a click of the button that reads "Stop".
   c. Open media
      i. File browser could be opened by clicking on the menu item "Open" under menu "File".
      ii. A media file could be selected there on to be included inside the Media Player.
   d. Seek media
      i. Seek bar could be used to fast forward to any specific point in the media time frame by dragging it along the horizontal line underneath.
   e. Add file to playlist
      i. The file playing could be added to a selected playlist by just clicking on the button with a '+' sign on it.
   f. Delete file from playlist
      i. This could be achieved by just clicking on the button with '-' sign on it.
   g. Create playlist
      i. An empty playlist is created when user clicks "Create Playlist".
   h. Delete playlist
      i. The selected playlist could be deleted by clicking on the button with a "X" on it.
   i. Add custom module
      i. A module could be included through the menu item "Add module" under "Module" menu. This opens up a file browser for the user to select the desired *.java file.
   j. Add custom skin

i. Different skins could be included just as any regular modules through the menu item “Add module” under “Module” menu. This opens up a file browser for the user to select the desired *.java file.

2. Which features were not implemented from Part 2?

   Features that were not implemented from Part 2:

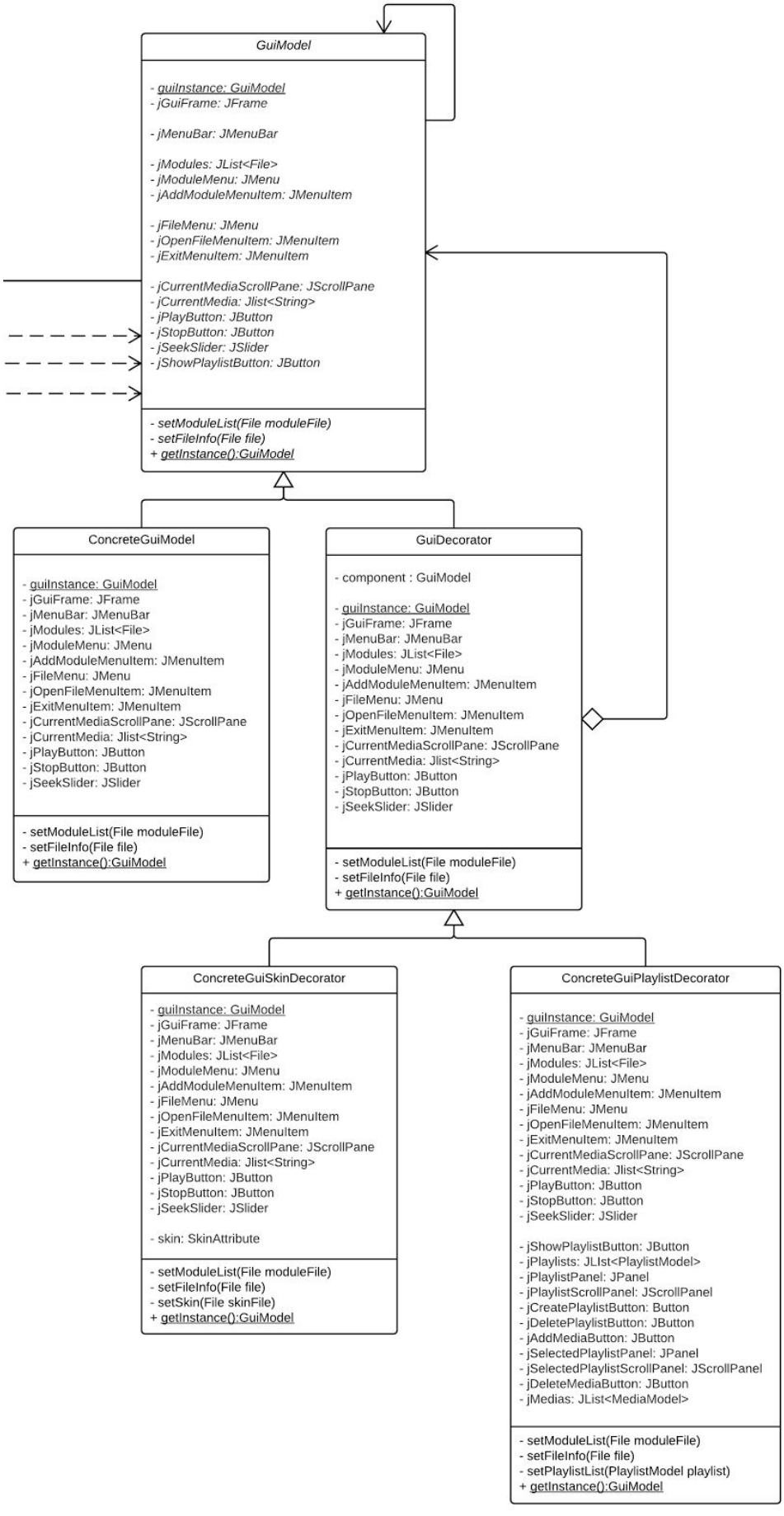
   We did not implement the following use cases:

   - Bookmark media
   - Crop media

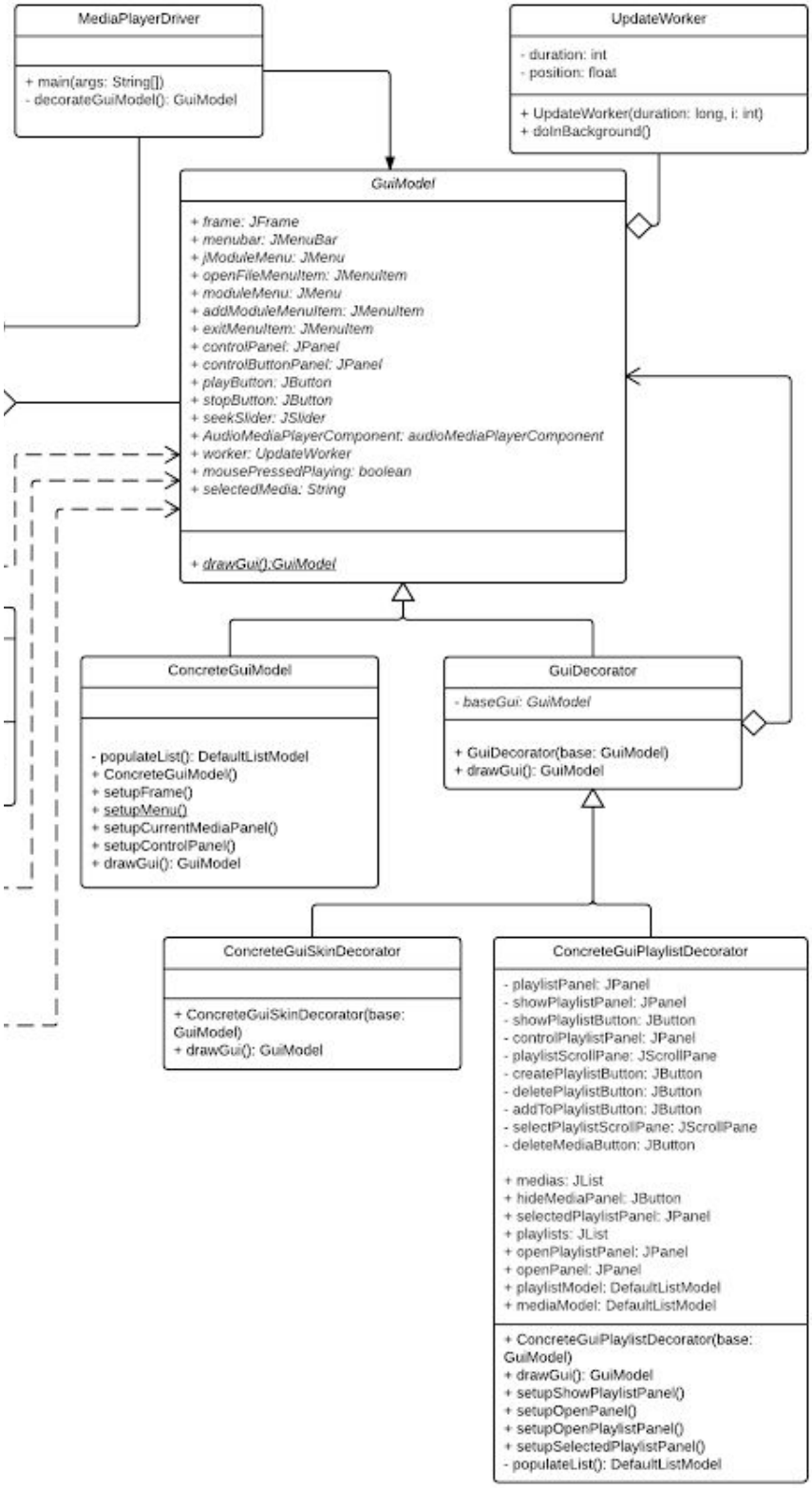
3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.

   We answer this question by discussing the changes in separate sections of the class diagram:

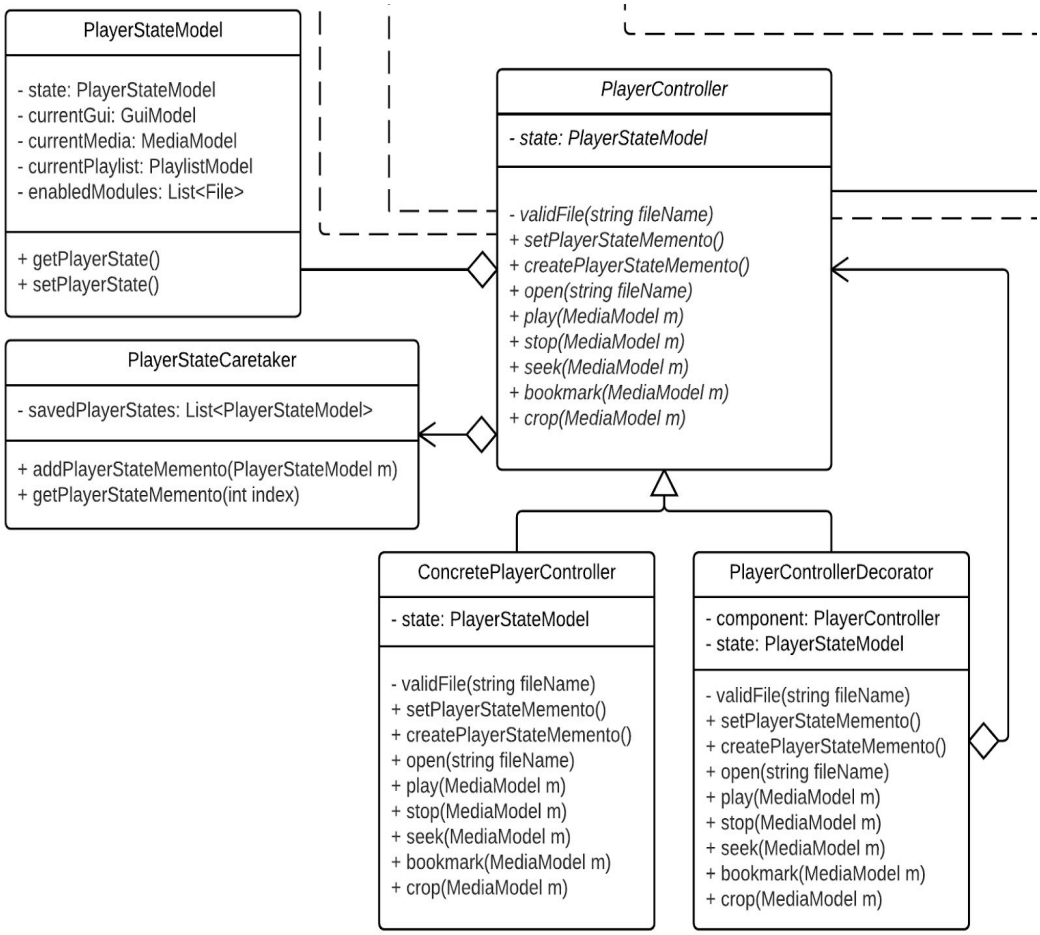
   I> GUI decorator: The idea was to decorate the GUI so that plug-ins such as skins and playlist options could be added to the player. This part of the class diagram did not change in terms of design. The attributes inside the classes varied slightly, but that was expected. Our initial design is below.


GuiModel
- guiInstance: GuiModel
- jGuiFrame: JFrame
- jMenuBar: JMenuBar
- jModules: JList<File>
- jModuleMenu: JMenu
- jAddModuleMenuItem: JMenuItem
- jFileMenu: JMenu
- jOpenFileMenuItem: JMenuItem
- jExitMenuItem: JMenuItem
- jCurrentMediaScrollPane: JScrollPane
- jCurrentMedia: Jlist<String>
- jPlayButton: JButton
- jStopButton: JButton
- jSeekSlider: JSlider
- jShowPlaylistButton: JButton
- setModuleList(File moduleFile)
- setFileInfo(File file)
+ getInstance():GuiModel
ConcreteGuiModel
- guiInstance: GuiModel
- jGuiFrame: JFrame
- jMenuBar: JMenuBar
- jModules: JList<File>
- jModuleMenu: JMenu
- jAddModuleMenuItem: JMenuItem
- jFileMenu: JMenu
- jOpenFileMenuItem: JMenuItem
- jExitMenuItem: JMenuItem
- jCurrentMediaScrollPane: JScrollPane
- jCurrentMedia: Jlist<String>
- jPlayButton: JButton
- jStopButton: JButton
- jSeekSlider: JSlider
- setModuleList(File moduleFile)
- setFileInfo(File file)
+ getInstance():GuiModel
GuiDecorator
- component : GuiModel
- guiInstance: GuiModel
- jGuiFrame: JFrame
- jMenuBar: JMenuBar
- jModules: JList<File>
- jModuleMenu: JMenu
- jAddModuleMenuItem: JMenuItem
- jFileMenu: JMenu
- jOpenFileMenuItem: JMenuItem
- jExitMenuItem: JMenuItem
- jCurrentMediaScrollPane: JScrollPane
- jCurrentMedia: Jlist<String>
- jPlayButton: JButton
- jStopButton: JButton
- jSeekSlider: JSlider
- setModuleList(File moduleFile)
- setFileInfo(File file)
+ getInstance():GuiModel
ConcreteGuiSkinDecorator
- guiInstance: GuiModel
- jGuiFrame: JFrame
- jMenuBar: JMenuBar
- jModules: JList<File>
- jModuleMenu: JMenu
- jAddModuleMenuItem: JMenuItem
- jFileMenu: JMenu
- jOpenFileMenuItem: JMenuItem
- jExitMenuItem: JMenuItem
- jCurrentMediaScrollPane: JScrollPane
- jCurrentMedia: Jlist<String>
- jPlayButton: JButton
- jStopButton: JButton
- jSeekSlider: JSlider
- skin: SkinAttribute
- setModuleList(File moduleFile)
- setFileInfo(File file)
- setSkin(File skinFile)
+ getInstance():GuiModel
ConcreteGuiPlaylistDecorator
- guiInstance: GuiModel
- jGuiFrame: JFrame
- jMenuBar: JMenuBar
- jModules: JList<File>
- jModuleMenu: JMenu
- jAddModuleMenuItem: JMenuItem
- jFileMenu: JMenu
- jOpenFileMenuItem: JMenuItem
- jExitMenuItem: JMenuItem
- jCurrentMediaScrollPane: JScrollPane
- jCurrentMedia: Jlist<String>
- jPlayButton: JButton
- jStopButton: JButton
- jSeekSlider: JSlider
- jShowPlaylistButton: JButton
- jPlaylists: JList<PlaylistModel>
- jPlaylistPanel: JPanel
- jPlaylistScrollPanel: JScrollPanel
- jCreatePlaylistButton: Button
- jDeletePlaylistButton: JButton
- jAddMediaButton: JButton
- jSelectedPlaylistPanel: JPanel
- jSelectedPlaylistScrollPanel: JScrollPanel
- jDeleteMediaButton: JButton
- jMedias: JList<MediaModel>
- setModuleList(File moduleFile)
- setFileInfo(File file)
- setPlaylistList(PlaylistModel playlist)
+ getInstance():GuiModel

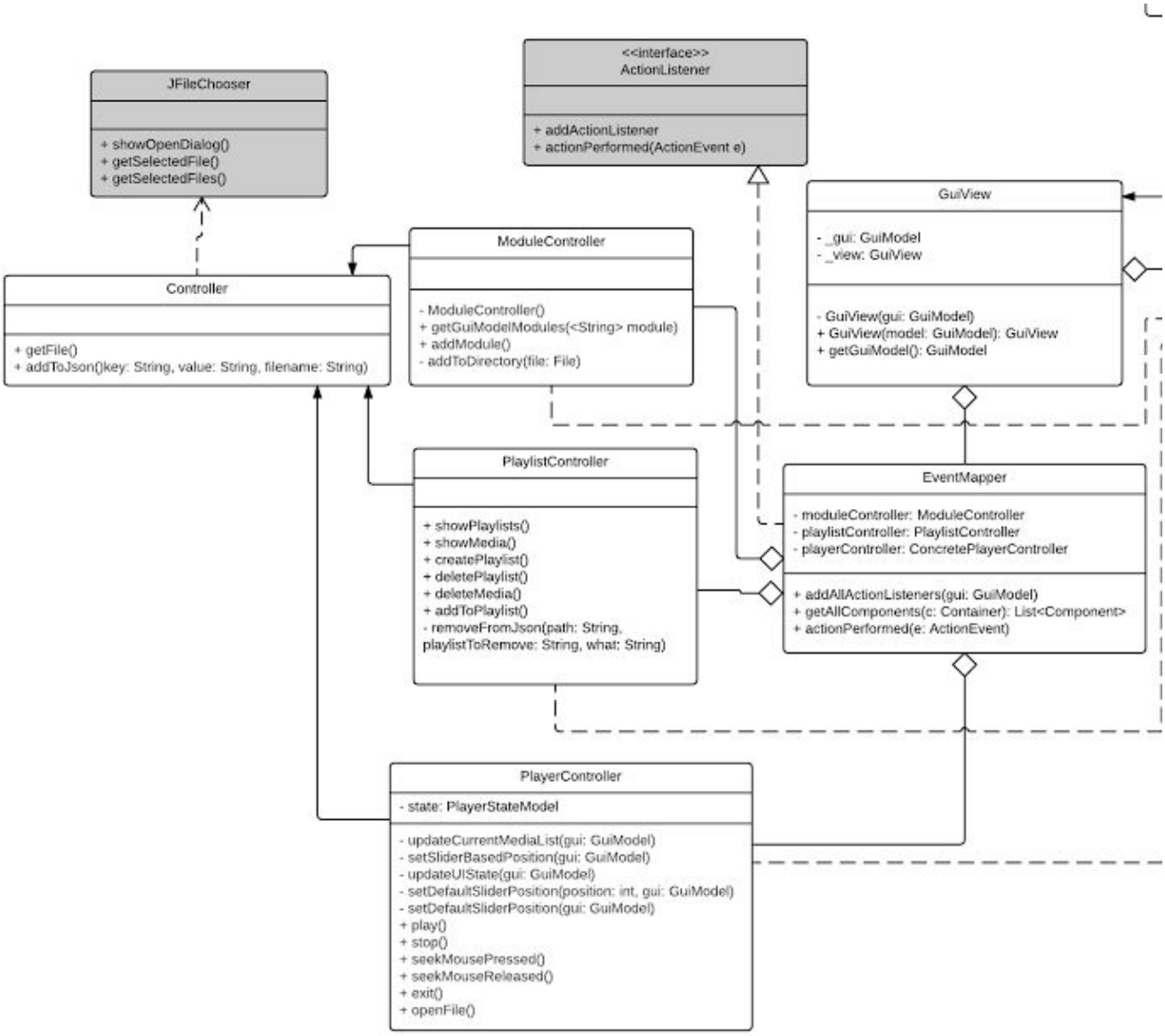
This is our final GUI decorator design:



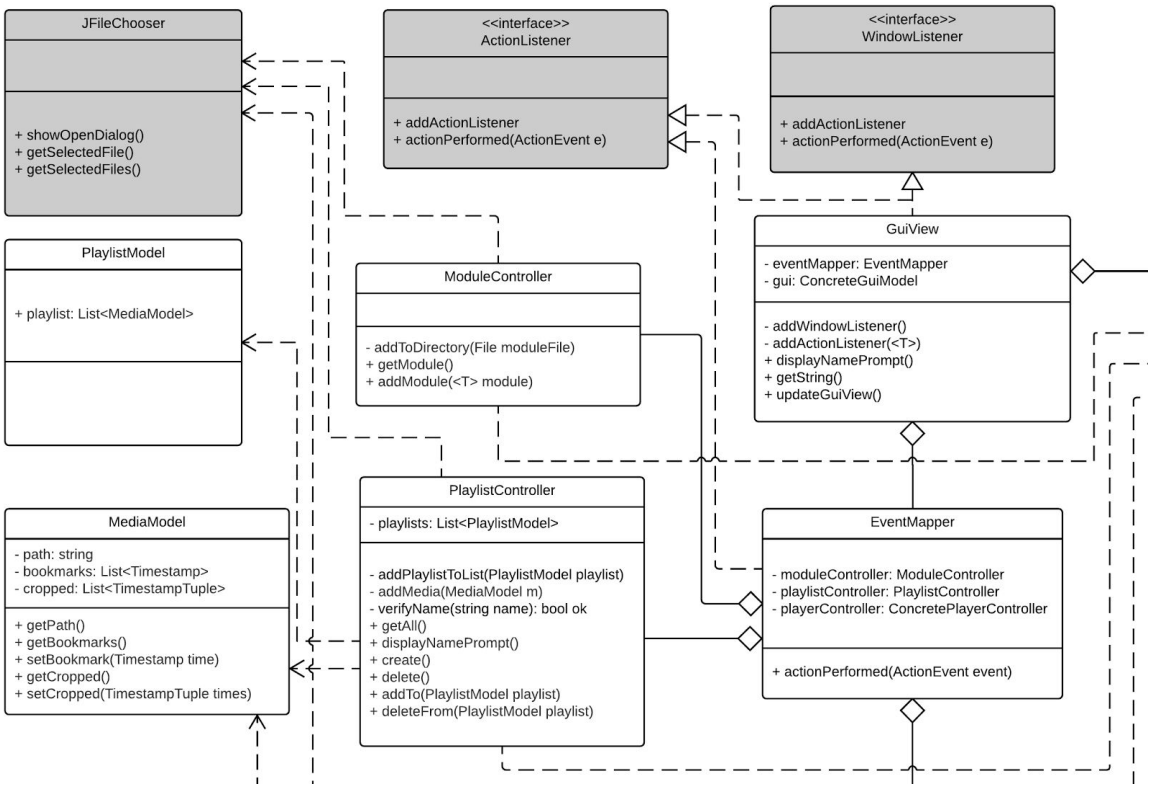
II> Player Controller: The initial idea was to make the player controller logic to be decoratable to allow developers to add custom logic to the player itself. In addition, the memento pattern was imagined to try and save the state of the player before every plug-in or player exit in order to ensure that the state of the player could be reverted to the last stable state in case of an unexpected crash. The below diagram shows the idea:



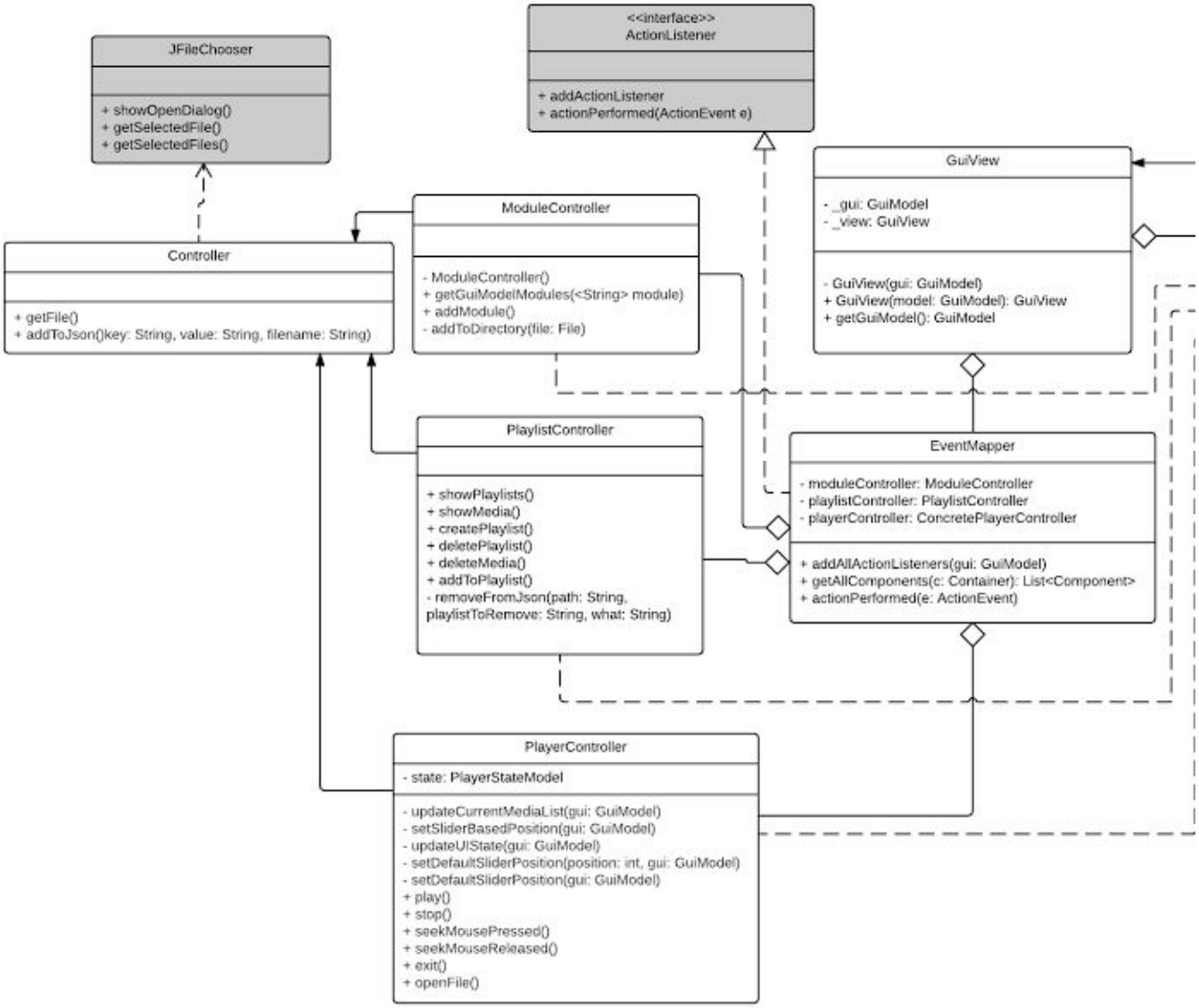
However, we were unable to finish implementing the memento pattern due to time constraints. We decided not to decorate the PlayerController based on our experience adding playlist functionality as a module. The playlist module included a PlaylistGuiModelDecorator and a PlaylistController. The PlaylistController had completely separate functionality than the PlayerController, so we decided it was better to follow the principle of separation of concerns. Similarily, the SkinGuiModelDecorator that we added did not need a controller for its functionality; therefore, decorating the PlayerController did not make sense. Our final class diagram is as follows. It shows all the controllers along with the player controller:



III> The remaining section that showed the observer pattern was initially as follows:



This didn’t change much either, except for the names of some attributes and the addition of some helper functions:



4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

   Design patterns used in the implementation,

   - Observer Pattern:
     - We used the observer pattern built into Java Swing. We implemented an event mapper class that added action listeners to all JComponents in our decorated view. Our controllers then received notifications of user interaction through the event mapper.
   - Decorator Pattern:
     - This pattern was used in the context of the GUI. The main feature of this project was to allow developers to plug in their modules to customize the

player. We decorated the GUI to allow a developer to plug in modules to customize the GUI look and feel.

- Memento Pattern:
    - We considered using the memento pattern to save the player state to make the process of adding and removing modules more safe. The idea was to give the user a fallback point if a module interfered with the normal player functionality. We did not have enough time to implement this design pattern.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

    We have learned that modularization of code is extremely beneficial, especially in the stage of conception of code structure. Mapping down classes, their relations and all the functions in the right places beforehand makes coding logic much easier, as the layout is well defined. We avoid “the blob”.

    The other thing we realized is that requirements gathering is a bit of a task. The initial planning made us really think about what we needed to do, and made us validate our ideas and visions. Without having done this initially, we believe we would have ended up asking a lot of questions during implementation, and would have wasted a lot of time going back and forth, unable to decide how to proceed.

    The point is, we knew what we were doing and knew where we were going as a result of the initial planning. This ensured that our goal was well defined, and thus planning the execution, distributing the load and actually coding logic was easy to finish in a stipulated time span. We realized that without the planning and the diagrams, the estimation of the time required to finish the implementation, and the distribution of workload would have both been very difficult.