The Course Site Generator™

Software Design Description

Author: Richard McKenna
Navin Abichandani
Debugging Enterprises™

Abstract: This document describes the software design for the Course Site Generator™, a program that will be used to help a course instructor build and update a course website.

Based on IEEE Std 1016™-1998 (R2009) document format

1 Introduction

This is the Software Design Description (SDD) for the Course Site Generator™ application. Note that this document format is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

This document is to serve as the blueprint for the construction of the Course Site Generator application. This design will use UML class diagrams to provide complete detail regarding all packages, classes, instance variables, class variables, and method signatures needed to build the application. In addition, UML Sequence diagrams will be used to specify object interactions post-initialization of the application, meaning in response to user interactions or timed events.

1.2 Scope

For this project the goal is to provide instructors teaching courses with the ability to easily make and update course Web sites. There is a common structure to the pages and so there will be limitations on customization, but any instructor in any department at any University will be able to create a site from this application.

1.3 Definitions, acronyms, and abbreviations

Case Diagram – A UML document format that specifies how a user will interact with a system.

Class Diagram – A UML document format that describes classes graphically. Specifically, it describes their instance variables, method headers, and relationships to other classes.

Desktop Java Framework – The software framework to be developed in tandem with the Course Site Generator application such that additional workspace-style applications can easily be constructed.

Document Object Model (DOM) – a tree data structure maintained by the browser that contains all content for the currently loaded Web page.

Framework – In an object-oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

GUI – Graphical User Interface, visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

HyperText Markup Language – a markup language used to describe Web pages. Web pages are text files encoded in HTML that can employ JavaScript and Stylesheets to build and style content.

IEEE – Institute of Electrical and Electronics Engineers, the "world's largest professional association for the advancement of technology".

Java – A high-level programming language that uses a virtual machine layer between the Java application and the hardware to provide program portability.

Sequence Diagram – A UML document format that specifies how object methods interact with one another.

JavaScript – the default scripting language of the Web, JavaScript is provided to pages in the form of text files with code that can be loaded and executed when a page loads so as to dynamically generate page content in the DOM.

Properties Manager – Framework that uses loading properties from XML files to be used all over the application.

Stylesheet – a static text file employed by HTML pages that can control the colors, fonts, layout and other style components in a Web page.

UML – Unified Modeling Language, a standard set of document formats for designing software graphically. Use

1.4 References

IEEE Std 830™-1998 (R2009) – IEEE Recommended Practice for Software Requirements Specification

Course Site Generator™ SRS – Debugging Enterprises' Software Requirements Specification for the Course Site Generator application.

1.5 Overview

This Software Design Description document provides a working design for the Course Site Generator software application as described in the Course Site Generator Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks to be designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to specify how the classes should be constructed. Section 4 will provide the Method-Level System Viewpoint, describing how methods will interact with one another. Section 5 provides deployment information like file structures and formats to use. Section 6 provides a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using the Creately editor and Violet UML editor.

2 Package-Level Design Viewpoint

As mentioned, this design will encompass both the Course Site Generator application and the Desktop Java Framework to be used in its construction. In building both we will heavily rely on the Java API to provide services. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

2.1 Course Site Generator and Desktop Java Overview

The Course Site Generator and DesktopJava framework will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in home packages.

Figure 2.1: Course Site Generator and Desktop Java Overview

2.2 Java API Usage

Both the framework and the mini-game application will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.

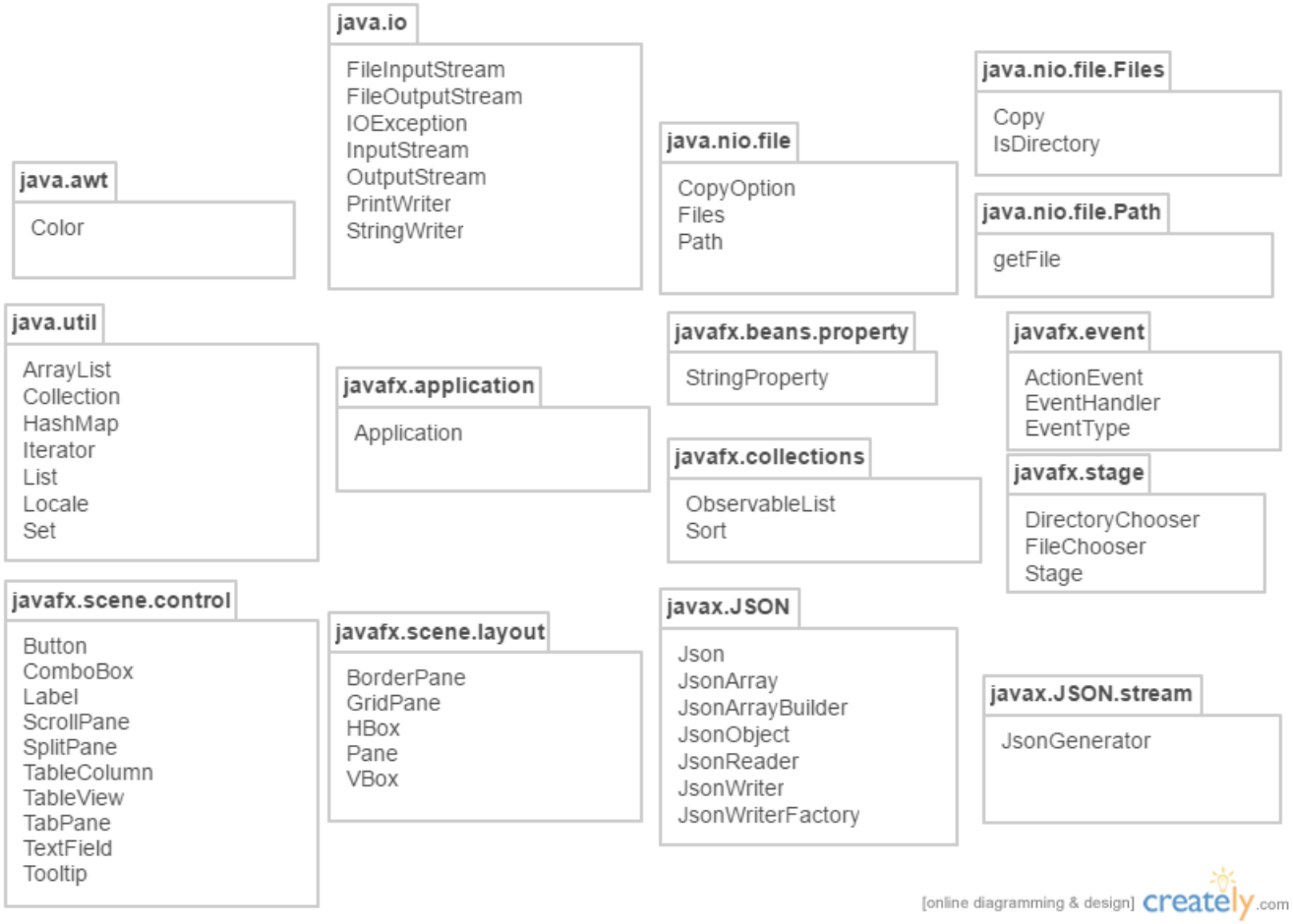


Figure 2.2: Java API Usage

2.3 Java API Usage Descriptions

Tables 2.3.1-2.3.16 below summarize how each of these classes will be used.

Class/Interface	Use
Color	Used to gain access to colors when using css in the class.

Table 2.3.1: Uses for Classes in the Java API's java.awt package

Class/Interface	Use
FileInputStream	Used to obtain input bytes from a file when loading a json.
FileOutputStream	Used as an output stream to write into JSON files.
IOException	Used if the user tries to export the files to a wrong place or for other I/O exceptions.
InputStream	Holds the input bytes so that they can be read by a JSONReader.
OutputStream	Holds the output bytes so that they can be put into a JSON file.
PrintWriter	Used to actually perform the write to a file.
StringWriter	Used to collect the output from a buffer and construct a string.

Table 2.3.2: Uses for Classes in the Java API's java.io package

Class/Interface	Use
CopyOption	Used to configure how to copy a file.
Files	Used to gain access to the methods within the java.nio.files.Files class and use the File datatype.
Path	Used to get the getFile method in java.nio.files.Path and to use the Path datatype.

Table 2.3.3: Uses for Classes in the Java API's java.nio.file package

Class/Interface	Use
getFile	Gets a File representation of the path and will be used when exporting the file.

Table 2.3.4: Uses for Classes in the Java API's java.nio.path package

Class/Interface	Use
Copy	Used in export to copy a file to another directory.
IsDirectory	Used in export to check if the location that the user picked is an actual directory.

Table 2.3.5: Uses for Classes in the Java API's java.nio.file package

Class/Interface	Use
ArrayList	Used to emulate a stack for the undo and redo in the application.
Collection	Used to observable list type and sort method in the collection class.
Hashmap	For holding all of the TA's, Recitations, etc. and to hold the data in the Office hours grid.
Iterator	Used to iterate through the Hashmap, such as when deleting the name from the TA list, it should also delete from the grid.
List	Used for holding information that can be manipulated, such as combo boxes when a user puts a start time and the end time has to begin after that time.
Locale	Lets the JVM know that we are in the US and should therefore respect its language and geographic region.
Set	Used when traversing through a hashmap using an iterator such as when deleting a TA name from the office hours grid.

Table 2.3.6: Uses for Classes in the Java API's java.util package

Class/Interface	Use
Application	Used to create and start the GUI.

Table 2.3.7: Uses for Classes in the Java API's javafx.Application package

Class/Interface	Use
StringProperty	A wrapper class to strings and will be used in the hashmap to display names onto the office hour grid.

Table 2.3.8: Uses for Classes in the Java API's javafx.beans.property package

Class/Interface	Use
ObservableList	Used to hold all of the teaching assistants and is used over lists so that listeners can track changes to the list.
Sort	Used to sort the list and is used when the user enters a new TA.

Table 2.3.9: Uses for Classes in the Java API's javafx.collections package

Class/Interface	Use
ActionEvent	For getting information about an action event like which button was pressed.
EventType	Used for giving what type of event happen, such as a keyEvent or mouseEvent.
EventHandler	Used for responding to an event, like a button press. We will provide our own custom implementation of this interface.

Table 2.3.10: Uses for Classes in the Java API's javafx.event package

Class/Interface	Use
DirectoryChooser	Used to let the user choose the directory location when exporting.
FileChooser	Used to let the user choose the JSON file that will be used.
Stage	Used to create the primary stage of the GUI.

Table 2.3.11: Uses for Classes in the Java API's javafx.stage package

Class/Interface	Use
Button	Used for creating any buttons, such as an add button.
ComboBox	Used for creating a drop down box, such as choosing a team .
Label	For writing the time and date labels in the first row and the first column.
ScrollPane	For the office hours grid, so that the user can scroll through the times.
SplitPane	Used to create the split between the TA assistants pane and the office hours grid pane.
TableColumn	Used for showing the recitations made, schedules made, etc.
TableView	Used for creating an TA table with the data in the table stored as an observable list.
TabPane	Used for creating the tabs in which the user can choose what data to put into the website.
TextField	For getting information on recitations, teams, etc.

Table 2.3.12: Uses for Classes in the Java API's javafx.scene.control package

Class/Interface	Use
KeyCode	Used to get the key the user clicks such as when the user deletes a name from the TA table.
KeyCodeCombination	Used to get the combination of keys the user clicks such as when undoing an action.
KeyCombination	Used to make sure a user clicks on a combination of keys, such as undoing an action.
KeyEvent	Used to see if the user pressed a key, such as when deleting a TA name.
MouseEvent	Used to see if the user pressed a mouse button, such as when clicking on a TA name.

Table 2.3.13: Uses for Classes in the Java API's javafx.scene.input package

Class/Interface	Use
BorderPane	Used to lay out panes in the top, right, center, left, and bottom of the pane. i.e parents of the add/edit panes.
GridPane	Used to create the office hours grid.
HBox	A pane used to lay out its pane children in a single horizontal row. i.e calendar boundaries in schedule tab.
Pane	Base class for Pane. i.e add/edit information in Recitation tab.
VBox	A pane used to lay out its pane children in a single horizontal column.

Table 2.3.14: Uses for Classes in the Java API's javafx.scene.layout package

Class/Interface	Use
Json	Class used for creating the JSON processing objects. i.e JsonArray, JsonObject, etc.
JsonArray	Used to hold the TAs list and TAs in each office hour time that are read in from the JSON file.
JsonArrayBuilder	Used to build a JSON array, and is needed turn the builder into a JSONArray.
JsonObject	Used to hold a Json file, and objects created from the file. i.e TA's info gets loaded into a JsonObject.
JsonReader	Used to read in a JSON file. i.e Loading a JSON file
JsonWriter	Used to writing in a JSON file to an output source. i.e Saving into a JSON file
JsonWriterFactory	Used to create JSON writer instances.

Table 2.3.15: Uses for Classes in the Java API's javax.JSON package

Class/Interface	Use
JsonGenerator	Writes JSON data to an output source in a streaming way and is used with JSONWriter

Table 2.3.16: Uses for Classes in the Java API's java.JSON.stream package

3 Class-Level Design Viewpoint

As mentioned, this design will encompass both the Course Site Generator application, Properties Manager, and the Desktop Java Framework. The following UML Class Diagrams reflect this. Note that due to the complexity of the project, we present the class designs using a series of diagrams going from overview diagrams down to detailed ones.

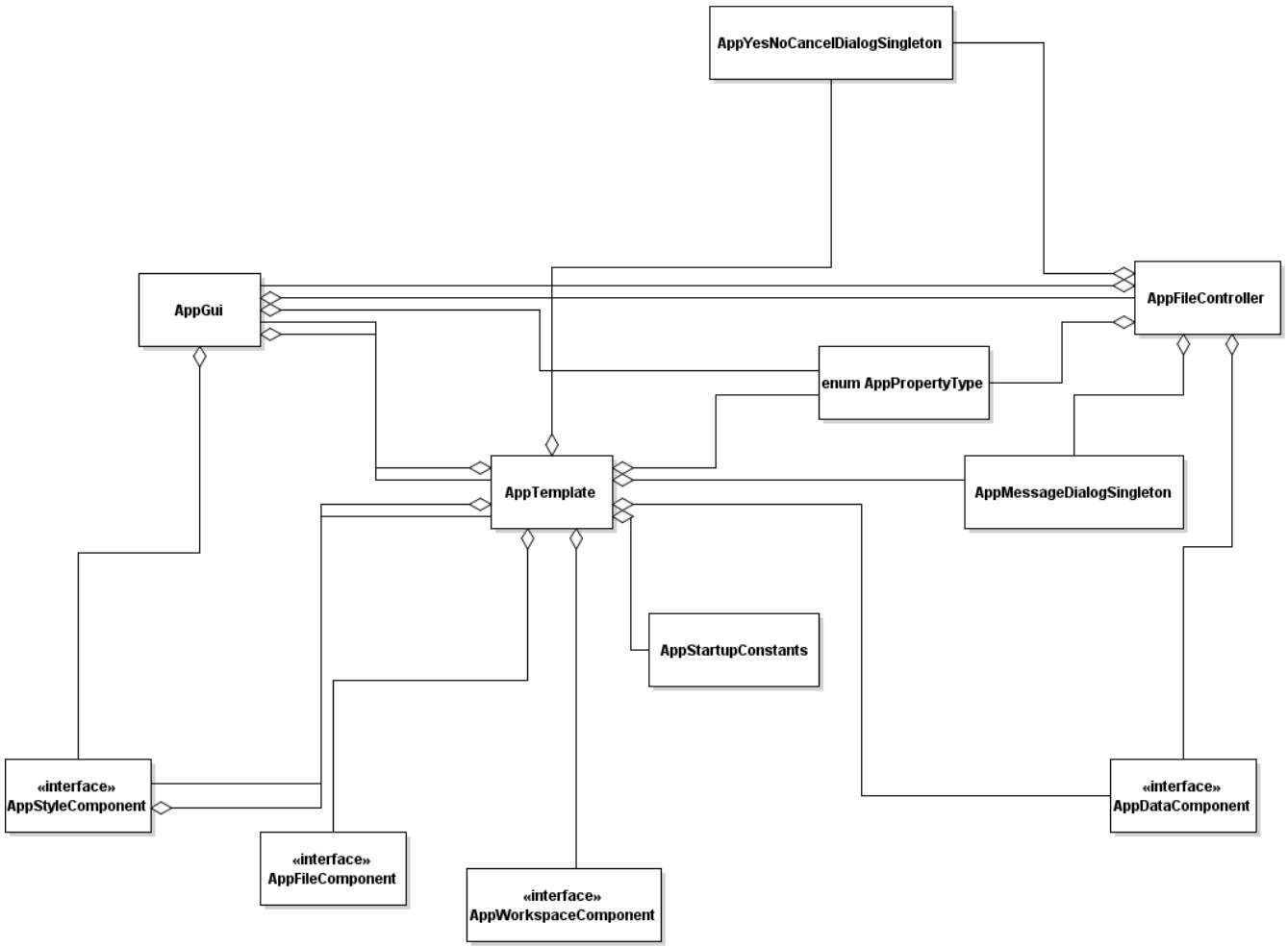


Table 3.1: Desktop Java Framework Overview UML Class Diagram

Table 3.2: Properties Manager Overview UML Class Diagram

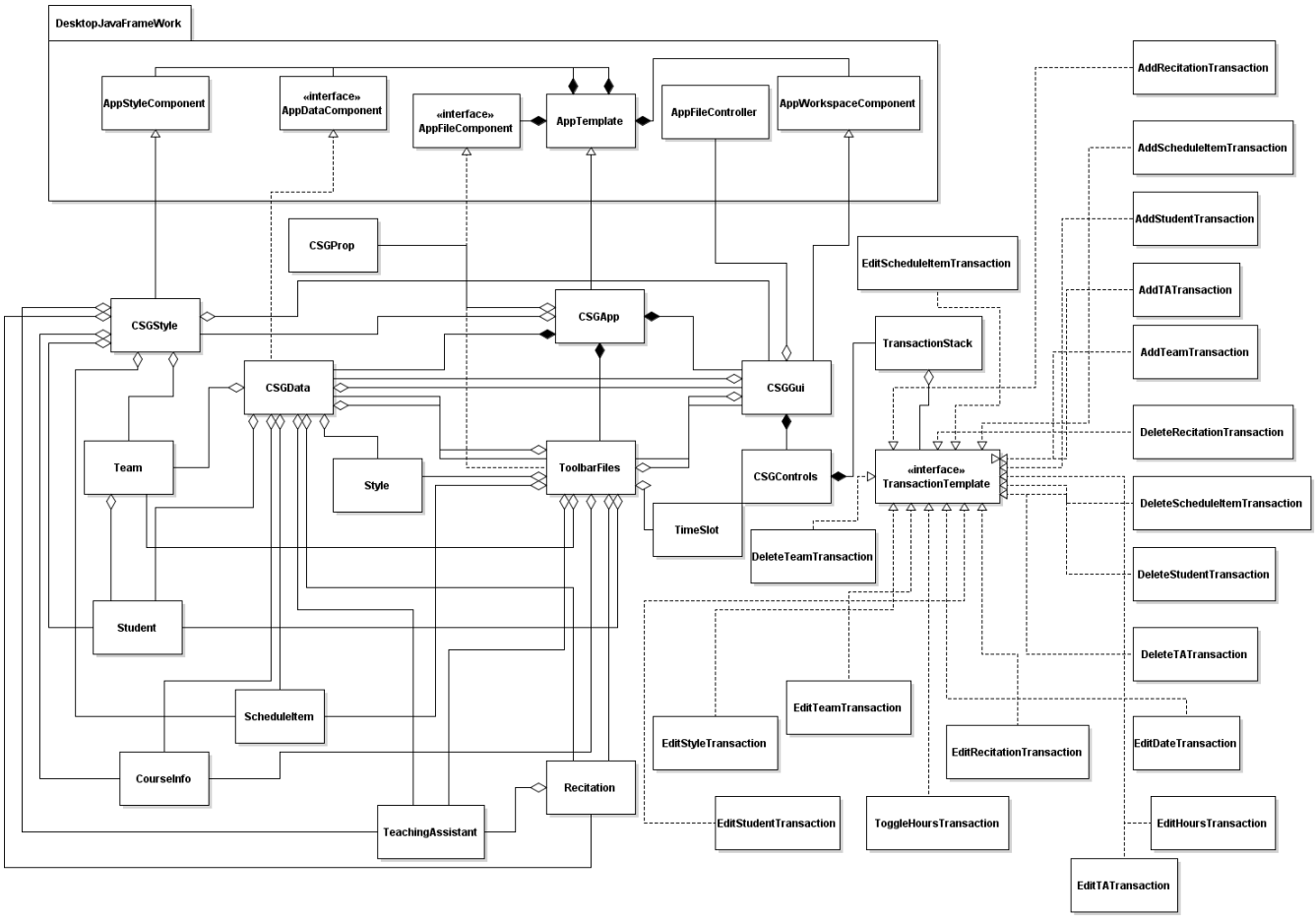


Table 3.3: Course Site Generator Overview UML Class Diagram

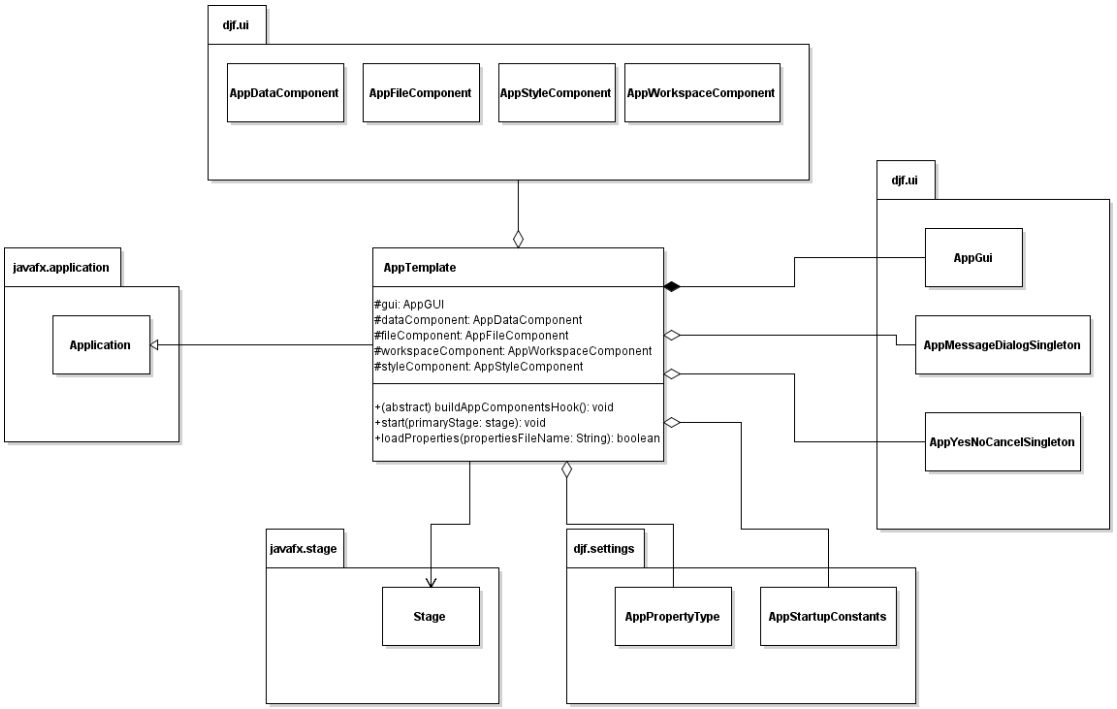


Table 3.4: Detailed AppTemplate UML Class Diagram

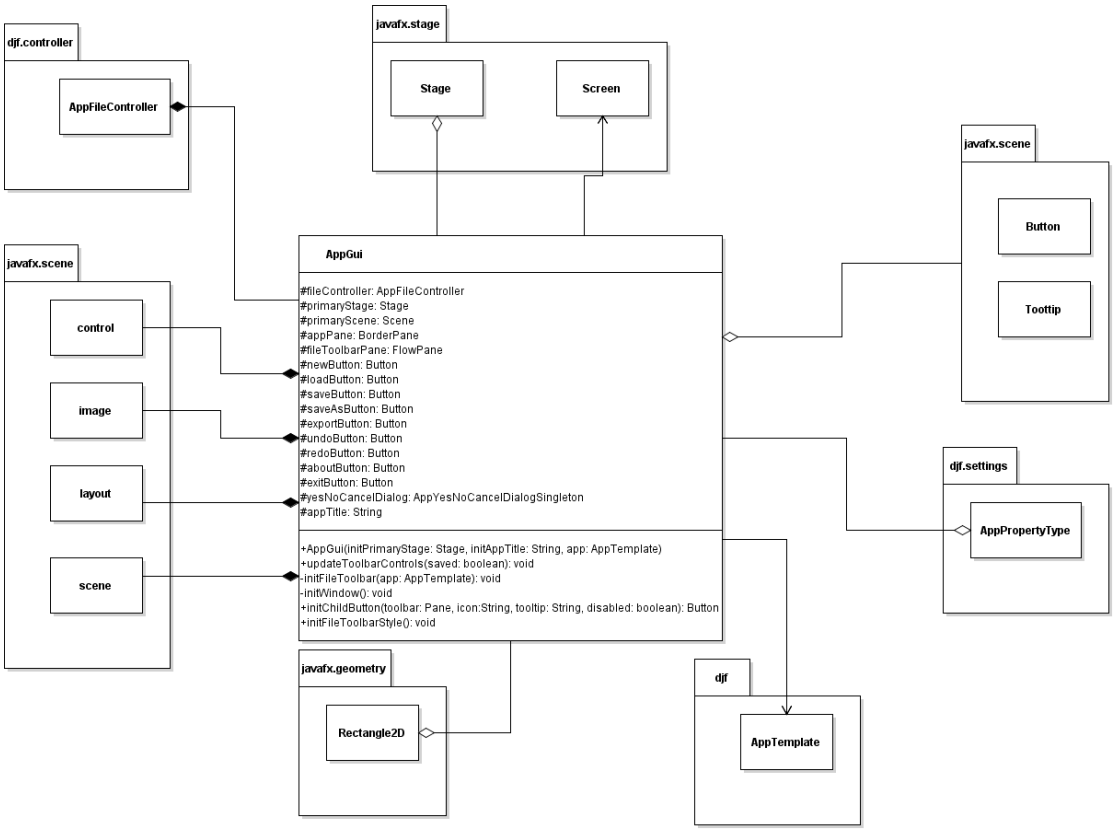


Table 3.5: Detailed AppGUI UML Class Diagram

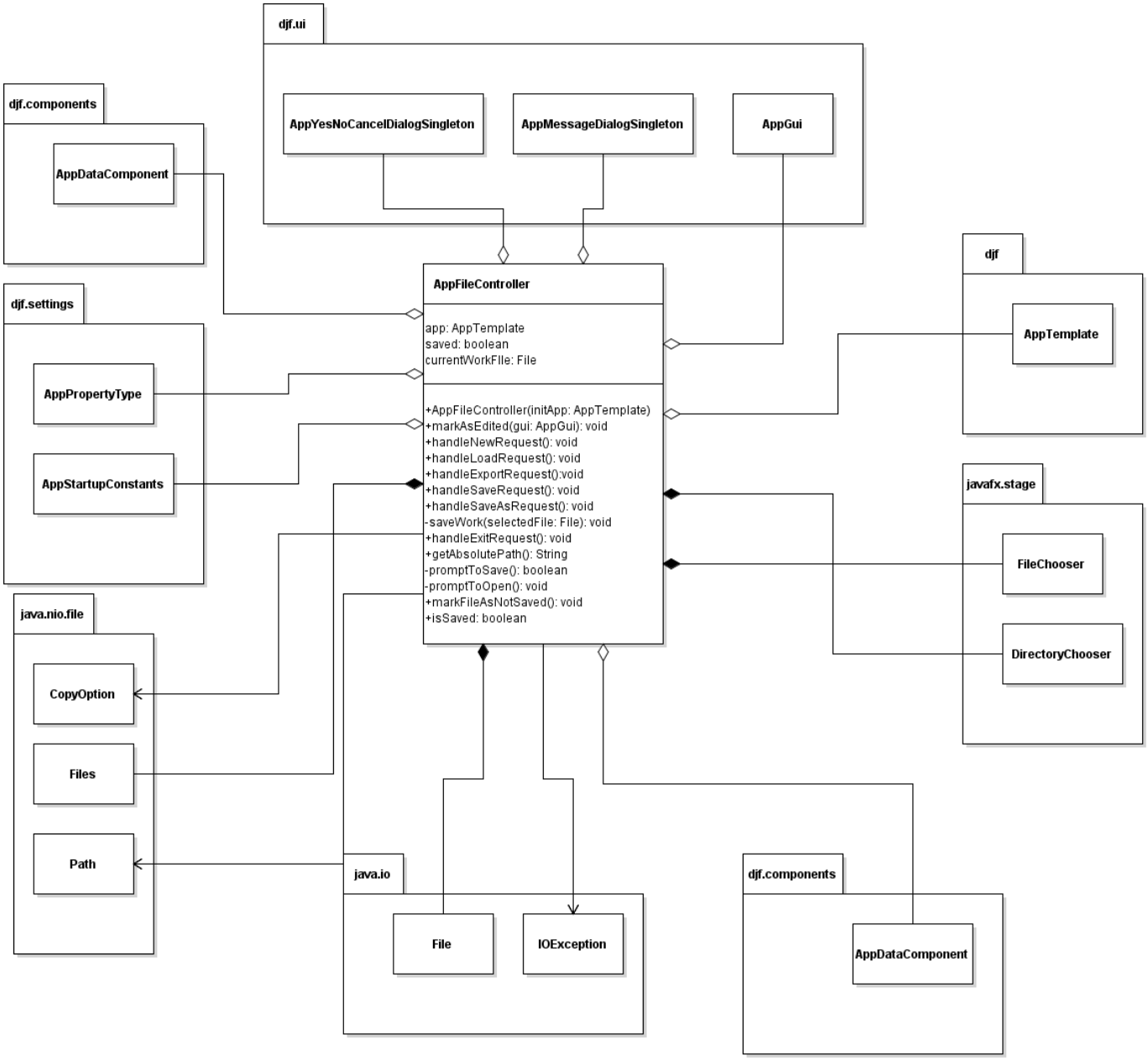


Table 3.6: Detailed AppFileController UML Class Diagram

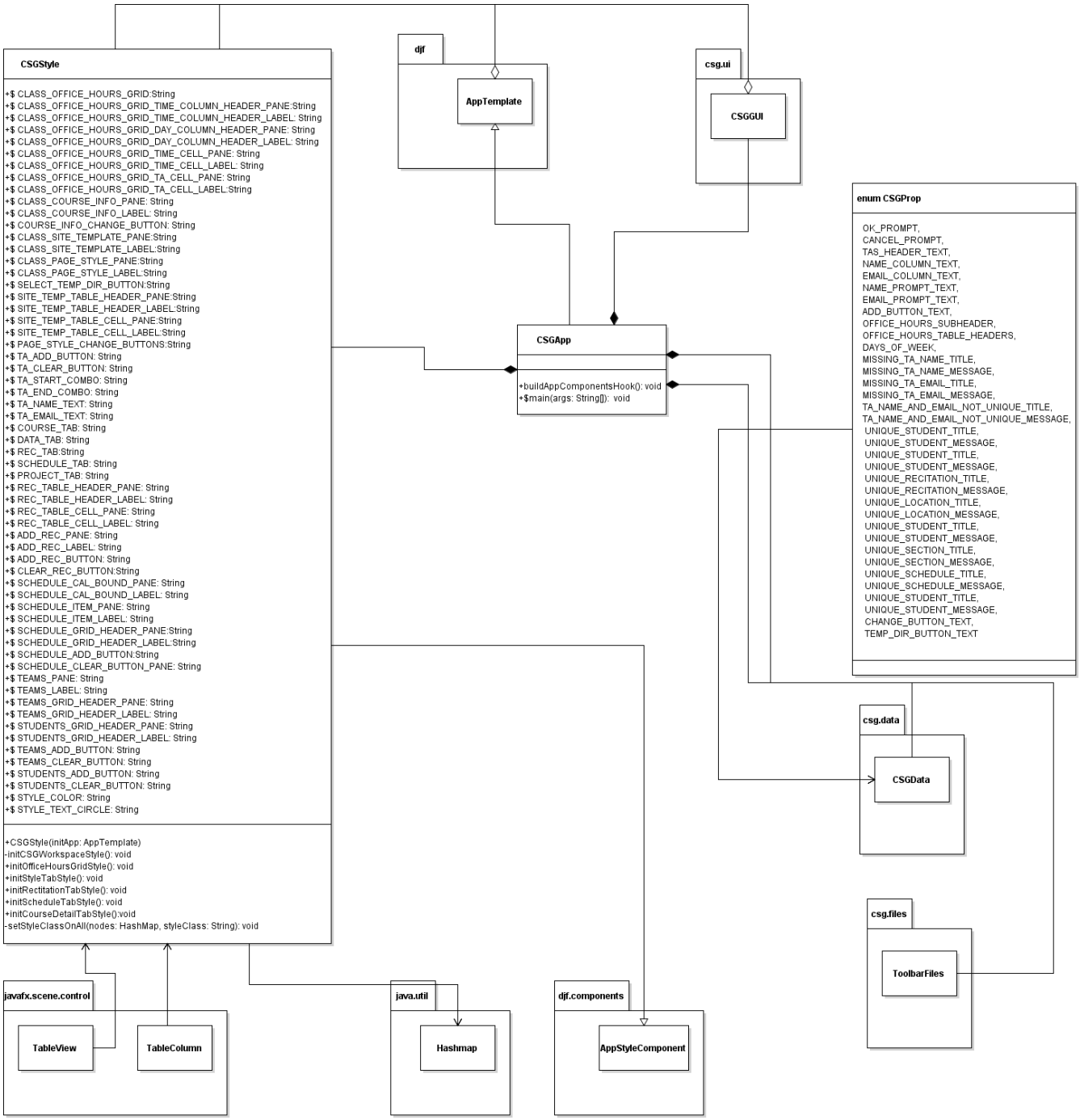


Table 3.7: Detailed CSGAPP, CSGStyle, and CSGProp UML Class Diagram

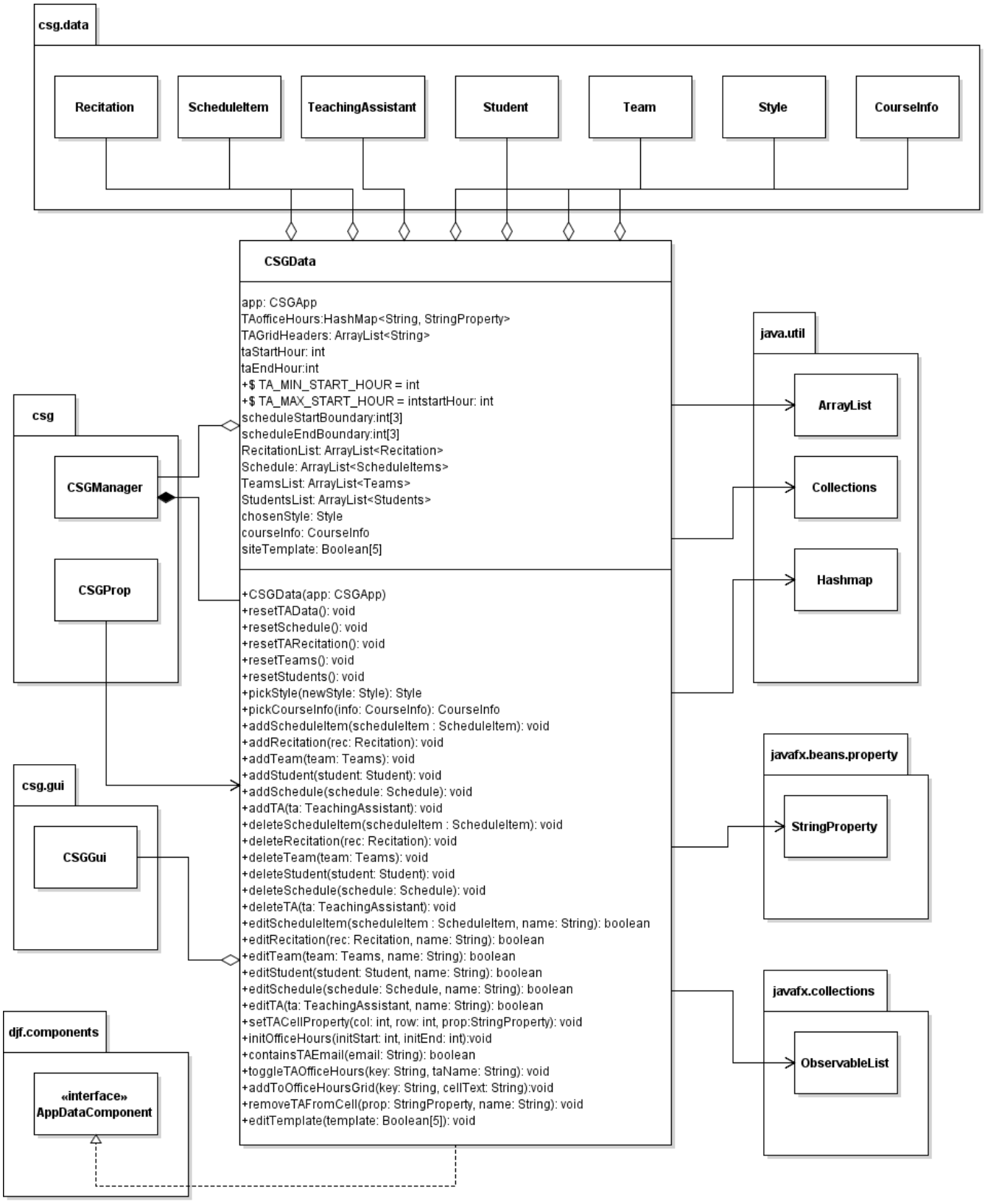


Table 3.8: Detailed CSGData UML Class Diagram

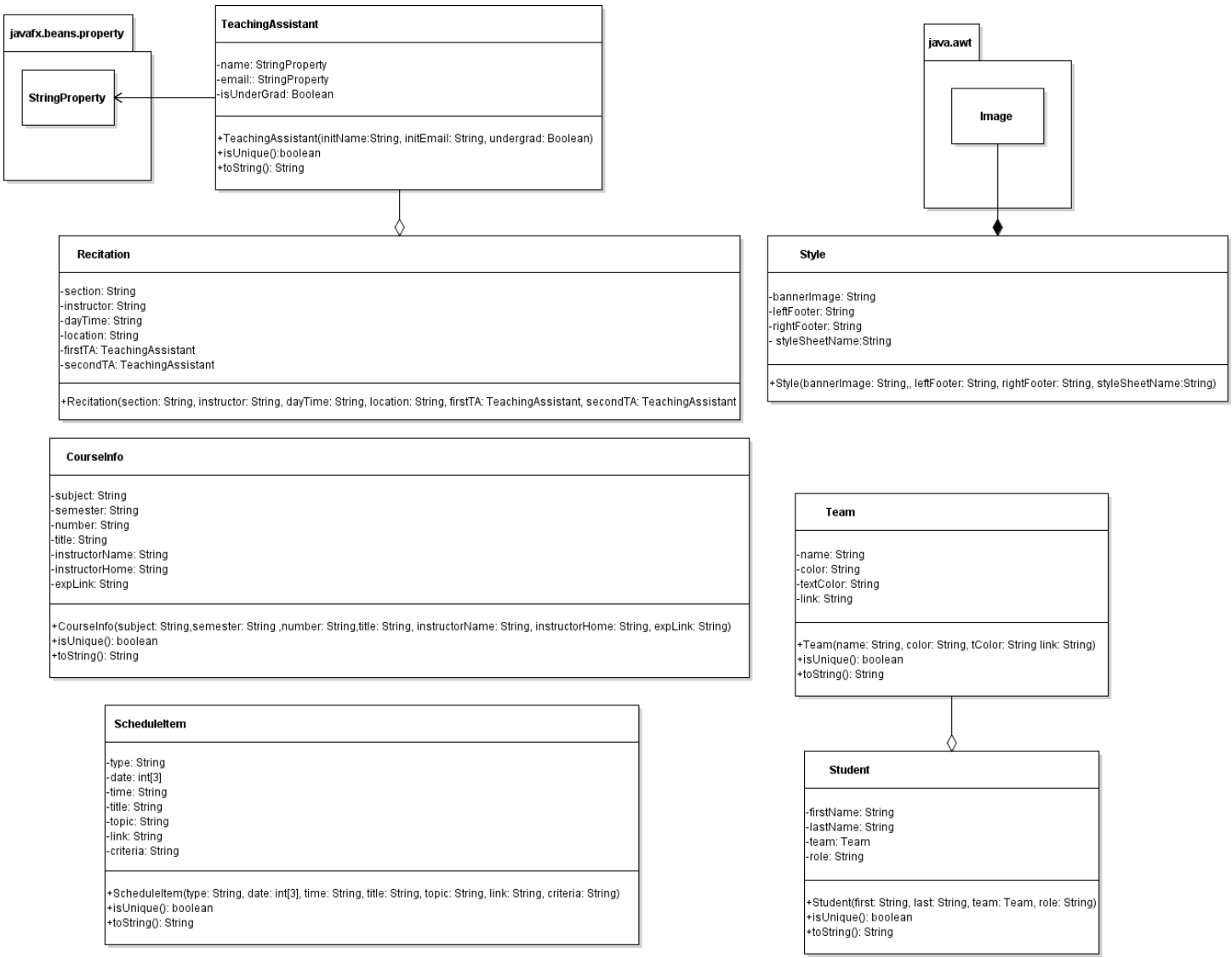


Table 3.9: Detailed TeachingAssistant, Style, Recitation, CourseInfo, ScheduleItem. Team, and Student UML Class Diagram

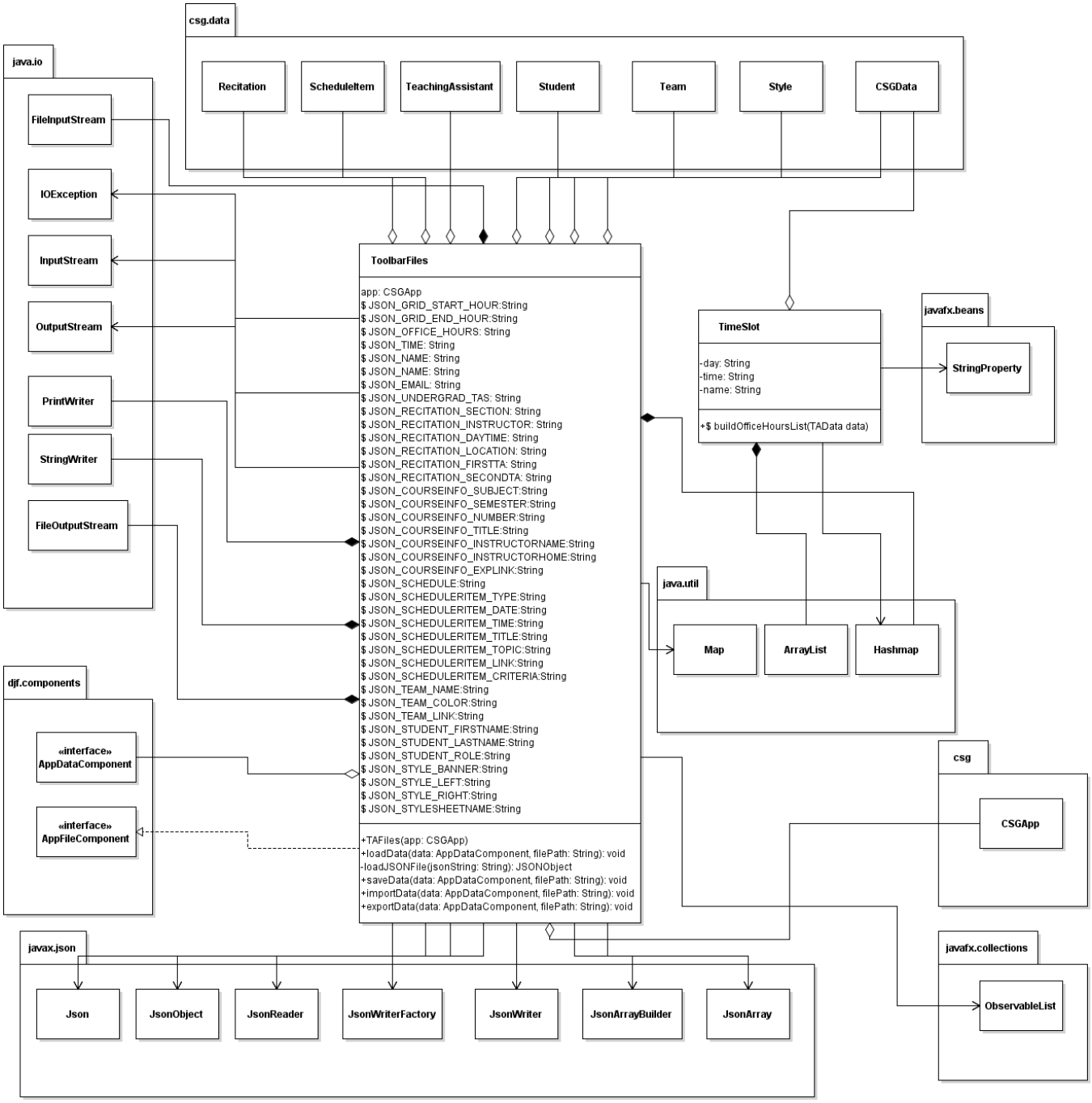


Table 3.10: Detailed ToolbarFiles and TimeSlot UML Class Diagram

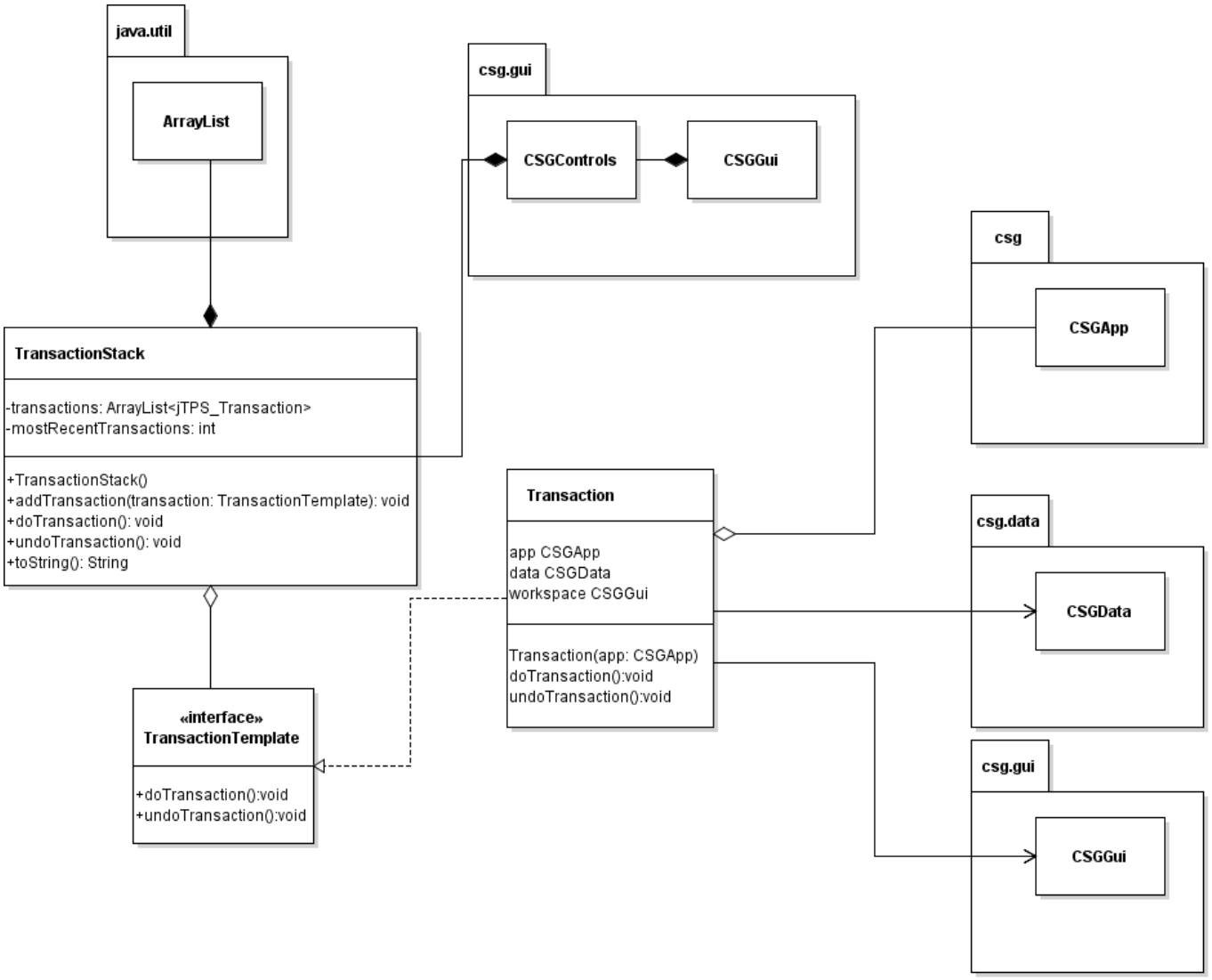


Table 3.11: Detailed TransactionStack, TransactionTemplate and Transaction UML Class Diagram

Note: Since there are 19 transactions implementing TransactionTemplate, they have been simplified to Transaction. All of the transaction will be aggregated with CSGApp and associated with CSGData and CSGGui. However, there will be a few more variables in each class depending on what the transaction is.

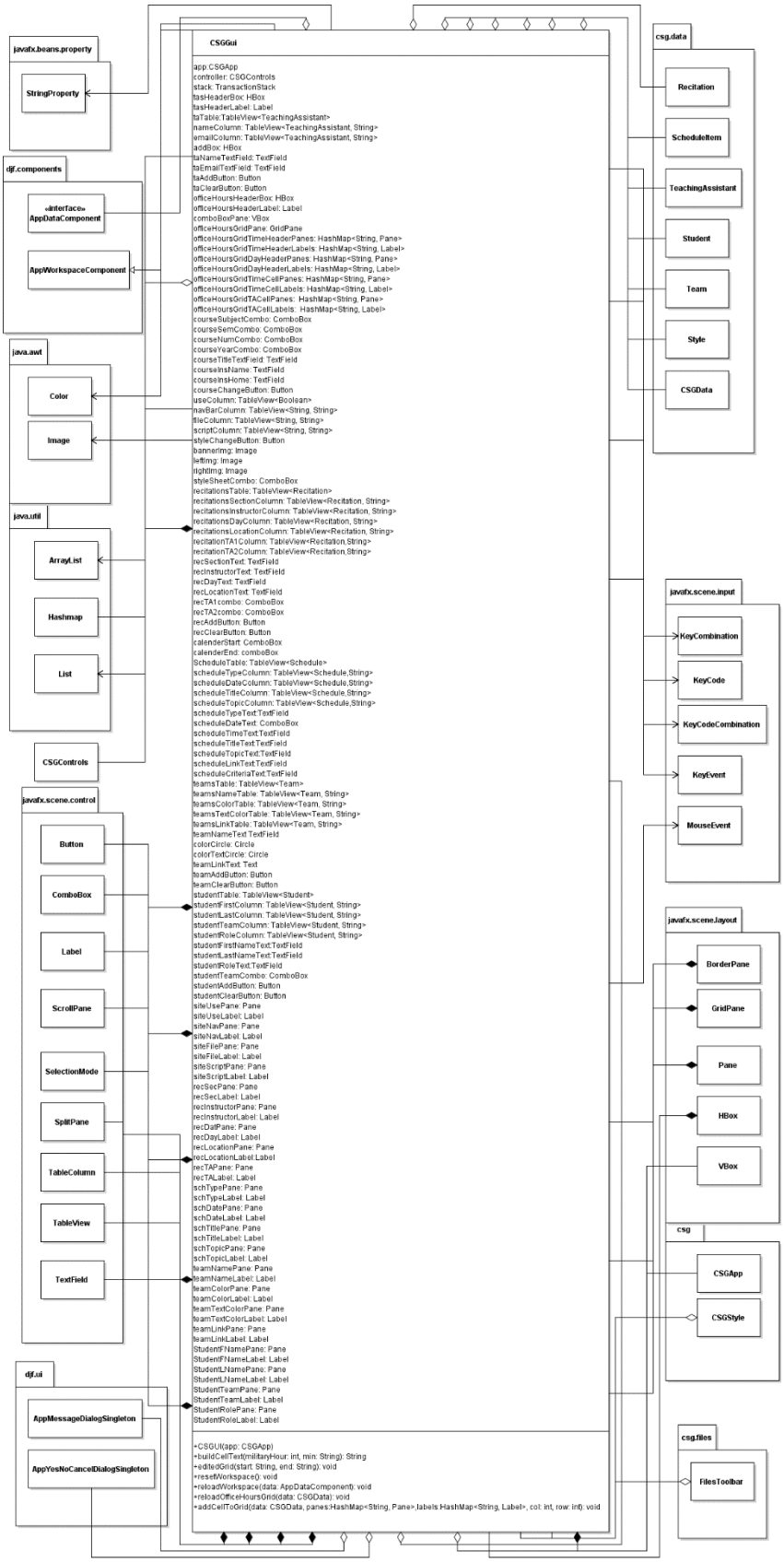


Table 3.12: Detailed CSGGui UML Class Diagram

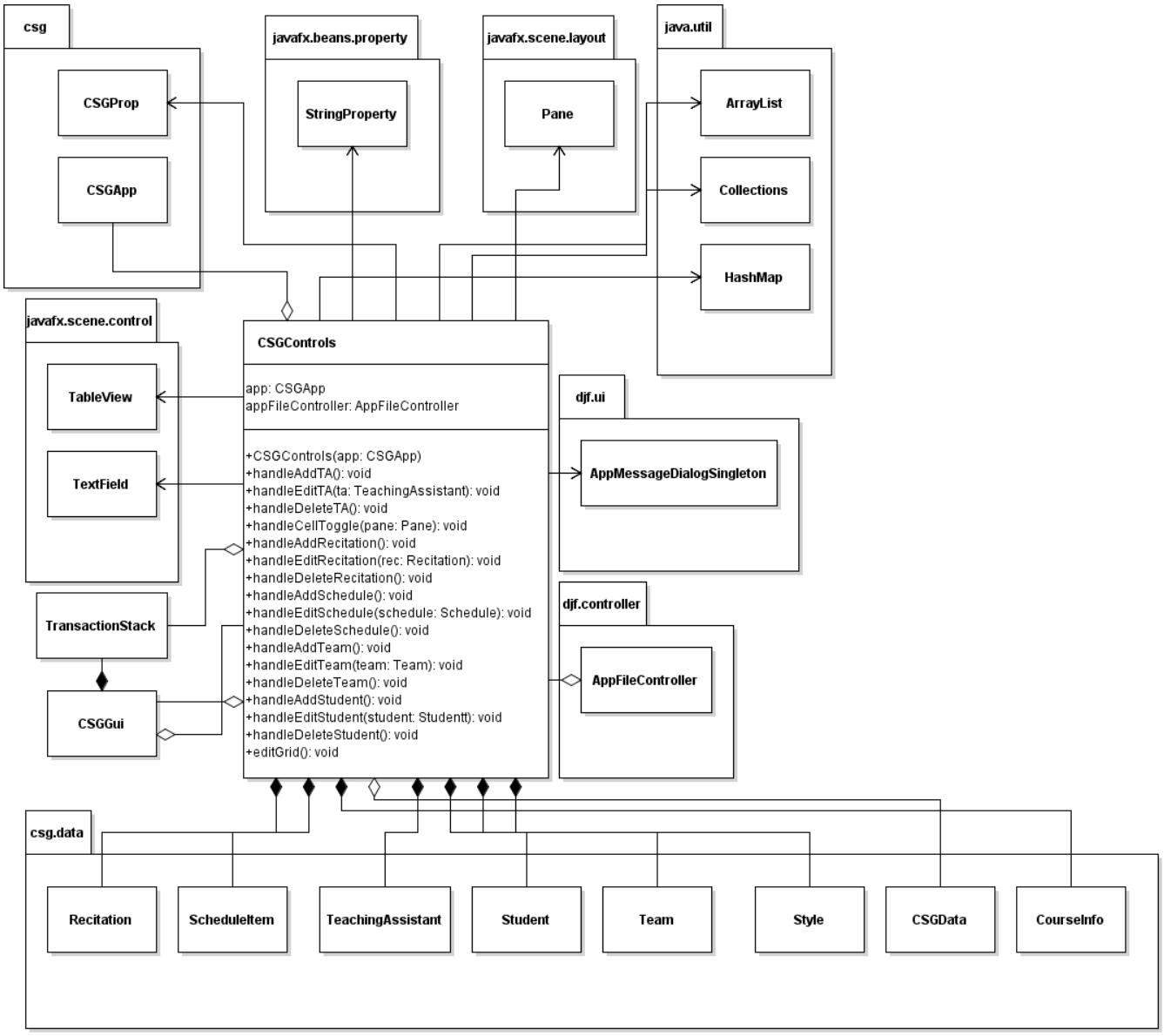


Table 3.13: Detailed CSGControls UML Class Diagram

4 Method-Level Design Viewpoint

Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.

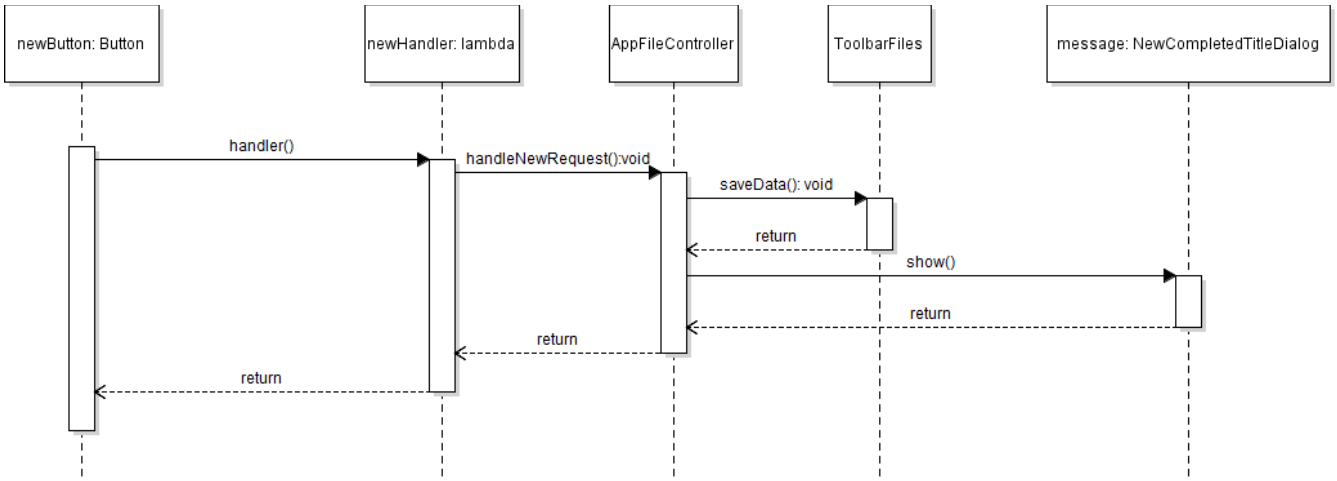


Figure 4.1: newButton UML Sequence Diagram

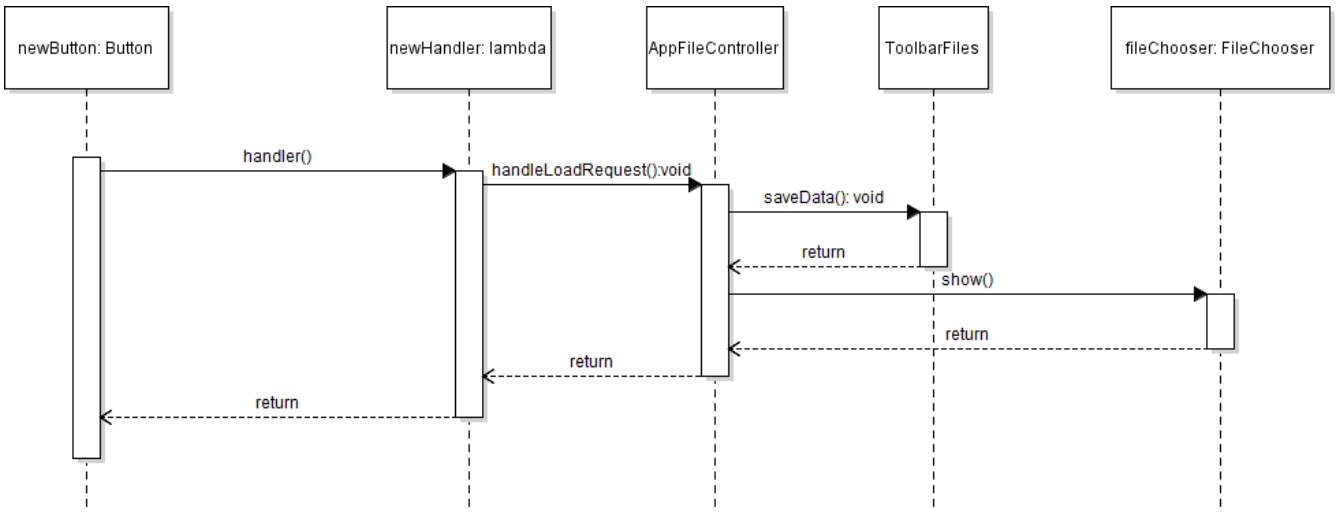


Figure 4.2: loadButton UML Sequence Diagram

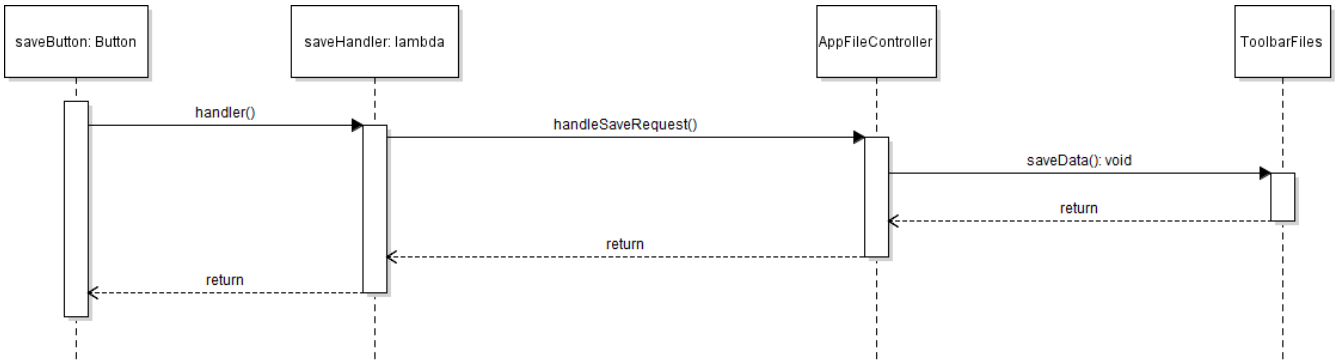


Figure 4.3: saveButton UML Sequence Diagram

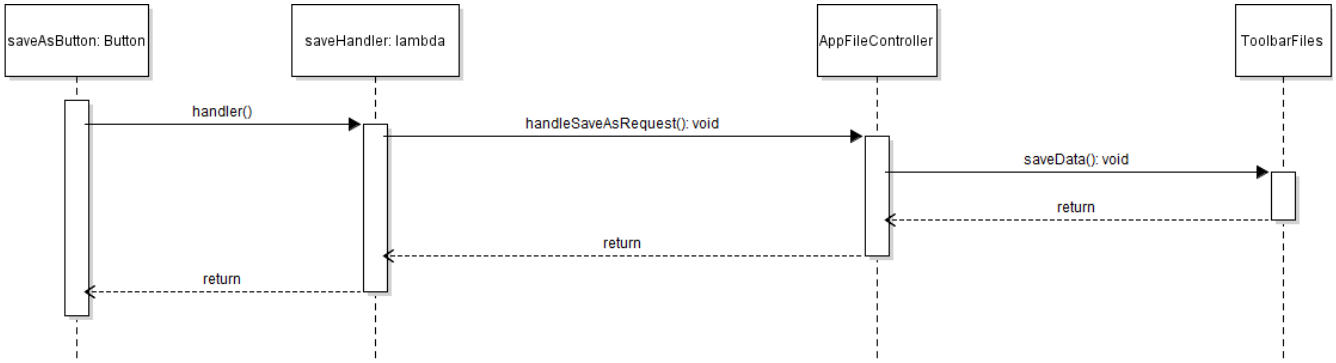


Figure 4.4: saveAsButton UML Sequence Diagram

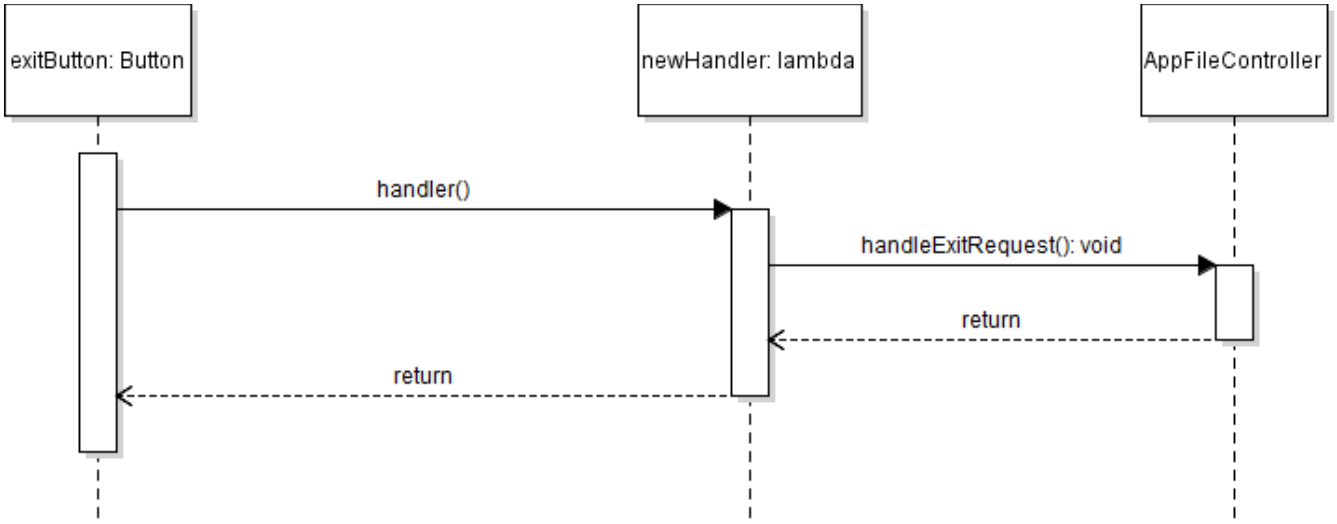


Figure 4.5: exitButton UML Sequence Diagram

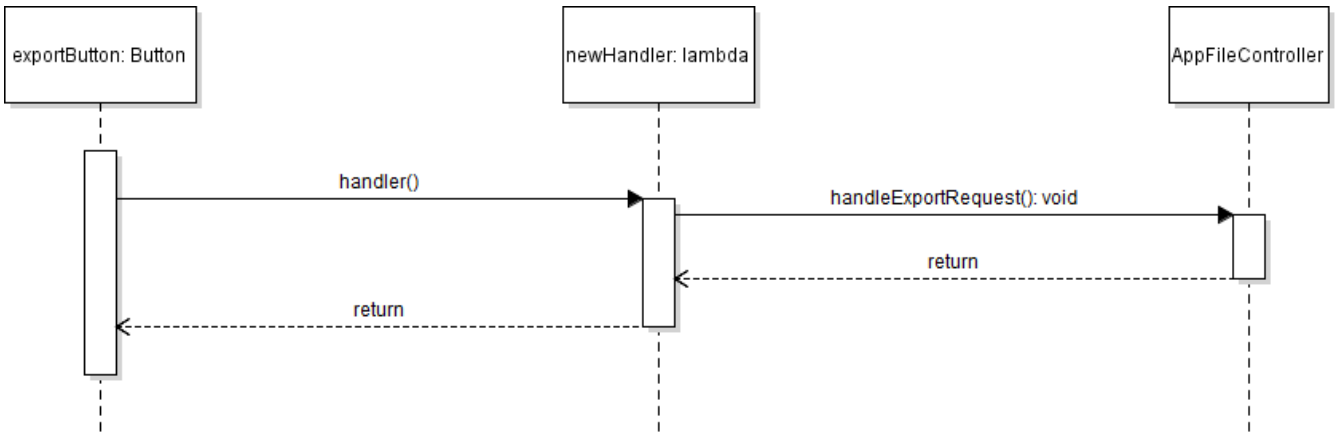


Figure 4.6: exportButton UML Sequence Diagram

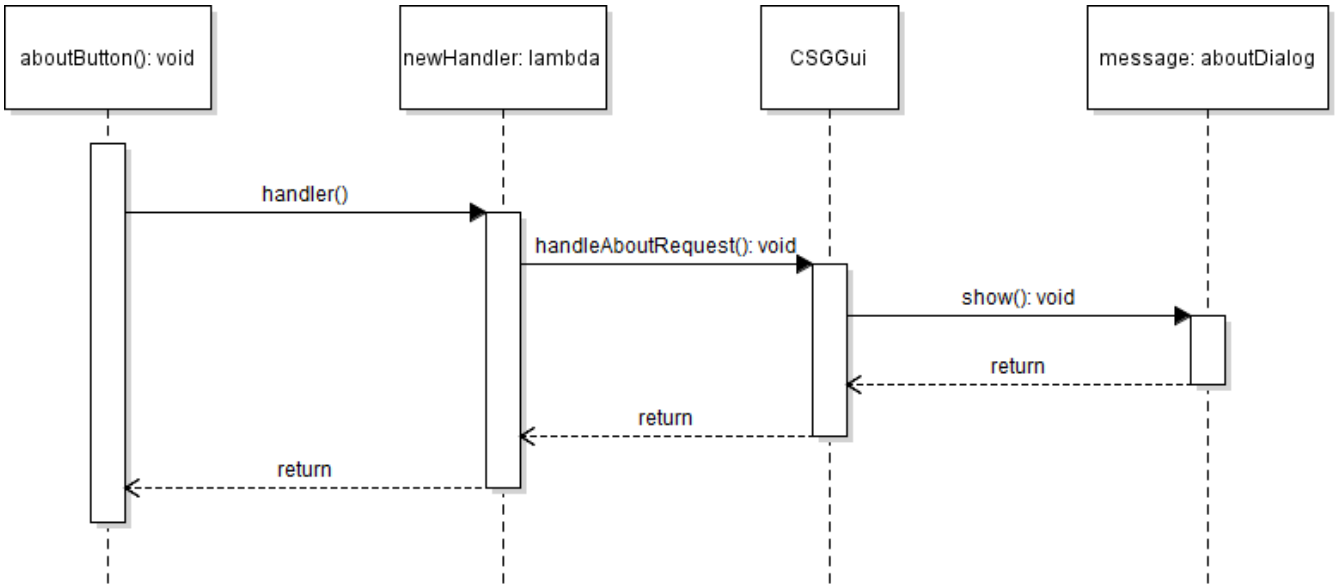


Figure 4.7: aboutButton UML Sequence Diagram

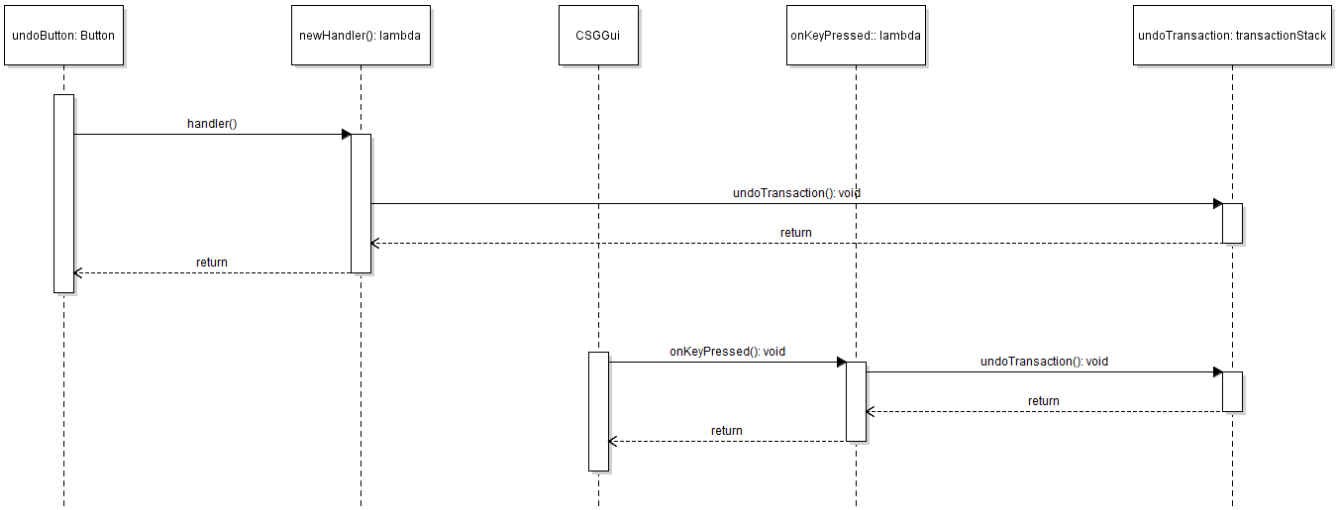


Figure 4.8: undoButton and Ctrl- Z UML Sequence Diagram

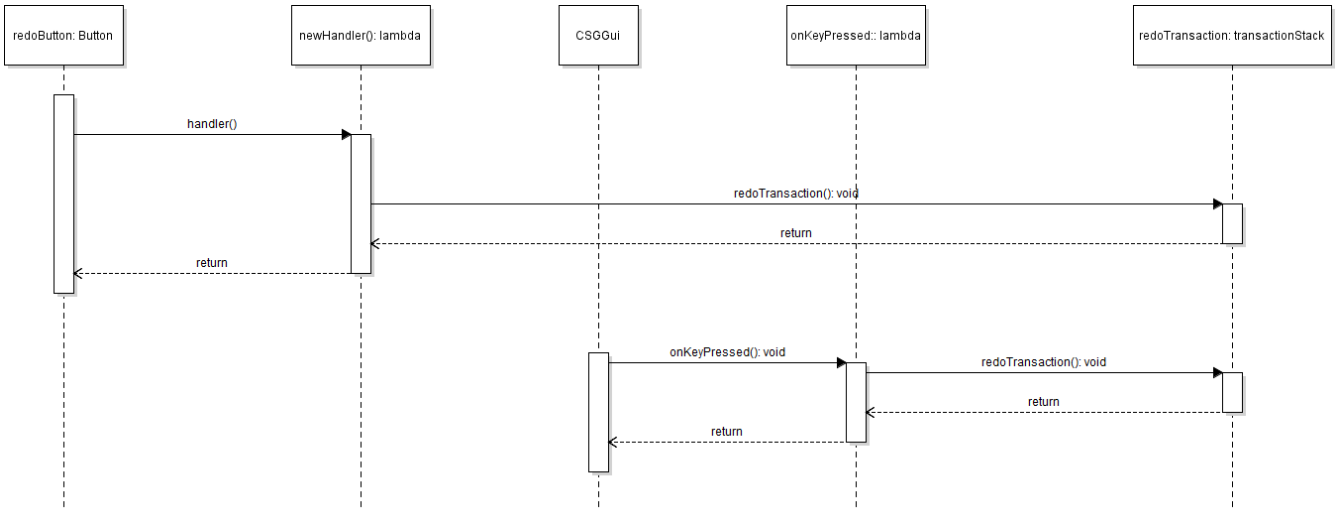


Figure 4.9: redoButton and Ctrl- Y UML Sequence Diagram

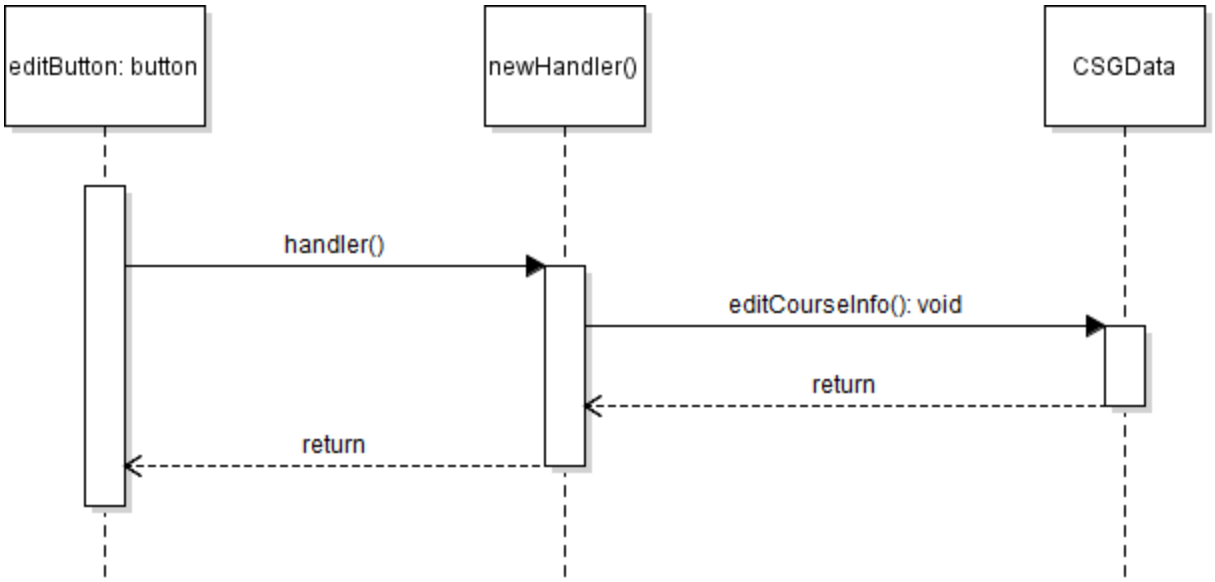


Figure 4.10: EditCourseInfoButton UML Sequence Diagram

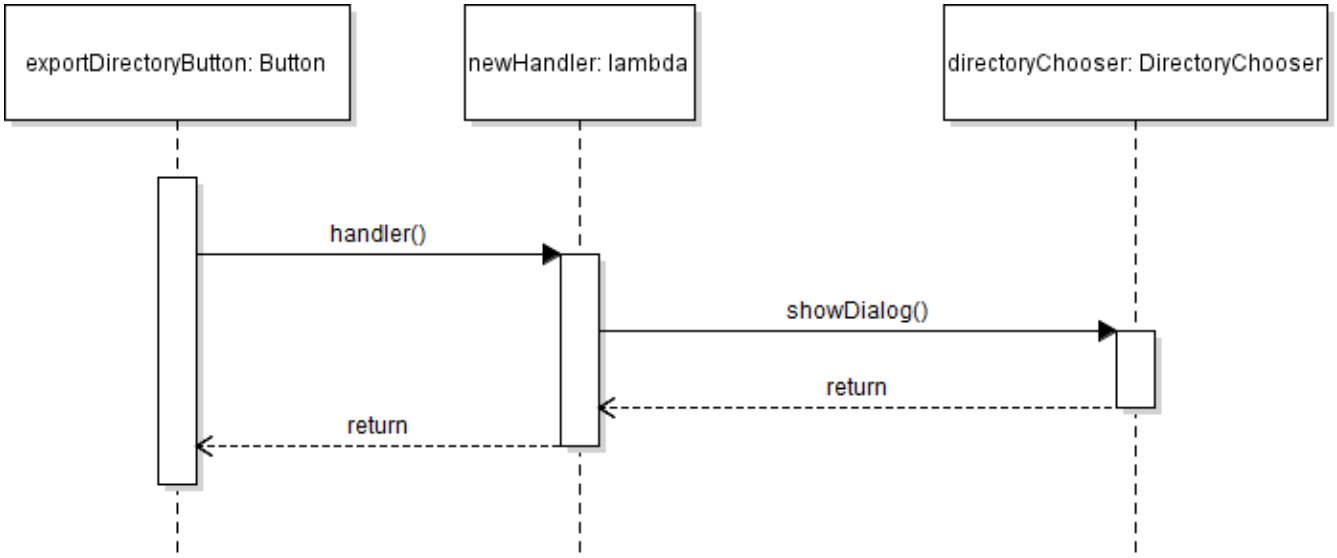


Figure 4.11: selectExportDirectoryButton UML Sequence Diagram

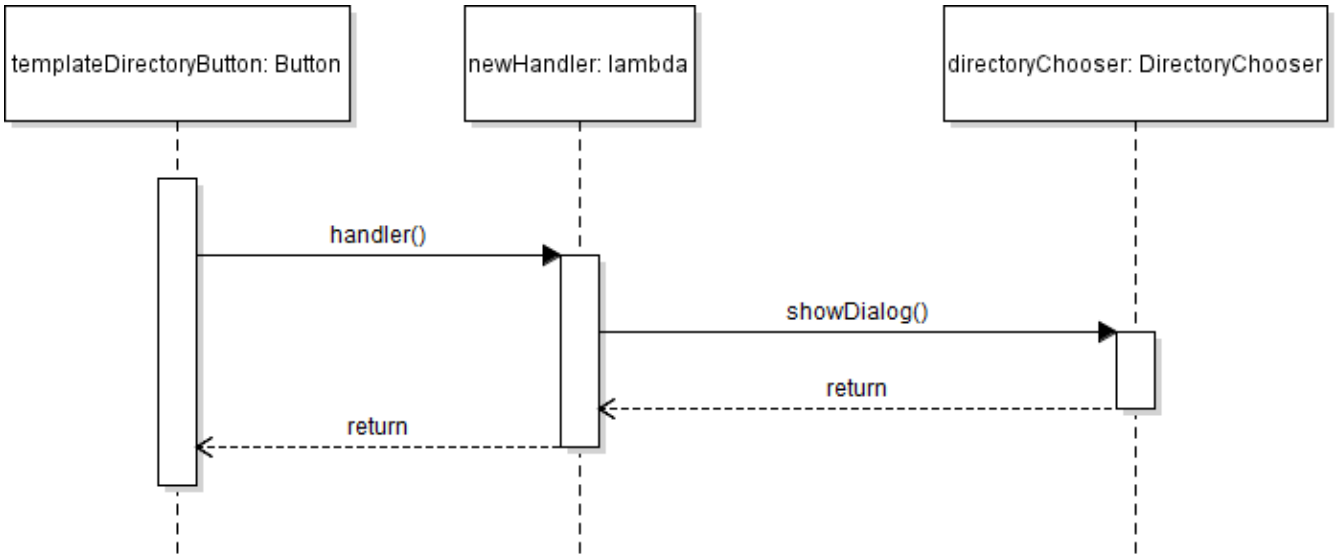


Figure 4.12: selectTemplateDirectoryButton UML Sequence Diagram

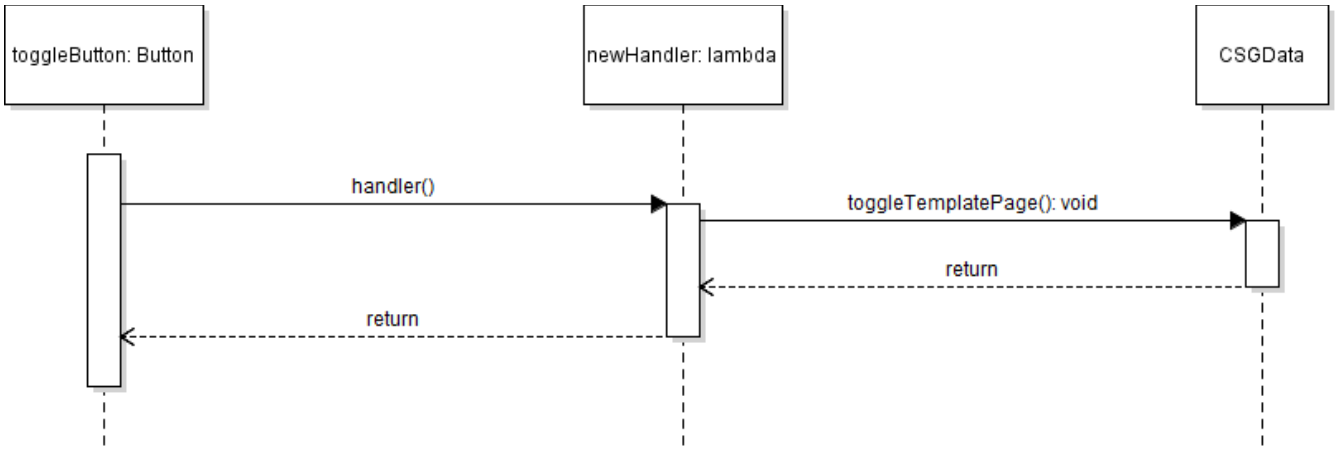


Figure 4.13: toggleUseTemplatePage UML Sequence Diagram

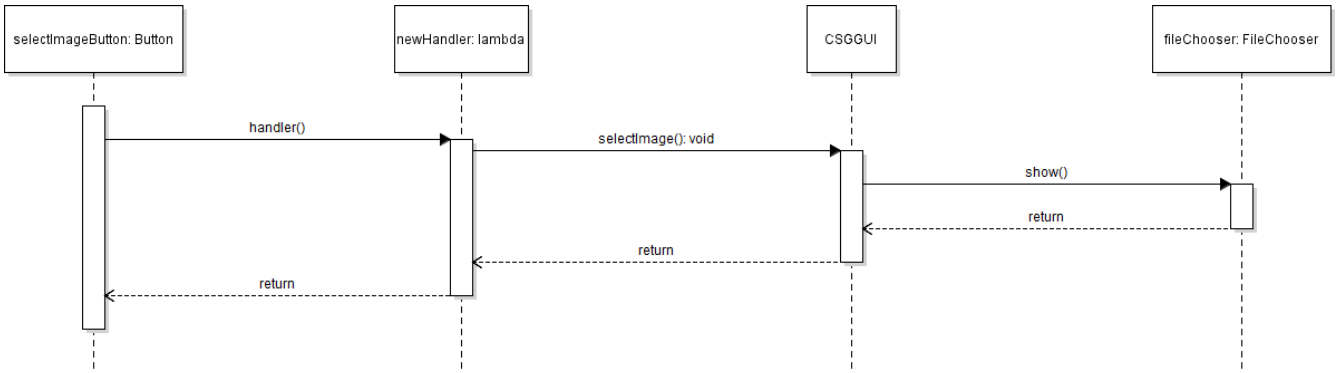


Figure 4.14: selectBrandingImageButton UML Sequence Diagram

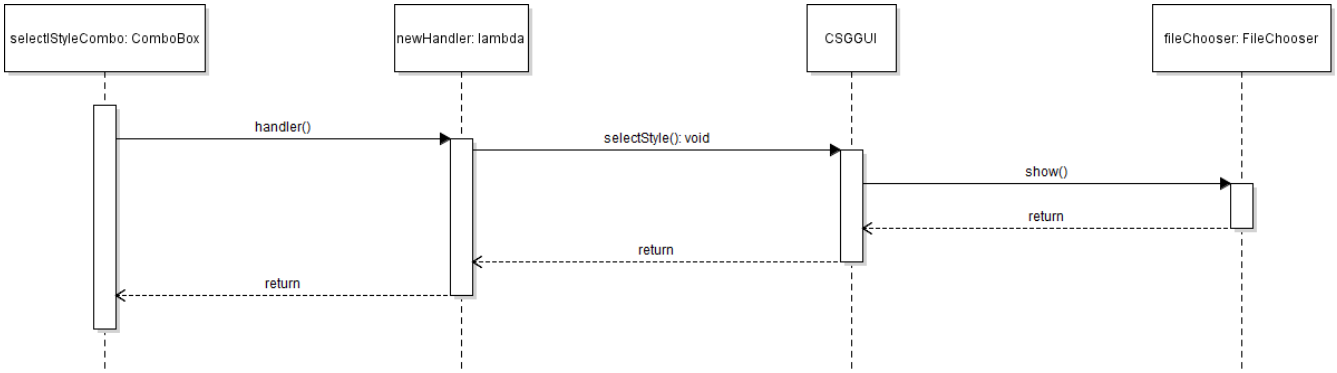


Figure 4.15: selectStyleSheetButton UML Sequence Diagram

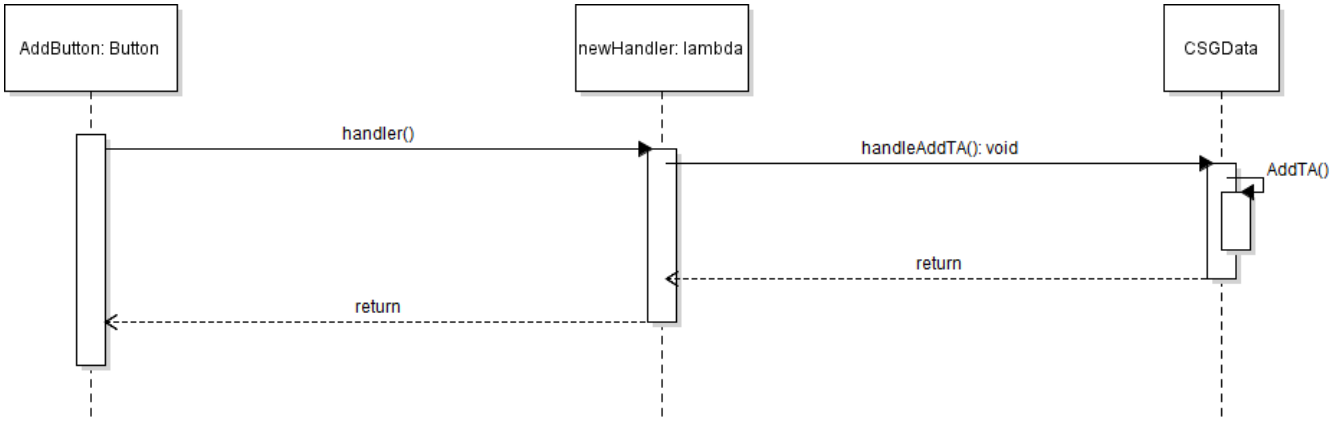


Figure 4.16: addTAButton UML Sequence Diagram

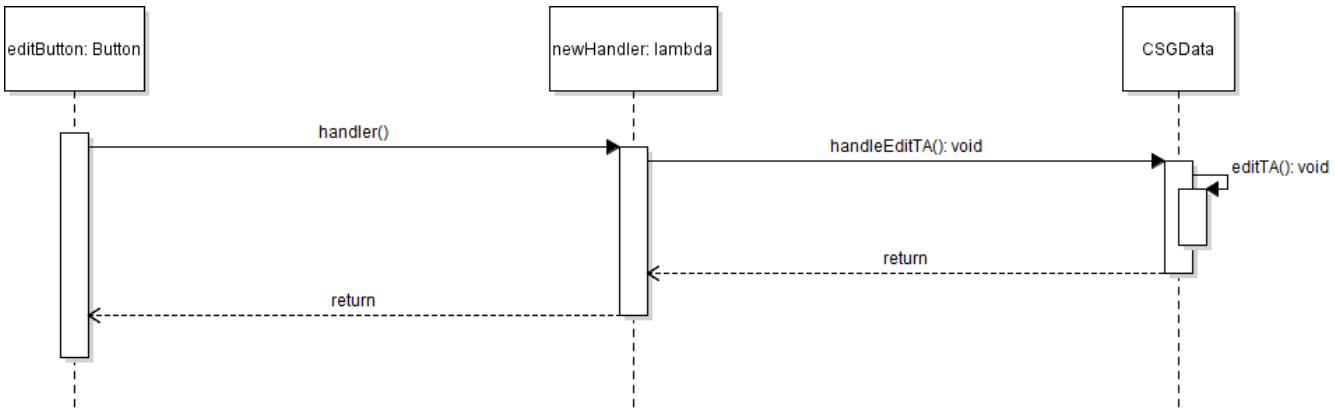


Figure 4.17: editTAButton UML Sequence Diagram

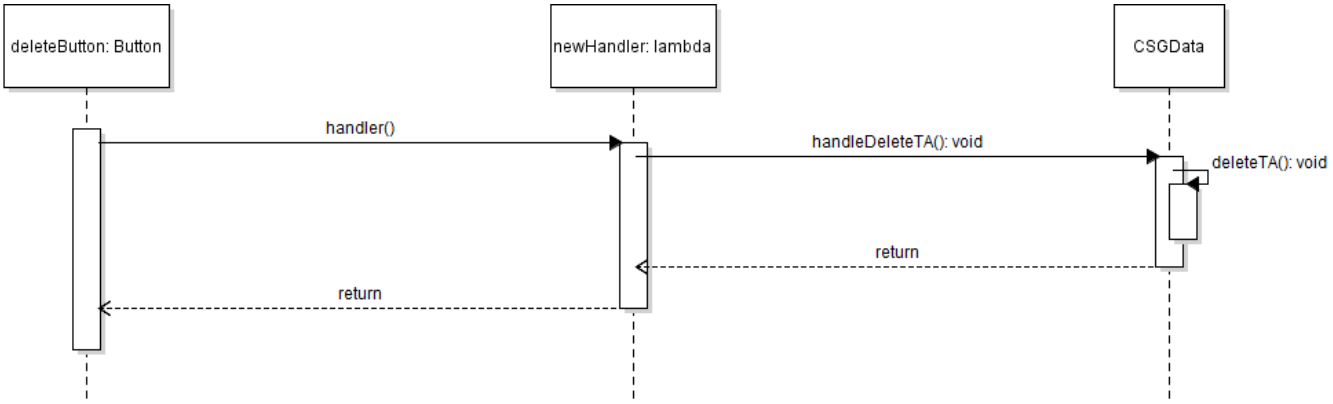


Figure 4.18: deleteTAButton UML Sequence Diagram

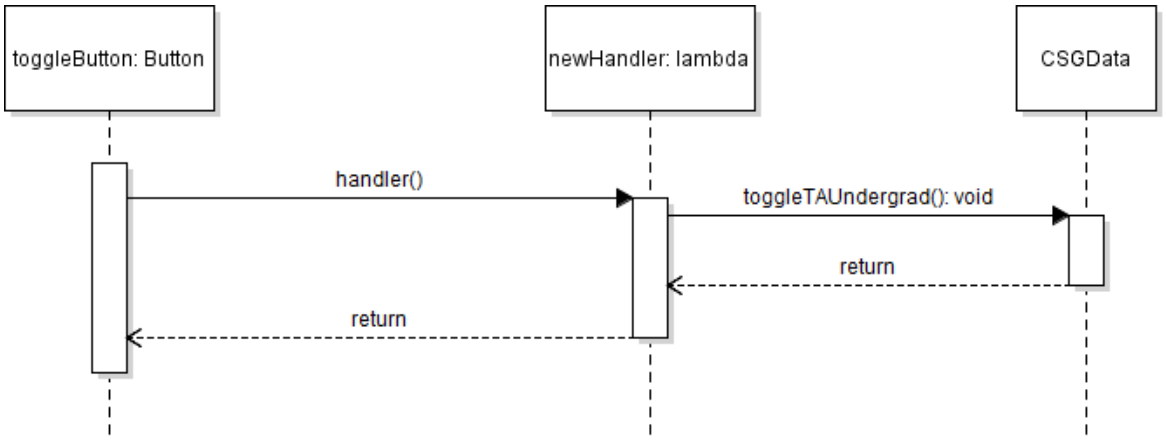


Figure 4.19: Toggle TA Undergrad UML Sequence Diagram

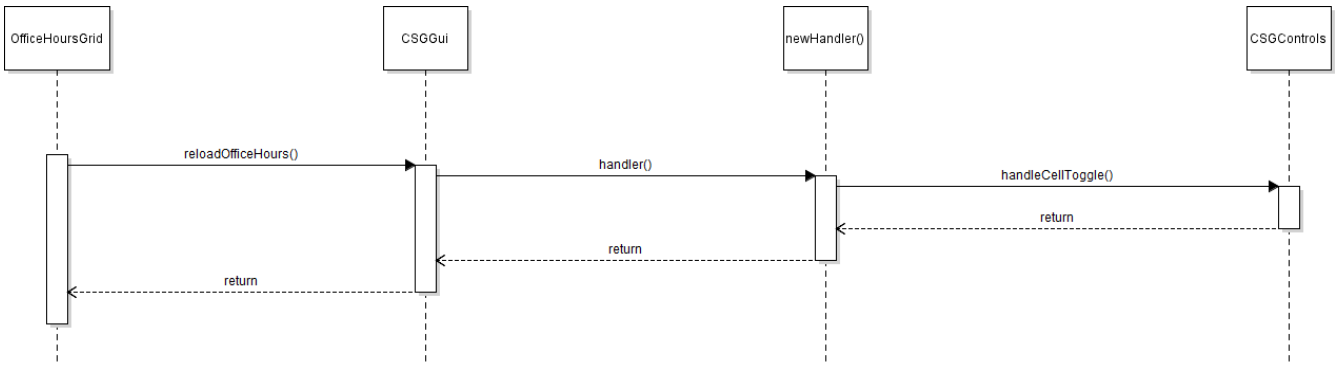


Figure 4.20: Toggle TA OfficeHours UML Sequence Diagram

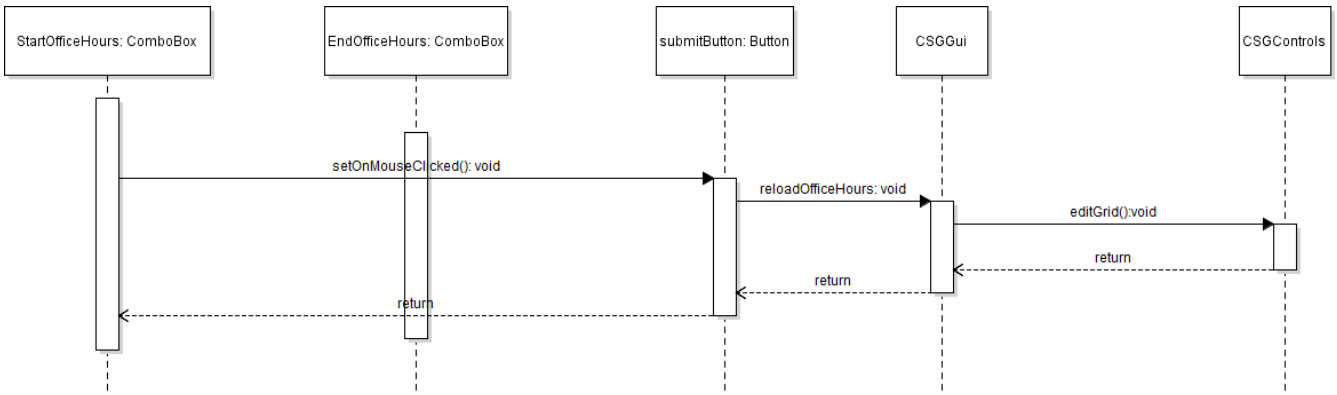


Figure 4.21: Change Office Hours UML Sequence Diagram

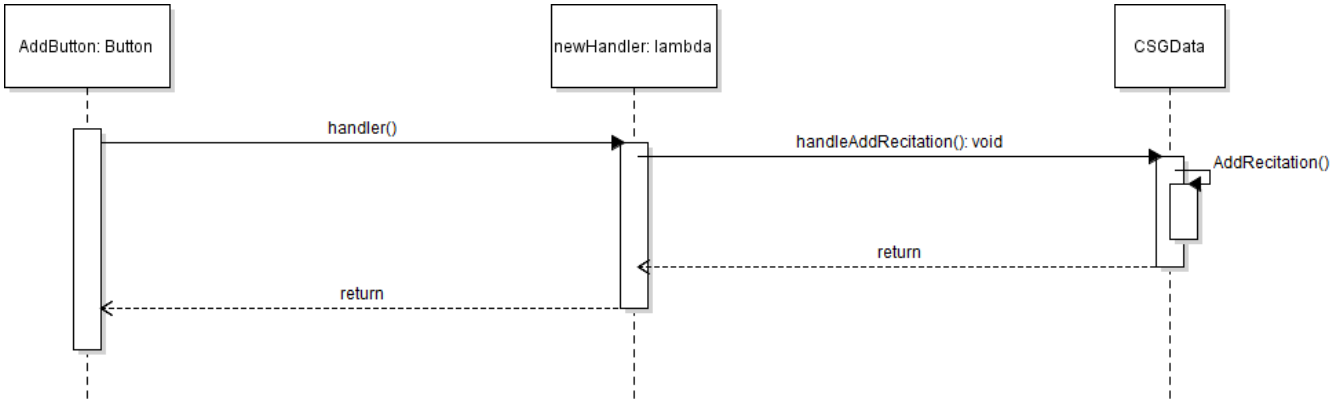


Figure 4.22: addRecitationButton UML Sequence Diagram

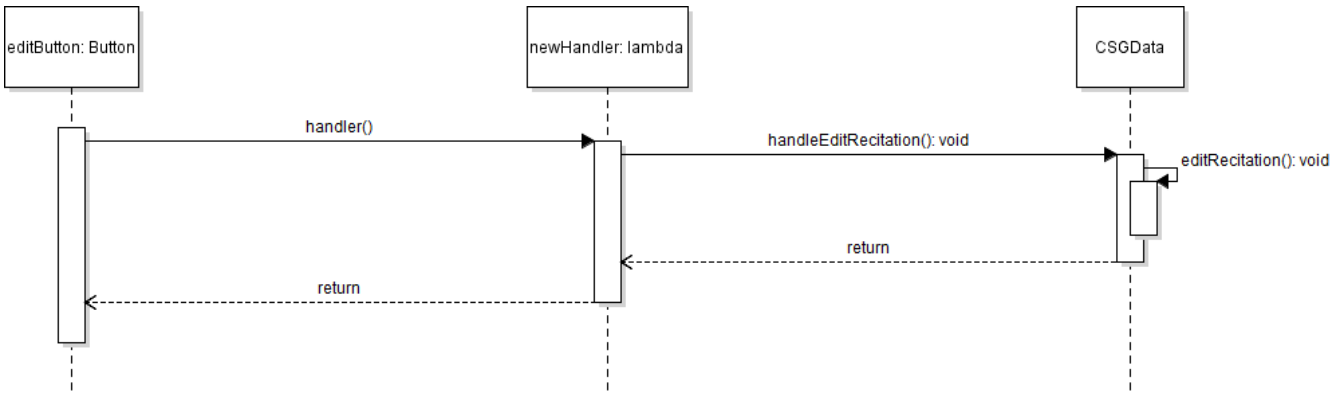


Figure 4.23: editRecitationButton UML Sequence Diagram

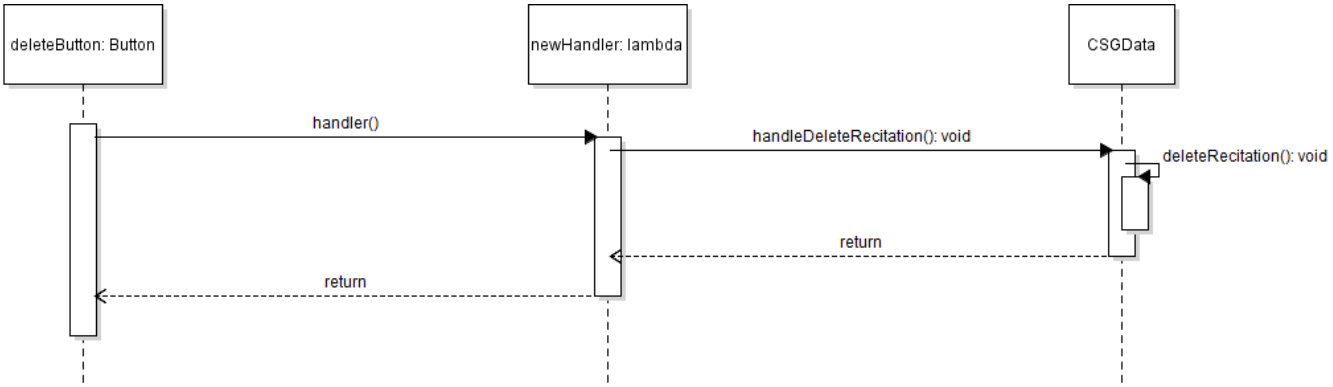


Figure 4.24: deleteRecitationButton UML Sequence Diagram

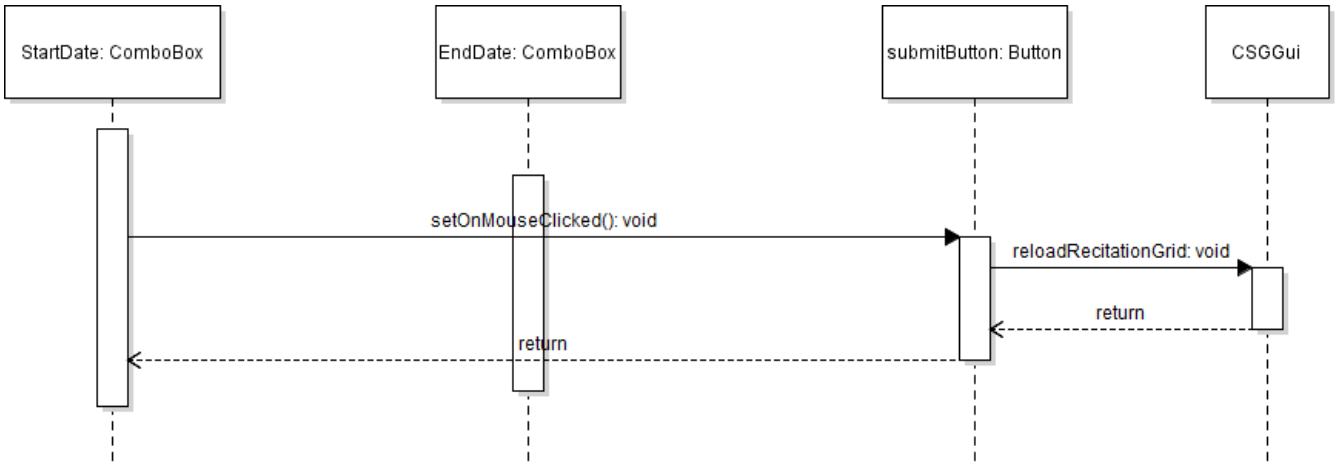


Figure 4.25: editRecitationDatesButton UML Sequence Diagram

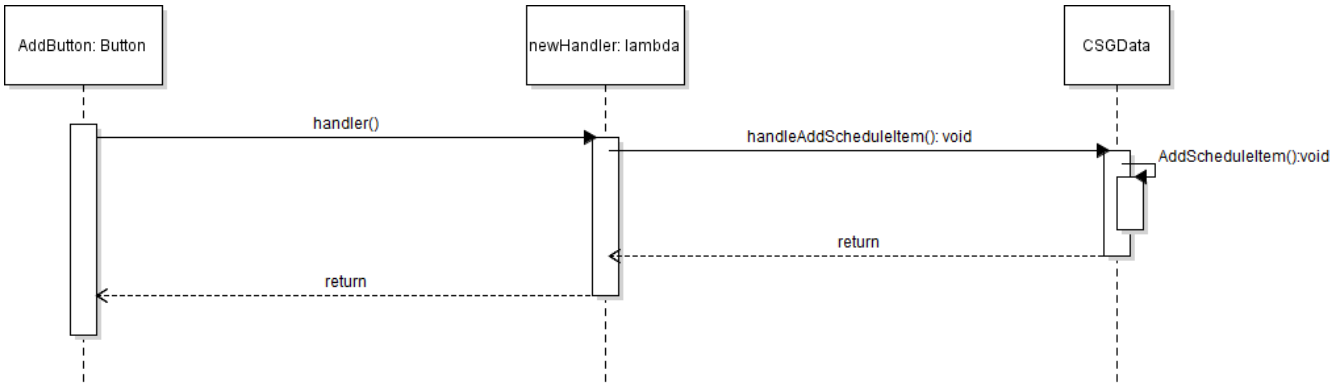


Figure 4.26: addScheduleItemButton UML Sequence Diagram

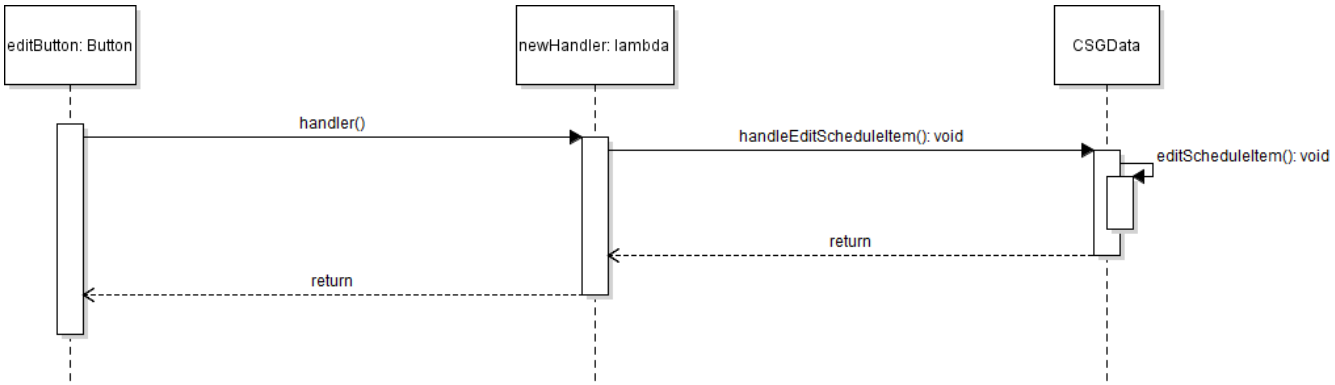


Figure 4.27: editScheduleItemButton UML Sequence Diagram

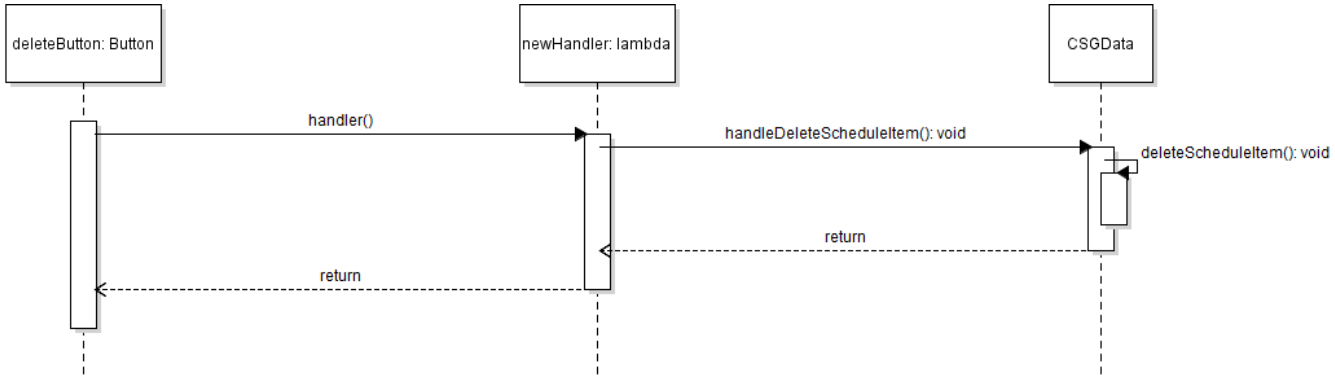


Figure 4.28: deleteScheduleItemButton UML Sequence Diagram

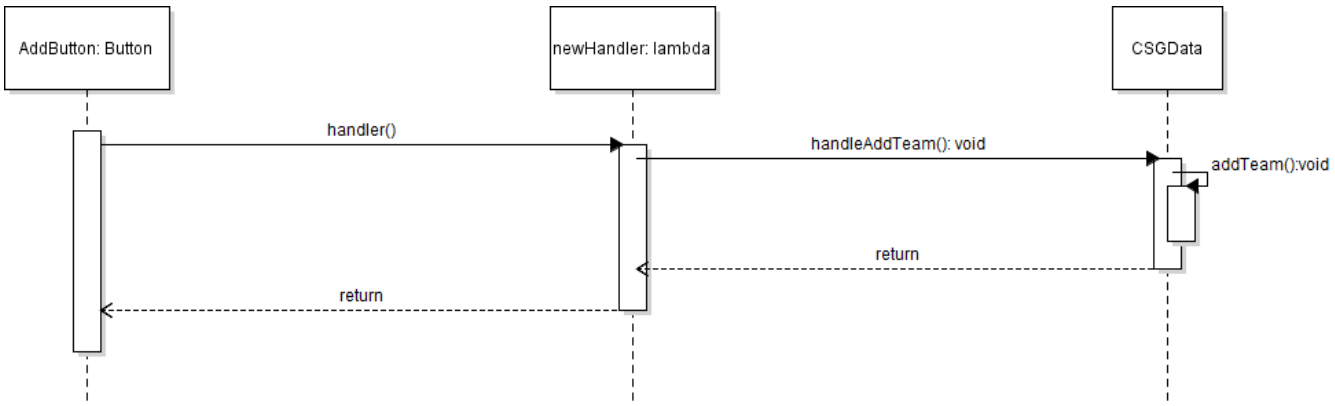


Figure 4.29: addTeamButton UML Sequence Diagram

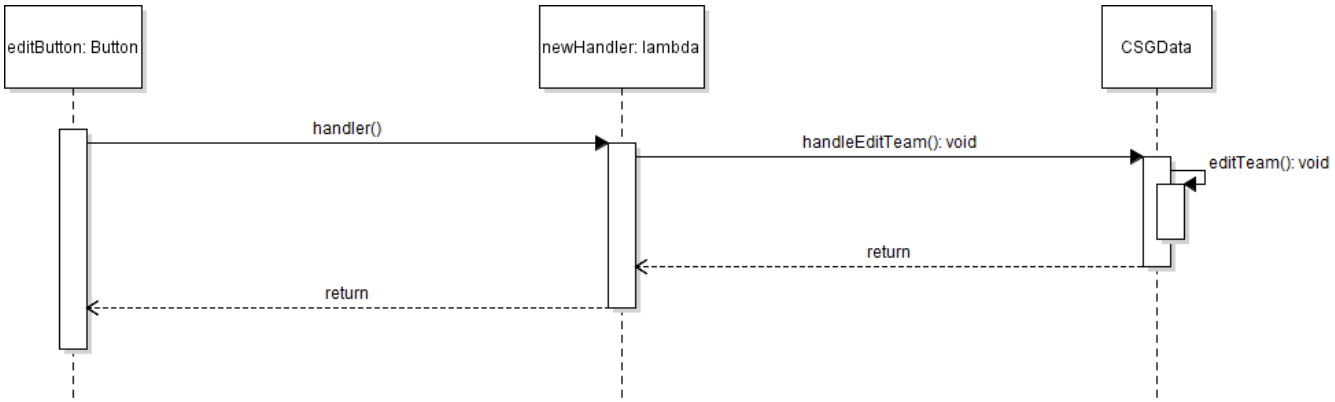


Figure 4.30: editTeamButton UML Sequence Diagram

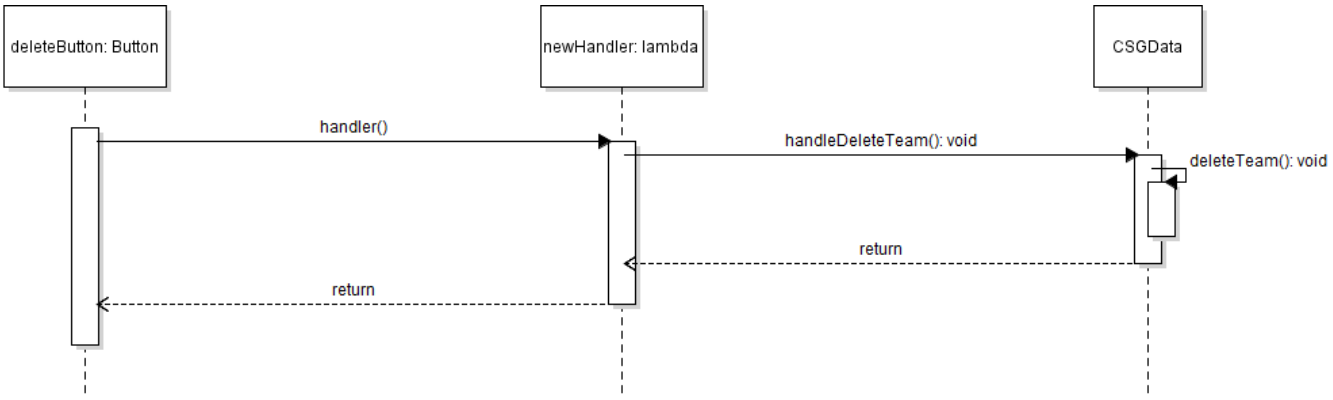


Figure 4.31: deleteTeamButton UML Sequence Diagram

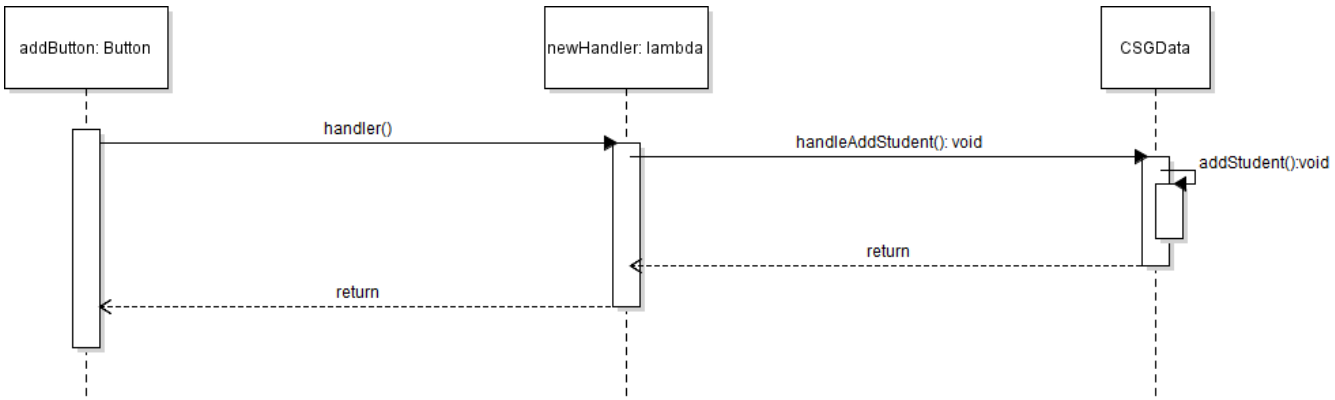


Figure 4.32: addStudentButton UML Sequence Diagram

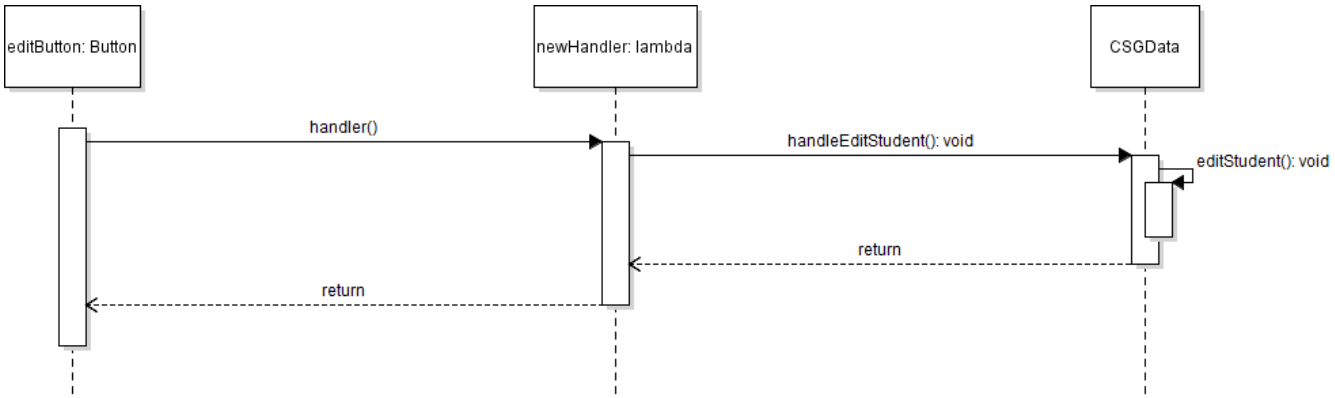


Figure 4.33: editStudentButton UML Sequence Diagram

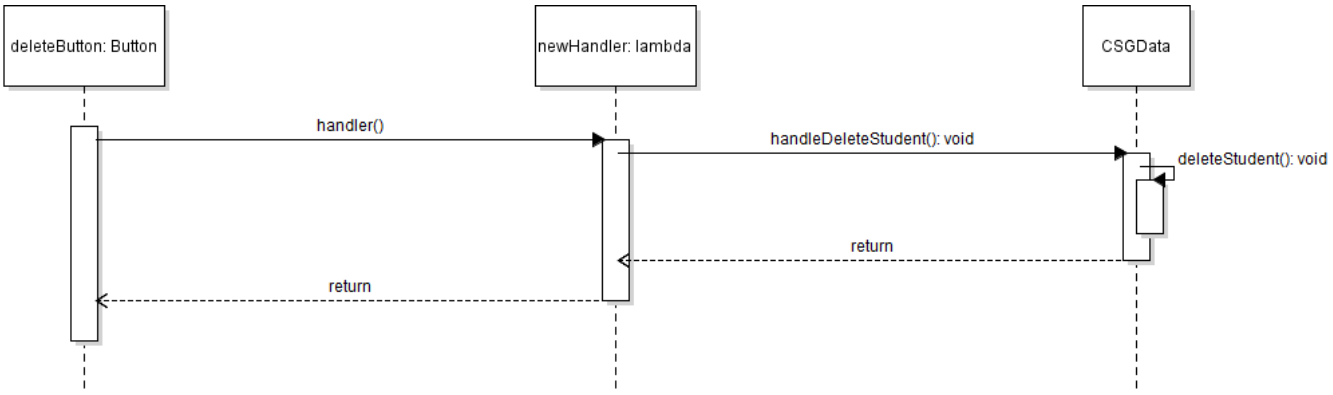
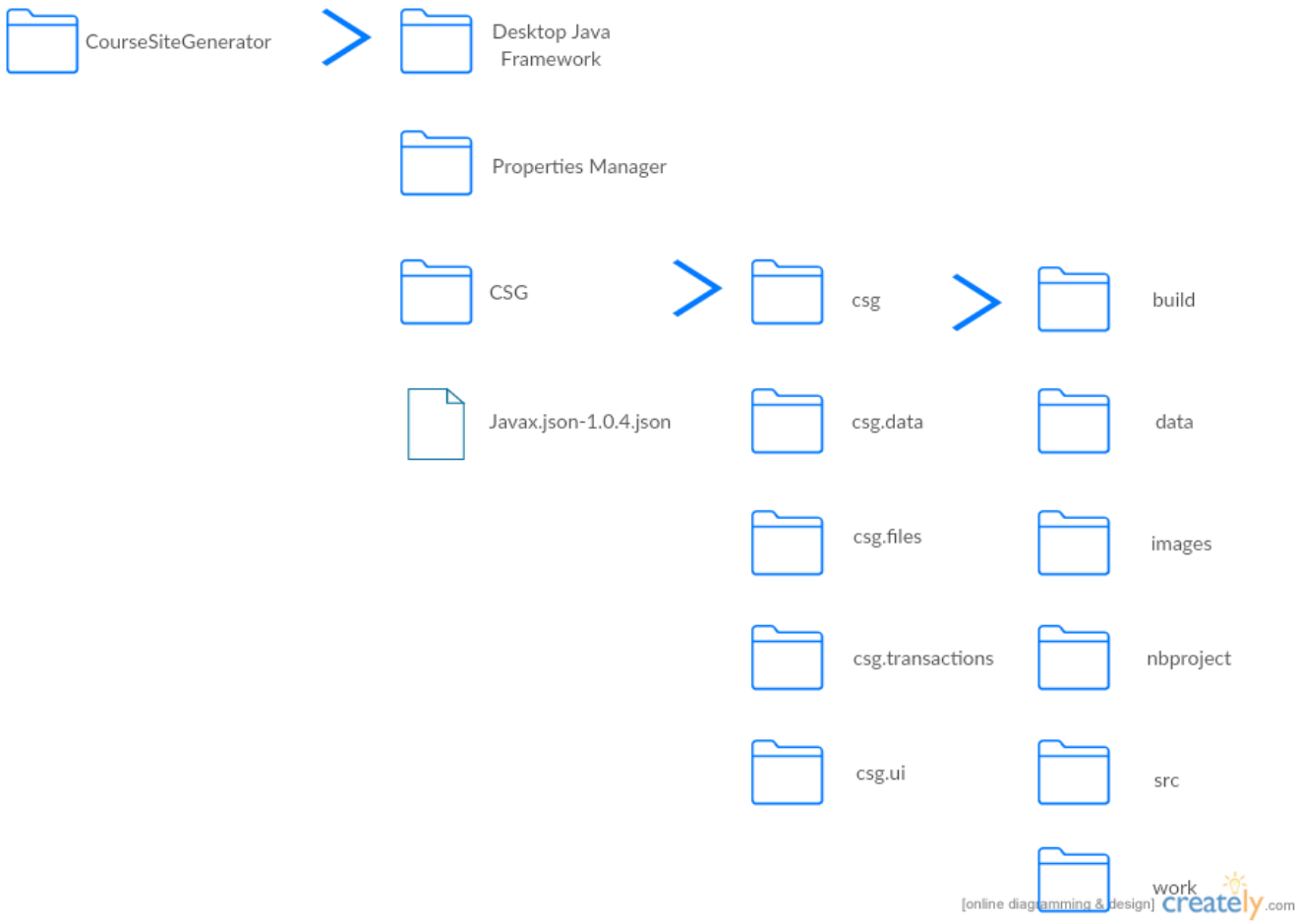


Figure 4.34: deleteStudentButton UML Sequence Diagram

5. File Structure and Format

Note that the Desktop Java Framework will be provided inside DesktopJavaFramework.jar, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the Course Site Generator application and will be included in the deployment of a single, executable folder named CSG. Note that all necessary data and art files must accompany this program. Figure 5.1 specifies the necessary file structure the launched application should use. Note that all necessary images should of course go in the image directory



5. 1 CourseSiteGenerator File Structure

The json files provide the file and state names for all website objects states in the application. The files are raw json files that can be used to describe website objects, and will be put into a folder. Each json file will be structured as followed:

TAsData.json:

```
{
	“startHour”: “ “,
	“endHour”: “ “,
	“undergrad_tas”:[
		{
			“name”: “ ”,
			“email”: “ ”,
			“isUndergrad”: “ “,
		}
	],
	“officeHours”:[
		{
			“day”: “ ”,
			“time”: “ “,
			“name”: “ “
		}
	]
}
```

This can be described as:

- startHour: A string that ranges from 0 to 23 that holds the start time of the grid.
- endHour: A string that ranges from 0 to 23 that holds the end time of the grid.
- undergrad_tas: Holds the TAs that will be in the Course Site.
- name: A string that holds the name of the TA.
- email: A string that holds the email of the TA.
- isUndergrad: A string that is “true” if the TA is an undergrad or “false” if not.
- officeHours: Holds the office hours of each time slot.
- day: A string that holds the day of the time slot.
- time: A string that holds the time of the time slot.
- name: A string that holds the name of the TA in the time slot.

RecitationsData.json:

```
{
	“recitations”: [
		{
			“section”: “ “,
			“instructor”: “ “,
			“day/time”: “ “,
			“location”: “ “,
			“firstTa”: “ “,
			“secondTa”: “ “
		}
	]
}
```

This can be described as:

- recitation: Holds the list of recitations that will be in the Course Site.
- section: A string that holds the section number of the recitation.
- instructor: A string that holds the name of the instructor for the recitation.
- day/time: A string that holds the day and time of the recitation.
- location: A string that holds the location of where the recitation will be.
- firstTA: A string for the name of the first TA. This TA’s name will be in the TA list.
- secondTA: A string for the name of the second TA. This TA’s name will be in the TA list.

SchedulesData.json:

```
{
	“startDate”: “ “,
	“endDate”: “ “,
	“scheduleItems”: [
		{
			“type”: “ “,
			“date”: “ “,
			“title”: “ “,
			“topic”: “ “,
		}
	]
}
```

This can be described as:

- startDate: A string that gives the start date of the schedule items.
- endDate: A string that gives the end date of the schedule items.
- scheduleItems: Holds the list of scheduleItems that will be in the Course Site.
- type: A string that holds what the type of the scheduleItem is.
- date: A string that holds the date of the scheduleItem.
- title: A string for the name of the scheduleItem.
- topic: A string for what the scheduleItem is about.

TeamsAndStudents.json:

```
{
	“teams”: [
		{
			“name”: “ “,
			“color”: “ “,
			“textColor”: “ “,
			“link”: “ “,
		}
	]
	“students”: [
		{
			“firstName”: “ “,
			“secondName”: “ “,
			“team”: “ “,
			“role”: “ “,
		}

	]
}
```

This can be described as:

- teams: Holds the list of teams that will be in the Course Site.
- name: A string that holds the name of the team.
- color: A string that holds the color correlated with the project.
- textColor: A string that holds the text color correlated with the project.
- link: A string that holds the link for the team.
- students: Holds the list of students that will be in the Course Site.
- firstName: A string that holds the first name of the student.
- secondName: A string that holds the last name of the student.
- team: A string for the team that the student is in.
- role: A string for what the student’s role in the team is.

Projects.json:

{

“subject”: “ “,

“number”: “ “,

“semester”: “ “,

“year”: “ “,

“title”: “ “,

“instructorName”: “ “,

“instructorHome”: “ “,

“templateDirectory”: “ “,

“banner”: “ “,

“leftFooter”: “ “,

“rightFooter”: “ “,

“styleSheet”: “ “,

}

This can be described as:

- subject: A string that holds the subject of the site.
- number: A string that holds the course number of the subject.
- semester: A string that holds the semester for the course site.
- year: A string that holds the year for the course site.
- title: A string that holds the title of the site.
- instructorName: A string that holds the name of the instructor teaching the course.
- instructorHome: A string that holds the instructor’s home page link.
- templateDirectory: A string that holds the directory to the template.
- banner: A string that holds the file link to the banner image.
- leftFooter: A string that holds the file link to the left footer image.
- rightFooter: A string that holds the file link to the right footer image.
- styleSheet: A string for the name of the stylesheet.

6. Supporting Information

Note that this document should serve as a reference for those implementing the code, so we'll provide a table of contents to help quickly find important sections.

6.1 Table of contents

6.2 Appendixes

N/A