The *Code Check™*

Software Design Description

Author: Nick Gundobin

Debugging Enterprises™

July, 2017 Version 1.0

Abstract: This document describes the software design for Code Check, an application to maintain and check student's code for plagiarism.

Based on IEEE Std 830TM-1998 (R2009) document format

1 Introduction

Code Check™ is a desktop application that makes it easier for Teaching Assistants to extract student submissions into a proper format to setup grading as well as setting up checks for code plagiarism (what we'll call a code check). 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 Code Check 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

The goal of this application is for Teaching Assistants to have an easier time extracting student assignments for the purposes of grading and plagiarism checking. This application will take a collection of student submissions zipped up into a single file downloaded from Blackboard and will then extract these submissions and organize them into directories of unzipped work, ready to grade. In addition, Teaching Assistants will be given the opportunity to extract and organize the submitted source code to help with code plagiarism checks. The design contains description of the Code Check and also frameworks such as DesktopJavaFramework and PropertiesManager that have been created by Professor Richard Mckenna. Note that Java is the target language for this software design.

1.3 Definitions, acronyms, and abbreviations

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

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

JSON (JavaScript Object Notation) - a lightweight data-interchange format.

JavaFX - a software platform for creating and delivering desktop applications, as well as rich internet applications (RIAs) that can run across a wide variety of devices.

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.

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

UI (User Interface) - the space where interactions between humans and machines occur.

UML – Unified Modeling Language, a standard set of document formats for designing software graphically. Use Case Diagram – A UML document format that specifies how a user will interact with a system.

XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

1.4 References

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

1.5 Overview

This Software Design Description document provides a working design for the Code Check software application as described in the Code Check 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 VioletUML editor.

2 Package-Level Design Viewpoint

As mentioned, this design will encompass the Code Check application, DesktopJavaFramework and PropertiesManager to be used in its construction. In building of our Code Check application 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 DesktopJavaFramework, PropertiesManager and CodeCheckApp Overview

The CodeCheckApp will be designed and developed in tandem with PropertiesManager and DesktopJavaFramework. Figure 2.1 specifies all the components to be developed and places all classes in home packages.

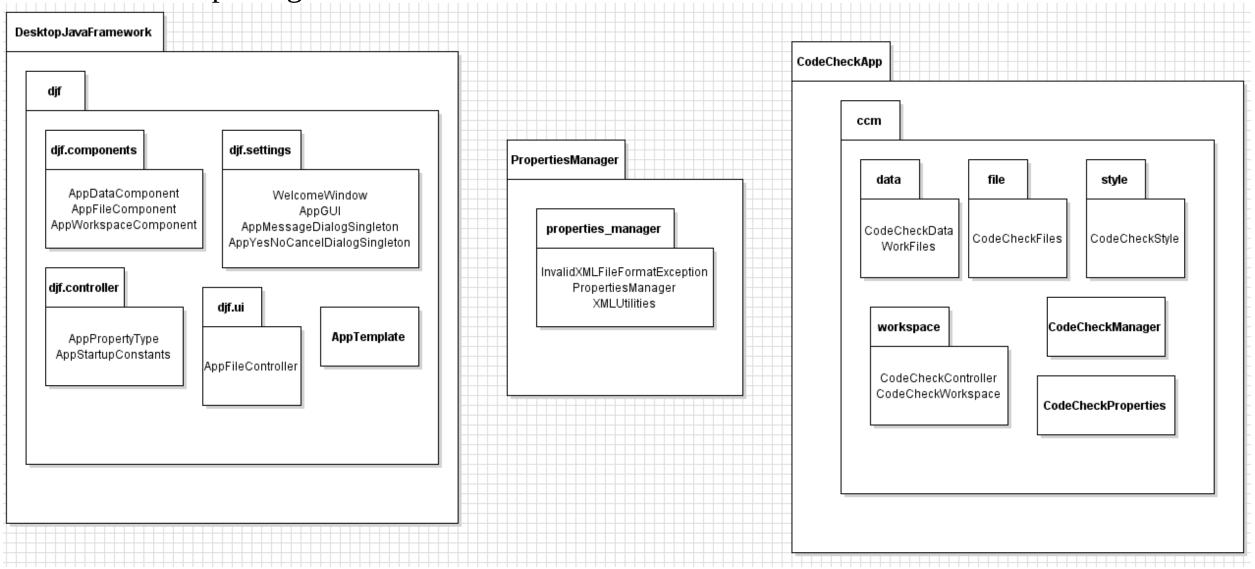


Figure 2.1: Design Packages Overview

2.2 Java API Usage

Check Code application will be developed using the Java programming language with help of some CSS for UI design as well as JSON and XML formats. As such, this design will make use of the classes specified in Figure 2.2.

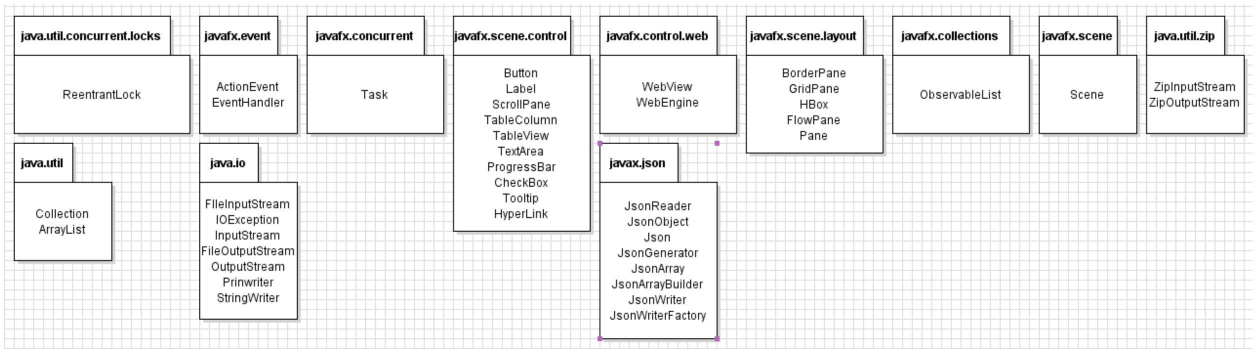


Figure 2.2: API Usages

2.3 Java API Usage Descriptions

Tables 2.1-2.10 below summarize how each of these classes will be used

Class/Interface	Use
ReentrantLock	Text input component that allows a user to enter multiple lines of plain text.

Table 2.1: Uses for classes in the Java API's java.util.concurrent package

Class/Interface	Use
ActionEvent	An Event representing some type of action.
EventHandler	Handler for events of a specific class / type.

Table 2.2: Uses for classes in the Java API's javafx.event package

Class/Interface	Use
Task	Tasks exposes additional state and observable properties useful for programming asynchronous tasks in JavaFX, as defined in the Worker interface.

Table 2.3: Uses for classes in the Java API's javafx.concurrent package

Class/Interface	Use
Button	A simple button control. The button control can contain text and/or a graphic. A button control has three different modes
Label	Label is a non-editable text control. A Label is useful for displaying text that is required to fit within a specific space, and thus may need to use an ellipsis or truncation to size the string to fit.

ScrollPane	A Control that provides a scrolled, clipped viewport of its contents. It allows the user to scroll the content around either directly (panning) or by using scroll bars.
SelectionMode	JavaFX uses enumeration SelectionMode to specify how many items maybe selected.
SplitPane	A control that has two or more sides, each separated by a divider, which can be dragged by the user to give more space to one of the sides, resulting in the other side shrinking by an equal amount.
TableColumn	A TableView is made up of a number of TableColumn instances. Each TableColumn in a table is responsible for displaying (and editing) the contents of that column.
TableView	The TableView control is designed to visualize an unlimited number of rows of data, broken out into columns.
TextArea	Text input component that allows a user to enter multiple lines of plain text.
ProgressBar	A specialization of the ProgressIndicator which is represented as a horizontal bar.
CheckBox	A tri-state selection Control typically skinned as a box with a checkmark or tick mark when checked.
Tooltip	Tooltips are common UI elements which are typically used for showing additional information about a Node in the scenegraph when the Node is hovered over by the mouse.
HyperLink	An HTML like label which can be a graphic and/or text which responds to rollovers and clicks.

Table 2.4: Uses for classes in the Java API's javafx.scene.control package

Class/Interface	**Use**
Pane	Base class for layout panes which need to expose the children list as public so that users of the subclass can freely add/remove children.
FlowPane	FlowPane lays out its children in a flow that wraps at the flowpane's boundary.

GridPane	GridPane lays out its children within a flexible grid of rows and columns. If a border and/or padding is set, then its content will be layed out within those insets.
BorderPane	BorderPane lays out children in top, left, right, bottom, and center positions.
HBox	HBox lays out its children in a single horizontal row.

Table 2.5: Uses for classes in the Java API's javafx.scene.layout package

Class/Interface	Use
ObservableList	A list that allows listeners to track changes when they occur.

Table 2.6: Uses for classes in the Java API's javafx.collections package

Class/Interface	Use
SimpleStringProperty	This class provides a full implementation of a Property wrapping a String value.
StringProperty	This class provides a full implementation of a Property wrapping a String value.

Table 2.7: Uses for classes in the Java API's javafx.beans.property package

Class/Interface	Use
WebEngine	WebEngine is a non-visual object capable of managing one Web page at a time.
WebView	WebView is a Node that manages a WebEngine and displays its content.

Table 2.8: Uses for classes in the Java API's javafx.control.web package

Class/Interface	Use
Scene	The JavaFX Scene class is the container for all content in a scene graph.

Table 2.9: Uses for classes in the Java API's javafx.scene package

Class/Interface	Use
ZipInputStream	This class implements an input stream filter for reading files in the ZIP file format. Includes support for both compressed and uncompressed entries
ZipOutputStream	This class implements an output stream filter for writing files in the ZIP file format. Includes support for both compressed and uncompressed entries.

Table 2.10: Uses for classes in the Java API's java.util.zip package

Class/Interface	Use
Collection	For storing groups of data, Values in a Map, i.e. TreeMap or HashMap are stored in Collections. We'll need to iterate through Collections for rendering.
ArrayList	Resizable-array implementation of the List interface.

Table 2.10: Uses for classes in the Java API's java.util package

Class/Interface	Use
FileInputStream	A FileInputStream obtains input bytes from a file in a file system. What files are available depends on the host environment.
IOException	Signals that an I/O exception of some sort has occurred. This class is the general class of exceptions produced by failed or interrupted I/O operations.
InputStream	This abstract class is the superclass of all classes representing an input stream of bytes.
FileOutputStream	A file output stream is an output stream for writing data to a File or to a FileDescriptor.

OutputStream	This abstract class is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.
PrintWriter	Prints formatted representations of objects to a text-output stream. This class implements all of the print methods found in PrintStream.
StringWriter	A character stream that collects its output in a string buffer, which can then be used to construct a string.

Table 2.11: Uses for classes in the Java API's java.io package

Class/Interface	**Use**
JsonReader	Reads a JSON object or an array structure from an input source.
JsonObject	JsonObject class represents an immutable JSON object value (an unordered collection of zero or more name/value pairs). It also provides unmodifiable map view to the JSON object name/value mappings.
JsonGenerator	Writes JSON data to an output source in a streaming way. The class Json contains methods to create generators for character or output streams (Writer and OutputStream).
JsonArray	JsonArray represents an immutable JSON array (an ordered sequence of zero or more values). It also provides an unmodifiable list view of the values in the array.
JsonArrayBuilder	A builder for creating JsonArray models from scratch. This interface initializes an empty JSON array model and provides methods to add values to the array model and to return the resulting array. The methods in this class can be chained to add multiple values to the array.
JsonWriter	Writes a JSON object or array structure to an output source.

JsonWriterFactory	Factory to create JsonWriter instances. If a factory instance is configured with some configuration, that would be used to configure the created writer instances.

Table 2.4: Uses for classes in the Java API's javafx.json package

3 Class-Level Design Viewpoint

As mentioned, this design will encompass the DesktopFrameWork application, PropertiesManager and CodeCheckApp. 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.

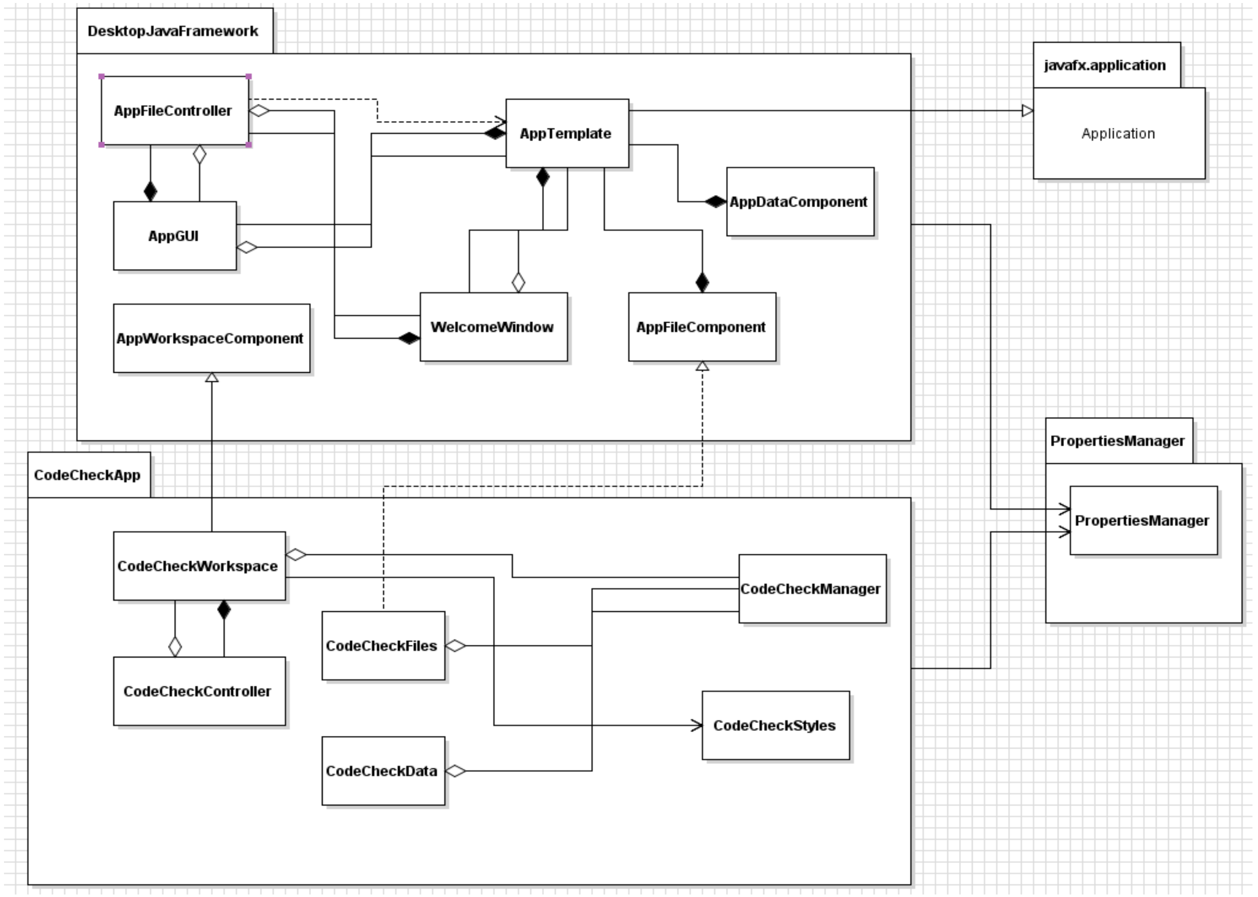


Figure 3.1: CodeCheckApp Overview UML Class Diagram

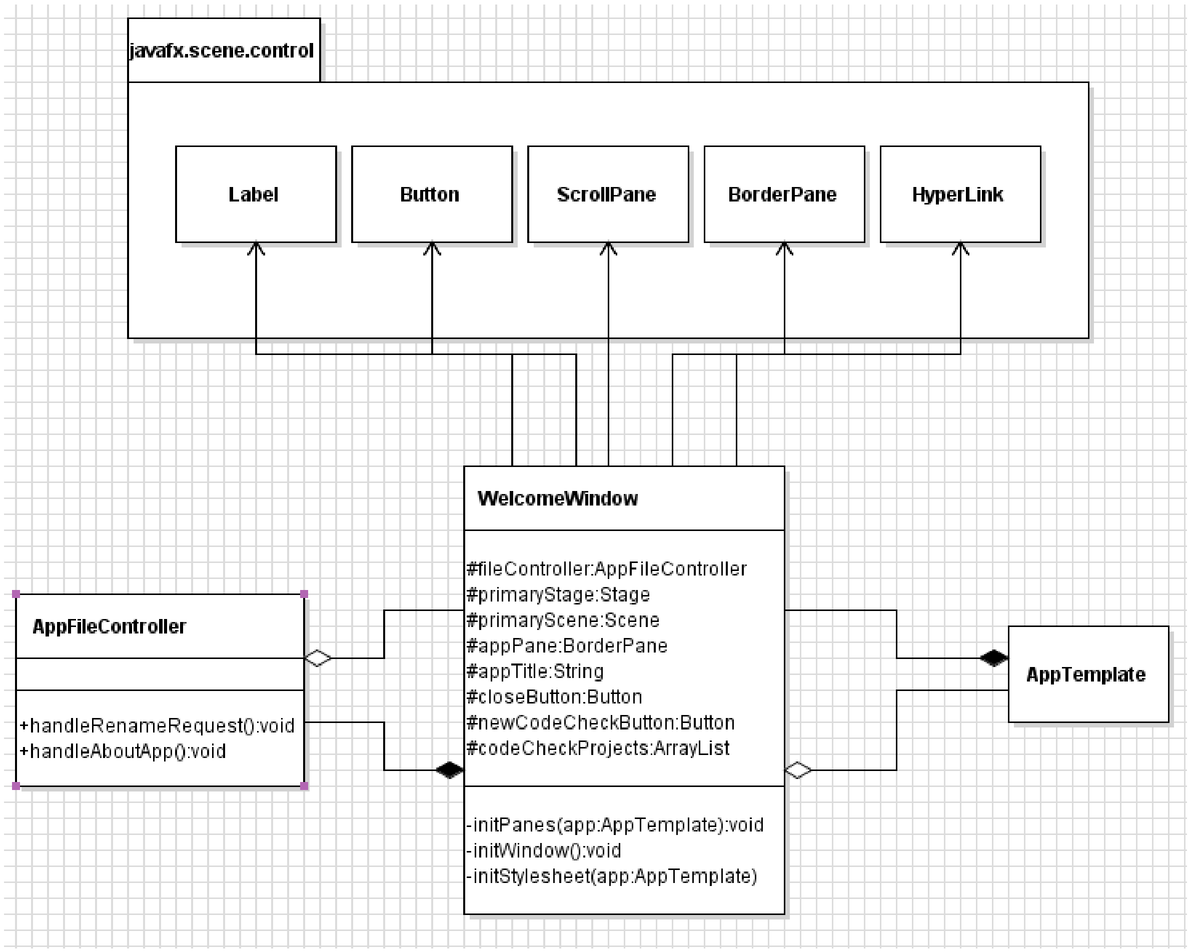


Figure 3.2: WelcomeWindow and modified AppFileController Overview UML Class Diagram

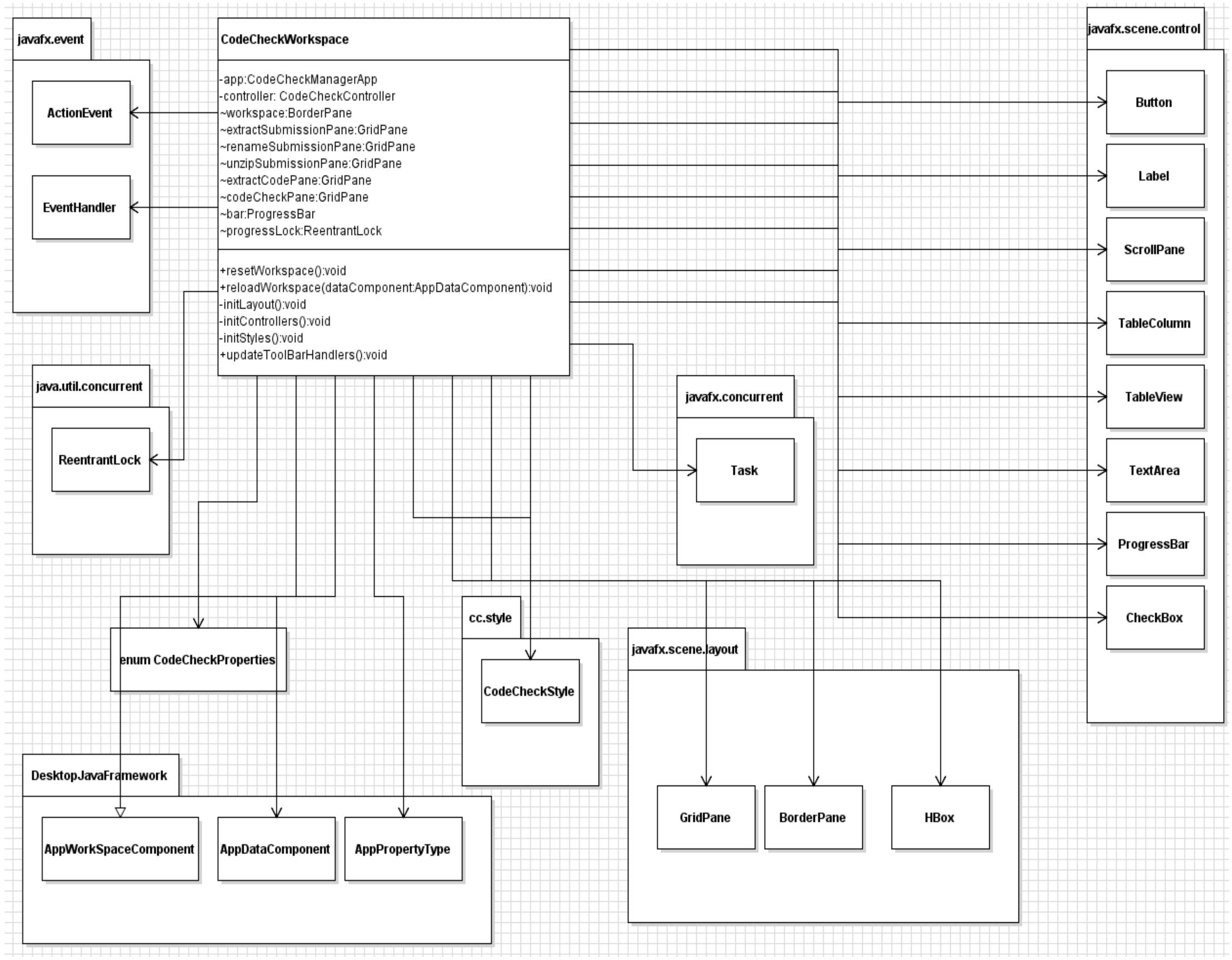


Figure 3.3: CodeCheckWorkspace Overview UML Class Diagram

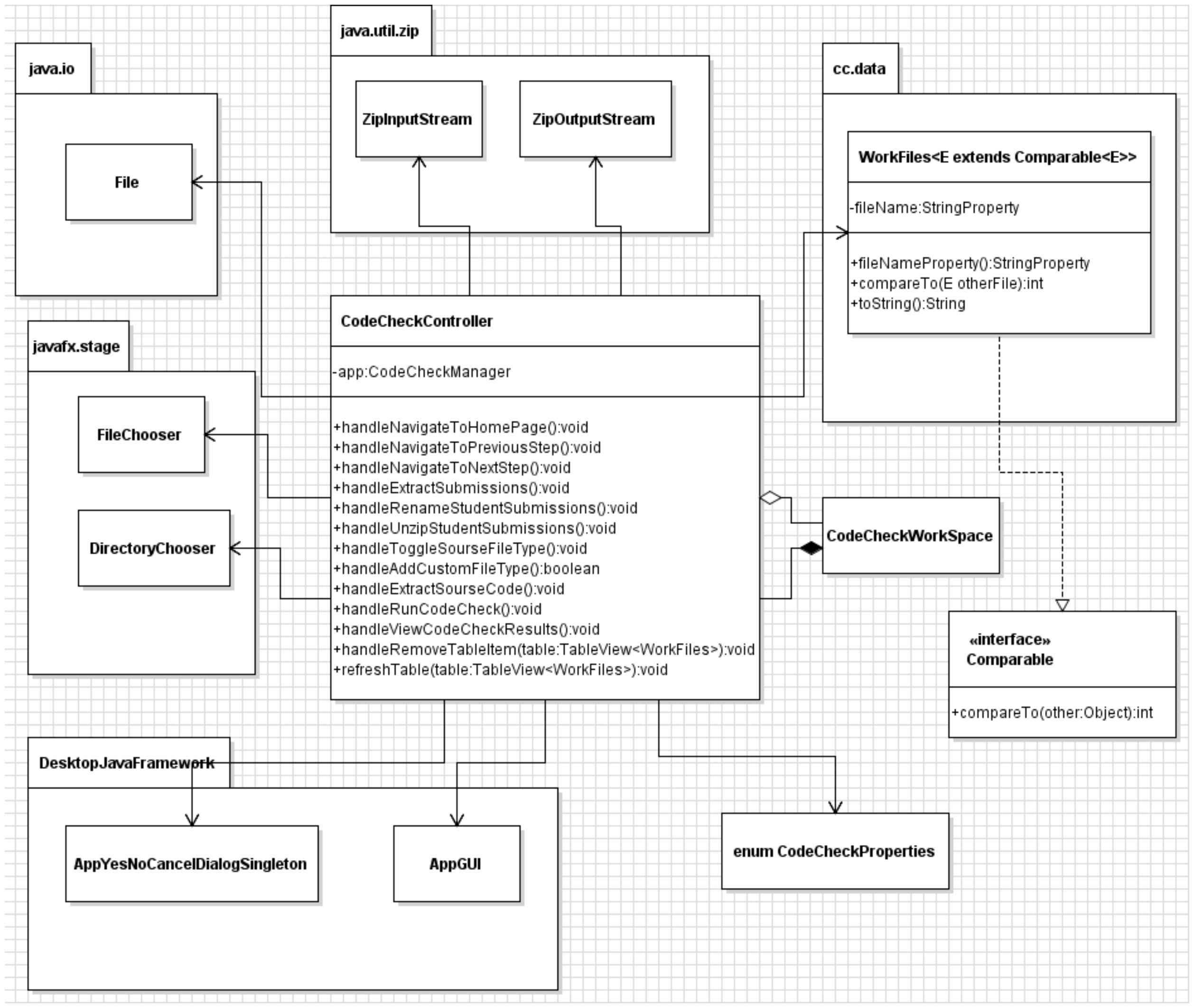

java.io
File
java.util.zip
ZipInputStream
ZipOutputStream
cc.data
WorkFiles<E extends Comparable<E>>
-fileName:StringProperty
+fileNameProperty():StringProperty
+compareTo(E otherFile):int
+toString():String
javafx.stage
FileChooser
DirectoryChooser
CodeCheckController
-app:CodeCheckManager
+handleNavigateToHomePage():void
+handleNavigateToPreviousStep():void
+handleNavigateToNextStep():void
+handleExtractSubmissions():void
+handleRenameStudentSubmissions():void
+handleUnzipStudentSubmissions():void
+handleToggleSourseFileType():void
+handleAddCustomFileType():boolean
+handleExtractSourseCode():void
+handleRunCodeCheck():void
+handleViewCodeCheckResults():void
+handleRemoveTableItem(table:TableView<WorkFiles>):void
+refreshTable(table:TableView<WorkFiles>):void
CodeCheckWorkSpace
«interface»
Comparable
+compareTo(other:Object):int
DesktopJavaFramework
AppYesNoCancelDialogSingleton
AppGUI
enum CodeCheckProperties

Figure 3.4: CodeCheckController and WorkFiles Overview UML Class Diagram

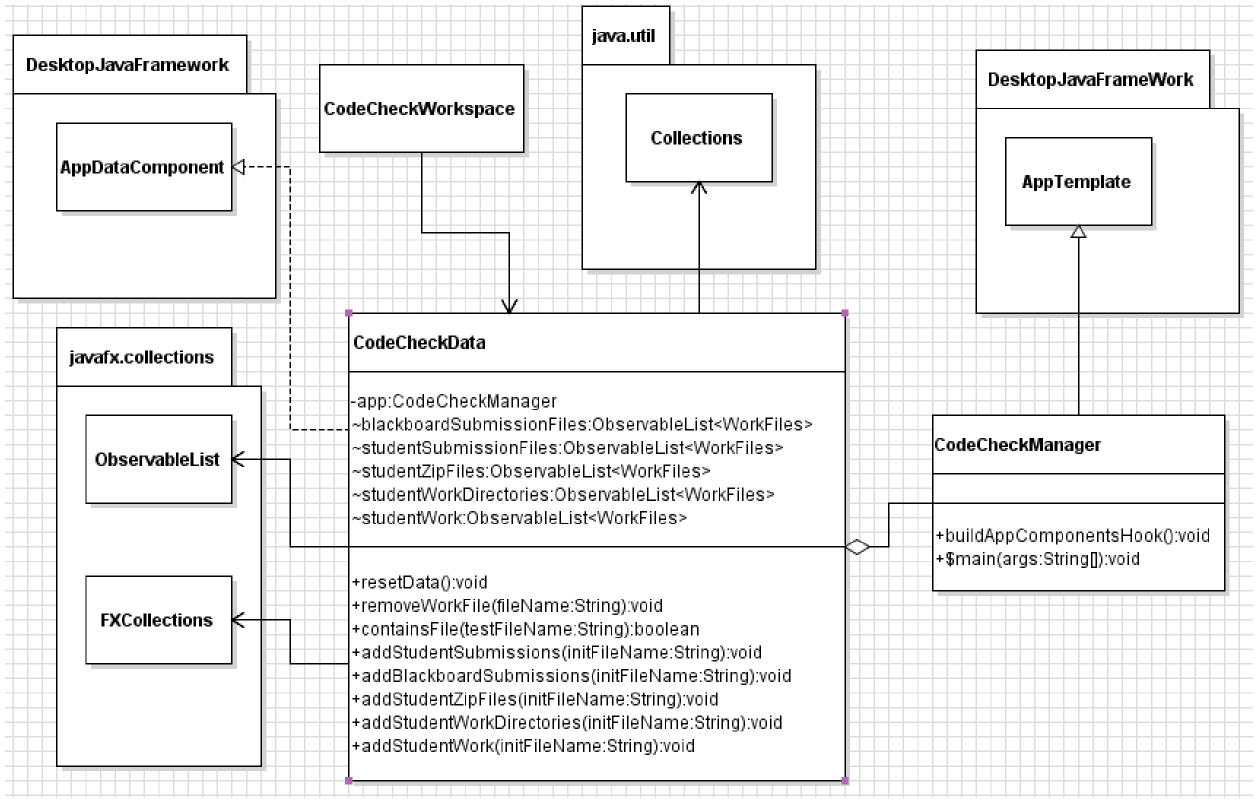

DesktopJavaFramework
AppDataComponent
CodeCheckWorkspace
java.util
Collections
DesktopJavaFrameWork
AppTemplate
javafx.collections
ObservableList
FXCollections
CodeCheckData
-app:CodeCheckManager
~blackboardSubmissionFiles:ObservableList<WorkFiles>
~studentSubmissionFiles:ObservableList<WorkFiles>
~studentZipFiles:ObservableList<WorkFiles>
~studentWorkDirectories:ObservableList<WorkFiles>
~studentWork:ObservableList<WorkFiles>
+resetData():void
+removeWorkFile(fileName:String):void
+containsFile(testFileName:String):boolean
+addStudentSubmissions(initFileName:String):void
+addBlackboardSubmissions(initFileName:String):void
+addStudentZipFiles(initFileName:String):void
+addStudentWorkDirectories(initFileName:String):void
+addStudentWork(initFileName:String):void
CodeCheckManager
+buildAppComponentsHook():void
+$main(args:String[]):void

Figure 3.5: CodeCheckData and CodeCheckManager Overview UML Class Diagram

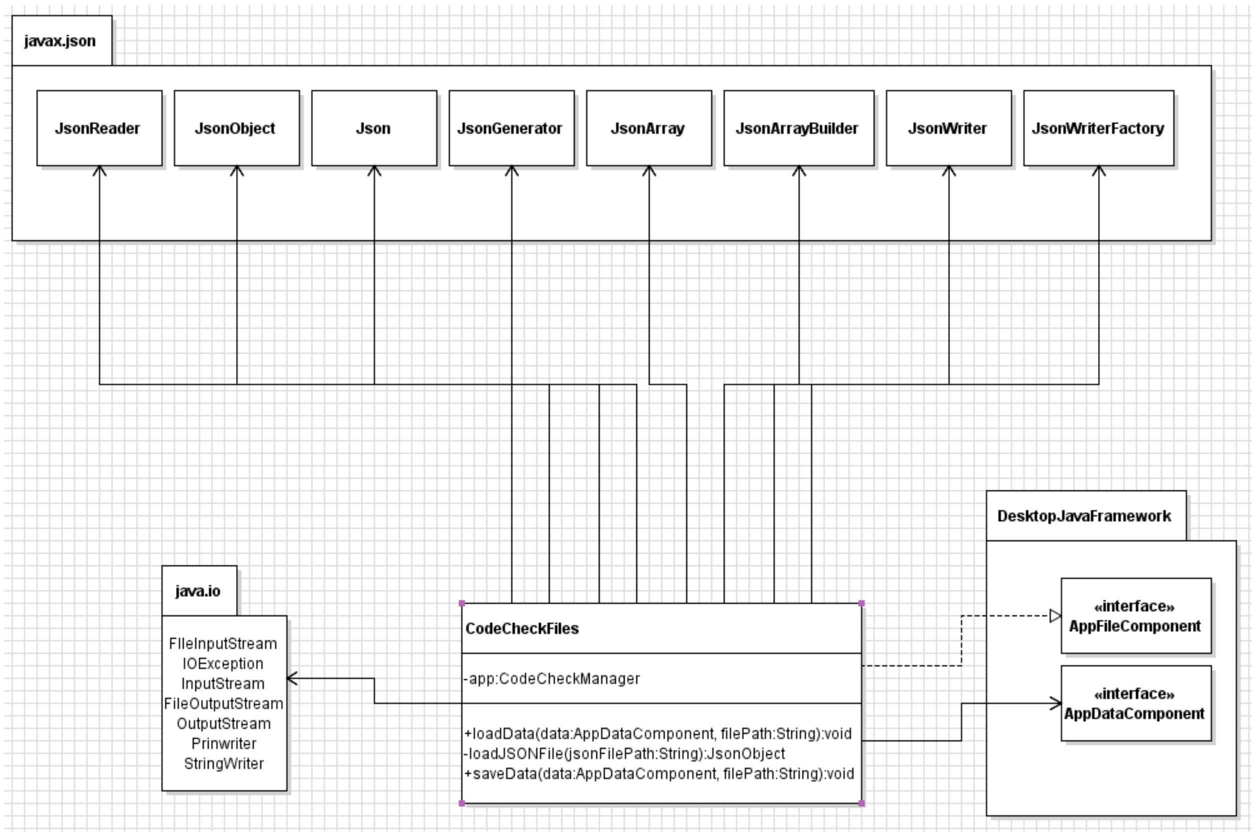


Figure 3.6: CodeCheckFiles Overview UML Class Diagram

3. 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.

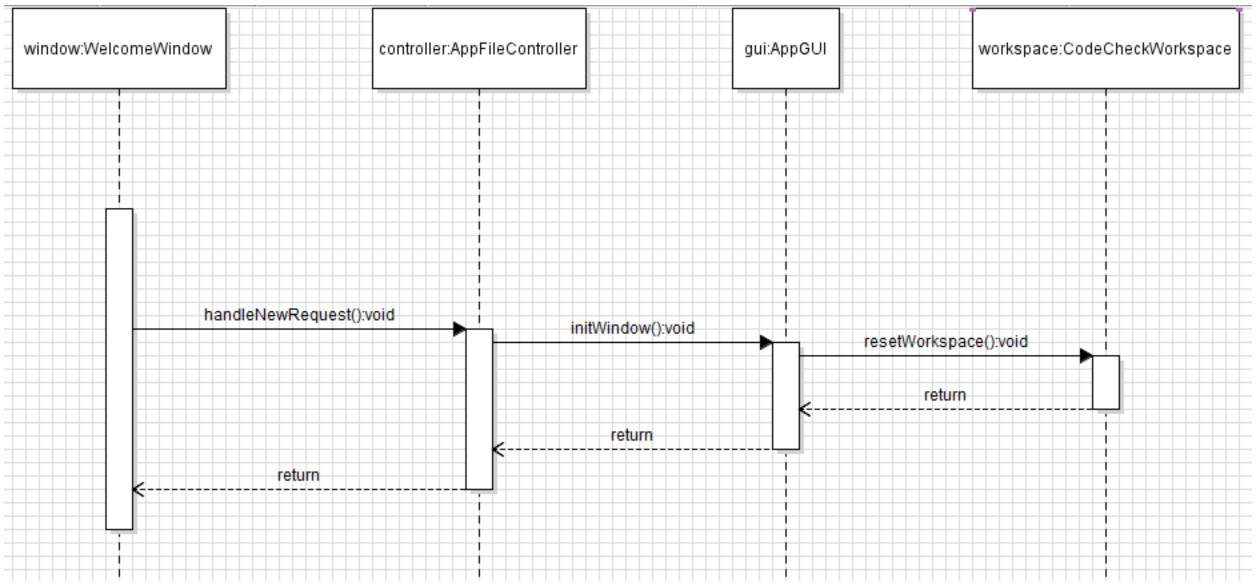


Figure 4.1: New Code Check UML Sequence Diagram

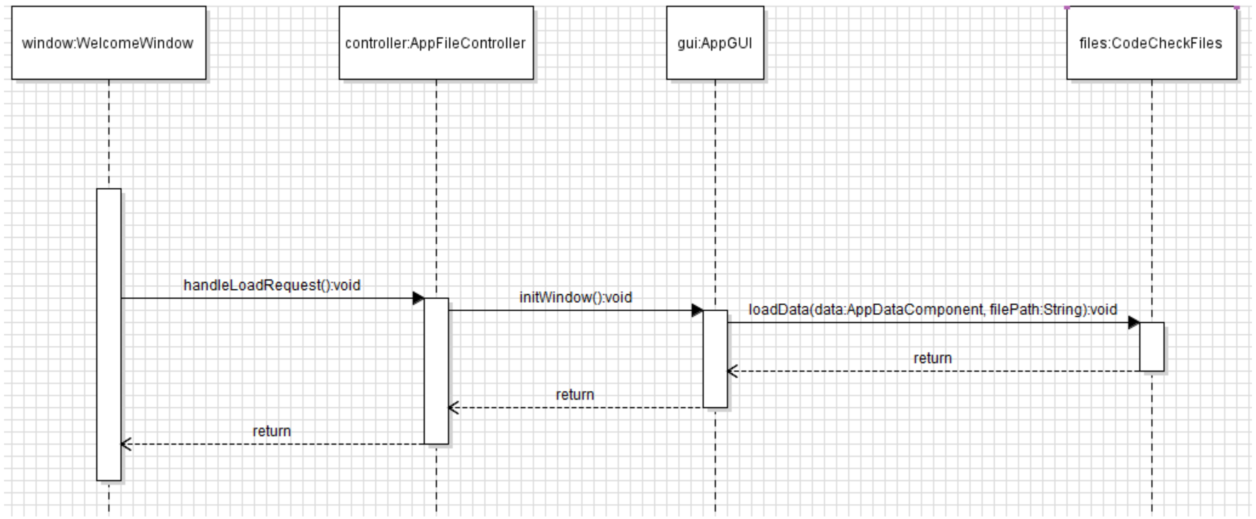


Figure 4.2: Load Code Check UML Sequence Diagram

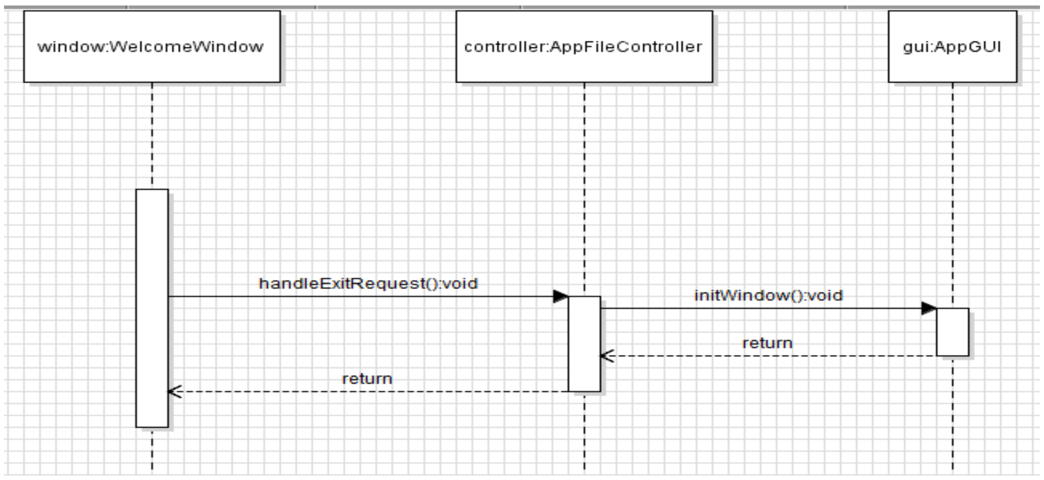


Figure 4.3: Close Code Check UML Sequence Diagram

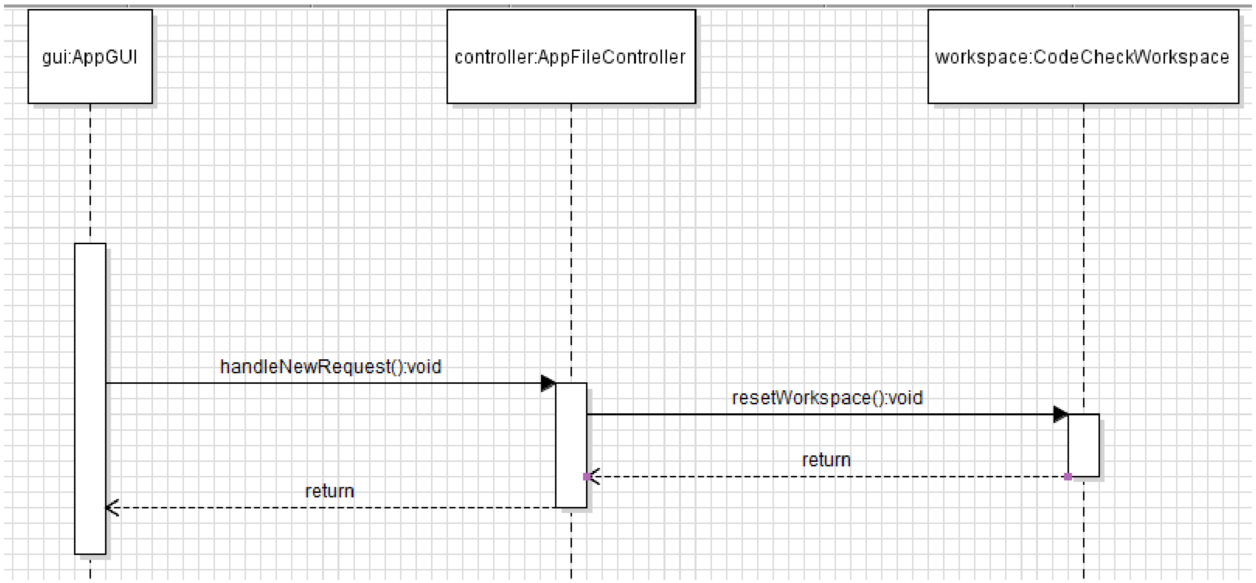


Figure 4.4: New Code Check from main window UML Sequence Diagram

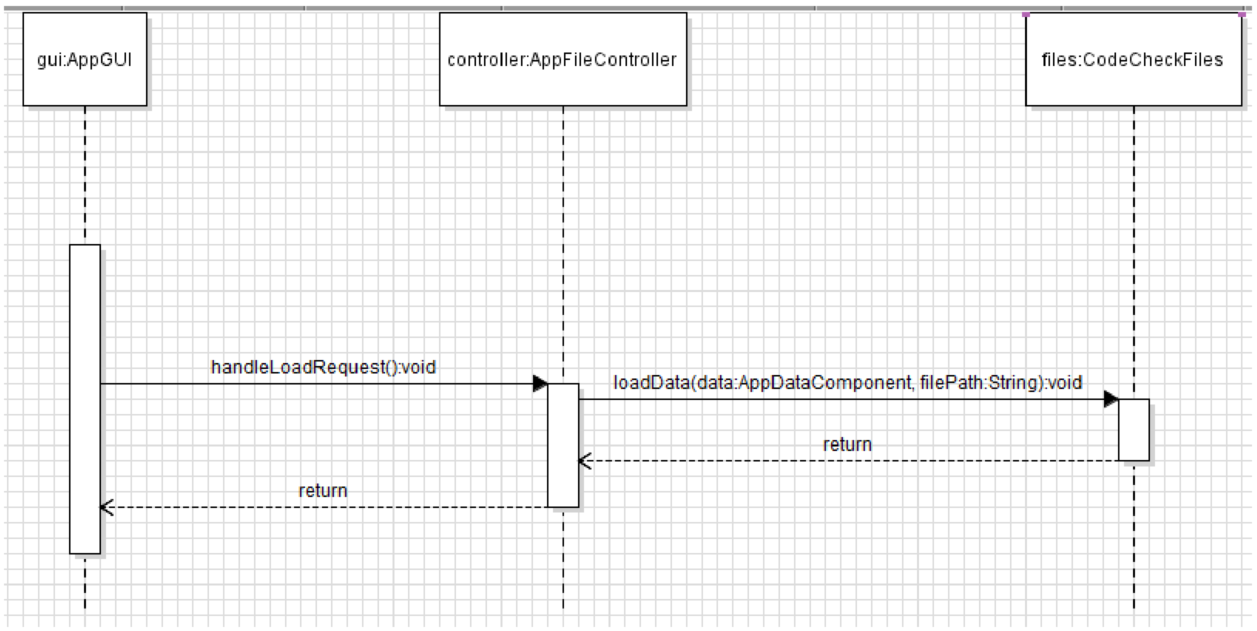

gui:AppGUI
controller:AppFileController
files:CodeCheckFiles
handleLoadRequest():void
loadData(data:AppDataComponent, filePath:String):void
return
return

Figure 4.5: Load Code Check from main window UML Sequence Diagram

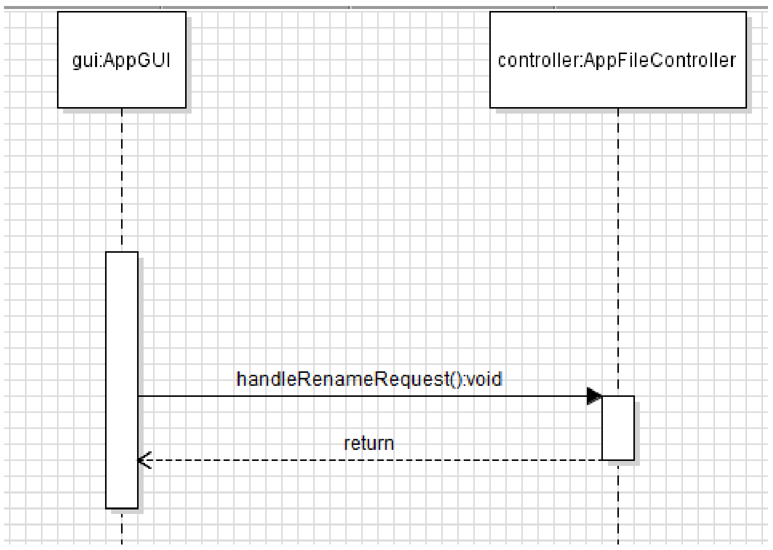


Figure 4.6: Rename Code Check from main window UML Sequence Diagram

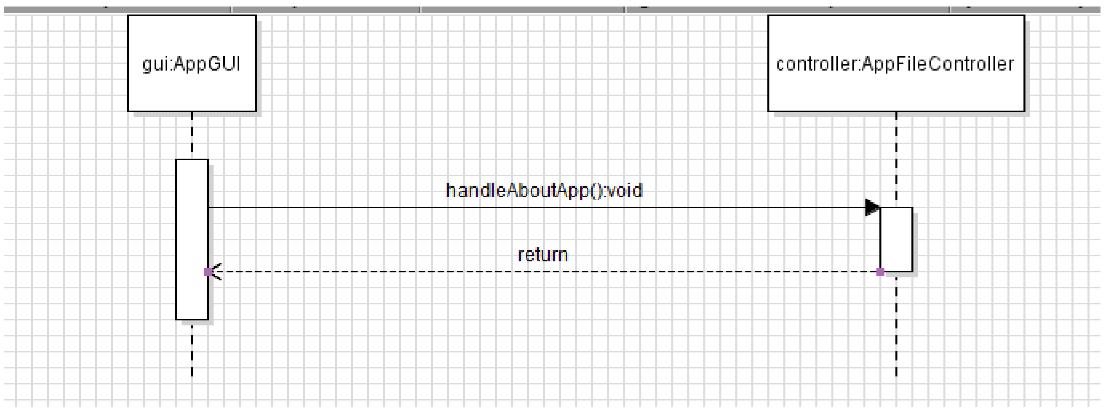


Figure 4.7: About Code Check UML Sequence Diagram

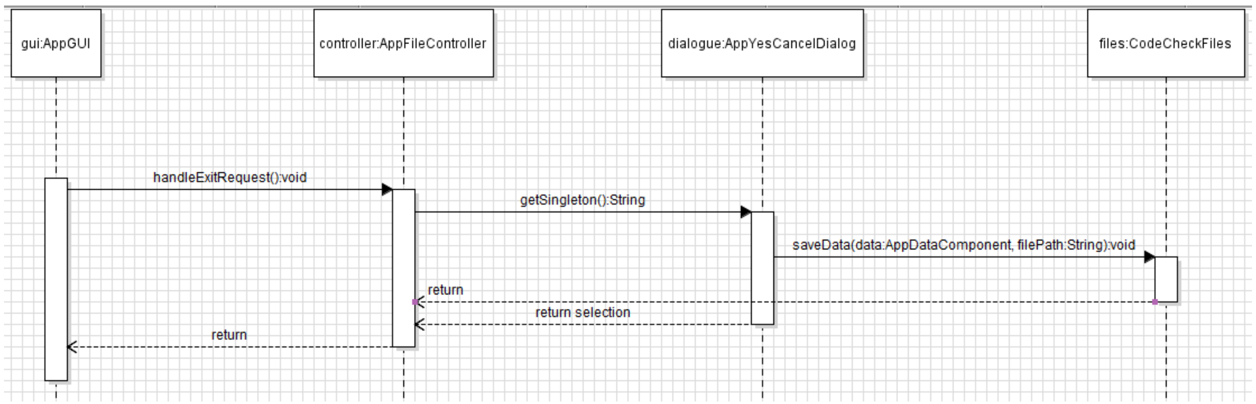

gui:AppGUI
controller:AppFileController
dialogue:AppYesCancelDialog
files:CodeCheckFiles
handleExitRequest():void
getSingleton():String
saveData(data:AppDataComponent, filePath:String):void
return
return selection
return

Figure 4.8: Exit Code Check UML Sequence Diagram

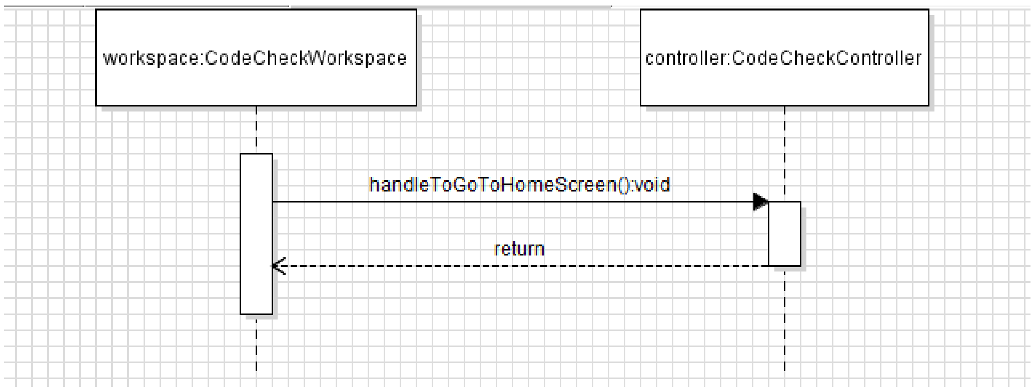

workspace:CodeCheckWorkspace
controller:CodeCheckController
handleToGoToHomeScreen():void
return

Figure 4.9: Navigate to Home Screen UML Sequence Diagram

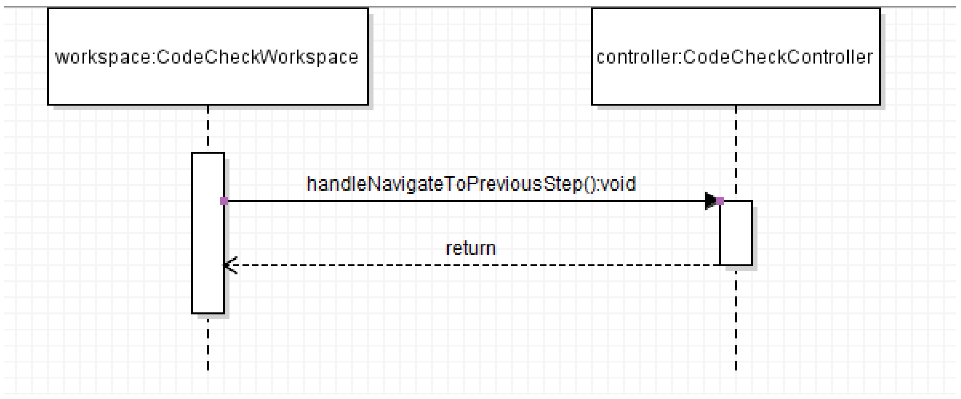

workspace:CodeCheckWorkspace
controller:CodeCheckController
handleNavigateToPreviousStep():void
return

Figure 4.10: Navigate to the previous step UML Sequence Diagram

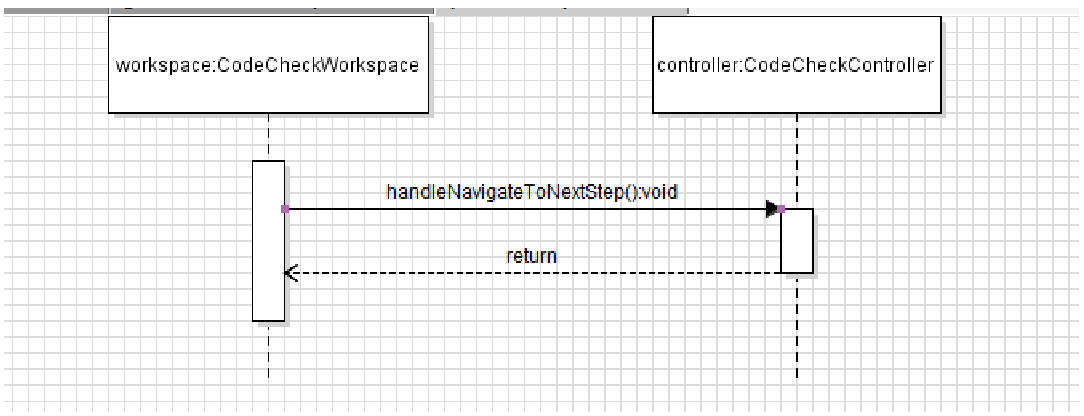

workspace:CodeCheckWorkspace
controller:CodeCheckController
handleNavigateToNextStep():void
return

Figure 4.11: Navigate to the next step UML Sequence Diagram

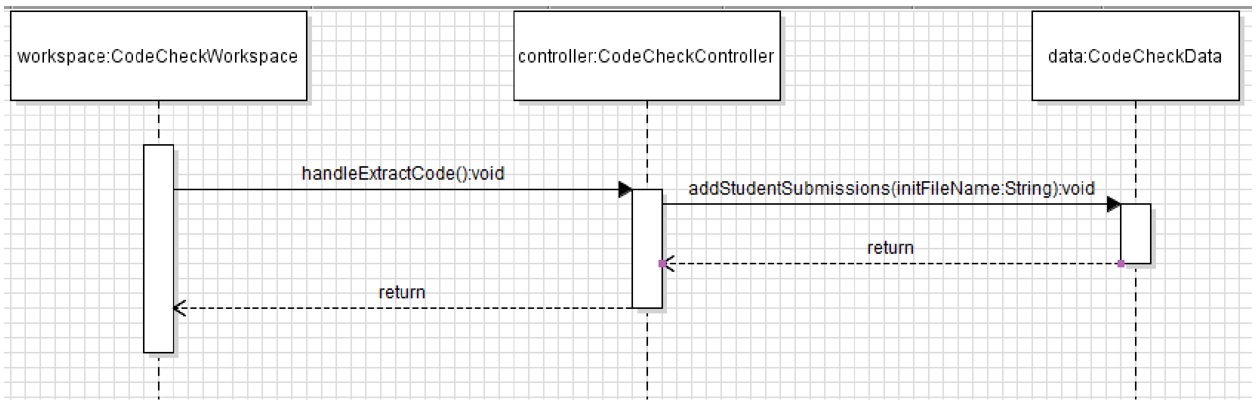

workspace:CodeCheckWorkspace
controller:CodeCheckController
data:CodeCheckData
handleExtractCode():void
addStudentSubmissions(initFileName:String):void
return
return

Figure 4.12: Extract Submissions UML Sequence Diagram

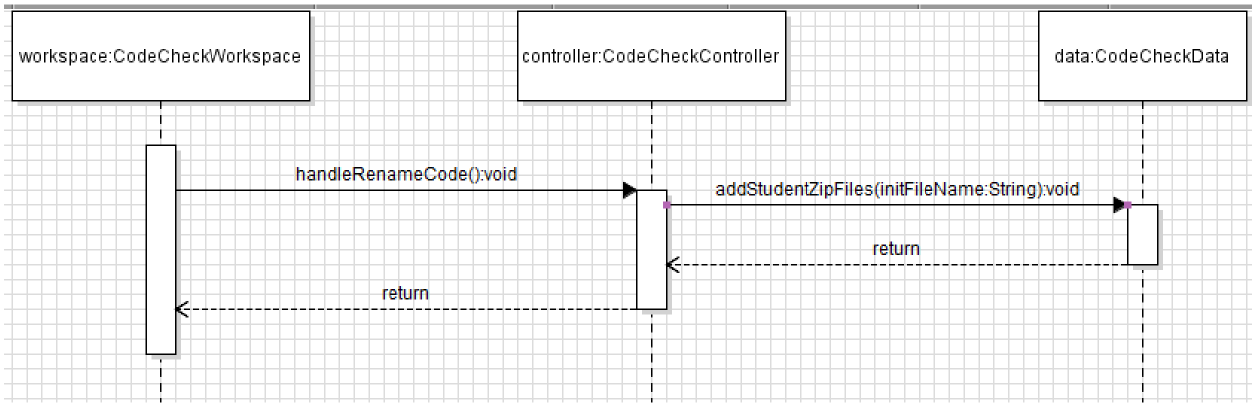

workspace:CodeCheckWorkspace
controller:CodeCheckController
data:CodeCheckData
handleRenameCode():void
addStudentZipFiles(initFileName:String):void
return
return

Figure 4.13: RenameStudent Submissions UML Sequence Diagram

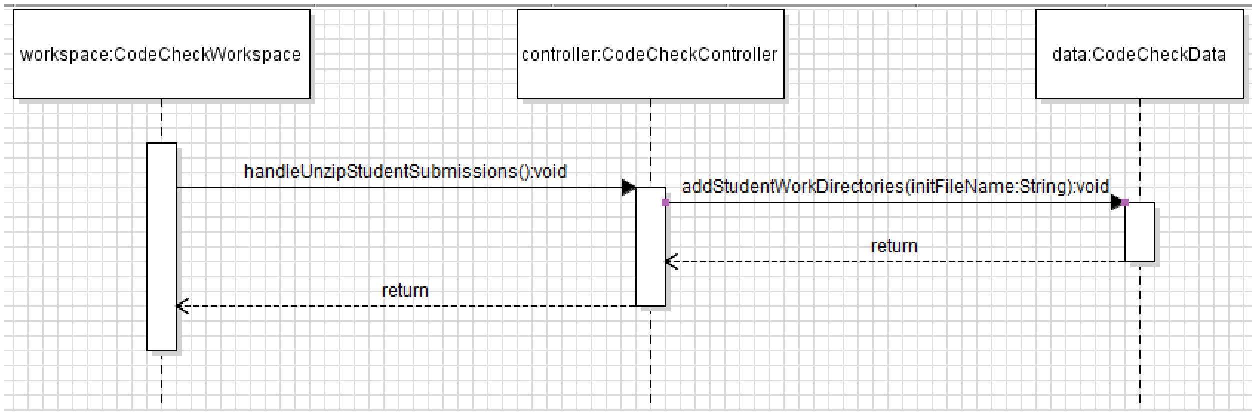

workspace:CodeCheckWorkspace
controller:CodeCheckController
data:CodeCheckData
handleUnzipStudentSubmissions():void
addStudentWorkDirectories(initFileName:String):void
return
return

Figure 4.14: Unzip Student Submissions UML Sequence Diagram

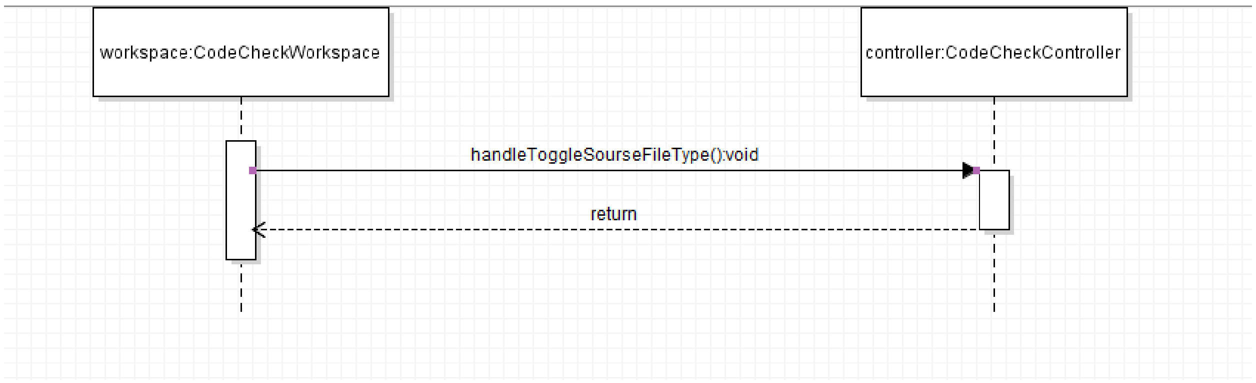

workspace:CodeCheckWorkspace
controller:CodeCheckController
handleToggleSourseFileType():void
return

Figure 4.15: Toggle Source File Type UML Sequence Diagram

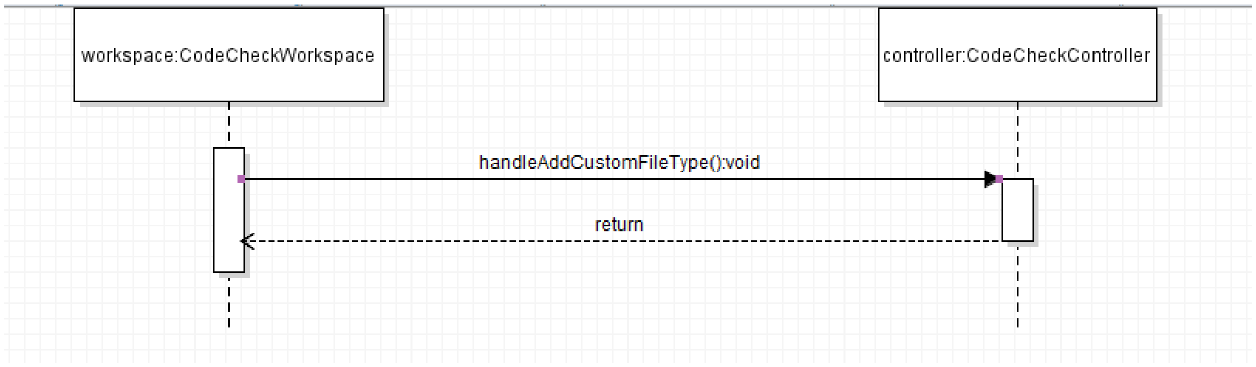

workspace:CodeCheckWorkspace
controller:CodeCheckController
handleAddCustomFileType():void
return

Figure 4.16: Add Custom File Type UML Sequence Diagram

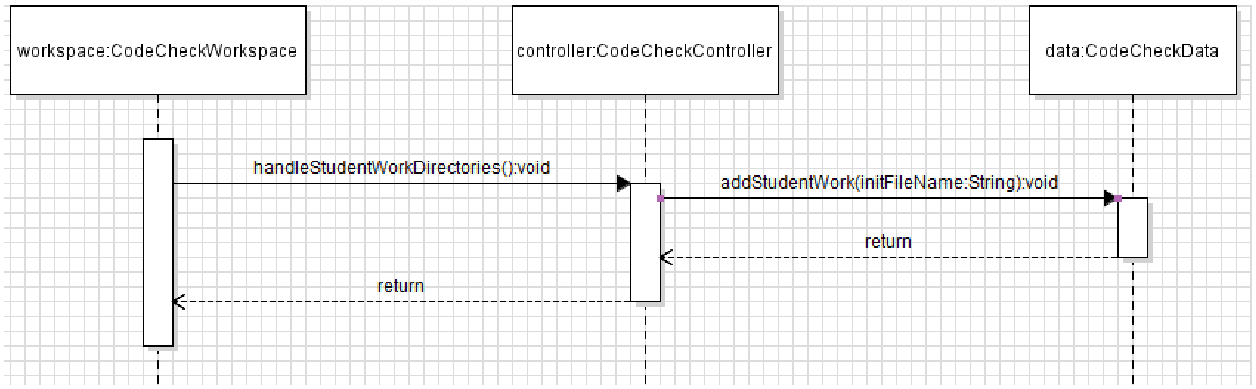

workspace:CodeCheckWorkspace
controller:CodeCheckController
data:CodeCheckData
handleStudentWorkDirectories():void
addStudentWork(initFileName:String):void
return
return

Figure 4.17: Extract Source Code UML Sequence Diagram

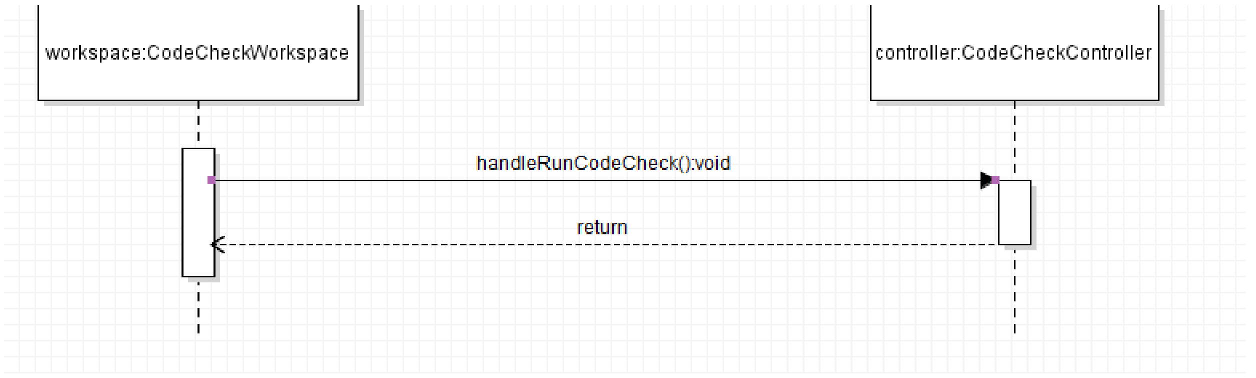

workspace:CodeCheckWorkspace
controller:CodeCheckController
handleRunCodeCheck():void
return

Figure 4.18: Run Code Check UML Sequence Diagram

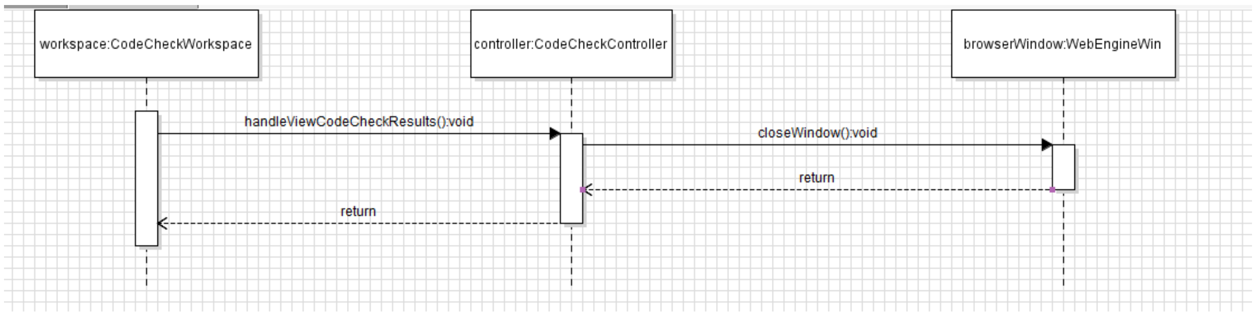

workspace:CodeCheckWorkspace
controller:CodeCheckController
browserWindow:WebEngineWin
handleViewCodeCheckResults():void
closeWindow():void
return
return

Figure 4.19: View Code Check Results UML Sequence Diagram

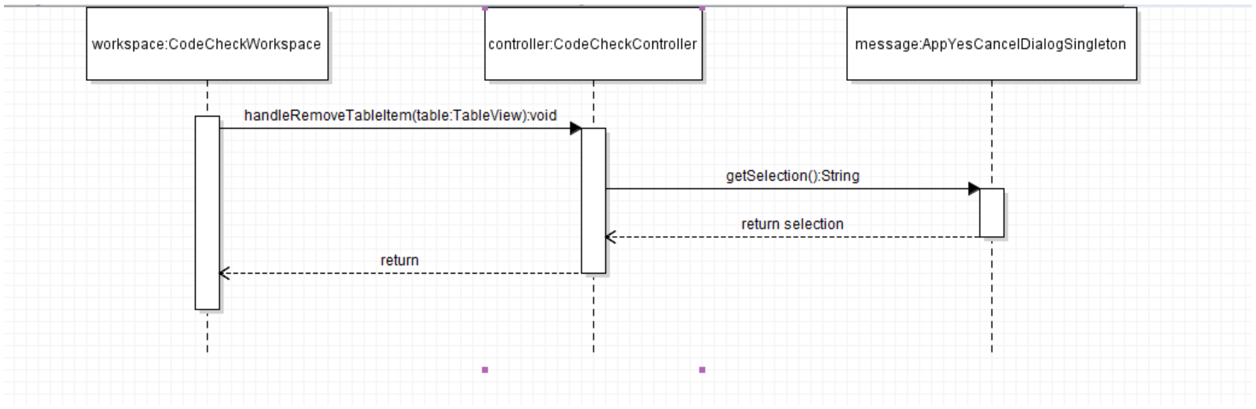

workspace:CodeCheckWorkspace
controller:CodeCheckController
message:AppYesCancelDialogSingleton
handleRemoveTableItem(table:TableView):void
getSelection():String
return selection
return

Figure 4.20: Remove Table Item UML Sequence Diagram

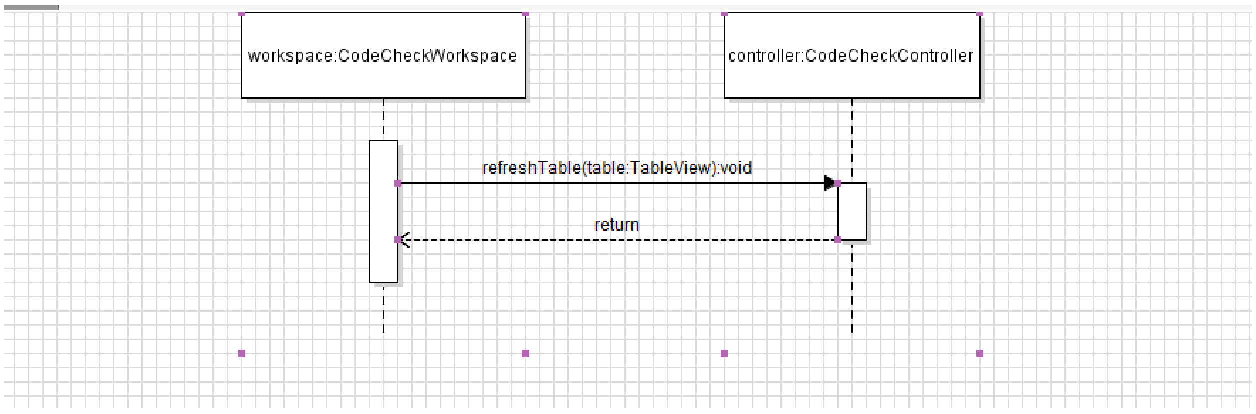

workspace:CodeCheckWorkspace
controller:CodeCheckController
refreshTable(table:TableView):void
return

Figure 4.21: Refresh TableView UML Sequence Diagram

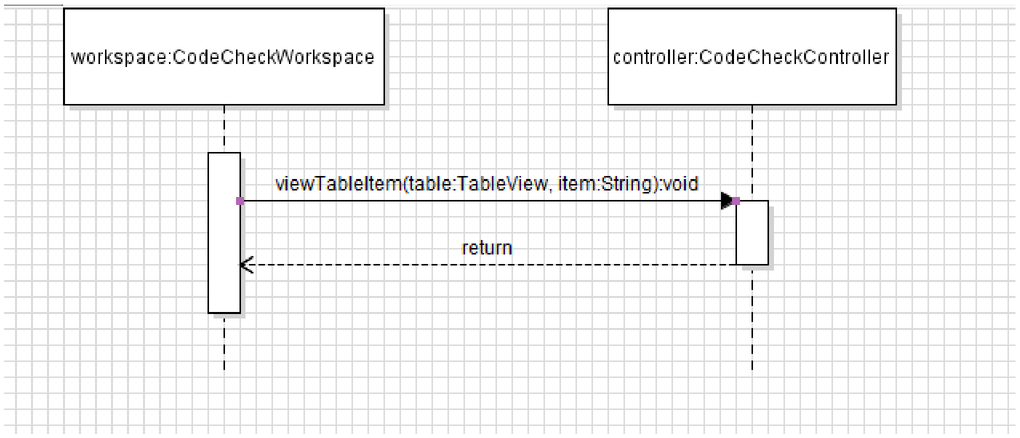
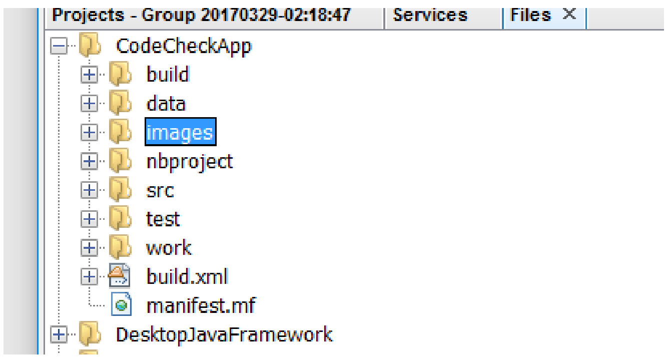

workspace:CodeCheckWorkspace
controller:CodeCheckController
viewTableItem(table:TableView, item:String):void
return

Figure 4.22: View Table Item UML Sequence Diagram

5 File Structure and Formats

Note that the DesktopJavaFramework and PropertiesManager will be provided inside DesktopJavaFramework.jar and PropertiesManager.jar, Java archive files that will encapsulate these frameworks. This should be imported into the necessary project for the CodeCheckApp application and will be included in the deployment of a single, executable JAR file titled CodeCheckApp.jar. 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, XML and schema files in *data* folder and JSON files and created CodeCheck folders in *work* directory respectively.



```
CodeCheckApp
    build
    data
    images
    nbproject
    src
    test
    work
    build.xml
    manifest.mf
DesktopJavaFramework
```

Figure 5.1: CodeCheckApp File Structure

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