

(CodEZ IDE)

(SOMA, Team 5)

(Stetson B, Owen N, Malcolm Z, Alexander S)

# **Software Design Specification & Project Delivery Report**

**Version: (2)**

**Date: (05/12/2025)**

## Table of Contents

1 Introduction.....	5
1.1 Goals and objectives.....	5
1.2 Statement of system scope.....	5
1.3 Reference Material.....	6
1.4 Definitions and Acronyms.....	6
2 Architectural design.....	7
2.1 System Architecture.....	7
2.2 Design Rational.....	11
3 Key Functionality design.....	12
3.1 Write Code.....	12
3.1.1 Write Code Use Cases.....	12
3.1.2 Processing sequence for Write Code.....	13
3.1.3 Structural Design for Write Code.....	14
3.1.4 Key Activities.....	14
3.1.5 Software Interface to other components.....	14
3.2 Run Code.....	15
3.2.1 Run Code Use Cases.....	15
3.2.2 Processing sequence for Run Code.....	17
3.2.3 Structural Design for Run Code.....	18
3.2.4 Key Activities.....	18
3.2.5 Software Interface to other components.....	18
3.3 Create Files.....	18
3.3.1 Create Files Use Cases.....	18
3.3.2 Processing sequence for Create Files.....	20
3.3.3 Structural Design for Create Files.....	21
3.3.4 Key Activities.....	21
3.3.5 Software Interface to other components.....	21
3.4 Save Files.....	21
3.4.1 Save Files Use Cases.....	21
3.4.2 Processing sequence for Save Files.....	23
3.4.3 Structural Design for Save Files.....	23
3.4.4 Key Activities.....	24
3.4.5 Software Interface to other components.....	24
4 User interface design.....	24
4.1 Interface design rules.....	24
4.2 Description of the user interface.....	24
4.2.1 CodEZ Page.....	24
5 Restrictions, limitations, and constraints.....	26

<b>6 Testing Issues (SLO #2.v).....</b>	<b>26</b>
6.1 Types of tests.....	26
6.2 List of Test Cases.....	26
<b>6.3 Test Coverage.....</b>	<b>27</b>
7 Appendices.....	27
7.1 Packaging and installation issues.....	27
7.2 User Manual.....	27
7.3 Open Issues.....	28
<b>7.4 Lessons Learned.....</b>	<b>28</b>
<b>7.4.1 Project Management &amp; Task Allocations (SLO #2.i).....</b>	<b>28</b>
<b>7.4.2 Implementation (SLO #2.iv).....</b>	<b>28</b>
7.4.3 Design Patterns.....	28
7.4.4 Team Communications.....	28
<b>7.4.4 Technologies Practiced (SLO #7).....</b>	<b>28</b>
7.4.5 Desirable Changes.....	29
7.4.6 Challenges Faced.....	29

# 1 Introduction

## 1.1 Goals and objectives

### CodEZ: An Accessible IDE for Beginner Programmers

Our application, CodEZ, is designed to make coding easy (EZ) for students of all technical backgrounds. We aim to provide an intuitive platform where beginners can learn any programming language without feeling overwhelmed.

#### Key Features & Goals:

- Minimalist, User-Friendly Interface
  - Goal: Prioritize core coding functions (e.g., editor, file management, console output) while progressively introducing advanced features.
  - Outcome: Reduces the initial learning curve, allowing new programmers to focus on fundamentals.
- Seamless Setup Experience
  - Goal: Enable a one-click (or near one-click) installation process, abstracting system configurations and dependencies.
  - Outcome: Beginners can start coding within minutes, lowering the barrier to entry.
- Clear Error Reporting & Debugging
  - Goal: Implement a simplified debugging panel that explains errors, their causes, and potential fixes in plain language.
  - Outcome: Empowers users to troubleshoot independently, building confidence and problem-solving skills.
- Lightweight Performance
  - Goal: Optimize the IDE to run smoothly on low-end hardware, such as budget student laptops.
  - Outcome: Ensures accessibility and reliability, even on less powerful devices.
- Expandability for Growth
  - Goal: Support modular additions (e.g., version control, advanced refactoring) as users advance.
  - Outcome: Provides a seamless transition from beginner to intermediate/advanced development within one tool.

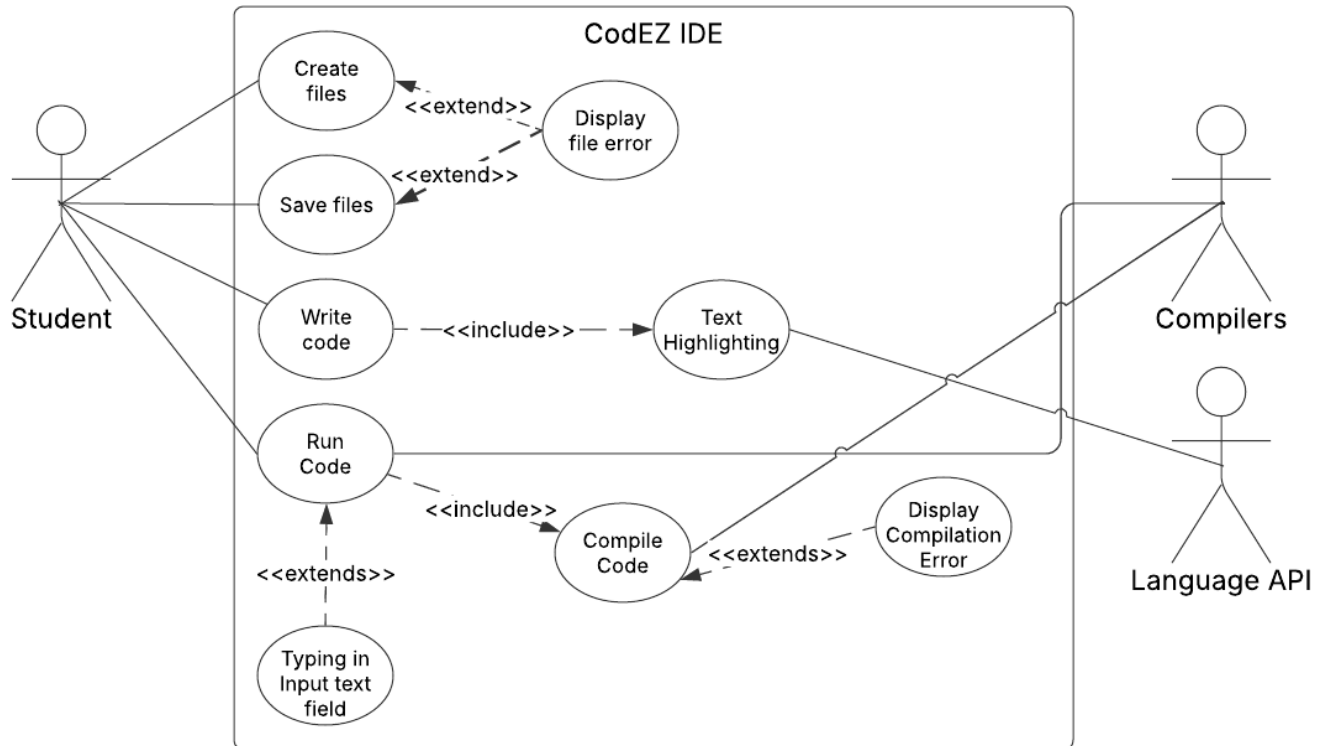
#### Security & Privacy:

User security and data privacy are foundational priorities in CodEZ's design.

## 1.2 Statement of system scope

The software will allow users to create and save Java files. Users can write code with the assistance of a language API that allows the user to understand the structure of their code and find errors more easily. When the user clicks the run button, the software will compile and execute the selected file using the selected program file path and the path to their installed compiler. And depending on if the program requires input, the user might need to type in the

input text field and send it to the executing process for it to finish. Both compilation and execution will be handled with minimal delay, and the output will be displayed within the software interface.



### 1.3 Reference Material

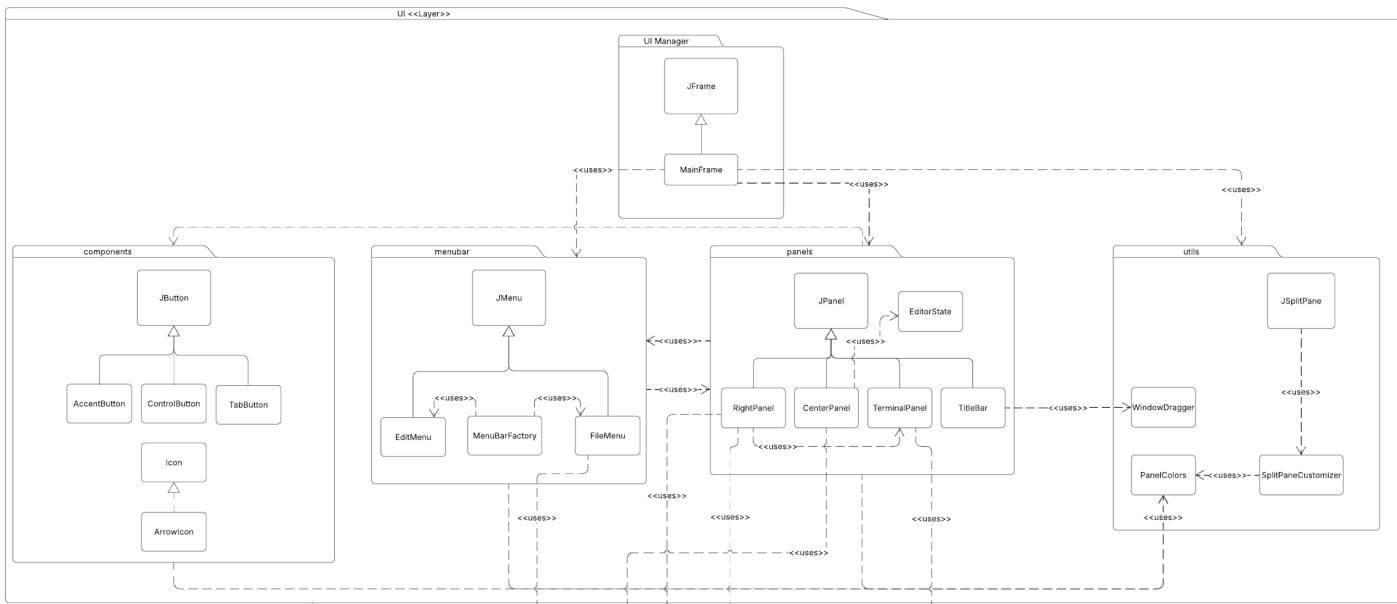
*n/a*

### 1.4 Definitions and Acronyms

IDE - Integrated Development Environment

## 2 Architectural design

### 2.1 System Architecture



The first package is a UI Layer package that is responsible for the visual aspect of our system, as well as the file system of our application, and uses the Windows Operating System to assist with file handling. It has 5 packages within, including:

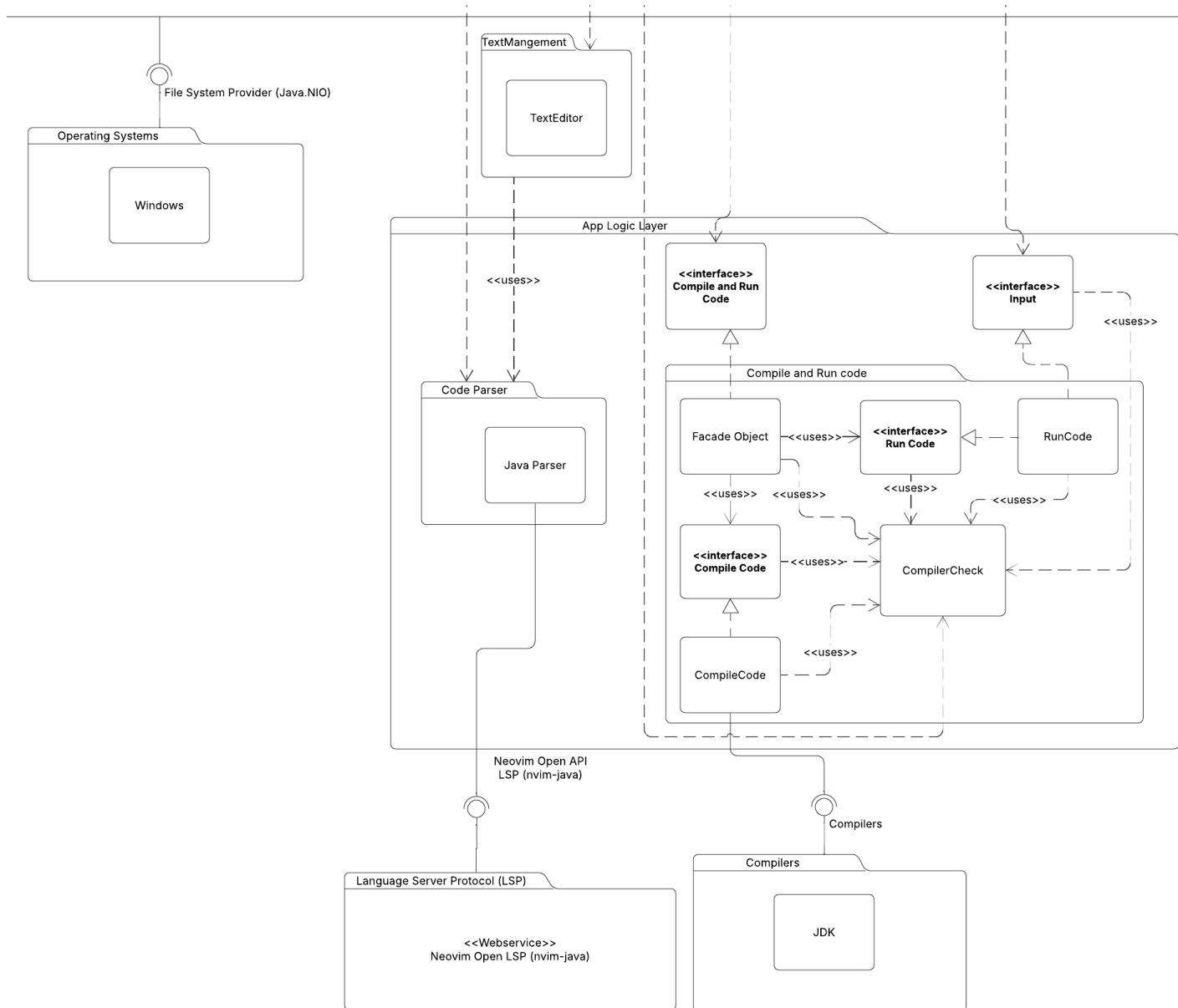
- UI Manager Package (uses the menubar, panels, and utils)
  - Module 1: JFrame
    - This is a predefined class in Java Swing. We're using it as the main window of our application. It acts like the outer frame or base where all other components (like panels, buttons, or text fields) get added and displayed to the user.
  - Module 2: MainFrame
    - Responsible for putting all of the features of the UI together.
- Menubar package (used by the UI manager & panels packages, and uses panels, utils, & the App Logic Layer)
  - Module 1: JMenu (parent class)
    - JMenu is a built-in class in Java Swing. We're using it to create the clickable menu options at the top of our app (like "File" or "Edit"). It helps group related actions so the user can find and choose what they want to do more easily.
  - Module 2: EditMenu (child class)
    - This is a class we created that extends JMenu. It holds menu items related to editing actions.
  - Module 3: FileMenu (child class)
    - This is another custom class that extends JMenu. It groups actions like "Open", "Save", or "Exit" under the "File" tab. This keeps the menu options neat and lets us control everything related to files in one spot in our code. This also reaches out to the Windows OS or more specifically

Java.NIO for file support. (external source call shown in the next image down below.)

- Module 4: MenuBarFactory (uses EditMenu & FileMenu)
  - is a helper class we made to build the top menu bar for the app. It creates a JMenu, adds the FileMenu and EditMenu, and then returns the whole thing so it can be shown in the main window. This keeps our setup clean and reusable.
- Panels Package (used by MainFrame & menubar, and uses menubar & utils & components & App Logic Layer)
  - Module 1: JPanel (parent class)
    - This is a predefined package in Java swing. We are using it as a visual and literal container to organize and group components, mainly buttons, in our UI.
  - Module 2: Right Panel (child class, uses TerminalPanel)
    - This panel sits on the right side of the app and communicates with the terminal area. It uses Terminal Panel to display the output area and lets users view results of their code after running it.
  - Module 3: Center Panel (child class, uses EditorState)
    - This is the main panel where users write and edit their code. It connects to EditorState, which helps manage the text content and tracks changes in the editor.
  - Module 4: Terminal Panel (child class, used by RightPanel)
    - This panel shows the terminal or output section. It prints messages, errors, or program output after code is run.
  - Module 5: TitleBar (child class, uses WindowDragger)
    - This is the custom title bar at the top of the window. It replaces the default OS window bar. It uses WindowDragger so users can click and drag the app window around manually.
  - Module 6: EditorState (used by Center Panel)
    - This class keeps track of the text and state of the code editor. It stores what the user types, helps update the editor when changes happen, and can be used later to support things like undo/redo (in the EditMenu module). The Center Panel uses it to manage what's shown and saved in the code area.
- Components Package (used by panels, and uses utils)
  - Module 1: JButton (parent class)
    - A built-in Swing class that creates clickable buttons. We're extending it to make custom buttons that fit the look and feel of our app.
  - Module 2: Accent Button (child class)
    - A styled button that stands out, for important actions, such as running and inputting. It's built from JButton but is green versus plain colors for the other buttons.
  - Module 3: Control Button (child class)
    - This is a custom button made from JButton, used mainly for window actions like close or minimize. It uses colors from the PanelColors utility.
  - Module 4: Tab Button (child class)



- A customized JButton used for switching tabs. It changes style when selected and uses PanelColors for consistent theming with the rest of the app.
- Module 5: Icon (swing interface)
  - An interface in Java Swing that defines methods for rendering a graphical icon, allowing custom icons to be created and used in various Swing components like buttons and labels.
- Module 6: ArrowIcon (implementing Icon)
  - A custom icon for drawing an arrow shape, implementing the Icon interface. It ensures the arrow is drawn with anti-aliasing for smoother edges and uses PanelColors for consistent theming, specifically for the arrow's color.
- Utils package (used by UI Manager & panels & menubar & components)
  - Module 1: JSplitPane (uses SplitPaneCustomizer)
    - A predefined Swing component used for creating a resizable, adjustable split view between two components. It's part of Java Swing and allows the user to dynamically adjust the size of the components within the split pane.
  - Module 2: SplitPaneCustomizer (uses PanelColors)
    - A helper class that customizes the style and appearance of JSplitPane, utilizing PanelColors for consistent theming across the application.
  - Module 3: PanelColors (used by panels, menubar, and components)
    - A class that defines the color scheme used throughout the UI, ensuring consistency in the appearance of panels, menus, and other components.
  - Module 4: MenuDragger
    - A utility class that enables draggable menus, allowing for enhanced user interaction with the menu bar by enabling dynamic positioning.



The next main package is the App Logic Layer package that is responsible for giving our application its functionality. This links with the previous UI Layer package through its panels. Additionally, there is a TextManagement package that assists the Code Parser package in App Logic Layer, with text highlighting in the CenterPanel. And the four main packages here are:

- Compile and Run code Package (implements a Compile and Run Code interface and Input interface, and \*sidenote\*, unfortunately everything related to this package needs reference to CompilerCheck due to the enum we created for the various programming languages supported. This has resulted in many dependencies, so a simple string would've solved this.)
  - Module 1: Facade Object (Implements Compile and Run Code interface)
    - Used to simplify interface and hide complexity. Implements the interface which is used by the RightPanel module from the UI Layer package.

- Module 2: <<Interface >> Compile Code (used by the FacadeObject, implemented in CompileCode)
  - Used as a contract of what CompileCode needs to implement and for loose coupling so not all interfaces are under one module.
- Module 3: <<Interface>> Run Code (used by the FacadeObject, implemented in RunCode)
  - Used as a contract of what RunCode needs to implement and for loose coupling so not all interfaces are under one module.
- Module 4: CompileCode (implements the Compile Code Interface, uses Compiler Check, and uses external sources – JDK compiler)
  - Compiles the code after checking if the user has a compiler, and if so, then it will use the compiler to compile the user's program file. If there was an error, it displays the error in the terminalPanel.
- Module 5: CompilerCheck (used by CompileCode)
  - Checks whether the user has a compiler or not, if so, it will store the path to the compiler.
- Module 6: RunCode (implements the Run Code Interface and Input Interface)
  - Runs the executable file and displays the results in the terminalPanel. Handles input by sending it to the running process.
- Code Parser Package (used by textEditor from TextManagement Package)
  - Module 1: Java Parser uses Neovim Open API LSP (nvim-java)
    - Parses Java code, using nvim-java, based on the structure defined in the Parser interface to enable syntax checking and highlighting.
- Text Management Package (used by CenterPanel in UI layer and uses CodeParser package)
  - Module 1: Text Editor
    - This TextEditor class is responsible for managing a Java Swing-based text editing component that integrates with a language server to support real-time semantic highlighting using a debounce mechanism and temporary file syncing.

## 2.2 Design Rational

### Rationale for Selecting Layered Architecture:

The system adopts a Layered Architecture due to its clear separation of concerns and modular structure, which aligns well with our IDE's requirements. Each layer interacts only with adjacent layers, simplifying development and maintenance. For instance:

- The User Interface (UI) layer depends on underlying subsystems in the app logic layer, but also handles the visuals and file system.
- These subsystems delegate functionality to core components like the Code Parser, compilation and run logic, and text management.
- The bottom layer integrates external tools (e.g., language compilers, OS, APIs).

This approach ensures maintainability and a logical flow of dependencies while avoiding unnecessary complexity.

### Alternative Considered: Client-Server Architecture

We evaluated Client-Server Architecture for its scalability and centralized control. However, the added overhead of network communication, synchronization, and deployment complexity made it impractical for our project's timeline and scope. Given that our IDE operates as a standalone application, the benefits of client-server did not outweigh its development costs.

## 3 Key Functionality design

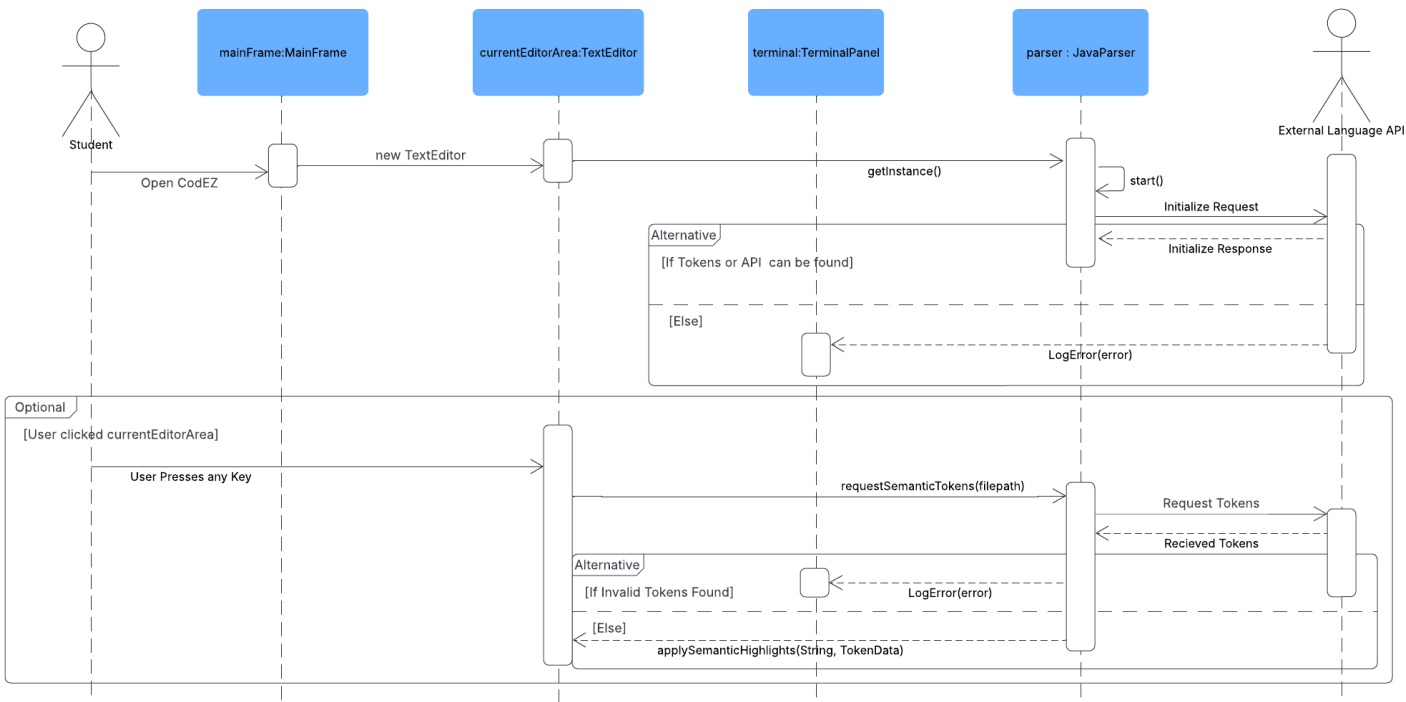
### 3.1 Write Code

#### 3.1.1 Write Code Use Cases

Use Case #1:	Write Code
Goal in Context	A student uses their keyboard to write a program in CodeEZ
Scope	CodeEZ IDE
Level	Primary Task
Primary Actor	Student
Preconditions	- CodeEZ IDE is opened - An active file is opened - User has selected the Text Editor window
Minimal Guarantee	Student will be able to type into the CodeEZ IDE
Success Guarantee	Student's code will be processed and highlighted (classes, enums, structs,

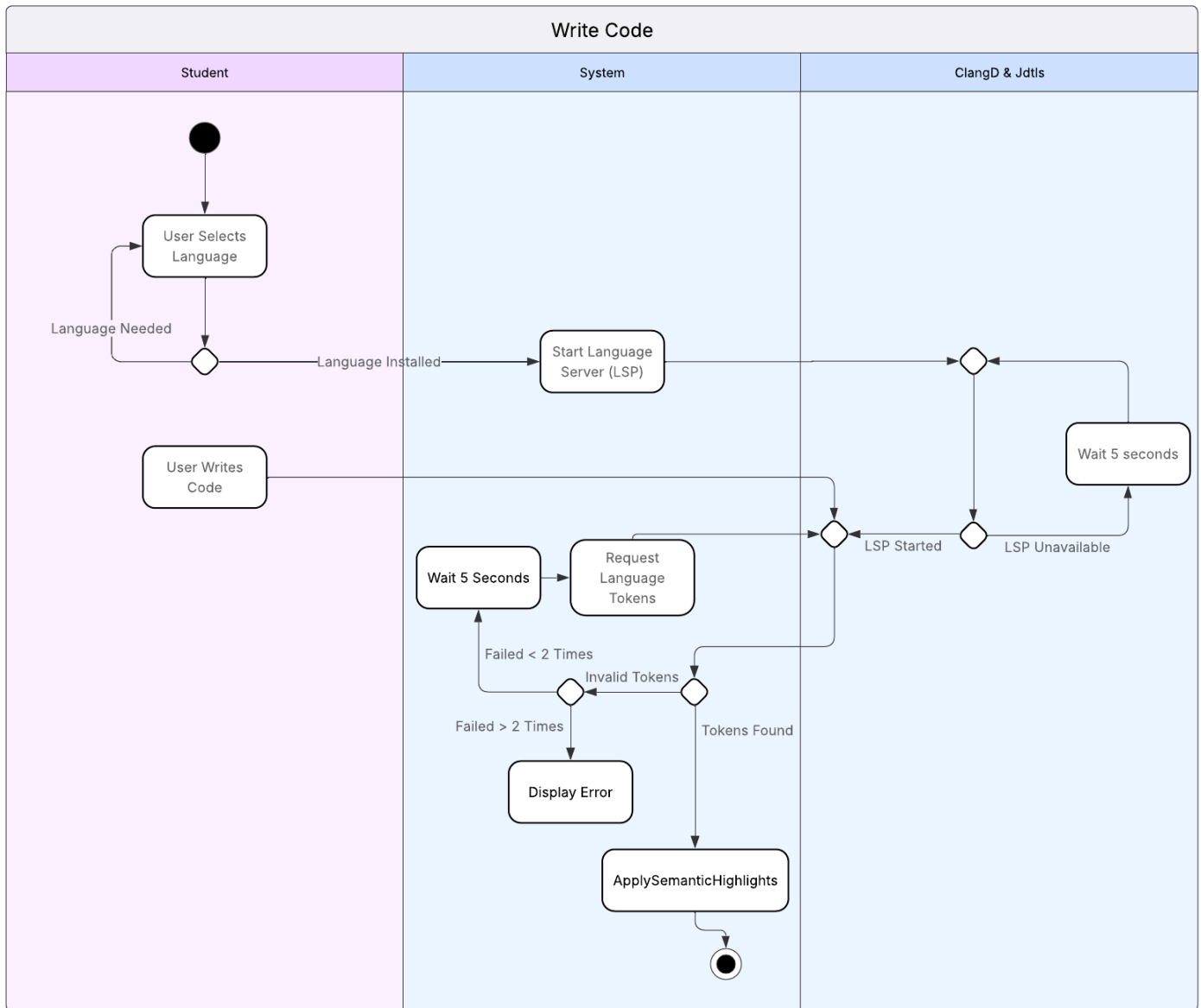
	variables, types, etc.)
<b>Trigger</b>	User clicks inside the Text Editor window and begins typing
<b>Step</b>	<b>Action</b>
<b>1</b>	User begins typing in the Text Editor
<b>2</b>	Program processes text with Language API's
<b>3</b>	Program takes extracted tokens and forms an Abstract Syntax Tree (AST)
<b>4</b>	Program uses AST to recolor class, enum, struct, variable, and type names
<b>Extension Step</b>	<b>Branching Action</b>
<b>2a: Language API is unreachable</b>	<b>a1:</b> Program issues error but allows user to continue programming without highlighting
<b>3a: AST cannot be created due to lack of resources</b>	<b>a1:</b> Program halts loading the AST <b>a2:</b> Program attempts to reload when resources become available

## 3.1.2 Processing sequence for Write Code





### 3.1.4 Key Activities



### 3.1.5 Software Interface to other components

External Components: Neovim Open API LSP(nvim-java) for the Java Parser.

## 3.2 Run Code

### 3.2.1 Run Code Use Cases

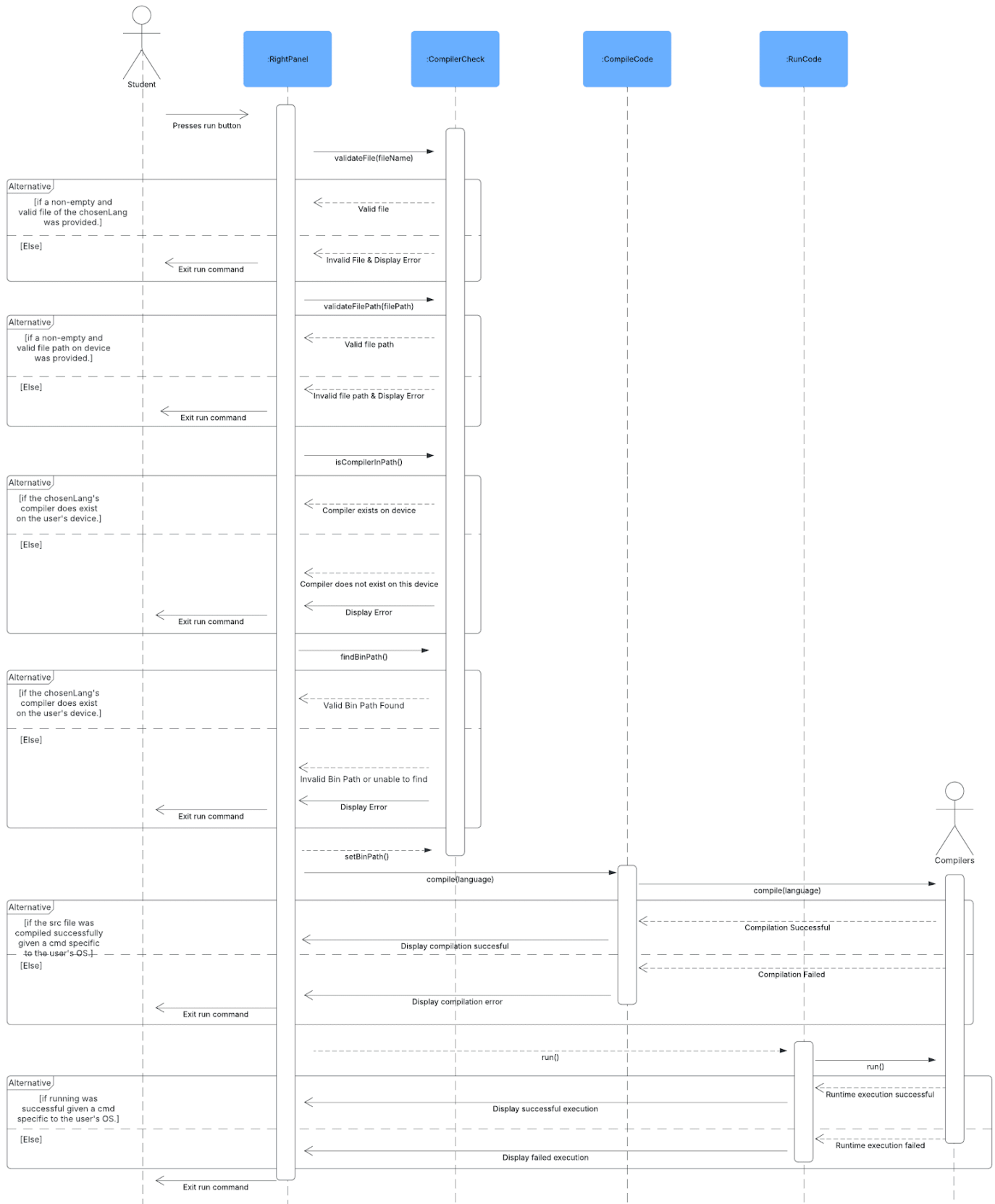
<b>Use Case #2:</b>	<b>Run code</b>
<b>Category</b>	<b>Details</b>
<b>Goal in Context</b>	A beginner student can run code.



<b>Scope</b>	CodEZ IDE
<b>Level</b>	Primary task
<b>Primary Actor</b>	Beginner student; novice programmer
<b>Preconditions</b>	- CodEZ must be running - A file must be opened
<b>Minimal Guarantee</b>	Error messages will be displayed.
<b>Success Guarantee</b>	Code is run successfully and displayed.
<b>Trigger</b>	Run button pressed by student.
<b>Success Scenario</b>	<b>Action Step</b>
<b>Step</b>	<b>Action</b>
1	The system checks if a file path and file were provided.
2	The system checks if the source file exists.
3	The system checks if the chosen language's compiler exists.
4	The system finds and stores the student's chosen language's compiler file path.
5	The system compiles the code given the user's operating system.
6	The system runs the code given the user's operating system.
7	The system captures the output and sends it to the UI.
<b>Extensions</b>	
<b>Extension Step</b>	<b>Branching Action</b>
<b>1a: User does not provide a file path or file name was not specified</b>	<b>a1:</b> Error message displayed on the UI; compilation and execution do not

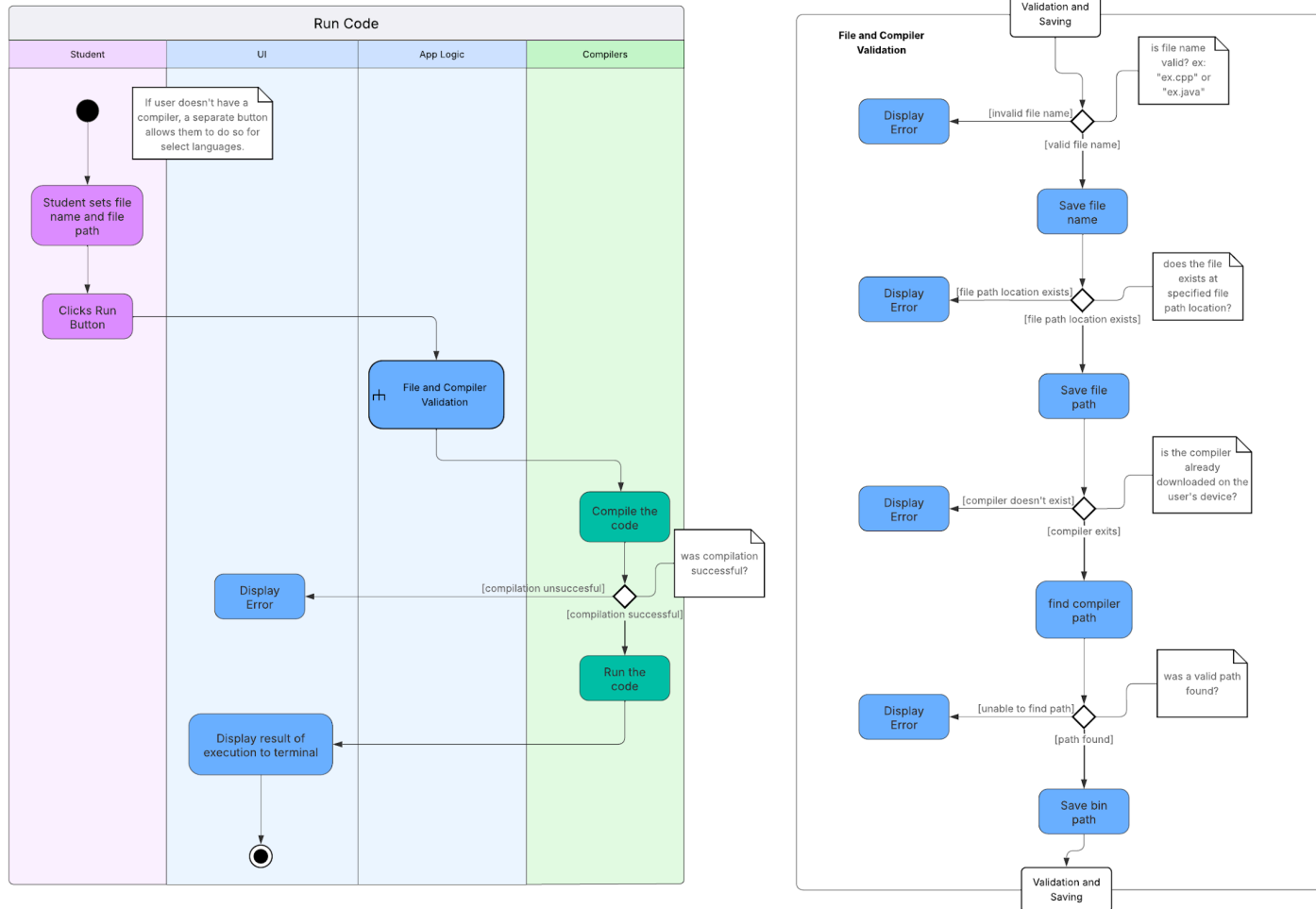
	proceed.
<b>2a: File doesn't exist in the user's system</b>	<b>a1:</b> Error message displayed on the UI; compilation and execution do not proceed.
<b>3a: Compiler does not exist on user's machine</b>	<b>a1:</b> Error message displayed on the UI; compilation and execution do not proceed.
<b>4a: Compiler file path not found in user's system</b>	<b>a1:</b> Error message displayed on the UI; compilation and execution do not proceed.
<b>4b: Compiler file path fails while system parses</b>	<b>b1:</b> Error message displayed on the UI; compilation and execution do not proceed.
<b>5a: User-provided code fails to compile</b>	<b>a1:</b> Error message displayed on the UI; execution does not proceed.
<b>6a: Runtime error occurs</b>	<b>a1:</b> Halt process until available; fail gracefully.

## 3.2.2 Processing sequence for Run Code





### 3.2.4 Key Activities



### 3.2.5 Software Interface to other components

External Components: JDK compiler for Java code.

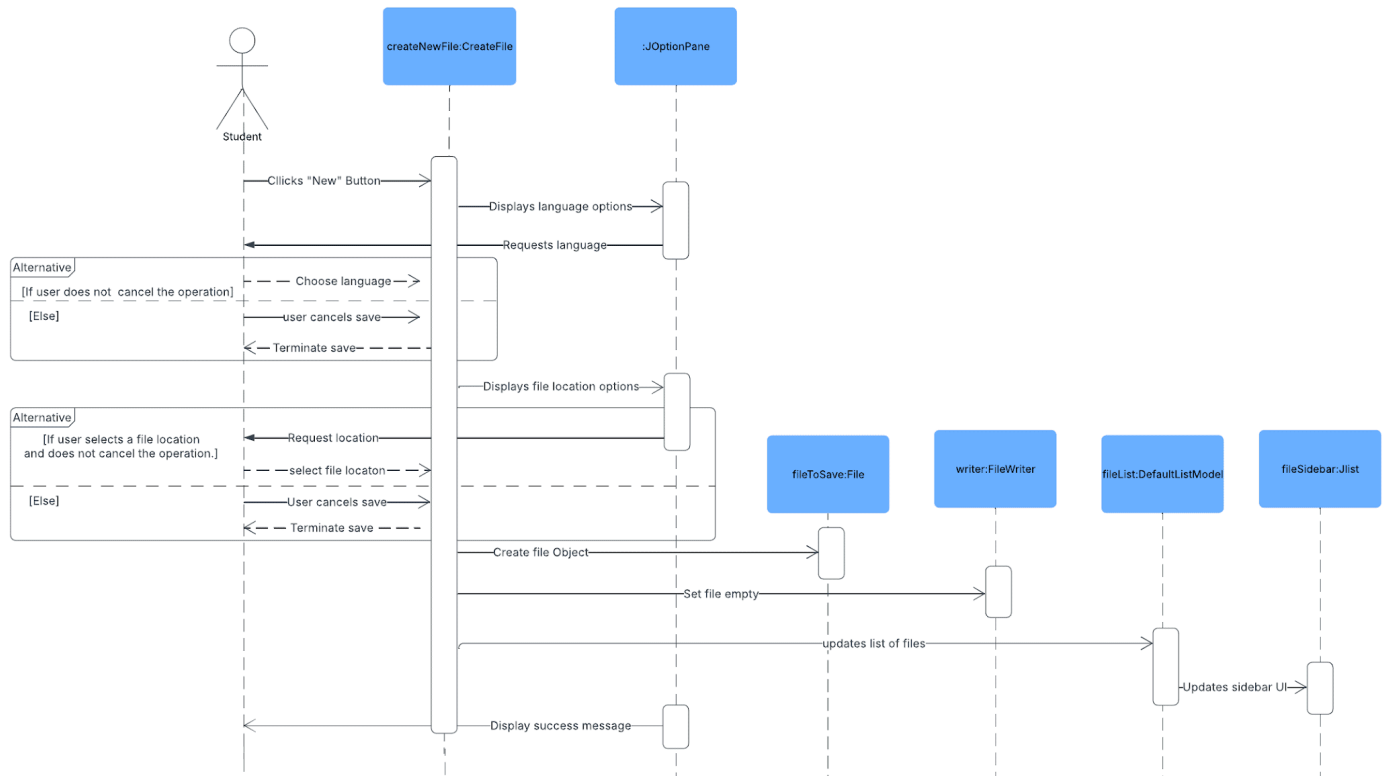
## 3.3 Create Files

### 3.3.1 Create Files Use Cases

<b>Use Case #3:</b>	<b>Create Files</b>
<b>Goal in Context</b>	A student uses their keyboard to write a program in CodeZ
<b>Scope</b>	CodeZ IDE
<b>Level</b>	Primary Task
<b>Primary Actor</b>	Student

<b>Preconditions</b>	The CodEZ IDE is open
<b>Minimal Guarantee</b>	Error case where file is not created, and an error message would display
<b>Success Guarantee</b>	File will be created and attached under the CodEZ IDE folder
<b>Trigger</b>	User selects "New" under the "File" dropdown
<b>Step</b>	<b>Action</b>
1	The system listens to the "New" button
2	User will then select file language
3	The system creates a default file name
4	The user will select the file destination
5	A new blank file is created
6	The new file is added to the UI sidebar
7	A successful message is displayed
<b>Extension Step</b>	<b>Branching Action</b>
<b>2a: Cancel option</b>	<b>a1:</b> User chooses cancel button, create file is terminated
<b>4a: Error Exception</b>	<b>a1:</b> Error choosing directory message displayed, create file terminated
<b>6a: Error Exception</b>	<b>a1:</b> Error creating the file message displayed, create file terminated

## 3.3.2 Processing sequence for Create Files

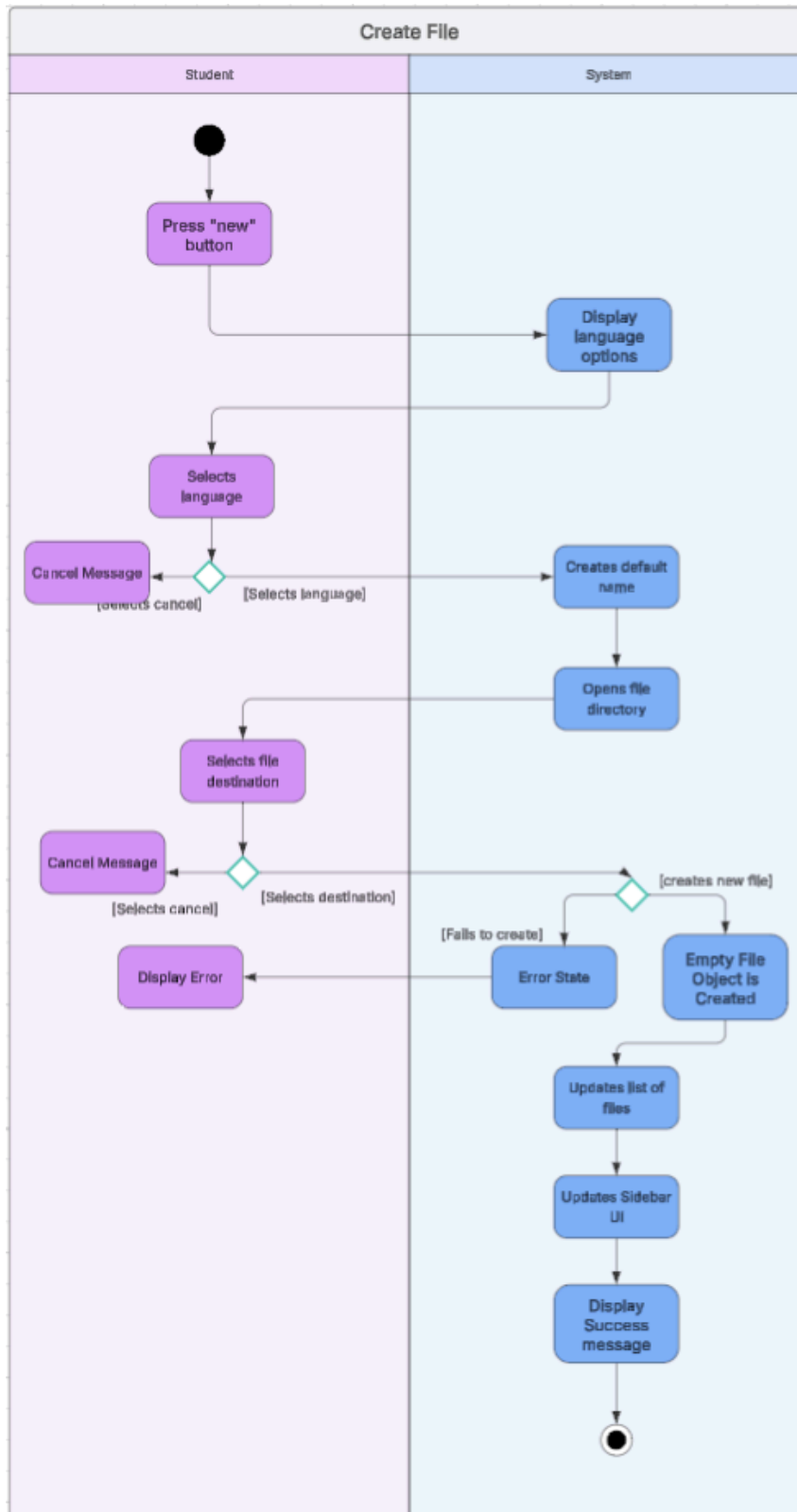


## 3.3.3 Structural Design for Create Files





## 3.3.4 Key Activities



### 3.3.5 Software Interface to other components

External Components: Windows OS.

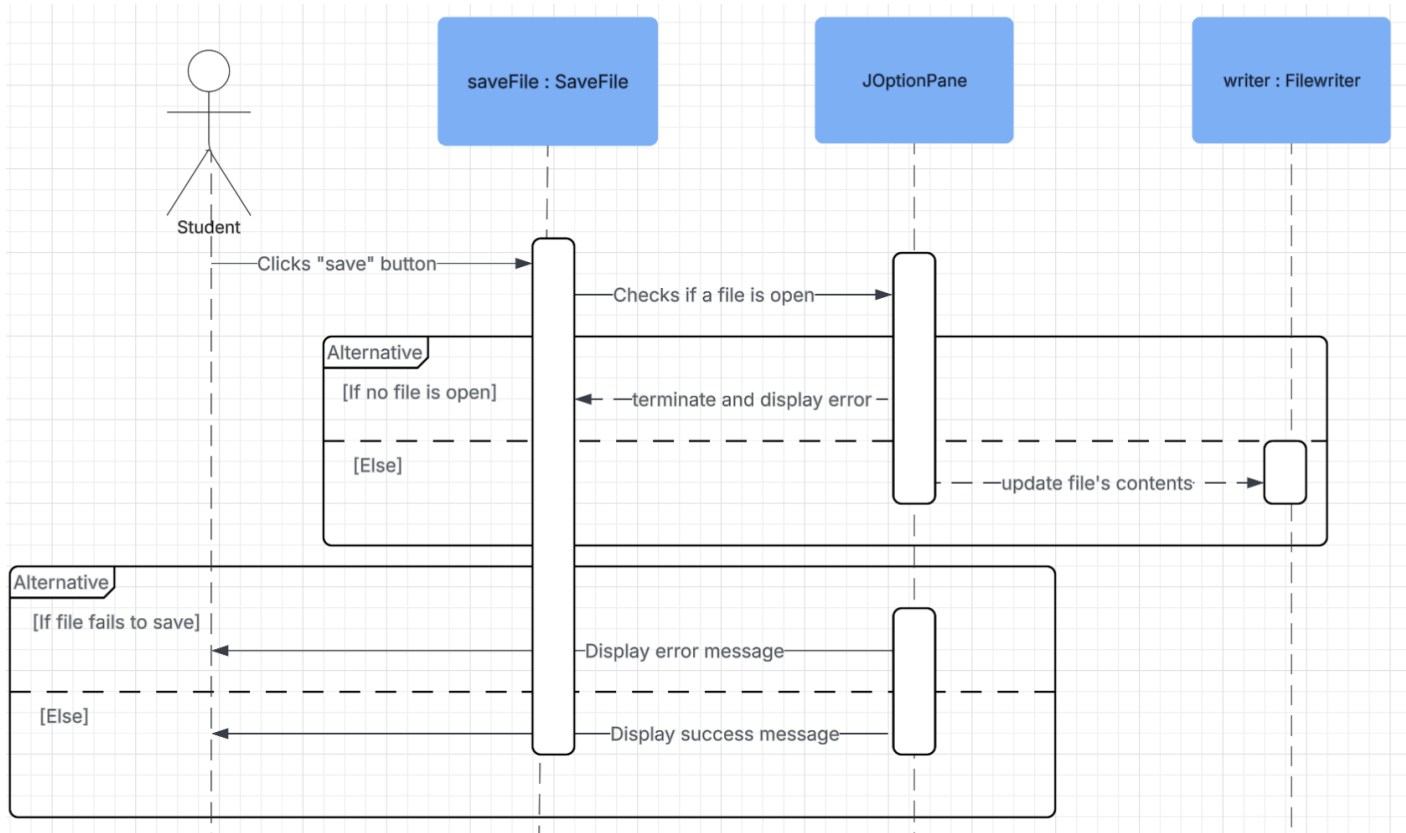
## 3.4 Save Files

### 3.4.1 Save Files Use Cases

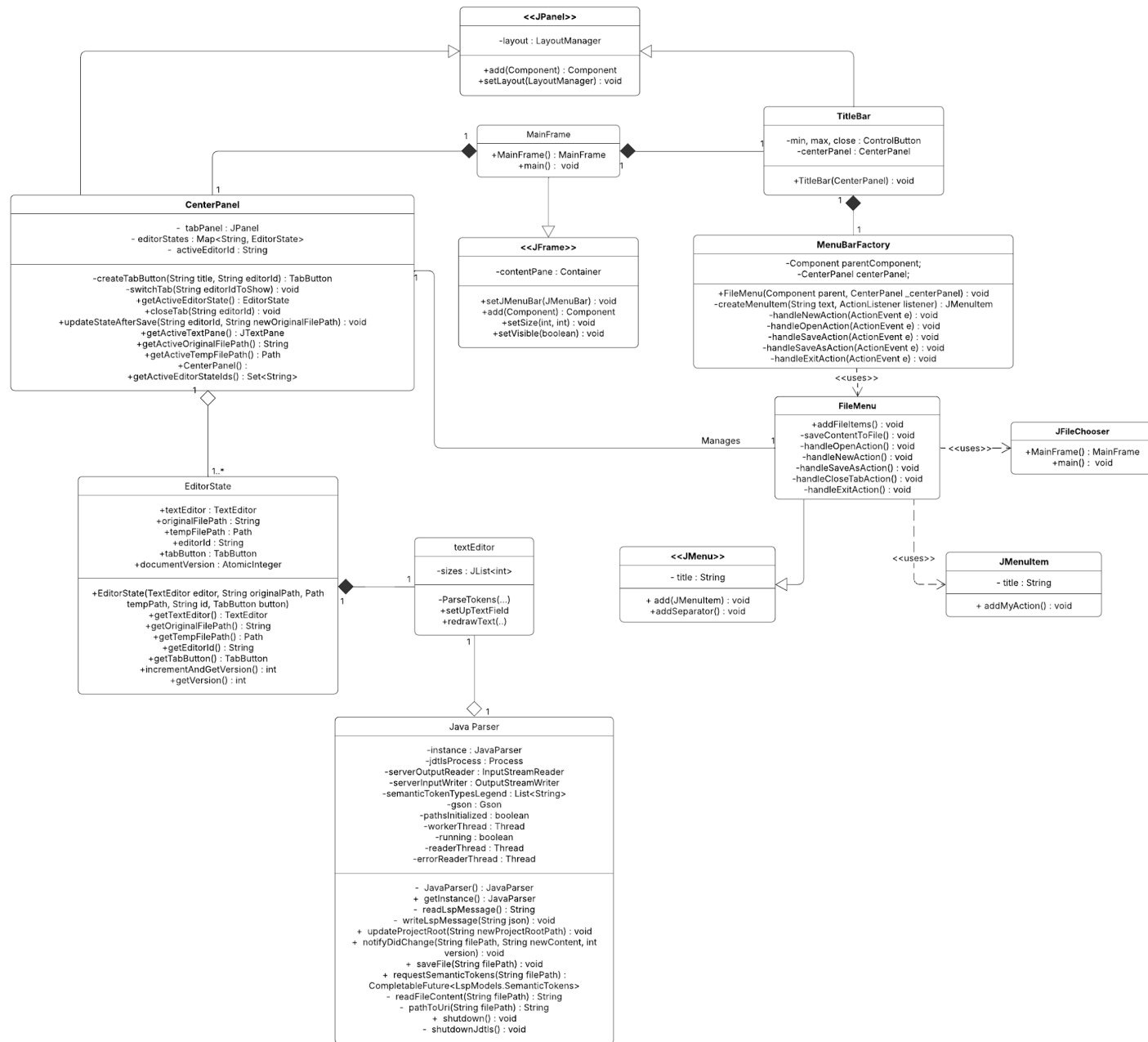
<b>Use Case #4:</b>	<b>Save Files</b>
<b>Goal in Context</b>	A student can select the save button and it will update the file in their directory
<b>Scope</b>	CodEZ IDE
<b>Level</b>	Primary Task
<b>Primary Actor</b>	Student
<b>Preconditions</b>	CodEZ IDE is open and a file is open
<b>Minimal Guarantee</b>	Error case where file is not created, and an error message would display
<b>Success Guarantee</b>	File is updated and saved into your directory
<b>Trigger</b>	User selects "Save" under the "File" dropdown
<b>Step</b>	<b>Action</b>
<b>1</b>	The system listens to the "Save" button
<b>2</b>	The system check if there is currently a file open
<b>3</b>	The file in your directory is overwritten with the code currently in the text editor
<b>4</b>	The UI displays a success message is displayed
<b>Extension Step</b>	<b>Branching Action</b>

<b>2a: System check</b>	<b>a1:</b> Error no file is open message, save file is terminated
<b>3a: Error Exception</b>	<b>a1:</b> Error saving the file message displayed, save file terminated

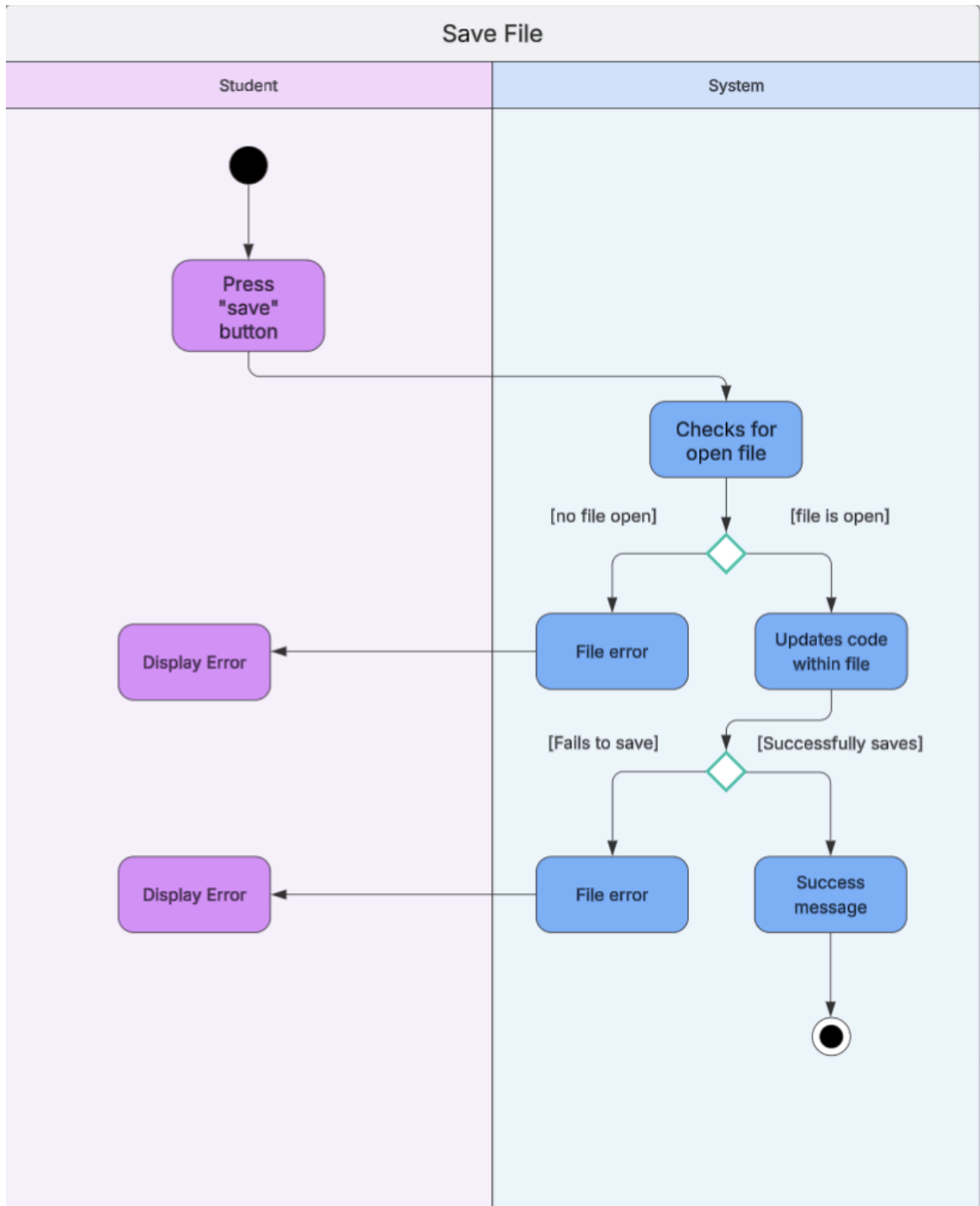
## 3.4.2 Processing sequence for Save Files



## 3.4.3 Structural Design for Save Files



## 3.4.4 Key Activities



### 3.4.5 Software Interface to other components

External Components: Windows OS.

## 4 User interface design

### 4.1 Interface design rules

- Use of 4 key colors: white, green, gray, and black.
- Primary colors used for tabs and panels text are white
- All borders and Buttons are green
- All remaining background spaces are gray
- All panels must be visible; Right Panel (Compiler Settings), Center Panel, and Terminal

### 4.2 Description of the user interface

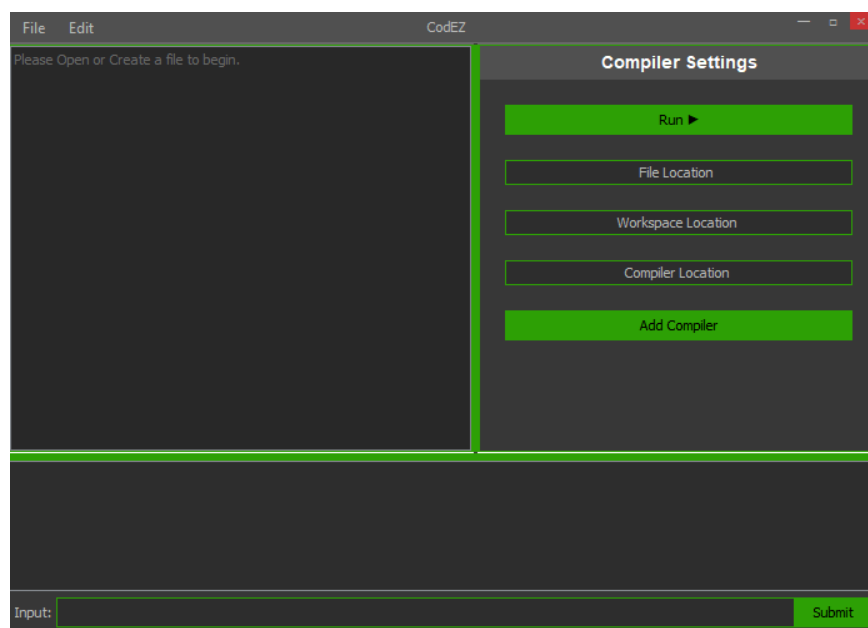
- The user interface is a visual front-end
- Will allow users to write in a text editor
- Will allow users to manually populate file fields
- Will allow users to access file settings dropdown menu which alter the apps state
- Will allow users to adjust tabs/panels to allow for a comfortable coding environment
- Will allow users to manually input and submit strings in the terminal

#### 4.2.1 CodeEZ Page

- The CodeEZ page will allow the user to open new and existing files, write and run code, view and type in the terminal

##### 4.2.1.1 Screen Images

Shown below is the UI for the CodeEZ Application



#### 4.2.1.2 Objects and Actions

- Menu on the top of the page that contains our “File”, “Edit”, “View”, and “Window” tabs
- “File” dropdown that contains the buttons “New”, “Save”, and “Open”
- “New” button will allow the user to create a new file upon selection, where the user will provide a file name
- “Save” button will allow the user to save a file currently in use in the CodEZ Application, on an initial save the user will be instructed to specify a folder location for the file to be saved
- “Open” button will allow the user to open a file located on their system
- “Compiler Settings” panel that will hold the “Run” button that will allow for the execution of code, and a “File Location” and “Workspace Location” that will contain the file path to the specified file and the folder location as well.
- “Terminal” panel that will output the result from the executed code, either ran successfully or with errors, will output respectively

## 5 Restrictions, limitations, and constraints

- Java only supported (only specify .java files in the run path)
- Must download specified language compiler for full use (Jdk 11).
- Windows system only supported, unable to support Mac/LinuxOS.
- Add Compilers button currently does not have any behaviour

## 6 Testing Issues (SLO #2.v)

### 6.1 Types of tests

You may consider the following types of tests:

- (1). **Performance Test** – for example, to ensure that the response time for information retrieval is within an acceptable range. You typically should provide a specific performance bounds. For example, the search process should not take longer than **30 seconds**.
- (2). **Accuracy Test** – for example, to determine if queries return the expected results.
- (3). **User Interface Test** – for example, to make sure the user interface is clear and easy to use with all types of users. Unfamiliar user can use the interface with minimal instruction and achieve the desired results.
- (4). **Security test** – for example, to ensure that users can only perform the tasks specified for their user group
- (5). **Repeatability Test** – for example, the software returns the same result for repeated queries.



## 6.2 List of Test Cases

1)

Test Type	<b>Functional Test</b>
Testing range	Run Code feature
Testing Input	<ol style="list-style-type: none"> <li>1) User's desired file path with wrong file name</li> <li>2) User's incorrect file path with desired file name</li> </ol>
Testing procedure	<ol style="list-style-type: none"> <li>1) Enter file path with wrong file name into file location text box. Click the Run button</li> <li>2) Enter incorrect file path with file name into file location text box. Click the Run button</li> </ol>
Expected Test Result	<ol style="list-style-type: none"> <li>1) Prompt: "error: file not found: [wrong file name]"</li> <li>2) Prompt: "Error: The file [desired file name] does not exist at path: [incorrect file path]. Error: invalid file."</li> </ol>
Testers	Malcolm Zartman
Test result	Passed

2)

Test Type	<b>Functional Test</b>
Testing range	Compiler Check feature (Windows vs MacOS user)
Testing Input	<ol style="list-style-type: none"> <li>1) Windows user</li> <li>2) Mac user</li> </ol>
Testing procedure	<ol style="list-style-type: none"> <li>1) Click the Run button on a Windows system.</li> <li>2) Click the Run button on a Mac system.</li> </ol>
Expected Test Result	<ol style="list-style-type: none"> <li>1) Prompt: "Your Java bin path was found at: C:/Program Files/Java/jdk-23.0.2/bin"</li> <li>2) Prompt: "Error: couldn't find or parse the bin path for Java."</li> </ol>
Testers	Malcolm Zartman & Owen Newberg
Test result	Passed (We know we can't support them yet)

3)

Test Type	<b>Functional Test</b>
Testing range	Compile Code feature (compilation error handling)
Testing Input	Syntax error in desired program file
Testing procedure	<p>Enter file path with file name of file with syntax error into the file location text box.</p> <p>Click the Run button.</p>
Expected Test Result	<p>Prompt:</p> <p>"</p> <p>... [JDK compilation error handling displayed]...</p>

	Compilation finished with exit code: 1 Error: compilation error"
Testers	Malcolm Zartman
Test result	Passed

4)

Test Type	<b>Functional Test</b>
Testing range	Run code's input feature (runtime error handling)
Testing Input	Incorrect data type (String when expecting an integer)
Testing procedure	Create a program file that is expecting an integer for input. Enter in the file path with file name in the file location text box. Click the Run button. Input a String into the input text box and press submit.
Expected Test Result	Prompt: "Program exited with code: 1"
Testers	Malcolm Zartman
Test result	Passed

5)

Test Type	<b>Functional Test</b>
Testing range	Open File feature
Testing Input	Opening a invalid file
Testing procedure	Start the program click Open and then directory is opened. Double click on a invalid file such as a file containing music or another invalid file.
Expected Test Result	Prompt: "Could not open file"
Testers	Owen Newberg
Test result	Passed

6)

Test Type	<b>Functional Test</b>
Testing range	Create new file feature
Testing Input	Using an invalid file extension
Testing procedure	Pressing the new button and then changing the file extension to: "test.java" when creating the new file and then clicking the create button.
Expected Test Result	New file created as "test.java.java" without errors
Testers	Owen Newberg and Malcolm Zartman
Test result	Passed

## 6.3 Test Coverage

n/a

## 7 Appendices

### 7.1 Packaging and installation issues

**WARNING: YOU MUST USE JDK-11** *this application contains libraries which are not compatible with later JDK versions*

- **Part 1 | Downloading the App**

- Use this link to download the entire app and for your convenience we've attached a known compatible version of java in the downloadable ZIP :  
[https://drive.google.com/file/d/1A\\_njw4L3PmgWHO3vsjIKe5HkVoABLW7X/view?usp=sharing](https://drive.google.com/file/d/1A_njw4L3PmgWHO3vsjIKe5HkVoABLW7X/view?usp=sharing)

If “jdk-11.0.26\_windows-x64\_bin” is already installed and path variables are initialized skip to part 3

- **Part 2 | Installing JDK - 11**

- Double Click “jdk-11.0.26\_windows-x64\_bin” to run the installer.
- Keep clicking **Next** → until you see **Finish**.
- **Check it worked:** open *Start (Windows Key)* ▶ *Command Prompt*, type `javac -version`, and press **Enter**. If you see a version number (for example you should see, `java version "11.0.26" 2025-01-21 LTS`), you're good to proceed to step 3!

- **If it didn't work** (*java command not found OR empty reply*) follow these steps:

open *Start (Windows Key)* ▶ *Edit the System Environment Variables* ▶ *Enter*

*Click 'Environment Variables' ▶ Find the 'System Variables' section ▶ Click 'New' and enter the following. . .*

- Variable Name: JAVA\_HOME
- Variable Path: C:\Program Files\Java\jdk-11\bin

Now repeat the **Check it worked** step

- **Part 3 | Run CodezIDE**

- Double Click “CodezApp.jar” to run the program

- **Part 4 | Coding In Codez**

- **Make a new Java file:** Click File ► New. Give your file a name that ends with `.java`, like `HelloWorld.java`.
- **Open an existing file:** If you are opening an existing `.java` file. Then Click File (Located in the title bar) ► Open. This will allow you to navigate to the `.java` file that you wish to open.
- **Type your code:** Once a file is active you may begin typing in the text area, to write your code!
- **Tell the app where Java lives:** Codez will automatically find your path variables and populate the compiler location.

However if you would like to compile and run your code with a specific version of java you may do the following.

In the right panel, click the “Compiler Location” box, and browse to the JDK bin folder, usually `C:\Program Files\Java\jdk-23\bin`.

- **Run the program:** Hit the big green **Run ►** button (top-right). The terminal at the bottom will output the result of the run code!
- **Save your work:** Click **File ► Save**. This will make sure that all of your changes are saved correctly.
- **Close the IDE:** Click the red × on the top right corner of the application. Or click *File ► Exit*.






## 7.2 User Manual

**Install Java once, code forever. (Disclaimer: Only for Windows users.)**

**==If you are having any trouble with following the instructions, follow this setup tutorial for further guidance== (In Final project video submission)**

- **Part 1 | Downloading the App**

- Use this link to download the entire app:  
<https://drive.google.com/file/d/1aSXXvxoemuuGcWQ3fEArtZuRY2TaGiTt/view?usp=sharing>.
- Once downloaded, make sure to unzip the folder by first double clicking to get the option to extract all from the zipped folder, and then open the CodezIDE folder.
- Once the folder is open, you should see something that looks like this (look at desired folder contents below):

 jdk-11.0.26_windows-x64_bin	5/9/2025 11:48 PM	Application	145,142 KB
 CodezApp.jar	5/9/2025 11:48 PM	JAR File	99 KB
 default_empty_project_root	5/9/2025 11:48 PM	File folder	
 jdtls_workspace	5/9/2025 11:48 PM	File folder	
 lib	5/9/2025 11:48 PM	File folder	

- **Part 2 | Installing JDK - 11 & configuring system path variables**

- Once you are in this folder, double click on the “jdk-11.0.26\_windows-x64\_bin” to start downloading the required compiler to use our IDE.
- Once opened follow the instructions and just continue to click next until the jdk-11 is installed. Now that you installed jdk-11, you can now test to see if it is usable by opening up command prompt (to get to command prompt, go to the windows search bar from your keypad or from the bottom of your screen, here is a screenshot of where to find it: (Click on the windows icon on the left side of the search bar.)



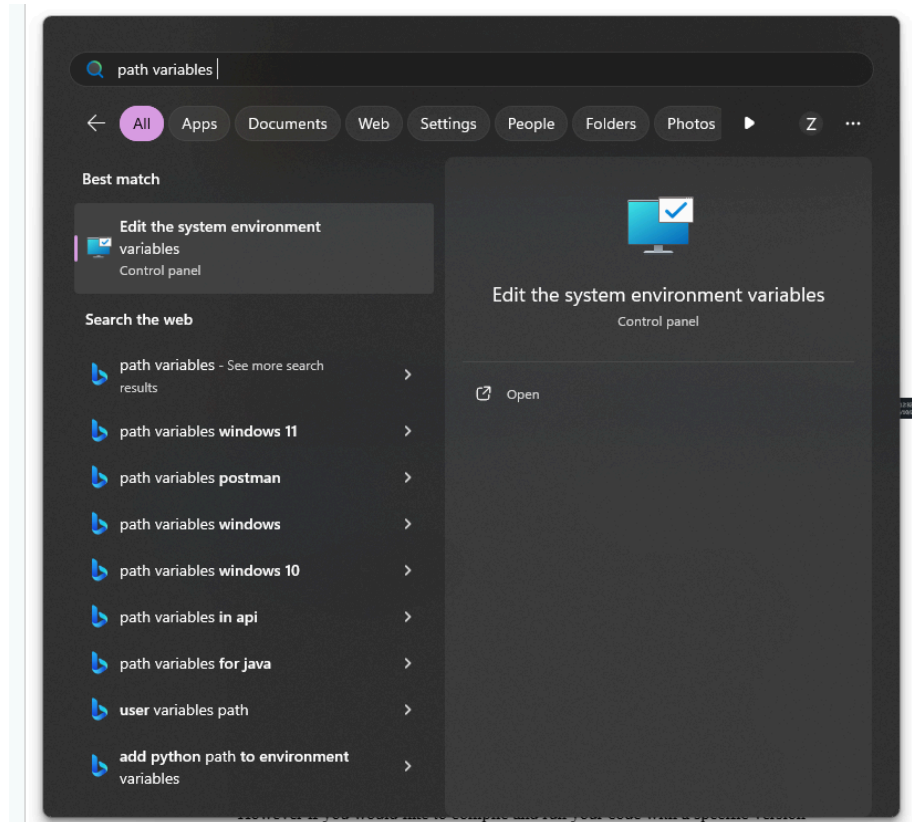
and here you will search command prompt or simply cmd), once you have command prompt open, enter in “Java -version” to make sure you have jdk-11 installed. It showed return the following result:

```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

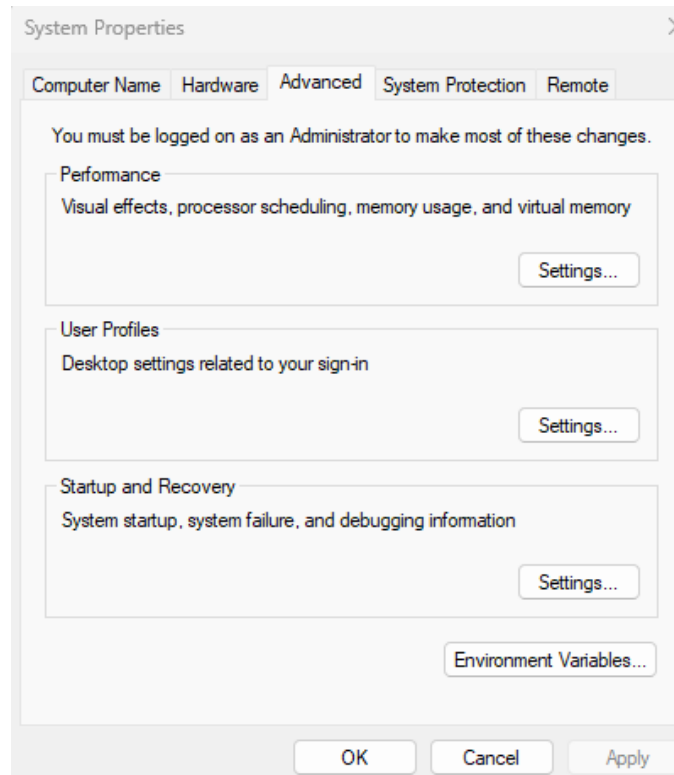
C:\Users\zartm001>Java -version
java version "11.0.26" 2025-01-21 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.26+7-LTS-187)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.26+7-LTS-187, mixed mode)

C:\Users\zartm001>
```

- If you do not see “java version ‘11....’” then something went wrong and you must proceed with the following steps. Otherwise, you’re good to proceed to step 3!
- Click on the windows icon again and press on the search bar and type in “path variables” to get this result:

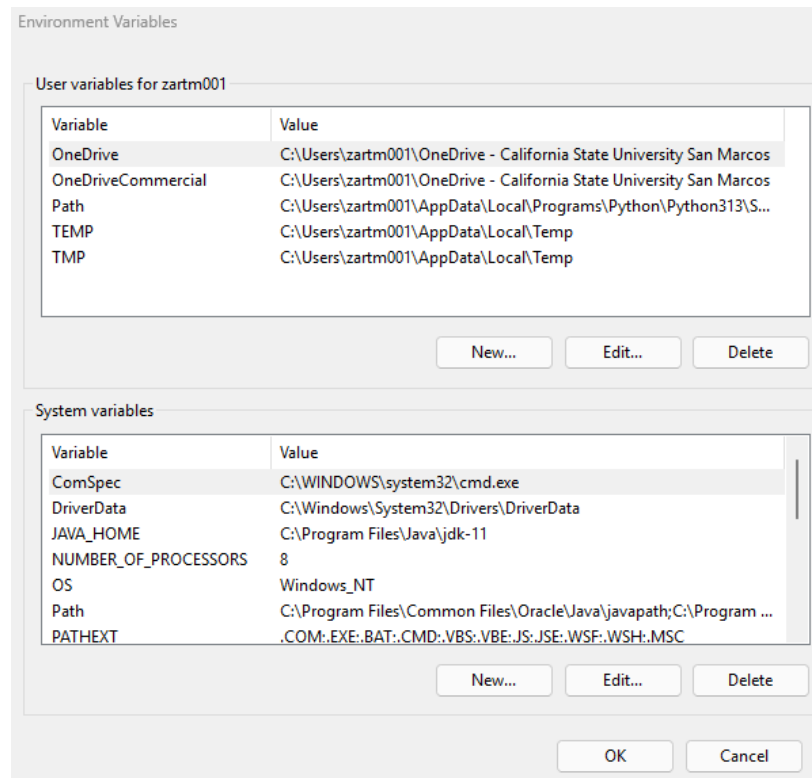


- Then click on the selection “Edit the system environment variables”, which is in your control panel.
- Once you are in the control panel, you should see something like this:



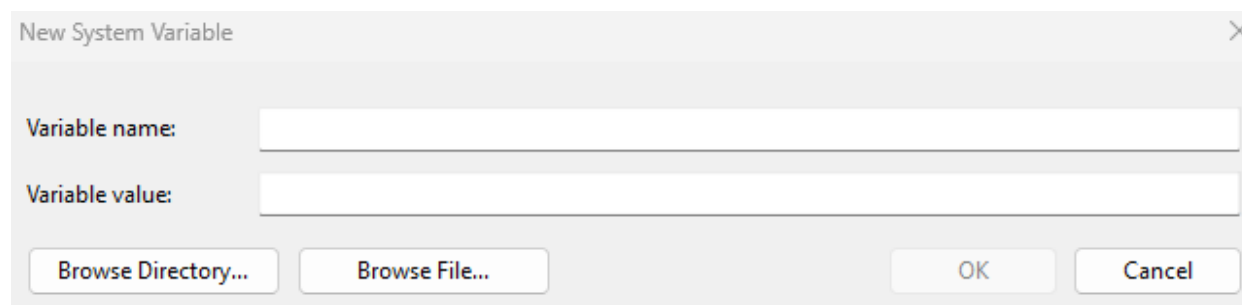
From here you will now press on “Environment Variables” which is towards the bottom of the window, right above the “OK”, “Cancel”, and “Apply” buttons.

- Once pressed, you will see something like this:



From here you will press the “New” button, if you don’t already have the “JAVA\_HOME” being associated with the file path to jdk-11. This is all under your “System variables” in the lower half of your screen.

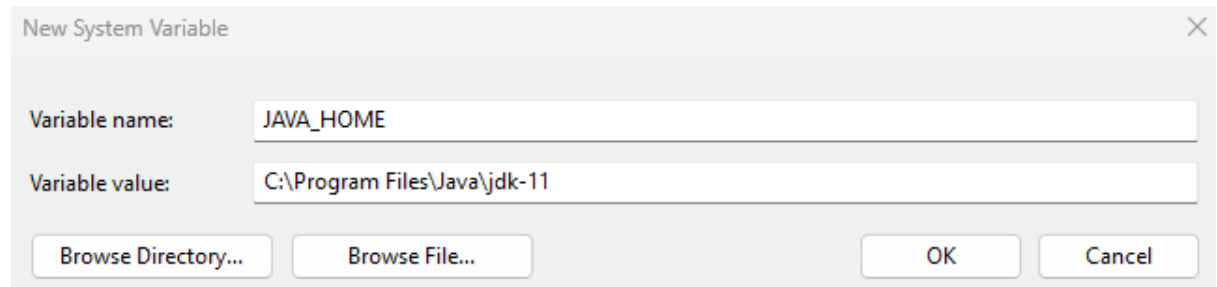
- Once the “New” button is pressed, you should see something like this:



From here you will put “JAVA\_HOME” in the text box to the right of where it says “Variable name:” and then you will browse your directory to where your downloaded jdk-11 is. Normally, it will be located in a folder named Java in your Program files folder which is in your Local Disk (C:) folder which is in your This PC folder.

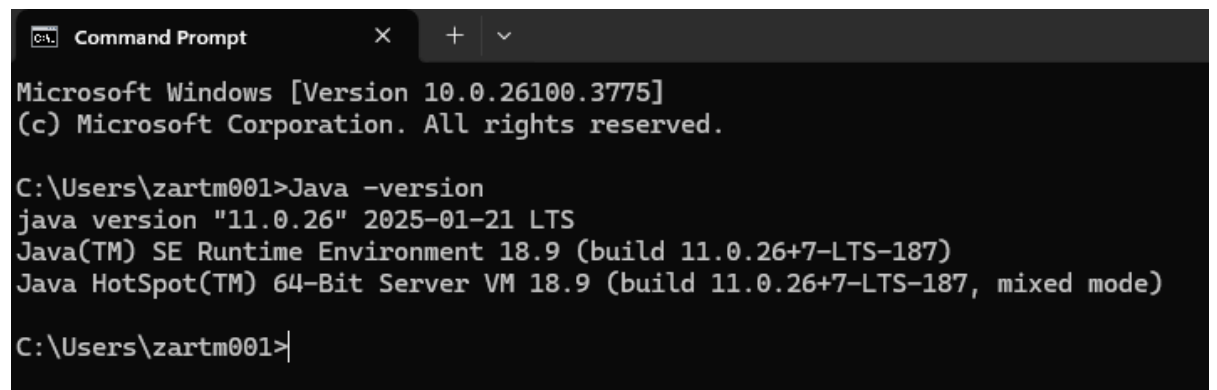
- Once you do the above instructions, you should have the text boxes filled in like such: (will vary depending on the location of your folder)





Once you are here, you can click OK and you should be able to see that System Variable show up now in the following step before creating this new one.

- Now that this is set up, you can now test to see if you have it by opening command prompt (to get to command prompt, go to the windows search bar—similar to previous step where we got to the “Edit the system environment variables” part— and here you will search command prompt or simply cmd.), once you have command prompt open, enter in “Java -version” to make sure you have jdk-11 installed. It showed return the following result:



```
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\zartm001>Java -version
java version "11.0.26" 2025-01-21 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.26+7-LTS-187)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.26+7-LTS-187, mixed mode)

C:\Users\zartm001>
```

If you do not see “java version ‘11....’” then something went wrong and re-try these steps, otherwise, you are now ready to run our app!

- **Part 3 | Run CodezIDE**
  - Double Click “**CodezApp.jar**” to run the program
- **Part 4 | Coding In Codez**
  - **Make a new Java file:** Click File ► New. Give your file a name that ends with **.java**, like **HelloWorld.java**.
  - **Open an existing file:** If you are opening an existing .java file. Then Click File (Located in the title bar) ► Open. This will allow you to navigate to the .java file that you wish to open.
  - **Type your code:** Once a file is active you may begin typing in the text area, to write your code!

- **Tell the app where Java lives:** Codez will automatically find your path variables and populate the compiler location.

However if you would like to compile and run your code with a specific version of java you may do the following.

In the right panel, click the “Compiler Location” box, and browse to the JDK bin folder, usually `C:\Program Files\Java\jdk-23\bin`.

- **Run the program:** Hit the big green **Run ▶** button (top-right). The terminal at the bottom will output the result of the run code!
  - If input is required for your program, there is a text box at the bottom of our IDE where you can enter in input when prompted and click the submit button when finished inputting into the text box. If you are unsure of when to input, just know that the program is only finished once you see “Program exited with code: 0” (0 if successful run).
- **Save your work:** Click **File ▶ Save**. This will make sure that all of your changes are saved correctly.
- **Close the IDE:** Click the red × on the top right corner of the application. Or click *File ▶ Exit*.

### 7.3 Open Issues

Features considered but not finished

- We weren’t able to fully support Mac or Linux users, while they could open and save files, they couldn’t compile or run them. This was mostly because we didn’t test enough with the Mac user on our team, so we missed the issue until it was too late to fix. Additionally, we weren’t able to fully implement C++ functionality either, while we were able to run programs, we weren’t able to add the text highlighting, so we decided to focus on our primary features. Due to C++’s removal we also didn’t implement the “add compiler” button behaviours, which would’ve helped users install a compiler or fetch the necessary binaries for their chosen language. We considered linking users to official setup guides or even handling the download and setup ourselves, but without time or proper support for those languages we decided to halt its development. Another feature we had to drop was letting users open an entire folder so all their program files would show up in the IDE, ready to edit and save. We also hoped to add error messages that would notify the user of syntax errors before compiling while this feature was a part of the JD TLS ecosystem more time was needed to develop UI support. The edit tab has no behaviours as of right now. Lastly there were error handling cases which due to focusing on refactoring and other critical features, were skipped. These include forcing the user to only be able to open .java files, robust JD TLS error handling for bad file paths and multithreading file operations (Open, Save, New) to avoid hitching. Even though we couldn’t include these extra features, we still covered all our main use cases and delivered a working IDE with no major broken functionality.

## 7.4 Lessons Learned

### 7.4.1 Project Management & Task Allocations (SLO #2.i)

- How your team allocate tasks and responsibilities and where could you improve in the future?
  - We allocate our tasks and responsibilities in a strategic way. From the gecko we knew who our frontend guy was and backend guys were. Once we had an idea of what tasks needed to be accomplished we further broke our tasks up for every person in our group. So, we ended up with Stetson handling the UI/frontend, Owen handling the file system, Malcolm handling the compilation and running, and Alex handling the text highlighting. This separation of concerns made our lives a whole lot easier when allocating specific tasks and when time came to implementation. We could improve on the allocation process by delegating a person on at least two tasks, so there would be at least 2 people on each task. This would lead to more pair programming sessions and another layer of bulletproofing possible risks. However, I will say that we always had tasks for everyone to do, so everyone always had something to do.
- How your team do project planning activities?
  - Our team conducts project planning activities by having our domain expert and project director, Alex, identify tasks that need to be done for each part of our system by decomposing the tasks over a project schedule timeline. We aimed to first achieve the tasks with little to no dependencies so we don't have to wait on any one person for a certain feature. This allowed us to prototype early and see if the given task were implemented properly. We also used experience-based techniques, in which Alex had some knowledge on how long a given task would take given the effort needed for it.
- How much effort do your team perform risk analysis along the process?
  - At the beginning of the semester, we conducted a risk analysis to identify and evaluate potential issues that could hinder our software development. While we kept this initial analysis in mind throughout the project, we didn't consistently revisit or expand on it during the development process, something that we know more successful teams typically do. Continuous risk evaluation is considered best practice, and this is an area where we could have improved. That said, we still engaged in risk analysis when deciding whether to implement new features, ensuring we considered potential impacts before committing to it. Additionally, because we laid solid groundwork early in the semester, we had a contingency plan in place to handle potential risks as they came up. One recurring issue was team members being out sick or busy with other responsibilities. Fortunately, our strategy of maintaining clear and consistent documentation allowed others to step in and continue the work without major setbacks. That said, incorporating more regular pair programming would have further strengthened our ability to manage these disruptions and improve overall team resilience.
- How did your team update each other's progress and adjust your plans?
  - We had weekly updates almost every Saturday or Sunday where we'd go over what we'd done, what we were working on, any issues we were running into, and what we planned to do next. Sometimes we even used lab time or met up after lecture when things got busy. These regular updates helped us stay on the same page and figure out if someone

needed help or if we could shift people around to keep things moving. Overall, our weekly updates kept the project running smoothly, and most of the adjustments we had to make were just from someone falling behind due to other responsibilities going on or us deciding to drop extra features that weren't essential.

- How did your team track changes and respond to changes occurred?
  - Before making any changes, we'd talk it over with the team and do a quick vote if we needed to, just to make sure everyone was on board. Once we agreed on something, we'd log it in our Discord server as part of our "commit list," where we kept track of tasks under "to-do," "in-progress," "done," and "frozen." That's where we'd post updates or plans for that week's sprint. It was an extra layer to help us keep track of changes beyond just talking about them.

#### 7.4.2 Implementation (SLO #2.iv)

- How did your team performed code review and refactoring to improve code quality? Provide a few code snippets (screenshots) that can best represent the team's coding quality.
  - So, we started out the semester going straight into production and trying to prototype as soon as possible, which was sweet since we had working software very early on in the semester. However, that led to a rather monolithic structure and once we got to the Design portion of this semester we saw how far off we were to good maintainable code. But, once we constructed a design, we had a very good idea on how we would refactor in which we separated concerns where we broke up one file into many smaller ones that execute a specific task. In the beginning we had a compileAndRun code file that handled all the compiling, running, error handling, and file fetching, which was very verbose and hard to pinpoint issues in the code. Once we refactored however, we broke that into a compile code class, run code class, compiler check class, and had error handling not only in those classes but also in the User-Interface layer before calling these functions. Not to mention, when interacting with the User-Interface layer, we implemented a facade pattern that minimized the amount of communication required to execute the desired functionality (compiling and running code). Additionally, we added some simple functions along with getters and setters that made code more clean, and interactions with other classes more simple. Here is a few screenshots to represent said changes:

// Getter to make code more clean and less verbose.

```
public static String getJavaRuntimePath(String userJavaPath) {
    if(userJavaPath.contains("bin/")){
        return userJavaPath + "java";
    } else if (userJavaPath.contains("bin")){
        return userJavaPath + File.separator + "java";
    } else {
        return userJavaPath + File.separator + "bin" + File.separator + "java";
    }
}

@Override
public void runCode(JTextArea uiOutput, String userSourcePath, String compilerPath, String userFileName, language chosenLang) {
    // Debug to UI output - show current state
    if (uiOutput != null) {
        uiOutput.append("Starting execution...\n");
    }

    ProcessBuilder run;
    try {
        if(chosenLang == language.Java){
            // getting the os of the machine
            String os = System.getProperty("os.name").toLowerCase();

            boolean hasJavaExtension = userFileName.endsWith(".java");
            String javaFileWithoutExtension = hasJavaExtension ? userFileName.replace(".java", "") : userFileName;

            String javaExecPath = getJavaRuntimePath(compilerPath);
        }
    }
}
```

// public functions used in a different layer, this helps in encapsulating the complexity of this class so other functions or variables aren't accessible to other files.

```
@Override
public boolean sendInputToProcess(String input) {
    if (isProcessRunning() && processWriter != null) {
        try {
            processWriter.write(input);
            processWriter.newLine(); // Add a newline
            processWriter.flush();
            return true;
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }
    return false;
}

public boolean isProcessRunning() {
    return isProcessAlive && currentProcess != null && currentProcess.isAlive();
}
```

// Facade to decrease the communication between classes. Now the UI layer only has to primarily communicate with the Facade rather than a multitude of classes.

```
// acts as a "dumb" facade; purely used for delegation of tasks
public class CarCFacade implements CompileAndRunCode_Interface {
    private final CompileCode_Interface compiler = new CompileCode();
    private final RunCode_Interface runner = new RunCode();

    @Override
    public int compileCode(JTextArea simpleOutput, String sourcePath, String compilerPath, String fileName, language langName){
        return compiler.compileCode(simpleOutput, sourcePath, compilerPath, fileName, langName);
    }

    @Override
    public void runCode(JTextArea simpleOutput, String sourcePath, String compilerPath, String fileName, language langName){
        runner.runCode(simpleOutput, sourcePath, compilerPath, fileName, langName);
    }
}
```

We also want to note that we did ongoing refactoring after creating our initial designs. At first, we tried to match our code to the designs, but that didn't fully work out because of how the implementation ended up. Once we were confident in the way we built it, we went back and refactored the design to better reflect the code. These weren't major changes, just small adjustments like adding or removing connections, renaming functions or classes, and occasionally tweaking the structure. After that, we kept the design updated to stay in sync with the code as we made minor additions.

- How much is your implementation consistent with your system design?
  - We would say that our implementation has reached a 95% consistency with our design. We aren't saying 100% accuracy due to the fact that we didn't include every single connection between classes due to the fact that it would've gotten very messy design wise. Other than that, not all attributes and member functions such as getters and setters are included in our design. We could've added everything that we have in our code in the design, but we thought what is included in our design is more than enough to replicate what we have in our implementation. The rest that we didn't include can be insinuated by a novice developer. All in all, we believe we came very close to replicating our design with our code, and vice versa.

### 7.4.3 Design Patterns

What are the design patterns used and why?

- Facade Pattern: We used a facade pattern so our run button class in our UI layer doesn't have to communicate with a multitude of different classes/interfaces with various functions associated with them. So, the facade allowed us to only give the run button class access to the functions necessary to accomplish the run button's feature, which is compiling and running code, which was put into a facade class. This made implementation for the UI developer a whole lot more simple.
- Singleton: The Singleton was used in order to only have one instance of the Parser class.

- Interface: We used a couple different kinds of interfaces for our design. The first way was through a facade pattern in which we had multiple interfaces that we piled into one for simplicity reasons (discussed more up above). Secondly, was plain old interfaces between the UI layer and the AppLogic layer. We had one that was between the inputButton class and the run code class in which they had an agreement of what function was needed.
- Inheritance: We used inheritance only in the UI layer for the different types of JPanels, JButtons, JFrame, and JMenus that are required to showcase our UI.

#### 7.4.4 Team Communications

How team communications were conducted and where could you improve in the future?

- We conducted communication through online weekly meetings on Discord where we discussed what we were working on, what we had done, what are we having problems with (if-any), and what we hope to start working on. Some weeks we weren't able to meet due to responsibilities from other classes, but for the most part we met and set the groundwork for success. Additionally, we have a Discord server for our project where we are able to contact each other at any time we may need assistance. Our team's communication has been very strong through the semester and we believe that has correlated with our success in the course and the completion of this project. We could improve our communication in the future by having quick discussions about our project before and after lecture and lab sessions to optimize our time. We could also have more frequent meetings over discord to act as another layer of check-ins at the very least.

#### 7.4.4 Technologies Practiced (SLO #7)

- GitHub (Desktop)
  - Push, Pull, Branch, Merge, Resolve
- Java 11, 23, 24 and Java Adoptium 23 (Configuring, Managing and Running)
  - Setting Up Environments, Packages and Compiling
  - Utilizing outside Jar libraries Gson 2.8.9 and JDTLS 1.9.0
  - Creating (Java ARchive) .JAR files
  - -encoding UTF8
  - -cp
- Gson 2.8.9
  - Using Struct-like Classes to format JSON messages
  - Parsing JSON messages and Sending JSON messages
- JDTLS 1.9.0, 1.44.0, 1.46.1, Snapshot
  - Learned Standard LSP protocol
    - Initialize Server, Receive, Send requests
  - Multi-Processing
    - Running Server in a separate process
- Java Language Features Learned
  - Multithreading
  - Process Builder
  - SwingUI
  - Utilizing Try Catch in a more formal app development setting
  - File management using NIO

- Supporting multiple device types
- GNU compiler
  - Had a setup and running C++ runtime
- Powershell and (CMD) Commands
  - dir vs ls
  - (CMD) ^ replace
  - xcopy vs Copy-Item

#### 7.4.5 Desirable Changes

Assume that you have another month to work on the project, what aspects of the system you would like to improve? What are the additional features you want to add to the system? [Each student should use a separate paragraph to respond to the questions]

- Stetson: The largest part of our system that we wanted initially was the support of 3 languages, Java, C++, and Python. I believe adding that would make our system much more viable for the larger Computer Science Community. Another aspect of our system that I wished we flushed out more was the “Left Panel”. This panel would show all of the other Java files within the folder that you are in. This would allow for easier transition between opening other files by being able to just click on them from the Left Panel. In the end though this functionality was scrapped to make time and space for more important backend functionality of our system.
- Owen: If we had another month on the project, the thing I would like to improve most was adding mac compatibility which we wanted to do especially me because I’m on mac but getting a working and best possible version of our app was more important than versatility but with more time would have been nice to add. Another thing would be supporting more languages would’ve been nice especially C++ as that’s mainly used at our school for beginner programmers. Another feature that I would’ve liked would have been a teams aspect. The reason is we were trying to replace replit as our goal and I think replit was good because of it’s teams feature and the usability for teachers and students but obviously that’s be a whole separate thing that was definitely not within our scope.
- Malcolm: If we had another month to work on the project, I would first want to add the “add compiler” button as stated in the 7.3 Open Issues section. That is something that would make the user’s experience a whole lot more friendly, in which the user won’t have to worry about what to download or how to do it. I would also like to work with Alex to figure out how to fully support C++, as well as, discuss how to make our app capable of expanding to a multitude of programming languages for both compiling, running, and text highlighting. At the end of the day, we are just trying to make coding easy! 😊
- Alex: If I had another month to work on this I think I’d add two things. . . I’d like to integrate the compiler installation as part of the app. Then I’d like to put a heavy focus on robustness. One of our primary goals with this app is to make coding EZ. So while more compilers, general language



support, and all those other fancy features are certainly where we had the most fun on this project. If I wanted to sell it, the next month needs to be spent on robustness. Ideally we could put together a proper user testing group with varying ranges of experience and really do our best to build a product that can be not only expanded but truly making it impossible for the user to fail. Then if there was still time I'd want to support C++, Python and JavaScript.

#### 7.4.6 Challenges Faced

Among requirements specification, system design, and system implementation, which one you think is the hardest task? Why? [Each student should use a separate paragraph to respond to the questions]

- Stetson: Personally for me, since I was handling all of the frontend UI the most difficult task was trying to figure out how to get Java Swing to work the way I wanted it to. For a while I had a lot of trouble overriding the default borders that swing would use and I ran into a real big headache with that. Luckily Alex was able to help me through it and showed me the solution to this problem. I would also like to bring up my limited understanding of GitHub when we first started this project. I didn't understand any of the terminology at the start, and it made me feel quite stuck for a while because I didn't know how to push things to main correctly. Thankfully my team members were with me every step of the way and made sure to support me through this. In the end, I learned a lot, and I'm proud of where we finished.
- Owen: I would say that system design is the hardest task and this is no matter how you do your project. For us it was difficult because we were already coding and then we were trying to find out how to do a mix of what we have already and what we need to change for what we learned about system design and how it lead to then refactoring. And then even if we did system design first I still think it's the hardest as you have to design this whole plan and then you may implement it and have to go back in your design and I just think everything no matter what comes down to your system design in software engineering.
- Malcolm: In my opinion, if you do system design right, that would be the hardest task. You have to due to the heavy lifting for the implementer in which you think about how packages should communicate and what key functions and variables should be in play for things to operate the way it is supposed to. So, lots of thinking is required here, at least if you want to make the implementation process more smooth. I will say, it does depend on the type of system though. For instance, an example during class where we had a spaceship has many spaceship parts. That seems simple right? But, there is so much behind the implementation of putting those parts together. So, context matters here but in this semester-long project, I would say the design portion was more challenging especially since we were all rather new to it.
- Alex: System implementation was the hardest for me because while I had a general idea of the best way to tackle it from the design more often than not a wrench would be thrown into the process that would require a redesign. Most of my struggles come from a lack of documentation on the fundamental inner workings of JDTLS and to some extent Swing UI having hidden defaults

called Look and Feel. There isn't time to play with these things nor did we have a reason or means to before prototyping other more critical systems. Libraries interacting differently also caused a massive 28 hour long headache that involved me spending 3 days trying to find a JDTLS version that's compatible with Java 23. Ultimately Java 23 and later installations cause a fundamental error with all versions of JDTLS as it sends extraneous JSON tokens out of sync breaking all functionalities of the library. While system design can be challenging for someone who hasn't touched UML before, I've worked in great detail with several of the design patterns and paradigms mentioned in the class so reading the designs was harder than translating the code into a clean and functional structure from our array of systems. It also helps that we had meetings to mock up the UI in Canva for Stetsons UI refactor and we all immediately prototyped standalone systems.