LearningTomorrow: Design Document

Version 1.1 – 2023.11.05

Created 2023.11.05

Project Name: LearningTomorrow

Tymofiy Sompura
Giovanni Faiz Lawand
Taras O Tsebro
Michael Jacob Jasinski

GitLab Repository:

https://mcsscm.utm.utoronto.ca/csc207_20239/group_32

SECTION: PROJECT IDENTIFICATION

LearningTomorrow is a platform that allows teachers to create a digital course for their students. LearningTomorrow upgrades the current learning experience for students using a combination of written notes, videos, and assessments such as practice quizzes and tests. The platform aims to create a better and more accessible learning experience for students, while letting them learn at their own pace. LearningTomorrow will enhance and add to the current functionality of educational learning platforms by firstly, adding in accessibility features that enable people of low vision and other people with disabilities to access LearningTomorrow's content. The features include text to speech, zooming in and zooming out of the content, turning on a colour filter for people with colorblindness, and increasing font size.

SECTION: USER STORIES

Lower Priority Value => Higher Priority

Name	ID	Owner	Description	Acceptance Criteria	Implementation Details	Priority	Effort
Add Lessons	1.1	Giovanni	Given that I am an instructor, I would like to be able to add separate lessons for my students to be able to go through in a digestible manner.	Provide the ability for students to go through lessons in smaller sections at a time.	Create a new ImageView with the associated text for the lessons, separate from the main section screen.	1	8
Element Insertions	1.2	Giovanni	As an instructor, it'd be great to be able to add different kinds of resources for learning, such as videos, pictures, and maybe even gifs.	Provide the ability for instructors to add different types of media.	Create a button on the toolbar for inserting elements, and make 3 buttons to decide whether they want to add a video, image, or gif. Then also add the label for the user to input the general position (top, bottom, middle).	1	8
Practice Quiz	1.3	Tymofiy	As a student, I would like to test myself after completing a lesson so that I make sure that I understood the content of the lesson	Allow the course creator to make quizzes. Allow the student to do the quiz with unlimited time and unlimited attempts.	Create a class that stores questions and randomly picks some number of questions to be given to the user.	1	5
Content Expand	1.4	Michael	As a student, I want to select a content section or material to display so that I may pick what to read / watch at	Provide an option to expand a given content section so that students may select the lesson	A button for each content or material name, that when chosen displays the contents of the content section.	3	3

			a given time to promote focus on one specific lesson or material.	or material that they desire to access.			
Content Collapse	1.5	Michael	As a student, I want to be able to hide a content section once I have completed it, to allow me to focus on the next task.	Provide an option for a student to hide a content section, and go back to the main page.	A button at the top of the content section that hides the given content section when clicked. This button must only be visible / accessible to the user once that section becomes displayed. (i.e. there must be something to hide)	3	3
Lesson Progression	2.1	Giovanni	As a student, I would like the ability to click a button that takes me to the next lesson in the sequence, from my current lesson.	Provide the ability to move to the next lesson, if applicable, from the current lesson.	A button on every lesson page, that loads the next lesson in sequence.	2	1
Check Course Completion	2.2	Tymofiy	As a student, I would like to see a list of lessons I have completed or not yet completed so that I see how much I've progressed through the course	A list of lessons is displayed with check marks beside completed lessons. The checkmarks have alt text for users with a screen reader	Once the user goes through the lesson, it should be marked as completed.	4	2
Content Filter	3.1	Michael	As a student, I want to filter the content sections available by section, topic, review, the content type (i.e. notes, video etc.) to help me navigate and narrow the content easier in order to make studying a specific concept easier.	Provide the option to search for specific lessons through a drop-down menu of topics and sections. There should always be content for one of the filter options so that no filter yields no content. (I.e. strictly relative filters)	Either a menu with a bunch of filter buttons, a drop down-select menu or a search bar that when the user selects their preference, the filter is then applied and content shown.	5	2
LATEX Integration	3.2	Tymofiy	As a math teacher, I would like to use LATEX to format mathematical formulas in the written notes and quizzes.	LATEX code is correctly compiled and displayed. Content editor is notified of any compilation	Use a library that renders LATEX code (most likely jlatexmath with integration into javaFX or jlatexmathfx)	3	5

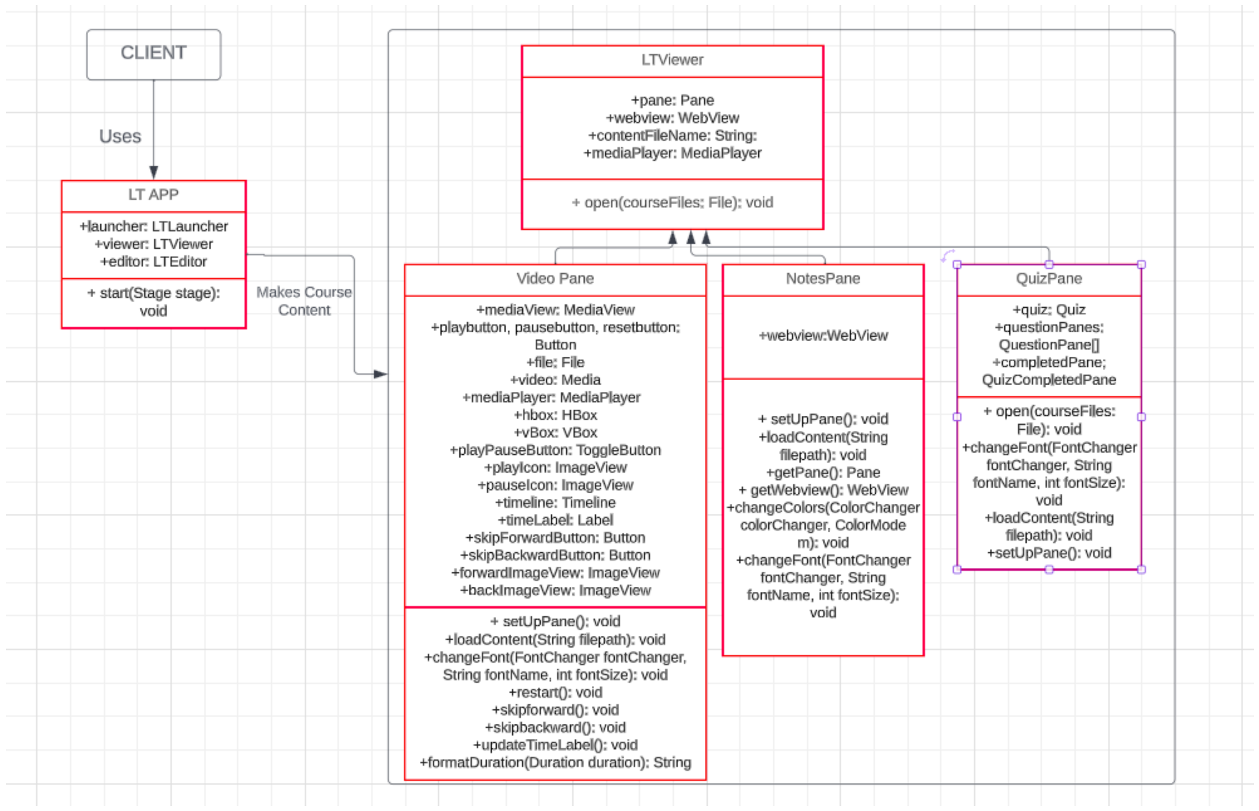
				errors.			
Awarding Course Certificate	3.3	Taras	As a student who has completed a course, I want receive a form of proof for completing the course so that I could prove that I have learned the given concepts taught within a course	Given that I am a user who has just completed all the requirements for a given course, when I reach the end of the courses' content I want to receive a course completion certificate listing the course.	At the end of the course once the user has adequately completed all the courses' materials the user will be presented with a .pdf certificate to download.	7	3
Accessibility Feature: Color Contrast Setting	4.1	Michael	As a color-blind student, I want to be able to change the color contrast settings of the application from 5 different filters, such as Protanope, Tritanope etc. so that I can better distinguish different parts of course notes and content.	Provide a minimum of 5 options including Tritanope, Protanope and other common colorblind modes for adjustable contrasts in the settings tab.	A settings window (scene) or toolbar that would have a colorblind tab or sub-category with a drop-down menu to select a color contrast option. It would then be applied after the contrast option is selected from the list of available options.	3	2
Accessibility Feature: Zoom In and Out	4.2	Giovanni	As a student, I am on the computer for a large portion of the day, where I often have to read off screens with small text (and other elements), and I end up getting a large headache and feeling very tired by the evening. So, I would like a website/application that I could use for school that wouldn't cause so much eye strain and be a little easier on the eyes.	Provide the option for the user to zoom in and out on the webpage.	Two buttons, one to increase zoom and one to decrease zoom. Every time the buttons are clicked, increase/decrease the zoom attribute by 5 percent of the current zoom.	2	3
Accessibility Feature: Change the font size	4.3	Tymofiy	As a person with partial vision loss, I would like to be able to change font size so that it is easier for me to read written notes.	Users are able to navigate to a text box using only keyboard or only mouse where they are able to input a font size. After changing font size, all text	Allow font size change in all components that deal with text.	1	2

				in the reader changes the font size.			
Accessibility Feature: Text-to-Speech	4.4	Taras	As a person with low-vision I would like to absorb the content acoustically so that I can access and learn the course's content without any accessibility issues.	Given that I am a visually-impaired course user interacting with course text, when I reach the end of the line and press the audible button then I want the text to be acoustically expressed.	Create a text-to-read speech button beside every line of text that when pushed will compile the given text in a line.	4	5

Section: UML Diagrams

Design Pattern #1: Factory Pattern (by Taras Tsebro)

Overview: This pattern will be used to implement LearningTomorrow's all-inclusive Mediaviewer designed to view videos, photos and audio in the course notes.


CLIENT
Uses
LT APP
+launcher: LTLauncher
+viewer: LTViewer
+editor: LTEditor
+ start(Stage stage): void
Makes Course Content
LTViewer
+pane: Pane
+webview: WebView
+contentFileName: String:
+mediaPlayer: MediaPlayer
+ open(courseFiles: File): void
Video Pane
+mediaView: MediaView
+playbutton, pausebutton, resetbutton: Button
+file: File
+video: Media
+mediaPlayer: MediaPlayer
+hbox: HBox
+vBox: VBox
+playPauseButton: ToggleButton
+playIcon: ImageView
+pauseIcon: ImageView
+timeline: Timeline
+timeLabel: Label
+skipForwardButton: Button
+skipBackwardButton: Button
+forwardImageView: ImageView
+backImageView: ImageView
+ setUpPane(): void
+loadContent(String filepath): void
+changeFont(FontChanger fontChanger, String fontName, int fontSize): void
+restart(): void
+skipforward(): void
+skipbackward(): void
+updateTimeLabel(): void
+formatDuration(Duration duration): String
NotesPane
+webview:WebView
+ setUpPane(): void
+loadContent(String filepath): void
+getPane(): Pane
+ getWebview(): WebView
+changeColors(ColorChanger colorChanger, ColorMode m): void
+changeFont(FontChanger fontChanger, String fontName, int fontSize): void
QuizPane
+quiz: Quiz
+questionPanes: QuestionPane[]
+completedPane: QuizCompletedPane
+ open(courseFiles: File): void
+changeFont(FontChanger fontChanger, String fontName, int fontSize): void
+loadContent(String filepath): void
+setUpPane(): void

Implementation Details: The UML diagram outlines these main components:

The LT App, like a factory pattern, processes the requests for the given client. Then after the given client the LT APP then communicates with the classes that make the course content. The LTViewer then is the class responsible for making the video, notes and quiz pane. All these panes are then used by LTViewer to provide the consumer with a good end user experience. The video pane is used to make panes that have videos in the given class. Notes pane processes given text to analyze and make panes for the course notes that make text notes for the students. Quiz pane then makes quizzes for the user. All these panes then make a great learning academy for the users.

How User Stories Are Used

As described in section USER STORIES the user story content expands and makes the given video pane, notes pane and quiz pane. The content expand and content collapse user stories are visible as the given course content can be viewed and accessed in the given time.

How SOLID Principles Are Used

S - single responsibility principle

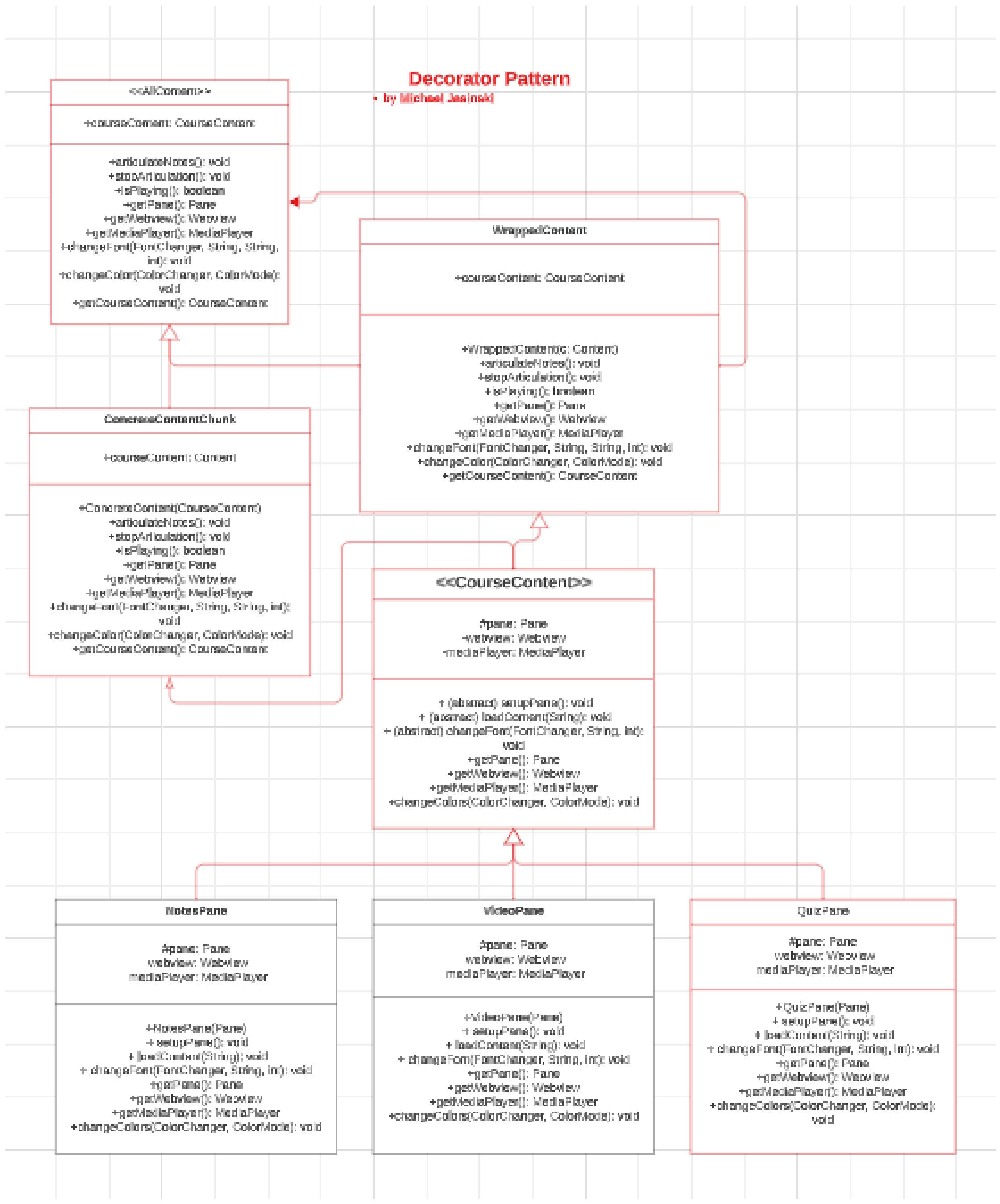
The given design pattern only has the single responsibility to view videos, view notes, and make quizzes. It only is supposed to make the notes for the learning platform.

O -Open-closed Principle

The principle is not violated as all classes can have additional functionality be extended by them onto other classes. No single class takes on multiple responsibilities, thus this principle holds true!

Design Pattern #2: Decorator Pattern (by Michael Jasinski)

Overview: This pattern will be used to implement LearningTomorrow's accessible notes
and video options. It will play the audio for the notes, and not change the audio for the video. The purpose of this is to add some much needed simple voiceovers for html files, for all types of students. Additionally, this adds the way to add accessible audio, if the teacher wishes.


Decorator Pattern
• by Michael Jasinski
<<AllContent>>
+courseContent: CourseContent
+articulateNotes(): void
+stopArticulation(): void
+isPlaying(): boolean
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeFont(FontChanger, String, String, int): void
+changeColor(ColorChanger, ColorMode): void
+getCourseContent(): CourseContent
WrappedContent
+courseContent: CourseContent
+WrappedContent(c: Content)
+articulateNotes(): void
+stopArticulation(): void
+isPlaying(): boolean
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeFont(FontChanger, String, String, Int): void
+changeColor(ColorChanger, ColorMode): void
+getCourseContent(): CourseContent
ConcreteContentChunk
+courseContent: Content
+ConcreteContent(CourseContent)
+articulateNotes(): void
+stopArticulation(): void
+isPlaying(): boolean
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeFont(FontChanger, String, String, int): void
+changeColor(ColorChanger, ColorMode): void
+getCourseContent(): CourseContent
<<CourseContent>>
#pane: Pane
-webview: Webview
-mediaPlayer: MediaPlayer
+ (abstract) setupPane(): void
+ (abstract) loadContent(String): void
+ (abstract) changeFont(FontChanger, String, Int): void
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeColors(ColorChanger, ColorMode): void
NotesPane
#pane: Pane
webview: Webview
mediaPlayer: MediaPlayer
+NotesPane(Pane)
+ setupPane(): void
+ loadContent(String): void
+ changeFont(FontChanger, String, Int): void
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeColors(ColorChanger, ColorMode): void
VideoPane
#pane: Pane
webview: Webview
mediaPlayer: MediaPlayer
+VideoPane(Pane)
+ setupPane(): void
+ loadContent(String): void
+ changeFont(FontChanger, String, Int): void
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeColors(ColorChanger, ColorMode): void
QuizPane
#pane: Pane
webview: Webview
mediaPlayer: MediaPlayer
+QuizPane(Pane)
+ setupPane(): void
+ loadContent(String): void
+ changeFont(FontChanger, String, Int): void
+getPane(): Pane
+getWebview(): Webview
+getMediaPlayer(): MediaPlayer
+changeColors(ColorChanger, ColorMode): void

This structure allows multiple things to occur. Firstly it allows the video mp4s to play the same. Secondly, it allows the audible description of the notes to be played, without it being directly attached to the notes object i.e. image. It also adds the potential to voice-overs of a quiz.

Implementation Details: The UML diagram outlines these main components:

- An abstract class CourseContent which denotes basic features of both the VideoPane and NotePane classes, which allow the direct playing of the audio files and video files.
- The wrapper WrapperContent modifies the behaviors of the content, in order to set and override original audio files if needed with their accessible counterparts.

- The non-wrapped counterpart the ConcreteContent allows the normal audio to be played and not overridden by the accessible counterpart or in the case of Notes, no voice over.
- The AllContent Interface ties all the functionality together by abstracting the wrapped and unwrapped counterpart, just showing the methods that will be used to play and stop the content

How User Stories Are Used

- This design pattern mainly appeals to the VoiceOvers / Low-Vision Accessibility user story. It aims to play an audio narration of the notes if the content is “wrapped”. Whereas it will be able to only play the base files if unwrapped (i.e. missing files or accessible mode / narration off etc.)
- Furthermore this design pattern and implementation also fall under the expanded and collapsed content section user stories, as the video and audio content is only to be playable when the section is opened, and stopped when closed. In other words, this design pattern enables the specific feature of the content to be played and closed when the section is expanded and collapsed respectively. .

How SOLID Principles Are Used

S - single responsibility principle

- This design pattern follows single responsibility principles as it is responsible only for managing and playing the accessible / non-accessible versions of the content
- Furthermore, it also splits up video from audio as they have two different

O -Open-closed Principle

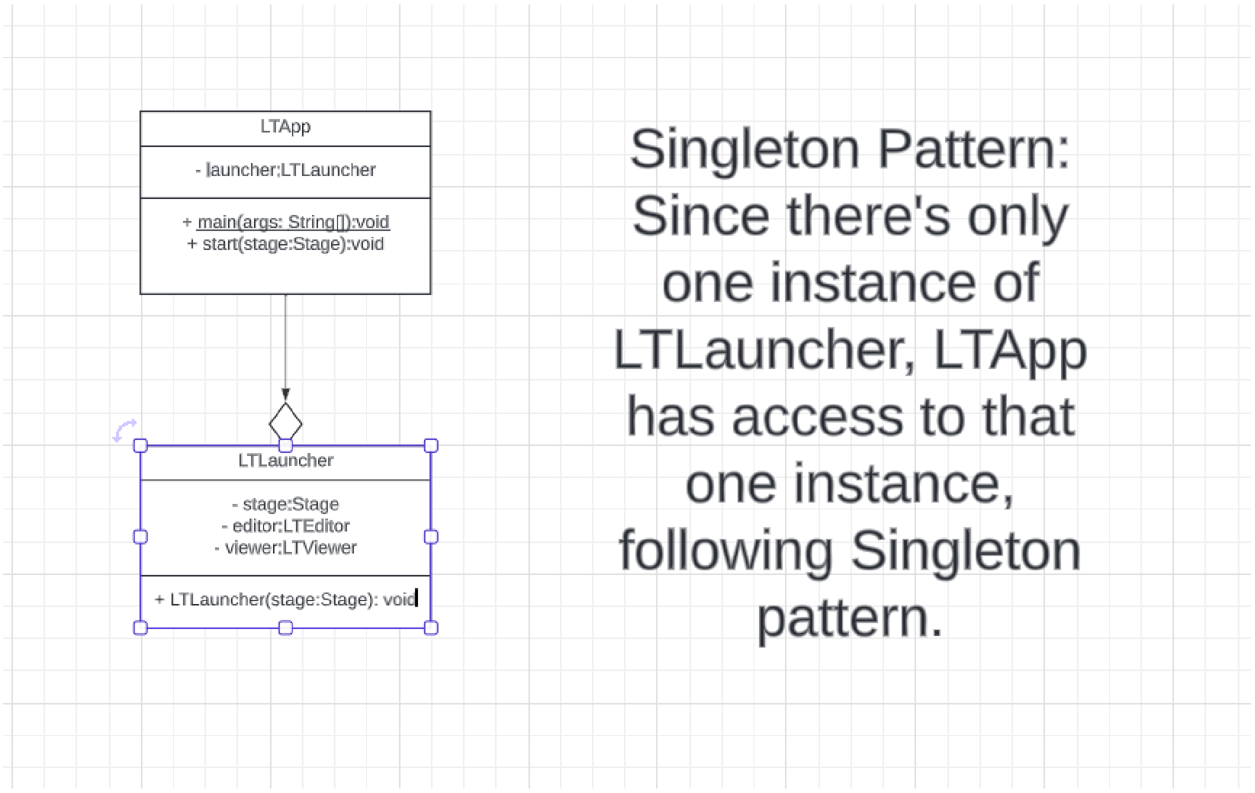
The principle holds as all classes are able to gain functionality through wrapping or extension.

I-Interface Segregation Principle

- Segregate the different types of playable content like notes audio and videos through a content abstraction
- Abstract the base functions and behaviors of the content through the ContentChunk interface, hiding the wrapped and unwrapped functionality / below inner-workings

<u>**Design Pattern #3: Singleton Pattern (by Giovanni Lawand)**</u>

Overview: This pattern is used on LTApp, which has access to the one instance of LTLauncher.


LTApp
- launcher:LTLauncher
+ main(args: String[]):void
+ start(stage:Stage):void
LTLauncher
- stage:Stage
- editor:LTEditor
- viewer:LTViewer
+ LTLauncher(stage:Stage): void
Singleton Pattern: Since there's only one instance of LTLauncher, LTApp has access to that one instance, following Singleton pattern.

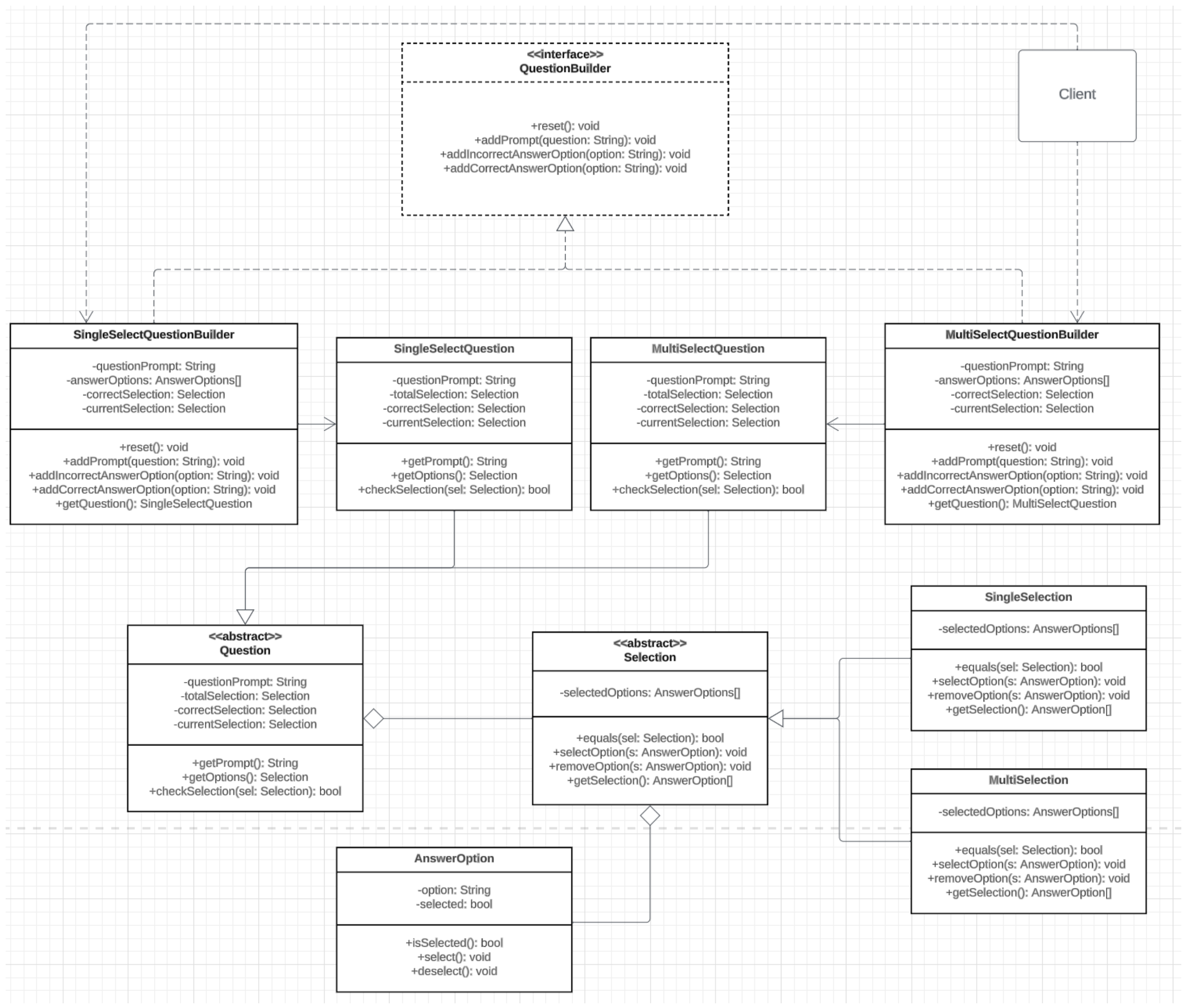
Implementation Details: The UML diagram outlines these main components:

- *LTApp* is the class with a method to launch the application.
- *LTLauncher* is the class with methods to load the user interface, which can then select the course files to load and display to the user.

The *Client* must have course files in a folder with the name of the course in the "Saved" directory of the application's code, with a format file so various features such as a table of contents can be properly generated. The *Client* can create a LTApp instance, and call the start method when they want to launch the application. In doing so, the LTLauncher's constructor performs all the actions required and displays the course according to how they've specified.

Design Pattern #4: Builder Pattern (by Tymofiy Sompura)

Overview: This pattern is used to simplify the construction of Question instances for practice quizzes.



<<interface>>
QuestionBuilder
+reset(): void
+addPrompt(question: String): void
+addIncorrectAnswerOption(option: String): void
+addCorrectAnswerOption(option: String): void
Client
SingleSelectQuestionBuilder
-questionPrompt: String
-answerOptions: AnswerOptions[]
-correctSelection: Selection
-currentSelection: Selection
+reset(): void
+addPrompt(question: String): void
+addIncorrectAnswerOption(option: String): void
+addCorrectAnswerOption(option: String): void
+getQuestion(): SingleSelectQuestion
SingleSelectQuestion
-questionPrompt: String
-totalSelection: Selection
-correctSelection: Selection
-currentSelection: Selection
+getPrompt(): String
+getOptions(): Selection
+checkSelection(sel: Selection): bool
MultiSelectQuestion
-questionPrompt: String
-totalSelection: Selection
-correctSelection: Selection
-currentSelection: Selection
+getPrompt(): String
+getOptions(): Selection
+checkSelection(sel: Selection): bool
MultiSelectQuestionBuilder
-questionPrompt: String
-answerOptions: AnswerOptions[]
-correctSelection: Selection
-currentSelection: Selection
+reset(): void
+addPrompt(question: String): void
+addIncorrectAnswerOption(option: String): void
+addCorrectAnswerOption(option: String): void
+getQuestion(): MultiSelectQuestion
<<abstract>>
Question
-questionPrompt: String
-totalSelection: Selection
-correctSelection: Selection
-currentSelection: Selection
+getPrompt(): String
+getOptions(): Selection
+checkSelection(sel: Selection): bool
<<abstract>>
Selection
-selectedOptions: AnswerOptions[]
+equals(sel: Selection): bool
+selectOption(s: AnswerOption): void
+removeOption(s: AnswerOption): void
+getSelection(): AnswerOption[]
SingleSelection
-selectedOptions: AnswerOptions[]
+equals(sel: Selection): bool
+selectOption(s: AnswerOption): void
+removeOption(s: AnswerOption): void
+getSelection(): AnswerOption[]
MultiSelection
-selectedOptions: AnswerOptions[]
+equals(sel: Selection): bool
+selectOption(s: AnswerOption): void
+removeOption(s: AnswerOption): void
+getSelection(): AnswerOption[]
AnswerOption
-option: String
-selected: bool
+isSelected(): bool
+select(): void
+deselect(): void

Implementation Details. The UML diagram outlines these main components:

- The *QuestionBuilder* interface which includes methods for adding question prompt and answer options
- The *Question* abstract class storing the question itself, answer options, correct option, and currently selected option
- The *Selection* abstract class which represents a selection of answer options. *Selection* is also used to represent all answer options for the question
- 2 implementations each for *Selection*, *Question*, and *QuestionBuilder*, one dealing with single selection questions, and the other dealing with multiple selection questions
- *AnswerOption* representing an answer option that can be selected

The *Client* creates an instance of either *SingleSelectQuestionBuilder* or *MultiSelectQuestionBuilder* depending on which type of question they want to create. Then, the *Client* can create the question by calling *addPrompt*, *addIncorrectAnswerOption*, and *addCorrectAnswerOption* by simply supplying a String, instead of having to worry about instantiating all the underlying objects like *Selection*. Furthermore, by using separate methods, the need for massive constructors is removed.