Even Semester (2022)

BINUS UNIVERSITY

BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:

Surname: Staniswinata **Given Name:** Stephanie **Student ID Number:** 2501997836

Course Code : COMP6699001 **Course Name :** Object-Oriented Programming

Class : L2CC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignments: Final Project

Submission Pattern

Due Date : 10 June 2022 **Submission Date :** 10 June 2022

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.

3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International's terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Stephanie Staniswinata

Table of Contents

Project Specification

App name: Focus Time

App description: A desktop app to improve focus and productivity.

A tool application to improve one's focus and productivity. Inspired by previous Algorithm and Programming final project. With the goal to improve an existing application. The same objective is to make learning and studying easier and also create an environment that is less distracting.

This application offers three different tools. A timer, flashcard, and formula book. One can use the timer to measure their study time and keep a record. Flashcards to create and also study using digital flashcards with the intention to encourage active recall learning and to use less paper if using traditional paper flashcards. The formula book is helpful to refer to formulas when solving problems, inspired by Math and Physics formula booklet from high school.

Inputs:

- Mouse / Trackpad: to click buttons that run methods and navigate through the app
- Keyboard: input values to be run later by methods

Output:

- A visual application made using Javafx, navigate using buttons

Module / Library / Package used:

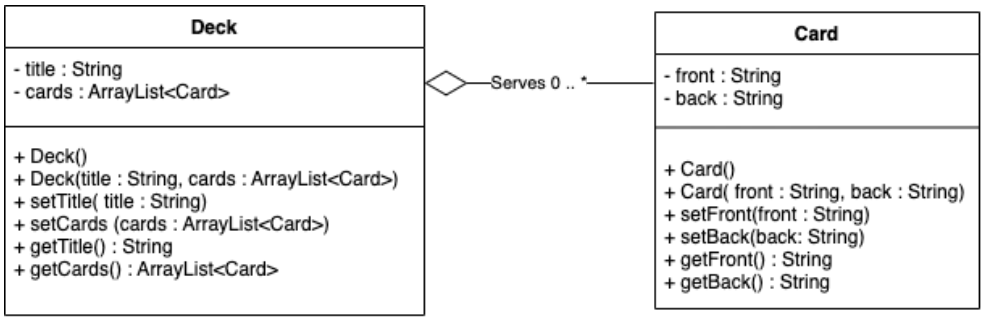
- Javafx -
- Gson - to work with JSON file and convert from and to JSON file
- Java NIO package - to write and read to a buffer

Solution Design

UML Diagram

A. Deck and Card class

Their relationship is HAS-A, aggregation relationship. The Deck class will have a string data type title and an array list with a Card object as data type (a collection of Card objects).




```
public class Deck {
// Gson name serialization - set key name
   @SerializedName("title")
   private String title;
// Gson name serialization - set key name
   @SerializedName("cards")
//    Association relationship between Deck and Card
//    Deck HAS-A Card object array
   private ArrayList<Card> cards;

// Empty constructor
   public Deck() {
   }

// Constructor with title and array of Card objects
   public Deck(String title, ArrayList<Card> cards){
       this.title = title;
       this.cards = cards;
   }

//    Setters
   public void setTitle(String title) {
       this.title = title;
   }

   public void setCards(ArrayList<Card> cards) {
       this.cards = cards;
   }

//    Getters
   public String getTitle() {
       return title;
   }
```

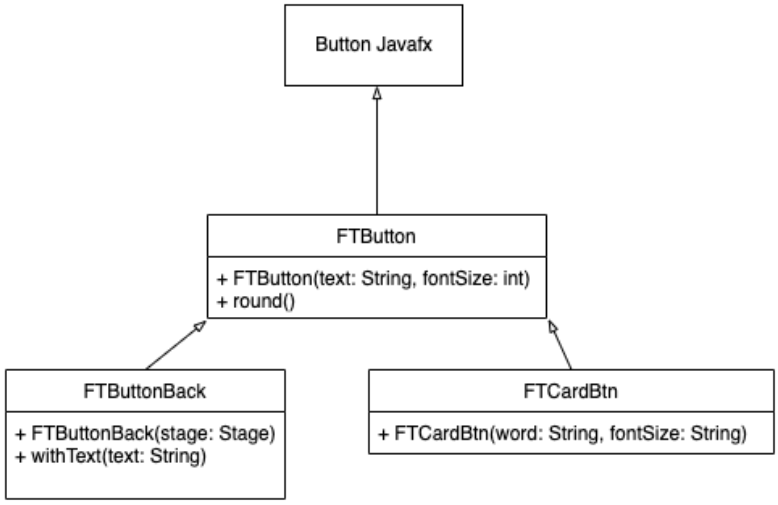
```
    public ArrayList<Card> getCards() {
        return cards;
    }
}
```

```
// Card class for every individual card stored in an array in Deck
// has two attributes, front and back - reflect the sides
public class Card {
// Gson name serialization - set key name
    @SerializedName("front")
    private String front;
// Gson name serialization - set key name
    @SerializedName("back")
    private String back;

//    Empty constructor - allow create Card object without initial values
    public Card() {
    }

//    Constructor with initializer
    public Card(String front, String back){
        this.front = front;
        this.back = back;
    }

//    Setters
    public void setFront(String front) {
        this.front = front;
    }

    public void setBack(String back) {
        this.back = back;
    }

//    Getters
    public String getFront() {
        return front;
    }

    public String getBack() {
        return back;
    }
}
```

B. Button Javafx, FTButton, FTButtonBack, FTCardBtn

All FTButton, extend the Button object from JavaFX. They can access all the methods. Created this to make it easier to create a themed button for the application. FTButton is for a general button, FTButtonBack is a back button that will link back to the main menu, while FTCardBtn is for illustrating a card, but it is just a big button. FTButton takes the ‘text’ attribute

to set it as the button text. In the round method, it set the height and width of the button to a 1:1 ratio.




```
// FTButton inherit the methods from Javafx Button object
public class FTButton extends Button{
// Constructor with text and fontSize attributes
   public FTButton(String text, int fontSize){
//        Super class to set the text to the text attribute from Button class
       super(text);
//        Set the text font to "Inter" font and desired fontSize
       setFont(Font.font("Inter", fontSize));
//        Set the text to wrap inside the button
       setWrapText(true);
//        Button styling - background color, height and width, border radius
       setStyle(
               "-fx-background-color: #FFFFFF; " + "-fx-pref-height: 45px; " + "-fx-pref-width: 200px;"
+ "-fx-background-radius: 30px;"
       );
//        When hovered, pointer change to hand pointer
       setCursor(Cursor.HAND);
   }

//    Method to change button to a 1:1 round button
   public void round(){
       setStyle(
               "-fx-background-color: #FFFFFF; " + "-fx-pref-height: 45px; " + "-fx-pref-width: 45px;" +
"-fx-background-radius: 30px;"
       );
   }
}
```

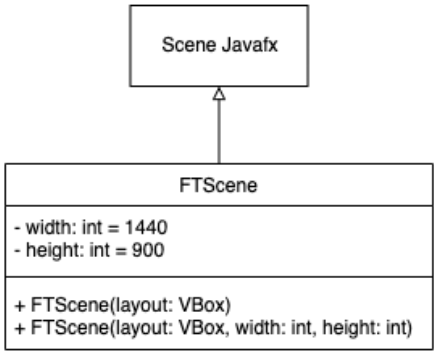
```
// Inherit the same methods from FTButton
// Use the superclass attribute - text and fontSize
public class FTButtonBack extends FTButton {
   public FTButtonBack(Stage stage) {
       super("<",20);
```

```
        setStyle(
                "-fx-background-color: #FFFFFF; " + "-fx-pref-height: 45px; " + "-fx-pref-width: 45px;" +
"-fx-background-radius: 30px;"
        );
//        When clicked, go to MenuScreen
        setOnAction(e -> Screens.MenuScreen(stage, "Welcome Back!"));
    }
//    Method to set a back button but instead of '<' can set to other text
//    Used for 'Done' button in learnCards screen
    public void withText(String text){
//        Set the button text - inherited from Button class
        setText(text);
        setStyle(
                "-fx-background-color: #FFFFFF; " + "-fx-pref-height: 45px; " + "-fx-pref-width: 200px;"
+ "-fx-background-radius: 30px;"
        );
    }
}
```

```
public class FTCardBtn extends FTButton{
//    Inherit the constructor
    public FTCardBtn(String word, int fontSize) {
//        From the superclass constructor method
        super(word, fontSize);
//        Set the Button size to fit the window to look like a card
        setStyle(
                "-fx-background-color: #FFFFFF; " + "-fx-pref-height: 600px; " + "-fx-pref-width:
1000px;" + "-fx-background-radius: 30px;"
        );
    }
}
```

C. Scene and FTScene

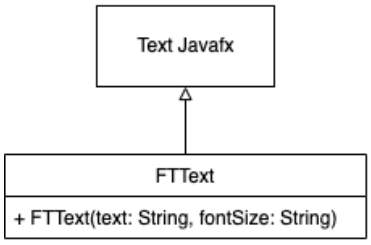
FTScene extend Scene, it has all access to Scene object from JavaFX. Created to ease theme-making. The constructors pass the VBox layout and also determine the height and width of the scene.



```
// FTScene to set the scene to my styling
// Prevent repetition
public class FTScene extends Scene {
//    Set initial / default width and height
   static int width = 1440;
   static int height = 900;

//    Constructor with default attribute
   public FTScene(VBox layout){
       super(layout, width, height);
//        Fill in the scene with color
       setFill(Color.web("#B2D7E2"));
//        Set the background the same color and also make CornerRadii and Insets empty
//        CornerRadii.EMPTY - to make the radius to 0, therefore square window
//        Inets.EMPTY - set the all offset inside the window to 0
       layout.setBackground(new Background(new BackgroundFill(Color.web("#B2D7E2"), CornerRadii.EMPTY,
Insets.EMPTY)));
   }
//    Constructor with attributes
   public FTScene(VBox layout, int width, int height){
       super(layout, width, height);
       setFill(Color.web("#B2D7E2"));
       layout.setBackground(new Background(new BackgroundFill(Color.web("#B2D7E2"), CornerRadii.EMPTY,
Insets.EMPTY)));
   }
}
```

D. Text and FTText

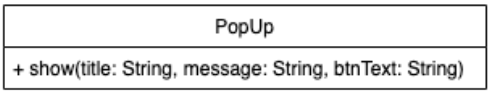
FTText inherit methods from Text JavaFX. It takes the ‘text’ and ‘fontSize’ attributes. Set the text to set the font that’s based on the fontSize to the ‘fontSize’.




```
public class FTText extends Text {

//    Set the text to "Inter" font with desired fontSize
//    Prevent repetition
   public FTText(String text, int fontSize){
       super(text);
       setFont(Font.font("Inter", fontSize));
   }
}
```

E. PopUp Class

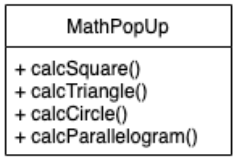
Created this class to make it easier to send a warning or error message to the user. The attributes are the title of the window, the error message or warning or prompt, and the button text.




```
public class PopUp {
//   Method to pop up the window
//   title - set window title
//   message - set the display message, or prompt
//   btnText - set the button text
   public static void show(String title, String message, String btnText){
//        Create a window
       Stage window = new Stage();
//        Disable the application to run without exiting this stage
       window.initModality(Modality.APPLICATION_MODAL);
//        Set window dimension
       window.setMinWidth(500);
       window.setMinHeight(350);
//        Set window title
       window.setTitle(title);

//        Create text that will display the message inside the window
       FTText text = new FTText(message, 20);
//        Set the size of wrap for the text
       text.setWrappingWidth(450);
//        Set the text to middle, and align center
       text.setTextAlignment(TextAlignment.CENTER);

//        Create a button with passed btnText and font 15
       FTButton btn = new FTButton(btnText,15);
//        Set the function of the button to close the window
       btn.setOnAction(e -> window.close());

//        Vertical layout to set all the elements
       VBox layout = new VBox(20);
       layout.getChildren().addAll(text, btn);
       layout.setAlignment(Pos.CENTER);

//        Set color, corner radius, and insets
       layout.setBackground(new Background(new BackgroundFill(Color.web("#B2D7E2"), CornerRadii.EMPTY,
Insets.EMPTY)));

//        Create a scene for the layout
       FTScene scene = new FTScene(layout, 500, 350);
//        Set scene to the window
       window.setScene(scene);
       window.showAndWait();
   }
}
```

F. MathPopUp Class

Created this class to make it easier to prompt the user to use the calculator feature, and it's in a different window from the main stage.




```
public class MathPopUp {

   public static void calcSquare(){
       Stage window = new Stage();
//        Disable the application to run without exiting this stage
       window.initModality(Modality.APPLICATION_MODAL);
//        Set stage size
       window.setMinWidth(500);
       window.setMinHeight(350);
//        Set stage title - header
       window.setTitle("Calculate Square");

       Square square = new Square("Square", "a plane figure with four equal straight sides and four
right angles");

       Label name = new Label(square.getName());
       name.setFont(Font.font("Inter", 20));
       name.setTextAlignment(TextAlignment.CENTER);
       Text desc = new Text(square.getDesc());
       desc.setFont(Font.font("Inter", 20));
       desc.setWrappingWidth(500);
       desc.setTextAlignment(TextAlignment.CENTER);

//        An empty text - placeholder to show the result after calculation
       Text result = new Text("");
       result.setFont(Font.font("Inter", 20));

//        Prompt and inputField for attributes
       Text inputSide = new Text("Side value: ");
       TextField side = new TextField();

       Text inputHeight = new Text("Height: ");
       TextField height = new TextField();

//        FTButton with Calculate text
       FTButton btn = new FTButton("Calculate",15);
       btn.setOnAction(e -> {
//            Try Catch - throw pop up box if there's an error
           try {
//                Store getText() to a value and set it to object attribute
               double sideValue = Double.parseDouble(side.getText());
               square.setWidth(sideValue);
               double heightValue = Double.parseDouble(height.getText());
               square.setHeight(heightValue);
//                Set the result from Methods.calculate to result text
//                Change the text to the output
```

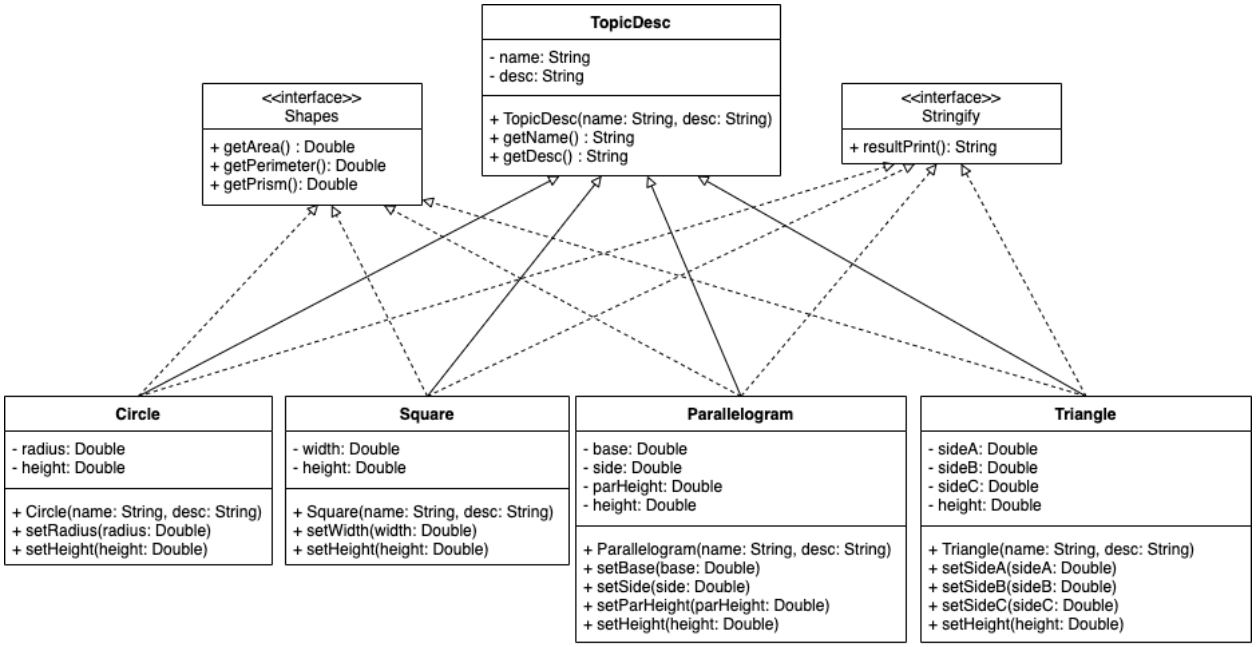
```
                result.setText(Methods.calculate(square));
//                  NumberFormatException - convert a string win an incorrect format to a numeric value
            }catch(NumberFormatException nfe){
//                  Pop up with warning text - value error
                PopUp.show("Value Error", "Input has to be filled or numbers. If none, input 0", "Ok");
            }
        });
//        Vbox for prompt1 and inputField1
        VBox input1 = new VBox(10);
        input1.getChildren().addAll(inputSide, side);

//        Vbox for prompt1 and inputField1
        VBox input2 = new VBox(10);
        input2.getChildren().addAll(inputHeight, height);

//        Horizontal for both inputs
        HBox inputs = new HBox(20);
        inputs.setAlignment(Pos.CENTER);
        inputs.getChildren().addAll(input1, input2);

//        Vertical for all elements
        VBox layout = new VBox(20);
        layout.getChildren().addAll(name, desc, inputs, result, btn);
        layout.setAlignment(Pos.CENTER);

//        Set the scene using FTScene, so it has the same theme, with
        FTScene scene = new FTScene(layout, 600, 500);
//        Set the scene to window
        window.setScene(scene);
//        Window shows and wait till it's closed by user, "x"
        window.showAndWait();
    }

//    No comments for the rest under, because all the same :)
    public static void calcTriangle(){...}

    public static void calcCircle(){...}

    public static void calcParallelogram(){...}
}
```

This class allow user to see the shape chosen and also the description of a shape. User also can input value to the TextField that will store it's value to a variable and use proper setters to the objects. When the user clicked on the 'Calculate' button, it will run the calculations and return the value to the result variable that will setText method to the calculation result.

I also added a try catch, if user input other than numbers to the TextField, it will display a pop-up from PopUp class that prompts the user to input with the right values. The error name is NumberFormatException, that will check if the imputed values are numbers or not.

G. TopicDesc Class
   To store the name of a topic and its description. Used for booklet subjects
H. Square, Circle, Parallelogram, Triangle class with interface Shapes and Stringify that also extends TopicDesc

Shares the same attribute but also polymorphism from the interfaces. The shapes mentioned also inherit methods from TopicDesc.




```java
// Class to create topic objects
public class TopicDesc {
//    Topic name
   private String name;
//    Description
   private String desc;

//    Constructor
   public TopicDesc(String name, String desc) {
       this.name = name;
       this.desc = desc;
   }

//    Getters
   public String getName() {
       return this.name;
   }

   public String getDesc() {
       return this.desc;
   }

}
```

```java
// Interface that has area, perimeter, and prism
public interface Shapes {
   Double getArea();
   Double getPerimeter();
   Double getPrism();
}
```

I use interface for shapes, because a class in Java can not inheritmore than one class. Which the first class it TopicDesc, therefore Shapes will be used to implement polymorphism too, since every shapes have different formula for area, perimeter, and prism.

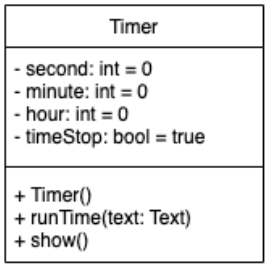
```
// Interface with resultPrint method, for everything that need printing
public interface Stringify {
   String resultPrint();
}
```

Interface Stringify made and used, because the same reason with Shapes. A class is not allowed to inherit from more than one class. Therefore this use to pass the resultPrint method. This can be used in Math, Physics, Chem, and other subjects that need a printed result from a calculation. In this program, I only have Math subject.

*For the shapes, check the code in github.

I. Timer

Timer class to create Timer objects, allow user to create multiple timer at the same time.




```
// Timer, a pop-up that will keep running in the background
public class Timer {
//    Initialization of vars
   private int second = 0;
   private int minute = 0;
   private int hour = 0;
   private boolean timeStop = true;

//    Constructor
   public Timer(){};

// runTime method to change the displayed value time changes over time
   public void runTime(Text text){
//        Every 60 sec passed, change minute to +1
       if(this.second == 60){
           this.minute++;
           this.second = 0;
       }
//        Every 60 minutes passed, change hour to +1
       if(this.minute == 60){
           this.hour++;
           this.minute = 0;
       }

//        Creation of variables
```

```
        String hourStr, minuteStr, secondStr;

//        When hour is divisible by 10 and equal to 0(int), add 0 in the start, so it would look life
ex= 05:00:00
        if((this.hour/10) == 0){
            hourStr = "0" + this.hour + ":";
        }else{
            hourStr = this.hour + ":";
        }

//        When minute is divisible by 10 and equal to 0(int), add 0 in the start, so it would look life
ex= 00:05:00
        if((this.minute/10) == 0){
            minuteStr = "0" + this.minute + ":";
        }else{
            minuteStr = this.minute + ":";
        }

//        When hour is divisible by 10 and equal to 0(int), add 0 in the start, so it would look life
ex= 05:00:00
        if((this.second/10) == 0){
            secondStr = "0" + this.second++;
        }else{
//            String value of - because there are no string indication for second, not like the others
            secondStr = String.valueOf(this.second++);

        }

//        Set the time to the text displayed
        text.setText(hourStr + minuteStr + secondStr);
    }
```

This block has constructor and runTime method. runTime method responsible to changing the clock text. It will change every second, 60 seconds, and 60 minutes. This will be used with the keyframe animation from JavaFX, will be run every second.

```
//    show method to display to the window
    public void show(){
        Stage timerWin = new Stage();
        timerWin.setTitle("FocusTime - Stopwatch");
        timerWin.setMinWidth(500);
        timerWin.setMinHeight(300);

//        Set the timer name - easier to keep track
        TextField name = new TextField();

//        Set the initial condition to 00:00:00
        FTText clock = new FTText("00:00:00", 40);
//        To change text, run the runTime method, for every one second duration
        KeyFrame keyframe = new KeyFrame(Duration.seconds(1.0), e -> runTime(clock));
//        Set timeline animation based on the keyframe time frame
        Timeline time = new Timeline(keyframe);
//        Play it non-stop until animation is stopped
        time.setCycleCount(Timeline.INDEFINITE);
//        Won't go back, it will loop non-stop
        time.setAutoReverse(false);
```

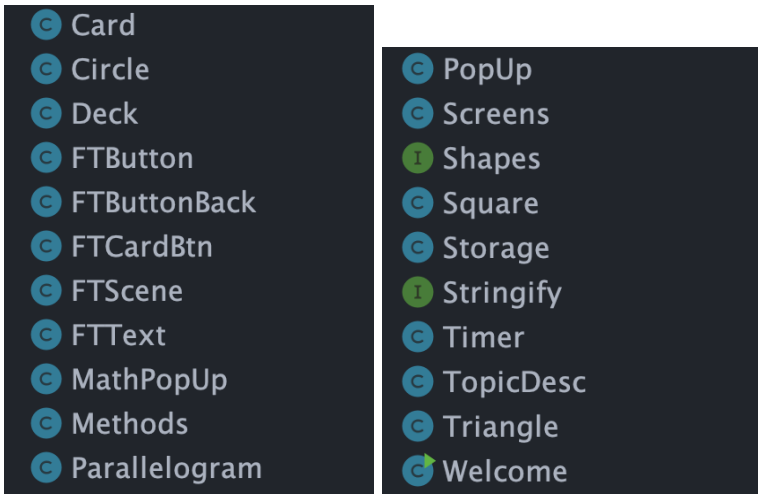
```
//        Display the status, empty - running, paused - when paused
        FTText statusTxt = new FTText("", 15);

//        Start the animation, and statusText to empty string
        FTButton startBtn = new FTButton("Start", 15);
        startBtn.setOnAction(e ->{
            if(timeStop){
                time.play();
                timeStop = false;
                statusTxt.setText("");
            }
        });

//        Pause animation, statusText to "Paused"
        FTButton pauseBtn = new FTButton("Pause", 15);
        pauseBtn.setOnAction(e ->{
            if(!timeStop){
                time.pause();
                timeStop = true;
                statusTxt.setText("paused");
            }
        });

//        Set everything to 0, pause the animation
        FTButton resetBtn = new FTButton("Reset", 15);
        resetBtn.setOnAction(e ->{
            this.hour = 0;
            this.minute = 0;
            this.second = 0;
            time.pause();
            clock.setText("00:00:00");
            statusTxt.setText("");
            if(!timeStop){
                timeStop = true;
            }
        });

        HBox buttons = new HBox(20);
        buttons.setAlignment(Pos.CENTER);
        buttons.getChildren().addAll(startBtn, pauseBtn, resetBtn);

        VBox clockNstatus = new VBox();
        clockNstatus.setAlignment(Pos.CENTER);
        clockNstatus.getChildren().addAll(clock, statusTxt);

        VBox layout = new VBox(20);
        layout.setAlignment(Pos.CENTER);
        layout.getChildren().addAll(name, clockNstatus, buttons);
        layout.setBackground(new Background(new BackgroundFill(Color.web("#B2D7E2"), CornerRadii.EMPTY,
Insets.EMPTY)));

        FTScene scene = new FTScene(layout, 500, 300);
        scene.setFill(Color.web("#B2D7E2"));
        timerWin.setScene(scene);
        timerWin.show();

//        Set when the window is closed, no matter if its paused, running, or reset
//        Will set everything to 0, same with reset button
        timerWin.setOnCloseRequest(e -> {
```

```
            this.hour = 0;
            this.minute = 0;
            this.second = 0;
            time.pause();
            clock.setText("00:00:00");
            statusTxt.setText("");
            if(!timeStop){
                timeStop = true;
            }
        });
    }
}
```

This block has show method, will display the time and also textField that user can use to name the Timer.

Discussion

In this project, I learned some new things in java. Other than implementing OOP such as inheritance, polymorphism, and interface. I also learned about JavaFX for styling and Gson for accessing and editing JSON files. Below are the java files in this project. I won't be discussing all of them. Some classes are simple classes with constructors, getters, and setters.



A. Welcome.java

This file allow user to run and launch the app. With initialization of the window for the app.

```
   @Override
   public void start(Stage stage) {
       stage.setTitle("FocusTime");

       Label welcomeText = new Label("Welcome to FocusTime");
       welcomeText.setFont(Font.font("Inter", 96));

       Label descText = new Label("Your one stop focus center");
       descText.setFont(Font.font("Inter", 20));

       Label labelName = new Label("Name: ");
       TextField input = new TextField();
       input.minWidth(40);
       labelName.setFont(Font.font("Inter", 32));

//        Button will run the MenuScreen screen with passed name from user input
       FTButton btn = new FTButton("Enter", 32);
       btn.setOnAction(e -> {
//            Check if name is null, pop up error message window
           if(input.getText() == null || input.getText().trim().isEmpty()){
               PopUp.show("Input error", "Name can't be empty", "Ok");
           }else{
               Screens.MenuScreen(stage, Methods.randomGreet() + input.getText());
           }
       });

       HBox inputName = new HBox();
```

```
        inputName.getChildren().addAll(labelName, input);
        inputName.setAlignment(Pos.CENTER);

        VBox layout = new VBox(10);
        layout.getChildren().addAll(welcomeText, descText, inputName, btn);

        layout.setAlignment(Pos.CENTER);

        FTScene scene = new FTScene(layout);
        stage.setScene(scene);
        stage.setResizable(false);

        stage.show();
    }
```

This function is to set the window to be display to the user. Label is a title text, bigger than normal text object. To set the font and size, use setFont method. Textfield was used to get user input and can store the value to a variable that we can use later. A button can be set to run a code by setOnAction and using lamba function (e ->). There is also error prevention in a shape of pop-up. If the name is null, it will pop an error message, if not it will continue to the MenuScreen and with one of the attribute being randomGreet() + name input from user. HBox means horizontal box, will store all children in a horizontal stack. While VBox, will store them or layout them vertically. Then FTScene responsible to get the final layout as a scene and it will be set to the stage using setScene method from Stage object.

```
    public static void main(String[] args) {
//        Launch the application
        launch();
    }
```

This function will allow the code to run and launch as an app, since its an extension of Application by JavaFX.

B. Storage.java

This file is used to store some data that will be used later in Method.java or Screens.java

```
public class Storage {
//    Use this later in the greetings for mainMenu
    public static String[] greetings = {"Hello, ", "Hi, " , "Welcome, " , "Hewwo, " , "Hey, ", "你好, ",
"こんにちは, "};

//    For english subject, easier to edit
    static TopicDesc jargon = new TopicDesc("Jargon", "Special words or expressions that are used bu a
particular profession or group and are difficult for others to understand");
    static TopicDesc logos = new TopicDesc("Logos", "The appeal to logic");
    static TopicDesc pathos = new TopicDesc("Pathos", "The appeal to emotion");
    static TopicDesc ethos = new TopicDesc("Logos", "The character or emotions of a speaker or writer
that are expressed in the attempt to persuade an audience");
}
```

C. Methods.java

This file contains multiple methods used in this application.

```
//    Method to randomly use greetings from the greetings list in storage
   public static String randomGreet(){
//        Random class initialized
       Random random = new Random();
//        Get the length of list, set to max
       int max = Storage.greetings.length;
//        Use max as end range for random
       int numRand = random.nextInt(max);
//        Return the greeting from the list based on random
       return Storage.greetings[numRand];
   }
```

This function is to generate random greeting for the main menu, accessing the list from Storage.java. Gives randomness to the user.

```
//    Method to load the deck to show in the library
   public static ArrayList<FTButton> loadDeckLib(Stage stage) throws IOException {
//        Use gson package
       Gson gson = new Gson();
//        User reader to read the json file
       Reader read = Files.newBufferedReader(Paths.get("decks.json"));
//        Initialized an arrayList to store FTButton objects
       ArrayList<FTButton> allDeck = new ArrayList<>();

//        Create a json array and convert json from the file to jsonArray
       JsonArray array = gson.fromJson(read, JsonArray.class);
//        To check if the array is empty
       if(array == null){
//            Return allDeck Arraylist
//            Which an empty arrayList - will display nothing
           return allDeck;
//        When the array is not empty
       }else{
//            For loop through the json array
           for(int i = 0 ; i <array.size();i++){
//                Initialized ArrayList for Card objects
               ArrayList<Card> cardArr = new ArrayList<>();
//                JsonArray for cardList  - listing all the card in a deck - front and back
               JsonArray cardList = (JsonArray) ((JsonObject) array.get(i)).get("cards");
//                For loop through the cardList json array
               for(int j = 0; j < cardList.size(); j++){
//                    Make it as an object with the name cards, for each front-back card
                   JsonObject cards = (JsonObject) cardList.get(j);
//                    Get the value of the front key of Card object - access using json methods
//                    replaceAll used to get rid of the " mark at the beginning and end
                   String front = String.valueOf(cards.get("front")).replaceAll("^\"|\"$", "");
//                    Get the value of the back key of Card object - access using json methods
                   String back = String.valueOf(cards.get("back")).replaceAll("^\"|\"$", "");
//                    Create a card object with the front and back value from json
                   Card card = new Card(front, back);
//                    Add the card object to the cardArr
                   cardArr.add(card);
               }

//                Get the value of the name key of Deck object - access using json method
               String name =
String.valueOf(((JsonObject)array.get(i)).get("title")).replaceAll("^\"|\"$", "");
```

```
//                Create a button with deck's name
                FTButton btn = new FTButton(name, 20);
//                Create a Deck object with name from json and cardArr for Card objects array
                Deck deck = new Deck(name, cardArr);
//                Set the button to go to LearnScreen screen with the stage and deck object parameter
                btn.setOnAction(e -> Screens.LearnScreen(stage, deck));
//                Add the button to the button collection that will be displayed called allDeck
                allDeck.add(btn);
            }
        }
//        Return allDeck, to display all the deck buttons
        return allDeck;
    }
```

This function is used to load the Deck objects from JSON file named “decks.json”. Every Deck objects are stoed in JSON format in JSON file. Using Gson package, we are allowed to read and also write JSON. In this function, it will read a JSON file and use for loop to create each Deck and append it to an ArrayList<FTButton>, where Decks are turned into buttons and will be displayed on screen. Proper use of getters are used here. Uses Reader and fromJson method allow the program to get data from a JSON file.

```
//    Method to store just made Deck to json file
    public static void doneCreating(Deck deck, Stage stage) throws IOException {
//        Initialize gsonBuilder object with format (from setPrettyPrinting())
        Gson database = new GsonBuilder().setPrettyPrinting().create();

//        Create Type data type variable to get the return of a list of Deck data type
        Type listType = new TypeToken<List<Deck>>(){}.getType();
//        FileReader to read the "decks.json" file
        FileReader reader = new FileReader("decks.json");
//        Create a list of Deck from the reader and based it on the type return from listType
        List<Deck> deckList = database.fromJson(reader, listType);
//        Close the reader
        reader.close();

//        Check if deckList is empty
        if(deckList == null){
            deckList = new ArrayList<>();
        }

//        Add deck from userInput to deckList list
        deckList.add(deck);
//        Create a file writer named writer with the target to "decks.json" file
        FileWriter writer = new FileWriter("decks.json");
//        Write a json to the file, with the value from deckList
        database.toJson(deckList, writer);
//        Close the writer
        writer.close();

//        Go to the CardScreen screen with passed stage attribute
        Screens.CardScreen(stage);
    }
```

This function have two attributes, Deck object and Stage object. This method is responsible to store the deck that was just made by the user to JSON file. It has FileWriter and also toJson method to write the Deck object to JSON format.

```
//    Method to return the square result, allow to print it to the screen
//    Run the passed Square object
   public static String calculate(Square square){
       return square.resultPrint();
   }
//    All the same method, but different passed attribute
   public static String calculate(Triangle triangle){
       return triangle.resultPrint();
   }
   public static String calculate(Circle circle){
       return circle.resultPrint();
   }
   public static String calculate(Parallelogram parallelogram){
       return parallelogram.resultPrint();
   }
```

These methods will return a string that displays the shape name and also result of calculation to the screen. There are four different methods, that has the same name but take different data type, this was used to make get easier access on calculating. It just use the interface method from Stringify.java resultPrint() and it will print the value from each shapes.

D. Screens.java

This file contains all the screen, everything is navigated using buttons in the application.

```
//    Menu Screen - welcome and name input
   public static void MenuScreen(Stage stage, String name){
//        Display the name to the window
       Label userName = new Label(name);
//        Set the font and size
       userName.setFont(Font.font("Inter", 96));

//        Create Timer button, when clicked, display the Timer Popup
       FTButton timerBtn = new FTButton("Timer", 32);
       timerBtn.setOnAction(e -> {
//            Create a Timer object - allow multiple timer to run at the same time
           Timer timer = new Timer();
//            Display the timer window
           timer.show();
       });

//        Crate FlashCard button, when clicked, go to CardScreen display
       FTButton cardBtn = new FTButton("FlashCard", 32);
       cardBtn.setOnAction(e -> {
           try {
               CardScreen(stage);
           } catch (IOException ex) {
               ex.printStackTrace();
           }
       });

//        Create Booklet button, when clicked, go to BookletScreen display
```

```
        FTButton bookletBtn = new FTButton("Booklet", 32);
        bookletBtn.setOnAction(e -> BookletScreen(stage));

//        All buttons set in a horizontal layout
        HBox buttons = new HBox(20);
        buttons.setAlignment(Pos.CENTER);
        buttons.getChildren().addAll(timerBtn, cardBtn, bookletBtn);

//        Vertical layout for all elements
        VBox layout = new VBox(40);
        layout.setAlignment(Pos.CENTER);
        layout.getChildren().addAll(userName, buttons);

//        Set the scene to the stage
        FTScene scene2 = new FTScene(layout);
        stage.setScene(scene2);
    }
```

This MenuScreen displayed three different buttons to navigate through the app. Timer to pop-up a timer tot the screen. FlashCard will navigate to the FlashCard Screen. Booklet button will bring users to the Booklet screen.

```
//    CardScreen - display  collection of deck cards
    public static void CardScreen(Stage stage) throws IOException {
        stage.setTitle("FocusTime - Flashcard");

        Label title = new Label("FlashCard");
        title.setFont(Font.font("Inter", 80));
        title.setLayoutX(550);
        title.setLayoutY(30);
//        Show direction to navigate
        Text direction = new Text("Click card deck to learn");
        direction.setFont(Font.font("Inter", 20));
        direction.setLayoutX(630);
        direction.setLayoutY(140);

//        Back button to the main menu
        FTButtonBack backBtn = new FTButtonBack(stage);
        backBtn.setLayoutX(50);
        backBtn.setLayoutY(50);

//        To create deck
        FTButton createBtn = new FTButton("+ Create", 20);
        createBtn.setLayoutX(1200);
        createBtn.setLayoutY(50);
//        To createDeck display
        createBtn.setOnAction(e -> createDeck(stage));

//        Pane - easier to use x and y to plot the position of elements
        Pane navbar = new Pane();
        navbar.getChildren().addAll(createBtn, backBtn, title, direction);

        VBox decks = new VBox(20);
        decks.setAlignment(Pos.CENTER);

//        The returned value from loadDeckLib will be displayed here
//        Assign the array returned to an array and use for loop to
```

```
//        create button for each element
       ArrayList<FTButton> array = Methods.loadDeckLib(stage);
       for (FTButton ftButton : array) {
//            Display the buttons in Vbox layout
           decks.getChildren().add(ftButton);
       }

       VBox layout = new VBox(40);
       layout.setAlignment(Pos.CENTER);
       layout.getChildren().addAll(navbar, decks);

       FTScene scene = new FTScene(layout);
       stage.setScene(scene);
   }
```

This screen displays the deck library, it displays Deck cards and user can click them and will be navigated to the chosen Deck. This screen uses the method loadDeckLib, that return an ArrayList<FTButton> that was made from JSON file.

```
//    LearnScreen has a big button act as a card that can be flipped, to learn the chosen deck
   public static void LearnScreen(Stage stage, Deck deck){
       stage.setTitle("FocusTime - Learn Flashcards");

//        Return the deck name using getter
       Label deckName = new Label(deck.getTitle());
       deckName.setFont(Font.font("Inter", 50));

//        Allow iteration through the card array
       ListIterator<Card> i = deck.getCards().listIterator();

//        Object reference that can be updated and changes automatically
       AtomicReference<Card> card = new AtomicReference<>(deck.getCards().get(0));

//        String variable to store the first card front's side
       String show = card.get().getFront();
//        FTCardBtn - object with a big sized button and take superclass of FTButton
       FTCardBtn cardBtn = new FTCardBtn(show, 30);
//        When clicked, change the text to the back or front
       cardBtn.setOnAction(e -> {
           if(Objects.equals(cardBtn.getText(), card.get().getFront())){
               cardBtn.setText(card.get().getBack());
           }else{
               cardBtn.setText(card.get().getFront());
           }
       });

//        Use the iterator and previous button to navigate to the card before
       FTButton prevWord = new FTButton("<", 20);
       prevWord.round();
       prevWord.setOnAction(e -> {
           if(i.hasPrevious()){
               card.set(i.previous());
               cardBtn.setText(card.get().getFront());
           }else{
//               When it's the top of the list, give pop up
               PopUp.show("Beginning of the deck", "This is the first card.", "Ok");
           }
```

```
        });

//        Use iterator and next button to navigate to the card after
        FTButton nextWord = new FTButton(">", 20);
        nextWord.round();
        nextWord.setOnAction(e -> {
            if(i.hasNext()){
                card.set(i.next());
                cardBtn.setText(card.get().getFront());
            }else{
                PopUp.show("End of the deck", "This is the last card. Congratulations!", "Ok");
}
        });

//        Done button that act as a back button
        FTButtonBack doneBtn = new FTButtonBack(stage);
        doneBtn.withText("Done");

        HBox mainBtn = new HBox(40);
        mainBtn.setAlignment(Pos.CENTER);
        mainBtn.getChildren().addAll(prevWord, cardBtn, nextWord);

        VBox layout = new VBox(20);
        layout.setAlignment(Pos.CENTER);
        layout.getChildren().addAll(deckName, mainBtn, doneBtn);

        FTScene scene = new FTScene(layout);
        stage.setScene(scene);
    }
```

This screen displays a card (a big button), next and previous button, and done button. Card button made out of FTCardBtn, and when clicked, it will ‘swap’ form front to back side. Next and previous button will go to next or previous card on the ArrayList<Card>. While the done button is made out of FTButtonBack, it has the same function as back button, but it has the ‘Done’ writing. Here I use iterator in java that allows me to iterate through an ArrayList and return a value from an object. I also use AtomicReference, that allows the code to automatically update the value of a variable.

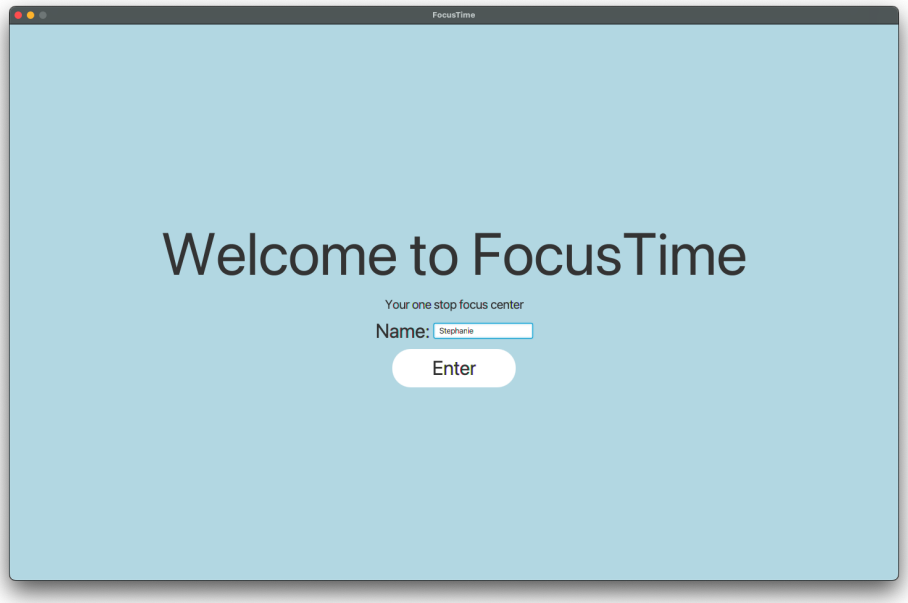
```
//    FormulaScreen - show the subjects for the booklet
    public static void BookletScreen(Stage stage){
        stage.setTitle("FocusTime - Booklet");

        Label title = new Label("Subjects");
        title.setFont(Font.font("Inter", 80));
        title.setLayoutX(550);
        title.setLayoutY(30);

        FTButtonBack backBtn = new FTButtonBack(stage);
        backBtn.setLayoutX(50);
        backBtn.setLayoutY(50);

        Pane navbar = new Pane();
        navbar.getChildren().addAll(backBtn, title);

//        Buttons to go to subject screen
```

```
        FTButton englishBtn = new FTButton("English", 20);
        englishBtn.setOnAction(e -> englishScreen(stage));
        FTButton mathBtn = new FTButton("Math", 20);
        mathBtn.setOnAction(e -> mathScreen(stage));

        HBox buttons = new HBox(30);
        buttons.setAlignment(Pos.CENTER);
        buttons.getChildren().addAll(englishBtn, mathBtn);

        VBox layout = new VBox(20);
        layout.setAlignment(Pos.CENTER);
        layout.getChildren().addAll(navbar, buttons);

        FTScene scene = new FTScene(layout);
        stage.setScene(scene);
    }
```

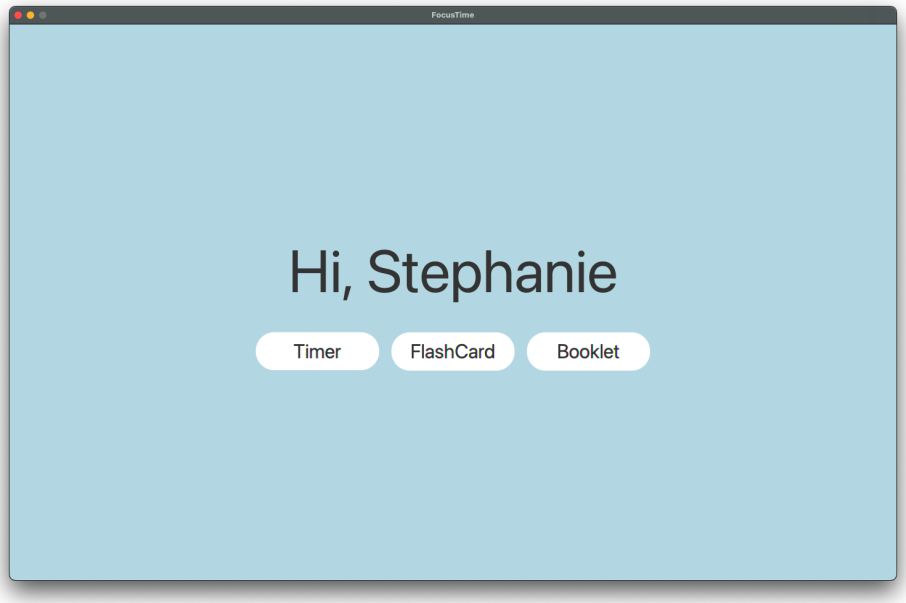
This BookletScreen displays the ‘English’ and ‘Math’ subject button that will navigate the user to the English and Math Menu. English will be displayed with PopUp class, while Math will be displayed with MathPopUp, that can take input to be calculated based on the chosen shape class.

Evidence of Working Program

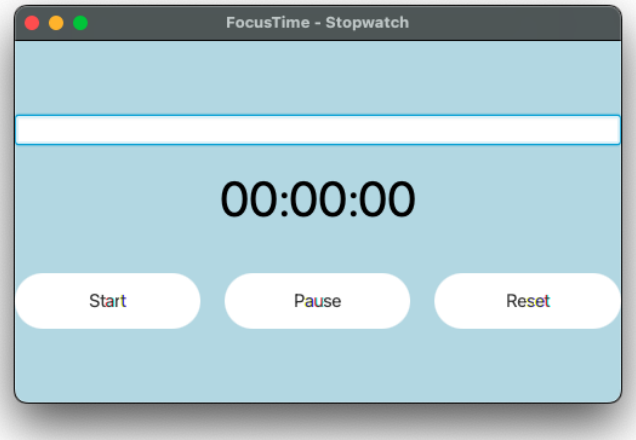
1. Welcome page, user input name.



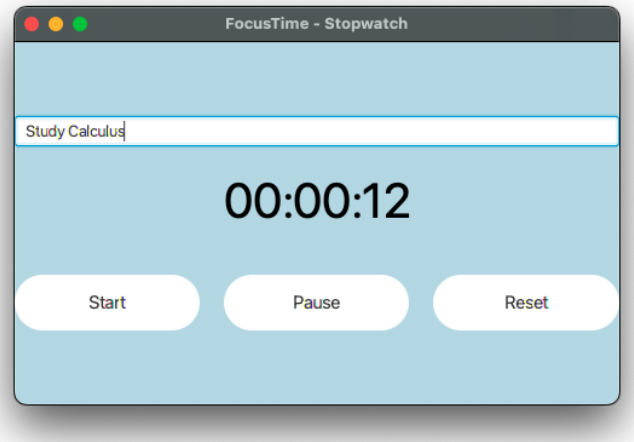
2. Go to the menu page, the user can choose between three options.



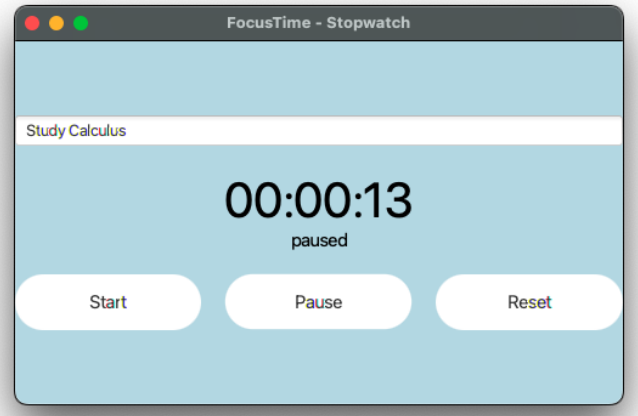
3. Timer button is clicked, and a timer pop-up will appear.



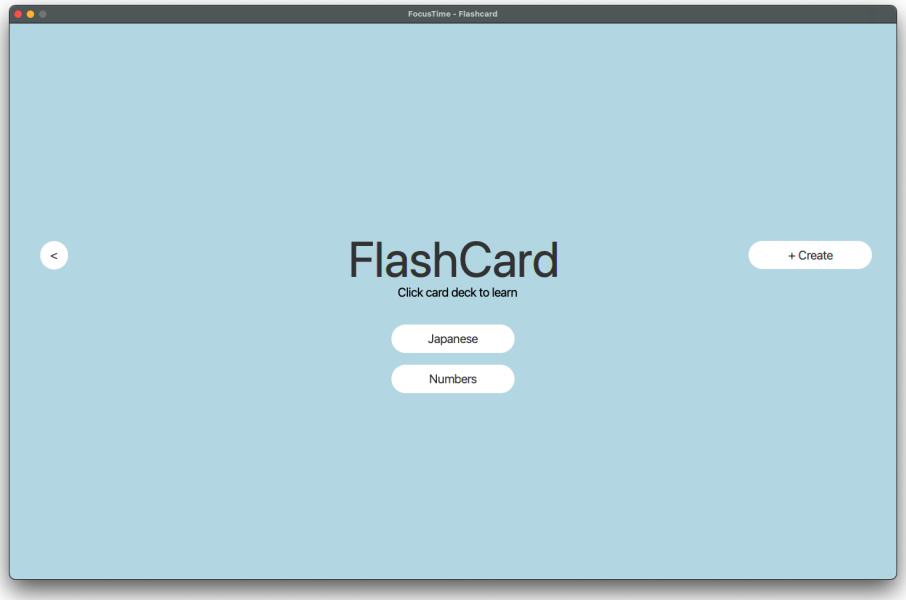
4. Name the timer and start button to start the timer



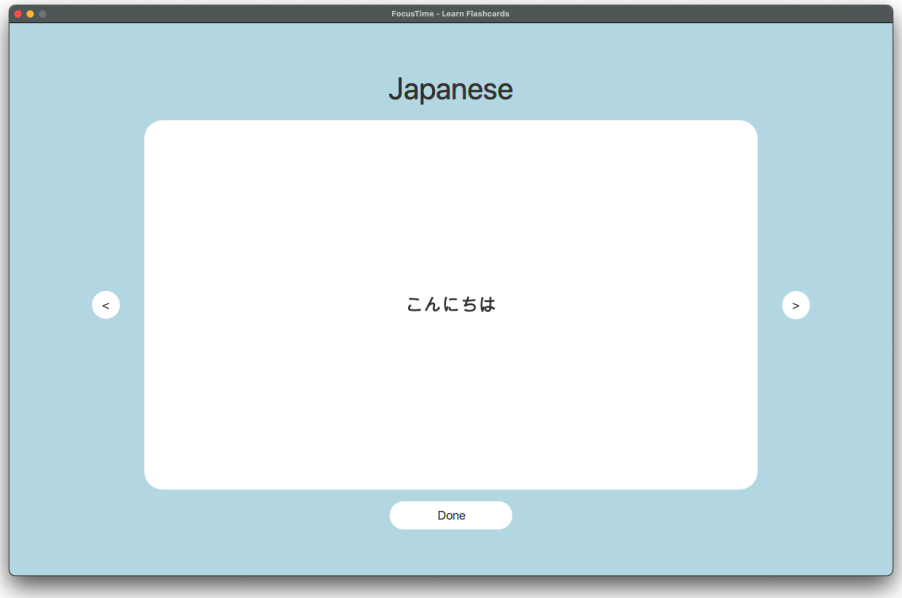
5. The pause button to pause the timer, when clicked, shows the timer status. The reset button will set the timer to 0.



6. The flashCard button will bring the user to the FlashCard library.

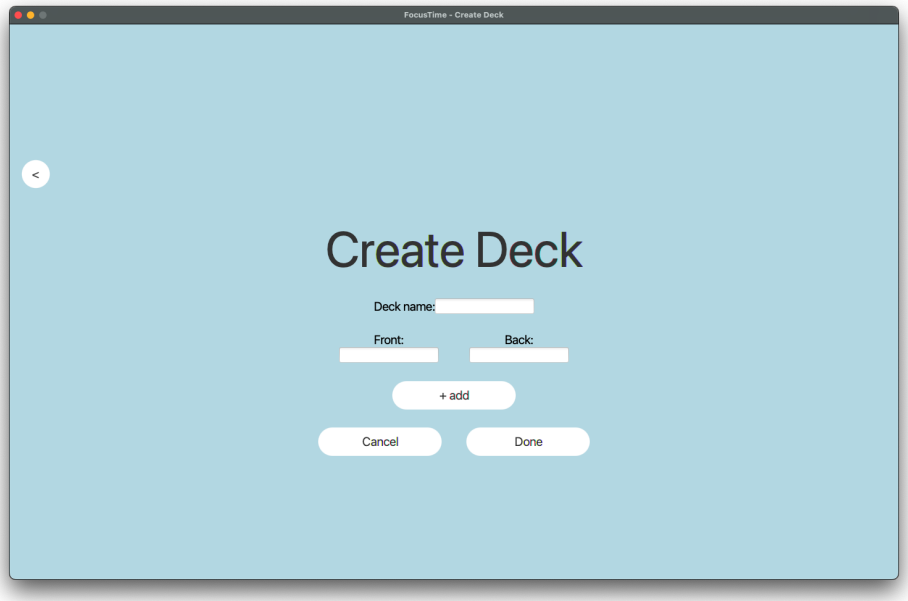


7. When the user clicks one of the decks, will bring them to the learning screen.

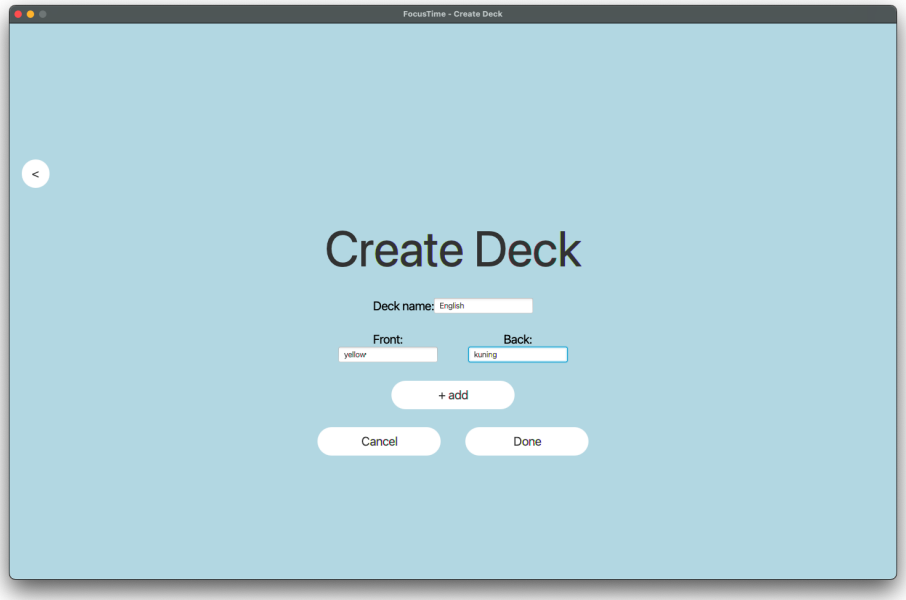


8. The previous and next buttons will iterate through the deck. When clicking the card in the middle, it will flip it and reveal the backside.

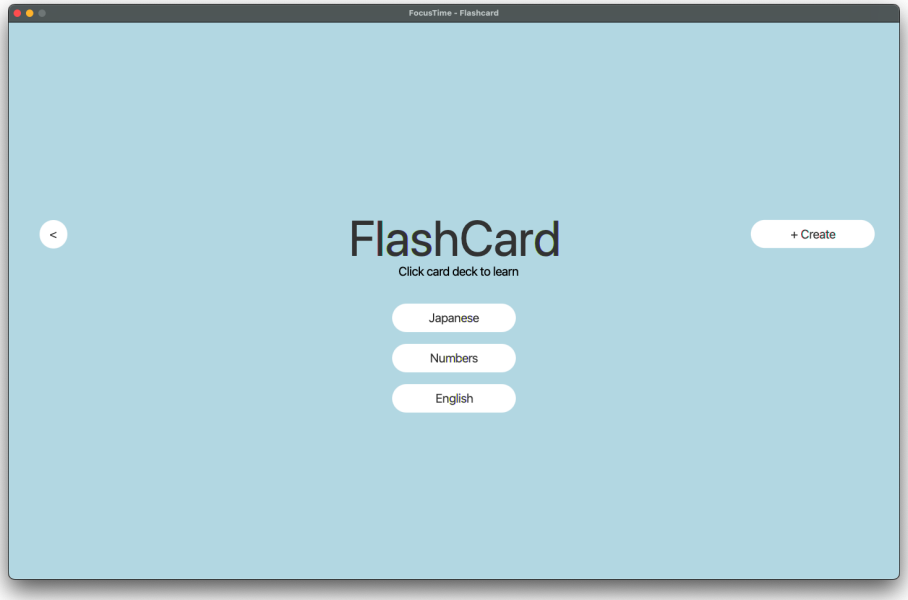
9. Done button will bring the user back to the main menu.
10. The create button in the flashcard screen allows the user to make their own deck. When clicked brings the user to the create page.



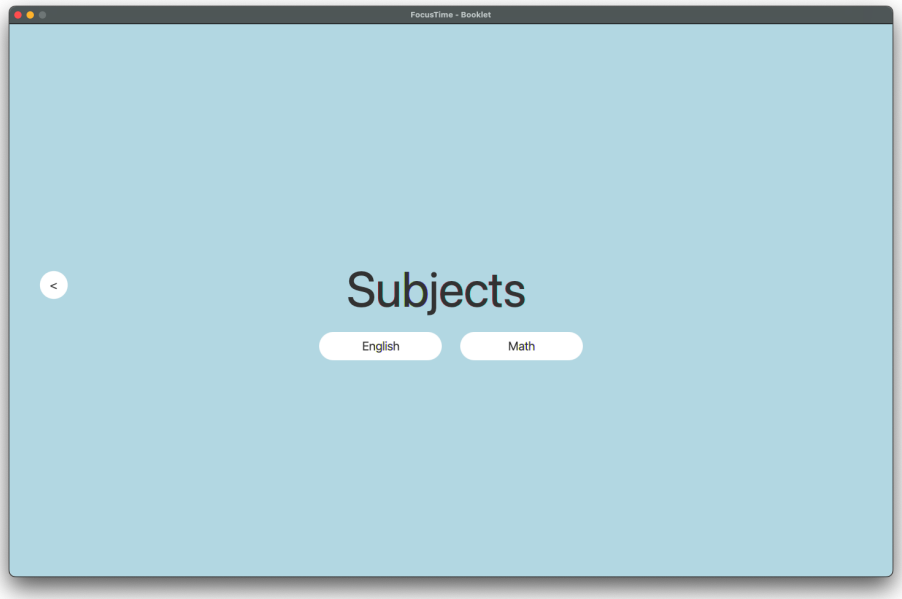
11. Users can input the name of the deck and add cards using the add button.



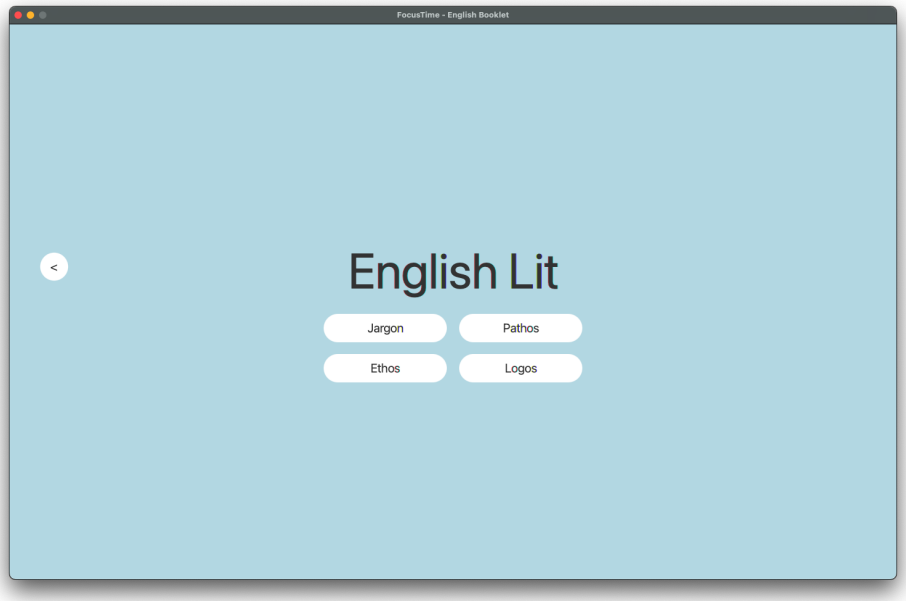
12. When done, click the done button and the new deck will be saved in a JSON file and displayed in the library.

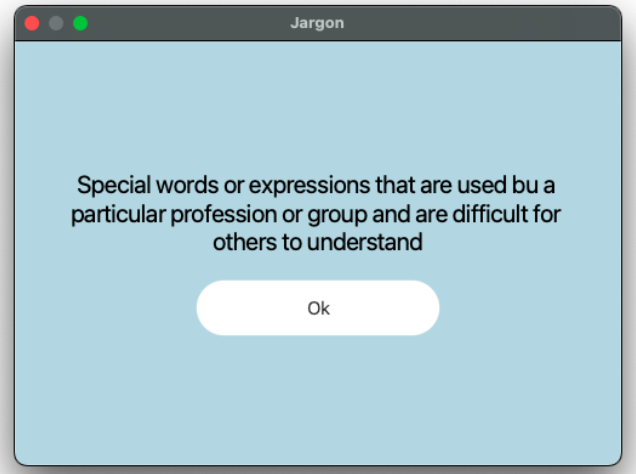


13. The booklet button will bring the user to a booklet section. Where it displays English and Math subjects.

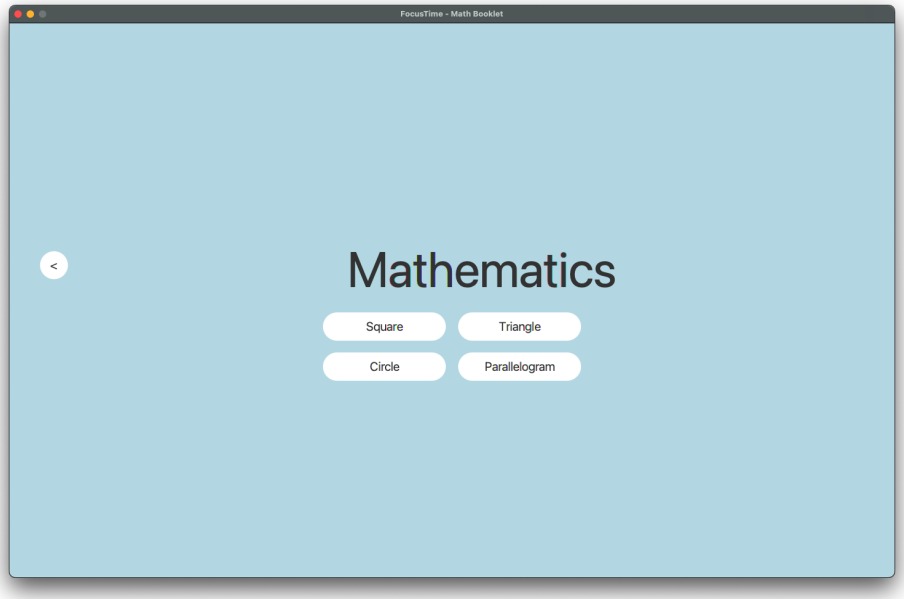


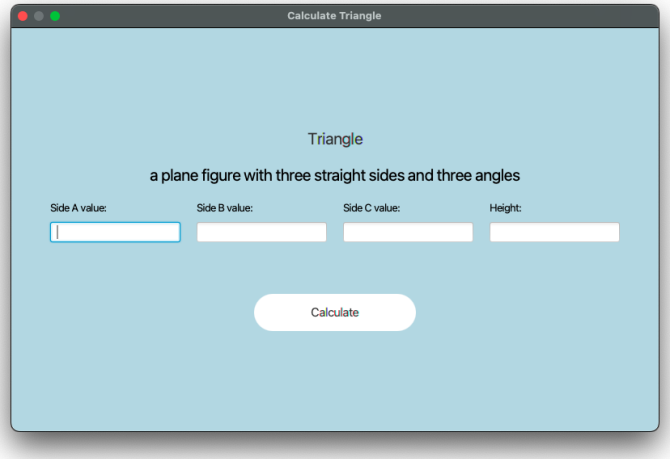
14. English button will bring the user to the English booklet. When a word is clicked, it will display the definition. In this case, literary device, jargon.



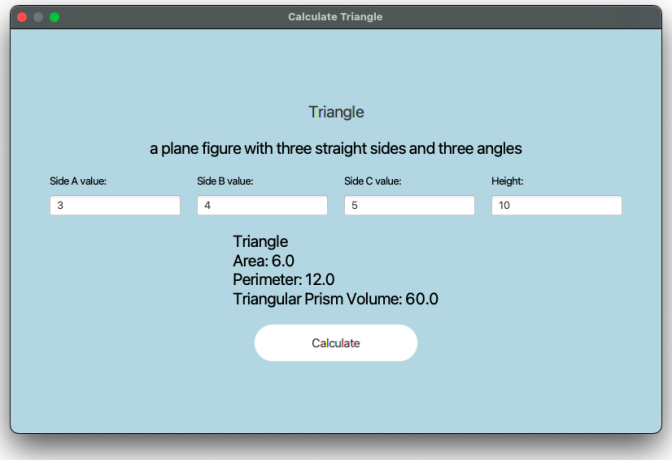


15. For Math subjects, users can pick a topic to see the description and use the calculator.





16. Users can fill in the box and click the calculate button to get the result.


Calculate Triangle
Triangle
a plane figure with three straight sides and three angles
Side A value:
3
Side B value:
4
Side C value:
5
Height:
10
Triangle
Area: 6.0
Perimeter: 12.0
Triangular Prism Volume: 60.0
Calculate

Resources

JavaFX tutorial video
https://youtube.com/playlist?list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG
Java documentation
https://docs.oracle.com/javase/7/docs/api/
NumberFormatException
https://rollbar.com/blog/java-numberformatexception/#:~:text=The%20NumberFormatException%20is%20an%20unchecked,(e.g.%20int%2C%20float).
JavaFX overview
https://jenkov.com/tutorials/javafx/overview.html
Iterator in Java
https://www.tutorialspoint.com/java/java_using_iterator.htm

Video Demo

https://youtu.be/iwWe9vjKtbE

Github Repo

https://github.com/steph45607/Steph_FocusTime.git