Summative Task: Stock Ticker

In this project, I will be creating a Stock Ticker which will implement a variety of plug-ins/libraries which will show you the rates of stocks as they increase or decrease. This code will show the most popular stocks at the moment and the best you should invest in as we can look into the pricing and the percentage of increase within the last 3 days. This code will be an excellent tool for those who wish to invest into stocks and I will be creating this for my brother, who also invests into stocks.

Team Members:

Throughout this project I will be working alone. My roles in this project would be to create the planning, create the code and edit the entire project. As doing this project alone may be challenging, I fully think I am capable of doing so as I have done it last year and this is also a great learning curve for me for coding.

Requirements:

Functional Requirements:

- Ability to track multiple stock symbols.
- Generate random stock prices for each symbol at regular intervals.
- Display the stock prices in a user-friendly format (e.g., GUI or console).
- Allow the program to run for a specified duration or until the user decides to stop.
- Allow the user to add or remove stock symbols dynamically.
- Integrate with an external API to fetch real-time stock prices.
- O Save stock prices to a database or file for future reference.

Technical Requirements:

- Stock Price Generation: Implement a mechanism to generate random stock prices or integrate with an external API to fetch real-time data.
- Timer or Scheduler: Use a timer or scheduler mechanism to update stock prices at regular intervals.
- User Interface: Determine the type of user interface you want to display the stock prices (e.g., GUI, console output).
- Libraries or Frameworks: Identify and choose any necessary libraries or frameworks for tasks such as GUI development, API integration, or database interactions.

Data Considerations:

- Stock Symbol Data: Decide how you will obtain and store the stock symbols you want to track (e.g., hard-coded, user input, database).
- Historical Data: Determine if you need to access and display historical stock prices, and consider how you will store and retrieve this data if necessary.

Usability and User Experience:

- Ensure the user interface is intuitive, easy to navigate, and visually appealing.
- Provide clear instructions or help options for users to understand the functionality of the stock ticker.
- Consider implementing error handling and informative error messages for any potential issues or failures.

Testing and Validation:

- Implement testing strategies to validate the functionality and reliability of the stock ticker code.
- Perform both unit testing (testing individual components) and integration testing (testing the interaction between components).
- Consider edge cases and error scenarios to ensure proper handling and graceful degradation.

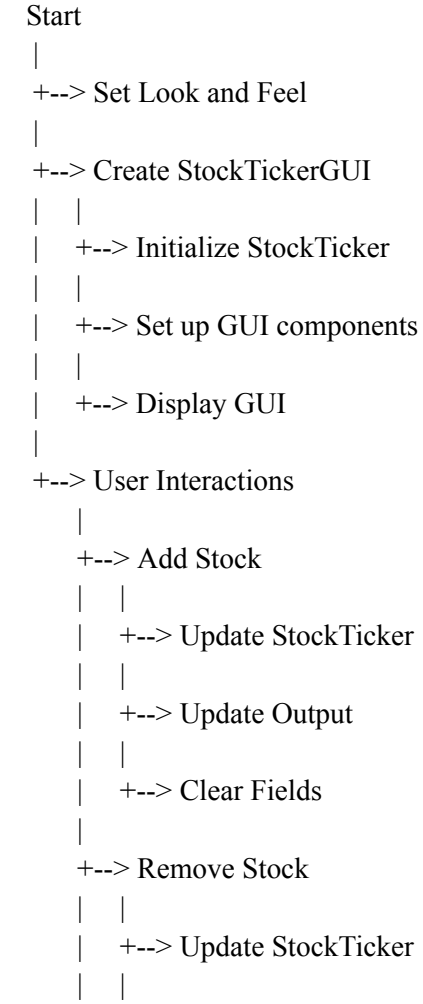
Documentation:

- Document the project requirements, design decisions, and code structure.
- Provide clear and concise instructions on how to set up and run the stock ticker code.
- Document any external APIs used, including authentication or access requirements.

General Algorithm:

1. Start the program.
2. Create an instance of the StockTickerGUI class.
3. Initialize the StockTicker object within the StockTickerGUI class
4. Set up the GUI components, including the main frame, panels, labels, text areas, buttons, and their respective event listeners..
5. Display the GUI to the user.
6. Display the stock prices in a graphical user interface (GUI) or a console output, showing the stock symbol and the current price.
7. When the user interacts with the GUI, the corresponding event listeners are triggered, executing the following actions:
8. When the "Add Stock" button is clicked:
    a. Retrieve the symbol entered in the symbol text field.
    b. Create a new Stock object with the symbol and default values for name and price.
    c. Add the stock to the StockTicker object.
    d. Update the output text area to display the updated stock list.
    e. Clear the symbol text field.
9. When the "Remove Stock" button is clicked:
    a. Retrieve the symbol entered in the symbol text field.
    b. Remove the stock with the corresponding symbol from the StockTicker object.

    c. Update the output text area to display the updated stock list.
    d. Clear the symbol text field.
10. When the "Update Stock Prices" button is clicked:
    a. Update the prices of all stocks in the StockTicker object with random changes.
    b. Update the output text area to display the updated stock list.
11. When the "Sort by Price" button is clicked:
    a. Sort the stocks in the StockTicker object by price in ascending order.
    b. If multiple stocks have the same price, sort them by symbol in alphabetical order.
    c. Update the output text area to display the sorted stock list.
12. When the "Save to File" button is clicked:
    a. Open a file chooser dialog for the user to select a file to save the stocks.
    b. If a file is selected, save the stocks from the StockTicker object to the file.
13. When the "Load from File" button is clicked:
    a. Open a file chooser dialog for the user to select a file to load stocks from.
    b. If a file is selected, load the stocks from the file into the StockTicker object.
    c. Update the output text area to display the loaded stock list.
14. The program continues to listen for user interactions until it is closed by the user.



```
|    +--> Update Output
|    |
|    +--> Clear Fields
|
+--> Update Stock Prices
|    |
|    +--> Update StockTicker
|    |
|    +--> Update Output
|
+--> Sort by Price
|    |
|    +--> Sort StockTicker
|    |
|    +--> Update Output
|
+--> Save to File
|
```

Prototype:

1) Title: The top panel displays the title "Stock Ticker."

2) Stock List: The central panel contains a scrollable text area where the list of stocks will be displayed. Initially, the area is empty.

3) Add Stock: The form panel at the bottom allows users to add a stock to the ticker. The "Symbol" label is followed by a text field where the user can enter the stock symbol. Clicking the "Add Stock" button will add the stock to the ticker and update the stock list.

4) Remove Stock: Users can remove a stock by entering its symbol in the text field and clicking the "Remove Stock" button. The stock will be removed from the ticker and the stock list will be updated.

5) Update Stock Prices: Clicking the "Update Stock Prices" button will simulate random price changes for all the stocks in the ticker. The stock list will be updated with the new prices.

6) Sort by Price: Clicking the "Sort by Price" button will sort the stocks in the ticker based on their prices in ascending order. If multiple stocks have the same price, they will be sorted based on their symbols. The stock list will be updated accordingly.

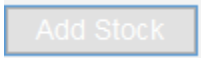
7) Save to File: Clicking the "Save to File" button will prompt the user to choose a file location to save the stock information. The stocks will be saved in a comma-separated format (CSV) with symbol, name, and price fields.

8) Load from File: Clicking the "Load from File" button will prompt the user to choose a file containing stock information in CSV format. The stocks will be loaded from the file, replacing the existing stocks in the ticker. The stock list will be updated accordingly.

User: Adds a stock

User: Symbol: AAPL

User: [Clicks the "Add Stock" button]



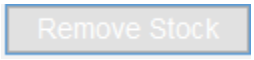
Stock Ticker with Added Stock

The stock with the symbol "AAPL" is successfully added to the ticker, and the stock list is updated to display the stock information.

User: Removes a stock

User: Symbol: GOOGL

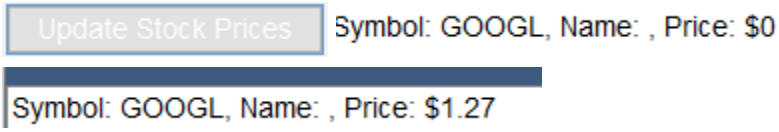
User: [Clicks the "Remove Stock" button]



Stock Ticker with Removed Stock

The stock with the symbol "GOOGL" is successfully removed from the ticker, and the stock list is updated to reflect the change.

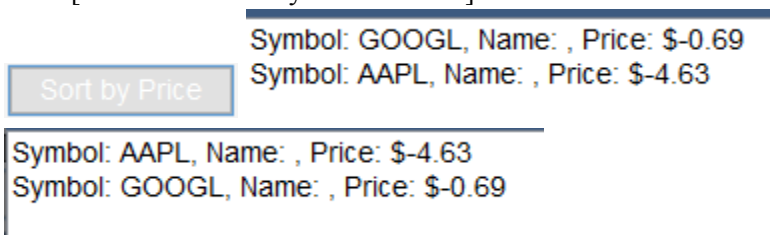
User: [Clicks the "Update Stock Prices" button]



Stock Ticker with Updated Stock Prices

The stock prices are randomly updated, and the stock list is updated with the new prices.

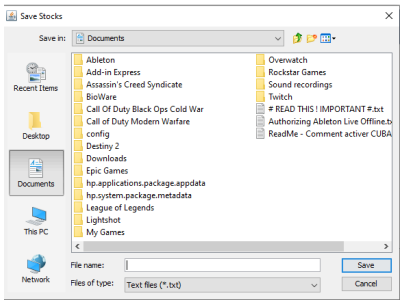
User: [Clicks the "Sort by Price" button]



Stock Ticker with Sorted Stock List

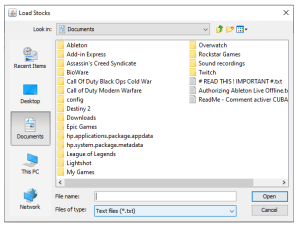
The stocks in the ticker are sorted based on their prices in ascending order. The stock list is updated to reflect the sorted order.

User: [Clicks the "Save to File" button and selects a file location]



The stocks are successfully saved to the specified file location.

User: [Clicks the "Load from File" button and selects a file containing stock information]



The stocks are successfully loaded from the file, replacing the existing stocks in the ticker. The stock list is updated to

Pseudocode 1:

```
class Stock:
    attributes:
        symbol
        name
        price

    constructor(symbol, name, price):
        initialize symbol, name, and price attributes

    getters and setters for symbol, name, and price

    toString():
        format and return the stock details as a string
```

Pseudocode 2:

```
class StockTicker:
    attributes:
        stocks (list of Stock objects)

    constructor():
        initialize an empty list of stocks

    addStock(stock):
```

```
    add the given stock to the list of stocks
    print "Stock added successfully"

  removeStock(stock):
    remove the given stock from the list of stocks
    print "Stock removed successfully"

  updateStockPrice():
    for each stock in stocks:
      generate a random price change between -5 and 5
      update the stock's price with the new price
    print "Stock prices updated successfully"

  sortByPrice():
    sort the stocks list by price in ascending order
    if two stocks have the same price, sort them by symbol

  saveStocksToFile(fileName):
    try:
      open the file with the given fileName for writing
      for each stock in stocks:
        write the stock details (symbol, name, price) to the file
      print "Stocks saved to file successfully"
    except IOException:
      print "Error saving stocks to file"

  loadStocksFromFile(fileName):
    try:
      clear the stocks list
      open the file with the given fileName for reading
      for each line in the file:
        split the line into symbol, name, and price parts
        create a new Stock object with the parts
        add the stock to the stocks list
      print "Stocks loaded from file successfully"
    except IOException:
      print "Error loading stocks from file"

  getStocks():
    return the list of stocks
```

Pseudocode 3:

```
class StockTickerGUI:
  attributes:
    stockTicker (an instance of StockTicker class)
    outputTextArea
    symbolTextField

  constructor():
    initialize the stockTicker attribute
```

```
    setup the user interface

  setupUI():
    set the window title, layout, and dimensions

    create a header panel with a title label

    create a stock list panel with a scrollable text area

    create a form panel with input fields and buttons

    add the panels to the main frame

    set action listeners for buttons to perform respective operations

    display the GUI

  updateOutput():
    clear the outputTextArea
    retrieve the list of stocks from stockTicker
    for each stock in the list:
      append the stock details to the outputTextArea

  clearFields():
    clear the symbolTextField
```

Pseudocode 4:

```
main():
  set the look and feel of the GUI to the system's default

  create a new instance of StockTickerGUI and display it
```

Summative Daily Calendar

DATE	TASK ACCOMPLISHED	Log
May 16, 2023	Selected Idea for summative.	✔
May 20, 2023	Start of Planning (Start Introduction)	✔
May 23, 2023	Start of Planning (Finish Introduction)	✔
May 27, 2023	Create general algorithm and start pseudocode	✔
May 29, 2023	Finish general algorithm	✔
May 31, 2023	Work on pseudocode and start the prototype	✔
Jun 1, 2023	Finish prototype and continue pseudocode	✔
Jun 2, 2023	Finish Pseudocode and Start UML Diagrams	✔
Jun 3, 2023	Continue UML Diagrams and Edit Prototype	✔
Jun 6, 2023	Finish Editing Prototype and Continue UML Diagrams	✔
Jun 9, 2023	Finish UML diagrams	✔
Jun 10, 2023	Finish Editing all of The Plan then Begin Code	✔
Jun 11, 2023	Continue Code	✔
Jun 13, 2023	Continue Code	✔
Jun 15, 2023	Finish Code	✔
Jun 19, 2023	Prepare for In Class Presentation	✔