Final Group Project: Python

EGC 112 - Programming 1B

International Institute of Information Technology – Bangalore

Submission: 6th December, 2024, EoD

Instructions: This is a group project where one group will comprise of 5 students. The groups will be decided, and projects will be assigned randomly by the TAs. Each group will be led by a TA who will be your contact person in case you face any issues while building this project. The project should be submitted in a zip folder with the name of the folder as Your_Group_No.zip. You need to submit the zip folder in LMS by the given deadline. The demo session of your group project will be held on 7th December, 2024. The schedule will be out soon.

List of the Group Projects

1)E-commerce Cart System

- **Description**: Design an e-commerce shopping cart system that manages different types of items, like `Electronics`, `Clothing`, and `Groceries`. Each item type should inherit from a base `Product` class, which contains attributes common to all items, such as `name`, `price`, `quantity`, and `product_id`. Each subclass will have unique characteristics based on the product category—`Electronics` might track warranty period, `Clothing` could include size and color, and `Groceries` might keep expiration dates.
- **Requirements**:
  - **Cart**: The main `Cart` class will manage items, allowing users to add, remove, and view items in their cart.
  - **Checkout**: Implement a `Checkout` feature that calculates the total price, applies any applicable discounts, and generates a receipt.
  - **Payment Processing**: Although it won't involve actual payment integration, students can simulate payment status (e.g., pending, completed) and include basic payment details.
- **File Handling**:
  - All items added to the cart should be saved to a file to allow users to retrieve their cart even if the application is restarted.

    - A `receipts` file can save a summary of each completed transaction, including details like itemized prices, discounts applied, and payment status. This file could simulate a history log for purchases.
- **Inheritance**:
    - **Base `Product` Class**: The base class will hold attributes like `name`, `price`, `product_id`, and `quantity`.
        - **Subclass Attributes**:
            - `Electronics` could include attributes like `warranty_period`.
            - `Clothing` could store `size` and `color`.
            - `Groceries` might track `expiration_date`.
- **Expected Functionalities**:
    - **Add/Remove Items**: Students should implement methods to add items to the cart and remove specific items.
    - **Calculate Total**: A method in the `Cart` class should calculate the total cost, applying any applicable discounts.
    - **Generate Receipt**: Upon checkout, a method should generate a receipt saved to a file, listing each item, its price, quantity, and the total amount.
- **Instructions for Students**:
    - **Focus on Inheritance**: Each product category should inherit from the base `Product` class, adding relevant attributes and behaviors.
    - **File Handling**: Implement file reading and writing to save cart state and purchase history.

Define Methods Independently: Students should brainstorm and implement specific methods, such as `calculate_discount` or `track_warranty`, that suit each category's needs. Also make use of static variables and methods wherever it is logically correct.

2)Online Quiz Management System

- **Description**: Create an online quiz management system that allows users to create, manage, and take quizzes. The system should have a base class Quiz that includes common attributes such as title, description, and questions. Subclasses can represent different quiz types, such as MultipleChoiceQuiz, TrueFalseQuiz, and ShortAnswerQuiz, each with unique question handling and scoring methods.

Requirements:

- **Quiz Creation and Management**: The main QuizManager class will handle quiz creation, allowing users to add questions and define quiz settings (e.g., time limits, scoring rules).
- **User Participation**: Implement user registration, allowing users to take quizzes and track their scores.
- **Results Analysis**: Provide users with feedback on their performance after taking a quiz, including score breakdowns and correct answers.

File Handling:

- Save quizzes to a file so that users can load them for future sessions.
- Store user results and performance statistics in a separate file to track progress over time.

Inheritance:

- **Base Quiz Class**: Contains attributes like title, description, and a list of questions.
    - **Subclass Attributes**:
        - MultipleChoiceQuiz: Could include options for each question.
        - TrueFalseQuiz: Might store true/false statements.
        - ShortAnswerQuiz: Could track correct answers as strings.

Expected Functionalities:

- **Add/Remove Questions**: Implement methods to add and remove questions from quizzes.
- **Take Quiz**: Users can start a quiz, answer questions, and submit their responses for scoring.
- **Generate Report**: After quiz completion, generate a report saved to a file that includes user performance and correct answers.

Instructions for Students:

- **Focus on Inheritance**: Each quiz type should inherit from the base Quiz class, with unique attributes and behaviors.
- **File Handling**: Implement file reading and writing to manage quizzes and user results.

Define Methods Independently: Students should come up with their own methods to manage quizzes, track user performance, and analyze results.

3)Music Library Management System

- **Description**: Create a music library management system that allows users to manage a collection of songs, albums, and artists. The base class `Song` will contain attributes such as `song_id`, `title`, `artist`, `album`, and `duration`. Subclasses can represent different genres, like `PopSong`, `RockSong`, and `ClassicalSong`, each with unique characteristics.
- **Requirements**:
  - **Music Collection Management**: The `MusicLibraryManager` class will manage the collection of songs, allowing users to add, update, and remove songs from the library.
  - **Playlist Creation**: Implement functionality for users to create and manage playlists based on their favorite songs.
  - **Search and Filter**: Include features to search for songs by artist, album, or genre.
- **File Handling**:
  - Save song and album details to files for persistence.
  - Maintain a playlist file that logs user-created playlists.
- **Inheritance and Static Methods**:
  - **Base `Song` Class**: Contains attributes like `song_id`, `title`, `artist`, `album`, and `duration`.
    - **Subclass Attributes**:
      - `PopSong`: May include `chart_position` and `release_year`.
      - `RockSong`: Could track `band_name` and `album_year`.
      - `ClassicalSong`: Might include `composer` and `orchestra`.
- **Expected Functionalities**:
  - **Add/Update/Delete Songs**: Implement methods for managing songs in the library.
  - **Manage Playlists**: Allow users to create and modify their playlists.
  - **Generate Music Report**: Provide an option to generate a report of songs in the library, saved to a file.
- **Instructions for Students**:

    - **Focus on Inheritance**: Each song genre should inherit from the base `Song` class, allowing for specific attributes

4)Hotel Reservation System

- **Description**: Develop a hotel reservation system that allows users to book rooms, manage bookings, and check availability. The base class `Room` will contain attributes like `room_id`, `type`, `price`, and `availability_status`. Subclasses can represent different room types, such as `SingleRoom`, `DoubleRoom`, and `Suite`, each with specific features.
- **Requirements**:
    - **Room Management**: The `HotelManager` class will manage the collection of rooms, allowing users to add, update, and remove rooms from the system.
    - **Booking Management**: Implement functionality for users to make bookings, modify reservations, and check in/out.
    - **Availability Tracking**: Include features to check room availability and generate booking reports.
- **File Handling**:
    - Save room details and booking records to files for persistence.
    - Maintain a booking history file that logs all reservations made through the hotel.
- **Inheritance:**
    - **Base `Room` Class**: Contains attributes like `room_id`, `type`, `price`, and `availability_status`.
        - **Subclass Attributes**:
            - `SingleRoom`: Could include `bed_size`.
            - `DoubleRoom`: May track `view_type`.
            - `Suite`: Might include `number_of_bedrooms` and `amenities`.
- **Expected Functionalities**:
    - **Add/Update/Delete Rooms**: Implement methods for managing rooms in the hotel.
    - **Manage Bookings**: Allow users to make, modify, and cancel bookings.
    - **Generate Booking Report**: Provide an option to generate a report of bookings made within a specific period, saved to a file.
- **Instructions for Students**:

    - **Focus on Inheritance**: Each room type should inherit from the base `Room` class, allowing for specific attributes and methods.
    - **File Handling**: Implement file reading and writing to manage rooms and bookings.

Define Methods Independently: Encourage students to brainstorm methods that suit their hotel management needs, such as `check_availability` or `get_room_details`.

5)Travel Booking System

- **Description**: Develop a travel booking system that allows users to search for and book flights, hotels, and rental cars. The base class `TravelPackage` will contain attributes like `package_id`, `destination`, `duration`, `price`, and `availability_status`. Subclasses can represent different types of packages, such as `Flight`, `Hotel`, and `CarRental`, each with specific features.
- **Requirements**:
    - **Package Management**: The `TravelManager` class will manage the collection of travel packages, allowing users to add, update, and delete packages.
    - **Booking Management**: Implement functionality for users to book travel packages and manage their itineraries.
    - **Availability Tracking**: Include features to check package availability and generate booking confirmations.
- **File Handling**:
    - Save package details and booking records to files for persistence.
    - Maintain a transaction history file that logs all bookings made through the system.
- **Inheritance:**
    - **Base `TravelPackage` Class**: Contains attributes like `package_id`, `destination`, `duration`, `price`, and `availability_status`.
        - **Subclass Attributes**:
            - `Flight`: May include `airline` and `departure_time`.
            - `Hotel`: Could track `hotel_name` and `rating`.
            - `CarRental`: Might include `car_model` and `rental_company`.

- **Expected Functionalities**:
    - **Add/Update/Delete Packages**: Implement methods for managing travel packages.
    - **Manage Bookings**: Allow users to book packages and manage their itineraries.
    - **Generate Booking Report**: Provide an option to generate a report of all bookings, saved to a file.
- **Instructions for Students**:
    - **Focus on Inheritance**: Each package type should inherit from the base `TravelPackage` class, allowing for specific attributes and methods.
    - **File Handling**: Implement file reading and writing to manage packages and booking data.

Define Methods Independently: Encourage students to think critically about the methods needed for travel management, like `get_package_details`

6)Home Automation System

- **Description**: Create a home automation system that allows users to control various devices in their homes. The base class `Device` will contain attributes like `device_id`, `name`, `status`, and `type`. Subclasses can represent different types of devices, such as `Light`, `Thermostat`, and `SecurityCamera`, each with unique features.
- **Requirements**:
    - **Device Management**: The `HomeManager` class will manage the collection of devices, allowing users to add, update, and control devices.
    - **Automation Rules**: Implement functionality for users to set automation rules for devices (e.g., turn on lights at sunset).
    - **Device Status Reports**: Include features to generate reports on device statuses.
- **File Handling**:
    - Save device details and automation rules to files for persistence.
    - Maintain a log file that records all device activities.
- **Inheritance and Static Methods**:

    - **Base `Device` Class**: Contains attributes like `device_id`, `name`, `status`, and `type`.
        - **Subclass Attributes**:
            - `Light`: May include `brightness` and `color`.
            - `Thermostat`: Could track `temperature` and `mode`.
            - `SecurityCamera`: Might include `resolution` and `recording_status`.
- **Expected Functionalities**:
    - **Add/Update/Control Devices**: Implement methods for managing devices.
    - **Set Automation Rules**: Allow users to create automation rules for device control.
    - **Generate Device Status Report**: Provide an option to generate a report of device statuses, saved to a file.
- **Instructions for Students**:
    - **Focus on Inheritance**: Each device type should inherit from the base `Device` class, adding specific attributes and methods.
    - **File Handling**: Implement file reading and writing to manage devices and automation rules.
    - **Define Methods Independently**: Encourage students to think critically about the methods needed for device management, such as `get_device_details` or `execute_automation_rule`.

7)Budget Planner System

Description: Design a budget planner system that allows users to manage their monthly finances by tracking income, expenses, and savings goals. The system will categorize different types of financial transactions and generate reports on spending and savings over time. Each transaction type should inherit from a base `Transaction` class, containing attributes common to all transactions, such as amount, date, and category. Subclasses will represent different transaction types, such as `Income`, `Expense`, and `Savings`, each with unique attributes.

Requirements:

1. **Budget Management**: The main `BudgetPlanner` class will manage all financial transactions, allowing users to add, edit, and delete transactions, view their budget summary, and track remaining funds.

2. **Spending Categories**: Provide various expense categories like `Groceries`, `Bills`, `Entertainment`, and `Transportation` for better organization.
3. **Savings Goals**: Allow users to set savings goals for the month and track their progress towards achieving them.
4. **Monthly Reports**: Generate monthly reports showing income, total expenses by category, and amount saved. Reports should be saved to a file for later review.

File Handling:

- **Transaction History**: All transactions should be saved to a file so users can retrieve their financial history even if the application is restarted.
- **Monthly Report**: Each report file will save a summary of the month's income, expenses by category, and any savings, allowing users to track financial progress over time.

Inheritance:

- **Base `Transaction` Class**: The base class will contain attributes like `amount`, `date`, and `category`.
- **Subclass Attributes**:
  - **Income**: Could include attributes like `source` (e.g., salary, freelance).
  - **Expense**: Might track `expense_type` (e.g., essential, non-essential).
  - **Savings**: Could include `goal_name` and `target_amount`.

Expected Functionalities:

- **Add/Edit/Delete Transactions**: Students should implement methods to add new transactions, update existing ones, and delete specific entries.
- **Calculate Budget Summary**: A method in the `BudgetPlanner` class should calculate the total income, total expenses, and remaining balance each month.
- **Generate Monthly Report**: A method to generate a report saved to a file, detailing income, categorized expenses, and savings progress.

Instructions for Students:

- **Focus on Inheritance**: Each transaction type should inherit from the base `Transaction` class, with attributes and behaviors suited to income, expenses, or savings.

- **File Handling**: Implement file reading and writing to save transaction data and monthly reports.

Define Methods Independently: Students should brainstorm and implement methods, such as `track_goal_progress` or `calculate_expense_percentage`, that suit

budget tracking needs. Use static methods where it's appropriate, like calculating averages across all budget months.

8)Personal Diary Application

Description: Create a personal diary application that allows users to write and categorize diary entries. The application will support various entry types (like daily reflections, event notes, and mood logs), each inheriting from a base `DiaryEntry` class. This system will allow users to search entries by date, emotion, or keywords, and view summaries of past entries. The diary entries will be stored to provide users with a continuous record of their thoughts and experiences.

Requirements:

1. **Diary Management**: The main `DiaryManager` class will allow users to add, edit, and delete diary entries. It will also support searching and filtering based on criteria like date or emotion.
2. **Entry Categorization**: Organize entries by different categories, such as `DailyReflection`, `EventNote`, and `MoodLog`, each with distinct attributes.
3. **Entry Summarization**: Allow users to view summaries of their entries over specific time periods (e.g., weekly or monthly) and show trends in mood or frequent themes.
4. **Mood Tracking**: Incorporate mood tags or emoji-based ratings to allow users to express their emotions for each entry.

File Handling:

- **Diary Entries File**: Save each entry to a file so users can retrieve their entries across sessions. Each entry will include the date, type, and content.
- **Summary Reports**: Generate summary reports by week or month and save them to a file for future reference. Reports could include frequent themes, average mood scores, or notable events.

Inheritance:

- **Base `DiaryEntry` Class**: The base class will contain attributes such as `date`, `content`, and `tags`.
- **Subclass Attributes**:
    - **DailyReflection**: Could include attributes like `gratitude` notes or `challenges`.
    - **EventNote**: Might track `location` and `attendees`.
    - **MoodLog**: Could include `mood_rating` and `emotion_tags`.

Expected Functionalities:

- **Add/Edit/Delete Entries**: Implement methods for creating new entries, updating existing entries, and deleting specific ones.
- **Search and Filter Entries**: Allow users to search entries by keywords, filter by date, or sort by mood tags.
- **Generate Summary Report**: After a specified period (like a week or month), generate a report saved to a file, which includes entry counts, recurring themes, and average mood data.

Instructions for Students:

- **Focus on Inheritance**: Each entry type should inherit from the base `DiaryEntry` class, with unique attributes to suit daily reflections, events, or mood tracking.
- **File Handling**: Implement file reading and writing to save diary entries and summary reports.

Define Methods Independently: Students should brainstorm and implement specific methods, such as `analyze_mood_trends` or `filter_by_emotion`, to enhance the diary's personalization. Use static methods when appropriate, like calculating the average mood across all entries for a given time frame.

9)Stock Market Tracker and Visualizer

Description: The Stock Market Tracker and Visualizer is an application that provides users with up-to-date stock information and historical performance insights for various companies. It enables users to track selected stocks, analyze historical trends, and compare multiple stocks over time. Users can easily monitor the stock market and visualize changes, helping them make informed decisions.

Requirements:

1. **Stock Data Tracking**: Users should be able to add and remove stocks they want to track, with each stock displaying relevant details such as the symbol, company name, and current market price.
2. **Daily Data Updates**: The application should update stock prices, volume, and other key market data daily, ensuring users have access to the latest information.
3. **Historical Performance Visualization**: Users should be able to view historical price trends for selected stocks through easy-to-read line charts. These visualizations should allow users to understand stock performance over different time periods (e.g., last week, month, year).
4. **Stock Comparison**: The application should enable users to select multiple stocks and display them on a single graph, allowing for a quick visual comparison of their price movements over time.
5. **Report Generation**: Users should have the option to generate and save a report that summarizes historical performance, current prices, and selected comparisons, allowing them to keep a record of market trends and their chosen stocks.
6. **Persistent Storage**: The application should save user preferences, including tracked stocks and saved reports, so users can continue from where they left off in future sessions.

Instructions for Students: (Suggestions)

Use Classes for Core Entities

- **Stock Class**: Create a Stock class to represent individual stocks, encapsulating attributes like symbol, company_name, price_history, and methods such as get_historical_data() to fetch past data, get_current_price() to update price, and calculate_price_change() to calculate metrics.
- **Tracker Class**: Use a StockTracker class to manage all stocks the user is tracking. This class would contain a list or dictionary of Stock instances and

methods for adding, removing, and updating stocks. It can also handle tasks like saving the list of tracked stocks to a file.

Separate Data Retrieval Logic

- **DataFetcher Module**: Isolate the web scraping or API calls in a DataFetcher class or module. This class would contain methods like fetch_stock_data(symbol) and fetch_historical_data(symbol, period), abstracting the specifics of data retrieval and making it easy to switch sources (e.g., from scraping to an API) without changing core application logic.
- By isolating this in a separate module, future developers can easily update or replace data retrieval without impacting other parts of the app.

Create Visualization and Report Modules

- **Visualization Module**: Write a Visualizer class that contains methods for rendering different types of graphs (e.g., plot_price_trend(stock_data) for line charts and plot_volume(stock_data) for bar charts). This keeps visualization code separate from core business logic, and allows easy updates or additions to chart types.
- **Report Module**: Have a ReportGenerator class to handle report creation and saving. It could include methods like generate_comparison_report() to compile comparison data and save it in various formats (CSV, JSON, or text).

Organize Code in Modules

- Create a clear directory structure, breaking out key components into separate modules. For instance:

```
stock_tracker/
├── data_fetcher.py       # Handles data retrieval
├── stock.py              # Defines Stock class
├── stock_tracker.py      # Manages multiple stocks
├── visualization.py      # Plots and visualizes stock data
├── report.py             # Generates and saves reports
```

```
└── main.py                # Application entry point
```

- This structure keeps each file focused on a single responsibility, making the codebase more readable and easier to test and maintain.

10) IMDb Movie Ratings and Trends

Description: The IMDb Movie Ratings and Trends application allows users to analyze movie data, track rating trends, and compare movies by genre, release year, or popularity. The app will scrape data like ratings, genres, and release dates from IMDb, enabling users to explore changes in ratings over time, compare movies within specific genres, and generate reports on top-rated movies. Each movie is represented by a Movie class that stores relevant information and enables trend analysis.

Requirements:

1. **Movie Data Tracking**: Users should be able to search for and add movies they want to analyze. Each movie should display details like title, release year, genre, and IMDb rating.
2. **Data Scraping**: The application should scrape IMDb to retrieve movie information (such as rating, release year, genre, and director). It should also allow users to fetch updated ratings periodically.
3. **Rating Trends Visualization**: Users should be able to view rating trends for selected movies or genres over different time frames. Line charts should display rating changes over time for individual movies, while bar charts can show average ratings across genres or directors.
4. **Genre and Yearly Comparison**: The app should allow users to filter movies by genre or release year and visualize trends, such as average ratings by genre or rating distribution by year.
5. **Report Generation**: Users should have the option to generate and save reports on selected movies or genres. Reports could include details on average ratings by genre, top-rated movies in a specific period, or rating trends over time.
6. **Persistent Storage**: The application should save user preferences and selected movies, so users can return to their saved data in future sessions.

Suggestions for Code Modularity and Extensibility

To make the code modular, extensible, and maintainable, follow these guidelines:

1. **Use Classes for Core Entities**:

   a. **Movie Class**: Create a Movie class that stores basic information like title, release_year, genre, and rating. This class can also include methods for calculating rating changes over time or checking if the movie meets specific criteria (e.g., top-rated within a genre).
   b. **MovieCollection Class**: Use a MovieCollection class to manage multiple Movie instances. This class can handle adding and removing movies, updating movie data, and retrieving lists of movies by genre or year.
2. **Separate Data Retrieval Logic**:
   a. **DataFetcher Module**: Isolate data scraping logic within a DataFetcher class or module, containing methods like fetch_movie_data(title) to retrieve data from IMDb. Separating this logic makes it easy to update data sources or API calls without affecting other parts of the application.
3. **Organize Code into Modules**:
   a. Set up a directory structure to keep each component of the app focused on a single responsibility:

```
movie_trends/

├── data_fetcher.py        # Handles IMDb data retrieval

├── movie.py               # Defines the Movie class

├── movie_collection.py    # Manages multiple movies

├── visualization.py       # Contains visualization logic

├── report.py              # Generates and saves reports

└── main.py                # Application entry point
```

4. Create Visualization and Report Modules:

- **Visualization Module**: Use a Visualizer class that contains methods to plot different graphs, like plot_rating_trends(movie_data) for line charts and plot_genre_comparison(genre_data) for bar charts. This separation keeps visualization code independent and easily extensible.

Report Module: Write a ReportGenerator class to compile and save movie analysis reports, such as top movies in a genre or average ratings by year.