Document for the OOP Project:

Our Team:

1. **Abdur Rafay Bhutta (Reg. no: 004378/BSSE/F24)**

- **GitHub:** https://github.com/AbdurRafay007/OOP-Project
- **LinkedIn:** www.linkedin.com/in/abdur-rafay-4b77952b0

2. Muzamilullah Mohibi (Reg. no: 007473/BSSE/F24)

- **GitHub:** https://github.com/Muzamilullah/business-budgetting-system
- **LinkedIn:** https://www.linkedin.com/in/muzamilullah-mohibi-74a4a033b/

3. Syed Saif Ali (Reg. no: 004335/BSSE/F24)

- **GitHub:** https://github.com/saifali512/Business-Management-System.git
- **LinkedIn:** https://www.linkedin.com/in/syed-saif-ali-494762354/

What did we use in the code:

1. **user Class Definition** (Abstract Base Class for authentication)

- Private members (email, password)
- Static member (userCount)
- Constructor
- Pure virtual function (displayDetails())
- existingUser() method
- signup() method
- login() method

2. Static Member Initialization (int user::userCount = 0;)

3. Business Class Definition (Inherits from user, business details)

- Private members (company info, capital)
- setData() method

- checkCapitalThreshold() method
- updateCapital() method
- Getter methods (getCompanyName(), etc.)
- Friend function declaration (compareInitialAndCurrentCapital)
- displayDetails() override

3. Function Definition (Friend function for Business)

4. Shares Class Definition (Base class for share information)

- Protected members (valuation, shares, price)
- setData() method
- Getter methods (getValuation(), etc.)

5. **Transactions Class Definition** (Manages income/expense records)

- Private nested struct record
- Private Income and Expense vectors
- addIncome() method
- displayIncome() method
- totalIncome() method
- addExpense() method
- displayExpense() method
- totalExpense() method

6. **Inheritance for ShareHolders Class Definition** (Inherits from Shares, manages shareholder details)

- Protected nested struct ShareHolderInfo
- Public holders vector
- addShareHolders() method

- totalShares() method
- getAllHolders() method
- displayShareHolders() method

7. **Inheritance for ProfitCalculator Class Definition** (Inherits from ShareHolders, calculates profit)

- Private members (profit values, tax)
- Constructor
- calculateProfit() method
- calculateProfitPerShareholder() method
- displayProfitPerShareholder() method
- Getter methods (getProfitBeforeTax(), etc.)

8. **Report Class Definition** (Virtual Inheritance from ProfitCalculator, generates reports)

- generateReport() method
- saveReportToFile() method

10. **main() Function** (Program entry point and main menu logic)

- Object Instantiation
- Login/Signup Loop
- Main Dashboard Loop (User choices and function calls based on choice)

Here is the detailed explanation for each segment:

The details will be in this format:
1. Why we used it ?
2. And how it works?

Section: user Class Definition

This section declares the basic user class, which is a central piece for user access management in our Business Management System. It's created as an abstract base class to make certain all user types (such as businesses) have common aspects of authentication.

user Class (Abstract Base Class for Authentication)

- Why we use it: We use the user class as a base to establish a common contract for all entities that need to log in or sign up. This allows us to handle user authentication centrally and ensures consistency across different user roles we might introduce later.
- How it works: It specifies the mandatory structure and general behaviors required from any user account, such as storing credentials and offering authentication methods. By rendering it abstract, we make sure that specific user types extend upon these basic features.

Private Members (email, password

- Why we use them: We store the email address and password of the user in these members, which are essential to identify and authenticate the users. Making them private is an elementary measure towards security to avoid direct, unauthorized modification or access outside the class.
- How they work: When a user registers, his/her email and password are stored within these private variables and then safely appended to a file. On login, the provided credentials are matched against these stored values to authenticate the user.

Static Member (userCount)

- Why we use it: userCount is used to keep track of how many user objects are created over the lifetime of the whole application. This might be helpful for stats or knowing how the system's being used without having to cycle through all current users.
- How it works: It is a single variable common to all user class instances (and derived classes). Each time a user object (or an object of a class inherited from user) is instantiated, its constructor increments this count.

Constructor (user())

- Why we use it: The constructor is tasked with creating new user objects upon creation. Here, its main job is to automatically increment the userCount.
- How it works: Each time user() is invoked (either explicitly or implicitly when a Business object is instantiated), it runs the code defined inside its body. In this case, it merely increments one to the shared userCount variable.

Pure Virtual Function (displayDetails())

- Why we use it: We made displayDetails() a pure virtual function to make sure that any class that extends user will have to implement its own individual method of showing its details. As there is

no specific "details" a generic user can display other than email/password (which we don't want). This compels the source classes to override what makes sense to them.

- How it works: The = 0 syntax declares the user class abstract, so you can't instantiate a direct user object. It is a placeholder or a contract that guarantees Business (and any other user type in the future) will have a displayDetails() method specific to their details.

existingUser() Method

- Why we use it: It is essential to avoid duplicate user signups. It rapidly scans whether an email address entered at the time of signup already exists in our users database (user.txt).
- How it works: It scans the user.txt file, matching each recorded email with the one being entered. Upon finding a match, it immediately indicates that the user has already signed up, avoiding the new signup process.

signup() Method

- Why we use it: The signup() function takes care of the entire signup process for a new user account. It walks the user through entering their email and password and then securely stores this data.
- How it works: It asks the user to input an email and password. It initially invokes existingUser() to search for duplicates. If the email is not already present, it then adds the new email and password to the user.txt file to ensure registration was a success.

login() Method

- Why we use it: The login() method offers the functionality for registered users to access the system. It checks their credentials against their stored records.
- How it works: It accepts the input of the user's email and password. It reads from the user.txt file, trying to find a corresponding email and password pair. If both match, it provides access; else, it tells the user about invalid credentials.

Section: Business Class Definition

The Business class is a core element of our system, which will store and manage all the business-specific information of a registered business.

Business Class (Inherits from user, business details)

- Why we use it: This class is a representation of a specific business in our system. Inheriting from user gives it the capability to deal with its own login and signup functionalities, which is an excellent example of code reuse.

- How it works: A Business object holds detailed attributes like company name, owner, capital, and employees. It extends the basic user functionality with business-specific data and operations, allowing us to manage company profiles separately.

Private Members (company info, capital)

- Why we use them: They hold all the sensitive business information that determines a business, e.g., companyName, ownerName, businessType, numberOfEmployees, initialCapital, currentCapital, companyID, and capitalThreshold. Declaring them private safeguards this confidential business information from being unintentionally or intentionally altered outside the class.
- How they work: Data is allocated to these variables mostly via the setData() method. After they are stored, the private members make up the entire profile of an enterprise in the system, which maintains its integrity and consistency.

setData() Method

- Why we use it: The setData() method offers an organized approach to entering all of the information pertinent to a business object. It encapsulates the process of collecting a number of individual pieces of information into one, convenient function.
- How it works: The method asks the user to input each bit of business data (owner name, owner, ID, etc.) separately. It then assigns those inputs to the respective private member variables of the Business class.

checkCapitalThreshold() Method

- Why we use it: This method serves as an in-built alert mechanism for the financial health of the business. Businesses need to check the level of capital.
- Functionality: It checks the currentCapital of the business against a set capitalThreshold. If the currentCapital is at or above this threshold, it outputs an alert message to the console.

updateCapital() Method

Why we use it: This method offers a controlled manner for altering the currentCapital of the business. This is necessary for showing financial inflows or outflows.

Works: It accepts a newCapital value as an argument and updates the currentCapital private member. Then, after updating, it calls checkCapitalThreshold() immediately to determine whether the new level of capital will prompt any alerts.

Getter Methods (getCompanyName(), etc.)

- Why we use them: Getter methods such as getCompanyName(), getOwnerName(), getCurrentCapital(), etc., offer secure and controlled access to the private data of the Business object. Although the data is private, other components of the program frequently need to read it.
- How they work: Every getter method just returns the value of its corresponding private member variable. They do not permit modification, hence data encapsulation and an inability to manipulate the business's internal state directly.

Friend Function Declaration (friend void compareInitialAndCurrentCapital(const Business &b);)

- Why we use it: We make compareInitialAndCurrentCapital a friend because it must be able to see the private initialCapital and currentCapital of a Business object in order to do its comparison. That enables a non-member function to have privileged access for a particular, associated purpose.
- Works: By being made a friend within the Business class, this member function has the rights to read privately the members (initialCapital, currentCapital) of any Business object it is passed, even though it itself is not a member of the Business class.

displayDetails() Override

- Why we use it: This approach gives a particular manner in which a Business object can display its individual specifics. It's an override because the inherited user class defined a pure virtual displayDetails(), compelling Business to overload it with its own implementation.
- Process: When displayDetails() is invoked on a Business object, it outputs all of the information that has been stored about the company, i.e., its name, owner, ID, business type, employees, and capital amounts.

Section: compareInitialAndCurrentCapital Function Declaration

This independent function is designed to assess the fiscal performance of a company by comparing its capital at two time points.

- Why we use it: This method offers a simple and concise financial health check for a company, whether it is profitable, loss-making, or steady. It is created as a non-member function to enable flexible analysis apart from the rigid methods of the Business class, but still requiring access to its fundamental financial information.
- Operation: It accepts a Business object (passed by constant reference to prevent redundant copying and alteration) and accesses directly its private initialCapital and currentCapital through

its friend declaration. It computes simple arithmetic to derive the difference and announces the business's status (profit, loss, or no change) and the precise amount.

Section: Shares Class Definition

The Shares class is the basic building block for the management of share-related information in our business system. It establishes the fundamental characteristics of a company's shares.

- Why we use it: This class consolidates all the essential data regarding a company's shares, such as its overall worth and amount of issued shares. It is a base class so that any component in the system handling shareholders or dividend distribution can always retrieve this underlying share information.
- Operation: It encapsulates the most important financial measures of shares and includes ways to fill and get this data. This structure supports reusability because other classes that deal with shares can inherit or use this class.

Protected Members (valuation, numberOfShares, pricePerShare)

- Why we use them: These members hold the total estimated value of the company, the number of shares it holds, and the computed value of each share individually. They are made protected so that classes that extend Shares (such as ShareHolders) can access them directly, making related calculations easier, yet keeping them hidden from unrelated outside code.
- How they work: user input determines valuation and numberOfShares, and pricePerShare is computed by dividing valuation by numberOfShares. These variables together establish the composition and worth of the company's equity.

setData() Method

- Why we use it: This method provides a clear and controlled way to input the core share information for a company. It ensures that the pricePerShare is correctly calculated as soon as the valuation and total shares are known.
- How it works: It asks the user for the company's total valuation and the numberOfShares. It has validation to avoid zero shares, then calculates the pricePerShare automatically and displays it based on the inputs given.

Getter Methods (getValuation(), etc.)

- Why we use them: Getter methods such as getValuation(), getNumberOfShares(), and getPricePerShare() enable other components of the program to access the share-related information securely without directly accessing the protected members. This maintains encapsulation.
- How they work: Any getter function merely returns the current value of its corresponding protected member variable. They give read-only access, and therefore the internal share data cannot become inconsistent and only gets changed through specified methods such as setData().

Section: Transactions Class Definition

The Transactions class is one of the most important parts that works assiduously to record and take care of all the business financial inflows (income) and outflows (expenses). It serves as the system's financial book.

Transactions Class (Manages income/expense records)

- Why we use it: This class has all operations concerning daily tracking of the company's financial activities. It allows you to easily add, view, and summarize income and expenses of number of months. By having a class for transactions, we can neatly separate concerns in managing these financial assets.
- How it works: It stores individual income and expense records of the number of months user want to enter, which have a description and an amount. It has specific methods for working with these records, including adding new records and summing totals.

Private Nested Struct record

- Why we use it: The record struct is a small, specialized data type, tailored to contain information for a single financial entry (either income or expense). Declaring it private and nested inside Transactions ensures it's only accessible and usable by the Transactions class itself.
- How it works: It packages a string description (e.g., "Sales", "Rent") with an int amount into a single unit. This format makes it easier to store transaction information, enabling every income or expense item to be easily defined.

Private Income and Expense Vectors

- Why we use them: They are std::vector containers that dynamically hold all the individual record entries for income and expenses, respectively. We make them private to ensure that the list of transactions can be accessed and modified only through the controlled methods of the Transactions class, ensuring data integrity.
- How they work: As financial records of new income or expenses are entered, they get added to the respective vector. The vectors expand or contract based on requirement, effectively storing all the financial transactions that have been recorded.

setMonth() and getMonth Methods:

- Why do we use them: These methods set the number of months for which the user want to enter the income and expense records.
- How they work: The setMonth() ask user to enter the number of months for which he has to enter the records and store them to totalMonthsToRecord veriable. getMonth() is used to return the total number of months.

addIncome() Method

Why we use it: The purpose of this method is to offer users an interface to enter and record fresh sources of income of the business. The method helps them navigate to enter a description as well as the respective amount.

How it works: It employs a loop to continuously ask the user to input an income source description and amount, populating record objects. Each filled record is added to the Income vector, enabling multiple income entries to be entered simultaneously.

addExpense() Method

- Why we use it: Like addIncome(), this method lets the user record new expenses made by the business. It records what the money was spent on and the amount.
- How it works: It keeps asking for an expense description and amount over and over, generating record objects. The record objects are then inserted into the Expense vector, which enables multiple expense entries input.

displayAllIncome() and displayIncomeForMonth() Methods

- Why we used them: displayAllIncome() is used to display the income of all the months of which the user entered the records and displayIncomeForMonth() is used to display the income for each month.

- How it works: In these two methods, the loops are used to go through all the income records for all months and for each month and add the records.

displayAllExpense() and displayExpenseForMonth() Methods

- Why we used them: displayAllExpense() is used to display the expense of all the months of which the user entered the records and displayExpenseForMonth() is used to display the expenses for each month.
- How it works: In these two methods, the loops are used to go through all the expenses records for all months and for each month and add the records.

generateMonthlyReport() and saveMonthlyReportToFile() Methods

- Why we used them: These methods are used to generate the report of monthly records including income and expenses per month as asked by the user and save it to a file.
- How it works: They collect the records from the previous methods in which the records are saved and generate a report for each month, then using the fstream it stores it to a file which is generated as per user instructions.

Section: ShareHolders Class Definition

ShareHolders class is exclusively designed to handle all data related to the people or organizations that hold shares in the company. It extends the fundamental share framework established in the Shares class.

ShareHolders Class (Derived from Shares, handles shareholder information)

- Why we use it: This class broadens the idea of Shares to link share ownership with actual people. Inheriting from Shares, it has direct access to company valuation and share price, which are required for calculating individual shareholder percentage and values.
- How it works: A ShareHolders object holds a list of single shareholder records. It applies the data in its underlying Shares class to calculate proportional ownership and value in each shareholder's holdings.

Protected Nested Struct ShareHolderInfo

- Why we use it: This inner struct is a template for holding all the important information regarding one shareholder in an orderly, easy-to-understand manner. It holds their name, ID, number of

shares, and computed share-of-ownership percentage and value. Declaring it as protected enables child classes (such as ProfitCalculator) to use these details if necessary, without corrupting it from outside code.

- How it works: With every new shareholder added, an object of ShareHolderInfo is instantiated and filled with their respective information. This keeps all the respective information of each shareholder together.

Public holders Vector

- Why we use it: The holders vector is an active list storing all the separate ShareHolderInfo structs. It lets us deal with any number of shareholders in an efficient way. Although public, direct access used by other classes at present (such as ProfitCalculator and Report) for processing shareholder information simplifies integration.
- How it works: Upon entering a shareholder's information, a ShareHolderInfo object is created and subsequently added (or "pushed back") into this holders vector. This vector thus contains the complete list of all shareholders.

addShareHolders() Method

- Why we use it: This method gives us the organized process of entering and adding new shareholders. It makes sure that important information, such as their number of shares and percentage of ownership, calculated appropriately, is entered correctly.
- Functionality: It first collects information for the main owner, then enters a loop to include other shareholders if needed. For every shareholder, it asks for name, national ID, and number of shares, then computes his percentage ownership and totalShareValue depending on the company's total shares and price, and appends it to the holders vector.

totalShares() Method

- Why we use it: In this method, the total of all the shares held by all registered shareholders is computed. This amount is usually required for several calculations, particularly to ensure that distributed shares do not exceed company shares or for distributing profit.
- How it works: It loops through each ShareHolderInfo entry in the holders vector. For each shareholder, it adds his/her numberOfShares to a running total, then returns this final sum.

getAllHolders() Method

- Why we use it: This technique offers a mechanism for other segments of the program, like the Report or ProfitCalculator classes, to access a copy of the whole list of shareholder details. This enables them to operate on or display shareholder data without having direct access to the holders vector.
- How it works: It returns a std::vector<ShareHolderInfo> which is a copy of the internal holders vector. This allows external functions to work with the shareholder data while preserving the original data within the class.

displayShareHolders() Method

- Why we use it: This method generates a formatted, easy-to-read table showing all the details of every registered shareholder. It's essential for providing a clear overview of company ownership.
- How it works: It initially prints overall company share details (valuation, total shares, price per share) inherited from the Shares base class. Subsequently, it traverses the holders vector, printing each shareholder's name, ID, owned shares, ownership percentage, and total share value neatly in tabular form.

Section: ProfitCalculator Class Definition

The ProfitCalculator class is the key analytical part of the system that calculates the overall profitability of the company and the distribution of that profit to shareholders.

ProfitCalculator Class (Derives from ShareHolders, computes profit)

- Why we are using it: This class consolidates all the business profit calculation logic, which involves tax allowances and payment to owners. By inheriting from ShareHolders, it automatically has visibility of the list of shareholders and total shares, which are required for profit per share calculation.
- Working: It takes the entire income and expenses to calculate profit before and after tax, and then employs the shareholder data to distribute the profit proportionally. This hierarchical structure sensibly bundles allied financial analysis functions.

Private Members (profit values, tax)

- Why we use them: Private variables such as totalProfitBeforeTax, totalProfitAfterTax, taxRate, taxAmount, profitPerShareAfterTax, profitCalculated are used to hold all the intermediate and

final results of calculations done for profits. They are made private so that the calculated financial figures are consistent and cannot be modified directly from outside the class.

- How they work: taxRate is initialized upon object creation. The other members are filled while the calculateProfit and calculateProfitPerShareholder methods run, retaining the different steps of profit calculation.

Constructor (ProfitCalculator(double tax = 0.15))

- Why we use it: The constructor is employed to create a ProfitCalculator object, particularly initializing the default taxRate if not supplied. This renders the class adaptable to varying tax situations.
- How it works: Upon the creation of a ProfitCalculator object, this constructor is invoked. It sets the given (or default) tax rate to the taxRate private member and initializes other profit-related fields to zero, readying the object to perform calculations.

calculateProfit(Transactions &t) Method

- Why we use it: This method is the main workhorse for calculating the company's net profit. It is dependent on the Transactions class to obtain raw income and expense information.
- How: It accepts a Transactions object by reference in order to call its totalIncome() and totalExpense(). Then it computes totalProfitBeforeTax, multiplies it by taxRate to find taxAmount, and lastly, computes totalProfitAfterTax, as well as setting a flag that profit has been computed.

calculateProfitPerShareholder(ShareHolders &sh) Method

- Why we use it: This method is crucial for determining how the company's overall profit translates into individual earnings for each shareholder. It leverages the shareholder data inherited from the ShareHolders base class.
- How it works: It first checks if there are any shareholders. If so, it calculates the profitPerShareAfterTax by dividing the totalProfitAfterTax by the total number of shares (sh.totalShares()). Then, it iterates through each shareholder and calculates their individual profit based on the number of shares they own, storing these values in a vector<double> which it returns.

displayProfitPerShareholder(ShareHolders &sh) Method

- Why we use it: The method gives a precise and comprehensive explanation of how the calculated profit is shared among all shareholders. Transparency and reporting to investors require it.
- Working: It fetches the list of shareholders from the ShareHolders object that it is passed. For every shareholder, it applies the previously computed profitPerShareAfterTax and the number of shares the shareholder owns (numberOfShares) to calculate and display individual profit received by the shareholder, his/her name, and shares owned.

Getter Methods (getProfitBeforeTax(), etc.)

- Why we use them: Getter methods such as getProfitBeforeTax(), getProfitAfterTax(), getTaxRate(), and getTaxAmount() grant read-only, controlled access to the computed profit and tax values. This enables other components of the system (such as the Report class) to fetch these valuable monetary metrics without having direct access to the private members.
- How they work: Each getter merely returns the latest value of its equivalent private member variable, without compromising the integrity of profit calculations.

Section: Report Class Definition

Report class is the ultimate aggregation point in our system, and it is tasked with summarizing and displaying a complete picture of the business's current status, financial performance, and shareholder data.

Report Class (Virtual Inheritance from ProfitCalculator, generates reports)

- Why we use it: This class exists solely to consolidate data from all corners of the system (business information, transactions, shareholders, and profit calculations) into a unified, organic report. Its inheritance of ProfitCalculator (and by extension ShareHolders and Shares) makes readily available all of the financial and ownership information needed for assembly.
- How it works: The Report class itself does not retain much data; it is actually responsible for bringing and displaying information by calling methods from other classes such as Business, Transactions, ShareHolders, and ProfitCalculator. Application of virtual public inheritance in the case of ProfitCalculator is a C++ technique beneficial in more intricate inheritance structures so that in the event Report was to derive from two or more classes which in turn derived from some shared base (e.g., both Business and ProfitCalculator were to inherit from some common Entity class in some perverse inheritance structure), it would have just one shared sub-object of that shared base and prevent possible ambiguities or duplicated data. In this particular case, its

main advantage remains the logical connection and access to the functionality of ProfitCalculator.

Section: main() Function

The main() function is the entry point of the program and coordinates the general flow of the Business Management System. This is where everything comes together and provides the user interface.

main() Function (main menu)

- Why we use it: The entry point of any C++ program is the main() function; execution always starts here. It serves as the command center of this system in this system, displaying the user interface and controlling the flow of the program based on user input.
- How it works: It initializes the required class objects, displays the main menus (login/signup and then the dashboard), handles user decisions, and invokes the corresponding methods from the different classes for functions such as registering businesses, handling finances, or creating reports.

Object Instantiation

- Why we use it: We have to create instances (objects) of our classes before we can use the functionalities defined in them. This allocates memory to these objects and initializes them for use.
- How it works: Lines such as Business b;, Transactions t;, Shares s;, ShareHolders sh;, ProfitCalculator pc;, and Report r; instantiate individual objects of every class. The objects then store their own data and enable us to invoke their respective methods.

Login/Signup Loop

- Why we use it: This first loop is crucial in securing the system so that access to business management features is only available to authenticated users. It takes new users through registration and old users through logging in.
- What it does: It keeps showing a menu with "Sign Up," "Login," or "Exit." Depending on the user's input, it invokes b.signup() or b.login(). The loop runs indefinitely until a login success (isLoggedIn = true) or the user decides to exit.

Main Dashboard Loop (User options and function calls based on option)

- Why we use it: This is the central interactive section of the system once a user has logged in. It displays all the management options there are and enables the user to navigate and access the different functionalities of the system.
- It works: This loop continually shows a complete dashboard menu with many choices like "Register Business Info," "Add Income," "Print Report". It scans the user's selection and then makes a call to the appropriate method on the proper object using a switch statement (e.g., b.setData(), t.addIncome(), r.generateReport()), allowing the user to interact with the whole system.

Screenshots of the Output:

Login:

```
--- Welcome to Business Management System ---
1. Sign Up
2. Login
3. Exit

Enter your choice: 2
Email:
abc@gmail.com
Password:
123
Login successful
```

Dashboard:

```
--- Dashboard ---
1. Register Business Info
2. Enter the shares details
3. Add Income
4. Add Expense
5. View Total Income & Expenses
6. Display Shareholders Details
7. Calculate & Display Profit
8. Print Report for one month
9. Save Report of one month to a file
10. Print Total Report
11. Save Total Report to a data file
0. Exit
```

Company Details:

```
Enter your choice: 1

Enter the Name of the Company:
XYZ
Enter the Name of owner of the Company:
Jhon
Enter the capital of the company:
10000000
Enter the capital threshold of the company:
300000
```

Share Details:

```
Enter your choice: 2

Enter the valuation of company: 100000000
Enter the total number of shares: 1000
Price of one share: 100000

--- Enter OWNER's Shares Detail ---
Enter OWNER's name: Jhon
Enter OWNER's national ID: 001
Enter number of shares owned by OWNER: 700

--- Add Other Shareholders ---

Enter shareholder's name: Doe
Enter shareholder's national ID: 002
Enter number of shares owned: 300
All company shares have been allocated.
```

Adding Income per month:

```
Enter your choice: 3

Enter the number of months for which you want to enter the records:
2
Enter records for month: 1

Enter the source of income:
Sales
Enter the amount of income:
500000
Do you want to add income form other sources (Y/N)?:
n
Enter records for month: 2

Enter the source of income:
Sales
Enter the amount of income:
600000
Do you want to add income form other sources (Y/N)?:
n
```

Adding Income per month:

```
Enter your choice: 4

Enter records for month: 1

Enter the source of expense:
Salaries
Enter the amount spent:
200000
Do you want to add any other expense (Y/N)?:
n
Enter records for month: 2

Enter the source of expense:
Salaries
Enter the amount spent:
250000
Do you want to add any other expense (Y/N)?:
n
```

```
Total Income: 1.1e+06 PKR
Total Expense: 450000 PKR
```

Shareholders Details:

```
--- Company Share Information ---
Total valuation of the company: 100000000
Total number of shares of the company: 1000
Price of each share of the company: 100000

--- Shareholders Details ---
Name                National ID         Shares Owned      % Ownership       Share Value (PKR)
-------------------------------------------------------------------------------------------------
Jhon                001                 700               70.00             70000000
Doe                 002                 300               30.00             30000000
```

Profit distribution :

```
--- Profit Report for all months---
Total Profit (Before Tax): 650000.00 PKR
Tax Deducted (@ 15.00%): 97500.00 PKR
Total Profit (After Tax): 552500.00 PKR

--- Monthly Profit After Tax ---

Month 1:
  Total Income: 500000 PKR
  Total Expense: 200000 PKR
  Profit Before Tax: 300000 PKR
  Tax (15%): 45000.00 PKR
  Profit After Tax: 255000.00 PKR

Month 2:
  Total Income: 600000 PKR
  Total Expense: 250000 PKR
  Profit Before Tax: 350000 PKR
  Tax (15%): 52500.00 PKR
  Profit After Tax: 297500.00 PKR
```

Profit distribution among shareholders:

```
Profit Per Share (After Tax): 552.50 PKR
Shareholder: Jhon | Profit: 386750.00 PKR
Shareholder: Doe | Profit: 165750.00 PKR

--- Profit Distribution to Shareholders ---
Shareholder: Jhon
Shares Owned: 700
Profit Received: 386750.00 PKR
-----------------------------
Shareholder: Doe
Shares Owned: 300
Profit Received: 165750.00 PKR
-----------------------------

--- Capital Comparison ---
Initial Capital: 10000000 PKR
Current Capital Initial + Profit After Tax: 10650000.00 PKR
Capital has increased by: 650000.00 PKR
```

Report for one month:

```
Enter the month number for the report: 1

--- Monthly Financial Report for XYZ (Month 1) ---
------------------------------------------------------

Income for Month 1:
- Sales: 500000 PKR
Total income for month 1: 500000 PKR


Expense for Month 1:
- Salaries: 200000 PKR
Total Expense for month 1: 200000 PKR

Summary for Month 1:
  Total Income: 500000 PKR
  Total Expense: 200000 PKR
  Net Profit/Loss: 300000 PKR
------------------------------------------------------
```

Saved Report of one month to a file:

month.txt

```
--- Monthly Financial Report for XYZ (Month 1) ---
------------------------------------------------------

Income for Month 1:
- Sales: 500000 PKR
Total income for month 1: 500000 PKR

Expense for Month 1:
- Salaries: 200000 PKR
Total Expense for month 1: 200000 PKR

Summary for Month 1:
  Total Income: 500000 PKR
  Total Expense: 200000 PKR
  Net Profit/Loss: 300000 PKR
------------------------------------------------------
```

Generate a complete report:

```
====== Monthly Business Report ======
<<
Company Details:
Name of the Company: XYZ
Name of owner of the Company: Jhon
Capital of the company: 10000000

ALERT: Your capital has reached the threshold of 0 PKR.

Financial Summary:
Total Income: 1100000.00 PKR
Total Expense: 450000.00 PKR
Total Profit (Before Tax): 650000.00 PKR
Tax Amount (15.00%): 97500.00 PKR
Total Profit (After Tax): 650000.00 PKR

Shareholder Profit Distribution:

--- Company Share Information ---
Total valuation of the company: 100000000
Total number of shares of the company: 1000
Price of each share of the company: 100000
```

Saved all record to a file

data.txt

```
====== Monthly Business Report ======
Company Details:
Name: XYZ
Owner: Jhon
Capital: 10000000

ALERT: Your capital has reached the threshold of 0 PKR.

Financial Summary:
Total Income: 1.1e+06 PKR
Total Expense: 450000 PKR
Total Profit (Before Tax): 650000 PKR
Tax Amount (15%): 97500 PKR
Total Profit (After Tax): 650000 PKR
Shareholder Profit Distribution:
Name: Jhon, Shares: 700, Profit: 386750 PKR
Name: Doe, Shares: 300, Profit: 165750 PKR
====================================

```