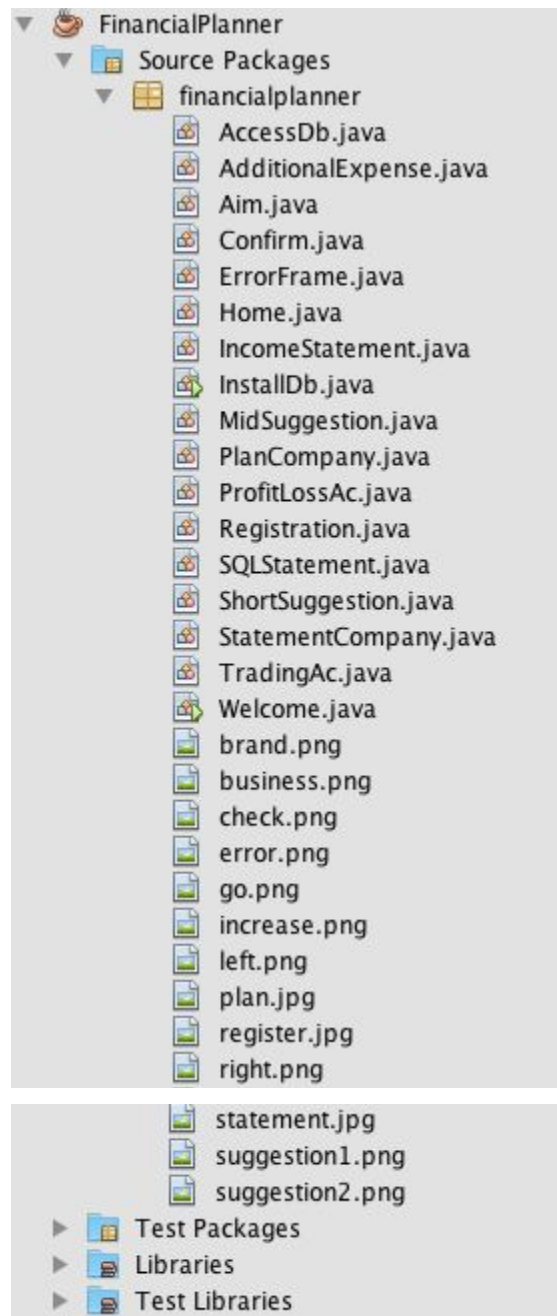


Criterion C: Development

Financial Planner is a program which helps any business to effectively manage their financial account. It provides an income statement. Also, based on the goal the business aims, it gives them suggestions on how they should manage.

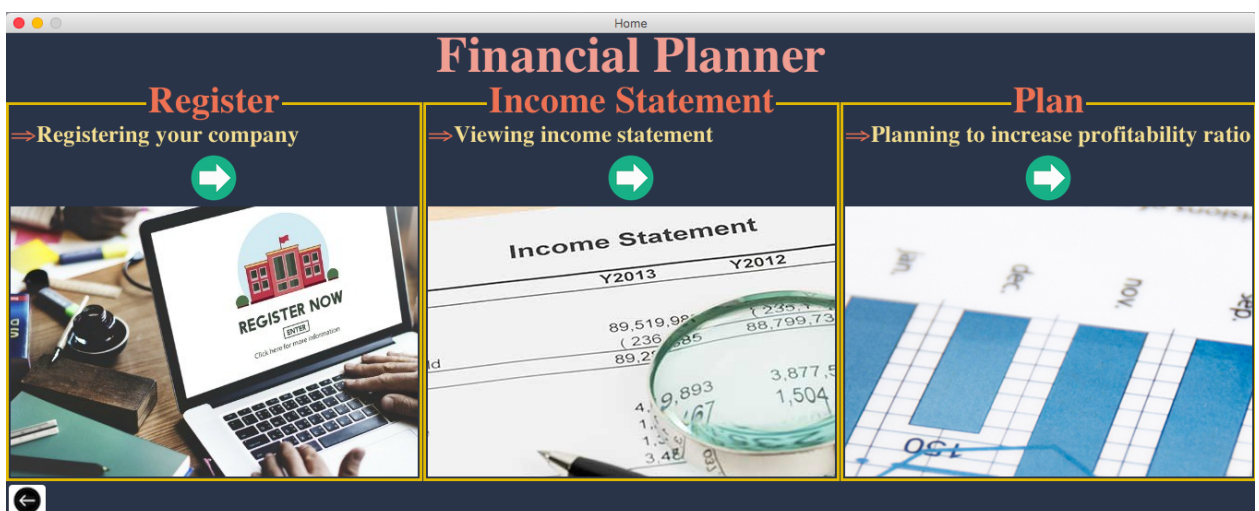
A. The structure of the product



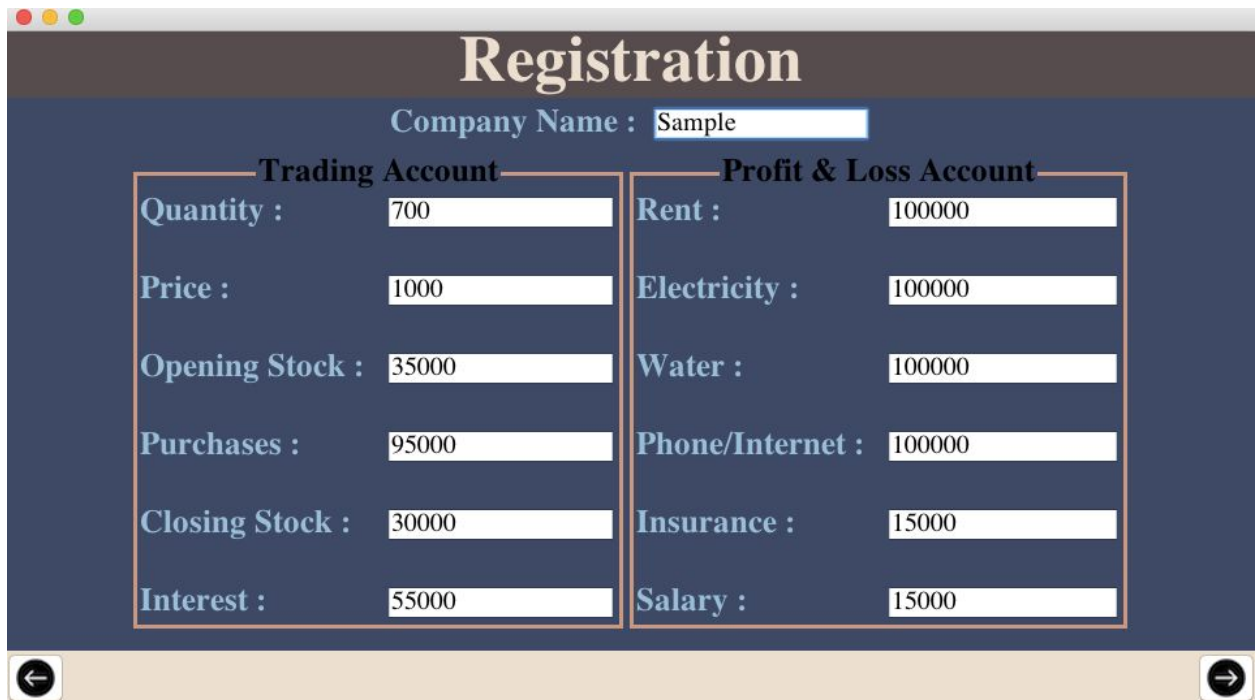
At first, in the Welcome frame, there is a brief description of the program.



Pressing the “START” button, it goes to the home frame which includes the three sections: “Register”, “Income Statement”, “Plan”. “Register” is for users who want to register new company. It asks their detailed financial information to provide a better analysis. In “Income Statement”, it illustrates the income statement so that users check their performance of the year. Lastly, in “Plan”, users view a plan to increase their profitability ratio. Moreover, direction buttons are in every frame helping users to easily move around.



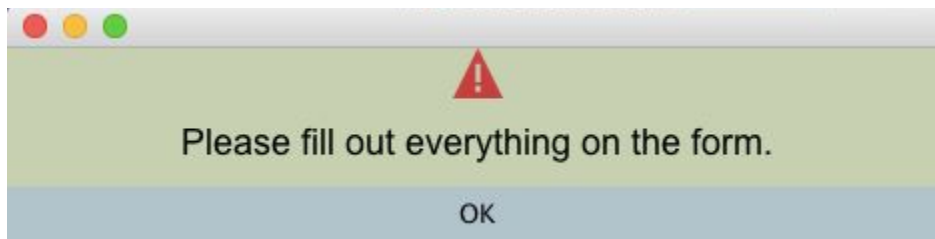
In “Register”, users need to input financial information. The frame is divided into two sections: Trading Account and Profit, and Loss Account.



The image shows a 'Registration' window with a dark blue background. At the top, the title 'Registration' is in a large, bold, serif font. Below it, 'Company Name : Sample' is displayed in a white text box. The main area is divided into two columns by a vertical line. The left column is titled 'Trading Account' and contains six rows of labels and input fields: 'Quantity : 700', 'Price : 1000', 'Opening Stock : 35000', 'Purchases : 95000', 'Closing Stock : 30000', and 'Interest : 55000'. The right column is titled 'Profit & Loss Account' and contains five rows: 'Rent : 100000', 'Electricity : 100000', 'Water : 100000', 'Phone/Internet : 100000', and 'Insurance : 15000'. At the bottom of the window, there is a light beige bar with a left-pointing arrow on the left and a right-pointing arrow on the right.

Trading Account		Profit & Loss Account	
Quantity :	700	Rent :	100000
Price :	1000	Electricity :	100000
Opening Stock :	35000	Water :	100000
Purchases :	95000	Phone/Internet :	100000
Closing Stock :	30000	Insurance :	15000
Interest :	55000	Salary :	15000

If they don't input any one of them or numerical value, error message pops up.



Users also write more expenses additionally. They write the name of the expense left and value of the expense on the right. If they click the plus button, the more space comes up. Also reaching the limit for the additional expense, they see warning quote.

Additional Expense

put name and value if you need

 :

+

←
→

Additional Expense

put name and value if you need

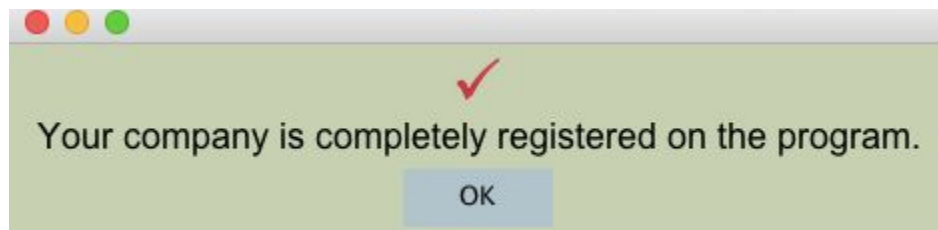
	:	
	:	
	:	
	:	
	:	
	:	

You reached the limit for the additional expense.

+

←
→

After that, a pop-up frame shows confirmation of their registration to the program.



In “Income Statement”, users input the name of the company to view income statement. The error pops when the wrong name is given.

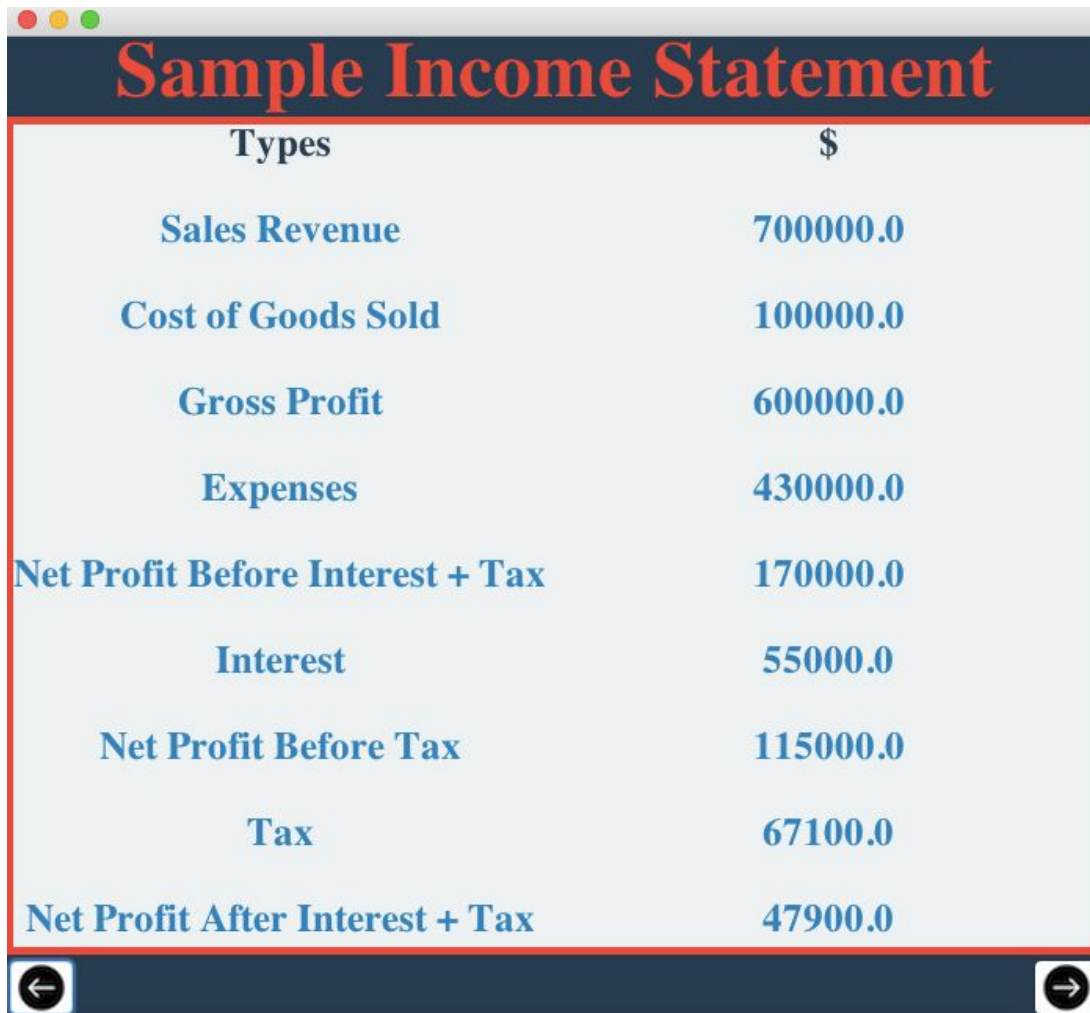
Enter the name of your company

 OK

←



On the next page, it shows users the complete income statement.

A screenshot of a "Sample Income Statement" window. The title bar has three colored buttons (red, yellow, green). The title "Sample Income Statement" is in large red font. The table has two columns: "Types" and "\$". The data is as follows:

Types	\$
Sales Revenue	700000.0
Cost of Goods Sold	100000.0
Gross Profit	600000.0
Expenses	430000.0
Net Profit Before Interest + Tax	170000.0
Interest	55000.0
Net Profit Before Tax	115000.0
Tax	67100.0
Net Profit After Interest + Tax	47900.0

The bottom of the window has a dark blue bar with a left arrow button on the left and a right arrow button on the right.

Furthermore, in "Plan", users input the name for the program to find the correct information and calculate both Net Profit Margin and Gross Profit Margin.



For the short term, it asks the user an ideal GPM from 1% to 5%. For the mid-term, it asks the user an ideal NPM from 6% to 10%. In the next page, it shows the user how much monetary value to improve and it gives them suggestions to achieve their goal. For the short term, it provides you goal GPM, ideal sales revenue, and the ideal cost of goods sold. For the long term, it gives a goal NPM and ideal expenses.

☒ Short Term
 ☐ Mid Term

Current Ratio	
Current Gross Profit Margin :	85.71%
Current Net Profit Margin :	24.29%

What is your ideal gross profit margin?

Increase it by

- ✓ 1%
- 2%
- 3%
- 4%
- 5%

Goal Gross Profit Margin : 86.71

You should increase Sales Revenue by \$ 52688.17

OR

You should decrease Cost of Goods Sold by \$ 7000.0

Methods

- using improved promotional strategies to persuade more customers to buy its products
- introducing new products with higher gross profit margins
- cutting prices of products sold in highly competitive markets in order to attract more customers
- sourcing suppliers with cheaper raw material prices, resulting in lower COGS



● Short Term ● Mid Term

Current Ratio

Current Gross Profit Margin : 85.71%

Current Net Profit Margin : 24.29%



What is your ideal net profit margin?

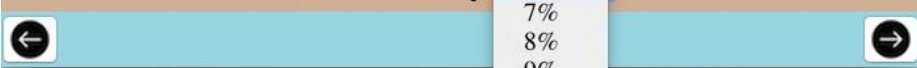
Increase it by **✓ 6%**

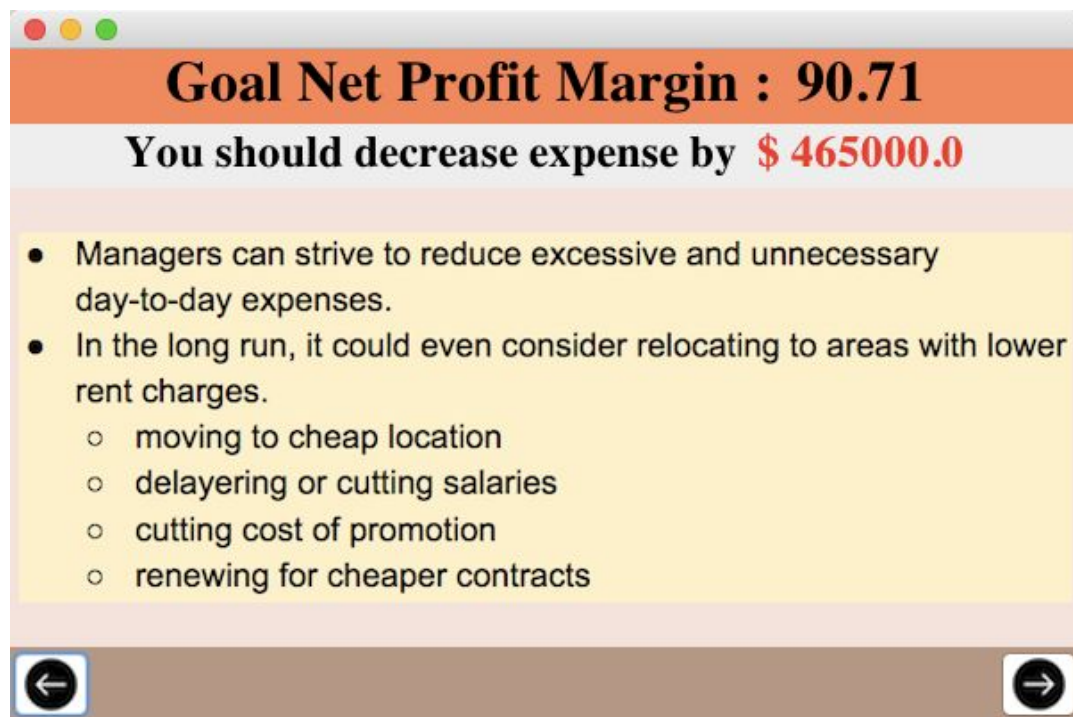
7%

8%

9%

10%





B. The techniques used in the development of the product

a. Graphical User Interface (GUI)

After discussions with advisor and client, I decided to use the graphics classes, such as Swing GUI and AWT GUI, provided in JDK for constructing my application. Since GUI is one of the well-known easy-to-use tools as user-friendly, I thought it matches my program.


```

package financialplanner;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.SQLException;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.border.Border;
import javax.swing.border.LineBorder;
import javax.swing.border.TitledBorder;

```

```

shortAnswer = new JLabel("Increase it by ");
midAnswer = new JLabel("Increase it by ");
increase = new JLabel(new ImageIcon(INCREASE_PATH));

//JPanel
buttonPane = new JPanel(new BorderLayout());
radioPane = new JPanel(new FlowLayout());
currentPane = new JPanel(new GridLayout(GRID_ROW, GRID_COL, GRID_HGAP, GRID_VGAP));
inputPane = new JPanel(new BorderLayout());
answerPane = new JPanel(new FlowLayout());
centerPane = new JPanel(new BorderLayout());

//JButton
leftButton = new JButton(new ImageIcon(LEFT_PATH));
rightButton = new JButton(new ImageIcon(RIGHT_PATH));

//JRadioButton
shortTerm = new JRadioButton("Short Term", true);
midTerm = new JRadioButton("Mid Term");

//ButtonGroup
radioGroup = new ButtonGroup();

//JComboBox
shortBox = new JComboBox<>(shortArray);
midBox = new JComboBox<>(midArray);

//Font
timesRoman = new Font("TimesRoman", Font.BOLD, FONT_T);
serif1 = new Font("Serif", Font.BOLD, FONT_S1);
serif2 = new Font("Serif", Font.PLAIN, FONT_S2);

//Color
navy = new Color(COLOR_NR, COLOR_NG, COLOR_NB);
aqua = new Color(COLOR_AR, COLOR_AG, COLOR_AB);

```

```

this.add(radioPane, BorderLayout.NORTH);
this.add(centerPane, BorderLayout.CENTER);
this.add(buttonPane, BorderLayout.SOUTH);
this.setVisible(true);

//JPanel
buttonPane.add(leftButton, BorderLayout.WEST);
buttonPane.add(rightButton, BorderLayout.EAST);
buttonPane.setBackground(sky);
radioPane.add(shortTerm);
radioPane.add(midTerm);
radioPane.setBackground(navy);
currentPane.add(currentGpm);
currentPane.add(curGpmVal);
currentPane.add(currentNpm);
currentPane.add(curNpmVal);
currentPane.setBorder(firstBorder);
currentPane.setBackground(yellow);
answerPane.add(shortAnswer);
answerPane.add(shortBox);
answerPane.setBackground(beige);
inputPane.add(shortQuestion, BorderLayout.NORTH);
inputPane.add(answerPane, BorderLayout.SOUTH);
inputPane.setBackground(pink);

centerPane.add(currentPane, BorderLayout.NORTH);
centerPane.add(increase, BorderLayout.CENTER);
centerPane.add(inputPane, BorderLayout.SOUTH);
centerPane.setBackground(yellow);

//ButtonGroup
radioGroup.add(shortTerm);
radioGroup.add(midTerm);

```

The pictures include import of Java Swings and AWTs of the class Aim, initialization of the variables, and the process of adding components. I organized the components with the JPanel, ButtonGroup, Border for users to easily view.

```

//actionPerformed
public void actionPerformed(ActionEvent e)
{
    String buttonCommand = e.getActionCommand();

    if (buttonCommand.equals("+"))
    {
        if (counter < 6)
        {
            if (counter == 1)
            {
                moreSymbol1.setFont(serif2);
                moreArea1.setFont(serif2);
                moreValue1.setFont(serif2);
                morePanel1.setBackground(brown);
                morePanel1.add(moreArea1, BorderLayout.WEST);
                morePanel1.add(moreSymbol1, BorderLayout.CENTER);
                morePanel1.add(moreValue1, BorderLayout.EAST);
                growPane.add(morePanel1);
                counter++;
            } else if (counter == 2)
            {
                moreSymbol2.setFont(serif2);
                moreArea2.setFont(serif2);
                moreValue2.setFont(serif2);
                morePane2.setBackground(brown);
                morePane2.add(moreArea2, BorderLayout.WEST);
                morePane2.add(moreSymbol2, BorderLayout.CENTER);
                morePane2.add(moreValue2, BorderLayout.EAST);
                growPane.add(morePane2);
                counter++;
            } else if (counter == 3)
            {
                moreSymbol3.setFont(serif2);
                moreArea3.setFont(serif2);
                moreValue3.setFont(serif2);
            }
        }
    }
}

```

This shows the actionPerformed of the class AdditionalExpense. By counting the number of clicking the button, it performs different actions according to the counter. In this case, it gives more JTextfields to input as the user clicks the plus button.

b. Object-Oriented Programming (OOP)

I programmed the program based on the Object-Oriented Programming having features of Encapsulation, Inheritance, and Polymorphism. By mutators and accessors, I could easily transfer, save, call the values between the classes. This helps me to keep codes well-organized. OOP also reduces duplicate codes and simplifies the testing for my program. Thus, it helped me to write the big size of codes efficiently.

```

public class TradingAc
{
    // instance variable
    private String name;
    private double salesRevenue;
    private double costOfGoodsSold;
    private double quantity;
    private double price;
    private double openingStock;
    private double purchases;
    private double closingStock;
    private double grossProfit;

    // 2 constructors
    public TradingAc()
    {
        name = null;
        salesRevenue = 0;
        costOfGoodsSold = 0;
        quantity = 0;
        price = 0;
        openingStock = 0;
        purchases = 0;
        closingStock = 0;
        grossProfit = 0;
    }

    public TradingAc(String name, double salesRevenue, double costOfGoodsSold,
        double quantity, double price, double openingStock, double purchases,
        double closingStock, double grossProfit)
    {
        this.name = name;
        this.salesRevenue = salesRevenue;
        this.costOfGoodsSold = costOfGoodsSold;
        this.quantity = quantity;
        this.price = price;
    }
}

```

```

// mutators + accessors
public String getName()
{
    return name;
}

public void setName(String name)
{
    this.name = name;
}

public double getSalesRevenue()
{
    return salesRevenue;
}

public void setSalesRevenue(double salesRevenue)
{
    this.salesRevenue = salesRevenue;
}

public double getCostOfGoodsSold()
{
    return costOfGoodsSold;
}

public void setCostOfGoodsSold(double costOfGoodsSold)
{
    this.costOfGoodsSold = costOfGoodsSold;
}

public double getQuantity()
{
    return quantity;
}

public void setQuantity(double quantity)
{
    this.quantity = quantity;
}

public double getPrice()
{
    return price;
}

public void setPrice(double price)
{
    this.price = price;
}

public double getOpeningStock()
{
    return openingStock;
}

public void setOpeningStock(double openingStock)
{
    this.openingStock = openingStock;
}

public double getPurchases()
{
    return purchases;
}

public void setPurchases(double purchases)
{
    this.purchases = purchases;
}

public double getClosingStock()
{
    return closingStock;
}

public void setClosingStock(double closingStock)
{
    this.closingStock = closingStock;
}

public double getGrossProfit()
{
    return grossProfit;
}

public void setGrossProfit(double grossProfit)
{
    this.grossProfit = grossProfit;
}

```

These are instance variables, two constructors, and mutators and accessors of the TradingAc.


```
// Database: FinancialAccount
public void insertIncomeStatement(TradingAc x, ProfitLossAc y)
{
    tradingObj = x;
    profitObj = y;

    dbQuery = "INSERT INTO IncomeStatement "
    + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try
    {
        s = financialConn.prepareStatement(dbQuery);
        s.setString(1, tradingObj.getName());
        s.setDouble(2, tradingObj.getSalesRevenue());
        s.setDouble(3, tradingObj.getCostOfGoodsSold());
        s.setDouble(4, tradingObj.getGrossProfit());
        s.setDouble(5, profitObj.getExpenses());
        s.setDouble(6, profitObj.getNetProfitBefore());
        s.setDouble(7, profitObj.getInterest());
        s.setDouble(8, profitObj.getNetProfitBfTax());
        s.setDouble(9, profitObj.getTax());
        s.setDouble(10, profitObj.getNetProfitAfter());
        status = s.executeUpdate();
    }
}
```

c. Database

```
//Declaration
String IncomeStatementTbl;
String TaxRangeTbl;

AccessDb objDb = new AccessDb();

//Create Db & tables
objDb.createDb("FinancialAccount");
IncomeStatementTbl = "CREATE TABLE IncomeStatement ( "
    + "CompanyName varchar(30), "
    + "SalesRevenue double, "
    + "CostOfGoodSold double, "
    + "GrossProfit double, "
    + "Expenses double,"
    + "NetProfitBefore double,"
    + "Interest double,"
    + "NetProfitBeforeTax double,"
    + "Tax double,"
    + "NetProfitAfterInterestAndTax double"
    + " )";
System.out.println(IncomeStatementTbl);
objDb.createTable(IncomeStatementTbl, "FinancialAccount");

objDb.createDb("Tax");
TaxRangeTbl = "CREATE TABLE TaxRange ( "
    + "Rate int, "
    + "Supplementary int, "
    + "Minimum int, "
    + "Maximum int"
    + " )";
System.out.println(TaxRangeTbl);
objDb.createTable(TaxRangeTbl, "Tax");
```

I used the Structured Query Language(SQL) to access and manipulate the Java databases. Database is the most critical technique as the main purpose of my program is giving financial plans using business data. I created two databases: FinancialAccount and Tax.

Financial Account has a table called IncomeStatement, as Tax has a table called TaxRange.

IncomeStatement holds the data of monetary values such as revenue and cost. TaxRange holds the data of the corporate tax rate based on the government's open data.

IncomeStatement is created by insertion of users' inputs while TaxRange is created only by a programmer.



The class SQLStatement has all methods of accessing both databases using SQL. To access the database, other classes use the methods of this class. In the picture below, the class AdditionalExpense uses the method getSupplementary and getRate to calculate the tax.

```
//find tax and set both tax and net profit after
profitObj.getNetProfitBfTax();
try
{
    SQLStatement sqlObj = new SQLStatement();
    profitObj.setTax(profitObj.calculateTax(sqlObj.getSupplementary(profitObj), sqlObj.getRate(profitObj)));
} catch (SQLException ex)
{
    String warning = "Tax is not found.";
    ErrorFrame errObj = new ErrorFrame(warning);
}
profitObj.setNetProfitAfter(profitObj.calculateNetProfitAfter());
this.setEnabled(false);
Confirm confirmObj = new Confirm(tradingObj, profitObj, this);
```

The picture below is the method `getValues` in the class `SQLStatement`. After getting the data into an array of double from database `FinancialAccount`, it returns the array.

```
public double[] getValues(String pName) throws SQLException
{
    double[] doubleArray = new double[9];
    dbQuery = "SELECT SalesRevenue, CostOfGoodSold, GrossProfit, Expenses, "
        + "NetProfitBefore, Interest, NetProfitBeforeTax, Tax, "
        + "NetProfitAfterInterestAndTax FROM IncomeStatement " + "WHERE CompanyName = ?";

    s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();

    while (rs.next())
    {
        doubleArray[0] = rs.getDouble("SalesRevenue");
        doubleArray[1] = rs.getDouble("CostOfGoodSold");
        doubleArray[2] = rs.getDouble("GrossProfit");
        doubleArray[3] = rs.getDouble("Expenses");
        doubleArray[4] = rs.getDouble("NetProfitBefore");
        doubleArray[5] = rs.getDouble("Interest");
        doubleArray[6] = rs.getDouble("NetProfitBeforeTax");
        doubleArray[7] = rs.getDouble("Tax");
        doubleArray[8] = rs.getDouble("NetProfitAfterInterestAndTax");
    }

    return doubleArray;
}
```

```
//setValues
public void setValues() throws SQLException
{
    double[] valueArr;
    SQLStatement sqlObj = new SQLStatement();
    valueArr = sqlObj.getValues(name);
    srValue = new JLabel(Double.toString(valueArr[0]));
    cogsValue = new JLabel(Double.toString(valueArr[1]));
    gpValue = new JLabel(Double.toString(valueArr[2]));
    expValue = new JLabel(Double.toString(valueArr[3]));
    npBfValue = new JLabel(Double.toString(valueArr[4]));
    intValue = new JLabel(Double.toString(valueArr[5]));
    npBfTxValue = new JLabel(Double.toString(valueArr[6]));
    txValue = new JLabel(Double.toString(valueArr[7]));
    npAfValue = new JLabel(Double.toString(valueArr[8]));
}
```

The `getValues` is used in the method `SetValues` of the class `IncomeStatement`. `SetValues` receives the returned array into the array which is the method variable.

C. The algorithmic thinking used in the development of the product

a. Register

In the class Registration, it sets the variables to build the income statement based on the inputs of users.

```
try
{
    tradingObj.setName(nameText.getText());
    tradingObj.setQuantity(Double.parseDouble(quantityText.getText()));
    tradingObj.setPrice(Double.parseDouble(priceText.getText()));
    tradingObj.setOpeningStock(Double.parseDouble(openText.getText()));
    tradingObj.setPurchases(Double.parseDouble(purchaseText.getText()));
    tradingObj.setClosingStock(Double.parseDouble(closeText.getText()));
    tradingObj.setSalesRevenue(tradingObj.calculateSalesRevenue());
    tradingObj.setCostOfGoodsSold(tradingObj.calculateCostOfGoodsSold());
    tradingObj.setGrossProfit(tradingObj.calculateGrossProfit());

    profitObj.setRent(Double.parseDouble(rentText.getText()));
    profitObj.setElectricity(Double.parseDouble(electricityText.getText()));
    profitObj.setWater(Double.parseDouble(waterText.getText()));
    profitObj.setInternet(Double.parseDouble(internetText.getText()));
    profitObj.setInsurance(Double.parseDouble(insuranceText.getText()));
    profitObj.setSalary(Double.parseDouble(salaryText.getText()));
    profitObj.setExpenses(profitObj.calculateExpenses());
    profitObj.setNetProfitBefore(profitObj.calculateNetProfitBefore(tradingObj.getGrossProfit()));
    profitObj.setInterest(Double.parseDouble(interestText.getText()));
    profitObj.setNetProfitBfTax(profitObj.calculateNetProfitBfTax());
}
```

Next, the class AdditionalExpense gets the inputs of additional expenses and checks a number of inputs by counter and switch statement which calculates sum of numerical values of additional expenses. Then, it puts the additional expenses by using the method addExpense.

```
switch (counter)
{
    case 0:
        sum = Double.parseDouble(value.getText());
        break;
    case 1:
        sum = Double.parseDouble(value.getText()) + Double.parseDouble(moreValue1.getText());
        break;
    case 2:
        sum = Double.parseDouble(value.getText()) + Double.parseDouble(moreValue1.getText())
            + Double.parseDouble(moreValue2.getText());
        break;
    case 3:
        sum = Double.parseDouble(value.getText()) + Double.parseDouble(moreValue1.getText())
            + Double.parseDouble(moreValue2.getText()) + Double.parseDouble(moreValue3.getText());
        break;
    case 4:
        sum = Double.parseDouble(value.getText()) + Double.parseDouble(moreValue1.getText())
            + Double.parseDouble(moreValue2.getText()) + Double.parseDouble(moreValue3.getText())
            + Double.parseDouble(moreValue4.getText());
        break;
    case 5:
        sum = Double.parseDouble(value.getText()) + Double.parseDouble(moreValue1.getText())
            + Double.parseDouble(moreValue2.getText()) + Double.parseDouble(moreValue3.getText())
            + Double.parseDouble(moreValue4.getText()) + Double.parseDouble(moreValue5.getText());
        break;
    default:
        break;
}
```

```
public void addExpense(double exp)
{
    setExpenses(getExpenses() + exp);
}
```

Based on new expenses, it calculates new gross profit and net profit before tax according to the gross profit, expenses, and interest. Then, it calculates tax needed in income statement.

```
public double calculateExpenses()
{
    double expense = getRent() + getElectricity() + getWater()
        + getInternet() + getInsurance() + getSalary();
    return expense;
}
```

```
// calculator methods
public double calculateSalesRevenue()
{
    double sR = getQuantity() * getPrice();
    return sR;
}

public double calculateCostOfGoodsSold()
{
    double cogs = getOpeningStock() + getPurchases() - getClosingStock();
    return cogs;
}

public double calculateGrossProfit()
{
    double gP = getSalesRevenue() - getCostOfGoodsSold();
    return gP;
}
```

```
public double calculateTax(int sup, int rate)
{
    double t = (double) sup + (getNetProfitBfTax() * rate/100);
    return t;
}
```

In Confirm, it finally inserts all values into the database after all calculations by method insertIncomeStatement.


```

// Database: FinancialAccount
public void insertIncomeStatement(TradingAc x, ProfitLossAc y)
{
    tradingObj = x;
    profitObj = y;

    dbQuery = "INSERT INTO IncomeStatement "
        + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try
    {
        s = financialConn.prepareStatement(dbQuery);
        s.setString(1, tradingObj.getName());
        s.setDouble(2, tradingObj.getSalesRevenue());
        s.setDouble(3, tradingObj.getCostOfGoodsSold());
        s.setDouble(4, tradingObj.getGrossProfit());
        s.setDouble(5, profitObj.getExpenses());
        s.setDouble(6, profitObj.getNetProfitBefore());
        s.setDouble(7, profitObj.getInterest());
        s.setDouble(8, profitObj.getNetProfitBfTax());
        s.setDouble(9, profitObj.getTax());
        s.setDouble(10, profitObj.getNetProfitAfter());
        status = s.executeUpdate();
    } catch (Exception ee)
    {
        String warning = "Error Inserting into Table of Database";
        ErrorFrame errObj = new ErrorFrame(warning);
    }
}

```

b. Income Statement

It finds data based on the company name which is inputted by user. It also checks if a company is not registered by method checkingNames.

```

public double[] getValues(String pName) throws SQLException
{
    double[] doubleArray = new double[9];
    dbQuery = "SELECT SalesRevenue, CostOfGoodSold, GrossProfit, Expenses,"
        + "NetProfitBefore, Interest, NetProfitBeforeTax, Tax, "
        + "NetProfitAfterInterestAndTax FROM IncomeStatement " + "WHERE CompanyName = ?";

    s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();

    while (rs.next())
    {
        doubleArray[0] = rs.getDouble("SalesRevenue");
        doubleArray[1] = rs.getDouble("CostOfGoodSold");
        doubleArray[2] = rs.getDouble("GrossProfit");
        doubleArray[3] = rs.getDouble("Expenses");
        doubleArray[4] = rs.getDouble("NetProfitBefore");
        doubleArray[5] = rs.getDouble("Interest");
        doubleArray[6] = rs.getDouble("NetProfitBeforeTax");
        doubleArray[7] = rs.getDouble("Tax");
        doubleArray[8] = rs.getDouble("NetProfitAfterInterestAndTax");
    }

    return doubleArray;
}

```

```

public Boolean checkingName(String pName) throws SQLException
{
    double revenue = 0.0;

    dbQuery = "SELECT SalesRevenue FROM IncomeStatement WHERE CompanyName = ?";
    s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();
    while (rs.next())
    {
        revenue = rs.getDouble("SalesRevenue");
    }

    return revenue != 0.0;
}

```

c. Plan

It gets the GPM and NPM by matching the name. Then it finds the ideal profit margin and other values by if and else statement. For the short term, it calculates the ideal GPM, sales revenue, and expenses. For the mid-term, it calculates an ideal NPM and expenses.

```

public double getGpm(String pName) throws SQLException
{
    double gpm;

    dbQuery = "SELECT SalesRevenue, GrossProfit FROM IncomeStatement "
        + "WHERE CompanyName = ?";

    s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();
    while (rs.next())
    {
        sr = rs.getDouble("SalesRevenue");
        gp = rs.getDouble("GrossProfit");
    }
    gpm = gp / sr * 100;
    return gpm;
}

//getNpm
public double getNpm(String pName) throws SQLException
{
    double npm;

    dbQuery = "SELECT SalesRevenue, NetProfitBefore FROM IncomeStatement "
        + "WHERE CompanyName = ?";
    s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();
    while (rs.next())
    {
        sr = rs.getDouble("SalesRevenue");
        np = rs.getDouble("NetProfitBefore");
    }
    npm = np / sr * 100;
    return npm;
}

```

```
index = midBox.getSelectedIndex();
try
{
    if (index == 0)
    {
        idealNpm = sqlObj.getGpm(name) + 5.0;
        expenseValue = sqlObj.getExpenseValue(name, idealNpm);
    } else if (index == 1)
    {
        idealNpm = sqlObj.getGpm(name) + 6.0;
        expenseValue = sqlObj.getExpenseValue(name, idealNpm);
    } else if (index == 2)
    {
        idealNpm = sqlObj.getGpm(name) + 7.0;
        expenseValue = sqlObj.getExpenseValue(name, idealNpm);
    } else if (index == 3)
    {
        idealNpm = sqlObj.getGpm(name) + 8.0;
        expenseValue = sqlObj.getExpenseValue(name, idealNpm);
    } else if (index == 4)
    {
        idealNpm = sqlObj.getGpm(name) + 9.0;
        expenseValue = sqlObj.getExpenseValue(name, idealNpm);
    }
}
```

```
index = shortBox.getSelectedIndex();
try
{
    if (index == 0)
    {
        idealGpm = sqlObj.getGpm(name) + 1.0;
        revenueValue = sqlObj.getRevenueValue(name, idealGpm);
        costValue = sqlObj.getCostValue(idealGpm);
    } else if (index == 1)
    {
        idealGpm = sqlObj.getGpm(name) + 2.0;
        revenueValue = sqlObj.getRevenueValue(name, idealGpm);
        costValue = sqlObj.getCostValue(idealGpm);
    } else if (index == 2)
    {
        idealGpm = sqlObj.getGpm(name) + 3.0;
        revenueValue = sqlObj.getRevenueValue(name, idealGpm);
        costValue = sqlObj.getCostValue(idealGpm);
    } else if (index == 3)
    {
        idealGpm = sqlObj.getGpm(name) + 4.0;
        revenueValue = sqlObj.getRevenueValue(name, idealGpm);
        costValue = sqlObj.getCostValue(idealGpm);
    } else if (index == 4)
    {
        idealGpm = sqlObj.getGpm(name) + 5.0;
        revenueValue = sqlObj.getRevenueValue(name, idealGpm);
        costValue = sqlObj.getCostValue(idealGpm);
    }
}
```



```

//getCostValue
public double getCostValue(double ideal)
{
    double cv;
    cv = cogs - ((sr * (100 - ideal)) / 100);
    return cv;
}

//getExpenseValue
public double getExpenseValue(String pName, double ideal) throws SQLException
{
    double ev;
    double e = 0.0;

    dbQuery = "SELECT Expenses FROM IncomeStatement WHERE CompanyName = ?";
    s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();
    while (rs.next())
    {
        e = rs.getDouble("Expenses");
    }
    ev = e - (((100.0 * gp) - (sr * ideal)) / 100.0);
    return ev;
}

```

```

//getRevenueValue
public double getRevenueValue(String pName, double ideal) throws SQLException
{
    double rv;
    String dbQuery;
    ResultSet rs;

    dbQuery = "SELECT CostOfGoodSold FROM IncomeStatement WHERE CompanyName = ?";
    PreparedStatement s = financialConn.prepareStatement(dbQuery);
    s.setString(1, pName);
    rs = s.executeQuery();
    while (rs.next())
    {
        cogs = rs.getDouble("CostOfGoodSold");
    }
    rv = ((100 * cogs) / (100 - ideal)) - sr;
    return rv;
}

```

word count: 1027