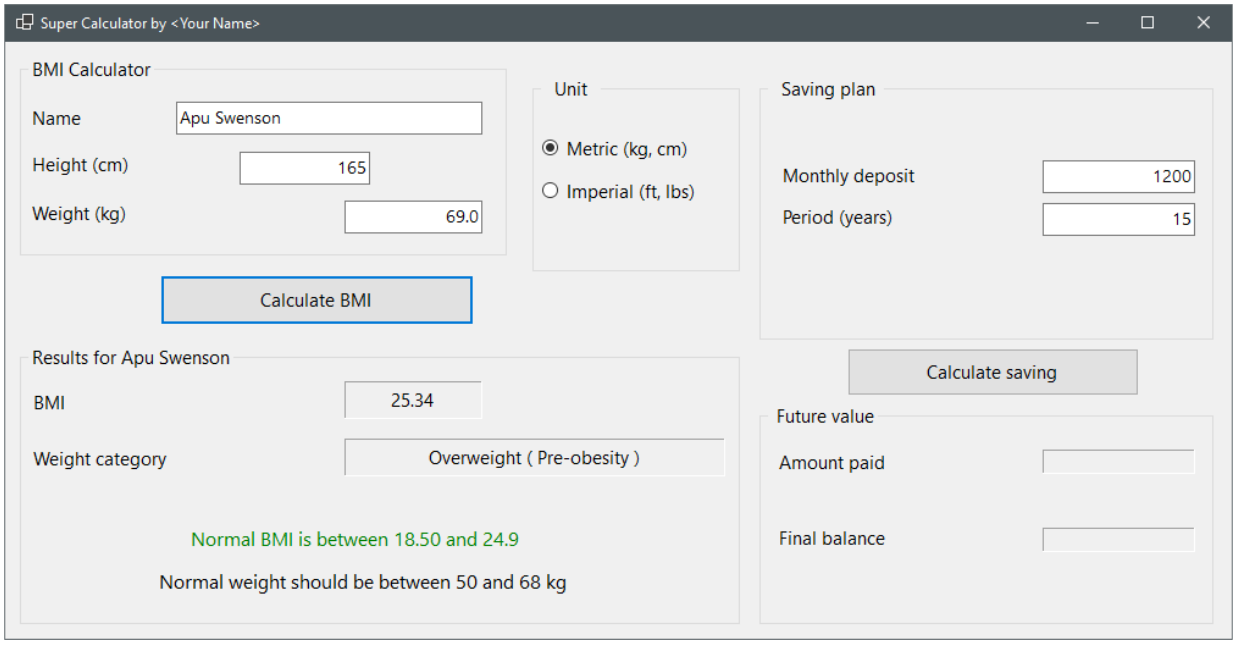
1. Objectives

Beginning with this assignment, we are going to go over to writing Windows Forms based desktop applications with graphical user interface (GUI) using some of the most common Windows Form controls. The main objectives are:

The main objectives of this assignment are:

- To begin writing Windows Forms based desktop applications with graphical user interface (GUI) using some of the most common Windows Form controls.
- To acquire input from different Windows Form controls and save them in instance variables.
- Perform input validation to avoid abnormal program termination and falsely results.
- Write and use getter and setter as well parameterized methods and methods with return value to establish communication between objects of classes.



Another important purpose of this assignment is to separate the presentation of data (GUI) from the logics (calculation and manipulation of data). In our previous assignments, we interacted with the user in the same class. From now on, we let the GUI- class (Form object), be responsible all interactions (input and output) with the user of the program, and let other classes serve the GUI by processing input data and producing the needed output data.

In case you have prior experience of Windows Forms, you may try working with Windows Presentation Foundation (WPF). You may also design the GUI and solve the problem in your

own way, as long as you meet the main requirements specified here in this document and maintain a good level of code quality.

This assignment has two parts.

a. BMI Calculator. Calculate the Body Mass Index using two types of units (metric and imperial, used in USA)

b. Calculate the future value of monthly saving over a period of time using a compound interest model.

To solve the BMI part, a great amount of help is provided, both here and in different documents in the module. Using the skills you gain from this part, you should be able to solve the second part (which is less complicated) without detailed help. Let's begin with the first part, the BMI Calculator.

2. BMI - Description

The Body Mass Index (BMI) is a measure of body fat and is commonly used within the health sector to determine whether the weight of adult men and women is healthy. BMI is calculated in relation to one's height using the following formulas:

BMI = weight in kg / (height * height in m^2) (Metric Units)

BMI = 703.0 *·weight in lb / (height * height in $inch^2$) (Imperial (U.S.) Units)

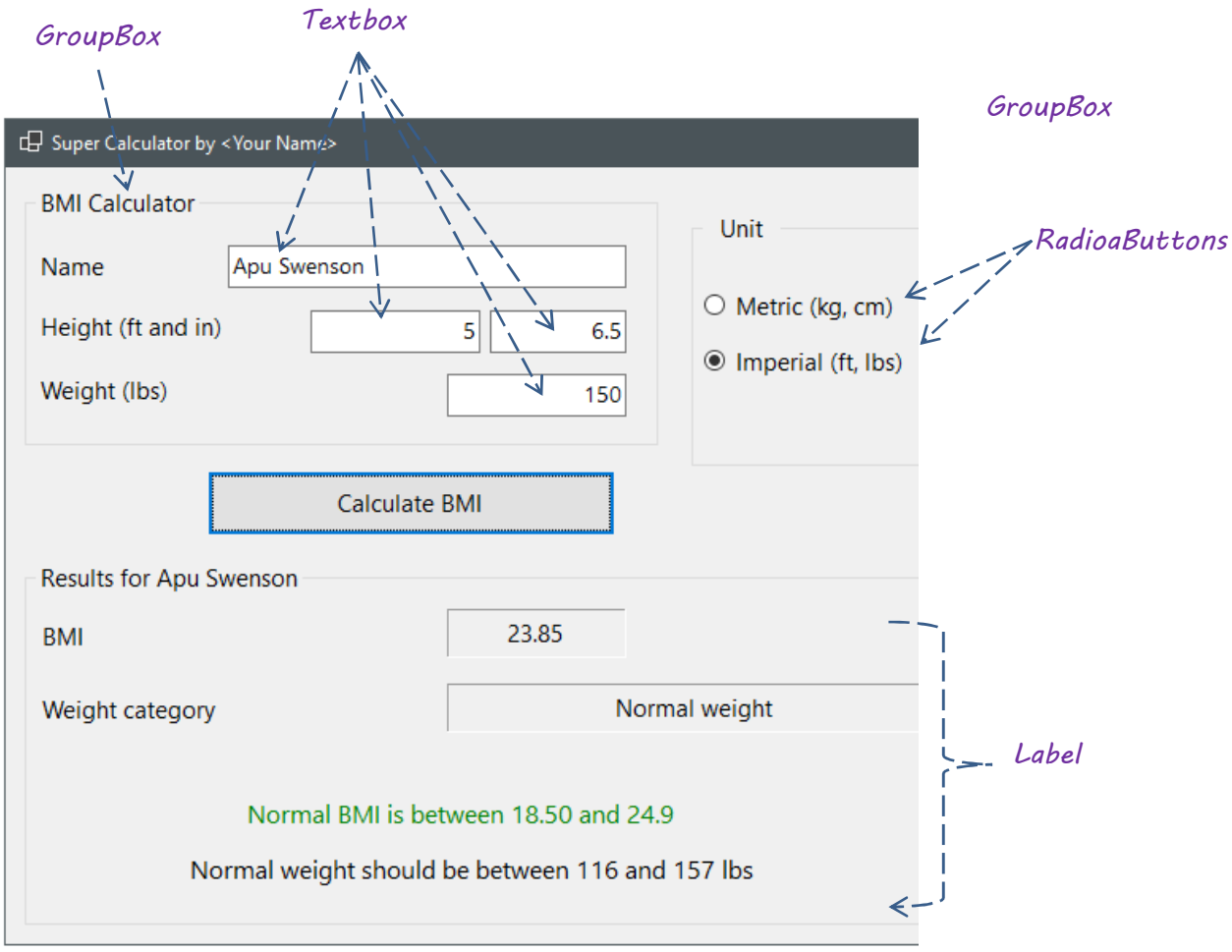
2.1 The above formulas are standard for adults not taking the age into consideration. The World Health Organization's (WHO) has recommended a body weight based on BMI values for adults, as summarized in the following table. It is used for both men and women, age 18 or older. For more information see: https://www.euro.who.int/en/health-topics/disease-prevention/nutrition/a-healthy-lifestyle/body-mass-index-bmi.

BMI	Nutritional status
Below 18.5	Underweight
18.5–24.9	Normal weight
25.0–29.9	Overweight (Pre-obesity)
30.0–34.9	Overweight (Obesity class I)
35.0–39.9	Overweight (Obesity class II)
Above 40	Overweight (Obesity class III)

2.2 or The user should have the choice of selecting the measurement system, Metric (cm, kg) Imperial (ft and in, lb) unit types. For the metric option, the user should provide the height values in meters (m) or centimeters (cm) and weight in kilograms (kg). For the Imperial option , the values should be given in feet (ft) and inches (in) and weight in pounds (lb)

3. BMI - To Do

Start Visual Studio (VS), and create a new project, .NET, Windows Forms Desktop App (or the corresponding .NET Framework template) for this assignment. Determine the input parameters you may need and which are to be provided by the user, and the output that your application should calculate and display to the user. Having these in mind, design your GUI using the proper Windows Forms controls from the **Toolbox** in Visual Studio. You can use the sample GUI-design shown below.



Label is to be used for read-only texts (output, captions, etc.). Textboxes are to be used **only** for input.

3.1 VS creates a starting form, on which you can draw your graphical components (more often called **controls**), and gives it the default name Form1. Rename Form1 to **MainForm**. This class should handle all the interactions (input/output) between the user and the

application, and it should not contain logics that are (or can be) a part of the BMI calculation.

3.2 For calculation of BMI values, write a class **BMICalculator**. This class should **not** interact with the user in any way, not even giving warnings or error messages. It neither should have access to the members of the **MainForm**. A non-GUI class should not know which other classes will be using them. Any other classes, including those not having a user interface should be able to create an object of this class and use its methods. **BMICalculator** should not be aware of that the **MainForm** will be using it. In other words, it should be usable by any other class including MainForm.

3.3 Write an enum **UnitTypes,** and define the unit types used in this application (Metric and Imperial). To create an enum, create a class and then change the keyword class to enum.

```
enum UnitTypes
{
    Metric,
    Imperial
}
```

Save this little enum in its own file UnitTypes.cs.

4. The class MainForm

4.1 The **MainForm** class should only have an instance variable, an object of the **BMICalculator** class. Use local variables for other purposes.

The only instance variable, so far. We will need another one for the next part.

```
public partial class MainForm : Form
{
    // create an instance of BMICalculator
    private BMICalculator bmiCalc = new BMICalculator ( );


    /// <summary>
    /// Initialize method
    /// </summary>
    1 reference
    public MainForm ( )
    {
        InitializeComponent ( );
        InitializeGUI ( );
    }
```

4.2 Click somewhere on the **MainForm** in VS to make the Form highlighted (the outer frame). Then go to the **Properties** and change the following:

4.2.1 **FormBorderStyle**: Select one of the fixed-types.

4.2.2 **MaximizeBox**: false

4.2.3 **StartPosition**: CenterScreen

4.2.4 **Text**: Give a title to your program (BMI Calculator) and make sure that you substitute the text <Your Name > by your real name.

4.3 Provide an **InitializeGUI** method and do your initializations there, e.g. clearing input-boxes, output controls, setting default values in controls (for example, you can select one of the option buttons as default), etc.

4.4 Use double.TryParse() to convert the contents of the **TextBoxes** used for input of height and weight into numerical values. You can write a method that you can use with all text boxes. The code for this method is given later in this document.

4.5 Instead of reading all input in one method, organize readings in separate methods.

```
private void ReadName ( )
{
    txtName.Text.Trim ( ); //Delete spaces at the beginning and end of the string
    if (!string.IsNullOrEmpty ( txtName.Text ))
        bmiCalc.Name = txtName.Text;  //no need for any type conversion
    else
        bmiCalc.Name = "Unknown";
}
```

5. The BMICalculator class

5.1 The **BMICalculator** class should only define instance variables for storing **input**, as in the figure below. Do not use any other **instance** variables, but you may (should) use as many local variable (variables inside a method) as you need-

```
class BMICalculator
{
    private string name = "No Name";
    private double height = 0;  //m, feet
    private double weight = 0;  //kg, lb
    private UnitTypes unit;
```

5.2 Weight and height should be floating-point values (double) greater than zero. As you may noticed, the BMI formula in the metric system uses meters. If you read the height in cm, you must divide it by 100 before you perform the calculation.

5.3 The application should allow the height to be given in a combination of feet and inches when the Imperial Unit is selected, although the formula uses inches in the imperial

system. On your GUI, you read feet and inches. Multiply the feet value by 12 and add the result to the inches so the height is all in inches.

5.4 Assign a default value for the name (“No name” or "Unknown") if the user does not provide one.

5.5 **Write setter and getter methods connected to the instance variables**, so MainForm can use to save input (given on the GUI) and get values stored in the instance variables, when requested. All output should be provided to the caller methods in the **MainForm** via the return statement of the methods.

5.6 The program should calculate and display the weights that correspond to the normal BMI limit values, i.e. 18.5 and 24.9 (see the Help section for hints).

6. Connection between MainForm and BMICalcultor

6.1 MainForm is responsible for handling input and output. BMICalculator is responsible for calculating values using the input receiving from the MainForm.

- MainForm declares and creates an instance of the BMCalculator (bmiCalc).
- MainForm validates the input that are given by the user on the GUI (e.g. contents of the textboxes). It then calls the setter methods of the bmiCalc to set (save) values in the object.

```
double inchValue = 0;
ok = double.TryParse(txtInch.Text, out inchValue);
if (ok)
bmiCalc.SetHeight(outValue * 12.00 + inchValue);//ft->in
```

- MainForm calls the methods of the BMICalculator to calculate and provide an output through their return statement.
- MainForm updates the GUI by displaying the output using the dedicated controls (e.g. labels).

Tre steps when the Calculate BMI button is clicked.

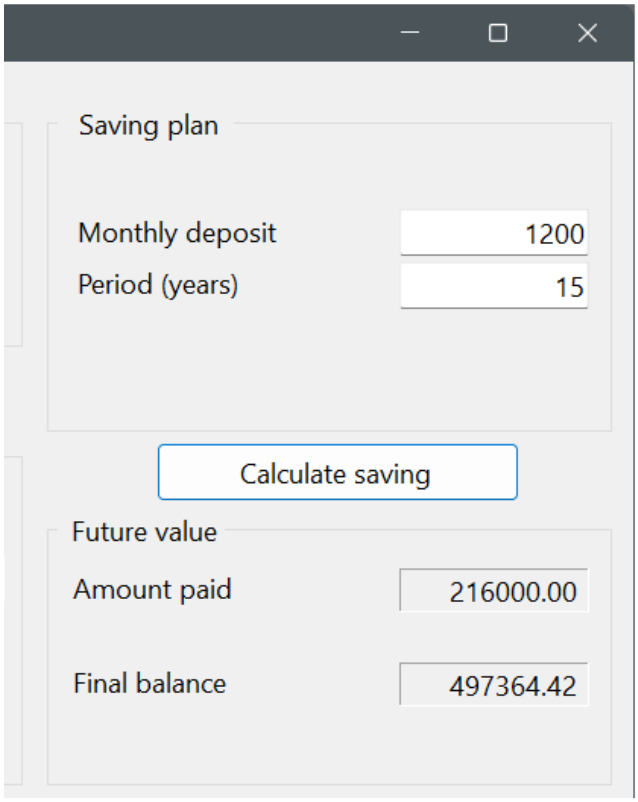
1. MainForm saves input given on the GUI and saves them in bmiCalc, using its set-methods.
2. MainForm calls methods of bmiCalc to perform calculations and receive output
3. MainForm displays the output on the GUI

Complete this part, run the program and make sure that everything works well. Then, move to the next part.

7. Saving Calculator - Saving plan

In compound interest calculation, the interest earned is added to your principal (capital) and the next the interested calculated, you will received interest on interest. Interest can be applied on a daily, monthly or yearly basis. It depends on the type of saving and the policies of the investment company. Long-term monthly saving can make you rich if you invest the money in stocks may make you rich if the growth rate is positive.

Compound interest is one of the most useful concepts in finance. It is the basis of everything from a personal savings plan to the long term growth of the stock market. Although, there are formulas that can be used to compute the effects of inflation, we neglect this computation in this part and neither are we going to account for costs and expense such as the stock fees.



7.1 To simplify the work, let the user input the monthly deposit and the period in years. Use the following constant values for the rest of variables.

Growth/interest rate: 10% (0.10) yearly which you can divide by 12 to calculate the monthly interest.

You may use the following algorithm to calculate the future value:

```
Let numOfMonths = years * 12.

balance = 0  //future value

Loop from 1 to numOfMonths:
   interestEarned =  rate  * balance
   balance += interestEarned + monthly saving
```

Display balance as the final balance (see the above figure).
Amount paid = monthly saving * numOfMonths.

7.2 Create a class SavingCalculator. Declare the needed instance variables (monthlyDeposit, interestRate, period, etc).

7.3 Write a method that calculates the future value using the above algorithm. You may of course use other formulas from the financial sources instead of the suggested looping.

7.4 Declare an instance variable in the MainForm for this calculator and do the necessary programming to make things work as shown in the above run example.

```
//Declare and create an instance of the SavingCalculator
   private SavingCalculator savingCalc = new SavingCalculator();
```

8. Functional Requirements

8.1 At program start, the Form should be clear of all design-time texts such as Label1, etc. (Form1 as the title of the form is not accepted). All input boxes (textboxes) and output controls (labels) should be empty at program start. Input boxes may have default values (for example Name = "No name").

8.2 The application should control the user input so it does not crash or give unexpected output for invalid input. Numeric values can be validated using the **TryParse** method of the data type being used, e.g. int.TryParse and double.TryParse.(as in above example). String values can be validated using string.IsNullOrEmpty.

8.3 The user should receive a notification when the input data is invalid. You can use a MessageBox for notifications.

You can copy the following method to the MainFrame class.

```
private double ReadDouble(string str, out bool success)
{
    double value = -1.00;
    success = false;
    if (double.TryParse(str, out value))
        success = true;

    return value;
}
```

You can then use the above method to validate and read data from a textbox, as below:

```
private void ReadSavingInput()
{
    bool success = true;

       double monthlyDeposit = ReadDouble(txtMonthlyAmount.Text, out success);
    if (success)
        savingCalc.SetMonthlySaving(monthlyDeposit);
    else
    {
        MessageBox.Show("Invalid value for monthly deposit!");
        return;  //exit the metod
    }

    //other code
```

8.4 All results should be correct for calculation of BMI and BMR. There are several sites in the Internet that you can check. The following page has both BMI and BMR calculators (BMR is only A and B grades).

https://www.thecalculatorsite.com/health/bmr-calculator.php

9. Structural Requirements

9.1 Textboxes should be used for input and Labels for output and other read-only information. Do not make a textbox work as a label by disabling the control (setting its Enabled property to false).

9.2 All instance variables should be declared as **private**. Public instance variables are **strictly** forbidden.

9.3 Every class and every method in the class should contain a brief but informative documentation in form of comments. Shortage of time is not an acceptable excuse.

9.4 Use appropriate names for all identifiers (names of classes, variables and method). Follow the general coding rules and suggestion available on Canvas.

9.5 **MainForm** should **only** use an instance of each calculator class, and a field for the **name** (name of the user, as it does not belong to any calculator class), as its fields. No other instance variables should be used.

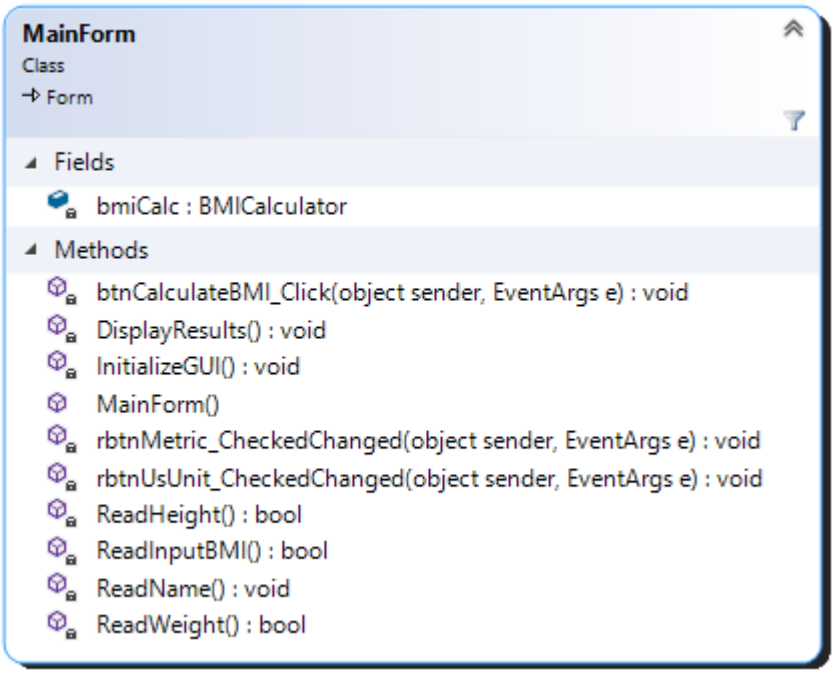
9.6 **MainForm** should work with the user interface. No other class should have access to the MainForm and its components. Instead, the **MainForm** will be using the calculator classes. For more details, see Part 1.

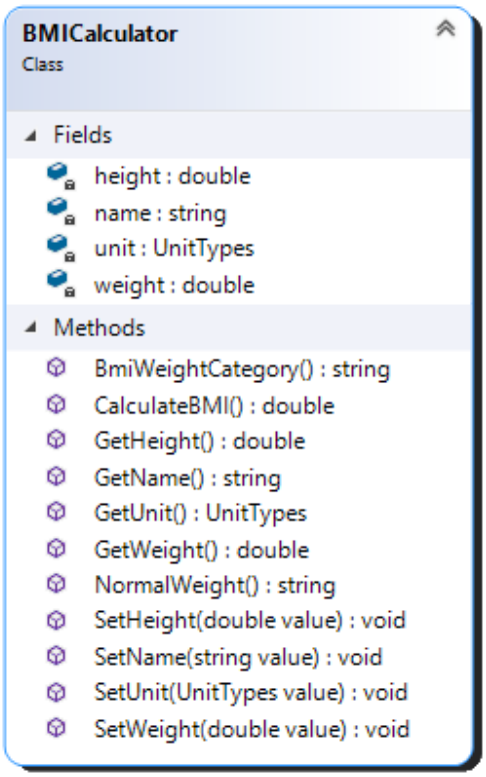
```
public partial class MainForm : Form
{
    private string name = "NoName";

    //Declare and create an instance of the BMI Calculator
    private BMICalculator bmiCalc = new BMICalculator();

    //Declare and create an instance of the SavingCalculator
    private SavingCalculator savingCalc = new SavingCalculator();
```

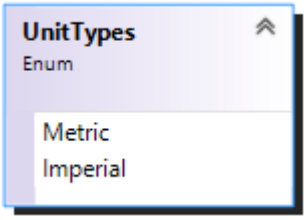
Note: Each set of **RadioButtons must b**e enclosed inside a container component such as a GroupBox. Otherwise different **RadioButtons** sets contained in the same component will be considered as one set. To separate the Female-Male buttons from the Activity Level buttons, you need to put at least one of the sets inside a separate GroupBox control – see the GUI image given earlier.

10. Help and Guidance (Part 1)

The class diagrams shows instance variables and methods of the two classes. You don't have the diagram and write every method. You can make your own implementation and your own methods.







If you are reading the height in cm, you have to change the given value to meters as required by the metric formula:

```
bmiCalc.SetHeight(outValue / 100.0);  //cm -> m
```

For the Imperial unit, the formula works with inches and therefore you must change the given value in feet to inches, and then add it to the inch part.

```
bmiCalc.SetHeight(outValue * 12.00 + inchValue);  //feet -> inches
```

To calculate the normal weight, you can use the lower and upper limits of BMI as below:

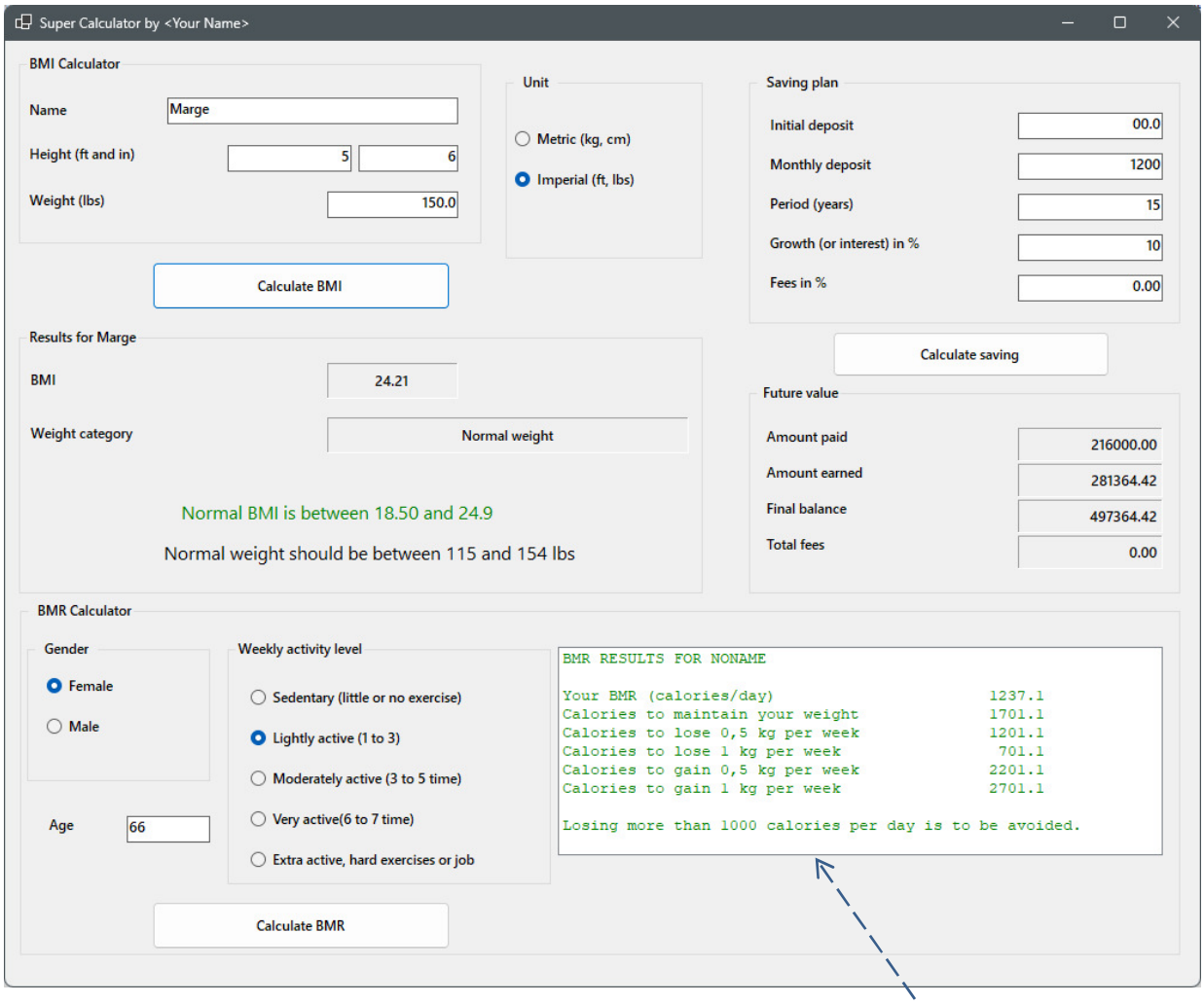
factor = 703 for US and 1 for metric
weight = height * height / factor;
weight1 = weight * 18.50; //low limit
weight2 = weight * 24.9; // high limit

11. Saving Calculator with improved features

This part includes Part 1 but the Saving Calculator has more features as you can see from the GUI below.

11.1 The **SavingCalculator**: all necessary input values are provided by the user. The

calculator performs more calculations and the results are displayed on the GUI



12. BMR Calculator

In our daily life, we need to calculate data to make decisions. There are many areas where a simple calculator facilitates the computation of values greatly. Examples of calculators are geometric calculator, date and time calculator, age calculator, marriage calculator and hundreds of other types.

In this part of the assignment, we are going to add a new feature to our application for calculating a health factor called **BMR-(Basic Metabolic Rate),** controlling daily calories.

Most people should think of their weight and the daily calories they take to keep themselves healthy. Our BMI and BMR Calculators will help them with these issues.
BMR (Basal Metabolic Rate) is an equation for estimating the number of calories you need to consume each day. The calculation is done according to the Mifflin - St Jeor equation. You can use the following formulas to solve the problem:

12.1 BMR values:

BMR = 10 * weight (kg) + 6.25 * height (cm) - 5 * age (y)

BMR Female = BMR -161
BMR Male = BMR + 5

12.2 BMRs to keep your current weight

maintainWeightBMRs = BMR * activity level factor (last column in table below).
(BMR calculated as in above)

12.3 Activity Level and multiplier factor

The values are to be taken from the table.

Group	Level Name	Description	Factor
0	Sedentary	Little or no exercise	**1.2**
1	Lightly active	Exercise 1 to 3 times a week	**1.375**
2	Moderately active	Exercises 4 to 5 times a week	**1.550**
3	Very active	Exercises 6 to 7 times a week.	**1.725**
4	Extra active	Hard exercise or physical job	**1.9**

12.4 Lose or gain weight:

To lose 0.5 (500 g), you need to cut off 500 and to lose 1 kg, 1000 calories from daily intake.

To lose 500 gr (0.5 kg) a week = maintainWeightBMRs – 500
To lose 1000 gr (1 kg) a week = maintainWeightBMRs –1000
To add 500 gr (0.5 kg) a week = maintainWeightBMRs + 500
To add 1000 gr (1 kg) a week = maintainWeightBMRs +1000