

Assignment 3 – Calculators V2

GUI with Windows Forms

This version is for grades B and A. If you are aiming for grades C and D, you can skip this version and instead implement Version 1 of this assignment, which is available on the Assignment 3 page in Canvas.

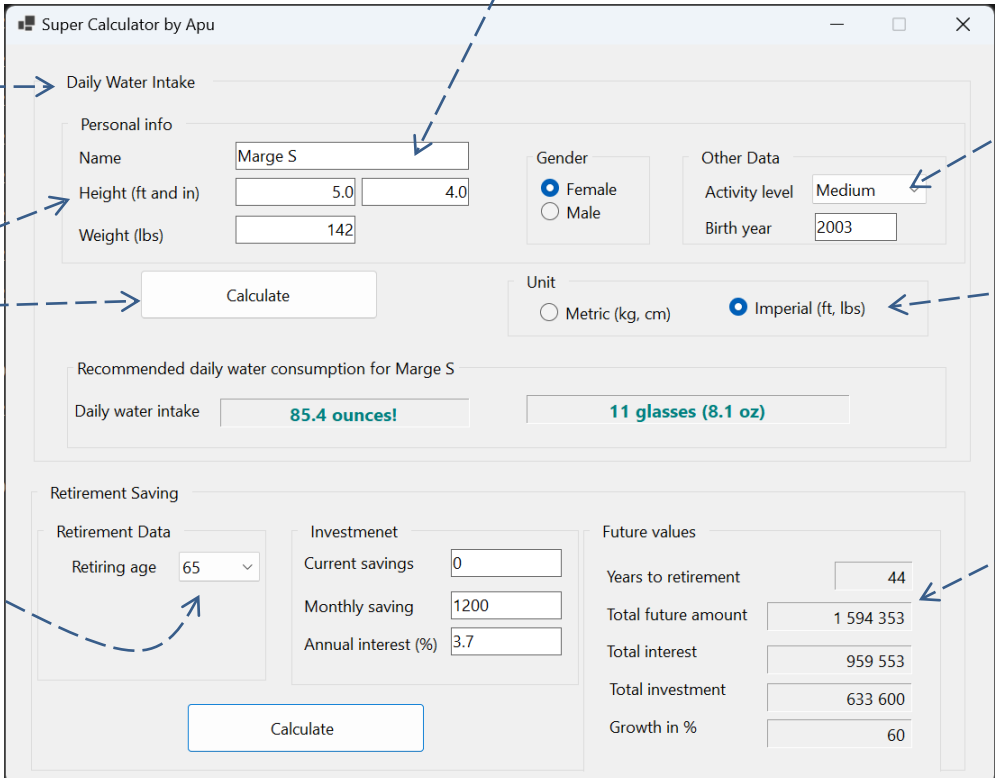
1. Objectives

The main objective of this assignment is to create a Windows Forms-based desktop application with graphical user interfaces (GUIs). More details are provided in a separate document, which you can find on the assignment page in Canvas.

It is **mandatory** to study and review the mentioned document before proceeding with this part. It is crucial that you use a structured solution, employing methods for every single task. You may review the description of Calculator V1, which is more detailed than this description, to get an idea of how to design and structure the solution.

2. The Graphical User-Interface (GUI)

Although you have the freedom to design your GUI in your own way, the following image provides a run-time example showing the Windows Forms controls used for various purposes.



The screenshot shows a Windows Forms application titled "Super Calculator by Apu". The interface is divided into two main sections: "Daily Water Intake" and "Retirement Saving".

Daily Water Intake Section:

- Personal info:** Includes text boxes for "Name" (Marge S), "Height (ft and in)" (5.0 and 4.0), and "Weight (lbs)" (142).
- Gender:** Radio buttons for "Female" (selected) and "Male".
- Other Data:** Includes a "ComboBox" for "Activity level" (Medium) and a text box for "Birth year" (2003).
- Unit:** Radio buttons for "Metric (kg, cm)" and "Imperial (ft, lbs)" (selected).
- Calculate Button:** A button to calculate the recommended daily water consumption.
- Results:** Displays "Recommended daily water consumption for Marge S" as "Daily water intake: 85.4 ounces! 11 glasses (8.1 oz)".

Retirement Saving Section:

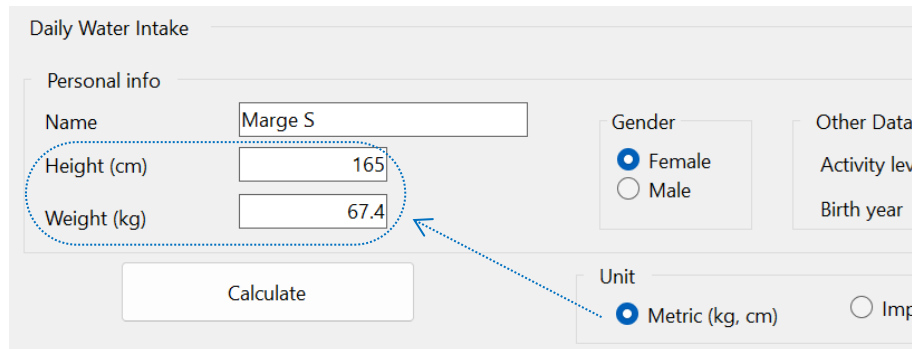
- Retirement Data:** Includes a "ComboBox" for "Retiring age" (65).
- Investment:** Includes text boxes for "Current savings" (0), "Monthly saving" (1200), and "Annual interest (%)" (3.7).
- Future values:** Includes text boxes for "Years to retirement" (44) and a table of calculated values:
 - Total future amount: 1 594 353
 - Total interest: 959 553
 - Total investment: 633 600
 - Growth in %: 60
- Calculate Button:** A button to calculate the future values.

Annotations with arrows point to various controls:

- GroupBox:** Points to the "Daily Water Intake" section header.
- Label:** Points to the "Name" label.
- Button:** Points to the "Calculate" button in the water intake section.
- TextBox:** Points to the "Name" text box.
- ComboBox:** Points to the "Activity level" dropdown.
- RadioButton:** Points to the "Imperial (ft, lbs)" radio button.
- Label:** Points to the "Retiring age" label.

These components are available in the Toolbox in Visual Studio

When the metric unit is selected, the height should be displayed in centimeters (cm) and the weight in kilograms (kg).



To hide a control such as `txtInch`, you can use its `Visible` property and set it to false by writing: `txtInch.Visible = false`. To display it again for the Imperial option, set the property to true.

Note: if you are not going to go for Grade A, draw only the upper part for the daily water intake.

3. Part 1: Daily Water Intake Calculator (Grade B)

Create a class named **WaterIntakeCalculator**, that calculates the recommended daily water intake based on a person's physical data.

The Water Intake Calculator helps users determine their recommended daily water intake based on their height, weight, gender, age, and activity level. The program should support both imperial and metric units and output the daily water intake in milliliters (ml) or ounces (oz), depending on the selected unit. Additionally, the program should convert the calculated volume into the equivalent number of glasses of water, assuming each glass holds 240 ml.

Use the following conversion factors:

Conversions:

Since the given formulas use the metric system, convert the height from feet and inches to centimeters as follows:

```
height = height * 12.0 + inches; //all in inch
height *= 2.54; //inches to cm
```

Similarly, convert the weight from pounds (lbs) to kilograms (kg) using the following formula:

```
weight = weightInLbs * 0.453592); //lbs -> kg
```

To convert milliliters to ounces, use this formula:

```
double ounces = milliliters * 0.033814;
```

To calculate the person's age from the birth year, use:

```
int age = DateTime.Now.Year - birthYear;
```

Base water requirement:

- 3.1 **Base intake (baseIntake)** = 33 ml per kg of body weight.
- 3.2 Adjust the base intake for the **gender**:
 - Male: Increase the base intake by 10% (multiply by 1.1).
 - Female: Decrease the base intake by 10% (multiply by 0.9)
- 3.3 Adjust the base intake based on **age**:
 - Increase the base intake by 10% (multiply by 1.1) for adults under 30.
 - Decrease the base intake by 10% (multiply by 0.9) for adults over 55.
 - No adjustment is made for individuals between 30 and 55.
- 3.4 Adjust the base intake based on **height**:
 - Increase the base intake by 5% (multiply by 1.05) for individuals taller than 175 cm.
 - Decrease the base intake by 5% (multiply by 0.95) for individuals shorter than 160 cm.
 - No adjustment is made for individuals between 160 cm and 175 cm.
- 3.5 Adjust the base intake based on **activity level**:
 - **Low** activity level: No adjustment.
 - **Medium** activity level: Increase the base intake by 20% (multiply by 1.2).
 - **High** activity level: Increase the base intake by 50% (multiply by 1.5)

Calculate the base intake and then multiply it by the applicable factors listed above. Create a method that returns the appropriate multiplier (factor) for each condition.

Structural requirements:

- 3.6 Create a **Person** class to store the physical data for an individual. Declare instance variables only for the user-given input. All instance variables must be private. Implement setter and getter methods to provide controlled access to these fields.
- 3.7 Use an **enum** to represent gender and another **enum** for activity level.
- 3.8 Write a method that calculates a person's age based on their birth year. (You can use **DateTime.Now.Year** in your calculation.)
- 3.9 Create a class **WaterIntakeCalculator** with **Person** as its only field: This class will handle all calculations related to water intake based on the data stored in the **Person** object

```
internal class WaterIntakeCalculator
{
    private Person person = new Person();

    1 reference
    public WaterIntakeCalculator(Person person)
    {
        this.person = person;
    }
}
```

4. Part 2: Retirement Saving Calculator (Grade A)

As you age, planning for retirement becomes increasingly important. In fact, it's wise to start thinking about retirement when you are young and have a steady income.

Create a class to calculate the future value of monthly savings when compound interest is applied. Compound interest, a critical concept in finance, is used in various areas, from personal savings to stock market growth.

With compound interest, the earned interest is added to the principal, meaning that you earn interest on both the initial principal and the previously accumulated interest. The frequency of interest compounding—whether daily, monthly, or yearly—varies based on the type of savings and the policies of the investment firm

4.1 Calculation formulas

To calculate the future value of monthly savings, you can either use a compound interest formula (as stated below) or apply a simple, but accurate, model where you calculate the value at the end of each month and accumulate the results. You may choose either of these two approaches

Alternative 1: Compound interest formula

$$FV = PV \times (1 + r)^n + PMT \times \left(\frac{(1 + r)^n - 1}{r} \right)$$

Where:

- **FV** is the future value of the investment (final balance).
- **PV** is the initial investment (initialInvestment), which is any saved amount already available before starting monthly payments.
- **r** is the monthly interest rate, calculated as the annual interest rate (in %) divided by 12.0/100.
- **n** is the total number of payments (numberOfPayments), calculated as the number of years multiplied by 12.
- **PMT** is the monthly contribution (monthlyContribution), representing the amount saved each month

The formula above calculates the compound interest and balance at the **beginning** of each month. However, alternative 2 (below) performs the calculations at the **end** of each month, which is a more natural approach.

Alternative 2: Manual calculation

In this method, both interest and fees are applied on a monthly basis. Interest is added to the current balance, while fees are subtracted at the end of each month. The values are calculated for each month, from 1 to the total number of months. The following algorithm can be used:

Note: You can set fees to 0 if you do not wish to consider expenses related to savings.

Algorithm:

1. **Balance** = initial deposit + monthly saving amount (this is the starting balance at the end of the first month).
2. **Months** = period (years) × 12.
3. **Monthly interest** = (annual interest rate in % / 100) ÷ 12.
4. **Monthly fees** = (fees in % / 100) ÷ 12.

For each month (from month 1 to the total number of months (Months)):

- Calculate the interest: **interest** = monthly interest × balance.
- Calculate the fees: **fees** = monthly fees × balance.
- Update the balance: **Balance** += interest – fees + monthly saving amount.
- Accumulate the total interest: **totalInterest** += interest.
- Accumulate the total fees: **totalFees** += fees.

The total amount paid is calculated as:

- **Total Amount Paid** = Initial Balance + (Months × Monthly Saving)

The growth rate in % can be count:

- $\text{growthRate} = \text{totalInterest} / \text{balance} * 100;$

4.2 Fields

Declare the necessary fields for the calculation and make use of the **Person** object when personal data, such as birth year and age, is required. For example, when starting the calculations, the **Person** object from the previous part can be passed to this class to include relevant personal information in the calculation process.:

```
public bool Calculate(Person person, int retirementAge)
{
    periodInYears = retirementAge - person.GetAge();

    //rest of the code
```

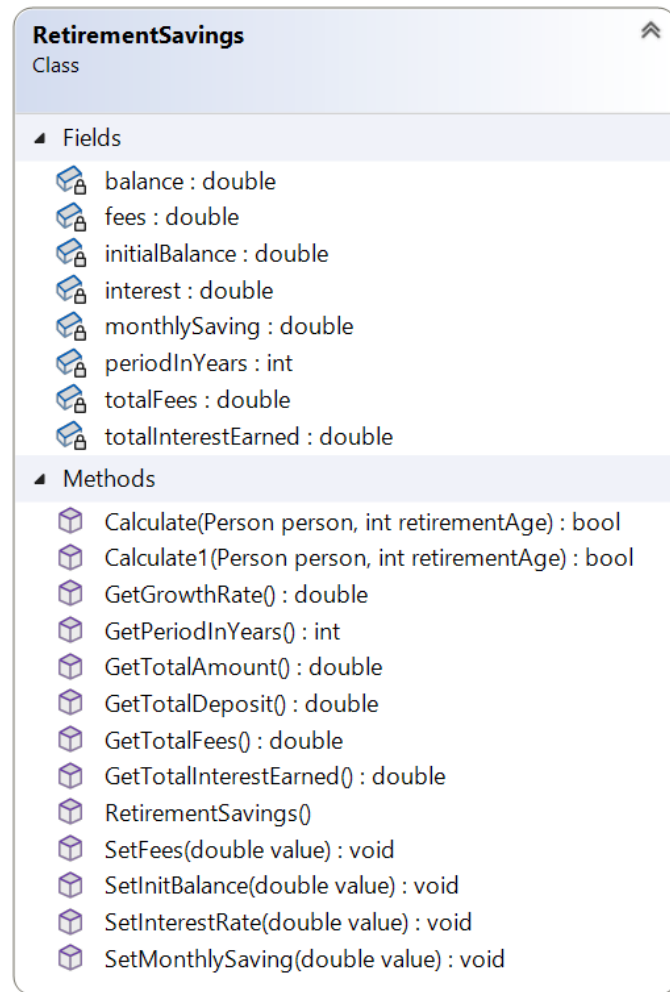
Retirement age

The user should be able to select their desired retirement age, ranging from 62 to 70 (or another range that you prefer), via a ComboBox in the graphical user interface (GUI). The selected retirement age will be used to compute the savings period, as illustrated in the above code snippet.

Class diagram

The class diagram below is provided to give you an overview of the contents of this class. It outlines the structure and key components, but you can customize your solution and name the class members as you wish.

Note that any members related to fees can be ignored, though you are free to include them in your design if desired.



Submission

Ensure that your project compiles without errors and that the program functions satisfactorily. Before submitting, run and thoroughly test your project to confirm everything is working as intended. Submit your assignment in the same manner as you did for the previous one, with all source files and project files properly included in the compressed file.

Good Luck!

Farid Naisan,

Course Responsible and Instructor