



MALMÖ HÖGSKOLA

**Malmö University**  
School of Technology

Programming Using C#, Basic Course

Selection and Iteration Algorithms

## **Assignment 2: TimeTracker**

**Mandatory**

[Farid Naisan](#)

University Lecturer  
Malmö University, Malmö Sweden

Sep 2013



## Assignment 2

### 1. Objectives

The main objective is to provide training in iteration and selection algorithms, things that are a part of a programmer's everyday life.

Several code excerpts are provided in this assignment to prepare you for your future assignments.

It is expected that you follow the guidelines given in the document "General Quality Standards and Guidelines" before beginning with this assignment.

### 2. Description

This assignment consists of a number of sections. In the first two sections, you will be writing a program that sums up numbers. Then you will write a class for converting the local currency to a foreign currency. In the last section, you are asked to write a program to help an employee to remember her/his schedule when she/he will be working weekends and also night shifts.

A menu is presented to the user at the program start. The user selects an item and the program performs the job using an object of the class related to the corresponding task.

In the first parts of the assignment some parts of the code is given to help you get started. If you wonder why the code is given in image format, the reason is to avoid copy paste. To write the code by yourself makes you more aware of the structure and the syntax and gives you the possibility to ponder why things are written in a certain way.

The input and output operations are performed from and to a console window. A different iteration algorithm is used in every part, except in the last section where you are given the freedom of selecting the algorithm by yourself.

The assignment will also use both forms of selection algorithm, i.e. `if-else` and `switch` statements. A good deal of hints and guidance is provided in the first parts to prepare you gradually to get on your own.



**Note:** It is not meant to use arrays in this assignment, but if you are an experienced array user, you may go ahead and do that. We will be working with arrays in a later assignment.

### 3. The Project

The figure here shows a project created in Visual Studio, containing the classes that you are expected to write. Even if you don't use any IDE and write your code in a simple editor, make sure that you gather your source code files under a separate directory for this assignment. It is this folder that will be referred to as “the project” in this document. Before going further make sure you understand the following:

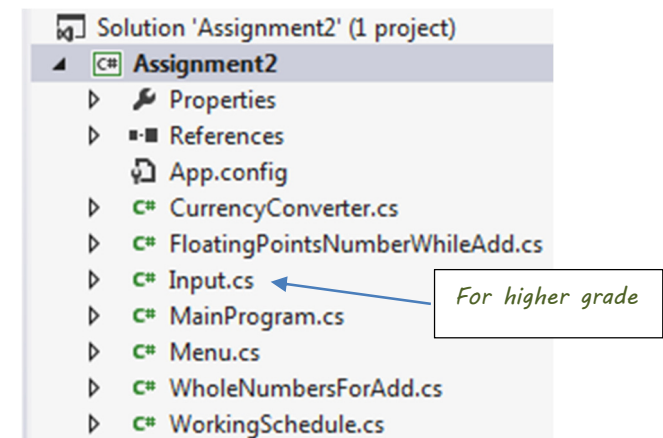
Applications normally have more than one class. Large application may contain hundreds or thousands of classes. In order to show how objects of classes together make an application work, a standard modelling language known as UML (Unified Modeling Language) is used. For more information about UML, search on Google for “UML tutorial” or visit <http://www.omg.org/>.

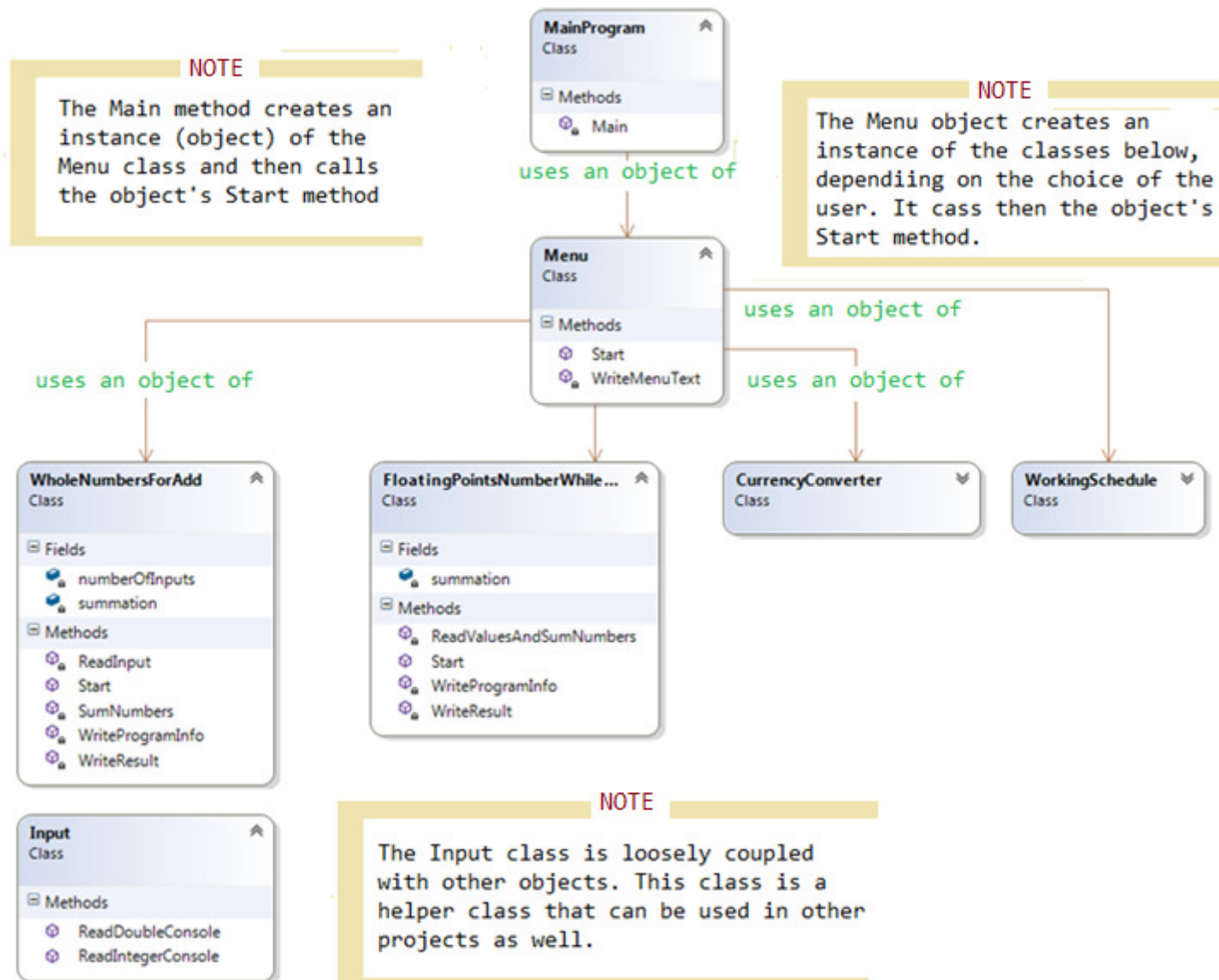
UML consists of many diagrams. One of the probably most used diagrams is called **Class Diagram**. This type of diagram shows the classes and how they are associated with each other.

A class diagram uses a box to represent a class. Each box has three compartments, for class-name, for attributes (fields or instance variables) and for methods of the class.

When an object depends on another object, it is said that the objects have an **association**. The association between classes are illustrated by arrows. The arrows go from the class that uses, i.e. creates an instance of, to the class that it points to. This sort of association is known to have a so called “**Has-a**” relation, meaning that one object has another object as its part. The association of the type “has a” is known as **aggregation**. A **Car** object has for example a number of **Wheel** objects, a **CDPlayer** object etc.

The Class Diagram below shows a solution for this assignment. Study the diagram carefully and try to understand the details. It is actually not very complicated. You can read from the diagram which class creates an instance which another class.







An object of the **MainProgram** class “has an” object of the **Menu** class which in turn has objects of the classes **FloatingNumberWhile**, **ExchangeDoWhile** etc. Notice that the arrow used for this purpose is drawn with open head.

#### 4. Basic Requirements

Here follows the basic and minimum requirements that are required to pass the assignment. So before handing in the assignment go back and read this section to check that you have fulfilled these requirements.

- 4.1. You can implement the application as a console application or a windows form application, or WPF. However, the description given in this document is for a console application and hence we recommend that you write your program as a console application). If you choose to implement your program as a Windows application you need to use the same type of loops, specified later in this document, as if you were to implement a console version.
- 4.2. Each of the classes that you implement in this assignment and in all the future assignments should be saved in a separate file. It is not allowed to have two classes in the same file.
- 4.3. The Main Menu must repeat until the user decides to exit the program.
- 4.4. Control of the user input (numbers, etc.) is not included as mandatory in this assignment for the Pass (G and C) grade. However, the menus must repeat themselves until a valid option is selected by the user.
- 4.5. Repeating menus must be done by using loops. Do not try to make this work, by circular method calls (e.g. call the Start recursively. Those who would like to make use of recursive methods, must know how such methods work and where they are appropriate to implement).
- 4.6. Do not forget to write your name and date on each file.
- 4.7. Do not submit your work before you have compiled and tested so everything works well. Projects that do not compile, will be rejected and returned immediately complementary work.
- 4.8. The classes described below are to be written and saved each in its own code file.



## 5. The Classes :

### 5.1. The MainProgram class (MainProgram.cs)

Do the following in this class:

5.1.1. Write a Main method.

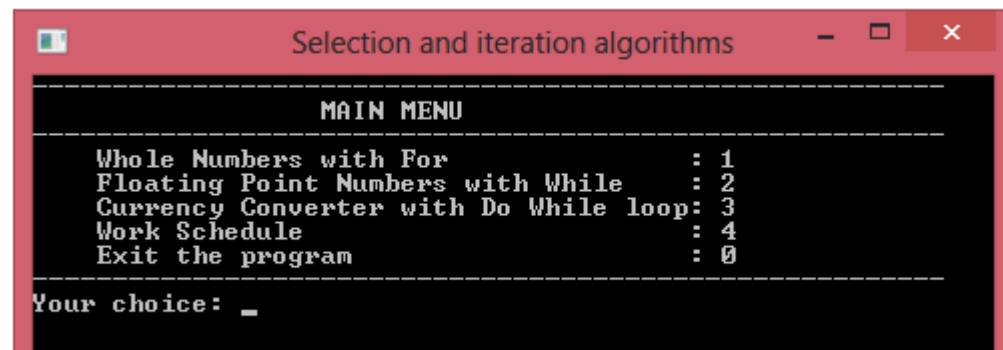
5.1.2. In the Main method create an object of the class Menu (see below) as in in the previous assignment.

5.1.3. Call the menu object's Start method.

### 5.2. The Menu class (Menu.cs)

Displays the menu and reads the menu input. The menu should look something like in the image above. Here you must use **switch-case** to handle the user input.

**Note:** You may change the looks but it should contain all of the 5 different choices. The menu is to be run in an endless loop and does not stop until the user chooses to exit.



### 5.3. The WholeNumbersForAdd class (WholeNumbersForAdd.cs)

The class is responsible for reading in the numbers and a summation of them (adding a negative number will subtract the number). Finally it will show the result when all the numbers have been added.



The Menu object creates an instance of this class when the user chooses option number1, and then calls its Start method. Using a **for loop** program the following in this class:

- 5.3.1. Ask the user for how many whole numbers to add.
- 5.3.2. Let the user give the numbers. Read each of the numbers into your program.
- 5.3.3. Sum up the numbers and show the results.
- 5.3.4. Return to Main Menu.

*This part comes  
from the Menu  
class*

*This is done in this  
class  
WholeNumbersForAdd*

**Mandatory** in this class:

- **for**-loop
- **int** or **long** for the numbers.

```
Selection and iteration algorithms

MAIN MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While loop: 3
Work Schedule                   : 4
Exit the program                 : 0
-----
Your choice: 1

++++++ Summation of whole numbers ++++++
Using a for-statement

Number of values to sum? 4

Please give the value no 1 (whole number): 977
Please give the value no 2 (whole number): 12
Please give the value no 3 (whole number): -459
Please give the value no 4 (whole number): -18
-----
The sum is      512

MAIN MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While loop: 3
Work Schedule                   : 4
Exit the program                 : 0
-----
Your choice: _
```



#### 5.4. The FloatingPointsNumberWhileAdd class (FloatingPointsNumberWhileAdd.cs)

Using a **while loop** (not a do while), read floating point numbers from the user and accumulate them to a total value. When the user writes a zero the loop ends and the summation is viewed on the Console window (adding a negative number will subtract the number).

**Note:** It is the user's responsibility to use the correct symbol for the decimal sign as per the Operating System's Regional Settings. In the example here, it is the period '.' that is the decimal sign. Windows in Swedish usually has the comma ',' as the default decimal sign.

The Menu object creates an instance of this class when the user chooses option number 2, and then calls its **Start** method. Using a **while** loop write code for the following:

- 5.4.1. The user gives a whole or a floating-point (number with a decimal part). Read each of the numbers into your program.
- 5.4.2. When the user writes in a zero value, sum up the numbers.
- 5.4.3. Display the results.
- 5.4.4. Return to the Main Menu.

*This is done in this class*  
**FloatingPointsNumberWhileAdd**

**Mandatory** for this part:

- **while** loop
- **double** type for the numbers.

```
Selection and iteration algorithms

MAIN MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While loop: 3
Work Schedule                   : 4
Exit the program                 : 0
-----
Your choice: 2

++++++ Summation of real numbers ++++++
              Using While-statement

Write 0 to finish!
Make sure to use correct decimal character.
-----
Write an amount to sum or zero to finish: 977
Write an amount to sum or zero to finish: 12.5
Write an amount to sum or zero to finish: -459.5
Write an amount to sum or zero to finish: -18.2
Write an amount to sum or zero to finish: 0
-----
This sum is 511.8

MAIN MENU
-----
Whole Numbers with For           : 1
Floating Point Numbers with While : 2
Currency Converter with Do While loop: 3
Work Schedule                   : 4
Exit the program                 : 0
-----
Your choice: <
```





### 5.5. The CurrencyConverter class (CurrencyConverter.cs)

Use a **do-while** loop (not a While loop). Read floating point numbers and adds them together (as above) and when the user writes a zero the input finishes. The application then asks for the currency name to exchange to and the exchange rate. Finally it displays the exchange rate for buying the currency entered.

This class takes action when the user select the option number 3. Using a do-while loop do the following in this class:

- 5.5.1. The user gives a whole or a floating-point (number with a decimal part). Read each of the numbers into your program.
- 5.5.2. When the user writes in a zero value, sum up the numbers.
- 5.5.3. Ask the user to give the name of the foreign currency to which the sum is to be converted. Read the value.
- 5.5.4. Ask the user to specify the exchange rate and read the value.
- 5.5.5. Convert the sum to a value in the foreign currency using the exchange rate:  
$$\text{value} = \text{sum} / \text{rate}.$$
- 5.5.6. Display the results and return to Main Menu.

**Mandatory** for this class:

- **do – while** loop
- **decimal** type for the numbers.

```
Selection and iteration algorithms
-----
Whole Numbers with For      : 1
Floating Point Numbers with While : 2
Currency Converter with Do While loop: 3
Work Schedule              : 4
Exit the program           : 0
-----
Your choice: 3

+++++ The Currency Converter +++++

Write 0 to finish input!
Make sure to use correct decimal character.

Write an amount or zero to finish: 4055
Write an amount or zero to finish: 160
Write an amount or zero to finish: -29.05
Write an amount or zero to finish: 0

Name of the foreign currency: Euro
Exchange rate: 8.64

-----
The sum is 4.185,95 kr

4.185,95 kr is converted to 484.48 Euro at the rate of 8,64
kr/Euro.

-----
MAIN MENU
-----
Whole Numbers with For      : 1
Floating Point Numbers with While : 2
Currency Converter with Do While loop: 3
Work Schedule              : 4
Exit the program           : 0
-----
Your choice:
```



### 5.6. The WorkingSchedule class (WorkingSchedule.cs)

In this class you are given a chance to analyze the problem and decide what kind of a loop to implement (you can even use a recursive function provided you know how it works). You have to explain your motivation that verifies your choice. Do this using comments in the code file above the class definition. Here is a description of the problem:

The user of this part is an employee who has to work every **3rd** weekend with start from week number **6**. She/he has also to work a night-shift every **5th** with start on week **one**. The schedule is to cover a **one-year** period (can work with more years if you would like to). Assume that the last week of the year is **52**.

Your program should display a list of the week numbers that the user has to work the weekends and the weeks that she/he should work nights

To interact with the user, you may follow the scenario below:

5.6.1. A menu is shown to the user for selecting the type of schedule, **Weekends** or **Nights**.

5.6.2. Display a list of weeks she/he scheduled for the selected option.

#### Mandatory:

- A new menu is to be provided so the user can select the type of the schedule.
- When exiting the menu, the program should return to the Main Menu; it should not exit the program.
- Use methods for every task as in the previous sections.
- Comment your code. Write also about motivation for the specific loop type you have used in your code.

```
Selection and iteration algorithms

YOUR SCHEDULE PROGRAM
Select from the menu which type of schedule you would like to see.

1 Show a list of the weekends to work
2 Show a list of the nights to work
0 Return to Main Menu

Your choice: 1
Your schedule of the above option is as follows:

Week 1      Week 4      Week 7
Week 10     Week 13     Week 16
Week 19     Week 22     Week 25
Week 28     Week 31     Week 34
Week 37     Week 40     Week 43
Week 46     Week 49     Week 52

-----

1 Show a list of the weekends to work
2 Show a list of the nights to work
0 Return to Main Menu

Your choice: 2
Your schedule of the above option is as follows:

Week 6      Week 11     Week 16
Week 21     Week 26     Week 31
Week 36     Week 41     Week 46
Week 51

-----

1 Show a list of the weekends to work
2 Show a list of the nights to work
0 Return to Main Menu

Your choice:
```



**Hint:** It is possible to write one method that can detect and display the schedule weeks. Think what differences the two cases have: start week (6 or 1), week interval (3or 5). The end week is the same.

## 6. Advanced Requirements (Not required for a Pass grade)

The requirements below are only for those who aim at higher grade (VG or A and B). These are in addition to the requirements already described. If you are not going for higher grade, jump to the next section.

6.1. Write a class called **Input** that contains shared (static) methods (methods that can be used without creating an instance of a class) that handles reading of integers and floating point numbers, e.g. two methods called **ReadIntegerConsole** and **ReadDoubleConsole** (if you are doing a Windows Form application then add the functions **GetInteger** and **GetDouble** instead to validate the input sent from the Form class as text and return an **int** and **double** respectively). This class is would work perfectly fine for adding more similar static functions later on. You must divide the code in each class into more than one method.

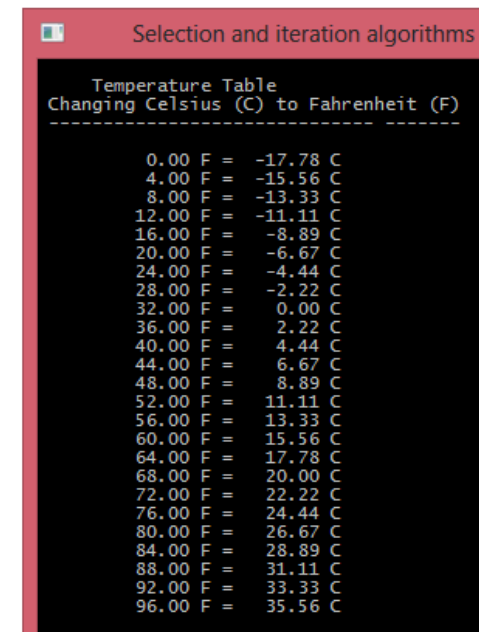
6.2. Call methods of the **InputClass** in all your classes whenever you need to read an integer or double from the console.

6.3. Write a class that calculates temperatures from 0 to 100 degrees Celsius to corresponding Fahrenheit values. Show the results as a table rounded to whole numbers, as in the figure shown here. The following formulas can be used:

```
toCelsius = 5.0 / 9.0 * (valueToConvert - 32)
toFahrenheit = 9.0 / 5.0 * valueToConvert + 32.0
```

6.4. Let this feature be a choice in the Main Menu. (No other user input is required but you may have).

6.5. Keep your code neatly structured and commented. Do not write long methods. Break down your tasks into shorter methods.



Selection and iteration algorithms

Temperature Table  
Changing Celsius (C) to Fahrenheit (F)

0.00	F =	-17.78	C
4.00	F =	-15.56	C
8.00	F =	-13.33	C
12.00	F =	-11.11	C
16.00	F =	-8.89	C
20.00	F =	-6.67	C
24.00	F =	-4.44	C
28.00	F =	-2.22	C
32.00	F =	0.00	C
36.00	F =	2.22	C
40.00	F =	4.44	C
44.00	F =	6.67	C
48.00	F =	8.89	C
52.00	F =	11.11	C
56.00	F =	13.33	C
60.00	F =	15.56	C
64.00	F =	17.78	C
68.00	F =	20.00	C
72.00	F =	22.22	C
76.00	F =	24.44	C
80.00	F =	26.67	C
84.00	F =	28.89	C
88.00	F =	31.11	C
92.00	F =	33.33	C
96.00	F =	35.56	C

Well, if you would like to try writing this application based on the explanations so far, you can stop here and skip the rest of the document. Otherwise, if you feel you need help with your writing your classes, follow the step-by-step instructions below.



## 7. Step by step instructions

In this assignment you will be given a lot of help in form of written code to get you going. In the future assignments, there will also be help documents available for you but try to learn from this assignment on to do things by yourself.

### 7.1. A work plan

When developing applications it is very important to be organized. Think of what to do and divide the big problem into smaller parts. A project as one big problem can be hard to grip but when you start to divide it into smaller and smaller parts and proceed in steps, it gets easier to control. In this assignment we will help you out with the planning and the steps with some ready code, but in the coming assignments you may not receive this much of finished code.

**Note:** Code is given intentionally as screen dumps. This is to prevent the copy-paste technique! I am quite confident that, by writing things by yourself, you will think and learn about the details.

The first step is to create a so called solution and a new project in Visual Studio. A solution is a container for projects and can contain one to many projects. The solution information is saved in a file with the **.sln** extension.

A project in Visual Studio manages classes, files, components and resources that are needed to make a program. The project information is saved with the extension **.csproj** and it is this or the **.sln** file you open when you load an existing C# project.

When you create a new C# project a solution is automatically created for you so for the time being you do not need to focus on the solution. If there is no solution file for a project then it will be automatically created when you open the project.

Start Visual Studio and choose "Create Project". Choose C# and Console application. Choose a folder to save to (or if you are happy with the default folder just check the path to the folder so you remember it later on) Write a name for your project, Assignment2 for instance, and press OK.

A tool or application like Visual Studio that integrates many development tools in one application is called IDE (Integrated Development Environment). Therefore, we may refer to Visual Studio as IDE.

1. Visual Studio has now prepared some default classes for you; one such class is **program.cs** in which the IDE has written a Main method. You can see it in the solution explorer that is normally located to the right of the screen. You can also see the project named Assignment2 and above it the solution also named Assignment2.
2. Rename the **program.cs** to **MainProgram.cs** in the Solution Explorer by right clicking the file name and choosing rename. Read the pop up window and answer yes. In the method Main we will later on create an object of the class Menu.
3. The next step will be creating the Menu class.

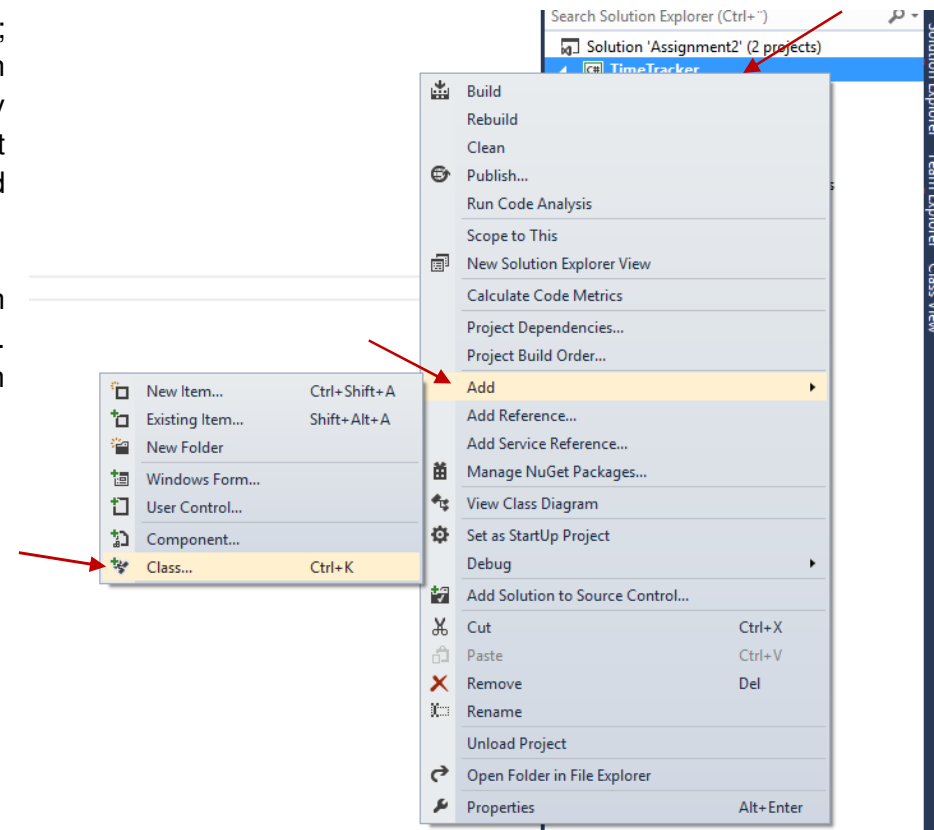
## 7.2. The Menu class (Menu.cs)

The menu class should do the following:

- Display the menu
- Read input from the user and save it in class variables
- At this time when you do not have all your classes written, display the message “Your selection was xx, and that is not implemented yet”. This is only for the purpose of testing – you don’t have to do that.

We will then add functionality to it after we have completed the other required classes. The menu class will show the menu with the five available choices 0-4 as illustrated before. The application will then wait for input from the user. It will then create an object of the class that you have written for that menu choice and call the **Start** function for that object. Proceed as follows:

1. Right click on the project in the Solution Explorer and select **Add**, then Class.





2. Name the class and press Ok.
3. Write a method in the menu class with the name **Start**. Prepare a construct in which you can then create an instance a class, depending on the user's choice.
4. The loop should show the menu, wait for the input (choice) and then:
  - a. Create an object of required class.
  - b. Call the object's Start method.
5. When object's Start menu has done its job, the execution will automatically return to Main Menu.
6. The menu should call a separate method, **WriteMenuText** that is responsible for presenting the text to the Console.
7. The **WriteMenuText** method should display the menu and the input prompt.
8. Don't forget to save often!
9. It is time to run the application for the first time. Since you have not written the function **WholeNumbersForAdd** (line35) you need to comment the lines 35 to 38 until later. Instead write a message to the user informing that this selection is not complete yet.

```


5 namespace TimeTracker
6 {
7     /// <summary>
8     /// Menu
9     /// This class handles all about displaying a menu to the user.
10    /// The user is given a number of choices. For each choice, an action
11    /// is taken by the object of the class.
12    /// The Menu repeats itself until the user wants to exit by choosing the
13    /// option zero.
14    /// </summary>
15    public class Menu
16    {
17        public void start()
18        {
19            int choice = -1;
20
21            while (choice != 0)
22            {
23                WriteMenuText(); //Show the menu
24                //Read user's choice
25                choice = int.Parse(Console.ReadLine());
26
27                //Depending on the value of the choice, create an instance of the
28                //the class displayed on the menu.
29                switch (choice)
30                {
31                    case 1: //Menu item 1 (The for-statement)
32                    {
33                        // Declare a local reference variable and
34                        // create an instance of WholeNumbersFor
35                        WholeNumbersForAdd sumObj = new WholeNumbersForAdd();
36
37                        //call the objects start method
38                        sumObj.Start();
39                        break;
40                    }
41                    // Continue...
42                }
43            }
44        }
45    }
46 }

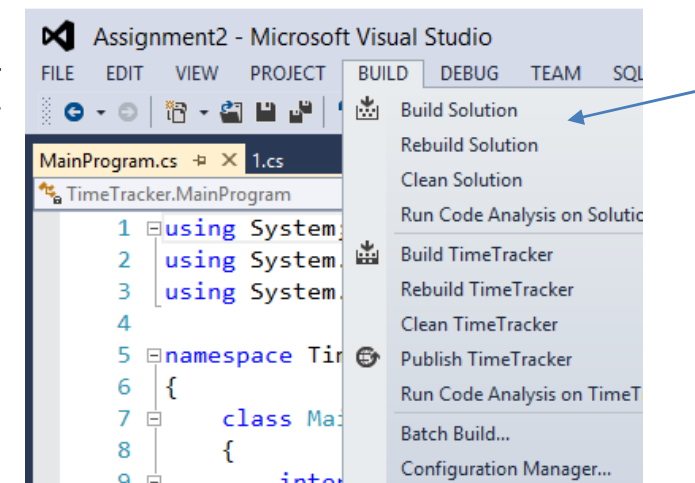
```



10. Go back to the **MainProgram** and create an object of the Menu (even though it is not completely finished yet). Call the Start function of the object (see code below).

```
/// <summary>
/// Starts the application
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    Menu menu = new Menu();
    menu.Start();
}
```

11. Build the application (compile and link) by selecting Build Solution or Rebuild Solution via the Build menu. You will be notified by the compiler if you have errors. Fix the errors and try again.
12. Run the application by clicking the icon  or pressing the key F5.
13. Test your application so it works well for the code you have written so far. Test for example so that the program only exits when the user enters a 0. When you are happy with the functionality, continue to the next step.





### 7.3. The Input Class.

*This class is not required for a Pass (G or C), but it is required for VG or A, B grades – if you are not aiming at a higher grade, skip this class and go to the next section!*

You can expect that a user may input invalid values intentionally or accidentally. If the user enters a text that does not form a number (for instance “2o5”) when you expect a number, the application will terminate abnormally when you try to convert the text into a number in your code. This is very annoying for the user and gives a bad user experience. So before converting the text to a number it is always better to check that it really is a number. Since this is a bit advanced you are given the code needed to do this.

The methods in this class are of the type **static**. This means that you do not need to make an instance of the class to use them.

```
public void Start()
{
    int choice = -1;

    while (choice != 0)        //Loop until the user writes a 0
    {
        WriteMenuText();      //Display the menu

        //Read the user input (0-4)
        choice = Input.ReadIntegerConsole();
    }
}
```

```
namespace Assignment2
{
    public class Input
    {
        /// <summary>
        /// Reads an integer from console untill a correct integer is recieved.
        /// </summary>
        /// <returns>An integer</returns>
        public static int ReadIntegerConsole()
        {
            int input;
            if (int.TryParse(Console.ReadLine(), out input))
                return input;
            else
                Console.WriteLine("Wrong input. Please try again: ");
            return ReadIntegerConsole();
        }

        /// <summary>
        /// Reads an double from console untill a correct double is recieved.
        /// </summary>
        /// <returns>An double</returns>
        public static double ReadDoubleConsole()
        {
            double input;
            if (double.TryParse(Console.ReadLine(), out input))
                return input;
            else
                Console.WriteLine("Wrong input. Please try again: ");
            return ReadDoubleConsole();
        }
    }
}
```

1. Create and complete class **Input** as shown here.
2. Change the **Menu** class to use the **ReadIntegerConsole** function instead of reading directly from the console as shown above.
3. Now that you have a class that guaranties correct numerical input, integer or a double, you can use its methods in the rest of the application whenever you need to read a numerical value.





### 7.4. The WholeNumbersForAdd Class (for-statement)

In this part we are going to work with the menu choice number 1 and get it to work. The class is going to sum a number of whole numbers that is collected from the console window. The class first asks for how many numbers to sum and then reads that many integers from the console.

1. Create a new class with the name **WholeNumbersForAdd** and save it. The class will be responsible for:  
  
Reading input  
  
Calculating the sum.  
  
Display the sum.
2. The code that follows can be used as a template for this class and the following classes.
3. Write the code and complete where it is marked. It is a requirement for this class to use a **For-loop**. When you know how many iterations is needed on forehand the For-loop is the most suitable loop to use.

The code continues to the next page:

```
1 //WholeNumbersForAdd.cs
2 //Created: Farid Naisan, 2013-06-01
3 //Revised:
4
5 using System;
6
7 namespace TimeTracker
8 {
9     /// <summary>
10    /// This class takes care of the whole process of
11    /// (1) reading input from the console window,
12    /// (2) performing the calculation and
13    /// (3) printing the results to a console window.
14    /// Let objects take care of everything that belongs to the object!!
15    /// </summary>
16    public class WholeNumbersForAdd
17    {
18        //Declare a variable (aka field, instance variable, or attribute)
19        private int numOfInput; //num of values to be added
20        private int sum; //result of the summation
21
22
23        //public void-method that performs the whole process
24        public void Start()
25        {
26            //Call the method which writes the program info, title, etc.
27            WriteProgramInfo();
28            ReadInput();
29            SumNumbers();
30            ShowResults();
31        }
32
33        //void-metoden that reads user input
34        private void ReadInput()
35        {
36            // Determine how many numbers there are to be added
37            Console.Write("Number of values to sum? ");
38            numOfInput = int.Parse(Console.ReadLine());
39            Console.WriteLine(); //blank line
40        }
41    }
```



4. When you have completed the code, you need to change the Menu class to initiate an object of this class and then run the Start method, as it is written in the sample code given earlier.
5. Rebuild(F5) and run the application again to test it.
6. Fix all errors and make sure everything works so good so far, before moving on to the next step.

```
42 private void WriteProgramInfo()
43 {
44     //you can use \n to put a blank line or use Console.WriteLine();
45     Console.WriteLine("\n\n ++++++ Summation of whole numbers ++++++");
46     Console.WriteLine("                Using a for-statement\n");
47     Console.WriteLine();    //blank line
48 }
49
50 //void-method that sums upp the numbers as they are read
51 //and the results are stored in the instance variable sum.
52 private void SumNumbers()
53 {
54     //Local variables
55     int index;    //counter variable
56     int num = 0;  //stores the value that the user gives
57
58     // A for-statement that iterates
59     for (index = 0; index < numOfInput; index++)
60     {
61
62         //WRITE YOUR CODE HERE
63
64     }
65 }
66 }//SumNumbers
67
68 private void ShowResults()
69 {
70     //print results to the console window
71     Console.WriteLine("-----\n");
72
73     //Note: 0 is the variable number,i.e. {0} will be
74     //substituted by the value stored in sum. \t is for tab
75     Console.WriteLine("The sum is \t{0}", sum);
76
77 }
78 }//class
79 }//namespace
```



### 7.5. The FloatingNumbersWhileAdd Class (while-statement)

In this section, the program lets the user feed in values that can be real numbers or integers. The program does not require the user to specify the number of values to be read. It stops reading when the user writes a zero value as shown in the screen dump image given earlier. A **while** statement is the best choice when the number of iterations is not known.

1. Create a new class **FloatNumbersWhileAdd.cs**.
2. The code for this class is provided on the next page. Your job is to complete the marked blocks. Note: Some code comments are made hidden to fit the image into the page.

**Hint:** When you compare two floating point numbers, it would be a good idea to compare the numbers rounded to a number of decimal places. Although for small numbers the comparison will work without rounding off the number, here is a recommended way. In the example below, comparison of a number to zero is done to a 7 decimal positions.. It is assumed that a number that is like 0.0000004 is practically zero. Math is a class from the System namespace that has many useful mathematical functions ready to use. Round is one of them.

```
//ReadInput is only an example
double number = ReadInput();

if ((number == 0.0))
{
    //code
}
```

can be written as

```
//ReadInput is only an example
double number = ReadInput();

if ((Math.Round( number, 7) == 0.0))
{
    //code
}
```

```
public class FloatingNumbersWhileAdd
{
    //Declare a variable (aka field, instansvariable, or attribute)
    private double sum; //result of the summation

    public void Start()
    {
        //Call the method which writes the program info, title, etc.
        WriteProgramInfo();
        ReadInputAndSumNumbers();
        ShowResults();
    }
    //Write a welcome text to the user
    private void WriteProgramInfo()
    {
        ➡ //ADD YOUR CODE HERE
    }

    private void ReadInputAndSumNumbers()
    {
        //Read a number.If the value is given as 0, end the iteration
        //otherwise accumulate the results in the insans-variable sum
        bool done = false;

        while (!done)
        {
            ➡ //ADD YOUR CODE HERE
            //Read a number from the Console (Input.ReadDoubleConsole can be called)
            //Check if the number is zero
            //    If yes, set done to true (to end the iteration)
            //    Else, add the number to Sum (add,not overwrite)

        }
    }
    private double ReadInput()
    {
        Console.WriteLine("Write an amount to sum or zero to finish: ");
        double num = double.Parse(Console.ReadLine());
        return num;
    }
    //Show results to the user
    private void ShowResults()
    {
        ➡ //ADD YOUR CODE HERE
    }
} //class
} //namespace
```



### 7.6. The CurrencyConverter Class (do-while statement)

In this part we first sum up a number of values as in the previous part. The difference is that we now deal with values representing a currency. We also introduce a new feature and that is to convert the accumulated amount in the local currency to a given foreign currency. To simplify the problem, the name of the currency and the exchange rate are given by the user.

No code is given for this part as there are only a couple of difference between this part and the previous part. Use a do while for the iteration and the data type `decimal` for the currency values.

### 7.7. The WorkingSchedule Class

With the training and the skills you have earned so far, you are now going to write this class without any guidance.

Refer to the description and the run example provided earlier (5.6).

#### Remember:

A **for**-loop is used when the number of iterations is known on forehand.

A **while** loop is used when the number of iterations is not known. The loop will continue until one or more expressions are valid. The expression(s) is tested before the loop runs.

A **do while** loop is used during the same conditions as a **while** loop. The difference is that a do-while loop is executed at least once. The expression(s) are validated after the first iteration.

### 7.8. Some extra hints

A special way of formatting values into a string can be used with both **Console.Write** and **Console.WriteLine** which gives a lot of possibilities.

```
Console.WriteLine("{0:C} can be exchanged to {1:f2} {2} with the exchange rate {3:C}/{4}.",  
    sum, foreginAmount, currencyName, rate, currencyName);
```



The expression **{0:C}** works like this: The compiler removes the brackets { }, gets the value from the variable 0 in the list and replaces the 0 with that. C tells that the variable is a currency. The runtime compiler adds the symbol for the local currency defined in Windows Regional Settings (Control Panel), **f2** tells the compiler that the number is a floating point number and that it should be rounded off to two decimal places.

Some more hints:

number += 1 is the same as: number = number + 1

text = "James " + "Bond"; will give the result "James Bond". The character '+' concatenates (puts together) strings. You can also use it as follows:

text = "James ";

text += " Bond"; //a blank char at the beginning

and the result will be the same as above "James Bond".

## 8. Submission

Compress all your files include the Properties folder using a **zip** or **rar** format (as explained in the document "Quality Standards and Guidelines"). Go the Assignment page (where you downloaded this document) in Its L, click the Submit Answer button and upload the compressed file. Do not upload singles files and do not send your solution by email even when Its L is not down (doesn't happen often). Its L is the only way to make your submission and it is there, your teachers can record your grade.

## Good Luck!

*Programming is fun. Never give up. Ask for help!*

**Farid Naisan,**

Course Responsible and Instructor