Journal

Programming & Problem Solving

With Jacob Van Silfhout – Level 5

# Start of Semester 2

WEEK 1 & 2

Introduction to the course outline of what we are to gain, achieve, produce and collaborate.

Weekly structure of course subjects that we will cover through to End of November. Including assessment and test dates.

Started off with GitHub. Online cloud storage and networking of programming and coding projects that we will be producing throughout the semester.

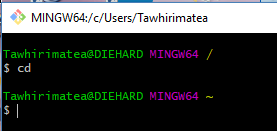
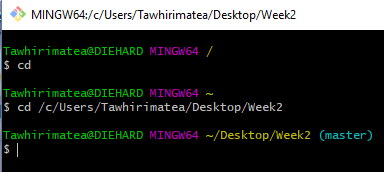
We started creating a GitHub account to push our codes, and any files and folders to, as well as pull them back to our computers when want.

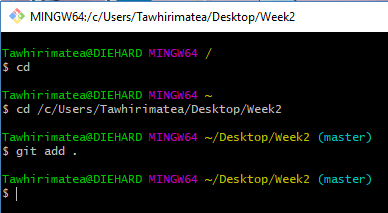
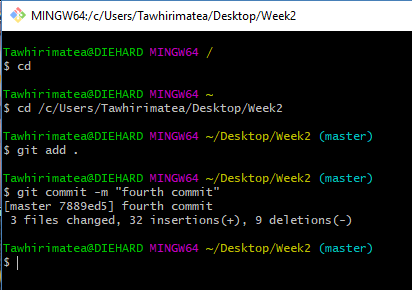
I previously created an account in semester 1 Level 4, but forgot the user name and password, So I had to create another account using my own personal email address and new username.

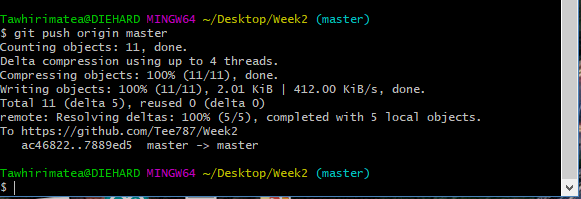
I had to familiarize myself with using GitHub again, as well as Git Bash to push and pull code.

Creating new repository folders, using Git Bash to pull the repository to my desktop creating a folder for my work to be saved to, and to push this work back to Git Hub account to be stored and accessed.

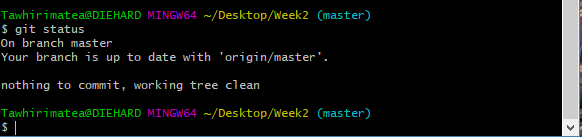
The one part I was stuck on was “how to push a different folder up to Git Hub. The solution I found was to type into Git Bash: “cd” (change directory), and hit enter. This is what reset Git Bash to begin to accept pushing a new folder to Git Hub. I was pushing a folder into another folder without changing the directory. (Nesting)

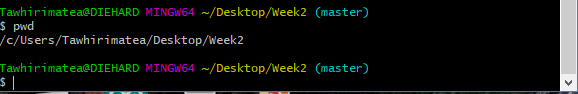
 



Checking the “status” of the folder I pushed up to my GitHub account



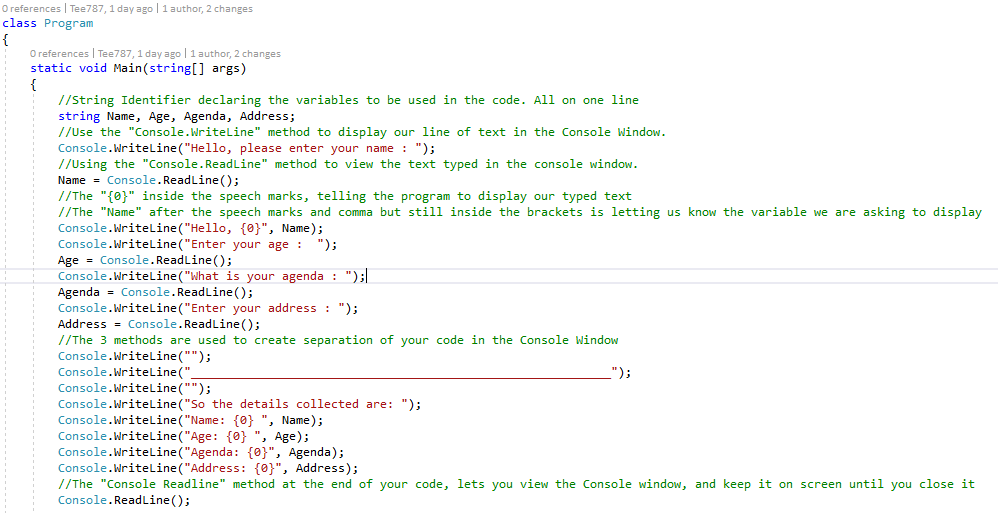
Also learnt how to check the directory path typing in: “pwd” and the status of whether your work was pushed successfully to GitHub.



Given task exercises to attempt to develop a basic programming code.

First task was to develop a program asking the user for their “name, age, agenda and address”. Then had to display as all the information gathered on the console window.

Here is what my code looked like:



Task 2 we had to develop a code for mathematical equations. Using the “Convert.ToInt32” method, along with the “if” and “else” methods, along with “==” characters.

Here are some snippets of these methods used, and the full code.

Convert.ToInt32 function used to convert the string to a integer, number



If statement



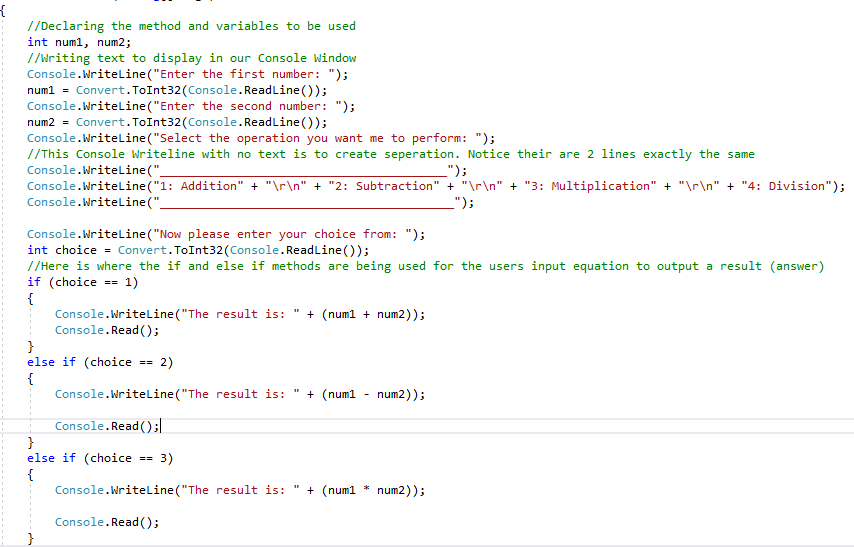
Else if Statement



Next Exercises:

Task.1

Full working code:

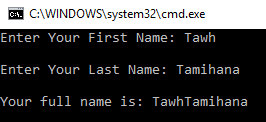
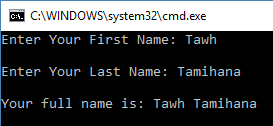


Task 3 we had to develop a code asking the user for the first and last name, and then displaying their full name in the console window.

I used the “Convert.ToString” method to convert the variable string to variable the full text variable function. E.g. Fname and Lname = First name and Last name

There was a section in the code that stumped me. I was trying to separate the first and last name with a space between the instead of one whole word.

e.g.

As you can see in the 1st image, there is no space between the First and Last name

But in the 2nd image there finally is.

With a lot of determination and not giving up, I finally found out how to achieve the final result without browsing the internet for the answer.

I was relieved and happy to have found the solution by myself

 As you can see in this image, it was only something minor that had to be added. A space between the speech marks inside the brackets was all it took.

Here is the full working code:



We then put all our work together to create a Menu to select from 3 choices in the Console window using “If” statements, and a “do” and “while” Loop to return to the main menu.

Unfortunately, with my menu, Task 3 /Choice 3 would not work. So have to source help from my Tutor to view my menu, and figure out what I did incorrectly, or what I haven’t added or removed to get it working properly.

I asked for help from my tutor to get my Menu working properly. It turns out that I had too many “if” statements which was why my code was not executing the last task on my menu. So we changed the “if” statement to a “else if” statement so the code can carry on through the menu.

A brace (}) was also in the wrong place while working with the “if” and “if else” statements.

After correcting all of these, my code finally worked!

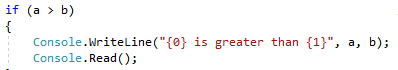
WEEK 3

New exercises to complete for week 3. Again using “if” statements.

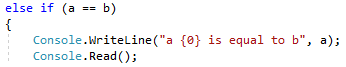
1. **Comparing integers:**



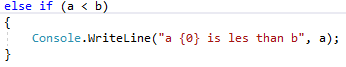
If statement, whether “a” is greater than “b”



If statement, whether “a” is equal to “b”



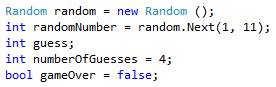
If statement, whether “a” is less than “b”



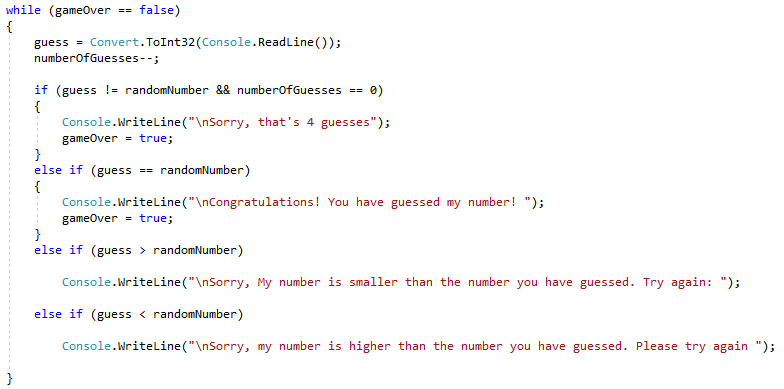
This exercise was easy. Learnt this from last semester in Level 4.

1. **Number guessing**

Using the” Random Class” which is a device that produces a sequence of numbers that meet certain statistical requirements for randomness.



Program code example:



This was also another exercise that I learnt from last semester in Level 4

1. **Grading exercise**

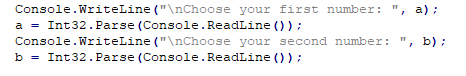
Another exercise from last semester

1. **Odd and Even numbers**

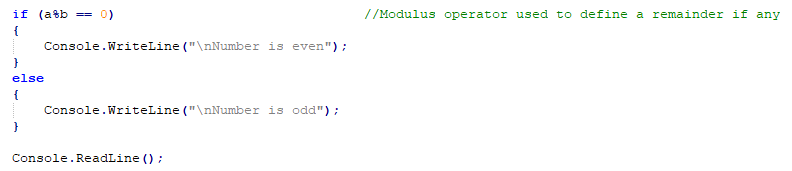
Integers declared with values target set

****

Int32.Parse method used

****

If statements used with a modulus operator to check for a remainder

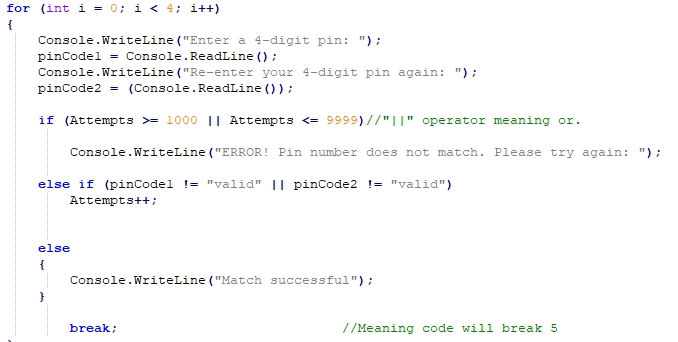
****

1. **New pin program**

Initializing the variables with different methods. Integer for numerical and a string for text output.

****

Using a for statement to define how many attempts to loop our code, along with else if statements to check whether the user input is valid or invalid, and finally an else statement at the end if the users input is successful, our code will end with a break.

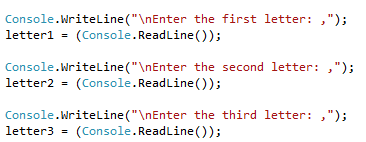
****

1. **Palindrome exercise**

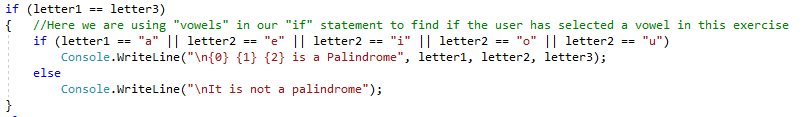
Declaring the variables with a string method used for a text.



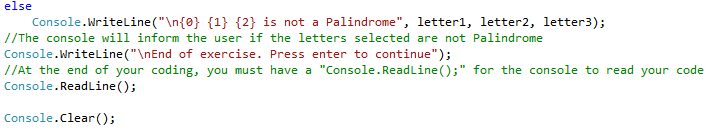
Setting up our code asking for user input



“if” and “else if” statements used. Vowel checking using the “||” “OR” operator



Output to console to inform the user whether a palindrome is present or not

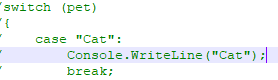


WEEK 4

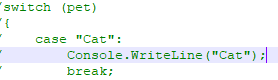
New task exercises this week. Switch and Case statements. Very similar to “if” and “if else”, but less coding needed to execute codes.

1. **Guess my favourite pet**

“switch & case” statements

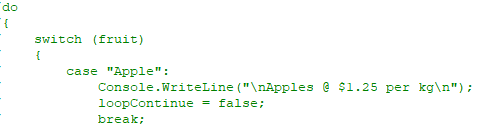


“default” statement



1. **Price of Fruit**

“do while” loop

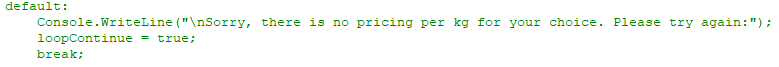


Boolean method used



True or false

‘default’ statement with bool used for true or false



“while” statement at the end

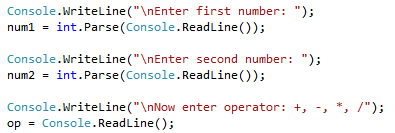


1. **Calculator**

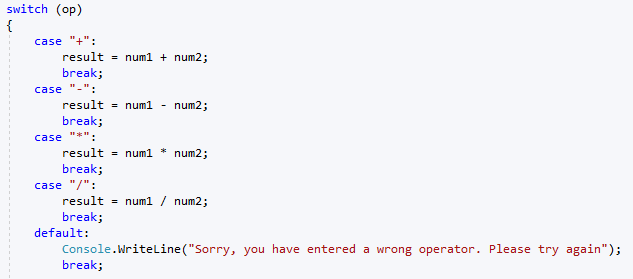
Variables and methods declared:



User input statements:



Switch case statement used:



Result output:



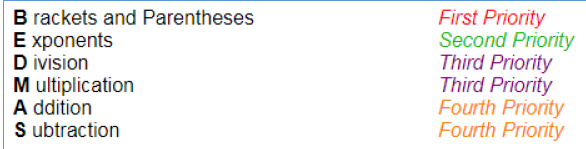
# WEEK 5

New exercises this week. “Loops and ifs”

# WEEK 6

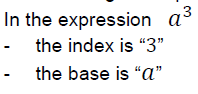
New Exercises: **“Algebra”**

Have to remember this term:



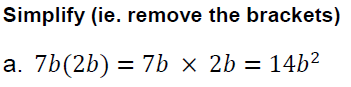
Index (indices)

There is a index and a base. E.g.



To the ‘power of’

Expanding 2^3 = 2\*2\*2, or xy^5 = x\*y\*y\*y\*y\*y, or (xy)^3 = x\*x\*x\*y\*y\*y

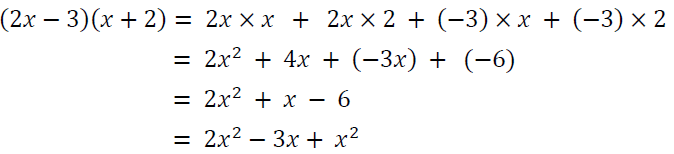


When you have two letters the same after the number, you compress with the expression as above. B to the power of 2.

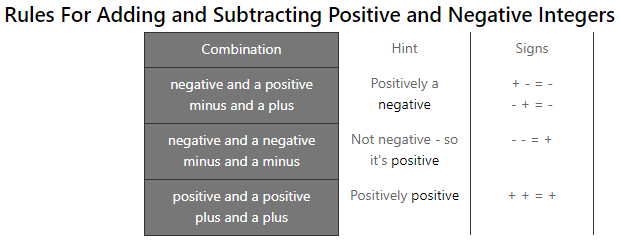
Just in front of the bracket will be a multiplication sign. Hence why the equation reads 7b \* 2b.



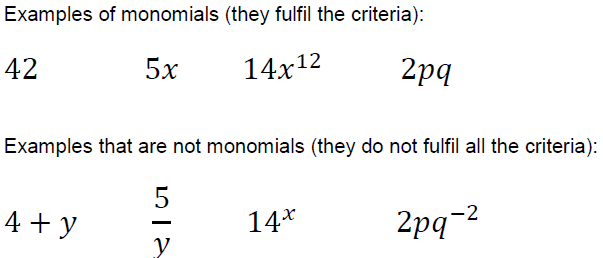
In this example, we are multiplying everything inside the brackets with the leading number in front remembering the addition sign to add in there as well. Simplifying to get our result.



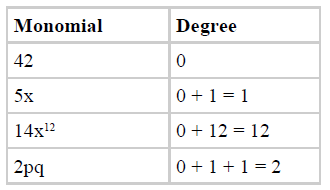
A more complicated example with subtraction and addition, hence remembering about adding our signs together. Here is an example to keep in mind:



Monomials and Polynomials



Here is an example breakdown:



Division Algebra:

2p + V2d + 2ydg

2dg

= 2p + V2d + 2ydg

2dg 2dg 2dg

= p + V2 + y

dg 2g

# WEEK.7

## Arrays

An array is a group of like-typed variables that are referred to by a common name. And each data item is called an element of the array. The data types of the elements may be any valid data type like char, int, float etc. and the elements are stored in a contiguous location. Length of the array specifies the number of elements present in the array. In C# the allocation of memory for the arrays is done dynamically. And arrays are kind of objects, therefore it is easy to find their size using the predefined functions. The variables in the array are ordered and each has an index beginning from 0.

< Data Type > : It define the element type of the array.

[ ] : It define the size of the array.

< Name\_Array > : It is the Name of array.

### Examples:

int[] x; // can store int values

string[] s; // can store string values

double[] d; // can store double values

Student[] stud1; // can store instances of Student class which is custom class

### Array Initialization:

type [ ] < Name\_Array > = new < datatype > [size];

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and Name\_Array is the name of array variable. And **new** will allocate memory to an array according to its size.

### **Examples : To Show Different ways for the Array Declaration and Initialization**

### **Example 1 :**

// defining array with size 5.

// But not assigns values

int[] intArray1 = new int[5];

Above statement declares & initializes int type array that can store five int values. The array size is specified in square brackets([]).

### **Example 2 :**

// defining array with size 5 and assigning

// values at the same time

int[] intArray2 = new int[5]{1, 2, 3, 4, 5};

In the above statement is same as, but it assigns values to each index in {}.

### **Example 3 :**

// defining array with 5 elements which

// indicates the size of an array

int[] intArray3 = {1, 2, 3, 4, 5};

In above statement, the value of the array is directly initialized without taking its size. So, array size will automatically be the number of values which is directly taken.

### Initialization of an Array after Declaration

Arrays can be initialized after the declaration. It is not necessary to declare and initialize at the same time using the **new**keyword. However, Initializing an Array after the declaration, it must be initialized with the new keyword. It can’t be initialized by only assigning values.

### **Example :**

*// Declaration of the array  
string[] str1, str2;*

*// Intialization of array  
str1 = new string[5]{ “Element 1”, “Element 2”, “Element 3”, “Element 4”, “Element 5” };*

*str2 = new string[]{ “Element 1”, “Element 2”, “Element 3”, “Element 4”, “Element 5” };*

### Code Example:

|  |
| --- |
| // C# program to illustrate creating an array  // of integers, puts some values in the array,  // and prints each value to standard output.  using System;  namespace geeksforgeeks {    class GFG {        // Main Method      public static void Main()      {            // declares an Array of integers.          int[] intArray;            // allocating memory for 5 integers.          intArray = new int[5];            // initialize the first elements          // of the array          intArray[0] = 10;            // initialize the second elements          // of the array          intArray[1] = 20;            // so on...          intArray[2] = 30;          intArray[3] = 40;          intArray[4] = 50;            // accessing the elements          // using for loop          Console.Write("For loop :");          for (int i = 0; i < intArray.Length; i++)              Console.Write(" " + intArray[i]);            Console.WriteLine("");          Console.Write("For-each loop :");            // using for-each loop          foreach(int i in intArray)              Console.Write(" " + i);            Console.WriteLine("");          Console.Write("while loop :");            // using while loop          int j = 0;          while (j < intArray.Length) {              Console.Write(" " + intArray[j]);              j++;          }            Console.WriteLine("");          Console.Write("Do-while loop :");            // using do-while loop          int k = 0;          do          {              Console.Write(" " + intArray[k]);              k++;          } while (k < intArray.Length);      }  }  } |

Run on IDE

### Output :

For loop : 10 20 30 40 50

For-each loop : 10 20 30 40 50

while loop : 10 20 30 40 50

Do-while loop : 10 20 30 40 50

One Dimensional Array

In this array contains only one row for storing the values. All values of this array are stored contiguously starting from 0 to the array size. For example, declaring a single-dimensional array of 5 integers :

int[] arrayint = new int[5];

Above array contains the elements from arrayint[0] to arrayint[4]. Here, the new operator has to create the array and also initialize its element by their default values. Above example, all elements are initialized by zero, Because it is the int type.

### Example :

|  |
| --- |
| // C# program to creating an array  // of the string as week days, store  // day values in the weekdays,  // and prints each value.  using System;  namespace geeksforgeeks {    class GFG {        // Main Method      public static void Main()      {            // declares an 1D Array of integers.          string[] weekDays;            // allocating memory for days.          weekDays = new string[] {"Sun", "Mon", "Tue", "Wed",                                         "Thu", "Fri", "Sat"};            // Displaying Elements of array          foreach(string day in weekDays)              Console.Write(day + " ");      }  }  } |

Run on IDE

### Output :

Sun Mon Tue Wed Thu Fri Sat

### Multidimensional Arrays

The multi-dimensional array contains more than one row to store the values. It is also known as a **Rectangular Array** in [C#](https://www.geeksforgeeks.org/introduction-to-c-sharp/) because it’s each row length is same. It can be a **2D-array** or **3D-array** or more. To storing and accessing the values of the array, one required the nested loop. The multi-dimensional array declaration, initialization and accessing is as follows :

// creates a two-dimensional array of

// four rows and two columns.

int[, ] intarray = new int[4, 2];

//creates an array of three dimensions, 4, 2, and 3

int[,, ] intarray1 = new int[4, 2, 3];

### Example :

|  |
| --- |
| // C# program to illustrate creating  // an multi- dimensional array  // puts some values in the array,  // and print them  using System;  namespace geeksforgeeks {    class GFG {        // Main Method      public static void Main()      {            // Two-dimensional array          int[, ] intarray = new int[, ] { { 1, 2 },                                           { 3, 4 },                                           { 5, 6 },                                           { 7, 8 } };            // The same array with dimensions          // specified 4 row and 2 column.          int[, ] intarray\_d = new int[4, 2] { { 1, 2 }, { 3, 4 },                                               { 5, 6 }, { 7, 8 } };            // A similar array with string elements.          string[, ] str = new string[4, 2] { { "one", "two" },                                              { "three", "four" },                                              { "five", "six" },                                              { "seven", "eight" } };            // Three-dimensional array.          int[,, ] intarray3D = new int[,, ] { { { 1, 2, 3 },                                               { 4, 5, 6 } },                                               { { 7, 8, 9 },                                             { 10, 11, 12 } } };              // The same array with dimensions          // specified 2, 2 and 3.          int[,, ] intarray3Dd = new int[2, 2, 3] { { { 1, 2, 3 },                                                    { 4, 5, 6 } },                                                    { { 7, 8, 9 },                                                  { 10, 11, 12 } } };            // Accessing array elements.          Console.WriteLine("2DArray[0][0] : " + intarray[0, 0]);          Console.WriteLine("2DArray[0][1] : " + intarray[0, 1]);          Console.WriteLine("2DArray[1][1] : " + intarray[1, 1]);          Console.WriteLine("2DArray[2][0] " + intarray[2, 0]);            Console.WriteLine("2DArray[1][1] (other) : "                                   + intarray\_d[1, 1]);            Console.WriteLine("2DArray[1][0] (other)"                               + intarray\_d[1, 0]);            Console.WriteLine("3DArray[1][0][1] : "                             + intarray3D[1, 0, 1]);            Console.WriteLine("3DArray[1][1][2] : "                            + intarray3D[1, 1, 2]);            Console.WriteLine("3DArray[0][1][1] (other): "                               + intarray3Dd[0, 1, 1]);            Console.WriteLine("3DArray[1][0][2] (other): "                               + intarray3Dd[1, 0, 2]);            // using nested loop show string elements          Console.WriteLine("To String element");          for (int i = 0; i < 4; i++)              for (int j = 0; j < 2; j++)                  Console.Write(str[i, j] + " ");      }  }  } |

Run on IDE

### Output :

2DArray[0][0] : 1

2DArray[0][1] : 2

2DArray[1][1] : 4

2DArray[2][0] 5

2DArray[1][1] (other) : 4

2DArray[1][0] (other)3

3DArray[1][0][1] : 8

3DArray[1][1][2] : 12

3DArray[0][1][1] (other): 5

3DArray[1][0][2] (other): 9

To String element

one two three four five six seven eight

### Jagged Arrays

An array whose elements are arrays is known as Jagged arrays it means “**array of arrays**“. The jagged array elements may be of different dimensions and sizes. Below are the examples to show how to declare, initialize, and access the jagged arrays.

### Example :

|  |
| --- |
| // C# program to single-dimensional jagged array  // that contains two single-dimensional array  // elements of different sizes.  using System;  namespace geeksforgeeks {    class GFG {        // Main Method      public static void Main()      {            /\*----------2D Array---------------\*/          // Declare the array of two elements:          int[][] arr = new int[2][];            // Initialize the elements:          arr[0] = new int[5] { 1, 3, 5, 7, 9 };          arr[1] = new int[4] { 2, 4, 6, 8 };            // Another way of Declare and          // Initialize of elements          int[][] arr1 = { new int[] { 1, 3, 5, 7, 9 },                           new int[] { 2, 4, 6, 8 } };            // Display the array elements:          for (int i = 0; i < arr.Length; i++)          {              System.Console.Write("Element [" + i + "] Array: ");              for (int j = 0; j < arr[i].Length; j++)                  Console.Write(arr[i][j] + " ");              Console.WriteLine();          }            Console.WriteLine("Another Array");            // Display the another array elements:          for (int i = 0; i < arr1.Length; i++)          {              System.Console.Write("Element [" + i + "] Array: ");              for (int j = 0; j < arr1[i].Length; j++)                  Console.Write(arr1[i][j] + " ");              Console.WriteLine();          }      }  }  } |

Run on IDE

### Output :

Element [0] Array: 1 3 5 7 9

Element [1] Array: 2 4 6 8

Another Array

Element [0] Array: 1 3 5 7 9

Element [1] Array: 2 4 6 8

It’s possible to mix jagged and multidimensional arrays. The jagged array is an array of arrays, and therefore its elements are reference types and are initialized to null.  
**Example :** To Declare and initialization of a single-dimensional jagged array which contains three two-dimensional array elements of different sizes.

|  |
| --- |
| // C# program to single-dimensional jagged array  // that contains three two-dimensional array  // elements of different sizes.  using System;  namespace geeksforgeeks {    class GFG {    // Main Method  public static void Main()  {        int[][, ] arr = new int[3][, ] {new int[, ] {{1, 3}, {5, 7}},                                      new int[, ] {{0, 2}, {4, 6}, {8, 10}},                                      new int[, ] {{11, 22}, {99, 88}, {0, 9}}};        // Display the array elements:      for (int i = 0; i < arr.Length; i++)      {          int x = 0;          for (int j = 0; j < arr[i].GetLength(x); j++)          {              for (int k = 0; k < arr[j].Rank; k++)                  Console.Write(" arr[" + i + "][" + j + ", " + k + "]:"                                                 + arr[i][j, k] + " ");              Console.WriteLine();          }          x++;          Console.WriteLine();      }  }  }  } |

Run on IDE

### Output :

arr[0][0, 0]:1 arr[0][0, 1]:3

arr[0][1, 0]:5 arr[0][1, 1]:7

arr[1][0, 0]:0 arr[1][0, 1]:2

arr[1][1, 0]:4 arr[1][1, 1]:6

arr[1][2, 0]:8 arr[1][2, 1]:10

arr[2][0, 0]:11 arr[2][0, 1]:22

arr[2][1, 0]:99 arr[2][1, 1]:88

arr[2][2, 0]:0 arr[2][2, 1]:9

## Points To Remember :

* GetLength(int): returns the number of elements in the first dimension of the Array.
* When using jagged arrays be safe as if the index does not exist then it will throw exception which is IndexOutOfRange.

# WEEK.8

## Methods:

Methods are generally the block of codes or statements in a program that gives the user the ability to **reuse** the same code which ultimately saves the excessive use of memory, acts as a **time saver** and more importantly, it provides a better **readability** of code. So basically, a method is a collection of statements that perform some specific task and return the result to the caller. A method can also perform some specific task without returning anything.

**Example :**

// Method Name --> GetCircleArea()

// Return Type ---> double

static double GetCircleArea(double radius)

{

const float pi = 3.14F;

double area = pi \* radius \* radius;

return area;

}

**Method Declaration**

Method declaration means the way to construct method including its naming.

**Syntax :**

<Access\_Modifier> <return\_type> <method\_name>([<param\_list>])



In C# a method declaration consists of following components as follows :

* **Modifier :** It defines access type of the method i.e. from where it can be accessed in your application. In C# there are Public, Protected, Private access modifiers.
* **Name of the Method :** It describes the name of the user defined method by which the user calls it or refer it. Eg. **GetName()**
* **Return type:** It defines the data type returned by the method. It depends upon user as it may also return void value i.e return nothing
* **Body of the Method :** It refers to the line of code of tasks to be performed by the method during its execution. It is enclosed between braces.
* **Parameter list :**Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, then empty parentheses () have to use out.

**Method Signature :** Method Signature is defined by mainly two parameters(number of parameters, type of the parameters and order of the parameters), One of them is **Method Name** and second one is its **Parameter list**.

**Method Naming :** Name of a method or a function in any programming language whether in C++ or Java or C# holds a great importance and is mainly used in order to call that method for its execution. For example, findSum, computeMax, setX and getX etc. There are certain pre-defined rules for naming methods which a user should follow :

* The method name must be some kind of Noun or a verb.
* It’s naming should be done in such a way that it must describe the purpose of that method.
* The first letter of the method name can be either small letter or a Capital letter, however, it is recommended to use the capital one.

These rules are not mandatory, but recommendable. Generally, a method has a unique name within the class in which it is defined but sometime a method might have the same name as other method names within the same class as method overloading is allowed in C#.

**The Method Body :** As discussed above the body of the method consists of statements of code which a user wants to perform. After the method has been declared, it is dependent on the user whether to define its implementation or not. Not writing any implementation, makes the method not to perform any task. However, when the user wants to perform certain tasks using method then it must write the statements for execution in the body of the method. The below syntax describes the basic structure of the method body :

**Syntax :**

<return\_type> <method\_name>(<parameter\_list>)

{

// Implementation of the method code goes here.....

}

**Method Calling**

**Method Invocation or Method Calling** is done when the user wants to execute the method. The method needs to be called for using its functionality. A method returns to the code that invoked it when:

* It completes all the statements in the method
* It reaches a return statement
* Throws an exception

**Example :** In the code below, a method named **Sum()** is called.

|  |
| --- |
| // C# program to illustrate  // method calling  using System;  namespace ConsoleApplication1 {    class Geeks {        // Here Sum() method asks for two      // parameters from the user and      // calculates the sum of these      // and finally returns the result.      static int Sum(int x, int y)      {                // there are two local variables              // 'a' and 'b' where 'a' is assigned              // the value of parameter 'x' and              // 'b' is assigned the value of              // parameter 'y'              int a = x;              int b = y;                // The local variable calculates              // the sum of 'a' and 'b'              // and returns the result              // which is of 'int' type.              int result = a + b;                return result;          }        // Main Method      static void Main(string[] args)      {          int a = 12;          int b = 23;            // Method Sum() is invoked and          // the returned value is stored          // in the local variable say 'c'          int c = Sum(a, b);            // Display Result          Console.WriteLine("The Value of the sum is " + c);      }  }  } |

Run on IDE

**Output :**

The Value of the sum is 35

**Method Parameters**

There might be certain situations the user want to execute a method but sometimes that method requires some value inputs in order to execute and complete its tasks. These input values are known as **Parameters** in a computer language terms. Now, these parameters can be either int, long or float or double or char. However, it depends upon the user requirements. The methods in C# can be classified into **different categories** based on return type as well as input parameters.

* **Example Program Without Parameters & Without Return Type**

|  |
| --- |
| // C# program to illustrate method Without  // Parameters & Without Return Type  using System;  namespace ConsoleApplication2 {  class Geeks {        // Here the method 'PrintSentence()'      // neither takes any parameter nor      // returns any value. It simply performs      // the required operations and prints      // the result within it.      static void PrintSentence()      {            Console.WriteLine("No parameters and return type void");      }        // Main Method      static void Main(string[] args)      {            // Method Invoking or Method calling          PrintSentence();      }  }  } |

* Run on IDE
* **Output :**
* No parameters and return type void
* **Example Program Without Parameters & With Return Value Type**

|  |
| --- |
| // C# program to illustrate the method Without  // Parameters & With Return Value Type  using System;  namespace ConsoleApplication3 {    class Geeks {        // This method takes no paramter,      // however returns the result obtained      static int sum()      {          int a = 78, b = 70, add;          add = a + b;          return add;      }        // Main Method      static void Main(string[] args)      {            // Here the calling variable          // is 'getresult'          int getresult = sum();            // Prining the value of          // 'getresult' variable          Console.WriteLine(getresult);      }  }  } |

* Run on IDE
* **Output :**
* 148
* **Example Program With Parameters & Without Return Value Type**

|  |
| --- |
| // C# program to illustrate Method With  // Parameters & Without Return Value Type  using System;  namespace ConsoleApplication3 {  class Geeks {        // This method take the side of      // the square as a parameter and      // after obtaining the result,      // it simply print it without      // returning anything..      static void perimeter(int p)      {            // Displaying the perimeter          // of the square          Console.WriteLine("Perimeter of the Square is " + 4 \* p);      }        // Main  Method      static void Main(string[] args)      {            // side of square          int p = 5;            // Method invoking          perimeter(p);      }  }  } |

* Run on IDE
* **Output :**
* Perimeter of the Square is 20
* **Example Program With Parameters & With Return Value Type**

|  |
| --- |
| // C# program to illustrate Method With  // Parameters & With Return Value Type  using System;  namespace ConsoleApplication4 {  class Geeks {        // This method asks a number from      // the user and using that it      // calculates the factorial      // of it and returns the result      static int factorial(int n)      {          int f = 1;            // Method to calculate the          // factorial of a number          for (int i = 1; i<= n; i++)          {              f = f \* i;          }            return f;      }        // Main Method      static void Main(string[] args)      {          int p = 4;            // displaying result by calling the function          Console.WriteLine("Factorial is : " + factorial(p));      }  }  } |

* Run on IDE
* **Output :**
* Factorial is : 24

**Advantages of using the Methods :**  
There are many advantages of using methods. Some of them are listed below:

* It makes the program well structured.
* Methods enhance the readability of the code.
* It provides an effective way for the user to reuse the existing code.
* It optimizes the execution time and memory space.

# WEEK.9

## Classes:

***Class and Object*** are the basic concepts of Object Oriented Programming which revolve around the real-life entities. A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods(member function which defines actions) into a single unit. In C#, classes support the polymorphism, inheritance and also provide the concept of derived classes and base classes.

**Declaration of class**

Generally, a class declaration contains only keyword **class**, followed by an **identifier(name)** of the class. But there are some optional attributes which can be used with class declaration according to the application requirement. In general, class declarations can include these components, in order:

* **Modifiers:** A class can be public or internal etc. By default modifier of class is *internal*.
* **Keyword class:** A *class* keyword is used to declare the type class.
* **Class Identifier:** The variable of type class is provided. The identifier(or name of class) should begin with a initial letter which should be capitalized by convention.
* **Base class or Super class:** The name of the class’s parent (superclass), if any, preceded by the *: (colon)*. This is optional.
* **Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the **: (colon)**. A class can implement more than one interface. This is optional.
* **Body:** The class body is surrounded by { } (curly braces).

Constructors in class are used for initializing new objects. Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

**Example:**

// declaring public class

public class Geeks

{

// field variable

public int a, b;

// member function or method

public void display()

{

Console.WriteLine(“Class & Objects in C#”);

}

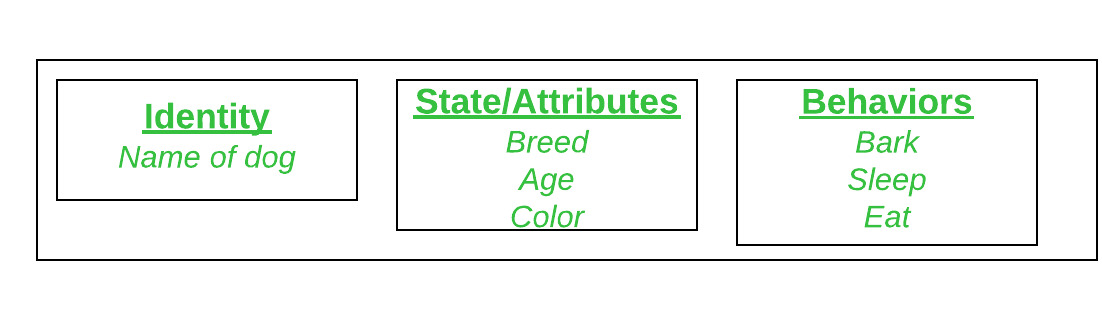
}

**Objects**

It is a basic unit of Object Oriented Programming and represents the real-life entities. A typical C# program creates many objects, which as you know, interact by invoking methods. An object consists of :

* **State:** It is represented by attributes of an object. It also reflects the properties of an object.
* **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.
* **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

Consider Dog as an object and see the below diagram for its identity, state, and behavior.

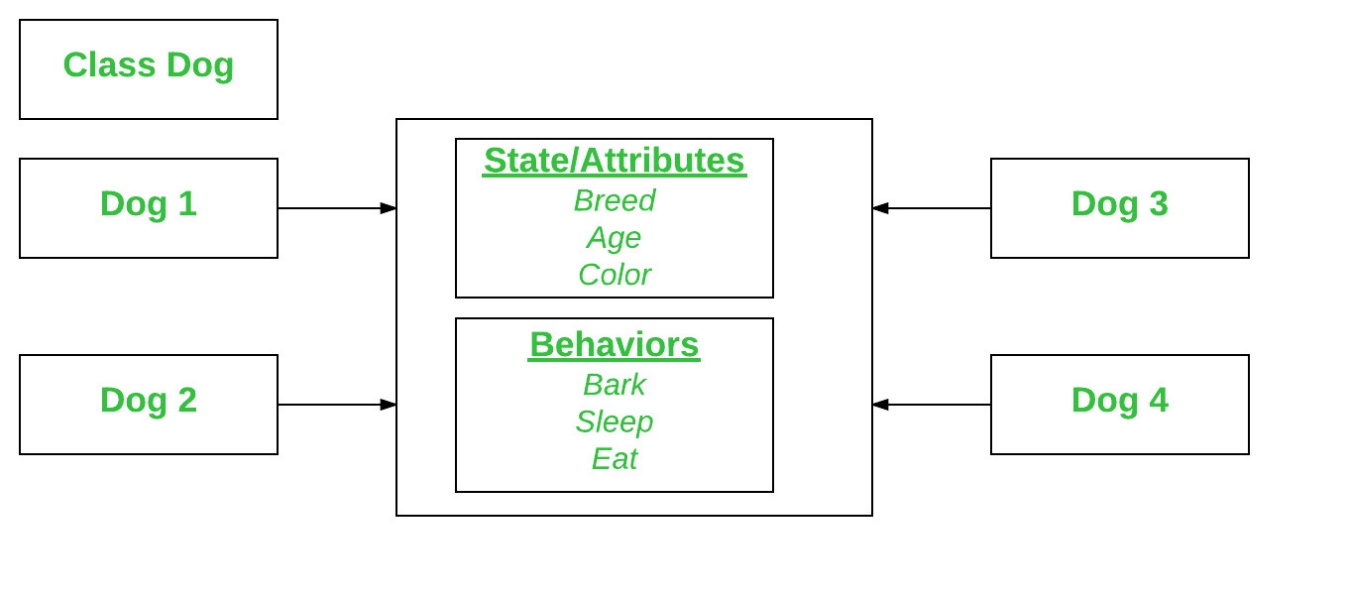


Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

**Declaring Objects (Also called instantiating a class)**

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

**Example:**



As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name.

Dog tuffy;

If we declare reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

**Initializing an object**

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

**Example:**

|  |
| --- |
| // C# program to illustrate the  // Initialization of an object  using System;    // Class Declaration  public class Dog {        // Instance Variables      String name;      String breed;      int age;      String color;        // Constructor Declaration of Class      public Dog(String name, String breed,                    int age, String color)      {          this.name = name;          this.breed = breed;          this.age = age;          this.color = color;      }        // method 1      public String getName()      {          return name;      }        // method 2      public String getBreed()      {          return breed;      }        // method 3      public int getAge()      {          return age;      }        // method 4      public String getColor()      {          return color;      }        public String toString()      {          return ("Hi my name is " + this.getName()                  + ".\nMy breed, age and color are " + this.getBreed()                  + ", " + this.getAge() + ", " + this.getColor());      }    // Main Method  public static void Main(String[] args)      {            // Creating object          Dog tuffy = new Dog("tuffy", "papillon", 5, "white");          Console.WriteLine(tuffy.toString());      }  } |

Run on IDE

**Output:**

Hi my name is tuffy.

My breed, age and color are papillon, 5, white

## Examples and comments:

namespace BankingApp

{

class Account

{

//Auto properties: Note; variable names have to be in Pascal casing.

public string Fname { get; set; }

public string Lname { get; set; }

public string AccountNum { get; set; }

public decimal Balance { get; set; }

//Constructor: This is where you can use the “Class” name only!!

public Account(string fname, string lname, string account, decimal balance)

{

Fname = fname;

Lname = lname;

AccountNum = account;

Balance = balance;

}

//Method: This is where we set our method to be called in the Main method.

public string Deets()

{

return Fname + " " + Lname + " " + AccountNum + " " + Balance;

}

}

//Main Class: This is where we construct our main code to be output to the console.

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Please enter account holders first name: ");

string fname = Console.ReadLine();

Console.WriteLine("Please enter account holders last name: ");

string lname = Console.ReadLine();

Console.WriteLine("Please enter account holders account number: ");

string account = Console.ReadLine();

Console.WriteLine("Please enter account balance: ");

decimal balance = decimal.Parse(Console.ReadLine());

Account a1 = new Account(fname,lname,account,balance);

//Method: here is where we are calling our method from inside our first class structure

Console.Clear();

Console.WriteLine(a1.Deets());

Must make sure we declare our variables with the right statements.