# Object Oriented Programming

Lab Manual 2

## Introduction

After a week of rigorous coding, Welcome back!

**You have learned all about the C# in the previous lab manuals. Let's move on to the next, new, and interesting concepts.**

Students, Object-Oriented Programming is different from Procedural programming as it is about creating objects that combines data in a single unit.

## Learning Objectives:

1. Package the related data as single unit (Class)
2. Create variables (Object) to use the data.
3. Explain Role of the Constructors (Default, Parameterized and Copy Constructors) and Memory Representation of the objects.

## Let's do some coding.

Suppose, we want to store the following data of different students.

1. name
2. matricMarks
3. fscMarks
4. ecatMarks
5. aggregate

Previously, we had used parallel arrays to store unlimited number of records like this.

```
int array_size = 5;
string [] sname = new string[array_size];
float [] matricMarks = new float[array_size];
float [] fscMarks = new float[array_size];
float [] ecatMarks = new float[array_size];
float[] aggregate = new float[array_size];
```

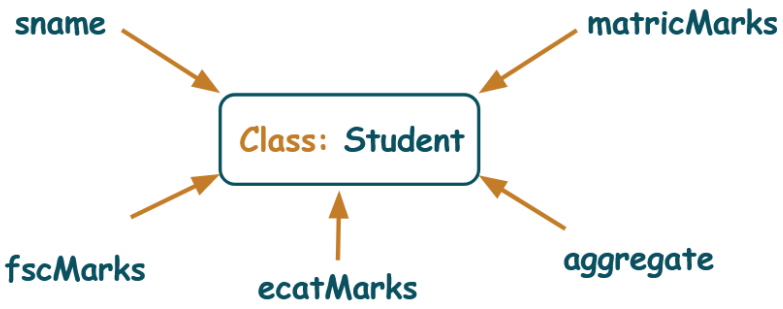
This issue with this approach is that the data of the student is highly correlated but it is stored in different disjoint arrays and linked with the index number.

To understand it more clearly, lets look at the example of 3 siblings who have 3 cabinets to store their belongings (shoes, clothes and books). Siblings were not mature yet therefore they dedicated 1 cabinet for shoes, another for clothes and another for books. 1st rack of each cabinet was for oldest sibling, 2nd for middle sibling and 3rd for youngest sibling.
As the children grew older (became more mature) they understood a better solution which was to take 1 cabinet for every sibling. This solution was more efficient in terms of management and privacy.

Same goes for our Solution. Now, we have become more mature and now, instead of parallel arrays we will see the solution with the Class.

## Class:

A class is a way to structure and organize code in a single unit such that it mirrors the real-world entities and actions you are modeling in your program.

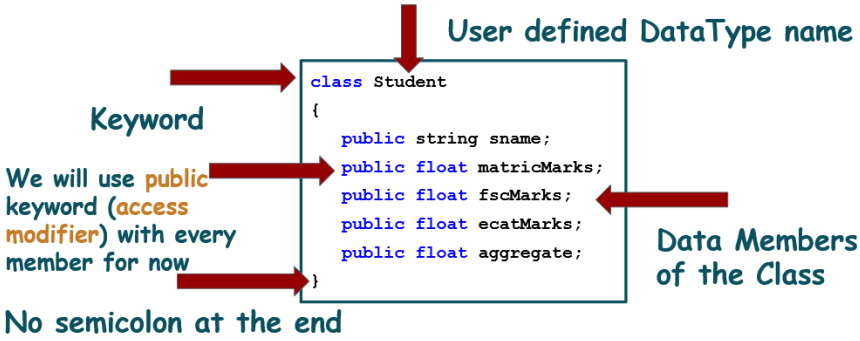



## Class Declaration:

**Syntax:**

```
class Student
{
   public string sname;
   public float matricMarks;
   public float fscMarks;
   public float ecatMarks;
   public float aggregate;
}
```

Following Figure shows a detailed explanation.

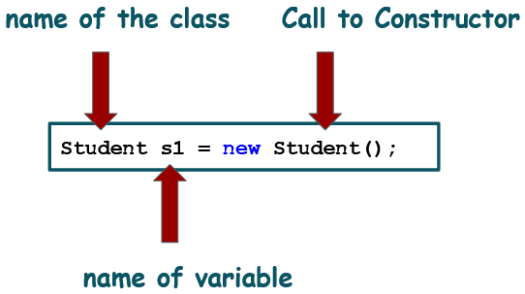



## Variables/Objects of the Class:

The Class acts as a user-defined data type, therefore, we can create its variables.

```
Student s1 = new Student();
```

Following figure shows a detailed explanation.




This variable of the class is also called the instance or object of the class.

## Assigning Values to the Objects of the Class:

```
Student s1 = new Student();
s1.sname = "ABC";
```

**Task:** Write a program that creates a new class of Student.

**Solution:**

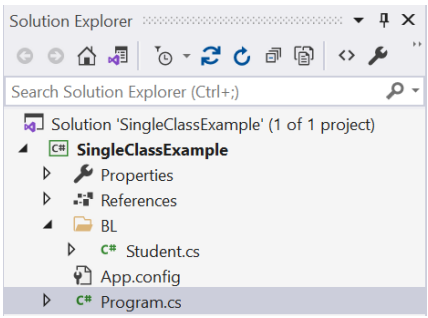
**Student.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SingleClassExample.BL
{
    2 references
    class Student
    {
        public string sname;
        public float matricMarks;
        public float fscMarks;
        public float ecatMarks;
        public float aggregate;
    }
}
```

**Program.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SingleClassExample.BL;

namespace SingleClassExample
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Student s1 = new Student();
            s1.sname = "ABC";
        }
    }
}
```

**Solution Explorer**




The code will generate a new class of Student where each student would have the following properties.

- string type Name
- float type matricMarks
- float type fscMarks
- float type ecatMarks
- float type aggregate

Now, in order to create a "new object" of class Student, we have declared a class type object in the main function.

**Student s1 = new Student();**

This line will "create a **new object** of class **Student**" having the above-defined properties. We have also assigned the sname property of the Student class by writing the following code.

**s1.sname = "ABC";**

We access the variables by using the (**dot .**) operator in front of the class object name. For example, to print the name of the s1 student on the console, we will use the following code.

**Console.WriteLine("Name: {0}", s1.sname);**

## Taking input from the User in Class Object and Storing in the Objects Array:

Taking input in class object variables is the same as taking input in any other variables in C#.
Look at the following code snippet to have a clear understanding of this concept.

**Task:** Write the Function to create a class object and take the Student data from the user and store them in the class object and return this object. Store those objects in the objects array and then create a function that takes the objects array and then prints all the students data.

**Solution**

Write the following code on your computer and execute the program by clicking on the start button.

```
static void Main(string[] args)
{
    Student[] studentsData = new Student[5];
    for (int x = 0; x < 5; x++)
    {
        studentsData[x] = takeStudentInput();
    }
    printStudentsData(studentsData);
    Console.Read();
}
1 reference
static Student takeStudentInput()
{
    Student s = new Student();
    Console.WriteLine("Enter Student Name");
    s.sname = Console.ReadLine();
    Console.WriteLine("Enter Student Matric Marks");
    s.matricMarks = float.Parse(Console.ReadLine());
    Console.WriteLine("Enter Student Fsc Marks");
    s.fscMarks = float.Parse(Console.ReadLine());
    Console.WriteLine("Enter Student Ecat Marks");
    s.ecatMarks = float.Parse(Console.ReadLine());
    return s;
}
1 reference
static void printStudentsData(Student[] studentsData)
{
    Console.WriteLine("Name \t MatricMarks \t FscMarks \t EcatMarks");
    for (int x = 0; x < 5; x++)
    {
        Console.WriteLine(studentsData[x].sname + "\t" + studentsData[x].matricMarks + "\t" + studentsData[x].fscMarks + "\t" + studentsData[x].ecatMarks);
    }
}
```

## Constructor of the Class:

## Default Constructor:

Whenever we create an object with class name and parenthesis, a function is automatically called.

**Task:** Write a program that creates a class of Student and try printing the values of attributes without assigning them any values.

**Solution:**

**Code:**

```
class Student
{
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```

```
Student s1 = new Student();
Console.WriteLine(s1.sname);
Console.WriteLine(s1.matricMarks);
Console.WriteLine(s1.fscMarks);
Console.WriteLine(s1.ecatMarks);
Console.WriteLine(s1.aggregate);
Console.Read();
```

**Output:**

```
0
0
0
0
```

Observing the output, it reflects that the compiler has automatically assigned "**zero**" value to these data members of the class. However, if we want to change these default values, we can implement a **default constructor** in the class definition to override this functionality.

## Creating a Default Constructor

We learned that when we don't explicitly define a constructor in a class, the compiler will provide a default constructor automatically. However, we can also write our own

constructor for the class. We have two options for writing our own constructor: we can either write a default constructor with no parameters, or we can write a parameterized constructor that takes one or more parameters.

In order to create a default constructor, you have to consider the following

- To create a default constructor, we use the **same name as the class**, followed by **parentheses** ()
- The constructor has the **same name as the class**, it is always **public**, and it does not have any **return type**.

**Task:** Write the code to create a default constructor that prints "Default Constructor Called".
**Solution:**

**Code:**

```
class Student
{
   public Student()
    {
      Console.WriteLine("Default Constructor Called");
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```

```
Student s1 = new Student();
Console.Read();
```

**Output:**

```
Default Constructor Called
```

**Hint**: Never forget to write public before the constructor's name.

**Congratulations !!!! You have just learned how to create a default constructor.**

## Setting Attribute values through Default Constructor

Now, we will use the definition of default constructor to assign our desired "default" values for the class objects to be created.

**Task:** Write the code to create a default constructor that assigns a default value to one of the attributes.

**Solution:**

Write the following code into the main function of the code and execute the program by clicking on the start button.

**Code:**

```
class Student
{
   public Student()
    {
      sname = "Jill";
    }
   public string sname;
   public float matricMarks;
   public float fscMarks;
   public float ecatMarks;
   public float aggregate;
}
```

```
Student s1 = new Student();
Console.Write(s1.sname);
Console.Read();
```

**Output:**

```
Jill
```

Now, all the newly created objects of class "**Student**" shall be assigned the value of "**Jill**" for the "**sname**" attribute of the class.

**Self Assessment Task 1(a)**: Create a default constructor that assigns values to all the attributes of the class.

You may use the Student class from the above-mentioned example of the code snippet.

**Self Assessment Task 1(b)**: Create multiple objects of the class and check if all the attributes in multiple objects have been assigned the default values.

## Parameterized Constructor

Moving onwards, there is another type of constructor that is known as "**Parameterized Constructor**". As the name reflects, this constructor receives parameters from the user which means that the user can define the values of the attributes at the time of object creation.

Look at the following code snippet to have a clear understanding of this concept.

**Task:** Write the code to create a class object and take the input name from the user first and then create the object with take name using parameterized constructor.

**Solution**

**Code:**

```
class Student
{
   public Student(string n)
    {
      sname = n;
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```

```
Student s1 = new Student("John");
Console.WriteLine(s1.sname);
Student s2 = new Student("Jack");
Console.WriteLine(s2.sname);
Console.Read();
```

**Output:**

```
John
Jack
```

Observe that this time, the "**Student constructor**" requires a single parameter from the user and assigns that value to the respective attribute of the class.

Similarly, you can create a parameterized constructor that will require a value for all the attributes of that class. In this way, however, the user would need to provide all the values that have been listed in the parameter list of the constructor function.

**NOTE:** You can not create an object without providing the list of parameters once you have defined a parameterized constructor. Attempting to do so would generate an error.

**Self Assessment Task 1(a)**: Create a Parameterized constructor that assigns values to all the attributes of the class.
You may use the Student class from the above-mentioned example of the code snippet.

**Self Assessment Task 1(b)**: Create multiple objects of the class and check if all the attributes in multiple objects have been assigned the unique values.

## Copy Constructor

Right now, we are initializing the attributes of an object using the Constructor (Default or Parameterized). There is another Constructor called **Copy Constructor** that creates a new object (**separate memory on the heap**) by copying variables from another object.

**Note**: The copy constructor requires a complete object as an argument.

**Code:**

```
class Student
{
   public Student()
   {
      Console.WriteLine("Default Constructor");
   }
   public Student(Student s)              //Copy Constructor
   {
      sname = s.sname;
      matricMarks = s.matricMarks;
      fscMarks = s.fscMarks;
      ecatMarks = s.ecatMarks;
      aggregate = s.aggregate;
   }

   public string sname;
   public float matricMarks;
   public float fscMarks;
   public float ecatMarks;
   public float aggregate;
}
```

Now, as you can see it will assign all the values from the received parameter to the class attributes.

Code: **Constructor Call**

```
Student s1 = new Student();
s1.sname = "Jack";
Student s2 = new Student(s1);
Console.WriteLine(s1.sname);
Console.WriteLine(s2.sname);
```

As you can see now, the copy constructor receives an object as parameter and inside the constructor function implementation, it is assigning the values of each attribute of the received object to each corresponding attribute of newly object to be created.

**NOTE:** Every time you create an object using a copy constructor, it creates new memory in HEAP. UNLESS and UNTIL it is important, you are advised to not make extra copies of the class objects using this method.

**Congratulations !!!! You have just learned how to create different types of constructors for creating class objects.**

**You may take a two minutes break, as there is much more code to come.**

### Challenge # 1:
### Student Management System with Class

**Task:** Write a Menu deriven program that shows the following four menu options

1. Add Student.
2. Show Students.
3. Calculate Aggregate
4. Top Students.

- Add Student allows users to add a Student's information that includes Student Name, matric marks, fsc marks, ecat marks.
    - **Hint:** Take input from the user in simple variables and then Use the parameterized constructor to create the object.
- Show Students displays all the added students on the screen.
- Calculate Aggregate will calculate the aggregate of all the available students.
- Top Students lists the information of the top 3 students.

**Note: Remember to make single responsibility functions for each functionality. Remember that you have seen how to pass the object as parameter to a function and how to return an object from a function. You have also seen how to create an array of objects. (Use it well)**

### Challenge # 2:
### Products Management System with Class

**Task**: Write a program that shows three menu options

1. Add Products.
2. Show Products.
3. Total Store Worth.

- Add Product allows the user to add product information that includes ID, Name, price, Category, BrandName, Country.
- Show Product displays all the added products on the screen.
- Total Store Worth calculates the sum of the price of all the products.

**Challenge # 3:**

**Task** Covert the signUp/signIn application that you developed in the previous lab by using the class concepts.

Make a class named MUser with 3 attributes namely

- Username
- Password
- Role

The data should be loaded from the file and loaded into the attributes of the class.

**Good Luck and Best Wishes !!**

**Happy Coding ahead :)**