

Constructors of Class



Review: Class and Object

We have made an object of the **class** with the following code.

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

```
student s1 = new student();
```

Review: Class and Object

We have made an object of the **class** with the following code.

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



```
student s1 = new student();
```

Review: Class and Object

This looks like a **function Call**.

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

```
student s1 = new student();
```



Constructor of the Class

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

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

```
student s1 = new student();
```



Constructor of the Class

We have not made the function, but C# creates one by **default** that reserves the memory in heap and sets **attributes** to the default values.

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

```
student s1 = new student();
```



Constructor of the Class

What will be the **Output** if the following lines are executed?

```
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();
```

Constructor of the Class

Output is:

```
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();
```


```
G:\OOP 2022\Week2\Week2\bin\Debug\Week2.exe
0
0
0
0
```


Constructor of the Class

The Constructor that was called is known as **Default Constructor** because it is called by default.

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

```
student s1 = new student();
```



Constructor of the Class

We can also choose which **default values** we want to give to the attributes.

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

```
student s1 = new student();
```

Constructor of the Class

For that we have to specifically write the **default constructor** in the class.

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

```
student s1 = new student();
```

Constructor of the Class

To create a **default constructor**, we use the same name as the **class**, followed by **parentheses ()**

```
class student
{
    public student()
    {
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```

```
student s1 = new student();
```

Constructor of the Class

The constructor has the **same name** as the class, it is always **public**, and it does not have any **return type**.

```
class student
{
    public student()
    {
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```

```
student s1 = new student();
```

Constructor of the Class

Following code prints on the console "Default Constructor Called" because the constructor `student()` is automatically called.

```
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();
```


Constructor of the Class

Following code prints on the console "Default Constructor Called" because the constructor `student()` is automatically called.

```
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();
```

 G:\OOP 2022\Week2\Week2\bin\Debug\Week2.exe
Default Constructor Called

Constructor of the Class

Now, We can use the **constructor** function to initialize any attributes of the class for the instance that is being created.

```
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();
```


Constructor of the Class

It means whenever the object is created its **sname** will be set to "Jill".

```
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.Read();
```

Constructor of the Class

We can check it by printing the value on the screen. The `Console.Write()` statement will print **Jill** on the screen.

```
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();
```

Constructor of the Class

We can check it by printing the value on the screen. The `Console.Write()` statement will print **Jill** on the screen.

```
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();
```



G:\OOP 2022\Week2\Week2\

Jill

Constructor of the Class

This will initialize **sname** as **Jill** for all objects that we will create.

```
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.WriteLine(s1.sname);
student s2 = new student();
Console.WriteLine(s2.sname);
Console.Read();
```

Constructor of the Class

What if we want to initialize **different name** using the same constructor?

```
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.WriteLine(s1.sname);
student s2 = new student();
Console.WriteLine(s2.sname);
Console.Read();
```

Constructor with Parameters

We can create constructors that can have different **parameters**.

```
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.WriteLine(s1.sname);
student s2 = new student();
Console.WriteLine(s2.sname);
Console.Read();
```

Constructor with Parameters

We can create constructors that can have different **parameters**. For example, the following constructor takes **name** as parameter.

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

Constructor with Parameters

We can create constructors that can have different **parameters**. For example, the following constructor takes **name** as parameter.

```
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();
```


Constructor with Parameters

We can create constructors that can have different **parameters**. For example, the following constructor takes **name** as parameter.

```
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();
```

 G:\OOP 2022\Week2\Week2\


John
Jack

Constructor with Parameters

Now, we need to pass **name** as argument, otherwise we will not be able to create the object of the class.

```
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();
```

 G:\OOP 2022\Week2\Week2\

John
Jack

Constructor with Parameters

Following code contains an **error** because we are trying to create an object with default constructor that does not exist any more.

```
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();
Console.WriteLine(s1.sname);
student s2 = new student();
Console.WriteLine(s2.sname);
Console.Read();
```

Constructor with Parameters

Now we can only create an object of the class by passing a **name** to the constructor.

```
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();
```

Constructor with Parameters

But if we want to call the **default constructor** as well then we have to give the definition of it as well.

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

Constructor with Parameters

But if we want to call the **default constructor** as well then we have to give the definition of it as well.

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

Constructor with Parameters

But if we want to call the **default constructor** as well then we have to give the definition of it as well.

```
class student
{
    public student()
    {
        sname = "Jill";
    }
    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();
Console.WriteLine(s1.sname);
student s2 = new student("Jack");
Console.WriteLine(s2.sname);
Console.Read();
```

Constructor with Parameters

But if we want to call the **default constructor** as well then we have to give the definition of it as well.

```
class student
{
    public student()
    {
        sname = "Jill";
    }
    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();
Console.WriteLine(s1.sname);
student s2 = new student("Jack");
Console.WriteLine(s2.sname);
Console.Read();
```



Constructor with any No. of Parameters

We can extend the constructor so it can take **any number** of parameters.

```
class student
{
    public student()
    {
        sname = "Jill";
    }
    public student(string n, float m)
    {
        sname = n;
        matricMarks = m;
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```

Constructor with any No. of Parameters

This constructor is expecting **two arguments** to be passed when object is created otherwise it will give an **error message**.

```
class student
{
    public student()
    {
        sname = "Jill";
    }
    public student(string n, float m)
    {
        sname = n;
        matricMarks = m;
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```



Constructor with any No. of Parameters

This constructor is expecting **two arguments** to be passed when object is created otherwise it will give an **error message**.

```
class student
{
    public student()
    {
        sname = "Jill";
    }
    public student(string n, float m)
    {
        sname = n;
        matricMarks = m;
    }
    public string sname;
    public float matricMarks;
    public float fscMarks;
    public float ecatMarks;
    public float aggregate;
}
```



```
student s1 = new student("Jack", 3F);
Console.WriteLine(s1.sname);
Console.WriteLine(s1.matricMarks);
Console.Read();
```

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 heap) by copying variables from another object.

Why we need Copy Constructor

Heap

Suppose, we copy **s1** into **s2** like this.

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

```
student s1 = new student();
s1.sname = "Jack";
Console.Read();
```

Stack

s1	A123
----	------

A123

sname	Jack
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Why we need Copy Constructor

Heap

Suppose, we copy **s1** into **s2** like this.

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

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

Stack

s1	A123
s2	A123

A123

sname	Jack
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Why we need Copy Constructor

Heap

Suppose, we copy **s1** into **s2** like this.

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

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

Stack

s1	A123
s2	A123

A123

sname	Jack
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Jack

Why we need Copy Constructor

Heap

Suppose, we copy **s1** into **s2** like this.

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

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

Stack

s1	A123
s2	A123

A123

sname	Jill
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Why we need Copy Constructor

Heap

Suppose, we copy **s1** into **s2** like this.

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

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

Stack

s1	A123
s2	A123

A123

sname	Jill
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Jack
Jill

Why we need Copy Constructor

Heap

It means this is a **Shallow Copy** pointing to the same location. A123

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

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

Stack

s1	A123
s2	A123

sname	Jill
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Jack
Jill

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();

s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A140

Heap

A123

sname	
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A140

Heap

A123

sname	A
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	B
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A123

Heap

A123

sname	A
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	B
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A123

Heap

A123

sname	C
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	B
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A123

Heap

A123

sname	C
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	B
matricMarks	
fscMarks	
ecatMarks	
aggregate	

This will print C on Console

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A123

Heap

A123

sname	C
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	B
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Now, the reference to A140 memory is lost

Working Example

Let's solve a working Example First.

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

```
student s1 = new student();
student s2 = new student();
s1.sname = "A";
s2.sname = "B";
s2 = s1;
s2.sname = "C";
Console.WriteLine(s1.sname);
Console.Read();
```

Stack

s1	A123
s2	A123

Heap

A123

sname	C
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	B
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Garbage Collector is a program that automatically runs and free the memory of the object that cannot be referenced.

Coming Back: Why we need Copy Constructor?

Heap

We want to create two separate objects; but in this way, new object is not created. So what to Do ?

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

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

Stack

s1	A123
s2	A123

A123

sname	Jill
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Jack
Jill

Copy Constructor

In the **Copy Constructor** we have to pass a complete object.

```
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;
}
```

Copy Constructor

In the **Copy Constructor** we have to pass a complete object.

```
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;
}
```

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

Copy Constructor

In the **Copy Constructor** we have to pass a complete object.

```
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;
}
```

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

Default Constructor
Jack
Jack

Copy Constructor

In the **Copy Constructor** we have to pass a complete object.

```
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;
}
```

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

Copy Constructor

In the **Copy Constructor** we have to pass a complete object.

```
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;
}
```

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

Default Constructor

Jack
Jack
Jill
Jack

Copy Constructor

Important thing to note is that copy constructor makes a new **object in the memory**, therefore separate memory is allocated to the new object.

This is what we call a **Deep Copy**.

Do not make the copies of the object unless it is highly needed.

Stack

s1	A123
s2	A140

Heap

A123

sname	
matricMarks	
fscMarks	
ecatMarks	
aggregate	

A140

sname	
matricMarks	
fscMarks	
ecatMarks	
aggregate	

Conclusion

- Whenever we create an object of a class a **constructor function** is called automatically.
- Constructor function **without any parameters** is called **default constructor**.
- 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**.
- We can also pass **multiple parameters** to a constructor when an object is being created.
- **Copy Constructor** copies the data of one object into another object



Learning Objective

Explain Role of the Constructors and Memory Representation of the objects.



Self Assessment: Class and Objects

1. Make following Class in C# for Circle.

- Two data members:
 1. radius (of the type double)
 2. color (of the type String).
- Make Three constructors:
 1. One with default constructor with default value of 1.0 and "red", respectively.
 2. One Constructor that takes only the radius as parameter
 3. And Another constructor which takes radius and color.
- Make a Copy Constructor to make a deep copy

