ZETECH UNIVERSITY

UNIT NAME: PROGRAMMING WITH.NET Using C#.

UNIT CODE: BIT 321

Lesson 6: C# Classes and objects

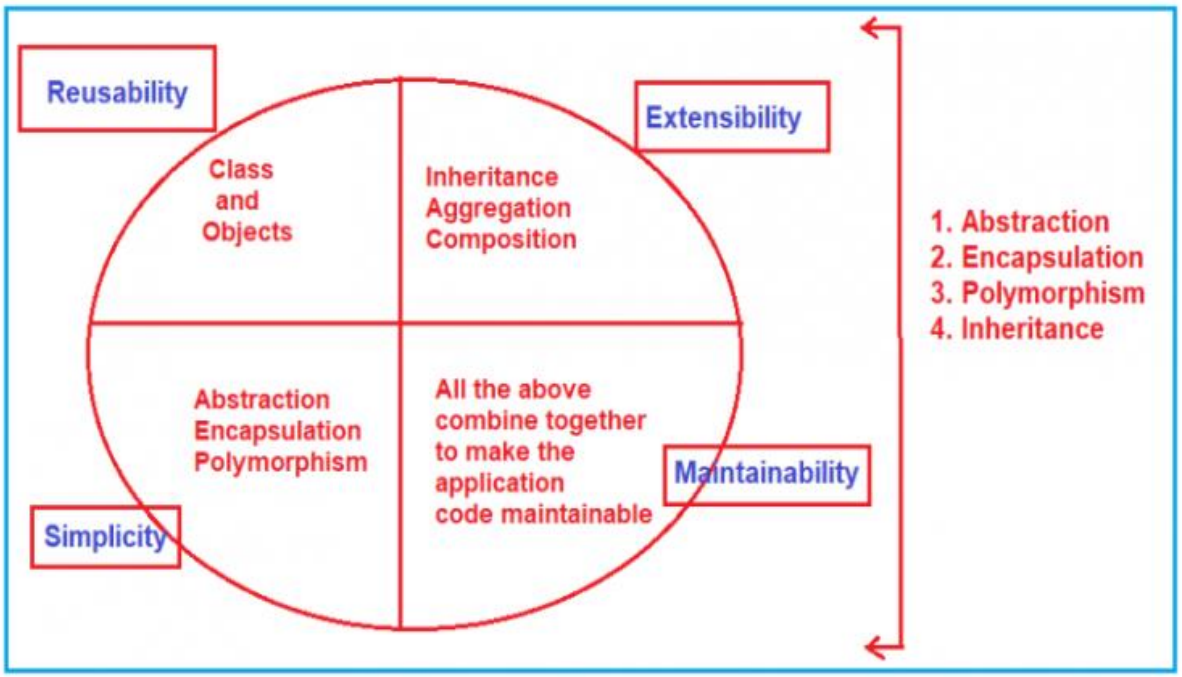
Objectives

By the end of this chapter you will be able to….

- Explain what Object Oriented Programming is,
- Describe the benefits of the Object Oriented programming approach and
- Understand the basic concepts of abstraction, encapsulation, generalization and polymorphism on which object oriented programming relies

Object-Oriented Programming (OOPs) | OOPs Concept in C#

- Object-Oriented Programming commonly known as OOPs is a technique, **not a technology.** It means it doesn't provide any syntaxes or APIs instead it provides suggestions to design and develop objects in programming languages.



Classes and Objects in C#

- A class is a data structure in C# that combines data variables and functions into a single unit. Instances of the class are known as objects.
- While a class is just a blueprint, the object is an actual instantiation of the class and contains data.

Declaring a Class in C#

In c#, classes are declared by using class keyword. Following is the declaration of class in c# programming language.

```
AccessSpecifier class NameOfClass

{

    // Member variables

    // Member functions

}
```

Class	Object
A class is a blueprint from which you can create the instance, i.e., objects.	An object is the instance of the class, which helps programmers to use variables and methods from inside the class.
A class is used to bind data as well as methods together as a single unit.	Object acts like a variable of the class.
Classes have logical existence.	Objects have a physical existence.
A class doesn't take any memory spaces when a programmer creates one.	An object takes memory when a programmer creates one.
The class has to be declared only once.	Objects can be declared several times depending on the requirement.

Objects

Object is an actual instantiation of the class and contains data.

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 the 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.

Declaring an Object in C

Example 1

Objects in C# are declared using the new keyword, which creates a new memory location for storing the object.

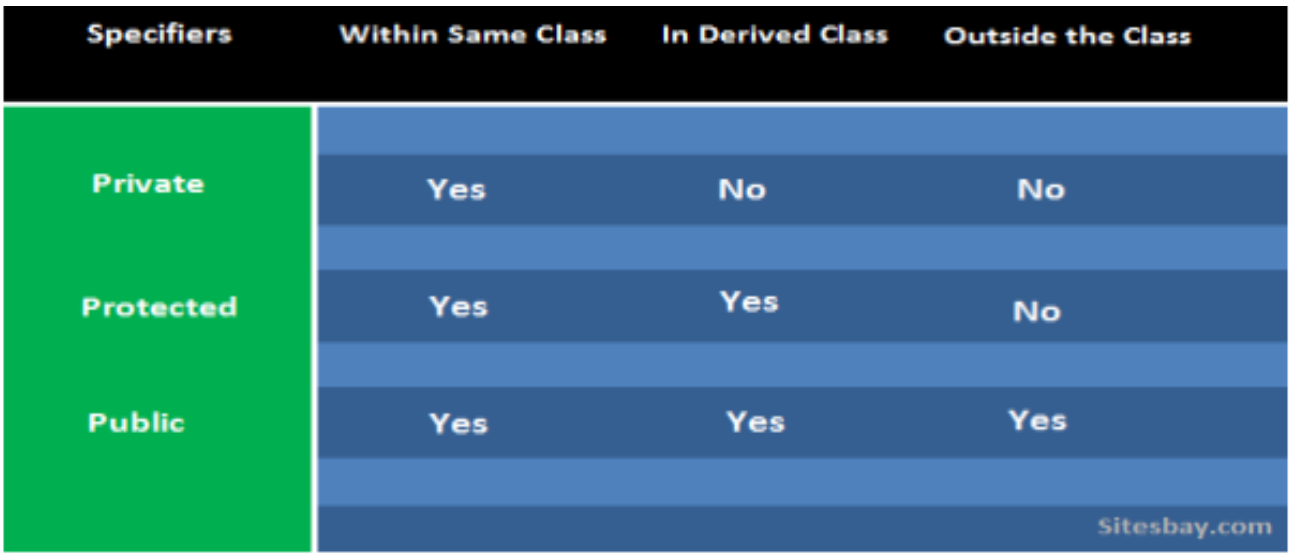
Syntax:

```
<class-name> <object-name> = new <class-name> (parameter—list);
```

Access specifiers in C

Access specifiers in C # defines how the members of the class can be accessed. C# has 3 new keywords introduced, namely. The access modifiers of C# allow us to determine which class members are accessible to other classes and functions, and which are not.

- Public -elements can be accessed by all other classes and functions.
- Private -elements cannot be accessed outside the class in which they are declared,
- Protected - elements are just like the private, except they can be accessed by derived/child classes.

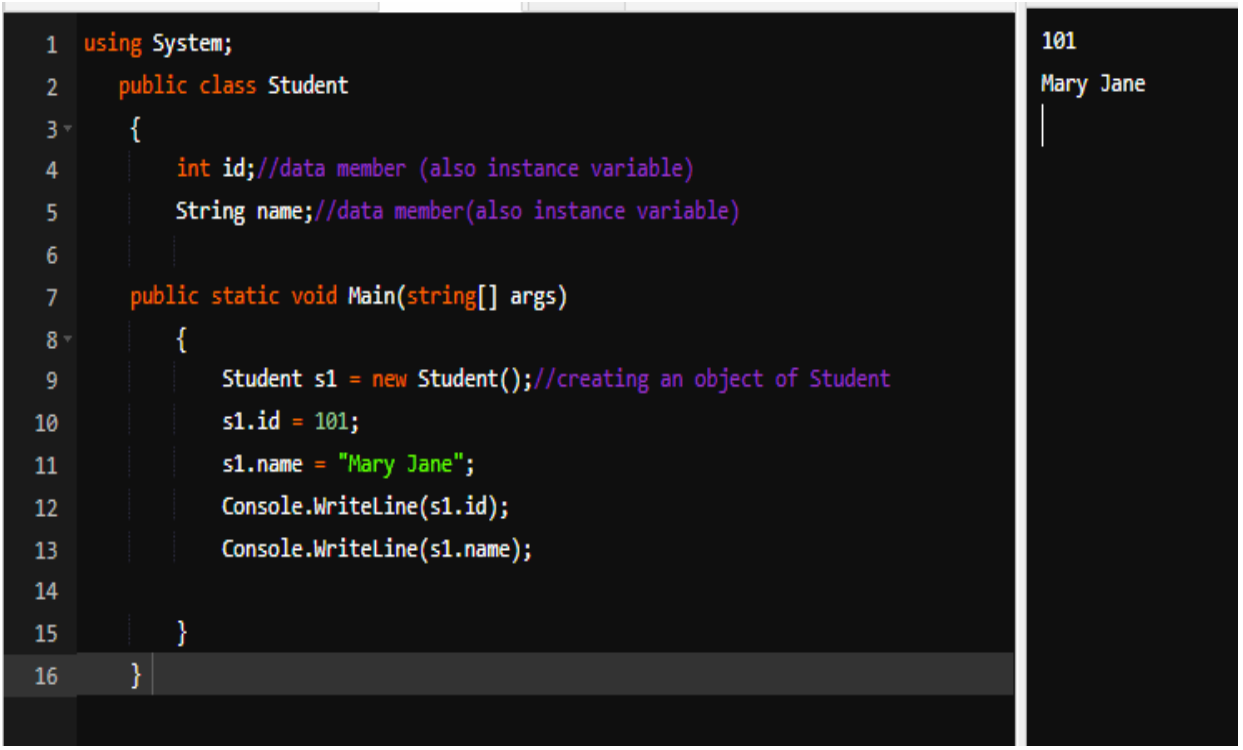


Example of a C# Class.

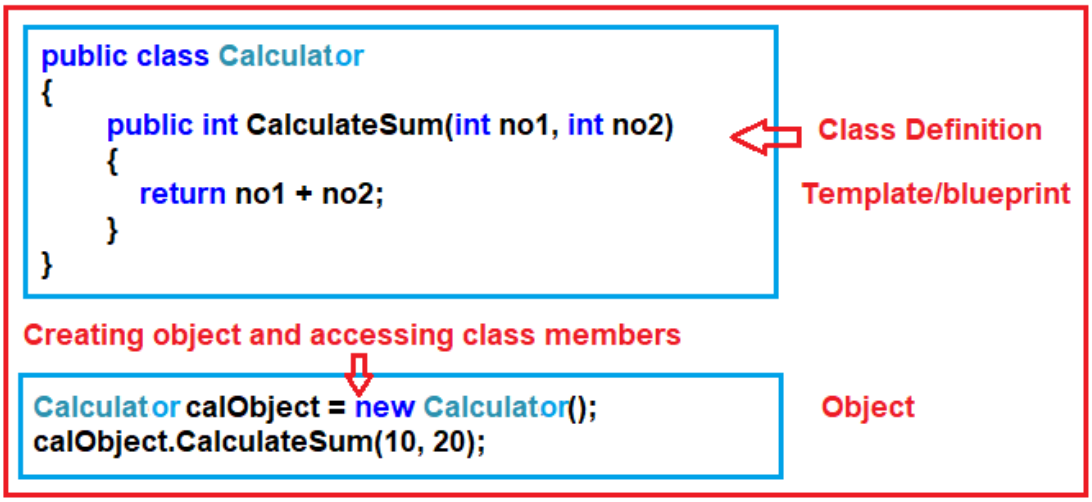
```
using System;
  public class Student
  {
     int id;//data member (also instance variable)
     String name;//data member(also instance variable)

   public static void Main(string[] args)
     {
        Student s1 = new Student();//creating an object of Student
        s1.id = 101;
        s1.name = "Mary Jane";
        Console.WriteLine(s1.id);
        Console.WriteLine(s1.name);

     }
  }
```



Example 2



The complete example code is given below.

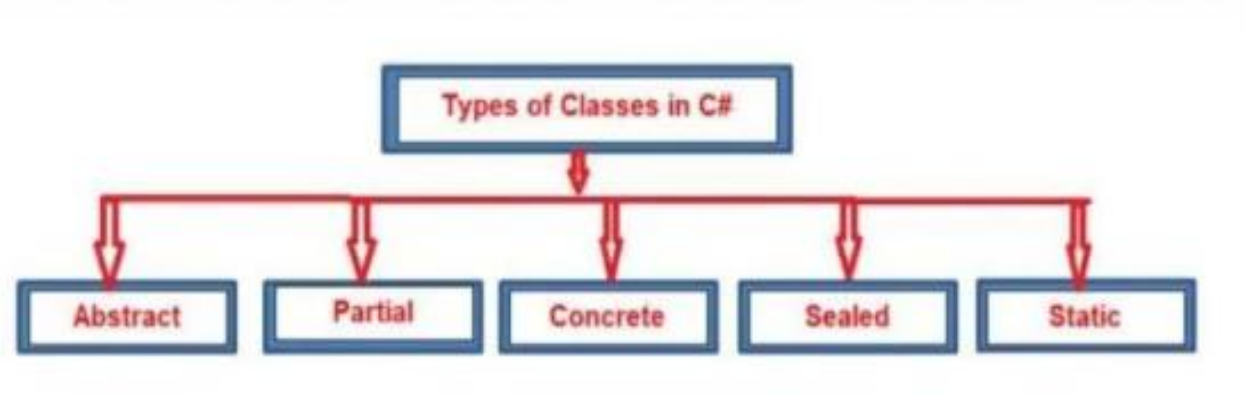
```
using System;
namespace ClassObjectsDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            //Creating object
            Calculator calObject = new Calculator();

            //Accessing Calculator class member using Calculator class object
            int result = calObject.CalculateSum(10, 20);

            Console.WriteLine(result);
            Console.ReadKey();
        }
    }

    //Defining class or blueprint or template
    public class Calculator
    {
        public int CalculateSum(int no1, int no2)
        {
            return no1 + no2;
        }
    }
}
```

Types of Classes in C#:



C# Abstract Class

- An abstract class is an incomplete class or special class we can't be instantiated.
    - The **purpose** of an abstract class is *to provide a blueprint for derived classes and set some rules what the derived classes must implement when they inherit an abstract class.*

- **We can use an abstract class as a base class and all derived classes must implement abstract definitions.** An abstract method must be implemented in all non-abstract classes using the override keyword. After overriding the abstract method is in the non-Abstract class. We can derive this class in another class and again we can override the same abstract method with it.

```
public abstract class AbsParent
{
    public void Add(int x, int y)
    {
        Console.WriteLine($"Addition of {x} and {y} is : {x + y}");
    }
    public void Sub(int x, int y)
    {
        Console.WriteLine($"Subtraction of {x} and {y} is : {x - y}");
    }
    public abstract void Mul(int x, int y);
    public abstract void Div(int x, int y);
}
```

What Are Static Classes?

Static classes are **classes that cannot be instantiated by their users.** All the members of static classes are static themselves. They can be accessed through the class name directly without having to instantiate them using the new keyword.

Let's take a look at a C# syntax when defining static classes:

```
static class ClassName
{
    //static methods
    //static data members
}
```

When Should We Use Static Classes?

Here are some of the scenarios where we might want to use static classes.

1. They are **perfect for implementing utilities.** Applications that don't need any modifications or utility classes could use static classes. *For example, in-built classes such as Math and System. Convert are some of the commonly used static classes.*
2. **Static classes consume fewer resources.** They do not require instantiation, hence, duplicate objects don't take up additional memory space. Theoretically, static classes offer better performance than their non-static counterparts. This is usually unnoticeable in practical applications.

A static class is like a normal class but:

1. We can't create any objects of it.
2. We can't even create subclasses of it (it is sealed).
3. It must contain only static members.
4. It can't contain instance constructors.

We use `static` keyword to define a static class. Let's take an example.

```
using System;

static class Circle
{
public static double pi = 3.14;

public static void PrintArea(double radius)
{
   Console.WriteLine($"Area is {pi*radius*radius}");
}

public static void PrintCircumference(double radius)
{
   Console.WriteLine($"Circumference is {2*pi*radius}");
}
}

class Test
{
static void Main(string[] args)
{
   Circle.PrintArea(2);
   Circle.PrintCircumference(2);
}
}
```

Output

```
Area is 12.56
Circumference is 12.56
```

C# Constructor

In C#, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C# has the same name as class or struct.

There can be two types of constructors in C#.

- Default constructor
- Parameterized constructor

Rules of a Constructor

- A constructor name must be the same as a class name.
- A constructor can be public, private, or protected.
- The constructor cannot return any value so cannot have a return type.
- A class can have multiple constructors with different parameters but can only have one parameter less constructor.
- If no constructor is defined, the C# compiler would create it internally.

Example: Constructor

```
class Student
{
    public Student ()
    {
        //constructor
           Student ()
    }
}
```

C# Parameterized Constructor

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

using System;

public class Employee

{

```
    public int id;
    public String name;
    public float salary;
    public Employee(int i, String n,float s)
    {
       id = i;
       name = n;
       salary = s;
    }
    public void display()
    {
       Console.WriteLine(id + " " + name+" "+salary);
    }
}
class TestEmployee{
   public static void Main(string[] args)
    {
       Employee e1 = new Employee(101, "Nzombo", 890000f);
       Employee e2 = new Employee(102, "Kabaka", 490000f);
       e1.display();
       e2.display();
```

```
    }
}
```

Output

101 Nzombo 890000

102 Kabaka 490000

- ✓ Class Revision Implement the above code using User input function (Read line () method)

Revision Questions

1. Consider a student class with feet and inches as attributes which describes the height of the student. Write a C# program to overload the + operator and to find the average of N students.
2. Write a C# program that calculates the Area of a rectangle using class and object concept.