

Inheritance

```
namespace Inheritance
{
    public class Employee
    {
        public string FirstName;
        public string LastName;
        public string Email;

        public Employee()
        {
            Console.WriteLine("Parent Class is called...");
        }
        public Employee(string firstName, string lastName, string email)
        {
            FirstName = firstName;
            LastName = lastName;
            Email = email;
        }
        public void ShowFullName()
        {
            Console.WriteLine(FirstName + " " + LastName);
        }
    }
    public class FullTimeEmployee : Employee
    {
        public float YearlySalary;
        public FullTimeEmployee()
        {
            Console.WriteLine("Child class FTE is called...");
        }
    }

    public class PartTimeEmployee : Employee
    {
        public float HourlySalary;
        public PartTimeEmployee() :
base("Rohit", "Tekchandani", "tekchandanirohit859@gmail.com") // To call other
contractor of base class
        {
            Console.WriteLine("Child class PTE is called...");
        }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            FullTimeEmployee FTE = new FullTimeEmployee();
            PartTimeEmployee PTE = new PartTimeEmployee();
            FTE.FirstName = "Mohit";
            FTE.LastName = "Panjabi";
            FTE.Email = "panjabimohit@gmail.com";
            FTE.YearlySalary = 56994;
            PTE.HourlySalary = 66;
            FTE.ShowFullName();
            PTE.ShowFullName();
            Console.ReadLine();
        }
    }
}
```

```
3544] Inheritance.exe
C:\Users\Admin\Desktop\CS_syntax\Inh
Parent Class is called...
Child class FTE is called...
Child class PTE is called...
Mohit Panjabi
Rohit Tekchandani
```

Why inheritance

```
public class FullTimeEmployee
{
    string FirstName;
    string LastName;
    string Email;
    float YearlySalary;

    public void PrintFullName()
    {
    }
}
```

```
public class PartTimeEmployee
{
    string FirstName;
    string LastName;
    string Email;
    float HourlyRate;

    public void PrintFullName()
    {
    }
}
```

A lot of code between these 2 classes is duplicated

Using inheritance

```
public class Employee
{
    string FirstName;
    string LastName;
    string Email;

    public void PrintFullName()
    {
    }
}
```

Move all the common
code into base
Employee class

```
public class FullTimeEmployee
{
    float YearlySalary;
}
```

```
public class PartTimeEmployee
{
    float HourlyRate;
}
```

FullTime and PartTime
employee specific code in
the respective derived
classes

Why Inheritance

Pillars of Object Oriented Programming

1. Inheritance
2. Encapsulation
3. Abstraction
4. Polymorphism

1. Inheritance is one of the primary pillars of object oriented programming.
2. It allows code reuse.
3. Code reuse can reduce time and errors.

Note: You will specify all the common fields, properties, methods in the base class, which allows reusability. In the derived class you will only have fields, properties and methods, that are specific to them.

Inheritance Syntax

```
public class ParentClass
{
    // Parent Class Implementation
}

public class DerivedClass : ParentClass
{
    // ChildClass Implementation
}
```

1. In this example DerivedClass inherits from ParentClass.
2. C# supports only single class inheritance.
3. C# supports multiple interface inheritance.
4. Child class is a specialization of base class.
5. Base classes are automatically instantiated before derived classes.
6. Parent Class constructor executes before Child Class constructor.

Interface

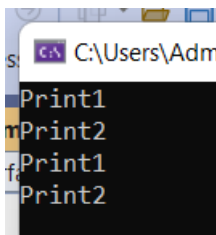
```
namespace Interface
{
    interface Icustomer1
    {
        void print1();
    }

    interface Icustomer2
    {
        void print2();
    }

    public class Customer : Icustomer1,Icustomer2
    {
        public void print1()
        {
            Console.WriteLine("Print1");
        }
        public void print2()
        {
            Console.WriteLine("Print2");
        }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            Customer C1 = new Customer();
            Icustomer1 C2 = new Customer();
            Icustomer2 C3 = new Customer();

            C1.print1();
            C1.print2();
            C2.print1();
            C3.print2();

            Console.ReadLine();
        }
    }
}
```



Interfaces

We create interfaces using **interface** keyword. Just like classes interfaces also contains properties, methods, delegates or events, but only declarations and no implementations.

It is a compile time error to provide implementations for any interface member.

Interface members are public by default, and they don't allow explicit access modifiers.

Interfaces cannot contain fields.

If a class or a struct inherits from an interface, it must provide implementation for all interface members. Otherwise, we get a compiler error.

A class or a struct can inherit from more than one interface at the same time, but where as, a class cannot inherit from more than once class at the same time.

Interfaces can inherit from other interfaces. A class that inherits this interface must provide implementation for all interface members in the entire interface inheritance chain.

We cannot create an instance of an interface, but an interface reference variable can point to a derived class object.

Introduction

Sample Programme:

```
// Namespace Declaration
using System;

class Program
{
    public static void Main()
    {
        // Write to console
        Console.WriteLine("Welcome to PRAGIM Technologies!");
    }
}
```

Using System declaration

The namespace declaration, **using System**, indicates that you are using the **System** namespace.

A namespace is used to organize your code and is collection of classes, interfaces, structs, enums and delegates.

Main method is the entry point into your application.