C sharp Book for Beginners:-

1. Sample Program

//Namespace Declaration

using System;

class Program;

{

public static void Main()

{

//write to console

Console.WriteLine("Welcome to Pragim Technologies!");

}

}

Using namespace 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 applications

Eg:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_1\_SampleProgram

{

class Program

{

static void Main1()

{

Console.WriteLine("Welcome to C# classes training part2!");

Console.ReadLine();

}

static void Main()

{

Console.WriteLine("Welcome to C# Training Classed");

Main1();

Console.ReadLine();

}

}

}

2. Reading and writing to console

using system;

calss Program

{

static void Main()

{

//Prompt the user for his name

console.WriteLine("Please enter your name");

//Read the name from console

string UserName=Console.Readline();

//concatenate name with hello world and print

Console.WriteLine("Hello"+UserName);

//peaceholder syntax to print name with hello world

//conole.WriteLine("Hello{0}",UserName);

}

}

Note:-C sharp is case sensitive

Eg:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_2\_WriteLineandReadline

{

class Program

{

static void Main()

{

Console.WriteLine("Please enter your FirstName");

string FirstName = Console.ReadLine();

Console.WriteLine("Please enter your LastName");

string LastName = Console.ReadLine();

Console.WriteLine("Hello {0},{1}", FirstName,LastName);

//Console.WriteLine("Hello" + UserName);

Console.ReadLine();

}

}

}

3. Built-in type in C#

#Bolean type==>Only true or false

#Integral Type==> sbyte,byte,short,unshort,int,uint,long,ulong,char

#Floating Type==> float and double

#Decimal Type

#String Type

For Eg:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_3Built\_in\_Types

{

class Program

{

static void Main()

{

bool b = true;

int i = 0;

Console.WriteLine("Min={0}", int.MinValue);

Console.WriteLine("Max={0}", int.MaxValue);

double d= 123.222334455;

Console.WriteLine(d);

Console.ReadLine();

}

}

}

4. String

string Name ="\"Ganesh\"";

string Name = "One\nTwo\nThree";

Q)What is Verbatim Literal?

Ans==>verbatim literal,is a string with an @ symbol perfix, as it @"Helo"

verbatim literals make escape sequences translate as normal printable

characters to enhance readbility

for eg:- without verbatim literal:-"c:\\Ganeh\\DotNet\\Training\\Csharp"

With verbatim literal:-@"c:\\Ganeh\\DotNet\\Training\\Csharp"

Eg:-

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_4\_StringDataType

{

class Program

{

static void Main()

{

string Name = @"c:\\Ganeh\\DotNet\\Training\\Csharp";

Console.WriteLine(Name);

Console.ReadLine();

}

}

}

5. Common operator in c

#Assignement Operator like =

# Arithmatic Operators like +,-\*,/,%

# Comparison Operators like ==,!=,>,>=,<,<=

# Conditional Operator like &&, ||

# Ternary Operator like ?:

# Null Coalescing Operator ??

Eg:-

sing System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_5\_CommonOPerator\_In\_C\_Sharp

{

class Program

{

static void Main(string[] args)

{

//Assignment Operator

int i = 10;

bool b = true;

//Aerithmatic Operator

int numerator = 10;

int Denominator = 2;

int result = numerator / Denominator;

Console.WriteLine("Result={0}", result);

//Comparison Operator

int Number = 10;

if (Number==10)

{

}

if (Number!=10)

{

}

//Conditional Operator

int N1 = 10;

int N2 = 20;

// if (N1==10 && N2==20)//both the condition should be satisfy

if (N1 == 10 || N2 == 30) //one condition should be satisfy

{

Console.WriteLine("Hello");

Console.ReadLine();

}

//Ternary Operator

/\*int num = 15;

bool IsNumber10;

if (num==10)

{

IsNumber10 = true;

}

else

{

IsNumber10 = false;

}

Console.WriteLine("Number==10 is {0}", IsNumber10);

Console.ReadLine();

\*/

int num = 15;

bool isNumber10 = Number == 10 ? true : false;

Console.WriteLine("Number==10 is {0}", isNumber10);

Console.ReadLine();

}

}

}

6.Nullable Type in C sharp

In C sharp typsses are divided into 2 broad categories

#Value types==>int, float, double,structs,enum etc

#References Types==> interface,class,delegates,arrays etc

#By default values types are non vullable to make them nullable use?

int i=0(i is non nullable, so i can not be set to null, i=null will generate compile error)

int?j=(j is nullable int,so j=null is legal)

Nullable types bridge the differences between c # types and Database types

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_6\_Nullable\_Types\_of\_C\_sharp

{

class Program

{

static void Main(string[] args)

{

/\*

bool? areyouMajor = null;

if (areyouMajor==true)

{

Console.WriteLine("User is Major");

Console.ReadLine();

}

else if(areyouMajor==false)

{

Console.WriteLine("User is not Major");

Console.ReadLine();

}

else

{

Console.WriteLine("User did not answer the question");

Console.ReadLine();

}\*/

int? ticketonsales = 5;

int availabletickets=ticketonsales??0;//null colleciong operator

/\* if (ticketonsales==null)

{

availabletickets = 0;

}

else

{

availabletickets =(int) ticketonsales;

}

\* \*/

Console.WriteLine("AvailableTickets={0}", availabletickets);

Console.ReadLine();

}

}

}

7. Data Type Conversion in C Sharp:-

Data type conversion in C sharp. There are two types of conversion i.e #implicit and Explicit

#Implicit conversion:-

Implicit conversion done by the compiler

1.When there is no loss of data in conversion

2. If there is no possibility of throwing the exception during the conversion

Example:-Converting an int to float will not lose any data and no exception will be thrown hence an implicit conversion can be done

Whereas when converting a float to an int , we lose the fractional part

so conversion is required. For explicit conversion we can use cast operator or the convert class in c sharp

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_7\_DataTypeConversion

{

class Program

{

static void Main(string[] args)

{

//not possible

/\*float f = 100.25;

int i = f;

Console.WriteLine(i);

Console.ReadLine();

\*/

//possible to do

/\* int i = 100;

float f = i;

Console.WriteLine(f);

Console.ReadLine();

\*/

/\* float f = 123.60F;

int i = Convert.ToInt32(f);

Console.WriteLine(i);

Console.ReadLine();

\*/

string strNumber = "100TG";

int Result=0;

bool IsConversionIsSucesseful= int.TryParse(strNumber,out Result);

// int i = int.Parse(strNumber);

if (IsConversionIsSucesseful)

{

Console.WriteLine(Result);

Console.ReadLine();

}

else

{

Console.WriteLine("Enter valid NUmber");

Console.ReadLine();

}

}

}

}

8

8.Array

An Array is a collection of similar data types

Examples:

int[] EvenNumbers= new int[3];

EvenNumbers[0]=0;

EvenNumbers[1]=2;

EvenNumbers[2]=4;

//Initilize and Assign Values in the same line

int[] OddNumbers={1,2,3};

Advantages:-Arrays are strongly typed.

Disadvantages=Arrays cannot grow in size once initialized.Have to rely on integral indices to store or retrive items from the array.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8\_Array

{

class Program

{

static void Main(string[] args)

{

int[] EvenNumbers = new int[3];

EvenNumbers[0] = 0;

EvenNumbers[1] = 2;

EvenNumbers[2] = 4;

Console.WriteLine(EvenNumbers[1]);

Console.ReadKey();

}

}

}

9. Comments in C sharp

#single lIne comments -//

#multiline Comments (-/\* \* /)

#xml Documentation Comments -///

Comments are used to do what the program does and what specific blocks or lines of code do. C# compiler ignores comments

To comment and Uncomment, there are two ways

\* 1. Use Designer

\* 2. Keyboard Shutcut: ctrl+K,ctrl+c and CTRL+K,Ctrl+U

Note:-Do not try to comment every line of code .Use comment only for blocks of lines of code that is difficult to understand

10. If statement in C sharp

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_10\_IFStatement

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("please enter a number");

int UserNumber=int.Parse( Console.ReadLine());

/\*

if (UserNumber == 1)

{

Console.WriteLine("your number is one");

Console.ReadLine();

}

else if (UserNumber==2)

{

Console.WriteLine("your number is 2");

Console.ReadLine();

}

else

{

Console.WriteLine("your number is not between 1 and 3");

Console.ReadLine();

}

\* \*/

if (UserNumber==10|| UserNumber ==20)

{

Console.WriteLine("your number is 10 or 20");

}

}

}

}

11. Switch Statement:-

Multiple if else statements can be replaced with a switch statement

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_11\_Switch\_Statement

{

class Program

{

static void Main(string[] args)

{

//Console.WriteLine("Please enter the number");

// int UserNumber = int.Parse(Console.ReadLine());

/\*if (UserNumber==10)

{

Console.WriteLine("your number is 10");

Console.ReadLine();

}

else if (UserNumber==20)

{

Console.WriteLine("your number is 20");

Console.ReadLine();

}

else if (UserNumber==30)

{

Console.WriteLine("your number is 30");

Console.ReadLine();

}

else

{

Console.WriteLine("your number is not 10,20, &30");

}

\*/

/\* switch (UserNumber)

{

case 10:

Console.WriteLine("your number is 10");

Console.ReadLine();

break;

case 20:

Console.WriteLine("your number is 20");

Console.ReadLine();

break;

case 30:

Console.WriteLine("your number is 30");

Console.ReadLine();

break;

default:

Console.WriteLine("Your number is not 10,20 and 30");

Console.ReadLine();

break;

}\*/

int TotalCOffeeCost = 0;

Start:

Console.WriteLine("Please select your coffe:1-Small, 2-Medium, 3-Large");

int userChoice = int.Parse(Console.ReadLine());

switch (userChoice)

{

case 1:

TotalCOffeeCost += 1;

break;

case 2:

TotalCOffeeCost += 2;

break;

case 3:

TotalCOffeeCost += 3;

break;

default:

Console.WriteLine("your choice {0} is invalid", userChoice);

goto Start;

}

Decide:

Console.WriteLine("Do you want to buy anoterh coffe-yes or No?");

string UserDecision = Console.ReadLine();

switch (UserDecision.ToUpper())

{

case "YES":

goto Start;

case "NO":

break;

default:

Console.WriteLine("Your choice{0} is invalid .Please try again...",UserDecision);

goto Decide;

}

Console.WriteLine("Thank you for shoping with us");

Console.WriteLine("Bill amout={0}", TotalCOffeeCost);

Console.ReadKey();

}

}

}

13. While Loop

\* While loop check the conditional first

\* If the condition is true, statement with in the lop are executed

\* This process is repeated as long as the condition evaluates

NOTE:-Don’t Forget to update the variable participating int he condition, so the loop can end ,same point

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_13\_WhileLoop

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Plese enter your target?");

int usertarget = int.Parse(Console.ReadLine());

int start = 0;

while (start<=usertarget)

{

Console.WriteLine(start);

start = start + 2;

}

Console.ReadLine();

}

}

}

14. Do While Loop

A do while loop checks its condition at the end of the loop. This means that the do loop is guaranteed to execute at least one minute. Do loops are used to present a menu to the user

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_14\_Do\_Whileloop

{

class Program

{

static void Main(string[] args)

{

string UserChoice = "";

do

{

Console.WriteLine("please enter your target?");

int UserTarget = int.Parse(Console.ReadLine());

int start = 0;

while (start < UserTarget)

{

Console.WriteLine(start + "");

start = start + 2;

}

Console.ReadLine();

do

{

Console.WriteLine("do you want to continue-Yes or NO");

UserChoice = Console.ReadLine().ToUpper();

if (UserChoice != "Yes" && UserChoice != "NO")

{

Console.WriteLine("Invalid choice Please select yes or NO");

Console.ReadLine();

}

} while (UserChoice != "YES" && UserChoice != "NO");

} while (UserChoice == "YES");

}

}

}

15. for Loop and For Each Loop

For Loop:-A for loop is very similar to while loop we do the initialization at the one place, condition check at another place and modifying the variable at another place, whereas for loop has all of these at one place.

Foreach Loop:-A foreach loop is used to iterate through the items in a collection. For Example, foreach loop can be used with arrays or collections such as Array List, HashTable and Gererics. We will cover collections and generics in a later session.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_15\_For\_and\_ForEach

{

class Program

{

static void Main(string[] args)

{

/\*

int[] Number = new int[3];

Number[0] = 101;

Number[1] = 102;

Number[2] = 103;

int i = 0;

for (int j = 0; j <3; j++)

{

Console.WriteLine(Number[j]);

}

foreach (int k in Number)

{

Console.WriteLine(k);

}

Console.ReadLine();

while (i < Number.Length)

{

Console.WriteLine(Number[i]);

i++;

}

Console.ReadKey();

\*/

/\* for (int i = 1; i <= 20; i++)

{

Console.WriteLine(i);

if (i==10)

{

break;

}

}

Console.ReadLine();

\*/

for (int i = 0; i <= 20; i++)

{

if (i%2==1)

{

continue;

}

Console.WriteLine(i);

}

Console.ReadLine();

}

}

}

16. Methods

Q)WHY METHODS?

Methods are also called as functions. These terms are used interchangeably. Methods are steamily useful because they allow you to define your logic once, and use it, at many places. Methods make the maintenance of your applications easier

Methods

\*[attributes]

\* access-modifiers return-type method-name(parameters)

\* {

\* Method Body

\* }

1. We will talk about attributes and access modifiers in later Session

2. Return type is any valid data types of void

3. Method name can be any meaningful Name

4. Parameters are optional

STATIC VS INSTANCE METHOD

#when the method declaration includes static modifiers, that method is said to be a static method

# When the static modifier is present, the method is said to be as instance method. #Static method is invoked using the class name, where as an instance methods is that multiple instance of a class can be created(or instaiated) and each instance has its own separate method. However, when a method is static, there are no instances of that

Method and you can invoke only that one definition of the static method.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_16Methods

{

class Program

{

static void Main(string[] args)

{

Program p = new Program();

p.EvenNumber(30);

int sum= p.Add(10, 20);

Console.WriteLine("Sum={0}", sum);

Console.ReadLine();

}

public int Add(int FirstNumber, int secondNumber)

{

return FirstNumber + secondNumber;

}

public void EvenNumber(int Target)

{

int start = 0;

while (start<=Target)

{

Console.WriteLine(start);

start = start + 2;

}

Console.ReadLine();

}

}

}

17. METHOD PARAMETER

There are 4 different types of parameters a method can have

1. Value Parameters:- Create a copy of parameters passed, so modifications does not affect each other’s

2. Reference Parameters:- The reference method parameter keyword on a method parameter causes a method to refer to the same variable that

was passed into the method. Any changes made to the parameter in the method will be reflected on that variable when control passed back

to the calling method.

3. Out parameter==> Use when you want a method to return more than one value.

4. Parameter Arrays:-The parms Keywords letes you specify a method parameter that takes a variable number of arguments. you can send comma-separated list of arguments, or an array, or no arguments.

b. parameter keyword should be the last one in a method declaration, and only one param keyword is permitted in a method declaration

NOTE:-Method Parameter VS Method Arguments

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_17\_MethodParameters

{

class Program

{

static void Main(string[] args)

{

// int i=0;

// SimpleMethod( ref i);

// Console.WriteLine(i);

// Console.ReadKey();

//Pass by vaue that ie(SimpleMethod(int i))

/\*I and J are pointing to different memory locatons. Operations one variable will no affect the value of the other variable\*/

//Pass by reference i.e.(SimpleMethod(int ref i))

/\*I and j are pointing to the same memory location. Operatons one variable will affect the value of the other variable.\*/

// }

// public static void SimpleMethod(ref int J)

//{

// J = 101;

//}

/\*

int Total = 0;

int Product = 0;

Calculate(10, 20, out Total, out Product);

Console.WriteLine("Sum={0} && Product={1}", Total, Product);

Console.ReadKey();

\*/

int[] Numbers = new int[3];

Numbers[0] = 101;

Numbers[1] = 102;

Numbers[2] = 103;

// ParamsMethod();

// ParamsMethod(Numbers);

ParamsMethod(1, 2, 3, 4);

}

/\* public static void Calculate(int FN, int SN, out int Sum, out int Product)

{

Sum = FN + SN;

Product = FN \* SN;

}

\*/

public static void ParamsMethod(params int[] Numbers)

{

Console.WriteLine("There are {0} elements", Numbers.Length);

foreach (int i in Numbers)

{

Console.WriteLine(i);

}

Console.ReadKey();

}

}

}

18. NAMESPACES

Q. WHY Namespaces?

Namespaces are used to organize your programs. They also provide assistance in avoiding name clashes

NAMESPACES

Namespaces don't correspond to file, directory or assembly names. They could be written in separate files and /or separate assemblies and still belong to the same namespaces.

Namespaces can be nested in 2 ways.

# Namespaces alias directives. Sometimes you may encounter a long namespace and wish to have it shorter this could be

#this could improve readability and still avoid name clashes with similarity named methods.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ProjectA.TeamA;

using System.Collections;

using System.Configuration;

//using PATA = ProjectA.TeamA;

//using PATB= ProjectA.TeamB;

namespace \_18\_Namespaces\_in\_CSharp

{

class Program

{

static void Main(string[] args)

{

// PATA.ClassA.Print();

// PATB.ClassA.Print();

ClassA.Print();

ClassA.Print();

ProjectA.TeamB.ClassA.Print();

Console.ReadKey();

}

}

}

namespace ProjectA

{

namespace TeamA

{

class ClassA

{

public static void Print()

{

Console.WriteLine("Team B print method");

}

}

}

}

namespace ProjectA

{

namespace TeamB

{

class ClassA

{

public static void Print()

{

Console.WriteLine("Team a print method");

}

}

}

}

19. Classes

Q. What is a Class?

So far in this video tutorial we have seen simple data types like int, float ,double etc. If you want to create complex custom types, then we can make use of classes

# A class consist of data and behavior. Class data is represented by its fields and behavior is represented by its method

# Purpose of a class constructor:-

The purpose of the class constructor is to initialize class fields’ class constructor is automatically called when instance of the class is created. Constructors do not have return value and always have the same name as the class Constructors do not have mandatory. If we do not provide a constructor, a default Parameter less constructor is automatically provided. Constructor can be overloaded by the number and types of parameters.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_19\_Classes

{

class Customer

{

string \_FirstName;

string \_LastName;

public Customer() :

this("NO firstName provided", "No last Name provided")

{

}

public Customer(string FirstName, string LastName)

{

this.\_FirstName = FirstName;

this.\_LastName = LastName;

}

public void PrintFullName()

{

Console.WriteLine("Full name={0}", this.\_FirstName + "" + this.\_LastName);

Console.Read();

}

~Customer()

{

//clearn up code

}

class Program

{

static void Main(string[] args)

{

//Customer c1 = new Customer("Pragim", "Technologies");

Customer c1= new Customer();

c1.PrintFullName();

}

}

}

}

20. Static and Instance class Members

When a class member includes a static modifier, the member is called as static member.

When no static modifier is present the member is called as non-static member or instance member

#Static member are invoked using class name, whereas instance member are invoked using instances (objects) of the class.

#An instance member belongs to specific instance (object) of class. If i create 3 objects of a class, i will have 3 sets of instance member in the memory, whereas there will ever be only one copy of a static member, no matter how many instance of a classes are created

NOTE:-Class members=fields, methods, properties, events, indexers, constructors.

Static Constructor

Static constructor is used to initialize static fields in a class .You declare a static constructor by using the static keywords in front of the constructor name. Static constructor is called only once, no matter how many instance you create. Static constructor are called before instance constructors

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_20\_Static\_and\_Instance\_class\_member

{

class Circle

{

static float \_PI ;

int \_Radius;

//static constructore does not have access modifiers

static Circle()

{

Console.WriteLine("static constructor called");

Circle.\_PI = 3.141F;

}

public Circle(int Radius)

{

Console.WriteLine("instance constructor called");

this.\_Radius = Radius;

}

/\*

public static void Print()

{

}

\*/

public float CalculateArea()

{

return Circle.\_PI \* this.\_Radius\*this.\_Radius;

}

}

class Program

{

static void Main(string[] args)

{

Circle C1 = new Circle(5);

float Area = C1.CalculateArea();

// Circle.Print();//calling static method

Console.WriteLine("Area={0}", Area);

Circle c2 = new Circle(6);

float Area2 = c2.CalculateArea();

Console.WriteLine("Area={0}", Area2);

Console.Read();

}

}

}

21. Inheritance

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 HourlySalary;

public void PrintFullName()

{

}

A lots of code between these two is duplicated

\*/

We can minimize the code by using the base class like above all the common code into base Employee class

public class Employee()

{

string FirstName;

string LastName;

string Email;

public void PrintFullName()

{

}

//full time and part time specific code in the respective derived classes

public class FulltimeEmployee

{

float YearlySalary;

}

public clTass partTimeEmployee

{

float YearlySalary;

}

\*/

#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, method in the base class,

, which allows reusability. In the derived class you will user only have fields, properties and methods that are specific to them.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_21\_Inheritance

{/\*

public class Employee

{

public string FirstName;

public string LastName;

public string Email;

public void printFullName()

{

Console.WriteLine(FirstName + " " + LastName);

Console.ReadKey();

}

}

public class FullTimeEmployee : Employee

{

public float YearlySalary;

}

public class PartTimeEmployee : Employee

{

public float HourlyRate;

}

\*/

public class ParentClass

{

public ParentClass()

{

Console.WriteLine("Parent class constructor called");

}

public ParentClass(string Message)

{

Console.WriteLine(Message);

}

public class ChildClass : ParentClass

{

public ChildClass():base("Derived class controlling parent class")

{

Console.WriteLine("Child class consturctor called");

}

}

class Program

{

static void Main(string[] args)

{

/\*

FullTimeEmployee FTE = new FullTimeEmployee();

FTE.FirstName = "Pragim";

FTE.LastName = "Tech";

FTE.YearlySalary = 50000;

FTE.printFullName();

PartTimeEmployee PTE = new PartTimeEmployee();

PTE.FirstName = "part";

PTE.LastName = "Time";

PTE.printFullName();

\*/

ChildClass CC = new ChildClass();

Console.ReadKey();

}

}

}

}

22. Method Hiding In C Sharp

# While inherit new class from base class we may not need the base class method in Derived class, so in that case we can hide the base class method to the derived class by Method hiding process.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_22\_Method\_Hiding\_in\_C\_sharp

{

public class Employee

{

public string FirstName;

public string LastName;

public void PrintFullName()

{

Console.WriteLine(FirstName + " " + LastName);

}

}

public class PartTimeEmployee : Employee

{

public new void PrintFullName()

{

Console.WriteLine(FirstName + "" + LastName + "\_Contractor");

//base.PrintFullName();

}

}

public class FullTimeEmployee : Employee

{

}

class Program

{

static void Main(string[] args)

{

FullTimeEmployee Fte = new FullTimeEmployee();

Fte.FirstName = "FullTime";

Fte.LastName = "Emploeyee";

Fte.PrintFullName();

PartTimeEmployee Pti = new PartTimeEmployee();

Pti.FirstName = "PartTime";

Pti.LastName = "Employee";

Pti.PrintFullName();

Console.ReadKey();

}

}

}

23. Polymorphism

Polymorphism

Polymorphism is one of the primary pillars of object-oriented programming. Polymorphism allows you to invoke derived class methods through a base class reference during runtime

In base class the method is declared virtual, and in derived class we override the same method. The virtual keyword indicates, the method can be overridden in any derived class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_23\_Ploymerphism\_in\_C\_sharp

{

public class Employee

{

public string FirstName = "FN";

public string LastName = "LN";

public void PrintFullName()

{

Console.WriteLine(FirstName + "" + LastName);

}

}

public class PartTimeEmployee : Employee

{

public void PrintFullName()

{

Console.WriteLine(FirstName + "" + LastName+ "-Parttime");

}

}

public class FullTimeEmployee : Employee

{

public void PrintFullName()

{

Console.WriteLine(FirstName + "" + LastName + "-Fulltime");

}

}

public class TemporaryEmployee : Employee

{

public void PrintFullName()

{

Console.WriteLine(FirstName + "" + LastName + "-Temporary");

}

}

class Program

{

static void Main(string[] args)

{

Employee[] employees = new Employee[4];

employees[0] = new Employee();

employees[1] = new PartTimeEmployee();

employees[2] = new FullTimeEmployee();

employees[3] = new TemporaryEmployee();

foreach (Employee e in employees)

{

e.PrintFullName();

}

Console.ReadKey();

}

}

}

24. Method Overriding vs. Method Hiding

Method Overriding

In method overriding a base class references variable pointing to a child class object, will invoke the overridden method int the child class

Method Hiding:-

In method hiding a base class reference variable pointing to a child class object, will invoke the hidden method in the base class.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_24\_MethodOverriding\_Vs\_Method\_Hidding

{

public class BaseClass

{

public virtual void Print()

{

Console.WriteLine("I am a base class print method");

Console.ReadKey();

}

}

public class DerivedClass : BaseClass

{

//public override void print() is use in case of overrride the method

public new void Print()

{

Console.WriteLine("I am a Derived class print method");

Console.ReadKey();

}

}

class Program

{

static void Main(string[] args)

{

BaseClass B = new DerivedClass();

B.Print();

DerivedClass d = new DerivedClass();

d.Print();

}

}

}