**S19: Generics and Reflection**

**Generics:-** introduced in C# 2.0. Generic is a class which allow user to define class and methods with placeholder. Without specific data type. Generic will be define using **< >.** System.Collections.Generic namespace is used. Using generic we can use any kind of data type. Its type safety: don’t want compile time error. Same placeholder need to pass. Eg:-If array then argument should be array.We can use generics with field methods and parameter. At compile time. We can create generic delegate, event, method, class. Using type of operator we can check the generic type at run time. Reuse the code. It removes boxing and unboxing capability. We can pass multiple value. Genric method can be declare using **virtual override and abstract.**

* **Generic Class:-** generic class used for any kind of data type. Class declaration followd by type parameter< >. Generic replace these parameter with datatype.
* **Reflections: -** it’s used to display metadata types. System. Reflection is used for reflection. It display the information about the assemblies. Type keyword is used to disaplya the type. It’s checked the type on the basis of dll. It uses difrent classes such as **Assembly:-** describe an assembly reusable versionable, and self-describing block of common language runtime application. AssemblyName, ConstructorInfo etc.

OopsSessions.cs

using OOPS\_\_AllSession;

using System;

using static OOPS\_\_AllSession.S11\_\_ClassAndTypes;

namespace Oops\_\_AllSession

{

class OopsSessions

{

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\*\*\*\*Welcome To Main Method\*\*\*\*\*\*\*\*\*\*\*");

S19\_\_GenericAndReflections generic = new S19\_\_GenericAndReflections();

generic.CalledGenericMethod();

}

}

}

S19\_\_GenericAndReflections.cs

using System;

using System.Collections.Generic;

using System.Reflection;

using System.Text;

namespace OOPS\_\_AllSession

{

class S19\_\_GenericAndReflections

{

public void CalledGenericMethod()

{

S19\_\_GenericAndReflections generic = new S19\_\_GenericAndReflections();

int[] number = new int[5];

number[0] = 1;

number[1] = 11;

number[2] = 110;

number[3] = 'C';

number[4] = 'D';

string[] city = { "Panvel", "Pune", "Nanded", "Latur", "Beed", "Mumbai", "Dubai" };

string[] fruits = { "Orange", "Mango", "PineApple", "Apple", "Strawberry", "Muskmelon", "Gauva" };

string[] names = { "Amit", "Amir", "Sahiba", "Arif", "Kanak", "Prakash","Akhil", "Shubham" };

generic.GenericMethod(number);

generic.GenericMethod(city);

generic.GenericMethod\_MultipleParameter(fruits,names);

GenericClass<string>.PrintData();

GenericClass<string>.Reflections();

}

public void GenericMethod<placeholder>(placeholder[] data)

{

for (int i = 0; i < data.Length; i++)

Console.Write(data[i]+" , ");

Console.WriteLine();

}

public void GenericMethod\_MultipleParameter<placeholder>(placeholder[] fruits, placeholder[] names)

{

Console.Write($"\nLength of Array is : {fruits.Length}\nFruits Are:-- ");

for (int i = 0; i < fruits.Length; i++)

Console.Write(fruits[i]+" , ");

Console.Write($"\n\nLength of Array is : {names.Length} \nNames Are: \t");

foreach (placeholder name in names)

Console.Write(name + ", ");

Console.WriteLine(typeof(placeholder));

}

}

class GenericClass<placeholderData>

{

placeholderData no { get; set; }

placeholderData no1 { get; set; }

public static void PrintData()

{

GenericClass<string> genericClass = new GenericClass<string>();

genericClass.no = "Imran";

Console.WriteLine("My Name is : "+genericClass.no);

//genericClass.no1 = 100; only define data type support

}

//Reflections\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static void Reflections()

{

Type typeCheking = typeof(String); /// Detail of Class String

Console.WriteLine("Type of : "+typeCheking.Name);

MethodInfo[] methodsInfo = typeCheking.GetMethods();

foreach (var methods in methodsInfo)

{

Console.WriteLine("Method Info:- "+methods.Name);

ParameterInfo[] parameter = methods.GetParameters();

foreach(var parameters in parameter)

Console.WriteLine("Parameter Info:- "+parameters.Name+" : "+parameters.ParameterType);

}

Console.WriteLine("\n\nYou Are In The Property Reflections\*\*\*\*\*\*\*\*\*\*");

PropertyInfo[] propertyInfo = typeCheking.GetProperties();

foreach(var property in propertyInfo)

Console.WriteLine("\tProperty Info: "+property);

}

}

}