**Singleton**

**ILog.cs**

**public interface ILog**

**{**

**void LogException(string message);**

**}**

**Log.cs**

**public sealed class Log : Ilog**

**{**

**private Log()**

**{**

**}**

**private static readonly Log LogInstance = new Log();**

**public static Log GetInstance()**

**{**

**return LogInstance;**

**}**

**public void LogException(string message)**

**{**

**Console.Write(“Log Here”);**

**}**

**}**

**EmployeeController.cs**

**public class EmployeeController**

**{**

**private ILog \_ILog;**

**public EmployeeController()**

**{**

**\_ILog = Log.GetInstance();**

**}**

**//Whenever Any Exception Occurred, the following OnException Method will Execute**

**protected override void OnException(ExceptionContext filterContext)**

**{**

**\_ILog.LogException(filterContext.Exception.ToString());**

**}**

**}**

**Factory**

**public interface ICreditCard**

**{**

**string GetCardType();**

**int GetCreditLimit();**

**int GetAnnualCharge();**

**}**

**public class CreditCardFactory**

**{**

**public static ICreditCard GetCreditCard(string cardType)**

**{**

**ICreditCard cardDetails = null;**

**if (cardType == "MoneyBack")**

**{**

**cardDetails = new MoneyBack();**

**}**

**else if (cardType == "Titanium")**

**{**

**cardDetails = new Titanium();**

**}**

**else if (cardType == "Platinum")**

**{**

**cardDetails = new Platinum();**

**}**

**return cardDetails;**

**}**

**}**

**static void Main(string[] args)**

**{**

**ICreditCard cardDetails = CreditCardFactory.GetCreditCard("Platinum");**

**if (cardDetails != null)**

**{**

**Console.WriteLine("CardType : " + cardDetails.GetCardType());**

**Console.WriteLine("CreditLimit : " + cardDetails.GetCreditLimit());**

**Console.WriteLine("AnnualCharge :" + cardDetails.GetAnnualCharge());**

**}**

**else**

**{**

**Console.Write("Invalid Card Type");**

**}**

**}**

**Dependency Injection**

**EmployeeDAL.cs**

**public interface IEmployeeDAL**

**{ List<Employee> SelectAllEmployees();**

**}**

**public class EmployeeDAL : IEmployeeDAL**

**{**

**public List<Employee> SelectAllEmployees()**

**{**

**List<Employee> ListEmployees = new List<Employee>**

**{**

**new Employee() { ID = 1, Name = "Pranaya", Department = "IT" },**

**new Employee() { ID = 2, Name = "Kumar", Department = "HR" },**

**new Employee() { ID = 3, Name = "Rout", Department = "Payroll" }**

**};**

**return ListEmployees;**

**}**

**}**

**EmployeeBL.cs**

**public class EmployeeBL**

**{**

**public IEmployeeDAL employeeDAL;**

**//Injecting the Dependency Object using Constructor means it is a Loose Coupling**

**public EmployeeBL(IEmployeeDAL employeeDAL)**

**{**

**this.employeeDAL = employeeDAL;**

**}**

**public List<Employee> GetAllEmployees()**

**{**

**return employeeDAL.SelectAllEmployees();**

**}**

**}**

**Program.cs**

**static void Main(string[] args)**

**{**

**EmployeeBL employeeBL = new EmployeeBL(new EmployeeDAL());**

**List<Employee> ListEmployee = employeeBL.GetAllEmployees();**

**foreach (Employee emp in ListEmployee)**

**{**

**Console.WriteLine($"ID = {emp.ID}, Name = {emp.Name}, Department = {emp.Department}");**

**}**

**}**

**Custom Exception**

**[Serializable]**

**public class StudentNotFoundException : Exception**

**{**

**public string StudentName { get; }**

**public StudentNotFoundException() { }**

**public StudentNotFoundException(string message)**

**: base(message) { }**

**public StudentNotFoundException(string message, Exception inner)**

**: base(message, inner) { }**

**public StudentNotFoundException(string message, string studentName)**

**: this(message)**

**{**

**StudentName = studentName;**

**}**

**}**

**Lamda Expression**

**public class LambdaExpression**

**{**

**public delegate string GreetingsDelegate(string name);**

**static void Main(string[] args)**

**{**

**GreetingsDelegate obj = (name) =>**

**{**

**return "Hello @" + name + " welcome to Dotnet Tutorials";**

**};**

**string GreetingsMessage = obj.Invoke("Pranaya");**

**Console.WriteLine(GreetingsMessage);**

**Console.ReadKey();**

**}**

**}**

**Deligate**

**public delegate void MyDelegate(string s);**

**class MyClass**

**{**

**public static void Hello(string s)**

**{**

**Console.WriteLine(" Hello, {0}!", s);**

**}**

**public static void Goodbye(string s)**

**{**

**Console.WriteLine(" Goodbye, {0}!", s);**

**}**

**public static void Main()**

**{**

**MyDelegate a, b, c, d;**

**a = new MyDelegate(Hello);**

**b = new MyDelegate(Goodbye);**

**c = a + b;**

**d = c - a;**

**Console.WriteLine("Invoking delegate a:");**

**a("A");**

**Console.WriteLine("Invoking delegate b:");**

**b("B");**

**Console.WriteLine("Invoking delegate c:");**

**c("C");**

**Console.WriteLine("Invoking delegate d:");**

**d("D");**

**}**

**}**

**Generic**

**class MyGenericClass<T>**

**{**

**private T GenericMemberVariable;**

**public MyGenericClass(T value)**

**{**

**GenericMemberVariable = value;**

**}**

**public T GenericMethod(T GenericParameter)**

**{**

**Console.WriteLine($"Parameter type: {typeof(T).ToString()}, Value: {GenericParameter}");**

**Console.WriteLine($"Return type: {typeof(T).ToString()}, Value: {GenericMemberVariable}");**

**return GenericMemberVariable;**

**}**

**}**

**static void Main()**

**{**

**MyGenericClass<int> integerGenericClass = new MyGenericClass<int>(10);**

**int val1 = integerGenericClass.GenericMethod(200);**

**Console.WriteLine(val1);**

**MyGenericClass<string> stringrGenericClass = new MyGenericClass<string>("Kamal");**

**string val2 = stringrGenericClass.GenericMethod("Pratap");**

**Console.WriteLine(val2);**

**}**

**Collection**

**ArrayList arrayList1 = new ArrayList();**

**arrayList1.Add(101);**

**foreach (var item in arrayList1)**

**{**

**Console.WriteLine(item);**

**}**

**Hashtable hashtable = new Hashtable();**

**hashtable.Add("EId", 1001);**

**foreach (object obj in hashtable.Keys)**

**{**

**Console.WriteLine(obj + " : " + hashtable[obj]);**

**}**

**Dictionary<string, string> dictionaryCountries = new Dictionary<string, string>();**

**dictionaryCountries.Add("UK", "London, Manchester, Birmingham");**

**foreach (KeyValuePair<string, string> KVP in dictionaryCountries)**

**{**

**Console.WriteLine($"Key:{KVP.Key}, Value: {KVP.Value}");**

**}**

**List<string> countries = new List<string>();**

**countries.Add("INDIA");**

**Ref Out**

**static void Main(string[] args)**

**{**

**int a = 15;**

**UpdateValue(ref a);**

**Console.WriteLine(a);**

**Console.ReadKey();**

**}**

**static void UpdateValue(ref int b)**

**{**

**b = 30;**

**}**

**Remove Duplicate**

**for (int i = 0; i < size; i++)**

**{**

**for (int j = i + 1; j < size; j++)**

**{**

**if (arr[i] == arr[j])**

**{**

**for (int k = j; k < size - 1; k++)**

**{**

**arr[k] = arr[k + 1];**

**}**

**size--;**

**j--;**

**}**

**}**

**}**

**Character Count**

**string str = "ssmmmjjkkkkrrr";**

**Dictionary<char, int> counts = new Dictionary<char, int>();**

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

**{**

**if (counts.ContainsKey(str[i]))**

**{**

**counts[str[i]]++;**

**}**

**else**

**{**

**counts.Add(str[i], 1);**

**}**

**}**

**foreach (var count in counts)**

**Console.WriteLine("{0} = {1}", count.Key, count.Value.ToString());**

**Reverse**

**num = int.Parse(Console.ReadLine());**

**while (num != 0)**

**{**

**reverse = reverse \* 10;**

**reverse = reverse + num % 10;**

**num = num / 10;**

**}**

**Pass Parameter**

**public static int RunTheMethod(Func<string, int> myMethodName)**

**{**

**int i = myMethodName("My String");**

**return i;**

**}**

**Optional Parameter**

**public static void ADDNumbers(int FN, int SN, int[] restOfTheNumbers = null)**

**{**

**int result = FN + SN;**

**if (restOfTheNumbers != null)**

**{**

**foreach (int i in restOfTheNumbers)**

**{**

**result += i;**

**}**

**}**

**Alter table Person add constraint Person\_GenderId\_FK FOREIGN KEY (GenderId) references Gender(ID)**

**NEXT VALUE for [dbo].[SequenceObject]**

**CREATE CLUSTERED INDEX IX\_Employee\_ID ON Employee(Id ASC);**

**CREATE NONCLUSTERED INDEX IX\_tblOrder\_ProductId ON dbo.tblOrder (ProductId)**

**INCLUDE ([Id],[CustomerId],[ProductName])**

**ROW\_NUMBER() OVER (ORDER BY Department) AS RowNumber**

**Retuning Stored Proc**

**CREATE PROCEDURE spGetResult**

**@No1 INT, @No2 INT, @Result INT OUTPUT**

**AS**

**BEGIN SET @Result = @No1 + @No2 END**

**DECLARE @Result INT**

**EXECUTE spGetResult 10, 20, @Result OUT**

**Function**

**CREATE FUNCTION SVF1(@X INT)**

**RETURNS INT**

**AS**

**BEGIN RETURN @X \* @X \*@X END**

**CREATE FUNCTION FN\_GetStudentDetailsByID**

**( @ID INT )**

**RETURNS TABLE**

**AS**

**RETURN (SELECT \* FROM Student WHERE ID = @ID)**

**CREATE FUNCTION MSTVF\_GetEmployees()**

**RETURNS @Table Table (ID int, Name nvarchar(20), DOB Date)**

**AS**

**BEGIN**

**INSERT INTO @Table SELECT ID, Name, Cast(DOB AS Date) FROM Employee**

**Return**

##### End

**Transaction**

**BEGIN TRANSACTION**

**INSERT INTO Product VALUES(110,'Product-10',600, 30)**

**INSERT INTO Product VALUES(110,'Product-10',600, 30)**

**IF(@@ERROR > 0)**

**BEGIN**

**Rollback Transaction**

**END ELSE BEGIN**

**Commit Transaction**

**END**

**Exception Handling**

**RAISERROR('Second Number Cannot be zero',16,1)**

**IF(@@ERROR <> 0)**

**BEGIN**

**PRINT 'Error Occurred'**

**END**

**Trigger**

**CREATE TRIGGER trInsertEmployee ON Employee**

**FOR INSERT --UPDATE / DELETE**

**INSTEAD OF INSERT --UPDATE / DELETE**

**AS**

**BEGIN**

**PRINT 'YOU CANNOT PERFORM INSERT OPERATION'**

**ROLLBACK TRANSACTION**

**END**

**View**

**CREATE VIEW vwTotalSalesPriceByProduct**

**WITH SCHEMABINDING**

**AS**

**SELECT Name,COUNT\_BIG(\*) AS TotalTransactions,SUM(ISNULL((QuantitySold \* UnitPrice), 0)) AS TotalSalesPrice**

**FROM dbo.ProductSales prdSales INNER JOIN dbo.Product prd ON prd.ProductId = prdSales.ProductId**

**GROUP BY Name**

**CREATE UNIQUE CLUSTERED**

**INDEX UIX\_vwTotalSalesPriceByProduct\_Name**

**ON vwTotalSalesPriceByProduct(Name)**

**Delete Duplicate**

**WITH EmployeesCTE as**

**(**

**SELECT\*, ROW\_NUMBER() over (PARTITION BY ID ORDER BY ID) as RowNumber**

**FROM Employees**

**)**

**DELETE FROM EmployeesCTE WHERE RowNumber>1**

**Count Character**

**Declare @myvar varchar(20),@substr Varchar(1),@strcount int**

**Set @myvar = 'POOJA'**

**Declare @i int=0**

**While(@i<LEN(@myvar))**

**Begin**

**Select @substr=SUBSTRING (@myvar, 1, 1)**

**Select @strcount=LEN(@myvar) - LEN(REPLACE(@myvar,@substr,''))**

**Select @substr+ ' Count is '+Cast(@strcount As varchar)**

**Select @myvar=REPLACE(@myvar,@substr,'')**

**End**

**1 To 10**

**; with CTE as**

**(**

**select 1 Number**

**union all**

**select Number +1 from CTE where Number<10**

**)**

**select \* from CTE**

**I Enumerable**

**List<int> integerList = new List<int>() {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};**

**IEnumerable<int> QuerySyntax = from obj in integerList**

**where obj > 5 select obj;**

**IEnumerable<int> EvenNumbers = Enumerable.Where(integerList, n => n % 2 == 0);**

**List<Student> studentList = new List<Student>()**

**{**

**new Student(){ID = 1, Name = "James", Gender = "Male"},**

**new Student(){ID = 2, Name = "Sara", Gender = "Female"}**

**};**

**IEnumerable<Student> QuerySyntax = from std in studentList**

**where std.Gender == "Male" select std;**

**IQueryable<Student> MethodSyntax = studentList.AsQueryable()**

**.Where(std => std.Gender == "Male");**

**foreach (var student in QuerySyntax)**

**{**

**Console.WriteLine( $"ID : {student.ID} Name : {student.Name}");**

**}**

**Linq Distinct**

**//Using Method Syntax**

**var MS = intCollection.Distinct();**

**//Using Query Syntax**

**var QS = (from num in intCollection**

**select num).Distinct();**

**Linq Join**

**var customers = new List<Customer>**

**{**

**new Customer { Id = 1, Name = "Alice" },**

**new Customer { Id = 2, Name = "Bob" }**

**};**

**var orders = new List<Order>**

**{**

**new Order { OrderId = 101, CustomerId = 1 },**

**new Order { OrderId = 102, CustomerId = 2 }**

**};**

**var result = from customer in customers**

**join order in orders on customer.Id equals order.CustomerId**

**select new { CustomerName = customer.Name, OrderId = order.OrderId };**

**foreach (var item in result)**

**{**

**Console.WriteLine($"Customer: {item.CustomerName}, Order ID: {item.OrderId}");**

**}**