**C# 3.0**

**1. What are new features of C#3.0?**

* Implicitly Typed Local Variables

Using "Var" keyword we can declare implicit type variable. In implicitly typed local variable declaration, the type of the local variable being declared is inferred from the expression used to initialize the variable.

E.g

int a = 5;

var b = 6;

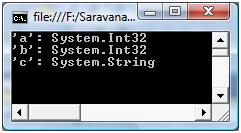
var c = "myName";

Console.WriteLine("'a': "+a.GetType());

Console.WriteLine("'b': " + b.GetType());

Console.WriteLine("'c': " + c.GetType());

Console.ReadLine();



* Object and Collection Initializers

In this new features we can initialize the members of the object using { and } parenthesis and separated by commas. Each member to which we are initializing should be accessible field or property, followed by an equal's sign and an expression or an object or value.

E.g

public class Employee

{

string \_empID;

public string EmpID

{

get { return \_empID; }

set { \_empID = value; }

}

string \_name;

public string Name

{

get { return \_name; }

set { \_name = value; }

}

long \_salary;

public long Salary

{

get { return \_salary; }

set { \_salary = value; }

}

}

//Object and Collection Initializers

Employee \_emp = new Employee() { EmpID = "1001", Name = "Sam", Salary = 10000 };

* Anonymous Types

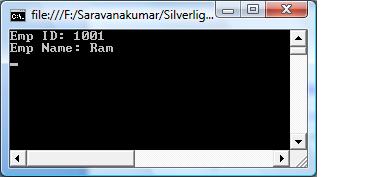
In c# 3.0, we can create an object of an anonymous type suing "new" keyword. It is nameless class type that inherits directly from "object". The members of an anonymous type are a sequence of read/write properties , it will be initialize while creating object.

var emp = new { ID = "1001", Name = "Ram", Designation = "SE" };

Console.WriteLine("Emp ID: "+emp.ID );

Console.WriteLine("Emp Name: " + emp.Name);

Console.ReadLine();



* Automatically Implemented Properties

In earlier for creating property inside a class, we need to create a private member and create a property that private member for accessing this member field. But in case of C#3.0 we can automatically implement the properties without member field.

E.g

public class Employee

{

public string EmpID

{ get; set; }

public string Name

{ get; set; }

public int Age

{ get; set; }

}

* Extension Methods

It is a new feature in c# 3.0, which allows developer to add functionality in existing class without modifying the existing class or recompiling the existing class or extending the existing class.

E.g: In this example we have extended the "string" class with new method "AppendedMyString()"

public static class ExtendedMethodForString

{

public static string AppendedMyString(this string s)

{

return s + "myNewStringAppended";

}

}

string a;

Console.WriteLine(a.AppendedMyString());

* Language integrated Query(LINQ)

**2. How will you declare implicit type variable?**

Implicit type variable can be declared using "var" keyword. In implicitly typed local variable declaration, the type of the local variable being declared is inferred from the expression used to initialize the variable.

While declaring implicit type variable it should follow these restrictions.

* The declarator must include an initializer.
* The initializer must be an expression.
* The initializer expression must have a compile-time type which cannot be the null type.
* The local variable declaration cannot include multiple declarators.
* The initializer cannot refer to the declared variable itself

Example:

var b = 6;

var c = "Hello";

var d = 1.0;

var numbers = new int[] { 1, 2, 3 };

var orders = new Dictionary();

var name; //Invalid declartion

**3. How will you declare collection of object using c# 3.0?**

Collections of object can be initialize using { and } parenthesis and separated by commas. In the below example we have declared the Employee class with EmpID and Name properties. While creating the collection of employee, we have created new instance for employee and assign value to all the properties as shown below.

Example:

public class Employee

{

string \_empID;

public string EmpID

{

get { return \_empID; }

set { \_empID = value; }

}

string \_name;

public string Name

{

get { return \_name; }

set { \_name = value; }

}

}

//Object and Collection Initializers

List empColl = new List { new Employee() { EmpID = "1001", Name = "Sam" }

, new Employee() { EmpID = "1002", Name = "Ram" } };

foreach (Employee emp in empColl)

{

Console.WriteLine(emp.Name );

}

**4. What is mean by Anonymous type?**

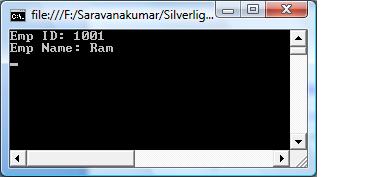
In c# 3.0, we can create an object of an anonymous type suing "new" keyword. It is nameless class type that inherits directly from "object". The members of an anonymous type are a sequence of read/write properties, it will be initializing while creating object.

var emp = new { ID = "1001", Name = "Ram", Designation = "SE" };

Console.WriteLine("Emp ID: "+emp.ID );

Console.WriteLine("Emp Name: " + emp.Name);

Console.ReadLine();



**5. How will you declare property in c# 3.0?**

In C#3.0 we can declare member of the class as Property without creating member field for specific property. In the below example we have created the Employee class with EmpID, Name and Age property.

Example:

public class Employee

{

public string EmpID

{ get; set; }

public string Name

{ get; set; }

public int Age

{ get; set; }

}

**6. What is mean by Extension method?**

It is a new feature in c# 3.0, which allows developer to add functionality in existing class without modifying the existing class or recompiling the existing class or extending the existing class.

While declaring implicit type variable it should follow these restrictions.

* • Extended method should be public and static
* • First parameter should be decorated with modifier "this", i.e it takes extenting class as first parameter and it is called as Instance Parameter.
* • No other modifers like "ref", "out" etc are allowed with "this" modifer
* • The instance parameter can not be a pointer type.
* • The instance parameter can not have the type of the type parameter. The below is not possible.

public static int Obj (this T param)

* • If an extension method conflicts with a member method of target type , always member method is get invoked instead of extension method.

Example: In this example we have extended the "string" class with new method "AppendedMyString()"

public static class ExtendedMethodForString

{

public static string AppendedMyString(this string s)

{

return s + "myNewStringAppended";

}

}

string a;

Console.WriteLine(a.AppendedMyString());

**7. What is the use of Extension method?**

* It is used to extends the existing type (either value type or reference type)by adding new methods without deriving it into a new type.
* Microsoft intelligence support, which could show all extension method accessible to a given identifier.

**8. What is mean by Predicate and Projection?**

**Predicate**– It is a Boolean expression that is intended to indicate membership of an element in a group. Example: it is used to define how to filter items inside a loop.

//Pridicate

(age) => age > 21;

**Projection**- is an expression that returns a type different from the type of its single parameter. E.g

//Projection: take sting as parameter and return int

(str) => str.Length