.NET PROGRAMMING (C#)
18 MCA 4 2 C

UNIT –III
ADVANCED FEATURE OF C#

FACULTY

Dr. K.ARTHI, MCA, M.Phil., Ph.D.

Assistant Professor,

Post Graduate and Research Department of Computer Applications,

Government Arts College (Autonomous),

Coimbatore – 641 018.

CONTENT

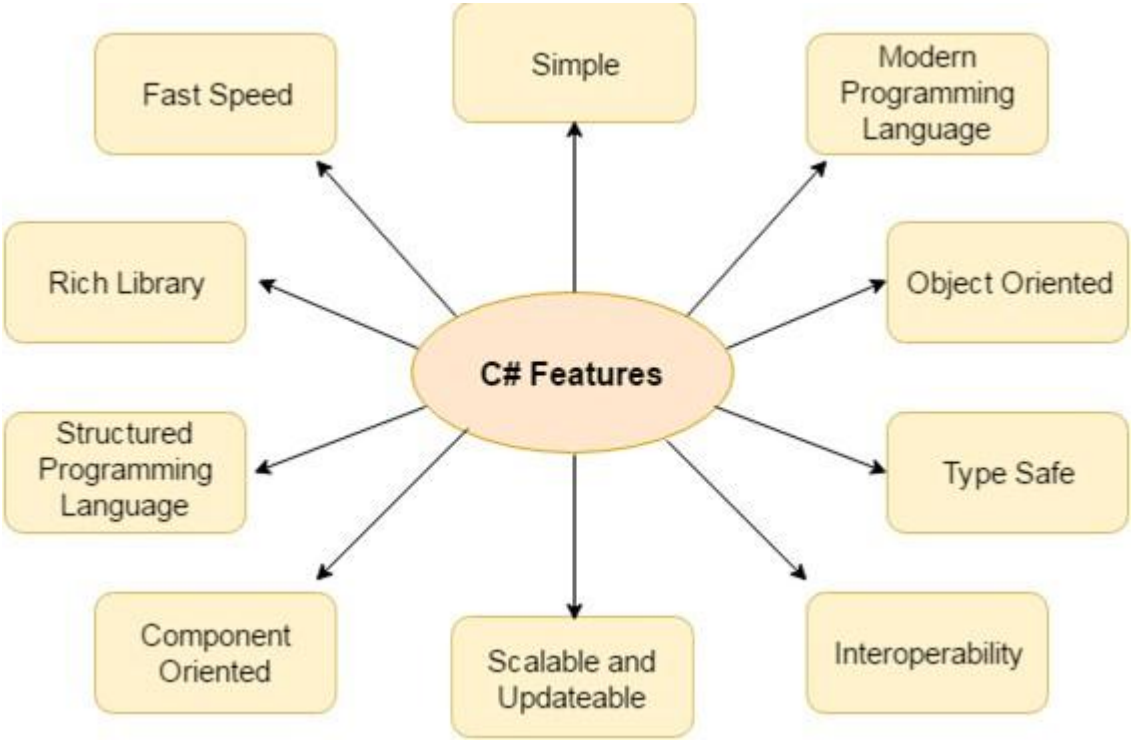
- **Advantage of C#**
  - Serialization
  - Deserialization
  - Serializing XML in C#
  - Multithreading
  - Reflection
  - Attributes
  - Properties
  - Indexers

ADVANCED FEATURES OF C#

- C# is object oriented programming language. It provides a lot of features that are given below.
- Simple - provides structured approach, rich set of library functions, data types etc.
- Modern programming language - based upon the current trend and it is very powerful and simple for building scalable, interoperable and robust applications.
- Object oriented - Procedure-oriented programming language it is not easy to manage if code grows as project size grow.

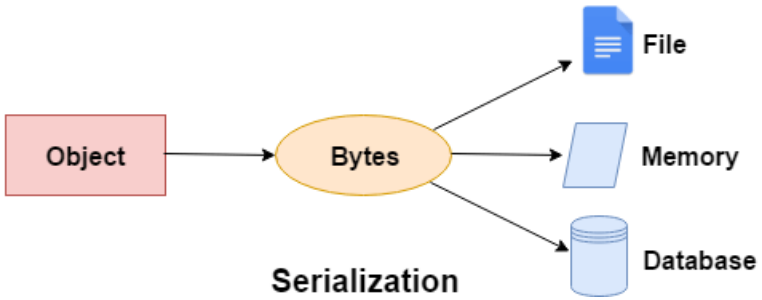
- Type safe - C# type safe code can only access the memory location that it has permission to execute.Therefore it improves a security of the program.
- Interoperability - Interoperability process enables the C# programs to do almost anything that a native C++ application can do.
- Scalable and Updateable - C# is automatic scalable and updateable programming language. For updating our application we delete the old files and update them with new ones.
- Component oriented - C# is component oriented programming language. It is the predominant software development methodology used to develop more robust and highly scalable applications.

- Structured programming language - C# is a structured programming language in the sense that we can break the program into parts using functions. So, it is easy to understand and modify.
- Rich Library- C# provides a lot of inbuilt functions that makes the development fast
- Fast speed - The compilation and execution time of C# language is fast.


C# Features
Simple
Modern Programming Language
Object Oriented
Type Safe
Interoperability
Scalable and Updateable
Component Oriented
Structured Programming Language
Rich Library
Fast Speed

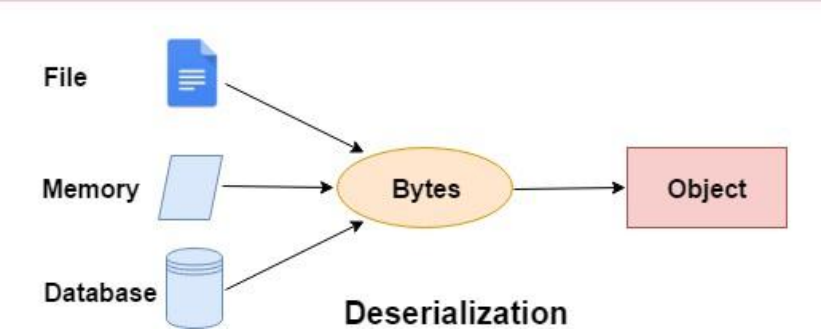
C# SERIALIZATION

- In C#, serialization is the process of converting object into byte stream so that it can be saved to memory, file or database. The reverse process of serialization is called deserialization.
- Serialization is internally used in remote applications.



C# DESERIALIZATION

- In C# programming, deserialization is the reverse process of serialization. It means you can read the object from byte stream.



SERIALIZING XML IN C#

- objects and classes can be serialized without adding any special directives or attributes to the code. By default, all public properties of a class are already serializable.
- The actual serialization is done by an instance of the class XmlSerializer, from the System.Xml.Serialization namespace. The serializer's constructor requires a reference to the type of object it should work with – which can be obtained by using the GetType() method of an instanced object, or a call to the function typeof() and specifying the class name as the only argument.

C# - MULTITHREADING

- A thread is defined as the execution path of a program. Each thread defines a unique flow of control.
- Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems.
- In C#, the System.Threading.Thread class is used for working with threads. It allows creating and accessing individual threads in a multithreaded application

- C# program starts execution, the main thread is automatically created.
- The threads created using the Thread class are called the child threads of the main thread.

EXAMPLE

```
using System;
using
System.Threading;
namespace
MultithreadingApplication
 {
class MainThreadProgram
{ static void Main(string[] args)
{Thread th = Thread.CurrentThread; th.Name = "MainThread"; Console.WriteLine("This
is {0}", th.Name); Console.ReadKey(); } } }
```

REFLECTION

- Reflection objects are used for obtaining type information at runtime.The classes that give access to the metadata of a running program are in the System.Reflectionnamespace.
- The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application.

APPLICATIONS OF REFLECTION

- It allows view attribute information at runtime.
- It allows examining various types in an assembly and instantiate these types.
- It allows late binding to methods and properties
- It allows creating new types at runtime and then performs some tasks using those types.

C# -ATTRIBUTES

- An attribute is a declarative tag that is used to convey information to runtime about the behaviors of various elements like classes, methods, structures, enumerators, assemblies etc. in your program.
- You can add declarative information to a program by using an attribute. A declarative tag is depicted by square ([ ]) brackets placed above the element it is used for.

SPECIFYING AN ATTRIBUTE

- Syntax for specifying an attribute is as follows –
- [attribute(positional_parameters, name_parameter = value, ...)]element
- Predefined Attributes
- AttributeUsage
- Conditional
- Obsolete

ATTRIBUTEUSAGE

- The pre-defined attribute AttributeUsage describes how a custom attribute class can be used. It specifies the types of items to which the attribute can be applied.
- Syntax for specifying this attribute is as follows −
- [AttributeUsage (validon, AllowMultiple = allowmultiple, Inherited = inherited)]

C# - PROPERTIES

- Properties are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called Fields. Properties are an extension of fields and are accessed using the same syntax. They use accessors through which the values of the private fields can be read, written or manipulated.
- Properties do not name the storage locations. Instead, they have accessors that read, write, or compute their values.

ACCESSORS

- The accessor of a property contains the executable statements that helps in getting (reading or computing) or setting (writing) the property. The accessor declarations can contain a get accessor, a set accessor, or both.
- Abstract Properties
- An abstract class may have an abstract property, which should be implemented in the derived class

C# - INDEXERS

- An indexer allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([ ]).
- Syntax
- A one dimensional indexer has the following syntax −
- element-type this[int index]
- { //The get accessor. get
- { //return the value specified by index } //The set accessor. set
- { //set the value specified by index } }

USE OF INDEXERS

- Declaration of behavior of an indexer is to some extent similar to a property. similar to the properties, you use get and set accessors for defining an indexer.
- However, properties return or set a specific data member, whereas indexers returns or sets a particular value from the object instanceDefining a property involves providing a property name. Indexers are not defined with names, but with the this keyword, which refers to the object instance.

OVERLOADED INDEXERS

- Indexers can be overloaded. Indexers can also be declared with multiple parameters and each parameter may be a different type. It is not necessary that the indexes have to be integers.

EXAMPLE

```
class IndexedNames
{ private string[] namelist = new string[size];
static public int size = 10;
public IndexedNames()
 { for (int i = 0; i < size; i++) { namelist[i] = "N.A."; } }
public string this[int index]
{ get { string tmp; if( index >= 0 && index <= size-1 )
 { tmp = namelist[index]; } else { tmp = "";
} return ( tmp ); } set { if( index >= 0 && index <= size-1 )
{ namelist[index] = value;
} }
```

```
} public int this[string name] { get { int index = 0; while(index < size)
{ if (namelist[index] == name)
 { return index; } index++; } return index;
} } static void Main(string[] args)
 { IndexedNames names = new IndexedNames();
 names[0] = "Zara"; names[1] = "Riz"; names[2] = "Nuha"; names[3] = "Asif";
names[4] = "Davinder"; names[5] = "Sunil"; names[6] = "Rubic"; //using the first
indexer with int parameter for (int i = 0; i < IndexedNames.size; i++) {
Console.WriteLine(names[i]); } //using the second indexer with the string
parameter Console.WriteLine(names["Nuha"]); Console.ReadKey(); } }}
```

THANK YOU

THE CONTENT IN THIS E-MATERIAL IS TAKEN FROM THE TEXTBOOKS AND REFERENCE BOOKS PRESCRIBED IN THE SYLLABUS