IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Understanding The Core Concepts Classes, Objects, Interface, Serialization, Delegate, Reflection And Type Class Of C Sharp Programming Language By Using Asp.Net.

Dr. Rishi Mathur
Assistant Professor
Department of Computer Science and Application
S.P.U. (P.G.) College, Falna, Rajasthan

ABSTRACT

A class in C# is a blueprint or template that is used for declaring an object. A class consists of member variables, functions, properties etc. and In C#, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc. In other words, object is an entity that has state and behavior. Here, state means data and behaviour means functionality. Object is a runtime entity, it is created at runtime. Object is an instance of a class. All the members of the class can be accessed through object. Interface in C# is a blueprint of a class. It is like abstract class because all the methods which are declared inside the interface are abstract methods. It cannot have method body and cannot be instantiated. It is used *to achieve multiple inheritance* which can't be achieved by class. It is used *to achieve fully abstraction* because it cannot have method body. Its implementation must be provided by class or struct. The class or struct which implements the interface, must provide the implementation of all the methods declared inside the interface. 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. C# delegates are similar to pointers to functions, in C or C++. A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime. Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class. In C sharp 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.Reflection namespace. 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. C# Type class represents type declarations for class types, interface types, enumeration types, array types, value types etc. It is found in System namespace. It inherits System.Reflection.MemberInfo class.

Keywords : Classes, Objects, Interface, Serialization, Deserialization, Delegate, Reflection, Type Class

C# | Class and Object

Class and Object are the basic concepts of Object-Oriented Programming which revolve around the real-life entities. A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods(member function which defines actions) into a single unit. In C#, classes support polymorphism, inheritance and also provide the concept of derived classes and base classes.

Declaration of class

Generally, a class declaration contains only keyword **class**, followed by an **identifier(name)** of the class. But there are some optional attributes that can be used with class declaration according to the application requirement. In general, class declarations can include these components, in order:

- **Modifiers:** A class can be public or internal etc. By default modifier of class is *internal*.
- **Keyword class:** A *class* keyword is used to declare the type class.
- **Class Identifier:** The variable of type class is provided. The identifier(or name of class) should begin with a initial letter which should be capitalized by convention.
- **Base class or Super class:** The name of the class's parent (superclass), if any, preceded by the *: (colon)*. This is optional.
- **Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the **: (colon)**. A class can implement more than one interface. This is optional.
- **Body:** The class body is surrounded by { } (curly braces).

Constructors in class are used for initializing new objects. Fields are variables that provide the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

```
// declaring public class
public class Geeks
{

    // field variable
    public int a, b;

      // member function or method
      public void display()
      {
          Console.WriteLine("Class & Objects in C#");
      }
}
```

Objects

It is a basic unit of Object-Oriented Programming and represents the real-life entities. A typical C# program creates many objects, which as you know, interact by invoking methods. An object consists of :

- **State:** It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

C# Interface

Interface in C# is a blueprint of a class. It is like abstract class because all the methods which are declared inside the interface are abstract methods. It cannot have method body and cannot be instantiated.

It is used *to achieve multiple inheritance* which can't be achieved by class. It is used *to achieve fully abstraction* because it cannot have method body.

Its implementation must be provided by class or struct. The class or struct which implements the interface, must provide the implementation of all the methods declared inside the interface.

the example of interface in C# which has draw() method. Its implementation is provided by two classes: Rectangle and Circle.

```
using System;
public interface Drawable
{
   void draw();
}
public class Rectangle : Drawable
{
   public void draw()
   {
      Console.WriteLine("drawing rectangle...");
   }
}
public class Circle : Drawable
{
   public void draw()
   {
      Console.WriteLine("drawing circle...");
   }
}
public class TestInterface
{
   public static void Main()
   {
      Drawable d;
      d = new Rectangle();
```

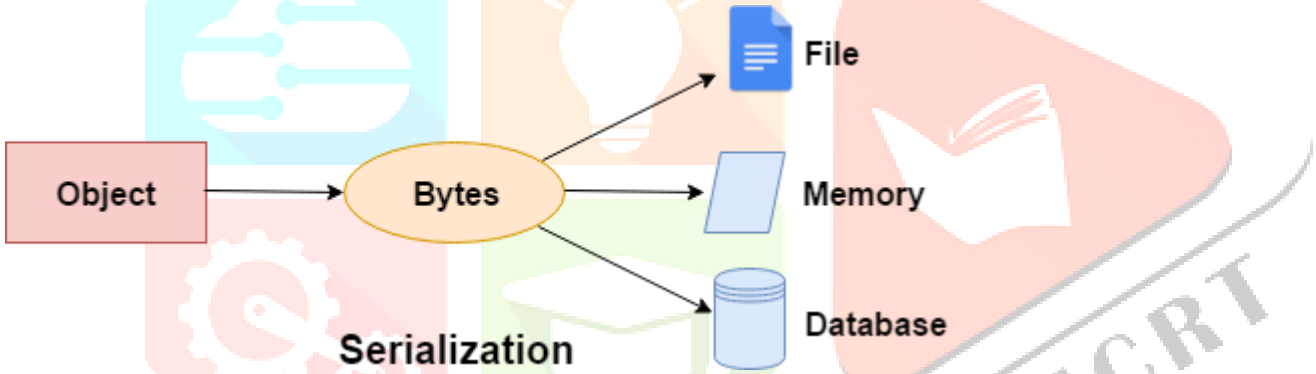
```
            d.draw();
            d = new
            Circle();
            d.draw();
        }
    }
```

Interface methods are public and abstract by default. You cannot explicitly use public and abstract keywords for an interface method.

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# Serializable Attribute

To serialize the object, you need to apply *SerializableAttribute* attribute to the type. If you don't apply *SerializableAttribute* attribute to the type, *SerializationException* exception is thrown at runtime.

C# Serialization example

```
using System;
using System.IO;
using
System.Runtime.Serialization.Formatters.Binary;
[Serializable]
class Student
{
   int rollno;
   string name;
```

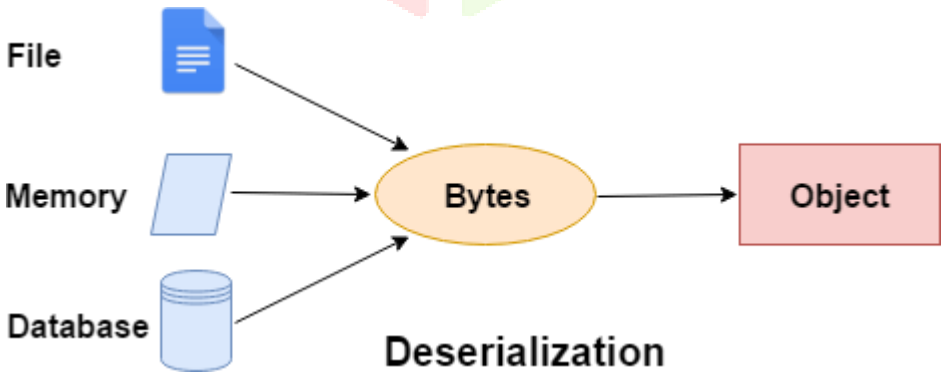
```
    public Student(int rollno, string name)
    {
        this.rollno = rollno;
        this.name = name;
    }
}
public class SerializeExample
{
    public static void Main(string[] args)
    {
        FileStream stream = new FileStream("e:\\sss.txt",
        FileMode.OpenOrCreate); BinaryFormatter formatter=new
        BinaryFormatter();

        Student s = new Student(101,
        "sonoo");
        formatter.Serialize(stream, s);

        stream.Close();
    }
}
```

C# Deserialization

In C# programming, deserialization is the reverse process of serialization. It means you can read the object from byte stream. Here, we are going to use **BinaryFormatter.Deserialize(stream)** method to deserialize the stream.



Example

```
using System;
using System.IO;
using
System.Runtime.Serialization.Formatters.Binary;
[Serializable]
class Student
{
   public int rollno;
   public string name;
   public Student(int rollno, string name)
   {
      this.rollno = rollno;
      this.name = name;
   }
}
public class DeserializeExample
{
   public static void Main(string[] args)
   {
      FileStream stream = new FileStream("e:\\sss.txt",
      FileMode.OpenOrCreate); BinaryFormatter formatter=new
      BinaryFormatter();

      Student
      s=(Student)formatter.Deserialize(stream);
      Console.WriteLine("Rollno: " + s.rollno);
      Console.WriteLine("Name: " + s.name);

      stream.Close();
   }
}
```

C# Delegates

In C#, delegate is a *reference to the method*. It works like *function pointer* in C and C++. But it is objected- oriented, secured and type-safe than function pointer.

For static method, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.

The best use of delegate is to use as event.

Internally a delegate declaration defines a class which is the derived class of **System.Delegate**.

C# Delegate Example

```
using System;
delegate int Calculator(int n);//declaring delegate

public class DelegateExample
{
   static int number = 100;
   public static int add(int n)
   {
      number = number + n;
      return number;
   }
   public static int mul(int n)
   {
      number = number * n;
      return number;
   }
   public static int getNumber()
   {
      return number;
   }
   public static void Main(string[] args)
   {
      Calculator c1 = new Calculator(add);//instantiating
      delegate Calculator c2 = new Calculator(mul);
      c1(20);//calling method using delegate
      Console.WriteLine("After c1 delegate, Number is: " +
      getNumber()); c2(3);
      Console.WriteLine("After c2 delegate, Number is: " + getNumber());

   }
}
```

C# Reflection

In C#, reflection is a *process to get metadata of a type at runtime*. The System.Reflection namespace contains required classes for reflection such as:

- Type
- MemberInfo
- ConstructorInfo
- MethodInfo
- FieldInfo
- PropertyInfo
- TypeInfo
- EventInfo
- Module
- Assembly
- AssemblyName
- Pointer etc.

The System.Reflection.Emit namespace contains classes to emit metadata.

C# Type class

C# Type class represents type declarations for class types, interface types, enumeration types, array types, value types etc. It is found in System namespace. It inherits System.Reflection.MemberInfo class.

C# Reflection Example: Get Type

```
using System;
public class ReflectionExample
{
  public static void Main()
   {
      int a = 10;
      Type type =
      a.GetType();
      Console.WriteLine(t
      ype);
   }
}
```

CONCLUSION :

C# is pronounced as "C-Sharp". It is an object-oriented programming language provided by Microsoft that runs on .Net Framework. C# programming language, we can develop different types of secured and robust applications like Window applications, Web applications, Distributed applications, Web service applications, Database applications etc.

Class and Object are the basic concepts of Object-Oriented Programming which revolve around the real-life entities. A class is a user-defined blueprint or prototype from which objects are created. Basically, a class combines the fields and methods(member function which defines actions) into a single unit. In C#, classes support polymorphism, inheritance and also provide the concept of derived classes and base classes. In this paper we understand about so many important things by using c sharp programming language like : Classes, Objects, Interfaces, Serialization and Deserialization, Delegate, Reflection and Type class.

REFERANCES

https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/classes
https://www.w3schools.com/cs/cs_classes.php
https://www.c-sharpcorner.com/UploadFile/0c1bb2/types-of-classes-in-C-Sharp1/
https://www.javatpoint.com/c-sharp-object-and-class
https://www.csharptutorial.net/csharp-tutorial/csharp-class/
https://www.techopedia.com/definition/25589/class-members-c-sharp
https://www.completecsharptutorial.com/basic/function-or-method.php
https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface
https://www.w3schools.com/cs/cs_interface.php
https://www.javatpoint.com/c-sharp-interface
https://www.c-sharpcorner.com/article/serialization-and-deserialization-in-c-sharp/
https://www.javatpoint.com/c-sharp-serialization
https://learn.microsoft.com/en-us/dotnet/standard/serialization/
https://www.tutorialspoint.com/Serialization-and-Deserialization-in-Chash
https://learn.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/how-to-hook-up-a-delegate-using-reflection
https://www.tutorialspoint.com/csharp/csharp_delegates.htm
https://stackoverflow.com/questions/44737722/c-sharp-delegate-vs-reflection
http://www.java2s.com/Tutorial/CSharp/0400__Reflection/CreateDelegateusingreflection.htm
https://www.javatpoint.com/c-sharp-delegates
https://www.tutorialspoint.com/csharp/csharp_reflection.htm
https://www.programiz.com/csharp-programming/reflection
https://www.javatpoint.com/c-sharp-reflection
https://www.c-sharpcorner.com/UploadFile/keesari_anjaiah/reflection-in-net/
https://learn.microsoft.com/en-us/dotnet/api/system.type?view=net-8.0
https://www.c-sharpcorner.com/UploadFile/0c1bb2/types-of-classes-in-C-Sharp1/
https://www.dotnetperls.com/type
https://www.dotnettutorial.co.in/2022/04/types-of-classes-in-c-with-example.html
https://dotnetpattern.com/csharp-reflection-type-class
https://www.tutorialspoint.com/csharp/csharp_classes.htm
https://www.w3schools.com/cs/cs_intro.php
https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
https://www.javatpoint.com/c-sharp-tutorial