Smit Vora

Learnings and Conclusion

Module-4

20. Enumerations – Enumerations are value-type user defined data types in C#. They are used for assigning constant values to integer. It makes constant values more readable. For ex – week of days, colors, directions.

20.1. **Syntax –**

```
enum Enum_variable
{
    string_1...;
    string_2...;
    .
    .
}
```

20.2. In above syntax, Enum_variable is the name of the enumerator, and string_1 is attached with value 0, string_2 is attached value 1 and so on. Because by default, the first member of an enum has the value 0, and the value of each successive enum member is increased by 1. We can change this default value.

20.3. Enum.GetNames(typeOf(enum_Name)) and Enum.GetValues(typeOf(enum_Name)) are functions used for getting the constant values of the enum and their index values respectively.

21. Handling Exceptions – Exceptions are the unexpected events occurring during the execution of program. Their response to be given is not known by program. They are handled by exception objects which call the exception handler code. The execution of exception handler is exception handling.

21.1. **Keywords**

try	Used to define a try block. This block holds the code that may throw an exception.

catch	Used to define a catch block. This block catches the exception thrown by the try block.
finally	Used to define the finally block. This block holds the default code.
throw	Used to throw an exception manually.

Syntax – (Multiple try-catch blocks)

```
try
{
  // statements that may cause an exception
}
catch(Specific_Exception_type obj)
{
  // handler code
}
catch(Specific_Exception_type obj)
{
  // handler code
}
.
.
.
finally
{
  //default code
}
```

21.2. **Exception Classes in C#** - C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from the **System.Exception** class are the

System.ApplicationException and **System.SystemException** classes.The **System.ApplicationException** class supports exceptions generated by application programs. Hence the exceptions defined by the programmers should derive from this class. The **System.SystemException** class is the base class for all predefined system exception.

21.3.

Exception Class	Description
ArgumentException	Raised when a non-null argument that is passed to a method is invalid.
ArgumentNullException	Raised when null argument is passed to a method.
ArgumentOutOfRangeException	Raised when the value of an argument is outside the range of valid values.
DivideByZeroException	Raised when an integer value is divide by zero.
FileNotFoundException	Raised when a physical file does not exist at the specified location.
FormatException	Raised when a value is not in an appropriate format to be converted from a string by a conversion method such as Parse.
IndexOutOfRangeException	Raised when an array index is outside the lower or upper bounds of an array or collection.
InvalidOperationException	Raised when a method call is invalid in an object's current state.
KeyNotFoundException	Raised when the specified key for accessing a member in a collection is not exists.
NotSupportedException	Raised when a method or operation is not supported.
NullReferenceException	Raised when program access members of null object.

OverflowException	Raised when an arithmetic, casting, or conversion operation results in an overflow.
OutOfMemoryException	Raised when a program does not get enough memory to execute the code.
StackOverflowException	Raised when a stack in memory overflows.
TimeoutException	The time interval allotted to an operation has expired.

22. **Events** - **Events** are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur.
    22.1. Event is raised in a class with a delegate in the same or some other class. The class which contains event is called **publisher** class. The class which accepts event is called **subscriber** class. This forms **publisher - subscriber model**. In the publisher class the event is raised after defining the delegate and defining the event itself.
    22.2. The publisher class object calls the event and it is notified to other objects as it consists of event definition.
    22.3. The subscriber class object contains the event handler. The delegate in the publisher class invokes the method(event handler) of the subscriber class.
    22.4. Example –

```
public delegate void VideoEncodedEventHandler(object source,
EventArgs args);

public event VideoEncodedEventHandler VideoEncoded;
```

23. **Basic File Operations -** File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc. There are two basic operation which is mostly used in file handling is reading and writing of the file. In C#, *System.IO* namespace contains classes which handle input and output streams and provide information about file and directory structure.
    23.1. **Write into a file StreamWriter Class -** The StreamWriter class implements TextWriter for writing character to stream in a particular

format. The class contains the following method which are mostly used.

Method	Description
Close()	Closes the current StreamWriter object and stream associate with it.
Flush()	Clears all the data from the buffer and write it in the stream associate with it.
Write()	Write data to the stream. It has different overloads for different data types to write in stream.
WriteLine()	It is same as Write() but it adds the newline character at the end of the data.

23.2. **Read into a file StreamReader Class** - The StreamReader class implements TextReader for reading character from the stream in a particular format. The class contains the following method which are mostly used.

Method	Description
Close()	Closes the current StreamReader object and stream associate with it.
Peek()	Returns the next available character but does not consume it.
Read()	Reads the next character in input stream and increment characters position by one in the stream
ReadLine()	Reads a line from the input stream and return the data in form of string
Seek()	It is use to read/write at the specific location from a file

23.3. **Copying file using File.Copy in C#** - If there is a need to copy file in our program then we can use File.Copy(sourceFileLoc,destFileLoc).We have to provide source file and destination file location.

23.4. **Deleting file using File.Delete in C# -** We use File.Delete(string filePath) in c# to delete a file

23.5. **Check if file exists –** To check if file exists or not we use File.Exists(string filePath)

23.6. **Append to a file in C# -** File.AppendText(string filePath) Method is an inbuilt File class method which is used to create a StreamWriter that appends UTF-8 encoded text to an existing file else it creates a new file if the specified file does not exist.

24. Interfaces And Inheritance

24.1. **Interfaces –** Interfaces are the another way of abstraction, in which they contain only abstract methods and properties. They show "what" should be implemented and the derived classes show "how" it should be implemented.

24.1.1. Interfaces specify what a class must do and not how.

24.1.2. Interfaces can't have private members.

24.1.3. By default all the members of Interface are public and abstract.

24.1.4. The interface will always defined with the help of keyword 'interface'.

24.1.5. Interface cannot contain fields because they represent a particular implementation of data.

24.1.6. Multiple inheritance is possible with the help of Interfaces but not with classes.

24.1.7. **Syntax of Interface Declaration –**

```
interface  <interface_name >
{
   // declare Events
   // declare indexers
   // declare methods
   // declare properties
}
```

24.1.8. **Syntax of Implementing Interface –**
class class_name : interface_name

24.1.9. **Advantages of Interfaces**

- It is used to achieve loose coupling.
- It is used to achieve total abstraction.
- To achieve component-based programming
- To achieve multiple inheritance and abstraction.

- Interfaces add a plug and play like architecture into applications.

24.2. **Inheritance –** It is mechanism in C# which enables one class to access methods and fields of other class. It is one of the characteristic of OOP.

24.2.1. **Super Class**: The class whose features are inherited is known as super class(or a base class or a parent class).

24.2.2. **Sub Class**: The class that inherits the other class is known as subclass(or a derived class).The subclass can add its own fields and methods in addition to the superclass fields and methods.

24.2.3. **Reusability**: Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

24.2.4. **Syntax** –

```
<acess-specifier> class <base_class> {
  ...
}

class <derived_class> : <base_class> {
  ...
}
```

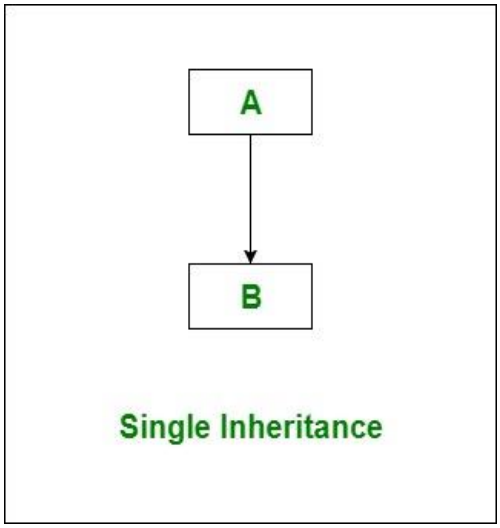
24.2.5. **Classification of Inheritance –**

24.2.5.1. **Implementation Inheritance –** This is an inheritance when a class is derived from another class.

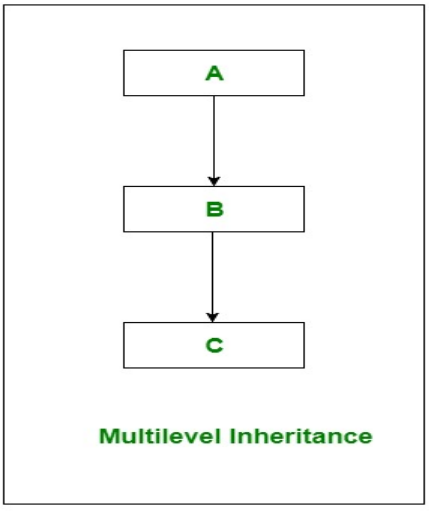
24.2.5.2. **Interface Inheritance –** This is an inheritance when a class is derived from an interface.

24.2.6. **Types Of Inheritances Supported in C#**

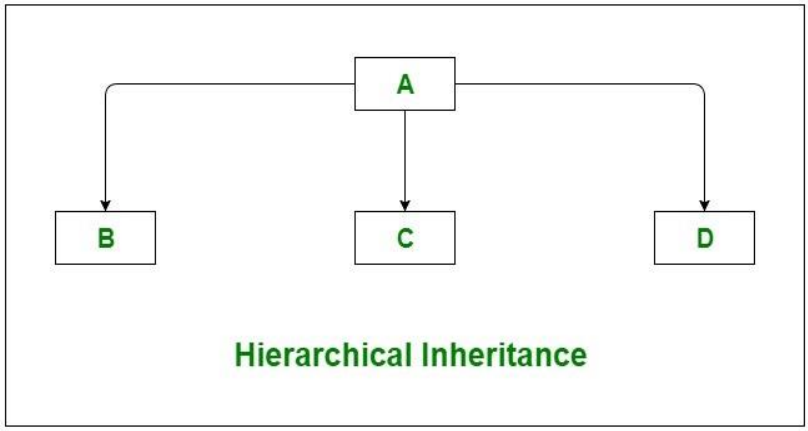
24.2.6.1. **Single Inheritance -** In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



24.2.6.2. **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.



24.2.6.3. **Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In below image, class A serves as a base class for the derived class B, C, and D.



24.2.7. **Some Points To remember about Inheritance –**

24.2.7.1. There can only be one superclass for each subclass.

24.2.7.2. Subclass cannot inherit constructor of superclass though it can invoke it.

24.2.7.3. Private members cannot be inherited directly but with properties values can be assigned and read.