RKIT Module – 4

Name: Adhyaru Keyur D.

Enumerations

- ✓ In C#, an enum (or enumeration type) is used to assign constant names to a group of numeric integer values.
- ✓ It makes constant values more readable, for example, WeekDays.Monday is more readable then number 0 when referring to the day in a week.
- ✓ An enum is defined using the enum keyword, directly inside a namespace, class, or structure.
- ✓ All the constant names can be declared inside the curly brackets and separated by a comma.

Example

```
enum Weekdays
{
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
}
```

Enum Values

- ✓ If values are not assigned to enum members, then the compiler will assign integer values to each member starting with zero by default.

Example

```
enum Weekdays
{
    Monday,      // 0
    Tuesday,     // 1
    Wednesday,   // 2
    Thursday,    // 3
    Friday,      // 4
    Saturday,    // 5
    Sunday       // 6
}
```

- ✓ You can assign different values to enum member. A change in the default value of an enum member will automatically assign incremental values to the other members sequentially.

Example

```
enum Categories
{
    Electronics,    // 0
    Food,           // 1
    Automotive = 6, // 6
    Arts,           // 7
    BeautyCare,     // 8
    Fashion         // 9
}
```

- ✓ You can even assign different values to each member.

Example

```
enum Categories
{
    Electronics = 1,
    Food = 5,
    Automotive = 6,
    Arts = 10,
    BeautyCare = 11,
    Fashion = 15,
    WomanFashion = 15
}
```

- ✓ The enum can be of any numeric data type such as byte, sbyte, short, ushort, int, uint, long, or ulong. However, an enum cannot be a string type.
- ✓ Specify the type after enum name as: type. The following defines the byte enum.

Example

```
enum Categories: byte
{
    Electronics = 1,
    Food = 5,
    Automotive = 6,
    Arts = 10,
    BeautyCare = 11,
    Fashion = 15
}
```

Methods

Enum.CompareTo

- ✓ Compare to method is user to compare two enums.

Enum.Format(Type, Object, String)

- ✓ Converts the specified value of a specified enumerated type to its equivalent string representation according to the specified format.

Format Parameters:

- ✓ g or G – If value is equal to a named enumerated constant, the name of that constant is returned; otherwise, the decimal equivalent of value is returned.
- ✓ X or x - Represents value in hexadecimal format without a leading "0x".
- ✓ D or d - Represents value in decimal form.

Enum.GetHashCode

- ✓ Returns the hash code for the value of this instance.

Enum.GetName

- ✓ Retrieves the name of the constant in the specified enumeration that has the specified value.

Enum.GetNames

- ✓ Retrieves an array of the names of the constants in a specified enumeration.

Enum.GetValues

- ✓ Retrieves an array of the values of the constants in a specified enumeration.

Enum.IsDefined

- ✓ Returns a Boolean telling whether a given integral value, or its name as a string, exists in a specified enumeration.

Enum.Parse

- ✓ Converts the string representation of the name or numeric value of one or more enumerated constants to an equivalent enumerated object.

Exception Handling

- ✓ Exception handling in C#, supported by the try catch and finally block is a mechanism to detect and handle run-time errors in code.
- ✓ The .NET framework provides built-in classes for common exceptions.
- ✓ The exceptions are anomalies that occur during the execution of a program. They can be because of user, logic or system errors.
- ✓ If a user (programmer) does not provide a mechanism to handle these anomalies, the .NET runtime environment provides a default mechanism, which terminates the program execution.

Try catch finally

- ✓ C# provides three keywords try, catch and finally to implement exception handling.
- ✓ The try encloses the statements that might throw an exception whereas catch handles an exception if one exists.
- ✓ The finally can be used for any clean-up work that needs to be done.

Example

```
try
{
    // Statement which can cause an exception.
}
catch(Type x)
{
    // Statements for handling the exception
}
finally
{
    //Any cleanup code
}
```

Throwing an Exception

- ✓ In C#, it is possible to throw an exception programmatically. The 'throw' keyword is used for this purpose. The general form of throwing an exception is as follows.
- ✓ Syntax: **throw** exception_obj;

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. For example, interrupts.
- ✓ Events are used for inter-process communication.

Using Delegates with Events

- ✓ The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the publisher class. Some other class that accepts this event is called the subscriber class. Events use the publisher-subscriber model.
- ✓ A publisher is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.
- ✓ A subscriber is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Points to Publishing an Event

1. First need to declare a delegate, that is the contract between the publisher and the subscriber.
2. Define the event based on the delegate.
3. Publish the event.

File

- ✓ C# includes static File class to perform I/O operation on physical file system.
- ✓ The static File class includes various utility method to interact with physical file of any type e.g. binary, text etc.
- ✓ Use this static File class to perform some quick operation on physical file.
- ✓ It is not recommended to use File class for multiple operations on multiple files at the same time due to performance reasons.
- ✓ Use FileInfo class in that scenario.

Points to Remember:

1. File is a static class to read\write from physical file with less coding.
2. Static File class provides functionalities such as create, read\write, copy, move, delete and others for physical files.
3. Static Directory class provides functionalities such as create, copy, move, delete etc for physical directories with less coding.
4. FileInfo and DirectoryInfo class provides same functionality as static File and Directory class.

Important Methods of Static File Class

Method	Usage
AppendAllLines	Appends lines to a file, and then closes the file. If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file.
AppendAllText	Opens a file, appends the specified string to the file, and then closes the file. If the file does not exist, this method creates a file, writes the specified string to the file, then closes the file.
AppendText	Creates a StreamWriter that appends UTF-8 encoded text to an existing file, or to a new file if the specified file does not exist.
Copy	Copies an existing file to a new file. Overwriting a file of the same name is not allowed.
Create	Creates or overwrites a file in the specified path.
CreateText	Creates or opens a file for writing UTF-8 encoded text.
Decrypt	Decrypts a file that was encrypted by the current account using the Encrypt method.
Delete	Deletes the specified file.
Encrypt	Encrypts a file so that only the account used to encrypt the file can decrypt it.
Exists	Determines whether the specified file exists.
GetAccessControl	Gets a FileSecurity object that encapsulates the access control list (ACL) entries for a specified file.
Move	Moves a specified file to a new location, providing the option to specify a new file name.
Open	Opens a FileStream on the specified path with read/write access.
ReadAllBytes	Opens a binary file, reads the contents of the file into a byte array, and then closes the file.
ReadAllLines	Opens a text file, reads all lines of the file, and then closes the file.
ReadAllText	Opens a text file, reads all lines of the file, and then closes the file.
Replace	Replaces the contents of a specified file with the contents of another file, deleting the original file, and creating a backup of the replaced file.
WriteAllBytes	Creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is overwritten.
WriteAllLines	Creates a new file, writes a collection of strings to the file, and then closes the file.
WriteAllText	Creates a new file, writes the specified string to the file, and then closes the file. If the target file already exists, it is overwritten.

Interface

- ✓ In the human world, a contract between the two or more humans binds them to act as per the contract.
- ✓ In the same way, an interface includes the declarations of related functionalities.
- ✓ The entities that implement the interface must provide the implementation of declared functionalities.
- ✓ In C#, an interface can be defined using the interface keyword.
- ✓ An interface can contain declarations of methods, properties, indexers, and events. However, it cannot contain fields, auto-implemented properties.

Example

```
interface IFile
{
    void ReadFile();
    void WriteFile(string text);
}
```

- ✓ You cannot apply access modifiers to interface members.
- ✓ All the members are public by default.
- ✓ If you use an access modifier in an interface, then the C# compiler will give a compile-time error "The modifier 'public/private/protected' is not valid for this item.".

Example

```
interface IFile
{
    protected void ReadFile(); //compile-time error
    private void WriteFile(string text);//compile-time error
}
```

Note: An interface can only contain declarations but not implementations.

Implementing an Interface

- ✓ A class or a Struct can implement one or more interfaces using colon (:).
- ✓ **Syntax**: <Class or Struct Name> : <Interface Name>

Note: Interface members must be implemented with the public modifier; otherwise, the compiler will give compile-time errors.

Explicit Implementation

- ✓ An interface can be implemented explicitly using <InterfaceName>.<MemberName>.
- ✓ Explicit implementation is useful when class is implementing multiple interfaces; thereby, it is more readable and eliminates the confusion.
- ✓ It is also useful if interfaces have the same method name coincidently.
- ✓ When you implement an interface explicitly, you can access interface members only through the instance of an interface type.

Note: Do not use public modifier with an explicit implementation. It will give a compile-time error.

Implementing Multiple Interfaces

- ✓ A class or struct can implement multiple interfaces. It must provide the implementation of all the members of all interfaces.
- ✓ It is recommended to implement interfaces explicitly when implementing multiple interfaces to avoid confusion and more readability.

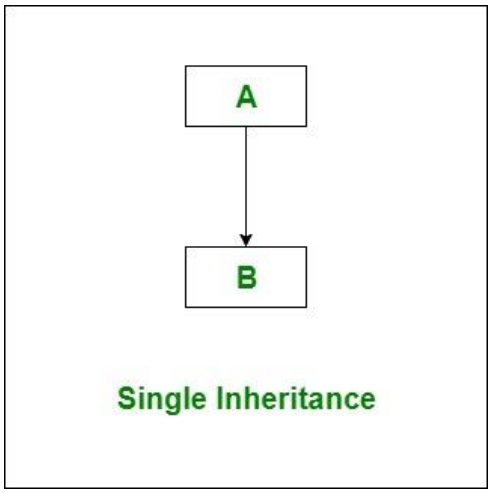
Points to Remember:

1. Interface can contain declarations of method, properties, indexers, and events.
2. Interface cannot include private, protected, or internal members. All the members are public by default.
3. Interface cannot contain fields, and auto-implemented properties.
4. A class or a struct can implement one or more interfaces implicitly or explicitly. Use public modifier when implementing interface implicitly, whereas don't use it in case of explicit implementation.
5. Implement interface explicitly using InterfaceName.MemberName.
6. An interface can inherit one or more interfaces.

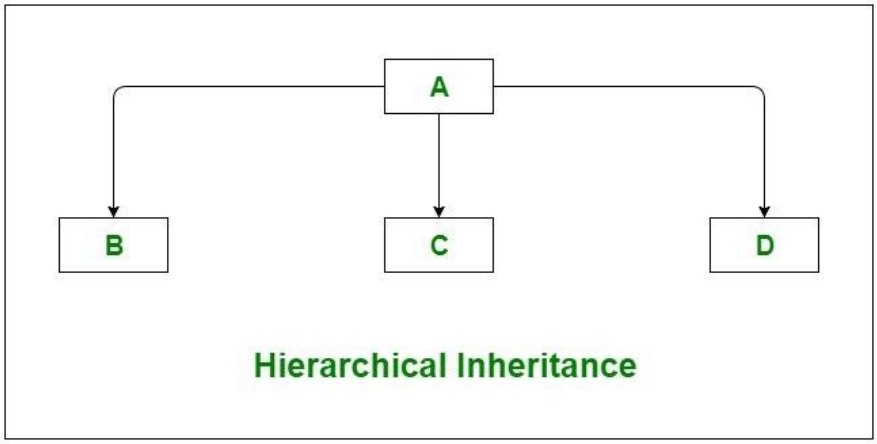
Inheritance

- ✓ Acquiring (taking) the properties of one class into another class is called inheritance. Inheritance provides reusability by allowing us to extend an existing class.
- ✓ The reason behind OOP programming is to promote the reusability of code and to reduce complexity in code and it is possible by using inheritance.
- ✓ The following are the types of inheritance in C#.

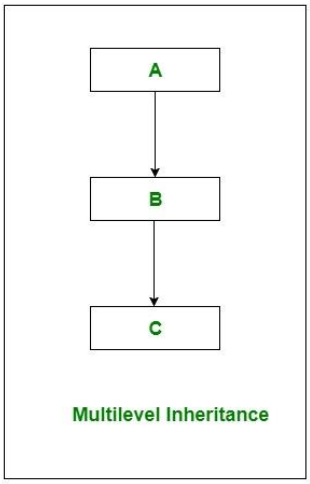
1.single



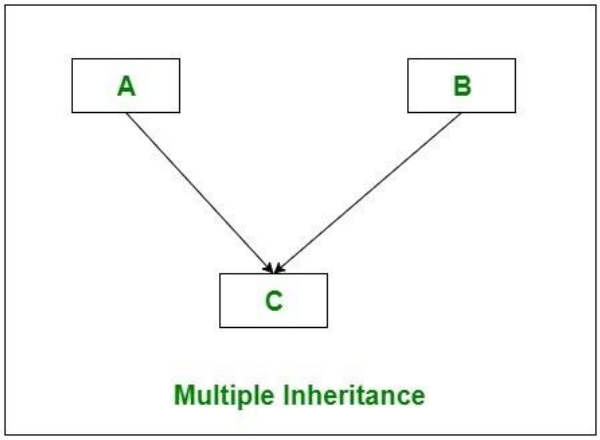
2.hierarchical



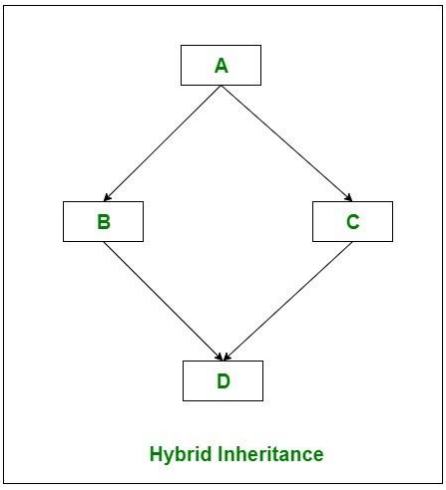
3.Multi level



4.Multiple (using interface)



5.Hybird (using interface)



- ✓ The inheritance concept is based on a base class and derived class. Let us see the definition of a base and derived class.
- ✓ **Base class** - is the class from which features are to be inherited into another class.
- ✓ **Derived class** - it is the class in which the base class features are inherited.