Enumerations

- 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. The following defines an enum for the weekdays.
- If values are not assigned to enum members, then the compiler will assign integer values to each member starting with zero by default. The first member of an enum will be 0, and the value of each successive enum member is increased by 1.
- We 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.
- We can even assign different values to each member.

- 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
- n enum can be accessed using the dot syntax: enum.member
- Explicit casting is required to convert from an enum type to its underlying integral type.

Handling Exceptions

- An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

- Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try, catch, finally, and throw.
- try − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- finally − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

- Syntax:

```
try {
  // statements causing exception
} catch( ExceptionName e1 ) {
  // error handling code
} catch( ExceptionName e2 ) {
  // error handling code
} catch( ExceptionName eN ) {
  // error handling code
} finally {
  // statements to be executed
}
```

- 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.
- The following table provides some of the predefined exception classes derived from the Sytem.SystemException class

- **System.IO.IOException** : Handles I/O errors.
- **System.IndexOutOfRangeException** : Handles errors generated when a method refers to an array index out of range.
- **System.ArrayTypeMismatchException** : Handles errors generated when type is mismatched with the array type.
- **System.NullReferenceException** : Handles errors generated from referencing a null object.
- **System.DivideByZeroException :** Handles errors generated from dividing a dividend with zero.
- **System.InvalidCastException :** Handles errors generated during typecasting.
- **System.OutOfMemoryException** : Handles errors generated from insufficient free memory.
- **System.StackOverflowException** : Handles errors generated from stack overflow.

Events

- Event gives a way for object to signal state change. A good example of event is in Graphical User Interface in which it notifies when user does something with controls.
- 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.
- 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. A publisher class object invokes the event and it is notified to other objects.
- Some other class that accepts this event is called the subscriber class. The delegate in the publisher class invokes the method of the subscriber class.
- Events use the publisher-subscriber model.

Basic file operations

- 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.

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 & inheritance

- **Interface:**
  - an interface includes the declarations of related functionalities. The entities that implement the interface must provide the implementation of declared functionalities.
  - 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.
  - 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
  - A class or a Struct can implement one or more interfaces using colon (:).
  - Syntax : Class or Struct Name> : <Interface Name>
  - 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.

- **Inheritance :**
  - One of the most important concepts in object-oriented programming is inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and speeds up implementation time.
  - When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.
  - The idea of inheritance implements the IS-A relationship. For example, mammal IS A animal, dog IS-A mammal hence dog IS-A animal as well, and so on.
  - A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.
  - Syntex:

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

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

  - Types of Inheritance:

- Single inheritance : It is the type of inheritance in which there is one base class and one derived class.
- Hierarchical inheritance : his is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.
- Multilevel inheritance : When one class is derived from another derived class then this type of inheritance is called multilevel inheritance.
- Multiple inheritance using Interfaces : class can inherit characteristics and features from more than one parent class. C# does not support multiple inheritances of classes. To overcome this problem we can use interfaces