Module-4

Understanding Document

PREPARED BY
PINAL PAMBHAR

Table of Contents

Enumerations

An enumeration type (or enum type) is a value type defined by a set of named constants of the underlying integral numeric type.

Declaring enum Variable:

```
enum <enum_name>
{
   enumeration list
};
```

- By default, the associated constant values of enum members are of type int; they start with zero and increase by one following the definition text order.
- You can explicitly specify any other integral numeric type as an underlying type of an enumeration type.
- You cannot define a method inside the definition of an enumeration 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 are 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:

- Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords.
- A try/catch block is placed around the code that might generate an exception.
- Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following.

```
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
}
```

- You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

Exception Classes:

- 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.

Exception Class	Description
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.

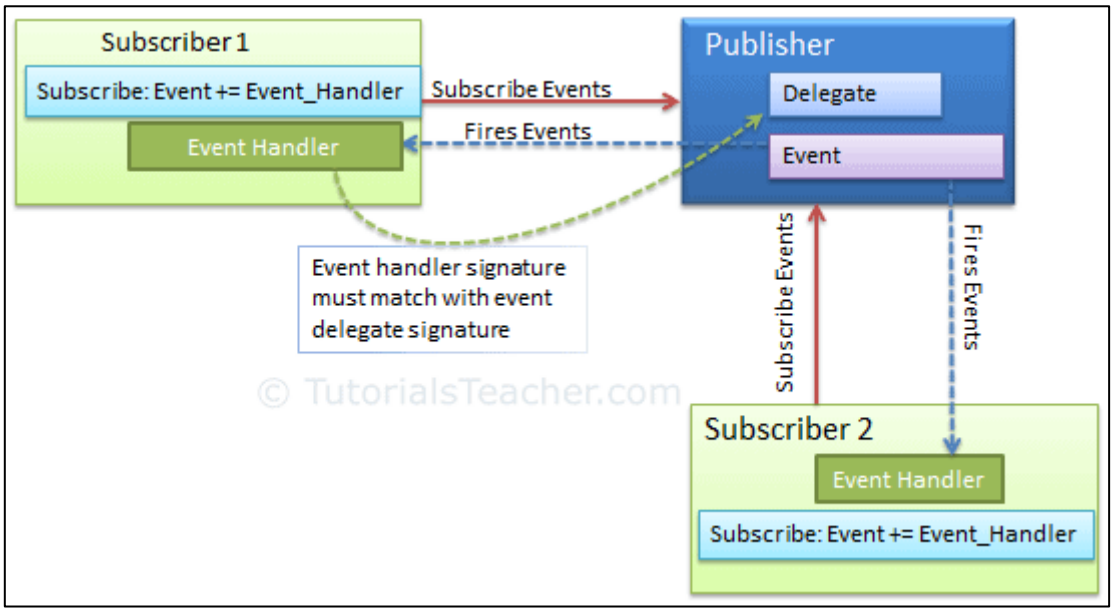
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Handling Exceptions:

- C# provides a structured solution to the exception handling in the form of try and catch blocks.
- Using these blocks the core program statements are separated from the error-handling statements.
- These error handling blocks are implemented using the try, catch, and finally keywords.
- You can also define your own exception. User-defined exception classes are derived from the Exception class.

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.



- The class that raises events is called Publisher, and the class that receives the notification is called Subscriber.
- There can be multiple subscribers to a single event.
- Typically, a publisher raises an event when some action occurred.
- The subscribers, who are interested in getting a notification when an action occurred, should register with an event, and handle it.

Basic file operations

- A file is a collection of data stored in a disk with a specific name and a directory path.
- In C# Provides static methods for the creation, copying, deletion, moving, and opening of a single file.

Below are some commonly used methods of System.IO namespace.

Method	Description
Create(String)	Creates or overwrites a file in the specified path.
Copy(String, String)	Copies an existing file to a new file. Overwriting a file of the same name is not allowed.
Delete(String)	Deletes the specified file.
Move(String, String)	Moves a specified file to a new location, providing the option to specify a new file name.
Open(String, FileMode)	Opens a FileStream on the specified path with read/write access with no sharing.
ReadAllLines(String)	Opens a text file, reads all lines of the file, and then closes the file.
ReadLines(String)	Reads the lines of a file.
WriteAllLines(String, String[])	Creates a new file, write the specified string array to the file, and then closes the file.
Replace(String, String, String)	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.
Exists(String)	Determines whether the specified file exists.
AppendAllLines(String,IEnumerable<String>)	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.

Interface & inheritance

Interface

- An interface looks like a class, but has no implementation.
- The only thing it contains are declarations of events, indexers, methods and/or properties.
- The reason interfaces only provide declarations is because they are inherited by structs and classes, that must provide an implementation for each interface member declared.
- It is used to achieve multiple inheritance which can't be achieved by class.

Syntax:

```
Interface interface_name
{
  // abstract events, indexers, methods and/or properties.
}
```

- Interfaces can’t have private members.
- By default all the members of interfaces are public and abstract.

Inheritance

- Inheritance is an important pillar of OOP (Object Oriented Programming).
- It is the mechanism in C# by which one class can inherit the features (fields and methods) of another class.

Important terminology:

- **Super Class:** The class whose features are inherited is known as super class (or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass (or a derived class, extended class, or child class).
- The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **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.

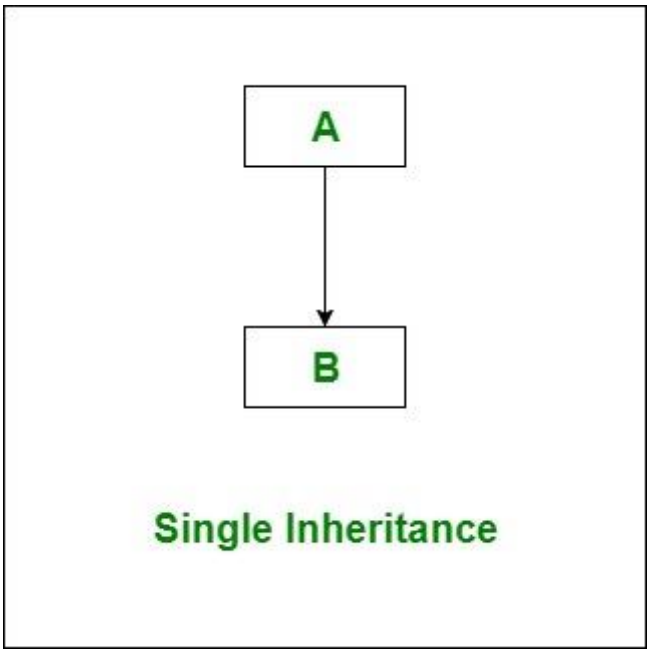
Syntax:

```
class derived-class : base-class
{
   // methods and fields
   .
   .
}
```

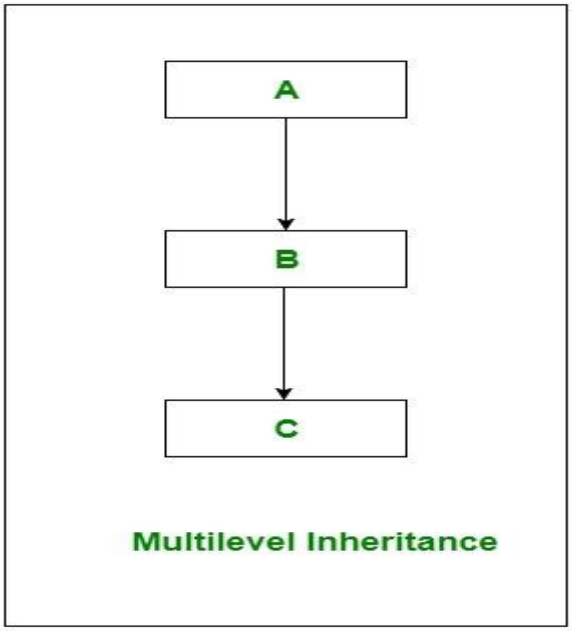
Types of Inheritance in C#:

Below are the different types of inheritance which is supported by C# in different combinations.

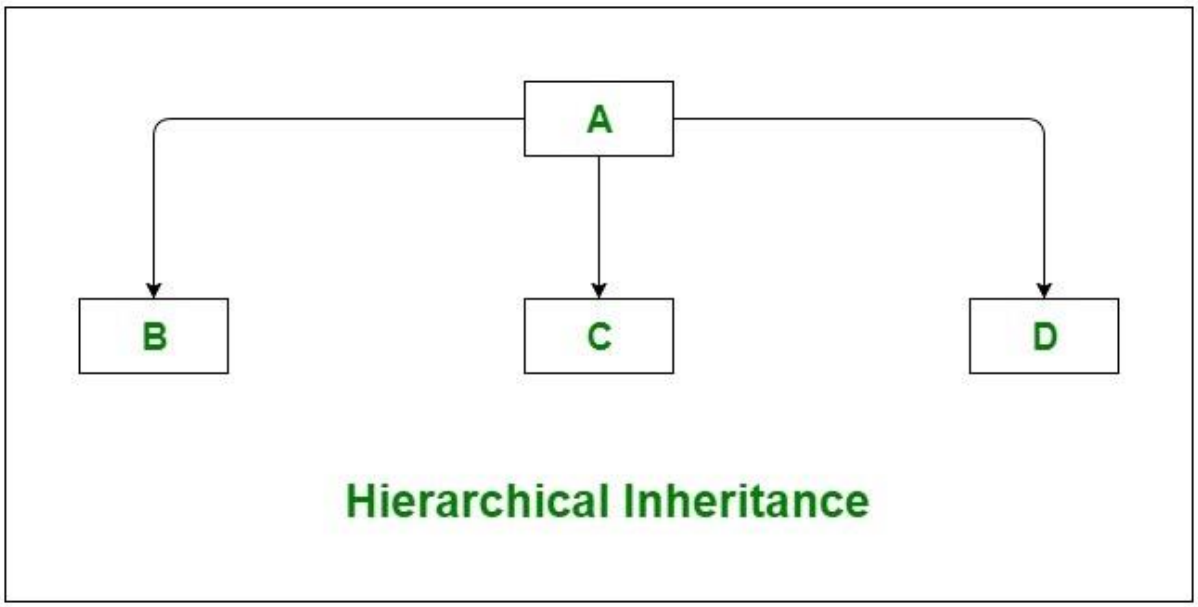
1. **Single Inheritance:** In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



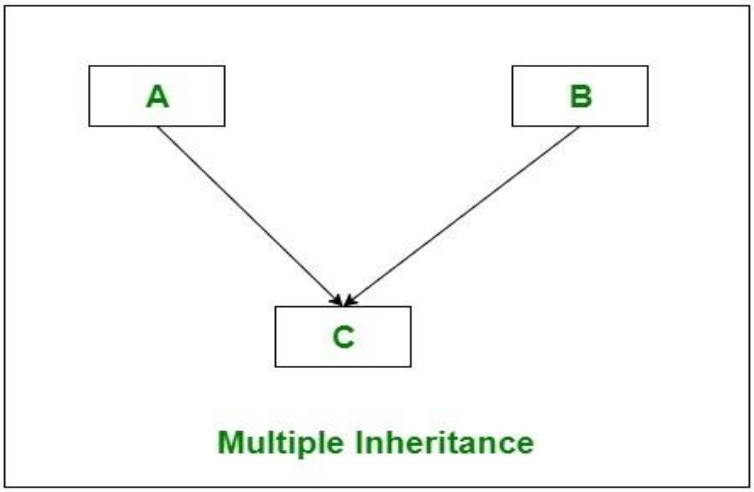
2. **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived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



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.



4. **Multiple Inheritance (Through Interfaces):** In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes.
    - Please note that C# does not support multiple inheritance with classes.
    - In C#, we can achieve multiple inheritance only through Interfaces. In the image below, Class C is derived from interface A and B.



5. **Hybrid Inheritance (Through Interfaces):** It is a mix of two or more of the above types of inheritance.
    - Since C# doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes.
    - In C#, we can achieve hybrid inheritance only through Interfaces.