PHASE 4

PREPARED BY BRIJESH KAMANI

Brijesh Kamani
FULL STACK DEVELOPER TRAINEE | RKIT

Contents

20. Enumerations

- *C# enum is a value type with a set of related named constants often referred to as an enumerator list.*
- *The C# enum keyword is used to declare an enumeration. It is a primitive data type, which is user-defined.*
- *Enums type can be an integer (float, int, byte, double etc.) but if you use beside int, it has to be cast.*
- *Enum is used to create numeric constants in the .NET framework. All members of the enum are of an enum type.*
- *There must be a numeric value for each enum type.*
- *The default underlying type of the enumeration elements is int.*
- *By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.*

Some points about enum

- *Enums are enumerated data types in C#.*
- *Enums are not for end-users, they are meant for developers.*
- *Enums are strongly typed constants. They are strongly typed, i.e., an enum of one type may not be implicitly assigned to an enum of another type even though the underlying value of their members are the same.*
- *Enumerations (enums) make your code more readable and understandable.*
- *enum values are fixed. enum can be displayed as a string and processed as an integer.*
- *The default type is int, and the approved types are byte, sbyte, short, ushort, uint, long, and ulong.*
- *Every enum type automatically derives from System.Enum and thus we can use System.Enum methods on enums.*
- *Enums are value types and are created on the stack and not on the heap.*
- *We can have the same value in the enum type.*

21. Handling Exceptions

- The C# language's exception handling features help you deal with any unexpected or exceptional situations that occur when a program is running. Exception handling uses the try, catch, and finally keywords to try actions that may not succeed, to handle failures when you decide that it's reasonable to do so and to clean up resources afterwards. Exceptions can be generated by the common language runtime (CLR), by .NET or third-party libraries, or by application code. Exceptions are created by using the throw keyword.
- In many cases, an exception may be thrown not by a method that your code has called directly, but by another method further down in the call stack. When an exception is thrown, the CLR will unwind the stack, looking for a method with a catch block for the specific exception type, and it will execute the first such catch block that it finds. If it finds no appropriate catch block anywhere in the call stack, it will terminate the process and display a message to the user.

Exceptions Overview

Exceptions have the following properties:

- *Exceptions are types that all ultimately derive from System.Exception.*
- *Use a try block around the statements that might throw exceptions.*
- *Once an exception occurs in the try block, the flow of control jumps to the first associated exception handler that is present anywhere in the call stack. In C#, the catch keyword is used to define an exception handler.*
- *If no exception handler for a given exception is present, the program stops executing with an error message.*
- *Don't catch an exception unless you can handle it and leave the application in a known state. If you catch System.Exception, rethrow it using the throw keyword at the end of the catch block.*
- *If a catch block defines an exception variable, you can use it to obtain more information about the type of exception that occurred.*
- *Exceptions can be explicitly generated by a program by using the throw keyword.*
- *Exception objects contain detailed information about the error, such as the state of the call stack and a text description of the error.*

- *Code in a finally block is executed regardless of if an exception is thrown. Use a finally block to release resources, for example to close any streams or files that were opened in the try block*

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
}
```

Summary:

- **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.*

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*** *class es.*

- *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.

Sr.No.	Exception Class & Description
1	**System.IO.IOException** Handles I/O errors.
2	**System.IndexOutOfRangeException** Handles errors generated when a method refers to an array index out of range.
3	**System.ArrayTypeMismatchException** Handles errors generated when type is mismatched with the array type.
4	**System.NullReferenceException** Handles errors generated from referencing a null object.
5	**System.DivideByZeroException** Handles errors generated from dividing a dividend with zero.
6	**System.InvalidCastException** Handles errors generated during typecasting.
7	**System.OutOfMemoryException** Handles errors generated from insufficient free memory.

8	**System.StackOverflowException** Handles errors generated from stack overflow.

22. Events

- *C# and .NET supports event driven programming via delegates. Delegates and events provide notifications to client applications when some state changes of an object. It is an encapsulation of idea that "Something happened". Events and Delegates are tightly coupled concept because event handling requires delegate implementation to dispatch events.*
- *Events enable a class or object to notify other classes or objects when something of interest occurs. The class that sends (or* raises*) the event is called the* publisher *and the classes that receive (or* handle*) the event are called* subscribers.

Events have the following properties:

- *The publisher determines when an event is raised; the subscribers determine what action is taken in response to the event.*
- *An event can have multiple subscribers. A subscriber can handle multiple events from multiple publishers.*
- *Events that have no subscribers are never raised.*
- *Events are typically used to signal user actions such as button clicks or menu selections in graphical user interfaces.*
- *In the .NET class library, events are based on the EventHandler delegate and the EventArgs base class.*
- *If an Event has multiple subscribers then event handlers are invoked synchronously.*

Syntax for the declaration of Event

```
public event EventHandler MyEvent;
```

Steps for implementing Event

To declare an event inside a class, first a Delegate type for the Event must be declared like below:

```
public delegate void MyEventHandler(object sender, EventArgs
 e);
```

Declare Event

```
public event MyEventHandler MyEvent;
```

Invoking an Event

```
if (MyEvent != null) MyEvent(this, e);
```

Invoking an Event can only be done from within the class that declared the Event.

23. Basic file operations

Stream

- *When you open a file for reading or writing, it becomes stream. Stream is a sequence of bytes traveling from a source to a destination over a communication path.*
- *The two basic streams are input and output streams. Input stream is used to read and output stream is used to write.*
- *The System.IO namespace includes various classes for file handling.*

The following table describes some commonly used classes in the System.IO namespace.

Class Name	Description
FileStream	It is used to read from and write to any location within a file
BinaryReader	It is used to read primitive data types from a binary stream
BinaryWriter	It is used to write primitive data types in binary format
StreamReader	It is used to read characters from a byte Stream
StreamWriter	It is used to write characters to a stream.
StringReader	It is used to read from a string buffer
StringWriter	It is used to write into a string buffer
DirectoryInfo	It is used to perform operations on directories
FileInfo	It is used to perform operations on files

Reading and writing in the text file

StreamWriter Class

- *The StreamWriter class in inherited from the abstract class TextWriter. The TextWriter class represents a writer, which can write a series of characters.*
- *The following table describes some of the methods used by StreamWriter class.*

Methods	Description
Close	Closes the current StreamWriter object and the underlying stream
Flush	Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream
Write	Writes to the stream

WriteLine	Writes data specified by the overloaded parameters, followed by end of line

StreamReader Class

- *The StreamReader class is inherited from the abstract class TextReader. The TextReader class represents a reader, which can read series of characters.*
- *The following table describes some methods of the StreamReader class.*

Methods	Description
Close	Closes the object of StreamReader class and the underlying stream, and release any system resources associated with the reader
Peek	Returns the next available character but doesn't consume it
Read	Reads the next character or the next set of characters from the stream
ReadLine	Reads a line of characters from the current stream and returns data as a string
Seek	Allows the read/write position to be moved to any position with the file

24. 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.*
- *So, what are interfaces good for if they don't implement functionality? They're great for putting together plug-n-play like architectures where components can be interchanged at will. Since all interchangeable components implement the same interface, they can be used without any extra programming. The interface forces each component to expose specific public members that will be used in a certain way.*

Syntax:

```
interface <interface_name>
{
        [properties];
              …
        [methods];
              …
}
```

Inheritance

- *Inheritance means acquiring the features and behaviors of a class by another class. It is a parent-child relationship. Using Inheritance methodology you can create a new class by using the existing class. It also referred to as reusability of the code. So when you use Inheritance you can reuse the code again and again.*
- *In Inheritance, Main class is called the base class or super class orparent class and the new class created from existing class are called as child class or sub class, child class, and derived class. We normally talk in terms of base class and derived class.*

Basic structure of Inheritance

```
class ParentClass
{

}
class ChildClass : ParentClass
{
}
```

Why we use inheritance?

- *It is used when we create a class and we want to reuse some of the methods or properties from existing class then that is a right way to implement inheritance.*

Types of inheritance

There are many types of Inheritance. I will explain all of them one by one.

Single Inheritance

- *Single Inheritance means there is a single base class which is implemented by single derived class is called as Single Inheritance. Here only one parent class and one derived class.*

Multilevel Inheritance

- *When you create a derived class which inherited from another derived class or in simple word if a class is created by using another derived class and this type of implementation is called as multilevel Inheritance*

Multiple Inheritance

- *C# does not support multiple inheritance due to the complexity of a code and it creates the diamond structure which create ambiguity.*

Hierarchical Inheritance

- *When we create a structure of project as like that where more than one derived classes are implemented from a same parent class or base class then that type of implantation is known as hierarchical inheritance.*

Advantages of Inheritance

1. *It reduces code redundancy.*
2. *It provides code reusability.*
3. *Reduces source code size and improves code readability.*
4. *After using this code is easy to manage and divided into parent and child classes.*
5. *It supports code extensibility by overriding the base class functionality within child classes.*

Disadvantages of Inheritance

1. *In Inheritance base class and child classes are tightly coupled. Hence, If you change the code of parent class, it will get affects to the all the child classes.*
2. *In the class hierarchy, many data members remain unused and the memory allocated to them is not utilized. Hence affect the performance of your program if you have not implemented inheritance correctly.*