Unit 1

Language Preliminaries

Unit coverage

- Fundamentals of .NET Framework and C# Basics.
- Advanced Object-Oriented Concepts.
- Programming Essentials and Code Optimization.
- Modern C# Features and Asynchronous Programming.

Use of base keyword

- Base keyword is used to access fields, constructors and methods of base class.
- Used when base and derived class have fields or methods of same name.
- If the fields or methods name are not similar then there is no need to use base keyword.
- For fields having similar name we can implement as follows:
  - base.field_name
- For methods having similar name we can implement as follows:
  - Public override void method_name(){}
  - ***virtual*** keyword is used to represent that the method can be override in the derived class.
  - ***override*** keyword is used to represent that the method has been override.
- Base class constructor is internally invoked

Method Hiding and Method Overriding

- In method hiding, the derived class defines a new method with the same name as the base class method.
- The base class method is not replaced but is hidden in the derived class.
- ***new*** keyword is used to hide the method of base class.
- In method overriding the derived class defines a new method with the same name as the base class method however the base class method is replaced not hidden.
- Two keywords ***virtual*** for base class and ***override*** for derived class is used.

Structure

- Structure is a value type data type.
- Used to group related variables under one name.
- Holds actual data rather than reference.
- Keyword ***struct*** is used for defining structure.
- Structures can have methods, fields, indexers, properties.
- Best for small, lightweight objects that group related data.
- No inheritance and default constructors.

Enums

- Short form of Enumeration.
- Special value type in C# that defines a set of named constants.
- Objective of enum is to define our own data types(Enumerated Data Types).
- Keyword enum is used to define own data types.
- enum improves type safety.
- enum can be traversed.
- Can be defined inside namespace, class and structure.
- Constants has default values which starts from 0 and incremented to one by one.
- However the default values can be changed.

Abstract Class

- Abstract class is a class that cannot be instantiated.
- Meaning objects cannot be created from abstract class.
- Provides ***data abstraction*** meaning hiding certain details and showing only essential information to the users.
- It serves as a base class for other classes to inherit from.
- Keyword ***abstract*** is used before class to create abstract class in C#.
- Can have abstract methods and regular methods.
- Mostly used for inheritance.

Sealed Class

- Classes that cannot be inherited by other classes.
- no other class can derive from a sealed class.
- helps to prevent further modification or extension of that class, ensuring that the class behaves exactly as defined.
- Keyword ***sealed*** is used to define sealed class.
- We can also have sealed methods which cannot be overridden further.

Interface

- Blue print of class.
- Like a class, Interface can have methods, properties, events, and indexers as its members.
- But interfaces will contain only the declaration of the members.
- The implementation of the interface's members will be given by class who implements the interface.
- By default all the members of Interface are public and abstract can't have private members.
- Defined with the help of keyword '***interface***'.
- Multiple inheritance is possible with the help of Interfaces but not with classes.

Interface

Syntax:

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

Implementation:

```
class class_name : interface_name
```

Polymorphism

- Polymorphism means having many form.
- Allows one action or behavior to work differently based on the object performing it.
- Two types of polymorphism:
  - Static Polymorphism
    - Linking a function with an object during compile time.
    - Function overloading is a technique to implement static polymorphism.
    - Function overloading can have multiple definition for the same function name.
    - The definition differs based on the type or number of arguments passed in the function.
  - Dynamic Polymorphism
    - Method call is determined at runtime.
    - Achieved through method overriding
    - The base class defines a method as ***virtual***, and derived classes ***override*** it to provide specific implementations.

Delegate

- A delegate is a variable that holds the reference to a method or pointer to a method.
- Defines what a method should look like.
- It can refer to more than one method of same return types and parameters.
- Once created we can assign any method to that delegate which matches its description.
- Syntax:
  - <modifier> delegate <return_type> <delegate_name>(<parameter_list>);
- Example:
  - public delegate void print_name(string name);
- Invocation:
  - <delegate_name> <instance_name> = new <delegate_name> <calling_method_name>;
- Example:
  - print_name print_n = new print_name()
- Multicast Delegate:
  - Delegate that holds the reference of more than one function.
  - Operators used +, -, +=, -=

Event

- Event is the notification mechanism which depends on delegate.
- It is dependent on delegate thus cannot be created without delegate.
- It is like a wrapper over delegate to improve its security.
- Mostly used in GUI programming.

Partial Class

```
public partial class Geeks {
    public void Display()
    {
        Console.WriteLine("Author's name is : " + Author_name);
        Console.WriteLine("Total number articles is : " + Total_articles);
    }
}
```

- Allows to split a class into multiple files.
- Created using ***partial*** keyword.
- Can also be used to split methods, interfaces and structures.
- Used while working on large projects, automatically generated code.

```
public partial class Geeks {
    private string Author_name;
    private int Total_articles;

    public Geeks(string a, int t)
    {
        this.Authour_name = a;
        this.Total_articles = t;
    }
}
```

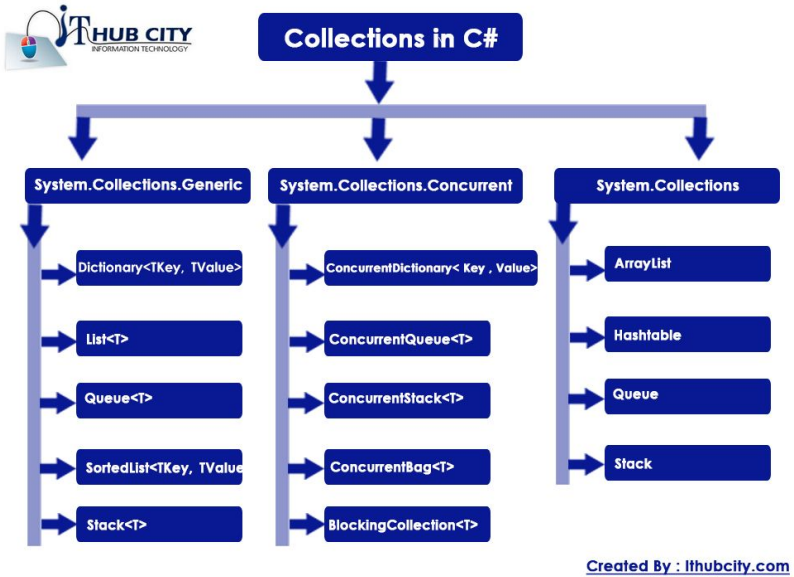
```
public class Geeks {
    private string Author_name;
    private int Total_articles;

    public Geeks(string a, int t)
    {
        this.Authour_name = a;
        this.Total_articles = t;
    }

    public void Display()
    {
        Console.WriteLine("Author's name is : " + Author_name);
        Console.WriteLine("Total number articles is : " + Total_articles);
    }
}
```

Collections

- Collections are data structures used to store, manage, and manipulate groups of objects in C#
- More flexible than arrays because they can grow or shrink dynamically
- Provides built-in methods to add, remove, search, and sort data.
- Operations that can be performed: store, update, delete, retrieve, search and sort.
- Are of two types non generic and generic
- Example: ArrayList



Generics

- Used when elements of collection contains only one data type.
- Enforces type safety so no other data type can be added.
- Optimal while retrieving as no need to determine the data type or convert it..
- Are implemented under System.Collection.Generic namespace
- Classes present are:
  - Stack<T>
  - Queue<T>
  - LinkedList<T>
  - SortedList<T>
  - List<T>
  - Dictionary<TKey, TValue>

LINQ (Language Integrated Query)

- A set of methods for querying and manipulating data in a declarative way, directly in C#.
- Two main syntax styles: Query Syntax (SQL-like) and Method Syntax (using extension methods like Where(), Select()).
- Key LINQ Methods
  - Where(): Filters data based on a condition.
  - Select(): Projects data into a new form.
  - OrderBy(): Sorts data in ascending order.
  - GroupBy(): Groups data based on a key.
- LINQ queries are not executed until they are iterated (e.g., using foreach() or converting to a collection with .ToList()).

Lambda Expression

- Used to represent anonymous methods i.e methods without name.
- Uses goes-to operator (=>)
- Used for writing inline methods or expression.
- Often used in LINQ queries.
- Syntas: (parameters) => expression_or_statement_block;
- Helpful when:
  - Writing concise code instead of defining a full method.
  - Clean and compact code.
  - For inline delegates and methods.

Exception Handling

- Achieved through three keywords: try, catch and finally.
- Try block contain statements which can cause an exception.
- Catch block contains statements for handling the exception.
- Finally block contains statements that will execute regardless of the outcome.
- All Exceptions are child class of base class Exception.